HASAC: Energy Adaptive Secure Firmware Updates for Critical IoT Systems

Sayon Duttagupta

COSIC, KU Leuven, Leuven, Belgium
sayon.duttagupta@esat.kuleuven.be

Abstract. Secure Firmware Over-The-Air (FOTA) updates are essential for Critical Infrastructure IoT (CI-IoT) devices, but impose severe energy costs on battery-powered nodes operating over Low Power Wide Area Networks such as NB-IoT. This paper presents HASAC (*Hardware-Aware Segmented Authenticated Compression*), a FOTA framework for resource-constrained IoT devices that minimises update energy by dynamically adapting cryptographic and compression strategies to device hardware capabilities and radio conditions. We implement HASAC on a Cortex-M33 platform and show that radio transmission and interconnect overhead account for more than 90% of the total update energy, while cryptographic computation contributes only a small fraction. We further demonstrate that on devices equipped with hardware security modules, AES-128-CCM achieves more than an order-of-magnitude improvement in throughput and energy efficiency compared to software implementations, even for small firmware segments. Building on these insights, HASAC dynamically selects between hardware-accelerated AES and software-optimised ChaCha20-Poly1305, and applies link-aware LZ4 compression only when it yields a positive energy return. Model-based evaluation demonstrates that, in deep coverage scenarios, HASAC reduces the total energy cost of secure firmware updates by up to 30% compared to conventional uncompressed baselines, reconciling strong security guarantees with long-term battery longevity.

Keywords: IoT security · Firmware update · Applied cryptography · NB-IoT · Energy efficiency · Cortex-M33

1 Introduction

Secure Firmware Over-the-Air (FOTA) updates are a prerequisite for the long-term security of Critical Infrastructure IoT deployments such as smart water meters, gas meters, and environmental monitoring nodes. These devices are typically battery-powered, installed in hard-to-access locations, and expected to operate for a decade or longer without maintenance. At the same time, they increasingly rely on Low Power Wide Area Networks (LPWANs), including Narrowband IoT (NB-IoT), to provide wide-area connectivity under challenging radio conditions. In such settings, firmware updates are among the most energy-intensive operations in the device lifecycle. In basements, underground pits, and

shielded utility locations, NB-IoT employs coverage enhancement through repetition and link adaptation, dramatically increasing airtime. As a result, the energy cost per delivered byte can grow by orders of magnitude, and a single firmware update can consume energy comparable to weeks or months of normal sensing and reporting operation.

Despite this, secure update mechanisms are commonly designed with a primary focus on computational efficiency. Prior work and industrial practice often optimise cryptographic primitives, assuming that cryptographic computation dominates energy consumption. The measurements show that this assumption is fundamentally incorrect for NB-IoT based firmware updates. Using a Cortex-M33 platform, we observe that radio transmission and interconnect overhead account for more than 90% of the total update energy, while on-device cryptographic processing contributes only a small fraction, particularly on platforms with hardware acceleration. In this regime, reducing transmitted bytes has a far greater impact on energy than reducing CPU cycles. This shift in perspective has important security implications. Operators frequently delay or avoid firmware updates to preserve battery lifetime, leaving devices exposed to known vulnerabilities. An adversary can further exploit this tension by triggering repeated or partial downloads to induce battery exhaustion. Energy, therefore, becomes a first-class security constraint, rather than a secondary performance concern. In addition, Cortex-M33 based devices exhibit significant hardware heterogeneity. Some platforms provide hardware support for cryptographic operations, while others rely entirely on software. Fixing a single cryptographic configuration across all devices is therefore suboptimal, either wasting silicon capabilities or imposing unnecessary computation costs. At the same time, firmware images exhibit highly uneven compressibility. Compressing all firmware data indiscriminately can waste energy when compression yields little reduction in transmitted bytes, particularly under good radio conditions.

These observations motivate *HASAC*, a secure firmware update framework that explicitly treats FOTA as an energy-constrained security problem. HASAC adapts cryptographic processing to device hardware capabilities and applies segment-wise, link-aware compression only when it yields a positive energy return. By aligning security mechanisms with both radio conditions and hardware features, HASAC reduces update energy while preserving strong authenticity and integrity guarantees. The contributions are as follows.

1. **Energy breakdown of secure firmware updates.** Using a Cortex-M33 platform with an NB-IoT modem, we measure the per-segment timing of radio reception, interconnect transfer, and on-device processing during secure FOTA. We show that communication dominates total energy consumption across realistic deployment conditions.
2. **Quantitative analysis of cryptographic cost in FOTA.** We benchmark software-optimised AES-CCM and ChaCha20-Poly1305 on Cortex-M33 devices, with and without hardware security modules. We show that hardware acceleration reduces cryptographic energy to a negligible fraction of the total update cost, even for small firmware segments.

3. **HASAC framework.** We propose HASAC, a secure update framework that adapts authenticated encryption to device hardware capabilities and applies segment-wise, link-aware LZ4 compression using an explicit energy break-even rule derived from measured radio and processing costs.
4. **Model-based evaluation under varying link conditions.** We develop an analytical energy model incorporating NB-IoT coverage enhancement and evaluate a 128 kB firmware update. In deep coverage scenarios, HASAC reduces total update energy by up to one third compared to conventional uncompressed baselines, while avoiding unnecessary computation in good coverage.

Artefact Availability: The entire source code and artefact underlying this paper, including the implementation and benchmarkings, and the exact firmware files used, can be found at `https://github.com/KULeuven-COSIC/HASAC/`.

2 Related Work

Secure firmware update mechanisms for IoT devices have been studied extensively, with most work focusing on authenticity, integrity, and rollback protection rather than energy efficiency. RFC 9019 defines a generic architecture for IoT firmware updates, introducing roles such as firmware server, device, and status tracker, together with signed manifests that bind metadata to firmware images [9]. RFC 9124 further specifies a detailed information model for such manifests, targeting constrained devices using compact CBOR encodings [8]. While these standards establish a solid security foundation, they deliberately abstract away transport costs and device-level energy constraints.

Several academic systems build on similar principles. ASSURED provides a secure update architecture with strong cryptographic guarantees and minimal trusted computing base requirements [1]. UpKit proposes a lightweight and portable update framework suitable for constrained IoT platforms [6]. The Secure FoTA Object specifies a standardised object model for authenticated firmware delivery in IoT management protocols [2]. These systems ensure the correctness and robustness of updates but treat communication and computation costs as secondary concerns. More recent work has explored formal assurance and certification of update procedures. Tacchella et al. introduce certified secure updates, focusing on verifiable correctness of update execution and compliance with security policies [11]. This line of work strengthens trust guarantees but does not address the operational cost of updates on battery-powered devices or under challenging radio conditions.

Survey papers consistently identify energy consumption as a practical challenge for IoT firmware updates, especially over LPWANs, but stop short of providing concrete models or adaptive mechanisms [4, 5]. In particular, prior work does not quantify the relative contribution of radio communication versus cryptographic processing, nor does it exploit hardware acceleration heterogeneity to optimise update energy.

In contrast to existing work, HASAC treats energy as a first-class security constraint. It combines measurement-grounded analysis, hardware-aware cryptographic selection, and link-aware compression to minimise update energy while preserving standard security properties. Rather than proposing a new update architecture, HASAC complements existing frameworks such as SUIT by optimising how secure updates are executed on real devices and real radio links.

3 System and Threat Model

3.1 System Model

We consider a battery-powered CI-IoT device built around a Cortex-M33 class microcontroller and connected to the network via an NB-IoT modem. Firmware updates are delivered over the NB-IoT downlink and transferred to the microcontroller over a serial interface. Due to coverage enhancement and link repetition, the effective energy cost per transmitted byte can vary significantly depending on deployment conditions. The device implements a secure boot chain with a minimal bootloader residing in protected flash. The bootloader verifies a signed update manifest, authenticates firmware segments, and installs a new firmware image into a staging region before activation. Firmware updates are segmented and processed incrementally to accommodate limited memory and to tolerate intermittent connectivity. We assume that devices may differ in their cryptographic capabilities, ranging from software-only implementations to platforms equipped with hardware security modules. HASAC is designed to adapt to this heterogeneity without changing the update protocol or trust assumptions.

3.2 Threat Model

We assume a network adversary with full control over the NB-IoT communication channel. The adversary can eavesdrop, inject, replay, delay, or modify update traffic, and can attempt to trigger repeated or partial firmware downloads. The adversary's goals include installing unauthorised firmware, rolling devices back to vulnerable versions, extracting firmware contents, or accelerating battery depletion. Each device is provisioned at manufacture time with a trust anchor used to verify update manifests. We assume this trust anchor and the secure bootloader are not compromised. Physical attacks, side-channel attacks, and compromise of server-side signing keys are out of scope.

4 Empirical Analysis and Energy Modelling

This section quantifies the energy cost of secure firmware updates through direct measurements and develops an analytical model that generalises these results across coverage conditions and update strategies. The goal is to isolate the relative contributions of communication and computation, and to identify when compression and cryptographic choices materially affect total update energy.

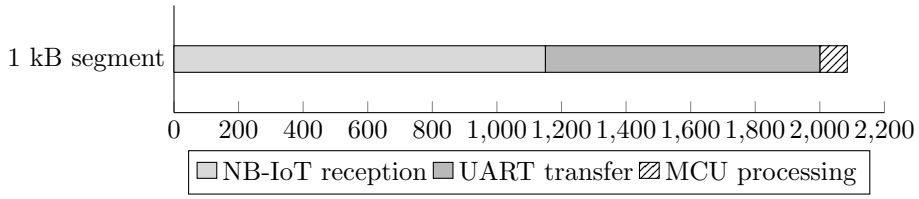

Fig. 1: Measured timing profile (in ms) for downloading and installing a 1 kB firmware segment. Communication dominates the update timeline, while on-device processing contributes only a small fraction.

4.1 Measurement Methodology

We implement an FOTA client that downloads firmware segments of 1 kB over NB-IoT. To capture the timing of each update phase, we instrument the firmware with GPIO markers around three operations: radio reception at the modem, data transfer over the UART interconnect, and on-device processing, including authenticated decryption and flash programming. Timing traces are collected using a logic analyser and averaged over multiple runs.

To validate power assumptions, we measure current draw using a Nordic Power Profiler Kit. During radio activity and active processing, current remains in the range of several milliamperes. To avoid optimistic estimates, we adopt a conservative constant active power model of 10 mA at 3 V, corresponding to 30 mW. While simplified, this model is sufficient to reason about relative energy contributions and break-even points.

4.2 Measured Timing Breakdown

Fig. 1 shows the measured timing for a single 1 kB firmware segment. Radio reception dominates the timeline at approximately 1150 ms, followed by UART transfer at roughly 850 ms. In contrast, on-device processing, including authenticated decryption and flash writes, requires only about 85 ms.

These measurements already indicate that optimising computation alone cannot significantly reduce update duration or energy, as the microcontroller is idle or waiting for I/O for most of the transaction.

4.3 Measured Energy per Segment

Using the measured timings and the active power model, the energy consumed during an interval of duration t is

$$E = P_{\text{active}} \cdot t. \tag{1}$$

For a 1 kB segment, this yields

$$E_{\text{radio}} \approx 0.03\ \text{W} \cdot 1.15\ \text{s} \approx 34.5\ \text{mJ}, \tag{2}$$

$$E_{\text{UART}} \approx 0.03\ \text{W} \cdot 0.85\ \text{s} \approx 25.5\ \text{mJ}, \tag{3}$$

$$E_{\text{CPU}} \approx 0.03\ \text{W} \cdot 0.085\ \text{s} \approx 2.55\ \text{mJ}. \tag{4}$$

Radio and interconnect together account for nearly 60 mJ per kilobyte, whereas cryptographic processing and flash writes contribute only a few millijoules. Communication, therefore, dominates computation by an order of magnitude.

4.4 Analytical Energy Model

To generalise these observations, we model the total energy of a firmware update as

$$E_{\mathrm{FOTA}} = E_{\mathrm{connect}} + \sum_{i=1}^{N} \left(E_{\mathrm{RX}}(S_i) + E_{\mathrm{UART}}(S_i) + E_{\mathrm{CPU}}(S_i) \right), \tag{5}$$

where the firmware image is divided into N segments S_i of size b bytes.

Radio reception time is modelled as

$$t_{\mathrm{radio}}(S_i) = \frac{|S_i|}{R_{\mathrm{eff}}} \cdot r_{\mathrm{CE}}, \tag{6}$$

where R_{eff} is the effective downlink rate under good coverage and r_{CE} captures coverage enhancement through repetition. The corresponding energy is

$$E_{\mathrm{RX}}(S_i) = P_{\mathrm{active}} \cdot t_{\mathrm{radio}}(S_i). \tag{7}$$

UART transfer and on-device processing are modelled similarly, with processing decomposed into decryption, optional decompression, and flash programming. Crucially, only the radio component scales with the coverage enhancement factor, while CPU energy remains unchanged. This asymmetry is central to my design.

4.5 Compression Behaviour

We analyse firmware compressibility using LZ4 [7] by partitioning real firmware images into 1 kB and 8 kB segments. For a segment S_i, the compression factor is

$$\mathrm{Ratio}_i = \frac{|S_i|}{|LZ4(S_i)|}. \tag{8}$$

Fig. 2 shows that compressibility varies significantly across segments, particularly at small granularity. Many 1 kB segments are effectively incompressible, while others compress substantially. Larger segments exhibit more stable compression ratios but increase memory requirements.

These results show that indiscriminate compression is inefficient. Compression must be applied selectively, balancing reduced radio energy against added processing cost. This observation directly motivates the design of HASAC.

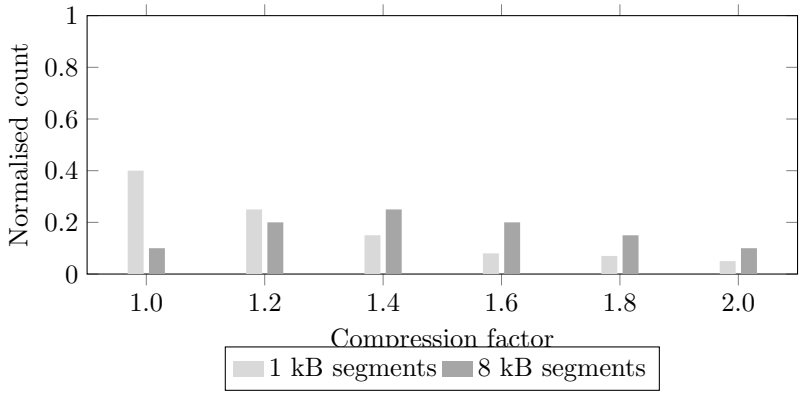

Fig. 2: Distribution of LZ4 compression factors for firmware segments. Compressibility varies widely at small granularity, motivating segment-wise decisions.

5 The HASAC Framework

Motivated by the empirical results and energy model in Sect. 4, we now present HASAC, a secure firmware update framework that treats energy as a first-class security constraint. HASAC explicitly separates communication-dominated costs from computation-dominated costs and adapts both cryptography and compression to minimise total update energy under realistic deployment conditions.

5.1 Design Principles

HASAC is guided by three design principles.

- **Radio first optimisation.** Since radio reception and interconnect transfer dominate update energy, HASAC prioritises reducing transmitted bytes. Optimising cryptographic computation alone yields limited benefit in NB-IoT deployments.
- **Hardware aware cryptography.** HASAC adapts its authenticated encryption scheme to the capabilities of the target device, exploiting hardware acceleration where available and using software-optimised constructions otherwise.
- **Segment wise adaptivity.** Compression decisions are taken per segment and depend jointly on segment compressibility and link conditions, ensuring that decompression never increases total energy.

5.2 Adaptive Cryptographic Selection

HASAC supports two standard AEAD schemes: AES-CCM [3] and ChaCha20-Poly1305 [10]. During provisioning, the device exposes whether a hardware cryptographic engine is available. This capability information is bound to the device identity and used by the update server when preparing firmware images.

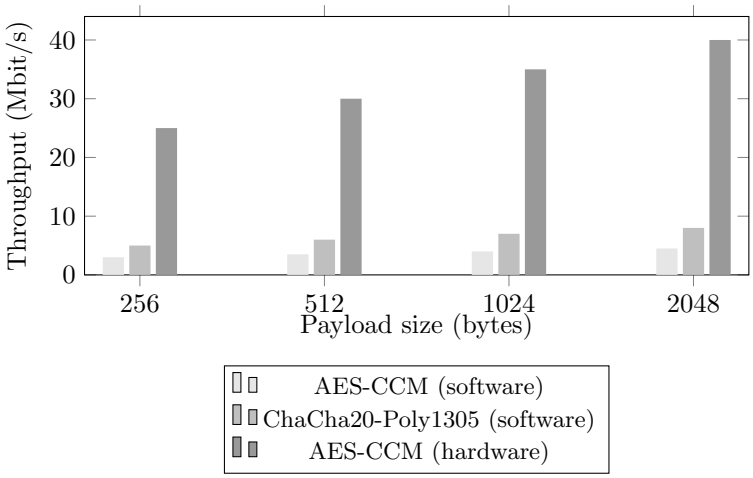

Fig. 3: Measured authenticated encryption throughput on a Cortex-M33 platform. Hardware-accelerated AES-CCM achieves an order of magnitude higher throughput than software implementations, while ChaCha20-Poly1305 outperforms software AES in the absence of acceleration.

On devices with hardware cryptographic support, HASAC selects AES-CCM. Each firmware segment is encrypted using a per-update key and a per-segment nonce, and the device performs decryption and authentication using the hardware engine. On devices without hardware support, HASAC selects ChaCha20-Poly1305, which provides equivalent security guarantees while offering substantially higher throughput and lower energy consumption in software.

Fig. 3 summarises the measured cryptographic throughput. In the absence of hardware acceleration, ChaCha20-Poly1305 outperforms software AES-CCM. When hardware acceleration is enabled, AES-CCM achieves an order of magnitude higher throughput, rendering cryptographic processing negligible compared to UART and radio costs even for small segments.

5.3 Segment Wise Compression Policy

Firmware images are partitioned into fixed-size segments aligned to the flash programming granularity. For each segment S_i, the server computes its LZ4 compressed form C_i and corresponding compression factor

$$\text{Ratio}_i = \frac{|S_i|}{|C_i|}. \tag{9}$$

HASAC selects a compression threshold T_{CE} based on the estimated coverage enhancement level of the target device. In good coverage, where repetition is minimal, HASAC applies a conservative threshold and compresses only strongly compressible segments. In deep coverage, where repetition amplifies the energy

cost per byte, HASAC lowers the threshold and compresses even weakly compressible segments.

For each segment, the server applies the rule

$$\text{send } C_i \text{ if } \text{Ratio}_i > T_{\text{CE}}, \text{ otherwise send } S_i. \tag{10}$$

The segment metadata encodes whether compression is applied and records the original length, allowing the device to correctly authenticate, decompress, and install each segment.

5.4 Manifest and Bootloader Integration

HASAC uses a signed manifest to bind firmware metadata to the update. The manifest specifies the firmware version, segment size, selected AEAD scheme, and a cryptographic hash of the complete firmware image. The manifest is authenticated using a public key provisioned on the device at manufacture time.

Upon receiving the manifest, the device verifies its signature and version constraints before entering the update phase. Each segment is authenticated individually using AEAD, decompressed if required, and written to a staging region in flash. After all segments are installed, the device verifies the hash of the reconstructed image against the manifest before activating the update.

This design ensures integrity and authenticity at both segment and image granularity while allowing HASAC to adapt cryptography and compression without weakening security guarantees. Compression is always applied before encryption to avoid information leakage through the ciphertext structure.

6 Implementation

We implement HASAC on Zephyr OS v4.0.99 using nRF Connect SDK. The prototype targets an nRF5340DK using a Cortex-M33 microcontroller and interfaces with an external NB-IoT modem over UART. The implementation focuses on the device-side update pipeline and on collecting precise timing and energy measurements required to instantiate the HASAC decision logic and energy model.

6.1 Cryptographic back ends

HASAC supports two authenticated encryption back ends that are selected at runtime based on hardware capabilities. On platforms equipped with a hardware security module, AES-128-CCM is executed through the PSA Crypto API and is transparently offloaded to the Arm CryptoCell present on the nRF5340. On platforms without hardware acceleration, we use a software-optimised ChaCha20-Poly1305 implementation from the same cryptographic stack to ensure comparable integration and avoid hand-tuned assembly.

All benchmarks explicitly include nonce handling, authentication tag generation and verification, and buffer management. Key import is performed once

Table 1: Measured cryptographic and decompression costs on a Cortex-M33 for a 1 kB payload. Results are averaged over 100 iterations.

Operation	**Time** (ms)	**Energy** (μJ)	**CPU** cycles	**Throughput** (kB/s)
Software AES-128-CCM	9.64	289.1	616,700	103.8
Software ChaCha20-Poly1305	1.59	47.7	101,700	629.1
Hardware AES-128-CCM	0.36	10.8	23,100	2,770.0
LZ4 decompression	0.02	0.7	1,445	44,280.0

per update session and is therefore excluded from per-segment measurements. Each benchmark encrypts or decrypts a 1024-byte payload for 100 iterations to amortise measurement noise.

As shown in Table 1, the measured results for a single 1024-byte segment at an assumed active power of 30 mW are as follows. Software AES-128-CCM requires approximately 9.6 ms and consumes 289 μJ. Software ChaCha20-Poly1305 completes in 1.6 ms at 48 μJ. Hardware-accelerated AES-128-CCM completes in 0.36 ms and consumes approximately 11 μJ. These results confirm that, once hardware acceleration is available, cryptographic processing contributes a negligible fraction of the total update energy.

6.2 Segment format and update pipeline

HASAC processes firmware images as fixed-size segments of 1024 bytes aligned to the flash programming granularity of the target device. Each segment is independently authenticated and optionally compressed.

On the server side, compression is applied first, followed by authenticated encryption. For a segment S_i with optional compressed form C_i, the transmitted payload is

$$\text{payload}_i = \begin{cases} \text{AEAD.Enc}(K, \text{nonce}_i, C_i, \text{aad}_i) & \text{if compressed,} \\ \text{AEAD.Enc}(K, \text{nonce}_i, S_i, \text{aad}_i) & \text{otherwise.} \end{cases}$$

The associated data aad_i binds the segment index, firmware version, compression flag, and expected uncompressed length.

On the device, each segment is first authenticated and decrypted. Decompression is applied only if indicated by the compression flag. The resulting plaintext is written to a staging region in flash. An end-to-end image hash included in the manifest detects missing, reordered, or replayed segments.

6.3 Measured timing and energy parameters

We measure per-segment timing for three phases using GPIO instrumentation and a logic analyser (Nordic Power Profiler Kit II). For a 1024-byte segment

under moderate coverage conditions, the measured mean timings are: NB-IoT reception approximately 1150 ms, UART transfer approximately 850 ms, and microcontroller processing, including cryptography and flash programming, approximately 85 ms. The energy costs correspond to approximately 34.5 mJ for radio reception, 25.5 mJ for UART transfer, and 2.6 mJ for processing.

LZ4 decompression of a 1024-byte segment requires approximately 0.022 ms and consumes less than 1 μJ. This cost is orders of magnitude smaller than the communication energy and motivates selective compression in deep coverage scenarios.

6.4 Break-even compression rule

HASAC applies compression only when the energy saved by transmitting fewer bytes exceeds the energy required for decompression. Let b be the uncompressed segment size and c the compressed size. Compression is beneficial when

$$\Delta E(b, c) = (b - c) \cdot e_{\text{comm}} - b \cdot e_{\text{decomp}} > 0,$$

where e_{comm} is the communication energy per byte and e_{decomp} is the decompression energy per byte.

The value of e_{comm} is derived from measured radio and UART costs and increases with NB-IoT coverage enhancement due to repetition. Since e_{decomp} remains constant, HASAC naturally becomes more aggressive in poor coverage conditions and conservative in good coverage. This rule directly instantiates the link-aware compression policy used in the evaluation.

7 Evaluation

This section compares a conventional secure firmware OTA baseline against HASAC for a 128 kB firmware image segmented into 1 kB blocks ($b = 1024$ bytes, $N = 128$). We use the measured per-segment timings from Sect. 4 and the constant active power of $P_{\text{active}} = 30$ mW. We report results for three NB-IoT coverage enhancement settings, modelled via a radio repetition factor $r \in \{1, 16, 128\}$ to represent CE0, CE1, and CE2.

7.1 Energy model instantiation

For a segment of size x bytes, we model the time as

$$t(x; r) = r \cdot t_{\text{radio}} \cdot \frac{x}{b} \; + \; t_{\text{UART}} \cdot \frac{x}{b} \; + \; t_{\text{proc}}(x), \tag{11}$$

and the corresponding energy as $E(x; r) = P_{\text{active}} \cdot t(x; r)$. We then sum over all segments. We anchor the communication timings to the measured 1 kB segment profile:

$$t_{\text{radio}} = 1.150 \text{ s}, \quad t_{\text{UART}} = 0.850 \text{ s}, \quad t_{\text{proc,meas}} = 0.085 \text{ s}.$$

To incorporate the cryptographic and decompression microbenchmarks without double-counting, we split processing as

$$t_{\text{proc}}(x) = t_{\text{other}} + t_{\text{crypto}}(x) + t_{\text{decomp}}(x), \tag{12}$$

where $t_{\text{other}} = t_{\text{proc,meas}} - t_{\text{AES-CCM,SW}}$ captures flash programming and non cryptographic logic.

7.2 Baselines

We consider a baseline that reflects common practice in constrained deployments: no compression, uniform authenticated encryption per segment, and no explicit adaptation to the radio conditions. Concretely, the baseline uses software AES-128-CCM and transmits all segments uncompressed.

Under this baseline, the total energy for a 128 kB update is:

$$E_{\text{base}}(r) = N \cdot P_{\text{active}} \cdot \Big(r \cdot t_{\text{radio}} + t_{\text{UART}} + t_{\text{other}} + t_{\text{AES-CCM,SW}} \Big). \tag{13}$$

7.3 HASAC configurations

HASAC adds two adaptation points. First, it selects a cryptographic back end based on whether the platform exposes a hardware security module. Second, it applies selective compression to reduce transferred bytes when it is energetically beneficial. To keep the evaluation fully specified, we instantiate the byte reduction achieved by HASAC as a coverage-dependent mean reduction in transferred bytes:

$$\gamma_{\text{CE0}} = 0.10, \quad \gamma_{\text{CE1}} = 0.20, \quad \gamma_{\text{CE2}} = 0.30,$$

meaning that the average transmitted segment size becomes $x = (1-\gamma) \cdot b$. With compression ratio γ, the HASAC energy for a device with a hardware security module is:

$$\begin{aligned} E_{\text{HASAC,HSM}}(r, \gamma) = N \cdot P_{\text{active}} \cdot \Big(& r \cdot t_{\text{radio}} \cdot (1-\gamma) + t_{\text{UART}} \cdot (1-\gamma) \\ & + t_{\text{other}} + t_{\text{AES-CCM,HW}} + t_{\text{LZ4,decomp}} \Big). \end{aligned} \tag{14}$$

For a device without a hardware security module, the $t_{\text{AES-CCM,HW}}$ gets replaced by $t_{\text{ChaCha20-Poly1305,SW}}$.

7.4 Results and Discussion

Table 2 summarises the modelled total energy for a 128 kB secure firmware update under three NB-IoT coverage enhancement levels. We compare a conventional baseline with HASAC configured both with and without a hardware security module. Two key observations emerge.

First, the dominant energy savings achieved by HASAC stem from reducing the number of bytes transmitted over the radio link. These savings increase

Table 2: Modelled total energy for a 128 kB secure firmware update. CE0, CE1, and CE2 are represented by repetition factors $r \in \{1, 16, 128\}$. HASAC uses mean byte reductions $\gamma_{\mathrm{CE0}} = 0.10$, $\gamma_{\mathrm{CE1}} = 0.20$, and $\gamma_{\mathrm{CE2}} = 0.30$.

Coverage	**Baseline** (J)	**HASAC (HSM)** (J)	**HASAC (No HSM)** (J)	**Savings (HSM No HSM)** (% / %)
CE0 ($r = 1$)	8.006	7.203	7.207	10.04 / 9.98
CE1 ($r = 16$)	74.246	59.425	59.430	19.96 / 19.95
CE2 ($r = 128$)	568.838	398.265	398.270	29.99 / 29.99

monotonically with coverage enhancement. In CE0, where repetition is minimal and communication is comparatively cheap, HASAC yields a modest reduction of approximately 10%. In CE2, where each transmitted byte is multiplied by a large repetition factor, HASAC reduces the total update energy by roughly 30%. This confirms that under NB-IoT conditions, optimisation of transferred bytes has a far greater impact than optimisation of local computation.

Second, the difference between HASAC with and without a hardware security module is negligible in the full FOTA energy budget. Although hardware-accelerated cryptography substantially reduces the local cryptographic cost per segment, this reduction is dominated by the radio and UART contributions. As a result, the total energy consumed by HASAC with hardware support differs by less than 1% from HASAC without hardware support across all coverage levels. This behaviour is consistent with the microbenchmarks reported earlier and demonstrates that cryptographic computation, whether accelerated or not, is not the limiting factor in secure firmware updates.

These results have two important implications. First, they explain why HASAC configurations with and without a hardware security module converge to nearly identical system-level energy consumption. The cryptographic back end influences CPU time but does not materially affect the dominant communication costs. Second, they justify HASAC's design choice to treat communication as the primary optimisation target and cryptography as a secondary, hardware-dependent concern. Overall, the evaluation confirms the central message of this paper. For secure firmware updates over NB-IoT, energy efficiency is primarily determined by radio and interconnect behaviour rather than cryptographic computation. Effective update mechanisms must therefore prioritise adaptive byte reduction and link-aware policies over micro-optimisations of crypto-primitives.

8 Conclusion

This paper presents HASAC, an energy-adaptive framework for secure firmware updates in Critical Infrastructure IoT deployments operating over NB-IoT. Through implementation measurements on a Cortex-M33 platform and a calibrated energy model, we demonstrated that the dominant cost of secure updates stems

from communication and interconnect overhead, rather than cryptographic computation. Even when hardware security modules are unavailable, the differences between cryptographic back ends are small within the full FOTA energy budget. HASAC exploits this observation by adapting cryptography to device capabilities and applying selective, link-aware compression to reduce transmitted bytes only when it is energetically beneficial. Model-based evaluation demonstrates that this approach yields modest savings in good coverage and substantial reductions, up to around 30%, in deep coverage conditions typical of critical infrastructure deployments. The results suggest that future secure update mechanisms should treat energy as a first-class security constraint and prioritise communication efficiency over micro-optimisations of cryptographic primitives. HASAC provides a practical blueprint for doing so without weakening security guarantees.

Acknowledgements

This work was supported in part by the Flemish Government through the Cybersecurity Research Program with grant number VOEWICS02, and by the European Union's Horizon Research and Innovation program under grant agreement No. 101119747 (TELEMETRY).

References

1. Asokan, N., Nyman, T., Rattanavipanon, N., Sadeghi, A.R., Tsudik, G.: ASSURED: Architecture for Secure Software Update of Realistic Embedded Devices. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems
2. Doddapaneni, K., Lakkundi, R., Rao, S., Kulkarni, S.G., Bhat, B.: Secure FoTA Object for IoT. In: 2017 IEEE 42nd Conference on Local Computer Networks Workshops (LCN Workshops) (2017)
3. Dworkin, M.: Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality. NIST Special Publication 800-38C
4. El Jaouhari, S., Bouvet, E.: Secure Firmware Over-The-Air Updates for IoT: Survey, Challenges, and Discussions. Internet of Things **18**, 100508 (2022)
5. Kolehmainen, A.: Secure Firmware Updates for IoT: A Survey. In: 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData) (2018)
6. Langiu, A., Boano, C.A., Schuß, M., Römer, K.: UpKit: An Open-Source, Portable, and Lightweight Update Framework for Constrained IoT Devices. In: 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS) (2019)
7. lZ4: Lz4 compression algorithm, https://lz4.github.io/lz4/
8. Moran, B., Tschofenig, H., Birkholz, H.: A Manifest Information Model for Firmware Updates in Internet of Things (IoT) Devices. RFC 9124 (Jan 2022)
9. Moran, B., Tschofenig, H., Brown, D., Meriac, M.: A Firmware Update Architecture for Internet of Things. RFC 9019 (Apr 2021)
10. Nir, Y., Langley, A.: ChaCha20 and Poly1305 for IETF Protocols (2018)
11. Tacchella, A., Beozzo, E., Crispo, B., Roveri, M.: Certified Secure Updates for IoT Devices. In: ICT Systems Security and Privacy Protection (IFIP SEC 2025)