Diamond: Design and Implementation of Breach-Resilient Authenticated Encryption Framework For Internet of Things

SAIF E. NOUMA, GOKHAN MUMCU, and ATTILA A. YAVUZ, University of South Florida, USA

Resource-constrained Internet of Things (IoT) devices, from medical implants to small drones, must transmit sensitive telemetry under adversarial wireless channels while operating under stringent computing and energy budgets. Authenticated Encryption (AE) is essential for ensuring confidentiality, integrity, and authenticity. However, existing lightweight AE standards lack forward-security guarantees, compact tag aggregation, and offline-online (OO) optimizations required for modern high-throughput IoT pipelines. We introduce `Diamond`, the first provable secure Forward-secure and Aggregate Authenticated Encryption (FAAE) framework that extends and generalizes prior FAAE constructions through a lightweight key evolution mechanism, an OO-optimized computation pipeline, and a set of performance-tiered instantiations tailored to heterogeneous IoT platforms. `Diamond` substantially reduces amortized offline preprocessing (up to 47%) and achieves up to an order-of-magnitude reduction in end-to-end latency for large telemetry batches. Our comprehensive evaluation across 64-bit ARM Cortex-A72, 32-bit ARM Cortex-M4, and 8-bit AVR architectures confirms that `Diamond` consistently outperforms baseline `FAAE` variants and NIST lightweight AE candidates across authenticated encryption throughput and end-to-end verification latency while maintaining compact tag aggregation and strong breach resilience. We formally prove the security of `Diamond` and provide two concrete instantiations optimized for compliance and high efficiency. Our open-source release enables reproducibility and seamless integration into IoT platforms.

CCS Concepts: • **Security and Privacy → Cryptography**.

Additional Key Words and Phrases: Lightweight Cryptography, AE, Wireless IoT

1 INTRODUCTION

The rapid growth of modern cyber-physical infrastructures has driven the widespread deployment of battery-powered embedded devices operating in unmanned and low-maintenance environments, ranging from medical implants and wearable devices [34] to smart city sensors [44], autonomous surveillance nodes [22], and unattended military ground sensors [31]. Such systems are increasingly operating autonomously and interacting with remote services, forming large-scale, distributed Internet of Things (IoT) networks for monitoring, analytics, and actuation [12].

Ensuring secure and efficient communication in constrained IoT networks remains challenging due to the inherent constraints of IoT hardware, including limited computational capabilities, restricted bandwidth, and stringent energy budgets [36, 38, 47]. IoT applications require robust security guarantees of data confidentiality, integrity, and authenticity, particularly when transmitting sensitive mission data or control instructions. Authenticated Encryption (AE) [7, 18, 29] provides the aforementioned security services by combining Symmetric Encryption (SE) with Message Authentication Codes (MACs), making it suitable for wireless and resource-constrained environments. Although widely used AE schemes such as AES-GCM [26] benefit from hardware acceleration and offer high throughput on many platforms [15], they do not fully leverage the potential of IoT infrastructures through lightweight cryptography, nor do they address the unique performance and security requirements imposed by the adversarial nature of unattended IoTs.

The compromise of long-term credentials remains a dominant attack vector in adversarial settings where attackers obtain full (physical) access to embedded or resource-constrained platforms [35]. Forward security aims to mitigate the risk of exposure of long-term secret keys [4, 45]. In essence, it

Authors' address: Saif E. Nouma, saifeddinenouma@usf.edu; Gokhan Mumcu, mumcu@usf.edu; Attila A. Yavuz, attilaayavuz@usf.edu, University of South Florida, 3720 Spectrum Blvd, Interdisciplinary Research Building (IDR)-400, Tampa, Florida, USA, 33612.

enforces key evolution such that exposure of the *current* secret key does not jeopardize confidentiality or authenticity of data produced in *prior* time periods [8, 45]. In other words, secret keys are updated forward in time (e.g., via a one-way function) so that past keys remain protected even after a compromise event [2]. Thus, forward security is vital against several attack vectors (e.g., malware and impersonation attacks) that target IoT networks [43]. Forward security is especially relevant for IoT communication networks against eavesdropping attacks, where adversaries may passively intercept encrypted traffic and later attempt decryption or tag forgery of encrypted data, given the compromised secret keys. While previous works have applied forward security in both private-key (e.g., symmetric encryption [2, 8, 23] and MACs [24, 48]) and public-key (e.g., digital signatures [6, 28]) settings, it remains under-explored in symmetric-key AE settings [29].

In parallel, aggregation is crucial to improve the efficiency of wireless communication in low-power wireless networks [42]. Aggregation compresses multiple per-message authentication tags into a single, compact aggregate authenticator via algebraic (e.g., XOR-sums [17, 42]) or hash-based (e.g., cryptographic hash-based [24, 48]) compositions. Therefore, it reduces per-message communication overhead and verification bandwidth, enabling high-throughput verification of large message batches. Aggregation is vital for bandwidth- and energy-constrained networks (e.g., LoRaWAN, ZigBee) where uplink rates are often limited to just a few kilobits per second. As such, aggregation has been extensively used in digital signatures (e.g., [28]) and MACs (e.g., [24, 29, 40]). In contrast, only a few works [29] propose aggregate MACs integrated into AEs.

Improving the efficiency of current AE standards is also equally critical for extending the energy lifetime of battery-powered constrained devices and for improving the End-to-End (E2E) delay of received data packets at the verifier side. OO cryptography amortizes the computational cost across an offline precomputation phase, where expensive input-independent cryptographic operations are performed, whereas the online phase executes constant-time input-dependent operations [13, 41]. Several works investigate OO properties in public-key (e.g., digital signature [33]) and symmetric-key encryption (e.g., [13]). However, we observe that leveraging OO for AEs has been overlooked, with only recent work [29] precomputing costly computations during the offline phase and demonstrating substantial performance improvements compared to vanilla AE standards. In contrast, the NIST lightweight AE standard, Ascon, does not possess OO capabilities due to its integrated sponge-based structure [20], which shows the bias in comparing NIST lightweight cryptography against widely used AE standards like AES-GCM.

In the following, we review related works, with an emphasis on the aforementioned properties.

1.1 Related Work and Limitations

Ascon [37] has been selected as the NIST Lightweight Cryptography (LWC) standard (NIST 800-232 standard) for both AE and hash functionalities. Although Ascon outperforms the currently deployed AE standards—AES-GCM (NIST SP 800-38D) [32] and ChaCha20-Poly1305 (RFC 8439) [27]—on small microcontrollers, it lacks OO cryptography, which stems from its sponge-based AE structure and the design choices to meet lightweight and side-channel-resilience goals. In particular, its integrated AE mode precludes ciphertext-independent AE structure, and its tag verification requires full payload decryption, making it incompatible with desirable efficient verification and low end-to-end delay. As noted in the NIST 800-232 standard [37], *"Ascon is developed to offer a viable alternative when the Advanced Encryption Standard (AES) may not perform optimally"*. In this work, we provide a comprehensive benchmarking and usage framework for AE standards, particularly when considering forward security, aggregation, and OO precomputation.

AES-GCM [32] is one of the most widely deployed authenticated-encryption (AE) standards, used in protocols such as DTLS 1.3 and IEEE 802.15.4 for constrained sensor and low-rate wireless personal-area networks (LR-WPANs). ChaCha20-Poly1305 [27], is also mandated in TLS 1.3 as the

preferred AE scheme for IoT-class processors (e.g., ARM Cortex-M) where hardware-accelerated AES is unavailable or inefficient. Both AES-GCM and ChaCha20-Poly1305 follow the same high-level design principle: *(i) Counter (CTR) Mode of Encryption.* providing OO Cryptography and parallelizable key-stream generation [13]. *(ii) Wegman-Carter MAC Authentication* [7]. instantiated with a fast universal hash function. AES-GCM employs AES-128 as its block cipher and GHASH (a binary-field polynomial hash) for authentication. In contrast, ChaCha20-Poly1305 uses the ChaCha20 stream cipher with Poly1305 as an efficient universal hash. Both AE standards can achieve OO property, which is especially relevant for low encryption overhead and efficient batch verification with a minimal end-to-end delay. Hiller et al. [13] showcase 75.9% latency reduction in encryption/decryption using AES in the CTR mode of operation. In practice, AES-GCM achieves high performance on platforms with specialized hardware instructions (e.g., AES-NI on Intel/AMD CPUs or embedded cryptographic accelerators). For example, the AE runtime of AES-GCM can be reduced by up to 70% when harnessing OO cryptography techniques [29]. ChaCha20-Poly1305, while typically slower on hardware-accelerated platforms, provides consistently fast pure-software performance across commodity hardware and low-end microcontrollers [7].

Existing AE primitives lack the aggregation property of authentication tags, i.e., they do not support compressing multiple tags into a single compact authenticator via a keyless aggregation method. In contrast, aggregation in the context of Aggregate MACs (AMACs) has been studied extensively [14, 17, 21, 24, 40, 42, 48]. Hash-based AMACs (e.g., [24, 48]) instantiate aggregation via recursive cryptographic hash calls and thereby inherit sequential immutability from the underlying cryptographic hash function but incur higher computational cost. Another line of work employs XOR-based aggregation (e.g., [9, 14, 21, 39, 40, 42]) due to its constant-time and linear-algebraic simplicity. Wagner et al. [42] provide a systematic analysis of different XOR-based aggregation methods across various communication channels, offering guidance on selecting the aggregation method based on network topology and conditions. More recently, Wagner et al. [40] implement aggregation methods on the Datagram Transport Layer Security (DTLS) 1.3 protocol and demonstrate up to 50% improvement in goodput and 15% reduction in energy usage on an embedded testbed equipped with an ARM Cortex-M4 microcontroller.

Existing work on forward security in symmetric-key cryptography originates with Bellare et al. [2], which provides formal treatments and security for forward-secure MACs and encryption but only briefly discusses AEs. In parallel, Ma et al. [24] proposed the first forward-secure and aggregate MAC (FssAggMAC), combining key evolution and tag aggregation with order integrity. Yavuz et al. [48] improved FssAggMAC by designing more efficient hash-based forward-secure aggregate authentication tailored for unmanned sensor networks. Later, Hirose et al. [14] revisited forward-secure AMACs with generalized constructions and formal security treatment. Beyond MACs, forward-secure encryption has also been developed, including puncturable and revocation-friendly primitives [11] as well as fast-forwardable key-evolution [8, 25].

To the best of our knowledge, no provable-secure FAAE primitive has been proposed. Prior works address only forward-secure encryption and AMACs, none of which leverage full OO potential.

Overall, there is a large gap in the state-of-the-art in achieving a lightweight, breach-resilient, precomputable, and compact authenticated encryption scheme, specifically tailored for constrained IoT environments by leveraging the aforementioned security and performance properties.

1.2 Our Contributions

In this work, *we propose* `Diamond`*, the first, to the best of our knowledge, provable-secure authenticated encryption framework that simultaneously achieves OO cryptography, tag aggregation, and forward security, ensuring minimal energy cost, low end-to-end delay with small bandwidth usage, and breach*

resilience against compromise attacks, which we believe to be attractive features towards efficient and robust IoT networks. The desirable properties of Diamond *include the following:*

• **Provable-secure Forward-secure and Aggregate AE Framework (FAAE)**: Diamond introduces a unified authenticated encryption framework that composes forward-secure key evolution with sequential aggregate authentication via universal hash-based MACs. Diamond uncovers synergies among forward-secure encryption and aggregate universal hash-based MACs overlooked by prior approaches while being backwards compatible with current standards and implementations. Therefore, it achieves modular construction in which per-message keys are evolved using lightweight pseudo-random functions (PRFs), and authentication tags are incrementally compressible without dropping unforgeability. Diamond achieves high efficiency and security while opening a path for alternative constructions through generalizable integrations and ease of implementation.

• **High Online Computation Efficiency via OO Cryptography**: OO cryptography amortizes input-independent cryptography computations into an offline stage, leveraging idle and power-transfer periods, thereby reducing the online latency for performance-critical use-case scenarios. Although well-studied for digital signatures, OO remain largely unexamined in the FAAE context. Diamond explores the synergies of various universal MACs and AEs in OO settings and achieves a significant online performance gain with only a modest increase in memory consumption. As highlighted in Table **??** and Section 5, Diamond consistently outperforms vanilla FAAE standards during the online authenticated encryption across architectures. For example, our $\texttt{Diamond}_2$ instantiation achieves up to 3.5× online speedup for a batch of 1024 16-byte telemetry on 32-bit Cortex-M4 embedded processor, with only 32KB extra storage. Even under stringent computing constraints on an 8-bit AVR ATmega2560, $\texttt{Diamond}_2$ exhibits high efficiency with 3.8× speedup compared to $\texttt{FAAE}_2$ (instantiated from the NIST lightweight standard, Ascon [37]). In wearable medical settings, this results in significant savings in battery life during online operations, as the next batch of OO keys can be supplied when the device is recharged while telemetry is uploaded. In aerial drone applications, latency reduction can improve flight safety and real-time telemetry transmission.

• **Minimal Offline Preprocessing via Lightweight Key Evolution**: In contrast to the key evolution from cryptographic hashing in Graphene counterpart [29], Diamond employs a more lightweight and computation-efficient evolution mechanism from PRFs, enabling fast update of key materials. Diamond achieves up to 40% and 47% reductions in offline preprocessing runtime compared to Graphene counterpart on two representative embedded platforms, a 64-bit ARM Cortex-A72 processor and an 8-bit AVR ATmega2560 microcontroller, respectively.

• **Low End-to-End Verification Latency For Large Payload Batches**: The proposed Diamond instantiations substantially minimize the E2E verification latency by coupling lightweight universal hashing and efficient online encryption. Our two instantiations, $\texttt{Diamond}_1$ and $\texttt{Diamond}_2$, exhibit efficient E2E scaling across message sizes (16-128 bytes). For example, when the IoT device is a constrained 8-bit AVR ATmega2560 and the IoT server is a commodity hardware, $\texttt{Diamond}_1$ achieves up to 3× lower E2E delay compared to baseline FAAE counterparts. The highly efficient variant $\texttt{Diamond}_2$, leveraging ChaCha20-Poly1305 (RFC standard), yields even higher acceleration with 8.1× lower E2E latency than $\texttt{Diamond}_1$ and up to 3.8× lower delay than the fastest NIST lightweight FAAE counterparts across both 16-byte and 128-byte payloads.

• **Comprehensive Benchmarks with Full-Fledged Implementation**: We implemented Diamond on commodity hardware (x86/64) and three constrained embedded devices: a 64-bit ARM Cortex-A72 processor, a 32-bit ARM Cortex-M4 microcontroller, and an 8-bit AVR ATmega2560 microcontroller. Our performance results highlight the efficiency of each instantiation, guiding its application context accordingly. To encourage public testing and reproducibility, we release our implementation

at: https://github.com/SaifNouma/Diamond . To the best of our knowledge, Diamond is the first comprehensive open-source FAAE framework on the aforementioned platforms.

1.3 Improvements over Preliminary Version

This work is an extension of our conference publication at IEEE MILCOM 2025 [29], with extended algorithmic descriptions and optimizations, formal and in-depth security analysis, and extended performance analysis. Our journal extension makes substantially expanded treatment, with more than doubling technical content, and introduces the following major improvements and contributions:

1) Additional Experimental Evaluation and Analysis on Constrained Devices. We significantly expand the performance analysis by incorporating two more constrained platforms, a 64-bit ARM Cortex-A72 edge processor and an ultra-low-power 8-bit AVR ATmega2560, with full benchmarking on Diamond instantiations and selected counterparts. Our extended analysis rigorously quantifies offline/online execution, their energy usage on the 8-bit platform, and end-to-end (E2E) batch verification latency. Diamond consistently outperforms Graphene on the selected platforms.

2) Comprehensive Algorithmic Formalization with Further Optimizations. We provide a comprehensive formalization of the proposed scheme (Section 4). We elaborated the main building blocks to construct Diamond: (i) forward-secure symmetric encryption based on counter mode of operation and (ii) forward-secure and aggregate MAC based on Carter-Wegman construction, in Fig. 3 and Fig. 4, respectively. Moreover, unlike hash-based key evolution used in initial Graphene, Diamond employ a more efficient key evolution instantiated from pseudo-random functions (PRF).

3) Formal and In-Depth Security Analysis. We introduce formal security definitions and prove that Diamond achieves forward-secure confidentiality, integrity, and authenticity in the standard model (Section 6), through tight reductions to the pseudo-randomness of the underlying PRF and the universality of the hash function. This yields minimal and transparent assumptions, which are substantially stronger than those attained in the initial Graphene.

2 PRELIMINARIES

2.1 Notations

The notations and acronyms are described in Table 1.

Table 1. List of Notations, Acronyms, and Their Descriptions

Notation	Description	Acronym	Description
$X \| Y$	Concatenation of variables X and Y	IoT	Internet of Things
$\|X\|$	Bit length of variable X	PRF	Pseudo-Random Function
$X \xleftarrow{\$} \mathcal{S}$	X sampled uniformly at random from set $\mathcal{S}$	CTR	Counter Mode (of encryption)
$\{0,1\}^*$	Set of binary strings of arbitrary length	OO	Offline-Online Cryptography
$\mathbb{Z}_q$	Finite field of order q	[F]SE	[Forward-secure] Symmetric-key Encryption
$\vec{X} = \{X_1, \ldots, X_n\}$	Collection of items; $x = \|\vec{X}\|$	[F][A]MAC	[Forward-secure] [Aggregate] Message Authentication Codes
		[F][A]AE	[Forward-secure] [Aggregate] Authenticated Encryption

2.2 Building Blocks

In the following, we define the main cryptographic primitives used in our proposed scheme.

Family of Functions with Extended Properties. A family of functions is defined as follows:

Definition 2.1. *A family of functions is represented as ($F \leftarrow \{f : \mathcal{K} \times \mathcal{M} \rightarrow \mathcal{T}\}$) where $\mathcal{K}$, $\mathcal{M}$, and $\mathcal{T}$ denote the key, input, and output space, respectively. We denote by κ, m, and τ the bit sizes of $\mathcal{K}$, $\mathcal{M}$, and $\mathcal{T}$, respectively. f is defined as follows:*

- *$\tau \leftarrow f_K(M)$: Given a key $K \in \mathcal{K}$ and an input $M \in \mathcal{M}$, it outputs $\tau \in \mathcal{T}$.*

Definition 2.2. *A family of functions $F \leftarrow \{f : \mathcal{K} \times \mathcal{M} \rightarrow \mathcal{T}\}$ is called a universal family of hash functions (UH) if $m < \tau$ and for all $M, M' \in \mathcal{M}$ with $M \neq M'$ and $K \xleftarrow{\$} \{0,1\}^\kappa$, $\Pr[\mathtt{UH}_K(M) = \mathtt{UH}_k(M')] < 2^{-\tau}$. Moreover, UH is ϵ-almost universal if $\forall M \neq M'$, $\Pr[\mathtt{UH}_k(M) = \mathtt{UH}_k(M')] < \epsilon$.*

Definition 2.3. *A family of functions $F \leftarrow \{f : \mathcal{K} \times \mathcal{M} \rightarrow \mathcal{T}\}$ is called a family of pseudo-random functions PRF if a random function $PF \leftarrow \{\mathtt{PRF}_K : \mathcal{M} \rightarrow \mathcal{T}, \forall K \in \mathcal{K}\}$ is indistinguishable from a random function uniformly selected from the set $\mathcal{F} = \{f : \mathcal{M} \rightarrow \mathcal{T}\}$ having domain $\mathcal{M}$ and range $\mathcal{T}$.*

Definition 2.4. *A pseudo-random generator (PRG) is a function $\mathtt{PRG} : \mathcal{K} \rightarrow \mathcal{K} \times \mathcal{T}$ that extends a uniformly random input $K \in \mathcal{K}$ to a longer random string $(K', Y) \in \mathcal{K} \times \mathcal{T}$.*

Definition 2.5. *A forward-secure pseudo-random generator (FPRG) is a scheme initially introduced by Bellare et al. [2] and consists of two algorithms $\mathtt{FPRG} = (\mathtt{Kg}, \mathtt{Upd})$, defined as follows:*

- *$S_1 \leftarrow \mathtt{FPRG.Kg}(1^\kappa, n)$: Given security parameter κ, it outputs an initial key S_1.*
- *$(S_{i+1}, K_i) \leftarrow \mathtt{FPRG.Upd}(S_i)$: Given S_i, it returns S_{i+1} and a key K_i if $i \leq n$ and $\perp$ otherwise.*

Bellare et al. [2] describes how FPRG with $\mathtt{FPRG.Upd} : \mathcal{K} \rightarrow \mathcal{K} \times \mathcal{T}'$ can be constructed from a secure $\mathtt{PRF} : \mathcal{K} \times \mathcal{M} \rightarrow \mathcal{T}$ via $\mathtt{FPRG.Upd}(K_i) \leftarrow \mathtt{PRF}(K_i, 1) \| \ldots \| \mathtt{PRF}(K_i, \lceil \frac{(\tau' + \kappa)}{\tau} \rceil), \forall i = 1, \ldots, n$.

Message Authentication Codes (MAC). A MAC enables a receiver to verify the integrity of message M via an authentication tag σ after transmission, given a shared secret key K. Formally:

Definition 2.6. *A Message Authentication Code $\mathtt{MAC} = (\mathtt{Kg}, \mathtt{Sign}, \mathtt{Ver})$ is a triple of algorithms:*

- *$k \leftarrow \mathtt{MAC.Kg}(1^\kappa)$: It takes the security parameter κ, and generates a secret key $K \xleftarrow{\$} \mathcal{K} = \{0,1\}^\kappa$.*
- *$\sigma \leftarrow \mathtt{MAC.Sign}(K, M)$: It takes $K \in \mathcal{K}$ and a message $M \in \mathcal{M}$, and outputs a tag $\sigma \in \mathcal{T}$.*
- *$b \leftarrow \mathtt{MAC.Ver}(K, M, \sigma)$: It outputs $b = 1$ if (M, σ)-pair is valid, or $b = 0$ otherwise.*

Carter-Wegman Construction. A MAC scheme can be instantiated via Carter-Wegman [10] given an ϵ-almost universal hash (UH) and a secure PRF. Formally, a *universal* MAC (UMAC) is defined as follows:

$$\mathtt{UMAC.Sign}((K_1, K_2), M, N) = \mathtt{UH}(K_1, M) + \mathtt{PRF}(K_2, N), \text{ where } K_1, K_2 \in \mathcal{K}, M \in \mathcal{M}, \text{ and } N \in \mathcal{N},$$

That is, UMAC consists of a nonce-based MAC: given a message $M \in \mathcal{M}$, it computes a tag $\sigma \leftarrow \mathtt{UMAC.Sign}((K_1, K_2), M, N)$ where $\mathtt{PRF}(K_2, N)$ can be precomputed during the *offline phase*.

Definition 2.7. *A forward-secure and sequential-aggergate message authentication code $\mathtt{FAMAC} = (\mathtt{Kg}, \mathtt{Sign}, \mathtt{Upd}, \mathtt{Agg}, \mathtt{AVer})$ consists of five algorithms:*

- *$K_1 \leftarrow \mathtt{FAMAC.Kg}(1^\kappa, n)$: It takes the security parameter κ, the maximum number of generated tags n, and generates an initial secret key $K_1 \xleftarrow{\$} \mathcal{K} = \{0,1\}^\kappa$.*
- *$\sigma_i \leftarrow \mathtt{FAMAC.Sign}(K_i, M_i)$: It takes $K_i \in \mathcal{K}$ and a message $M_i \in \mathcal{M}$, and outputs a tag $\sigma_i \in \mathcal{T}$.*
- *$b \leftarrow \mathtt{FAMAC.AVer}(K_1, \mathbf{M}_{1,i}, \sigma_{1,i})$: Given K_1, it outputs $b = 1$ if $(\mathbf{M}_{1,i}, \sigma_{1,i})$ is valid, or $b = 0$ otherwise.*
- *$K_{i+1} \leftarrow \mathtt{FAMAC.Upd}(K_i, n)$: It takes $K_i \in \mathcal{K}$ and n, and outputs K_{i+1} if $i < n$ and $\perp$ otherwise.*
- *$\sigma_{1,i+1} \leftarrow \mathtt{FAMAC.Agg}(\sigma_{1,i}, \sigma_{i+1})$: Given a forward-secure and aggregate tag $\sigma_{1,i}$ and current tag σ_{i+1}, it returns a constant-size aggregate tag $\sigma_{1,i+1}$.*

Symmetric Encryption (SE). A symmetric encryption (SE) ensures the confidentiality of transmitted data between two parties given a shared secret key K. Formally, SE is defined as follows:

Definition 2.8. *A symmetric encryption scheme $\mathtt{SE} = (\mathtt{Kg}, \mathtt{Enc}, \mathtt{Dec})$ is a triple of algorithms:*

- *$K \leftarrow \mathtt{SE.Kg}(1^\kappa)$: Given the security parameter 1^κ, it returns a secret key K.*
- *$C \leftarrow \mathtt{SE.Enc}(K, M)$: Given a secret key $K \in \mathcal{K}$ a message $M \in \mathcal{M}$, it returns a ciphertext $C \in \mathcal{T}$.*
- *$M' \leftarrow \mathtt{SE.Dec}(K, C)$: Given a secret key $K \in \mathcal{K}$ and a ciphertext $C \in \mathcal{T}$, it returns a plaintext M'.*

Definition 2.9. *A forward-secure symmetric encryption scheme* $\mathtt{FSE} = (\mathtt{Kg}, \mathtt{Enc}, \mathtt{Dec}, \mathtt{Upd})$ *consists of four algorithms and is presented as follows:*

- $K_1 \leftarrow \mathtt{FSE.Kg}(1^\kappa, n)$*: Given the security parameter* $\kappa \in \mathcal{K}$ *and the maximum number of encryptions* n*, it returns an initial secret key* K_1*.*
- $C_i \leftarrow \mathtt{FSE.Enc}(K_i, M_i)$*: Given* $K_i \in \mathcal{K}$ *a message* $M_i \in \mathcal{M}$*, it returns a ciphertext* $C_i \in \mathcal{T}$*.*
- $M'_i \leftarrow \mathtt{FSE.Dec}(K, C)$*: Given a secret key* $K_i \in \mathcal{K}$ *and a ciphertext* $C_i \in \mathcal{T}$*, it returns a plaintext* M'_i*.*
- $K_{i+1} \leftarrow \mathtt{FSE.Upd}(K_i, n)$*: It takes* $K_i \in \mathcal{K}$ *and* n*, and outputs* K_{i+1} *if* $i < n$ *and* $\perp$ *otherwise.*

3 MODELS

System Model.

Our system model consists of two primary entities, as illustrated in Fig. 1:

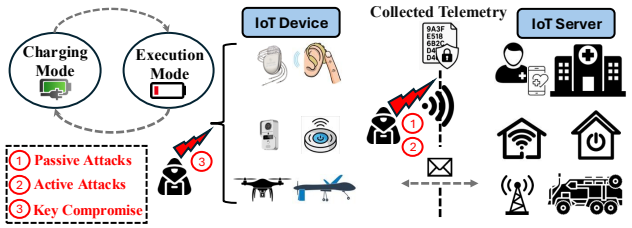


Fig. 1. Our system and threat models

- **Resource-constrained IoT devices.** (e.g., implantable medical sensors, autonomous unmanned aerial vehicles (UAVs), and smart-city robots) operate in adversarial and bandwidth-limited environments under stringent constraints on storage, memory, computation, and energy budgets. These endpoints continuously produce high-value telemetry (e.g., biomedical signals, flight stability parameters, or navigation/obstacle avoidance traces) that must remain confidentiality- and integrity-protected [28, 46]. Such store-and-forward pipelines arise in numerous domains, including but not limited to *(i) Medical IoT (IoMT).* Implantable glucose monitors and medical pacemakers periodically transmit physiological measurements to a patient-held gateway for clinical triage [34]. In adversarial settings, hackers exploit vulnerabilities (e.g., in Medtronic devices) to enable remote tampering that could alter insulin delivery or pacemaker rhythms, potentially endangering lives. *(ii) Autonomous UAV swarms.* Low-power drones perform reconnaissance and/or disaster-response mapping and generate mission telemetry that is stored and later relayed to a nearby edge node [31]. In 2024, the Federal Aviation Administration (FAA) estimates 2.8 million small drones in US airspace[1], which underscores the critical need for trustworthy and secure communication in drone networks, especially amid escalating cyber threats[2].
- **IoT Server (Edge Verifier).** is a storage- and computation-resourceful entity, being a patient's phone or a clinical gateway in IoMT, or a nearby access point in smart-home and smart-city applications. Upon receiving encrypted/authenticated payloads, the server performs verification and decryption and optionally offloads the data to a cloud server for analytics and autonomous actuation. We assume that the two main entities pre-share an initial secret key.

FAAE Model. A Forward-secure and sub-Aggregate Authenticated Encryption (FAAE) aims to achieve the security of both FSE and FAMAC, i.e., *data confidentiality* of the communicated traffic and

[1]https://www.faa.gov/data_research/aviation/aerospace_forecasts/2025-uas-and-aam-summary.pdf
[2]https://www.trendmicro.com/en_us/research/25/e/earth-ammit.html

data integrity and authenticity of generated encrypted data. Formally, an FAAE scheme consists of five algorithms (Kg, Upd, AuthEnc, Agg, AverDec):

- $\boldsymbol{K_1} \leftarrow \texttt{FAAE.Kg}(1^\kappa, n, b)$: Given the security parameter 1^κ, the maximum messages to be processed n, and the batch size b, it returns an initial secret key $\boldsymbol{K_1}$. $\boldsymbol{K_1}$ contain a single key when instantiated with a unified AE or multiple keys in case of generic AE construction via FSE and an FAMAC.
- $\boldsymbol{K_{i+1}} \leftarrow \texttt{FAAE.Upd}(\boldsymbol{K_{i+1}})$: Given the current secret key $\boldsymbol{K_i}$, it returns an updated K_{i+1}.
- $(C_i, \sigma_i) \leftarrow \texttt{FAAE.AuthEnc}(\boldsymbol{K_i}, M_i)$: Given the current secret key $\boldsymbol{K_i}$ and a message M_i, it returns a ciphertext C_i and its corresponding authentication tag σ_i.
- $\sigma_{i_1,i_2+1} \leftarrow \texttt{FAAE.Agg}(\sigma_{i_1,i_2}, \sigma_{i_2+1})$: Given an aggregate tag σ_{i_1,i_2} and σ_{i_2+1}, it returns σ_{i_1,i_2+1}.
- $\boldsymbol{M_i} \leftarrow \texttt{FAAE.AVerDec}(\boldsymbol{K_i}, \boldsymbol{C_i}, \sigma_{i,i+b-1})$: Given the current secret key $\boldsymbol{K_i}$, a batch of ciphertexts $\boldsymbol{C_i} := (C_i, \ldots, C_{i+b-1})$, and an aggregate tag $\sigma_{i,i+b-1}$, it returns the batch of plaintexts $\boldsymbol{M_i} := (M_i, \ldots, M_{i+b-1})$ if and only if the pair $(\boldsymbol{C_i}, \sigma_{i,i+b-1})$ is valid.

The maximum number of messages to be processed n is divided into $\lfloor \frac{n}{b} \rfloor$ epochs, each epoch contains b messages. As illustrated in Fig. 2, the sender (IoT device) evolves the private key by invoking FAAE.Upd(.) after every FAAE.AuthEnc(.) call. Then, it aggregates the current tag σ_i with the aggregate tag in the current epoch by invoking FAAE.Agg(.) algorithm, except for the first tag in each epoch. Overall, FAAE enable the sender to reduce the communication overhead from $O(n)$ into an adjustable $O(\frac{n}{b})$ based on the epoch size b.

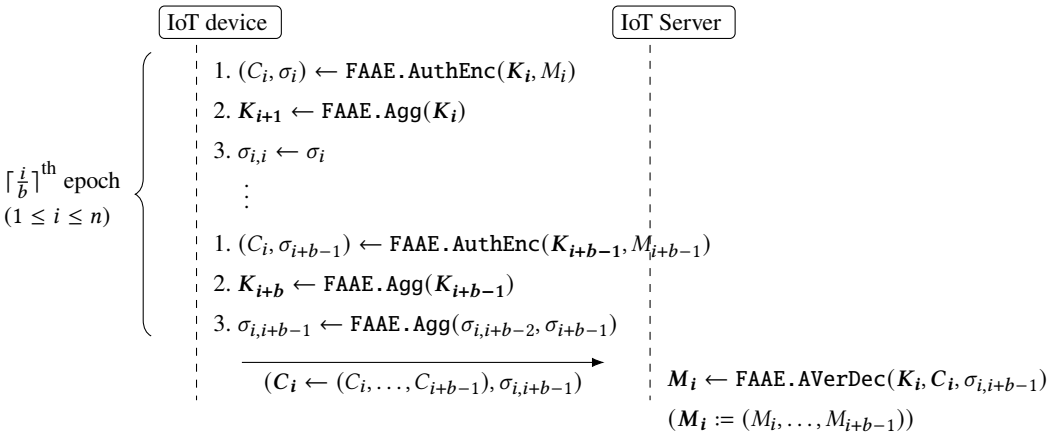


Fig. 2. FAAE Communication Flow

Threat Model. We assume a probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ capable of:

i. *Eavesdropping attacks.* monitoring and analyzing encrypted traffic over wireless channels.
ii. *Active attacks.* Modifying, dropping, or relaying encrypted data over communication channels.
iii. *Key compromise attacks.* Extracting the device's long-term secret keys during a system breach (e.g., malware) or a physical compromise (e.g., small drone captured). Upon a break-in, $\mathcal{A}$ aims to decrypt previously intercepted traffic using compromised secret keys and forge forward-secure and aggregate tags.

4 PROPOSED SCHEMES

A standard FAAE scheme typically applies conventional encryption with an HMAC. Its aggregation and forward security mechanisms are built on only a standard hash function [23]. However, this construction falls short of achieving the high efficiency and desirable properties outlined in Section 1,

such as precomputation OO capabilities for minimal online latency, flexible aggregation modes, all essential for our system model. While prior MACs have explored precomputation, these efforts were isolated, neglecting key evolution and encryption. In this work, we introduce Diamond an improved variant of Graphene and is designed to deliver near-optimal online computational efficiency and compactness, tailored for resource-constrained IoT environments.

Our generic Diamond construction instantiates an FAAE scheme by combining counter-mode (CTR) encryption with a universal MAC scheme (UMAC) following Encrypt-then-MAC construction [1], thereby allowing for efficient online running time thanks to the offline precomputation of both components. Diamond rely on FPRG to transform CTR-based SE and UMAC into an FSE and FAMAC.

Given the security parameter κ and the block size ℓ of the block cipher used in FSE, we define $\mathtt{PRF}_1 : \{0,1\}^\kappa \times \{0,1\}^\ell \to \{0,1\}^\ell$ and $\mathtt{PRF}_2 : \{0,1\}^\kappa \times \{0,1\}^\kappa \to \{0,1\}^\kappa$ as two PRF functions. For a maximum time periods of n, we instantiate FPRG based on $\mathtt{PRF}_2$ as follows:

$$\mathtt{FPRG.Upd}(S_i) \leftarrow (\mathtt{PRF}_2(S_i, 0), \mathtt{PRF}_2(S_i, 1)), \forall i = 1, \ldots, n$$

In the following, we present the building blocks of our proposed scheme Diamond: the generic constructions of FSE and FAMAC via CTR-based Enc and UMAC schemes, respectively.

4.1 Generic FSE Construction

Fig. 3 depicts the algorithmic description of our proposed CTR-based FSE construction. FSE is composed of the encryption scheme (SE) and the forward-secure pseudo-random generator (FPRG), where SE is instantiated using a PRF in the CTR mode of operation. Our construction harnesses the OO capabilities of CTR mode and batch precomputation of forward-secure private keys, thus enabling highly efficient online runtime during encryption and decryption phases.

4.1.1 Algorithmic Description. The key generation algorithm (FSE.Kg) accepts the security parameter κ and the maximum number of encryption operations n. Then, it generates an initial FPRG state S_1 and the private key K_1. FSE generates an initial internal state St that contains the current iteration i and randomly generated counter ctr to be used in the CTR-based encryption. We set the FSE secret key $\boldsymbol{K}_1$, which contains the FPRG internal state for the next time period and its secret key.

$\underline{\boldsymbol{K}_1 \leftarrow \mathtt{FSE.Kg}(1^\kappa, n)}$:
1: $S_1 \xleftarrow{\$} \{0,1\}^\kappa$
2: $(S_2, K_1) \leftarrow \mathtt{FPRG.Upd}(S_1)$
3: $St \leftarrow (i \leftarrow 1, ctr \leftarrow \{0,1\}^\ell)$
4: **return** $\boldsymbol{K}_1 \leftarrow (K_1, S_2)$

$\underline{\boldsymbol{K}_{i+1} \leftarrow \mathtt{FSE.Upd}(\boldsymbol{K}_i)}$ if $i \geq n$ then abort
1: $(S_{i+2}, K_{i+1}) \leftarrow \mathtt{FPRG.Upd}(S_{i+1})$
2: $St \leftarrow (i \leftarrow i+1, ctr \leftarrow ctr + \lceil m/\ell \rceil \bmod 2^\ell)$
3: **return** $\boldsymbol{K}_{i+1} \leftarrow (K_{i+1}, S_{i+2})$

$\underline{C_i \leftarrow \mathtt{FSE.Enc}(\boldsymbol{K}_i, M_i)}$:
$\underline{\tilde{C}_i \leftarrow \mathit{offline}(\boldsymbol{K}_i, ctr)}$
1: **for** $j = 1, \ldots, \lceil m/\ell \rceil$ **do**
2: $\quad \tilde{C}'_j \leftarrow \mathtt{PRF}_1(K_i, ctr + j)$
3: $\boldsymbol{K}_{i+1} \leftarrow \mathtt{FSE.Upd}(\boldsymbol{K}_i)$
4: $\tilde{C}_i \leftarrow \tilde{C}'_1 \| \ldots \| \tilde{C}'_{\lceil m/\ell \rceil}$
$\underline{C_i \leftarrow \mathit{online}(M_i, \tilde{C}_i)}$
5: **for** $j = 1, \ldots, \lceil m/\ell \rceil$ **do**
6: $\quad C'_j \leftarrow \tilde{C}'_j \oplus M'_j$
7: **return** $C_i \leftarrow C'_1 \| \ldots \| C'_{\lceil m/\ell \rceil}$

$\underline{M_i \leftarrow \mathtt{FSE.Dec}(\boldsymbol{K}_i, C_i)}$:
$\underline{\tilde{M}_i \leftarrow \mathit{offline}(\boldsymbol{K}_i, ctr)}$
1: **for** $j = 1, \ldots, \lceil c/\ell \rceil$ **do**
2: $\quad \tilde{M}'_j \leftarrow \mathtt{PRF}_1(K_i, ctr + j)$
3: $\boldsymbol{K}_{i+1} \leftarrow \mathtt{FSE.Upd}(\boldsymbol{K}_i)$
4: $\tilde{M}_i \leftarrow \tilde{M}'_1 \| \ldots \| \tilde{M}'_{\lceil c/\ell \rceil}$
$\underline{M_i \leftarrow \mathit{online}(C_i, \tilde{M}_i)}$
5: **for** $j = 1, \ldots, \lceil m/\ell \rceil$ **do**
6: $\quad M'_j \leftarrow \tilde{M}'_j \oplus C'_j$
7: **return** $M_i \leftarrow M'_1 \| \ldots \| M'_{\lceil c/\ell \rceil}$

Fig. 3. CTR-based FSE with OO Capability

The encryption algorithm (FSE.Enc) accepts the private key $\boldsymbol{K_i}$ and a message M_i, and is decomposed into two phases: *(i) offline phase.* computes a pre-ciphertext $\tilde{C}_i$ using PRF evaluations on the incremental counter ctr, where the PRF calls are determined based on the number of input chunks w.r.t PRF block size ℓ. Moreover, the private key is updated for the next time period. The latter ensures forward security of the FSE before transitioning to *online* stage. *(ii) online phase.* computes the ciphertext C_i by XORing each chunk M'_j of message M_i with the corresponding pre-ciphertext chunk. Note that the computational overhead of the online encryption is optimal with merely $\lceil m/\ell \rceil$ bitwise XOR operations of ℓ-bit strings, while the offline computational overhead equals $\lceil m/\ell \rceil$ PRF evaluations and one key update via FSE.Upd. The latter (FSE.Upd) evolves the private key $\boldsymbol{K_1}$ via FPRG and increments the counter ctr by $\lceil m/\ell \rceil$ modulo 2^ℓ.

The decryption algorithm (FSE.Dec) proceeds akin to the encryption algorithm (FSE.Enc) except that it accepts as input the current private key $\boldsymbol{K_i}$ and the ciphertext C_i instead of the plaintext M_i.

4.1.2 Instantiations. We propose different instantiations of CTR-based FSE using different block ciphers. *(1) AES-128.* AES-128 is the most widely standardized block cipher, with a 128-bit block size and strong security guarantees against known cryptanalysis. Using AES-128 in CTR mode allows FSE to benefit from well-established hardware support, particularly the AES-NI instruction set on modern CPUs and specialized cryptographic processors on constrained IoT devices. *(2) Chacha20.* Chacha20 is a stream cipher based on ARX (add–rotate–xor) operations and is designed for efficiency in software and resistance against side-channel attacks. Chacha20 processes data in 512-bit blocks (64 bytes), and requires fewer data-dependent operations, therefore incurs lower instruction overhead compared to AES when hardware acceleration is unavailable.

4.2 Generic FAMAC Construction

Fig. 4 illustrates the algorithmic description of our proposed universal FAMAC construction. It harnesses the OO properties of the universal MAC as described in Def. 2.2 to efficiently compute the MAC authentication tag during online runtime using a single UH evaluation.

4.2.1 Algorithmic Description. The key generation (FAMAC.Kg) accepts the security parameter 1^κ, the epoch size b, and an aggregation flag f_{agg}. We investigate different aggregation methods in FAMAC.Agg: *(1) Hash-based Accumulator*: computes a digest using a cryptographic hash function. It is an immutable aggregation (i.e., $\sigma \leftarrow H(\sigma_1||\sigma_2)$). *(2) Bitwise XOR*: is an efficient aggregation using XOR operations proportional to tag size. *(3) Modular Addition*: suitable for UMACs (e.g., Linear Congruential MAC (LC) [10]) where aggregation is performed via a modular addition (Add_q). This aggregation is additively homomorphic, allowing efficient batch verification. FAMAC.Kg generates two FPRG states and computes initial private keys for UMAC components (i.e., UH and PRF).

The tag generation algorithm (FAMAC.Sign) is split into two stages: *(i) offline*: accepts the private key $\boldsymbol{K_i}$ and computes the PRF component in the underlying UMAC (i.e., $\tilde{\sigma}_i$) and updates the PRF's private key. *(ii) online*: computes the remaining computation to generate the authentication tag σ_i and update the UH-related private key component. Unlike the CTR-based FSE, the FAMAC's key update algorithm (FAMAC.Upd) can be invoked from both offline (steps 1-2 in FAMAC.Upd) and online (steps 3-5) stages of the tag generation (FAMAC.Sign), or independently after the execution of FAMAC.Sign.

The tag verification algorithm (FAMAC.AVer) verifies a batch of received messages and a constant-size aggregate tag. During the offline stage, it computes individual PRF outputs and sequentially aggregates to obtain a constant-size digest $\tilde{\sigma}_{i,i+b-1}$. During the online stage, it computes individual UH outputs and sequentially aggregates into $\bar{\sigma}_{i,i+b-1}$. The aggregate authentication tag is computed by aggregating the final PRF and UH outputs via FAMAC.Agg.

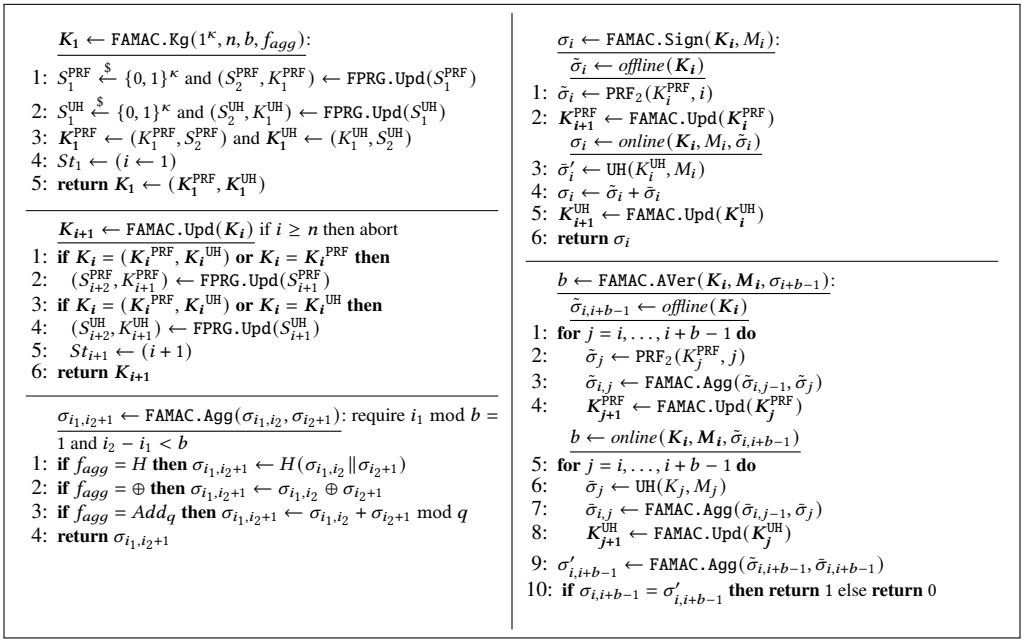


Fig. 4. Universal FAMAC with OO Capability

4.2.2 Instantiations. We propose two main instantiations of the universal FAMAC construction, each differing in the choice of the underlying universal hash function UH, while both $\texttt{PRF}_1$ and $\texttt{PRF}_2$ are instantiated with the AES block cipher. These instantiations are designed to cover a range of efficiency and hardware/software trade-offs while preserving the theoretical security guarantees.

(1) GHASH. This variant uses the widely deployed *GHASH* universal hash function [26] as UH. GHASH computes $\texttt{UH}(K, M)$ by interpreting each 128-bit message block as an element of the finite field $GF(2^{128})$ and evaluating a polynomial under multiplication by a secret hash subkey $K \in GF(2^{128})$. For a message M split into $q = \lceil m/16 \rceil$ blocks $\{M_i'\}$, GHASH computes $\texttt{UH}(K, M) = (((M_1' \cdot K \oplus M_2') \cdot K \oplus \cdots \oplus M_q') \cdot K$ where multiplication is performed modulo the irreducible polynomial $x^{128} + x^7 + x^2 + x + 1$. GHASH is an ε-almost-universal UH with $\varepsilon = 2^{-128}$ for 16-byte blocks. It is highly efficient in software and hardware, relying solely on XOR operations and carry-less multiplications over $GF(2^{128})$. GHASH is the standard UH in GCM mode of operation and is widely used in TLS, IPsec, and IEEE 802.1AE (MACsec).

(2) Poly1305. This variant employs the widely used *Poly1305* universal hash function [3] as UH. Poly1305 computes $\texttt{UH}(K, M) = (\sum_{i=1}^{q} C_i \cdot K^{q-i}) \pmod{2^{130} - 5}$, where K is 128-bit secret key derived from a PRF and $\{C_i \leftarrow f(M)\}_{i=1}^{q=\lceil m/16 \rceil}$ are evaluations of message blocks via a function f [3]. Poly1305 is ε-almost-universal UH with $\varepsilon = 2^{-103}$ for 16-byte blocks. It is highly efficient in software, leveraging 64-bit integer arithmetic and available vector instructions. This instantiation offers provable security and high performance on general-purpose CPUs and 32-bit microcontrollers (e.g., ARM Cortex M4). Poly1305 is widely used in numerous protocols (e.g., TLS 1.3) and present in standard-compliant security frameworks (e.g., OpenSSL, WireGuard).

4.3 Generic Diamond Contruction

Fig. 5 and Fig. 6 illustrate the overview design and algorithmic description of our proposed Diamond framework, respectively. It consists of an Encrypt-then-MAC construction of the CTR-based FSE and the universal FAMAC schemes while harnessing offline batch precomputation.

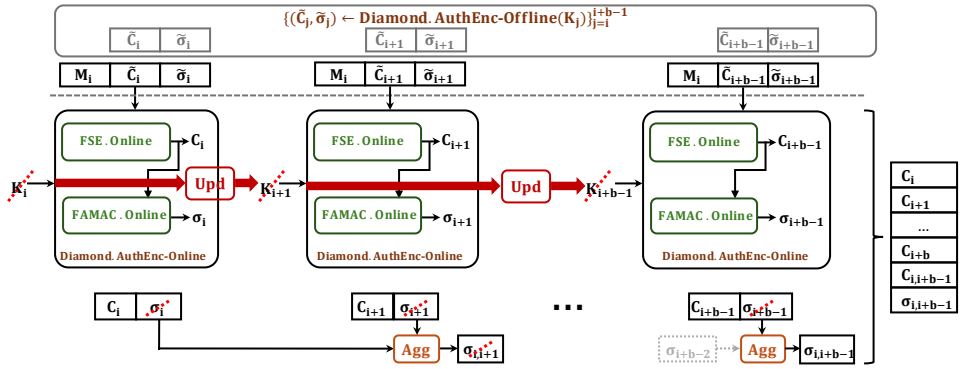


Fig. 5. Overview of Diamond Framework

$K_1 \leftarrow$ Diamond.Kg$(1^\kappa, n, b, f_{agg})$:
1: $K_1^{\text{FSE}} \leftarrow$ FSE.Kg$(1^\kappa, n)$
2: $K_1^{\text{FAMAC}} \leftarrow$ FAMAC.Kg$(1^\kappa, n, b, f_{agg})$
3: $St_i \leftarrow (i \leftarrow 1, ctr \xleftarrow{\$} \{0,1\}^\kappa)$
4: **return** $K_1 \leftarrow (K_1^{\text{FSE}}, K_1^{\text{FAMAC}})$

$K_{i+1} \leftarrow$ Diamond.Upd(K_i) if $i \geq n$ then abort
1: $K_{i+1}{}' \leftarrow$ FSE.Upd$(K_i{}')$
2: $K_{i+1}{}'' \leftarrow$ FAMAC.Upd$(K_i{}'')$
3: $St_i \leftarrow (i \leftarrow i+1, ctr \leftarrow ctr + \lceil m/t \rceil \bmod 2^\kappa)$
4: **return** $K_{i+1} \leftarrow (K_{i+1}{}', K_{i+1}{}'')$

$\sigma_{i_1,i_2+1} \leftarrow$ Diamond.Agg$(\sigma_{i_1,i_2}, \sigma_{i_2+1})$:
1: $\sigma_{i_1,i_2+1} \leftarrow$ FAMAC$(\sigma_{i_1,i_2}, \sigma_{i_2})$
2: **return** σ_{i_1,i_2+1}

$(C_i, \sigma_i) \leftarrow$ Diamond.AuthEnc(K_i, M_i):
$(\tilde{C}_i, \tilde{\sigma}_i) \leftarrow$ *offline* (K_i)
1: $\tilde{C}_i \leftarrow$ FSE.Enc-offline(K_i^{FSE})
2: $\tilde{\sigma}_i \leftarrow$ FAMAC.Sign-offline(K_i^{FAMAC})
$(C_i, \sigma_i) \leftarrow$ *online* $(K_i, M_i, \tilde{C}_i, \tilde{\sigma}_i)$
3: $C_i \leftarrow$ FSE.Enc-online$(M_i, \tilde{C}_i)$
4: $\sigma_i \leftarrow$ FAMAC.Sign-online$(K_i^{\text{FAMAC}}, C_i, \tilde{\sigma}_i)$
5: **return** (C_i, σ_i)

$M_i \leftarrow$ Diamond.AVerDec$(K_i, C_i, \sigma_{i,i+b-1})$:
$(\tilde{M}_i, \tilde{\sigma}_{i,i+b-1}) \leftarrow$ *offline* (K_i)
1: **for** $j = i, \ldots, i+b-1$ **do**
2: $\quad \tilde{M}_j \leftarrow$ FSE.Dec-offline(K_i^{FSE})
3: $\tilde{M}_i \leftarrow (\tilde{M}_i, \ldots, \tilde{M}_{i+b-1})$
4: $\tilde{\sigma}_{i,i+b-1} \leftarrow$ FAMAC.AVer-offline(K_i^{FAMAC})
$M_i \leftarrow$ *online* $(K_i, C_i, \tilde{M}_i, \tilde{\sigma}_{i,i+b-1})$
5: $b \leftarrow$ FAMAC.AVer-online$(K_i^{\text{FAMAC}}, C_i, \tilde{\sigma}_{i,i+b-1})$
6: **if** $b = 0$ **then return** $\perp$
7: **for** $j = i, \ldots, i+b-1$ **do**
8: $\quad M_j \leftarrow$ FSE.Dec-online$(C_j, \tilde{M}_j)$
9: **return** $M_i \leftarrow (M_i, \ldots, M_{i+b-1})$

Fig. 6. Generic Diamond Framework

4.3.1 Algorithmic Description. The key generation algorithm (Diamond.Kg) accepts the security parameter κ, the maximum AuthEnc(.) operations n, the epoch size b, and an aggregation flag f_{Agg}. It initializes independent key states for the forward-secure encryption scheme (FSE) and the forward-secure MAC scheme (FAMAC). An internal variable $St_i = (i, ctr)$ is maintained, where i is the epoch index and ctr is a counter initialized at random. The initial secret key K_1 is returned. At each iteration, the encryption and MAC secret keys are updated using their respective update algorithms.

The counter is increased by the number of consumed blocks $\lceil \frac{m}{\ell} \rceil$, and the previous secret key is securely deleted to ensure forward security.

To encrypt then authenticate a message M_i, `Diamond.AuthEnc` incurs two stages: *(i) Offline phase.* It precomputes $\tilde{C}_i$ and $\tilde{\sigma}_i$ using `FSE` and `FAMAC`. *(ii) Online phase.* It compute ciphertext C_i and partial authentication tag $\bar{\sigma}_i$. Finally, it outputs the ciphertext-tag pair (M_i, σ_i).

To perform verified decryption of a batch of ciphertexts, `Diamond.AVerDec` first performs: *(i) Offline phase.* Precompute decryption masks $\{\tilde{M}_j\}$ and the partial aggregate tag $\tilde{\sigma}_{i,i+b-1}$. *(ii) Online phase.* Verify the aggregate tag via `FAMAC.AVer-online`. If verification fails, output $\perp$. Otherwise, decrypt each ciphertext C_j as $M_j = \texttt{FSE.Dec-online}(C_j, \tilde{M}_j)$ and return $(M_i, \ldots, M_{i+b-1})$.

Upon every `AuthEnc` operation at iteration i, the sender securely deletes the (precomputed) keys.

4.3.2 Instantiations. We present two instantiations of `Diamond` by selecting AES-128 as the underlying $\texttt{PRF}_2$ and deriving the designs from the $\texttt{PRF}_1$ and the universal hash `UH` functions embedded within the corresponding `FSE` and `FAMAC` construction, respectively:

(1) $\texttt{Diamond}_1$. This variant integrates AES-128 as the stream cipher in the `FSE` algorithm and employs GHash as the universal hash function in `FAMAC`. GHash operates over the finite field $\mathbb{F}_{2^{128}}$ and provides 2^{-128}-universality, making $\texttt{Diamond}_1$ a strong, standard-compliant instantiation with the NIST security recommendations. The key update mechanism leverages a forward-secure pseudo-random generator (`FPRG`) instantiated with AES-128, which provide resilience against key compromise. We selected XOR-based aggregation to provide high efficient and minimal energy usage, unlike cryptographic hashing or modular addition.

(2) $\texttt{Diamond}_2$. This configuration integrates ChaCha20 with Poly1305 in the `FSE` and `FAMAC` algorithms, respectively, following the well-established `AE` construction [19]. Poly1305 operates modulo $2^{130} - 5$ and achieves 2^{-103}-universality, offering a strong security-performance balance for embedded and real-time systems. Compared to $\texttt{Diamond}_1$, $\texttt{Diamond}_2$ is optimized for software-centric and ARM-based platforms, which benefit from the ChaCha20 ARX design and Poly1305 high efficiency. Similar to $\texttt{Diamond}_1$, the key-update procedure relies on AES-128-based `FPRG` for forward-secure key evolution, while we also opt for XOR-based aggregation akin to $\texttt{Diamond}_1$.

5 PERFORMANCE EVALUATION

This section presents a detailed performance analysis of our proposed `Diamond` framework and its building blocks. We assess `Diamond` with `FAAE` counterparts from existing AE standards.

5.1 Evaluation Setup

Hardware Configuration. We experimentally evaluate the performance of the generic `Diamond` framework instantiated with its underlying `FSE` and `FAMAC` building blocks across a heterogeneous set of embedded and commodity hardware platforms. This cross-platform evaluation allows us to quantify the efficiency and scalability of `Diamond` under different computational and memory constraints, ranging from resource-constrained 8-bit microcontrollers (MCUs) to high-end general-purpose processors. The experimental platforms are summarized in Table 2:

1. Commodity Hardware (baseline, x86_64). As our reference platform, we use a desktop equipped with an Intel Core i9-9900K @ 3.6 GHz and 64GB of DDR4 memory. We consider optimizations such as AES-NI and AVX2 instruction sets, enabling hardware-accelerated evaluation of the AES-based instantiations of both `FSE` and `FAMAC` building blocks.

2. ARM Cortex A72 (edge-class, 64-bit). To assess `Graphene`'s performance on low-end edge devices, we use a Raspberry Pi 4 Model B featuring a 64-bit quad-core ARM Cortex-A72 SoC @ 1.8 GHz and equipped with 2GB of SDRAM memory. This platform represents a mid-tier embedded processor

(e.g., found in IoT gateways), therefore allowing the scalability assessment of Graphene in moderately constrained environments *without hardware-based cryptographic accelerators*.
3. ARM Cortex-M4 (MCU-class, 32-bit). For the MCU category, we use STM32F439ZI that integrates a 32-bit ARM Cortex-M4 @ 168MHz, with 2MB of flash and 256KB of SRAM. It features a hardware cryptographic accelerator supporting AES-128,192,256, SHA-256, and HMAC primitives. The Cortex-M4 architecture is a canonical representative of resource-constrained IoT devices.
4. AVR ATmega2560 (constrained MCU, 8-bit). For constrained MCUs, we use an Arduino Mega 2560 board based on the 8-bit AVR ATmega2560 @ 16MHz, equipped with 256KB of flash memory and 8KB of SRAM. This platform represents low-end devices without hardware acceleration. Thus, it enables characterizing its minimal footprint and the performance of lightweight instantiations of FSE and FAMAC under high resource constraints.

Table 2. Hardware configurations of our selected platforms.

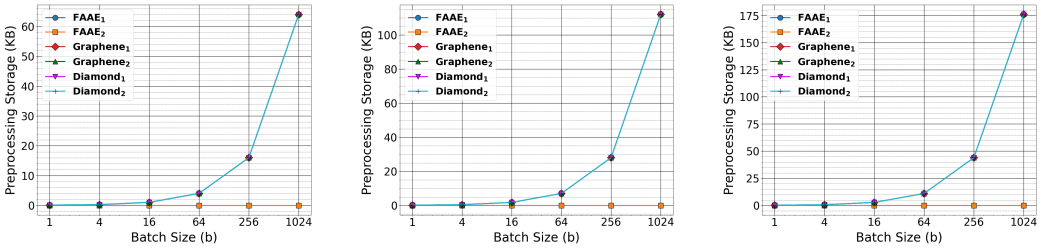
Characteristic	x86_64	Cortex-A72	Cortex-M4	AVR ATmega2560
CPU Architecture	64-bit CISC (Intel Core i9-9900K)	64-bit RISC (ARMv8-A)	32-bit RISC (ARMv7E-M)	8-bit RISC (AVR)
Core Configuration	8 cores / 16 threads	Quad-core	Single-core	Single-core
Frequency	3.6 GHz	1.8 GHz	168 MHz	16 MHz
Memory (SRAM/DRAM)	64 GB DDR4	2 GB SDRAM	256 KB SRAM	8 KB SRAM
Flash Storage	–	microSD / external	2 MB	256 KB
Crypto Acceleration	AES-NI, AVX2	None	AES, SHA-256, HMAC	None
Power Class	High-performance desktop	Edge-class (IoT gateway)	Low-power MCU	Ultra-low-power MCU

Software Configuration. Throughout the different heterogeneous platforms, we carefully select standard-compliant and most efficient libraries and open-source implementations, as follows:
1. On Commodity Hardware. We use OpenSSL[3] to implement cryptographic primitives (e.g., cryptographic hash functions, universal hash functions, and PRF functions).
2. On Edge Class. We use a cross-compiled build of OpenSSL targeting ARM Cortex-A72.
3. On MCU Class. We use wolfSSL[4] for Cortex-M4, given its compliance with cryptographic standards (e.g., FIPS 140-2/3, TLS 1.3) and optimized performance for embedded targets.
4. On 8-bit MCUs. To the best of our knowledge, there exist only a limited number of cryptographic software libraries, each offering a distinct and often incomplete set of implementations for fundamental cryptographic primitives. Consequently, no available library provides a unified, performance-optimized solution in order to implement Diamond variants. For instance, the Arduino Cryptography Library[5] offers the essential primitives required to instantiate Graphene. However, it lacks optimizations (e.g., at the assembly level). Furthermore, its object-oriented design introduces non-negligible computational and memory overheads. Likewise, the AVR-Crypto-Lib[6] was implemented in C and Assembly, and is over a decade old. It also does not support the Poly1305 universal hash function and integrated authenticated encryptions, which are integral for our performance analysis. The μNaCl framework[7] [16], also outdated by more than a decade and provides only Poly1305 as a universal hash and SHA-512 as its sole cryptographic hash function. In parallel, Cardoso et al. [5] proposed a unified authenticated encryption evaluation framework that includes existing implementations from earlier libraries. In our work, we benchmark our deployed cryptographic primitives across the aforementioned libraries and select the most efficient implementation.

[3]https://github.com/openssl/openssl
[4]https://github.com/wolfSSL/wolfssl/tree/master/IDE/STM32Cube
[5]https://github.com/rweather/arduinolibs
[6]https://github.com/cantora/avr-crypto-lib
[7]https://munacl.cryptojedi.org/atmega.shtml

Table 3. Components and parameters in FAAE (Ascon), Graphene, and Diamond schemes.

Scheme	$\mathbf{PRF}_1$ (in FSE)			UH (in FAMAC)			Key Update	Aggregation
	Name	Type	Key/State	Name	Modulus	Universality	Method	Operator
$\mathbf{FAAE}_1$	AES-128-GCM				(integrated)		HASH(SHA-256)	XOR
$\mathbf{FAAE}_2$	Ascon128a				(integrated)		HASH(AsconHash256)	XOR
$\mathbf{Graphene}_1$	AES-128	SPN	128 / 128	GHASH	2^{128}	2^{-128}	HASH(SHA-256)	XOR
$\mathbf{Graphene}_2$	ChaCha20	ARX	256 / 512	Poly1305	$2^{130}-5$	2^{-103}	HASH(SHA-256)	XOR
$\mathbf{Diamond}_1$	AES-128	SPN	128 / 128	GHASH	2^{128}	2^{-128}	FPRG(AES-128)	XOR
$\mathbf{Diamond}_2$	ChaCha20	ARX	256 / 512	Poly1305	$2^{130}-5$	2^{-103}	FPRG(AES-128)	XOR



(a) $m = 16$ Bytes (b) $m = 64$ Bytes (c) $m = 128$ Bytes

Fig. 7. Offline preprocessing storage overhead of Diamond variants under different payload and batch sizes

Counterpart Selection. To provide a fair performance evaluation, we evaluate Diamond instantiations against initial Graphene variants: i.e., Graphene-GCM and Graphene-Poly, which we now refer to as Graphene_1 and Graphene_2, respectively. We also included two FAAE instantiations, FAAE_1 and FAAE_2, which incorporate the FIPS standard AES-128-GCM and the NIST lightweight standard Ascon128a [37] as an integrated AEs, and SHA-256 and Ascon-hash-256 for key update, respectively. We selected the bitwise XOR operation as an aggregation method for all schemes. The cryptographic components and parameters of Diamond and its counterparts are depicted in Table 3.

Evaluation Metrics. Our evaluation addresses three key research questions (RQs) to fairly evaluate Diamond and its selected FAAE and Graphene counterparts:

RQ.1 *Preprocessing Overhead and Overall Energy Drawings.* What is the cost of offline preprocessing, and what is its impact on the overall energy usage, including online and transmission costs?

RQ.2 *Online Latency.* How much can OO optimization reduce encryption throughput and E2E verification delay on constrained IoT devices and nearby IoT servers, respectively?

RQ.3 *Fairness & Baseline.* How does Diamond compare fairly to Graphene and FAAE instantiations based on prior constructions and the NIST lightweight standard Ascon?

5.2 Performance Analysis

By combining the existing instantiations of FSE and FAMAC with the corresponding aggregation functions, we derive multiple Diamond variants, each offering a distinct performance advantage on at least one evaluation metric. Subsequently, we perform a comprehensive evaluation of these Diamond variants on diverse hardware platforms to identify the most suitable configuration.

Storage Overhead. At first, we analyze the storage overhead incurred during the *offline* stage of Diamond.AuthEnc execution, under different message sizes (i.e., m) and batch sizes (i.e., b). Fig. 7 illustrates the offline preprocessing storage footprint of the proposed Diamond variants and their baseline counterparts for varying batch sizes (2^0 to 2^{10}). Specifically, Diamond precomputes the pre-ciphertexts $\tilde{C}_i$ in the FSE instantiation and the PRF derived tags $\tilde{\sigma}_i$ under the FAMAC construction.

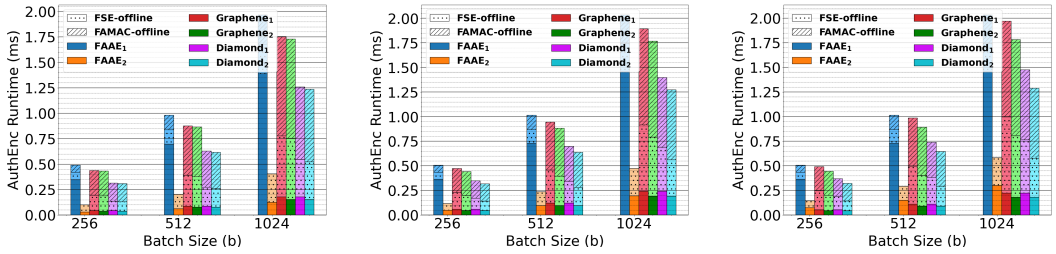


Fig. 8. Runtime of Diamond variants under different payload and batch sizes on x86_64 commodity platform

Recall that Graphene and Diamond incur identical storage overheads, as their distinguishing feature lies in the key evolution strategy (i.e., cryptographic hash update vs. PRF-based FPRG). For a batch size of 2^{10}, the total offline storage overhead reaches 60 KB and 175 KB when processing 16-byte and 128-byte message payloads, respectively. This footprint is well-supported by our selected platforms. It constitutes less than 12% of on-chip SRAM on the ARM Cortex-M4 and is negligible on the edge-class ARM Cortex-A72 systems. FAAE counterparts exhibit a minimal memory footprint, where they only store the FAAE secret key, which equals 32 and 16 bytes, for $\texttt{FAAE}_1$ and $\texttt{FAAE}_2$.

x86_64. Fig. 8 illustrates the runtime of the cumulative authenticated encryption (AuthEnc) overhead for our benchmarked schemes evaluated under varying payload lengths and batch sizes. The runtime encapsulate both the online processing costs and the *offline* preprocessing for the FSE and FAMAC algorithms. Given that the experimental platform is a resourceful multi-core x86_64 commodity hardware, the *online* execution footprints of the selected FAAE, Graphene, and Diamond constructions are comparable for moderate message sizes except for the standard $\texttt{FAAE}_1$. Moreover, the online evaluation latency across all variants of Graphene and Diamond is indistinguishable under the selected parameters, as the core comparison is reduced to the evaluation of the key update mechanism. Specifically, the $\texttt{Diamond}_1$ and $\texttt{Diamond}_2$ variants, optimized via PRF-based key update, demonstrate 1.43× and 1.46× speedups, respectively, over their initial $\texttt{Graphene}_1$ and $\texttt{Graphene}_2$ counterparts when instantiated over 128-byte payloads. This corresponds to achieving roughly a 30% reduction in cycle count. These savings translate directly into reduced energy usage and improved verification efficiency when payload lengths or batch sizes are further increased, making Diamond the best candidate for performance-critical deployments. Compared to their $\texttt{FAAE}_1$ counterpart, Diamond instantiations deliver substantial performance gains, achieving up to 9× reduction in online AuthEnc latency. Compared to NIST lightweight $\texttt{FAAE}_2$, $\texttt{Diamond}_2$ exhibits a nuanced cost profile by incurring moderately higher overhead on short 16-byte payloads (i.e., 1.22× slower) while surpassing on larger 128-byte telemetry by delivering 1.66× faster online AuthEnc throughput.

Cortex ARM A72. Fig. 9 illustrates the runtime of authenticated encryption operations on the edge device, which does not possess specialized hardware acceleration, unlike x86_64. Our experimental results show the high inefficiency of standard $\texttt{FAAE}_1$ where special AVX instructions are not available, while $\texttt{FAAE}_2$ (employing the NIST lightweight AE, Ascon128) exhibits slightly better efficiency compared to Graphene and Diamond instantiations for small 16-byte payloads. For example, the overall AuthEnc runtime of $\texttt{FAAE}_2$ is 4.67× faster compared to the highly efficient Diamond variants. However, for large (64-128 bytes) messages, the online AuthEnc runtime of Diamond outperforms that of $\texttt{FAAE}_2$ by being 1.45× faster. Comparing the instantiations of Diamond against Graphene, the $\texttt{Diamond}_1$ and $\texttt{Diamond}_2$ variants achieve 38.46% and 40.17% reduction in offline preprocessing

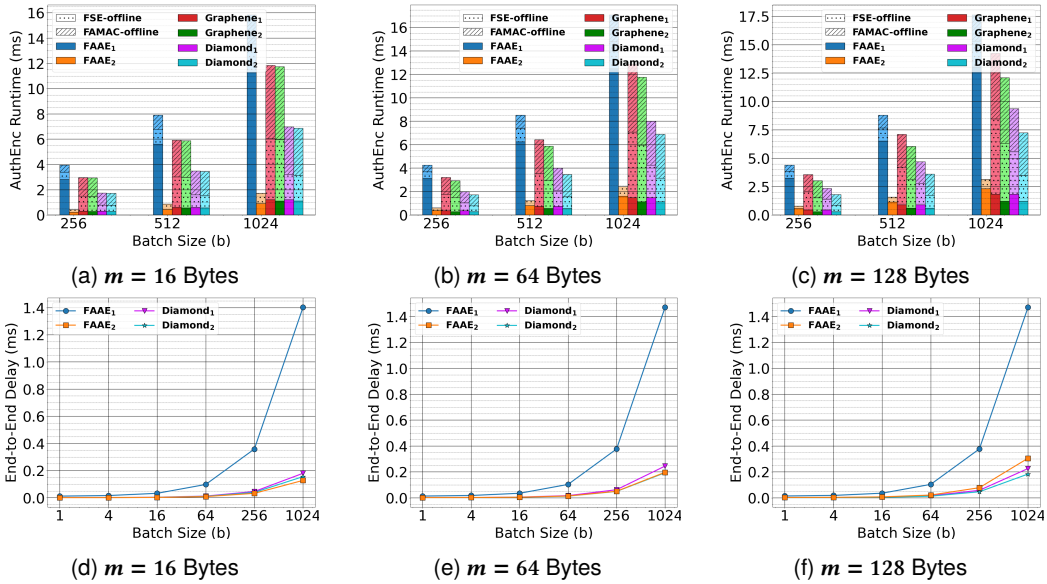


Fig. 9. AuthEnc Runtime overhead and End-to-End (E2D) delay of Diamond variants under different input and batch sizes on Cortex ARM-A72 platform. E2E delay assumes a resourceful verifier (i.e., x86_64 commodity).

runtime compared to that of the original $\texttt{Graphene}_1$ and $\texttt{Graphene}_2$ instantiations, respectively, for a payload size of $m = 64$ bytes. This is $\approx 8\%$ more computational savings compared to x86_64.

To enable a holistic comparison of our selected performance schemes, we further plot the end-to-end (E2E) delay (see Fig. 9d-9f) between the IoT device and resourceful verifier (e.g., edge device). As per Fig. 2, the E2E verification delay is defined by a single AuthEnc operation of the last message in the current batch at the IoT device and an online AverDec of the received authenticated batch of ciphertexts at the IoT server. We consider an x86_64 commodity hardware to perform the verified decryption (AverDec) of received batches of encrypted payloads and their aggregate authentication tag. We omit the network delay incurred during the communication of cryptographic payloads, given that the transmission overhead of all benchmarked schemes is equal. Fig. 9d-9f illustrates the advantage of $\texttt{Diamond}_2$ (i.e., instantiated with Chacha20 and Poly1305) for large input sizes (64-128 bytes) by being 1.5× faster E2E delay than $\texttt{FAAE}_2$, while $\texttt{FAAE}_2$ remains advantageous (i.e., 1.23× faster than $\texttt{Diamond}_2$) on small 16-byte payloads by having the lowest E2E delay.

Cortex ARM M4. Fig. 10 illustrates the authenticated encryption runtime and end-to-end delay on a 32-bit Cortex-M4 MCU. On this platform, we enable optimized execution of cryptographic primitives (i.e., AES-128 and SHA-256) via the hardware acceleration peripheral. When comparing Diamond against Graphene instantiations, the AuthEnc overhead of Diamond significantly reduces the offline FSE and FAMAC offline preprocessing cost by being 3.16× faster thanks to the high efficiency of AES-based PRF key updates compared to the original cryptographic hash operations. Compared to FAAE variants, the overall AuthEnc of Diamond variants outperforms $\texttt{FAAE}_1$ by a factor of $\approx 2\times$ while being comparable to $\texttt{FAAE}_2$ based on the NIST lightweight Ascon. The online AuthEnc runtime of Diamond variants is up to 3.5× and 1.68× faster than that of $\texttt{FAAE}_1$ and $\texttt{FAAE}_2$, respectively. This underscores the high efficiency of online Diamond variants on ARM Cortex-M4 controllers when harnessing their OO properties, compared to the most efficient FAAE counterparts.

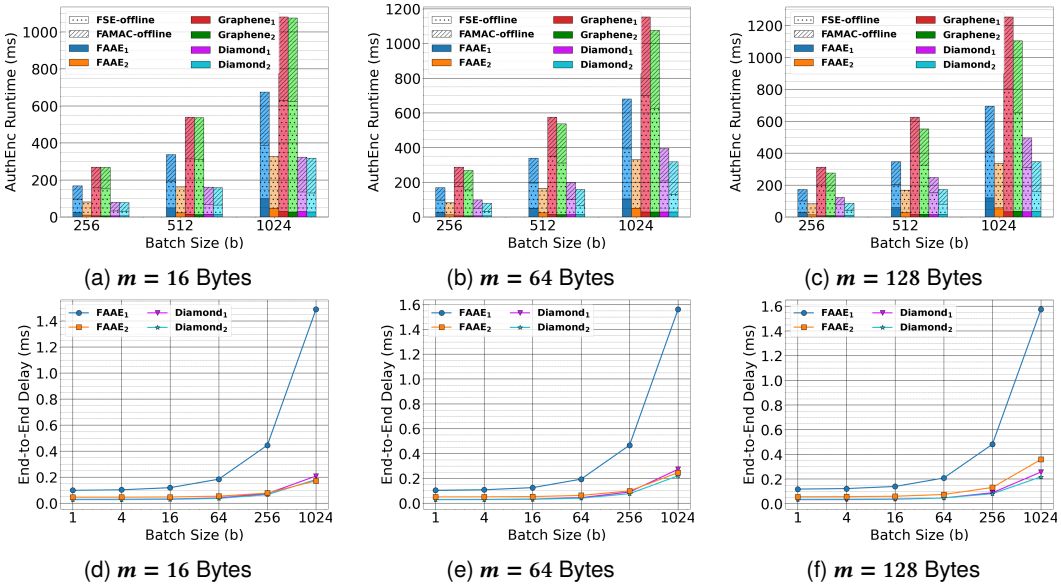


Fig. 10. AuthEnc Runtime overhead and End-to-End (E2D) delay of Diamond variants under different input and batch sizes on Cortex ARM-M4 platform. E2E delay assumes a resourceful verifier (i.e., x86_64 commodity).

Fig. 10d-10f depicts the end-to-end delay when considering an IoT device equipped with a 32-bit ARM Cortex-M4 MCU and a resourceful server with x86_64 commodity hardware. On 16-byte inputs, Fig. 10d demonstrates that $\texttt{Diamond}_2$ and $\texttt{FAAE}_2$ achieve relatively equal E2E delay while being 1.2× and 8.19× faster than $\texttt{Diamond}_1$ and $\texttt{FAAE}_1$, respectively. On larger 128-byte payloads, $\texttt{Diamond}_2$ exhibits the highest efficiency by being 7.2× and 1.64× faster than $\texttt{FAAE}_1$ and $\texttt{FAAE}_2$, respectively, while also being 1.18× faster than the AES-GCM-based instantiation $\texttt{Diamond}_1$.

AVR ATmega2560. Fig. 11 illustrates the AuthEnc operations and E2E delays given the ultra-low-power 8-bit AVR MCU in the order of seconds. Our experimental results showcase the high inefficiency of $\texttt{Graphene}_1$ and $\texttt{Diamond}_1$ which stems from the high cycle count of the employed universal hash function GHASH. In contrast, the overall AuthEnc runtime of $\texttt{Graphene}_2$ and its improved $\texttt{Diamond}_2$ is lesser compared to our selected FAAE counterparts. For example, on 16-byte input sizes, the AuthEnc of $\texttt{Diamond}_2$ is 1.63× and 2.29× compared to $\texttt{FAAE}_1$ and $\texttt{FAAE}_2$, respectively. Moreover, $\texttt{Diamond}_2$ achieves 47.25% reduction in total AuthEnc computational overhead compared to $\texttt{Graphene}_2$ while also being 1.39× faster than $\texttt{Diamond}_1$. On larger 128-byte payloads, $\texttt{Diamond}_2$ is still exhibiting high efficiency in terms of overall AuthEnc runtime with 3.49× faster than $\texttt{Diamond}_1$, and ≈ 2× faster than FAAE counterparts. The efficiency of the online AuthEnc overhead of $\texttt{Diamond}_2$ is more pronounced with 3.59× and 2.82× faster than $\texttt{FAAE}_1$ and $\texttt{FAAE}_2$ counterparts. This showcases that on highly constrained devices, $\texttt{Diamond}_2$ outperforms even $\texttt{FAAE}_2$ from the NIST lightweight standard, Ascon, on both online and total AuthEnc operations.

Across Fig. 11g–11i, the end-to-end (E2E) latency exhibits near-constant behavior. This stems from the fact that the per-item online AuthEnc cost on the IoT device dominates the amortized online verified-decryption cost (AverDec) over a large aggregated batch on the IoT server side. Across all evaluated payload sizes (16–128 bytes), $\texttt{Diamond}_2$ consistently outperforms both FAAE variants and $\texttt{Diamond}_1$. Notably, $\texttt{Diamond}_2$ achieves 3.79× and 2.7× lower E2E delay than the Ascon-based

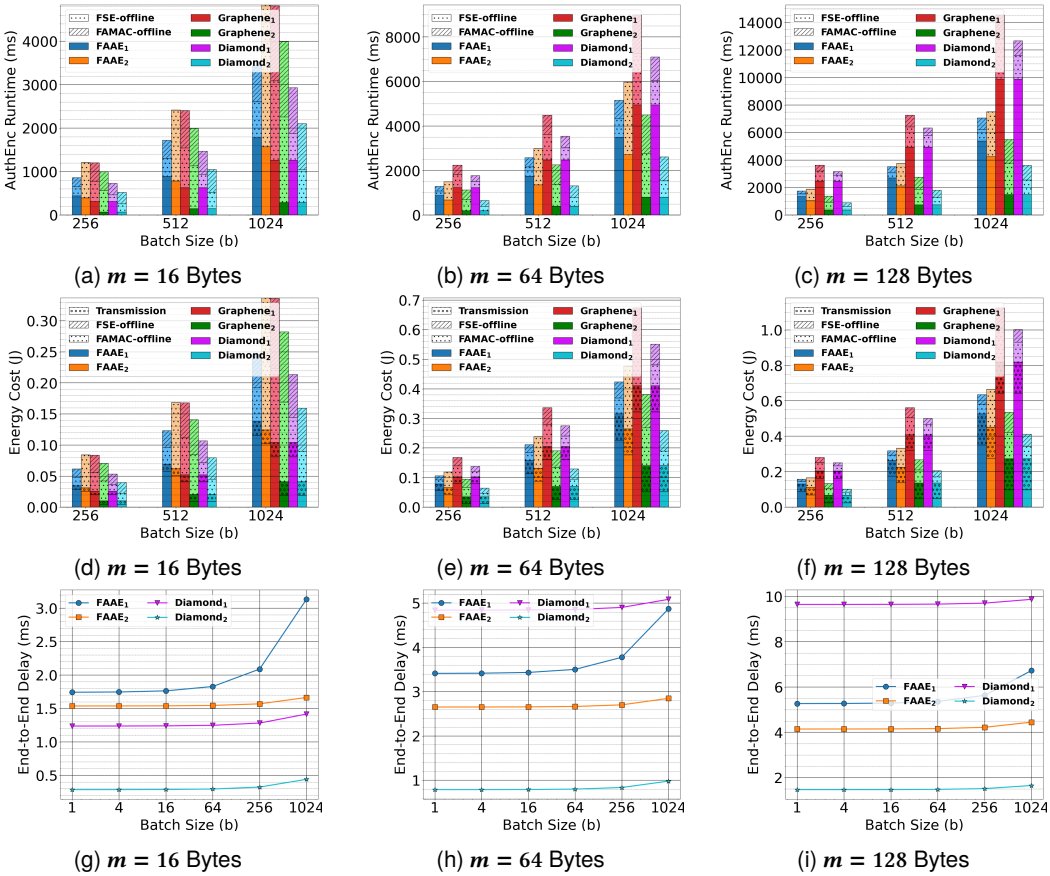


Fig. 11. AuthEnc Runtime overhead, energy cost of AuthEnc and data transmission, and End-to-End (E2E) delay of Diamond variants under different input and batch sizes on AVR ATmega2560 platform. E2E delay assumes a resourceful verifier (i.e., x86_64).

$\texttt{FAAE}_2$—the most optimized competing design—on 16-byte and 128-byte messages, respectively. These results confirm that $\texttt{Diamond}_2$, instantiated from ChaCha20 as the PRF for both symmetric encryption and forward-secure key evolution and Poly1305 as the universal hash in FAMAC, constitutes the most suitable instantiation on resource-constrained 8-bit AVR microcontrollers.

Energy Analysis on 8-bit AVR ATmega2560. To assess the impact of Diamond on the energy usage on constrained devices, we estimated the total energy consumption per AuthEnc operation using the MICAz energy model [30]. The MICAz node features an ATmega128L MCU operating at 16MHz, with 128KB flash and 4KB SRAM (i.e., similar computing capabilities to ATmega2560). Following the energy parameters in [30], the ATmega128L consumes approximately $E_{CPU} = 4.07$nJ per clock cycle, while the CC2420 ZigBee transceiver incurs $E_{TX} = 0.168$ μJ per transmitted bit.

Fig. 11d-11f depicts the energy cost of AuthEnc operation and transmission for batches of payloads and their authentication tags with different input sizes (16-128 bytes), with a breakdown illustrating the contribution of online AuthEnc, offline FSE and FAMAC computations, and the transmission overheads. For small telemetry payloads, the transmission cost dominates, often matching or exceeding the online AuthEnc energy of the most efficient schemes. $\texttt{Diamond}_2$ exhibits the lowest

online AuthEnc energy among all schemes, being approximately 36% lower than $\texttt{Diamond}_1$, 54–67% lower than Graphene variants, and up to 4× lower than FAAE variants. For larger 128-byte inputs, $\texttt{Diamond}_2$ requires 50–65% less total energy compared to $\texttt{Graphene}_1$ and $\texttt{FAAE}_1$ counterparts, and 35–45% less energy compared to $\texttt{Graphene}_2$. Across all input lengths, $\texttt{Diamond}_2$ consistently yields the lowest total energy, with reductions ranging from 50-75% compared to FAAE and 35–60% compared to Graphene variants. The results confirm that $\texttt{Diamond}_2$ is the most energy-optimal scheme, especially for compute- and energy-restricted IoT devices such as wearable devices, medical pacemakers, and resource-constrained sensor nodes.

6 SECURITY ANALYSIS

In this section, we present the security models of cryptographic primitives, used to construct our proposed scheme, Diamond. Then, we present security analysis of FSE, FAMAC, and Diamond.

6.1 Security Models

PRF. The security of PRF is defined by the following adversary's advantage:

$$\texttt{Adv}_{\texttt{PRF}}(t,q) = \max_{\mathcal{A}} |\Pr[f \xleftarrow{\$} \mathcal{F} : \mathcal{A}^{f(.)} = 1] - \Pr[K \xleftarrow{\$} \mathcal{K}, g \leftarrow \texttt{PRF}_K : \mathcal{A}^{g(.)} = 1]|,$$

where the maximum is over all adversaries $\mathcal{A}$ making at most q queries and running in time t. PRF is secure iff $\texttt{Adv}_{\texttt{PRF}}(t,q)$ is negligible.

PRG. The security of PRG is defined by the following adversary's advantage:

$$\texttt{Adv}_{\texttt{PRG}}(t,q=1) = \max_{\mathcal{A}} |\Pr[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}(\texttt{PRG}(K)) = 1] - \Pr[(K',Y) \xleftarrow{\$} \mathcal{K}\times\mathcal{T} : \mathcal{A}(Y) = 1]|,$$

where the maximum is over all adversaries $\mathcal{A}$ making at most one query and running in time t.

FPRG. Based on the seminal work of Bellare et al. [2], the security of FPRG is defined as follows:

$$\texttt{Adv}_{\texttt{FPRG}}(t_{\texttt{FPRG}}) \leq 2 \cdot n \cdot \texttt{Adv}_{\texttt{PRF}}(t_{\texttt{PRF}}, q_{\texttt{PRF}}),$$

where $q_{\texttt{PRF}} = \lceil \frac{(\tau'+\kappa)}{\tau} \rceil$ and $t_{\texttt{PRF}} = t_{\texttt{FPRG}} + O(n \cdot (\kappa + \tau'))$.

MAC. The standard security notion of MAC is Existential Unforgeability under Chosen Message Attacks (EU-CMA) [2], which is defined based on the following experiment:

Experiment $Expt^{\texttt{EU-CMA}}_{\texttt{MAC}}(\mathcal{A})$
$K \leftarrow \texttt{MAC.Kg}(1^\kappa)$
$(M^*, \sigma^*) \xleftarrow{\$} \mathcal{A}^{\texttt{MAC.Sign}(K,.)}(find)$
if M^* was not queried to $\texttt{MAC.Sign}(K,.)$ **and** $\texttt{MAC.Ver}(K, M^*, \sigma^*) = 1$ **then return** 1 **else** 0.

In $\text{Expt}^{\texttt{EU-CMA}}_{\texttt{MAC}}$, $\mathcal{A}$ is given access to the signing oracle $\texttt{MAC.Sign}(K,.)$ and aims to produce a forgery (M^*, σ^*). The advantage of $\mathcal{A}$ over MAC is $\texttt{Adv}^{\texttt{EU-CMA}}_{\texttt{MAC}}(t,q) = \max_{\mathcal{A}} \Pr[\text{Expt}^{\texttt{EU-CMA}}_{\texttt{MAC}}(\mathcal{A}) = 1]$, where the maximum is over all adversaries $\mathcal{A}$ making at most q queries and running in time t.

FAMAC. The standard security notion of FAMAC scheme is Forward-secure Aggregate EU-CMA (FA-EU-CMA), which is defined based on the following experiment:

Experiment $Expt^{\texttt{FA-EU-CMA}}_{\texttt{FAMAC}}(\mathcal{A})$
$K_1 \leftarrow \texttt{FAMAC.Kg}(n)$, $i \leftarrow 1$, $phase \leftarrow find$
while $i \leq n$ **and** $phase \neq forge$
$phase \leftarrow \mathcal{A}^{\texttt{FAMAC.Sign}(K_i,.)}(find)$
$K_{i+1} \leftarrow \texttt{FAMAC.Upd}(K_i)$ **and** $i \leftarrow i+1$
$(\boldsymbol{M}_{1,i^*}, \sigma_{1,i^*}) \leftarrow \mathcal{A}(forge, K_i)$
if $1 \leq i^* < i$ **and** $\boldsymbol{M}_{1,i^*}$ is not queried to $\texttt{FAMAC.Sign}(K_i,.)$ oracle **and** $\texttt{FAMAC.AVer}(K_1, \boldsymbol{M}_{1,i^*}, \sigma_{1,i^*}) = 1$ **then return** 0 **else return** 1.

In $\text{Expt}^{\texttt{FA-EU-CMA}}_{\texttt{FAMAC}}$, $\mathcal{A}$ operates in two stages: *(1) find.* $\mathcal{A}$ makes at most n queries to the signing oracle $\texttt{FAMAC.Sign}(K_i, .)$. *(2) forge.* $\mathcal{A}$ is given the current private key K_i during the break-in and asked to produce a non-trivial forgery $(\boldsymbol{M}_{1,i^*}, \sigma_{1,i^*})$ with index $i^* < i$. The advantage of $\mathcal{A}$ over FAMAC is $\text{Adv}^{\texttt{FA-EU-CMA}}_{\texttt{FAMAC}}(t, q) = \max_{\mathcal{A}} \Pr[\text{Expt}^{\texttt{FA-EU-CMA}}_{\texttt{FAMAC}}(\mathcal{A}) = 1]$, where the maximum is over all adversaries $\mathcal{A}$ making at most q queries and running in time t.

SE. The standard security notion of an SE scheme is INDistinguishability under Chosen Ciphertext Attacks (IND-CPA) [2] which is defined based on the following experiment:

Experiment $\text{Expt}^{\textit{IND-CPA}}_{\textit{Enc}}(\mathcal{A})$

$K \leftarrow \texttt{SE.Kg}(1^\kappa)$ and $b \xleftarrow{\$} \{0, 1\}$
$C_b \leftarrow \mathcal{A}^{\texttt{SE.Enc-LR}(K,.)}(find, M_0, M_1)$
$b' \leftarrow \mathcal{A}^{\texttt{SE.Enc-LR}(K,.)}(guess)$
If $|M_0| = |M_1|$ and $b = b'$ **then return** 1 **else return** 0.

In $\text{Expt}^{\texttt{IND-CPA}}_{\texttt{SE}}$, the adversary $\mathcal{A}$ operates in two phases: *(1) Find.* $\mathcal{A}$ is given access to the encryption left or right oracle $\texttt{SE.Enc-LR}(K, .)$ which accepts two distinct plaintexts (M_0, M_1), and computes/returns to $\mathcal{A}$ the ciphertext C_b based on a pre-defined bit b, generated during the setup phase. The challenger in the game computes the ciphertext C_b with index equal to a bit b, uniformly generated at random. *(2) Guess.* $\mathcal{A}$ returns a bit b'. $\mathcal{A}$ succeeds if $b = b'$. The advantage of $\mathcal{A}$ over SE is $\text{Adv}^{\texttt{IND-CPA}}_{\texttt{SE}}(t, q) = \max_{\mathcal{A}}\{|\Pr[\text{Expt}^{\texttt{IND-CPA}}_{\texttt{SE}}(\mathcal{A}) = 1] - \frac{1}{2}|\}$, where the maximum is over all adversaries $\mathcal{A}$ making at most q queries and running in time t.

FSE. The standard security notion of an FSE scheme is Forward-secure IND-CPA (F-IND-CPA), which is defined based on the following experiment:

Experiment $\text{Expt}^{\textit{F-IND-CPA}}_{\textit{FSE}}(\mathcal{A})$

$K_1 \leftarrow \texttt{FSE.Kg}(1^\kappa, n)$, $b \xleftarrow{\$} \{0, 1\}$, and $i \leftarrow 1$, $phase \leftarrow find$
while $i \leq n$ **and** $phase \neq guess$
  $(phase, C_b) \leftarrow \mathcal{A}^{\texttt{FSE.Enc-LR}(K_i,.)}(find, M_0, M_1)$
  $K_{i+1} \leftarrow \texttt{FSE.Upd}(K_i)$ **and** $i \leftarrow i + 1$
$b' \leftarrow \mathcal{A}^{\texttt{FSE.Enc-LR}(K_i,.)}(guess, K_i)$
if $|M_0| = |M_1|$ and $b = b'$ **then return** 1 **else return** 0.

The adversary $\mathcal{A}$ in this experiment operates in two phases similar to IND-CPA, whereas during the find phase, the challenger updates the private key K_i after each query made by $\mathcal{A}$ to $\texttt{FSE.Enc-LR}(K_i, .)$. During the guess phase, $\mathcal{A}$ is given access to the current secret key and asked to return a bit b'. The advantage of $\mathcal{A}$ over FSE is $\text{Adv}^{\texttt{F-IND-CPA}}_{\texttt{FSE}}(t, q) = \max_{\mathcal{A}}\{|\Pr[\text{Expt}^{\texttt{F-IND-CPA}}_{\texttt{FSE}}(\mathcal{A}) = 1] - \frac{1}{2}|\}$, where the maximum is over all $\mathcal{A}$s making at most q queries and running time in t.

FAAE. The security model of an FAAE is based on the security of its underlying (authenticated) encryption and MAC components. We consider forward security, which implies periodic key updates, an essential feature in adversarial IoT environments against key compromise attacks. For confidentiality, Diamond adopts F-IND-CPA. For integrity protection, we follow the security of FAAE [24, 48], with forward-secure and aggregate existential unforgeability against chosen message attacks (FA-EU-CMA). These notions collectively ensure that even an adaptive PPT adversary $\mathcal{A}$ can neither decrypt previously and currently encrypted telemetry traffic nor forge valid aggregate and forward-secure tags without detection. The F-IND-CPA and FA-EU-CMA security notions for FAAE are formalized by the following experiments:

Experiment $Expt^{F\text{-}IND\text{-}CPA}_{FAAE}(\mathcal{A})$

$K_1 \leftarrow \texttt{FAAE.Kg}(1^\kappa, n, b)$ and $c \xleftarrow{\$} \{0,1\}$
$i \leftarrow 1$ **and** $phase \leftarrow find$
while $i \le n$ **and** $phase \neq guess$
$(phase, C_c) \leftarrow \mathcal{A}^{\texttt{AuthEnc-LR}(K_i,.)}(phase, M_0, M_1)$
$K_{i+1} \leftarrow \texttt{FAAE.Upd}(K_i)$ **and** $i \leftarrow i+1$
$c' \leftarrow \mathcal{A}^{\texttt{AuthEnc-LR}(K_i,.)}(phase, K_i)$
if $|M_0| = |M_1|$ and $c = c'$ **then return** 1 **else return** 0.

Experiment $Expt^{FA\text{-}EU\text{-}CMA}_{FAAE}(\mathcal{A})$

$K_1 \leftarrow \texttt{FAAE.Kg}(1^\kappa, n, b)$
$i \leftarrow 1$ **and** $phase \leftarrow find$
while $i \le n-b+1$ **and** $phase \neq forge$
$phase \leftarrow \mathcal{A}^{\texttt{FAAE.AuthEnc}(K_i,.)}(phase)$
$K_{i+1} \leftarrow \texttt{FAAE.Upd}(K_i)$ **and** $i \leftarrow i+1$
$(C_{i^*}, \sigma_{i^*,i^*+b-1}) \leftarrow \mathcal{A}(phase, K_i)$
if $1 \le i^* < i$ and $(C_{i^*}, \sigma_{i^*,i^*+b-1}) \notin \mathcal{L} \leftarrow \{(C_{i_1}, \sigma_{i_1,i_1+b-1}) | \forall 1 \le i_1 \le i^*-b+1\}$ **and** $M_{i_1} = \texttt{FAMAC.AVer}(K_{i_1}, C_{i^*}, \sigma_{i^*,i^*+b-1})$ **then return** 1 **else return** 0.

The experiment $\text{Expt}^{\texttt{F-IND-CPA}}_{\texttt{FAAE}}$ is similar to $\text{Expt}^{\texttt{F-IND-CPA}}_{\texttt{Enc}}$. The adversary $\mathcal{A}$ in $\text{Expt}^{\texttt{FA-EU-CMA}}_{\texttt{FAAE}}$ outputs, as forgery, a batch of ciphertexts C_{i^*} and an aggregate tag σ_{i^*,i^*+b-1}. $\mathcal{A}$ succeeds if the produced forgery is not a combination of previous outputs of the $\texttt{FAAE.AuthEnc}(K_i, .)$ oracle during the find phase. The advantage of $\mathcal{A}$ over FAAE under F-IND-CPA and FA-EU-CMA security notions is $\texttt{Adv}^{\texttt{F-IND-CPA}}_{\texttt{FAAE}}(t,q) = \max_{\mathcal{A}} \{|\Pr[\text{Expt}^{\texttt{F-IND-CPA}}_{\texttt{FAAE}} = 1] - 1/2|\}$ and $\texttt{Adv}^{\texttt{FA-EU-CMA}}_{\texttt{FAAE}}(t,q) = \max_{\mathcal{A}} \Pr[\text{Expt}^{\texttt{F-IND-CPA}}_{\texttt{FAAE}} = 1]$, respectively, making at most q queries and in running time t.

6.2 Security Proof

Let FSE be the CTR-based forward-secure symmetric encryption scheme defined in Fig. 3, where $\texttt{PRF}_1$ is used in CTR mode to generate the keystream blocks and $\texttt{PRF}_2$ is used to instantiate the forward-secure pseudo-random generator FPRG.Upd to update the states and keys. We prove that if both $\texttt{PRF}_1$ and $\texttt{PRF}_2$ are secure pseudo-random functions, then FSE is F-IND-CPA-secure.

Theorem 1. *If FSE is constructed as above, then for every adversary $\mathcal{A}$ running in time t_{FSE} and making at most q_{FSE} encryption queries, there exist distinguishers $\mathcal{B}_1$ and $\mathcal{B}_2$ such that:*

$$Adv^{F\text{-}IND\text{-}CPA}_{FSE}(t_{FSE}, q_{FSE}) \le n \cdot Adv_{PRF_1}(t_{PRF_1}, q_{PRF_1}) + 2 \cdot n \cdot Adv_{PRF_2}(t_{PRF_2}, q_{PRF_2}),$$

where

$$q_{PRF_1} = q_{FSE} \cdot \left\lceil \frac{m}{\ell} \right\rceil, \quad t_{PRF_1} = t_{FSE} + O\left(q_{FSE} \cdot \left\lceil \frac{m}{\ell} \right\rceil\right),$$

$$q_{PRF_2} \le 2 \cdot n, \qquad t_{PRF_2} = t_{FSE} + O(n + \kappa).$$

Proof. We define a series of hybrid games $H_0 \Rightarrow H^{(1)}_1 \Rightarrow \cdots \Rightarrow H^{(1)}_n \Rightarrow H^{(2)}_1 \Rightarrow \cdots \Rightarrow H^{(2)}_n$

Game $\mathbf{H_0}$ *(Real experiment).* This is the real F-IND-CPA experiment for FSE: both $\texttt{PRF}_1$ and $\texttt{PRF}_2$ are instantiated as true PRF functions. Let $\texttt{Adv}_0 = \left|\Pr[H_0(\mathcal{A}) = 1] - \frac{1}{2}\right|$ denote $\mathcal{A}$'s advantage.

Game $\mathbf{H^{(1)}_i}$ *(Replacing PRF_1 in period i).* For $i \in \{1, \ldots, n\}$, we define $H^{(1)}_i$ to be identical to $H^{(1)}_{i-1}$ except that in period i all values $\tilde{C}_j \leftarrow \texttt{PRF}_1(K_i, ctr + j)$ used to generate CTR keystream blocks are replaced by independent uniform random ℓ-bit strings (i.e., $\tilde{C}_j \xleftarrow{\$} \{0,1\}^\ell$). All other computations remain unchanged. Let $\Delta^{(1)}_i = \left|\Pr[H^{(1)}_{i-1}(\mathcal{A}) = 1] - \Pr[H^{(1)}_i(\mathcal{A}) = 1]\right|$.

The distinguisher $\mathcal{B}_1$ is given oracle access to a function $O(\cdot)$ which is either $\texttt{PRF}_1(K^\star, \cdot)$ for $K^\star \xleftarrow{\$} \{0,1\}^\kappa$ or a true random function. It simulates the F-IND-CPA experiment for $\mathcal{A}$ as follows:

1. For the target period i, $\mathcal{B}_1$ computes the keystream blocks $\{\tilde{C}'_j \leftarrow O(ctr + j \bmod 2^\kappa)\}_{j=1,\ldots,\lceil m/\ell \rceil}$ and returns $C_i \leftarrow \tilde{C}_i \oplus M_i$ to $\mathcal{A}$ where $\tilde{C}_i \leftarrow \tilde{C}'_1 \| \ldots \| \tilde{C}'_{\lceil m/\ell \rceil}$.

2. For all other periods $j \neq i$, $\mathcal{B}_1$ samples uniformly random keys $K_j \xleftarrow{\$} \{0,1\}^\kappa$ and computes $\tilde{C}'_j \leftarrow \mathtt{PRF}_1(K_j, ctr + j \bmod 2^\kappa)$ and returns the final ciphertext C_i to $\mathcal{A}$ following $\mathtt{FSE.Enc}$.
3. All $\mathtt{PRF}_2$ evaluations in $\mathtt{FPRG}$ are computed honestly using uniformly random keys.

If O is instantiated with $\mathtt{PRF}_1(K^\star, \cdot)$, the simulation is identical to hybrid $H^{(1)}_{i-1}$; if O is random, the view matches $H^{(1)}_i$. Therefore, $\Delta^{(1)}_i$ is bounded by the distinguishing advantage of $\mathcal{B}_1$ (i.e., $\Delta^{(1)}_i \leq \mathtt{Adv}_{\mathtt{PRF}_1}(t_{\mathtt{PRF}_1}, q_{\mathtt{PRF}_1})$). Then, we sum $\Delta^{(1)}_i$ for all $i \in \{1, \ldots, n\}$ to obtain:

$$\left|\Pr[H^{(1)}_0(\mathcal{A}) = 1] - \Pr[H^{(1)}_n(\mathcal{A}) = 1]\right| = |\sum_{i=1}^{n} \Delta^{(1)}_i| \leq \sum_{i=1}^{n} |\Delta^{(1)}_i| \leq n \cdot \mathtt{Adv}_{\mathtt{PRF}_1}(t_{\mathtt{PRF}_1}, q_{\mathtt{PRF}_1}).$$

Game $\mathbf{H}^{(2)}_\mathbf{i}$ *(Replacing PRF$_2$ in FPRG).* Let $H^{(2)}_0$ be $H^{(1)}_n$, for each $i = 1, \ldots, n$, let $H^{(2)}_i$ be identical to $H^{(2)}_{i-1}$ except that i^{th} key update $(S_{i+2}, K_{i+1}) \leftarrow \mathtt{FPRG.Upd}(S_{i+1})$ is replaced by random generation $(S_{i+2} \xleftarrow{\$} \{0,1\}^\kappa, K_{i+1} \leftarrow \{0,1\}^\kappa)$. We denote $\Delta^{(2)}_i = \left|\Pr[H^{(2)}_{i-1}(\mathcal{A}) = 1] - \Pr[H^{(2)}_i(\mathcal{A}) = 1]\right|$.

The distinguisher $\mathcal{B}_2$ is given oracle access to a function $O(\cdot)$ which is either $\mathtt{PRF}_2(K^\star, \cdot)$ where $K^\star \leftarrow \{0,1\}^\kappa$ or a random function. It simulates the $\mathtt{F\text{-}IND\text{-}CPA}$ experiment for $\mathcal{A}$ as follows:

1. For the target update i, $\mathcal{B}_1$ computes (S_{i+2}, K_{i+1}) by querying $O(S_{i+1}\|0)$ and $O(S_{i+1}\|1)$.
2. For $j \neq i$, it computes $(S_{i+2}, K_{i+1}) \leftarrow (\mathtt{PRF}_2(S_{i+1}, 0), \mathtt{PRF}_2(S_{i+1}, 1))$ where $S_{i+1} \xleftarrow{\$} \{0,1\}^\kappa$.

If O is instantiated with $\mathtt{PRF}_2(S_{i+1}, .)$, the simulation is identical to $H^{(2)}_{i-1}$. Otherwise, if random, the view matches $H^{(2)}_i$. Therefore, the distinguishing advantage of $\mathcal{B}_2$ equals $\Delta^{(2)}_i$ (i.e., $\Delta^{(2)}_i \leq 2 \cdot \mathtt{Adv}_{\mathtt{PRF}_2}(t_{\mathtt{PRF}_2}, q_{\mathtt{PRF}_2})$). Then, we sum $\Delta^{(2)}_i$ for all $i \in \{1, \ldots, n\}$ to obtain:

$$\left|\Pr[H^{(1)}_n(\mathcal{A})] - \Pr[H^{(2)}_n(\mathcal{A})]\right| = |\sum_{i=1}^{n} \Delta^{(2)}_i| \leq \sum_{i=1}^{n} |\Delta^{(2)}_i| \leq 2 \cdot n \cdot \mathtt{Adv}_{\mathtt{PRF}_2}(t_{\mathtt{PRF}_2}, q_{\mathtt{PRF}_2}).$$

Final game $\mathbf{H}^{(2)}_\mathbf{n}$. In $H^{(2)}_n$, every $\mathtt{PRF}_{1,2}$ outputs is replaced by uniform random values. Hence, ciphertexts are generated by a fresh one-time pad, and all keys are uniformly random values. Therefore, the adversary's success probability $\Pr[H^{(2)}_n(\mathcal{A}) = 1]$ equals the random guess probability $1/2$. Notice that $\mathtt{Adv}^{\mathtt{F\text{-}IND\text{-}CPA}}_{\mathtt{FSE}}(t_{\mathtt{FSE}}, q_{\mathtt{FSE}}) = \max_{\mathcal{A}} |\Pr[\mathrm{Expt}^{\mathtt{F\text{-}IND\text{-}CPA}}_{\mathtt{FSE}}(\mathcal{A}) = 1] - 1/2|$ which is equivalent to $\mathtt{Adv}^{\mathtt{F\text{-}IND\text{-}CPA}}_{\mathtt{FSE}}(t_{\mathtt{FSE}}, q_{\mathtt{FSE}}) = \max_{\mathcal{A}} |\Pr[H^{(1)}_0(\mathcal{A}) = 1] - \Pr[H^{(2)}_n(\mathcal{A}) = 1]|$. That is,

$$\mathtt{Adv}^{\mathtt{F\text{-}IND\text{-}CPA}}_{\mathtt{FSE}}(t_{\mathtt{FSE}}, q_{\mathtt{FSE}}) \leq |\Pr[H^{(1)}_0(\mathcal{A}) = 1] - \Pr[H^{(1)}_n(\mathcal{A}) = 1]| + |\Pr[H^{(1)}_n(\mathcal{A}) = 1] - \Pr[H^{(2)}_n(\mathcal{A}) = 1]|$$

By combining the results of the hybrid games, we obtain the desired bound and conclude the proof:

$$\mathtt{Adv}^{\mathtt{F\text{-}IND\text{-}CPA}}_{\mathtt{FSE}}(t_{\mathtt{FSE}}, q_{\mathtt{FSE}}) \leq n \cdot \mathtt{Adv}_{\mathtt{PRF}_1}(t_{\mathtt{PRF}_1}, q_{\mathtt{PRF}_1}) + 2 \cdot n \cdot \mathtt{Adv}_{\mathtt{PRF}_2}(t_{\mathtt{PRF}_2}, q_{\mathtt{PRF}_2}).$$

□

Theorem 2. *Let* $\mathtt{FAMAC}$ *be the forward-secure aggregate* $\mathtt{MAC}$ *scheme in Fig. 4, instantiated from an ε-almost-universal hash* $\mathtt{UH}$ *and a pseudo-random function* $\mathtt{PRF}_2$. *Let $\mathcal{A}$ be any adversary running in time t_{FAMAC} and making at most q_{FAMAC} signing queries, where the epoch size is b and the tag space $\mathcal{T}$ is of size τ. Then there exists a distinguisher $\mathcal{B}_{PRF}$ against $\mathtt{PRF}_2$ such that*

$$\mathtt{Adv}^{\mathtt{FA\text{-}EU\text{-}CMA}}_{\mathtt{FAMAC}}(t_{FAMAC}, q_{FAMAC}) \;\leq\; n \cdot \mathtt{Adv}_{\mathtt{PRF}_2}(t_{PRF}, q_{PRF}) \;+\; q_{FAMAC} \cdot b \cdot \varepsilon \;+\; \frac{q_{FAMAC} \cdot b}{2^\tau},$$

where $q_{PRF} = O(q_{FAMAC} \cdot b)$ and $t_{PRF} = t_{FAMAC} + O(q_{FAMAC} \cdot b)$.

Proof. We prove the theorem by a sequence of hybrid games, as follows.

Game $\mathbf{H_0}$ *(Real experiment).* This is the real FA-EU-CMA experiment where each offline mask $\tilde{\sigma}_i \leftarrow \mathtt{PRF}_2(K_i^{\mathtt{PRF}_2}, i)$ and UH term $\bar{\sigma}_i \leftarrow \mathtt{UH}(K_i^{\mathtt{UH}}, M_i)$ are aggregated additively ($\sigma_i \leftarrow \tilde{\sigma}_i + \bar{\sigma}_i$).

Game $\mathbf{H_i^{(1)}}$ *(Replace PRF$_2$ output for epoch i).* For $i = 1, \ldots, n$, we define $H_i^{(1)}$ as $H_{i-1}^{(1)}$ except that in epoch i all offline tags $\tilde{\sigma}_i$ derived from $\mathtt{PRF}_2$ are replaced by uniformly random elements of $\mathcal{T}$. We define $\Delta_i^{(1)} = \left|\Pr[H_{i-1}^{(1)}(\mathcal{A}) = 1] - \Pr[H_i^{(1)}(\mathcal{A}) = 1]\right|$ to be the difference in the advantage of $\mathcal{A}$ in two consecutive games $H_{i-1}^{(1)}$ and $H_i^{(1)}$. The distinguisher $\mathcal{B}_{\mathtt{PRF}}$ is given access to an oracle $O(\cdot)$, which is either $\mathtt{PRF}_2(K_i, \cdot)$ or a truly random function. $\mathcal{B}_1$ chooses the target epoch i and simulates FA-EU-CMA for $\mathcal{A}$ as follows:

1. For epoch i, $\mathcal{B}_1$ queries $\bar{\sigma}_i \leftarrow O(M_i)$ and outputs $\sigma_i \leftarrow \bar{\sigma}_i + \tilde{\sigma}_i$ to $\mathcal{A}$ where $\tilde{\sigma}_i \leftarrow \mathtt{UH}(K_i^{\mathtt{UH}}, i)$.
2. For $j \neq i$, $\mathcal{B}_1$ computes σ_i following the FAMAC signing algorithm using uniformly random keys.

If O is instantiated with $\mathtt{PRF}_2$, the simulation equals $H_{i-1}^{(1)}$; if random, the view matches $H_i^{(1)}$. Hence $\Delta_i^{(1)} = \mathtt{Adv}_{\mathtt{PRF}_2}(t_{\mathtt{PRF}}, q_{\mathtt{PRF}})$. Then, we sum $\Delta_i^{(1)}$ over $i \in \{1, \ldots, n\}$ to obtain:

$$\left|\Pr[H_0(\mathcal{A}) = 1] - \Pr[H_n^{(1)}(\mathcal{A}) = 1]\right| \;\leq\; n \cdot \mathtt{Adv}_{\mathtt{PRF}_2}(t_{\mathtt{PRF}}, q_{\mathtt{PRF}}). \tag{1}$$

Game $\mathbf{H_n^{(1)}}$ *(All offline masks random).* In this game, all $\tilde{\sigma}_i$ are chosen uniformly at random, therefore independent of $\mathcal{A}$'s view. For each authentication tag $\sigma_i = \tilde{\sigma}_i + \bar{\sigma}_i$, a successful forgery requires $\sigma' - \bar{\sigma}' = \tilde{\sigma}$. Since $\tilde{\sigma}$ is uniformly random from $\mathcal{T}$, this equality holds with probability at most $\frac{1}{\tau}$ for any (M', σ') pair. The collisions in UH outputs provide another path to forgery. Since UH is an ε-almost-universal hash function, any pair of distinct messages collide with probability at most ε. By a union bound over $q_{\mathtt{FAMAC}} \cdot b$ total attempts:

$$\Pr[H_n^{(1)}(\mathcal{A}) = 1] \;\leq\; q_{\mathtt{FAMAC}} \cdot b \cdot \varepsilon \;+\; \tfrac{q_{\mathtt{FAMAC}} \cdot b}{\tau}. \tag{2}$$

Combining (1) and (2). We obtain the desired reduction and therefore concludes the proof.

$$\mathtt{Adv}^{\mathtt{FA\text{-}EU\text{-}CMA}}_{\mathtt{FAMAC}}(\mathcal{A}) \;\leq\; n \cdot \mathtt{Adv}_{\mathtt{PRF}_2}(t_{\mathtt{PRF}}, q_{\mathtt{PRF}}) \;+\; q_{\mathtt{FAMAC}} \cdot b \cdot \varepsilon \;+\; \tfrac{q_{\mathtt{FAMAC}} \cdot b}{\tau},$$

□

Theorem 3. *Let* Diamond *be a forward-secure and aggregate authenticated encryption (FAAE) scheme constructed from a CTR-based forward-secure encryption (FSE) and a universal forward-secure aggregate MAC (FAMAC), following Fig.3 and Fig.4, respectively.*

Let $\mathcal{A}$ be an adversary running in time t and making at most q queries to the Diamond-AE *oracle. Then, there exist adversaries $\mathcal{A}_{FSE}$ and $\mathcal{A}_{FAMAC}$ against the underlying FSE and FAMAC such that*

$$Adv^{F\text{-}IND\text{-}CPA}_{Diamond}(t, q) \leq b \cdot Adv^{F\text{-}IND\text{-}CPA}_{FSE}(t_{FSE}, q_{FSE}),$$

$$Adv^{FA\text{-}EU\text{-}CMA}_{Diamond}(t, q) \leq Adv^{FA\text{-}EU\text{-}CMA}_{FAMAC}(t_{FAMAC}, q_{FAMAC}),$$

where $t_{FSE}, t_{FAMAC} \leq t$, and $q_{FSE} = q_{FAMAC} = q + O(1)$.

Proof. We prove that the security of Diamond under the F-IND-CPA and FA-EU-CMA notions reduces to that of its underlying components: the forward-secure encryption FSE and the forward-secure aggregate message authentication FAMAC.

(1) Confidentiality Reduction to FSE. In the following, we construct an adversary $\mathcal{A}^{\mathtt{FSE}}$ that leverages $\mathcal{A}$ to break the F-IND-CPA security of FSE scheme.

Setup. $\mathcal{A}^{\mathtt{FSE}}$ begins by sampling an initial FAMAC key $K_1^{\mathtt{FAMAC}} \xleftarrow{\$} \{0,1\}^{|K_1^{\mathtt{FAMAC}}|}$ and internally runs $\mathcal{A}$. *Find.* Whenever $\mathcal{A}$ issues a left-or-right encryption query (M_0, M_1) to $\mathtt{AuthEnc\text{-}LR}(K_i, \cdot)$, $\mathcal{A}_{\mathtt{FSE}}$ forwards it to its own $\mathtt{FSE.Enc\text{-}LR}(K_i^{\mathtt{FSE}}, \cdot)$ oracle and receives back a ciphertext C_b. It then computes

$\sigma_i \leftarrow \texttt{FAMAC.Sign}(K_i^{\texttt{FAMAC}}, C_i^b)$ and updates the FAMAC key as $K_{i+1}^{\texttt{FAMAC}} \leftarrow \texttt{FAMAC.Upd}(K_i^{\texttt{FAMAC}})$. Finally, $\mathcal{A}^{\texttt{FSE}}$ returns (C_i^b, σ_i) to $\mathcal{A}$.

Guess. Whenever $\mathcal{A}$ outputs its final bit b', $\mathcal{A}^{\texttt{FSE}}$ also outputs b'.

$\mathcal{A}^{\texttt{FSE}}$ simulates the Diamond encryption oracle for $\mathcal{A}$. $\mathcal{A}^{\texttt{FSE}}$ succeeds in the F-IND-CPA experiment of FSE whenever $\mathcal{A}$ succeeds in guesses the left-or-right bit (i.e., $b = b'$). Therefore,

$$\texttt{Adv}^{\texttt{F-IND-CPA}}_{\texttt{Diamond}}(t, q) \leq b \cdot \texttt{Adv}^{\texttt{F-IND-CPA}}_{\texttt{FSE}}(t_{\texttt{FSE}}, q_{\texttt{FSE}}).$$

(2) Integrity Reduction to FAMAC. We now construct an adversary $\mathcal{A}^{\texttt{FAMAC}}$ that leverages $\mathcal{A}$ to break the FA-EU-CMA security of FAMAC:

Setup. $\mathcal{A}^{\texttt{FAMAC}}$ generates an initial FSE key $K_1^{\texttt{FSE}} \xleftarrow{\$} \{0,1\}^{|K_1^{\texttt{FSE}}|}$ to locally simulate the encryption component. It then launches $\mathcal{A}$ as a subroutine.

Find. When $\mathcal{A}$ queries $\texttt{Diamond-AE}(K_i, \cdot)$ on message M_i, $\mathcal{A}^{\texttt{FAMAC}}$ computes $C_i \leftarrow \texttt{FSE.Enc}(K_i^{\texttt{FSE}}, M_i)$ and then queries its own FAMAC.Sign oracle on C_i to obtain σ_i. It updates the encryption key $K_{i+1}^{\texttt{FSE}} \leftarrow \texttt{FSE.Upd}(K_i^{\texttt{FSE}})$ and returns (C_i, σ_i) to $\mathcal{A}$.

Forge. Eventually, $\mathcal{A}$ proceeds to the forge phase and outputs a non-trivial forgery $(\boldsymbol{C}_{i^*}, \sigma_{i^*, i^*+b-1})$, following the FA-EU-CMA experiment in Sec. 6.1. $\mathcal{A}^{\texttt{FAMAC}}$ also receives the current secret key $K_i^{\texttt{FAMAC}}$ (i.e., after a break-in). It verifies $\mathcal{A}$'s forgery using its verification oracle (i.e., $b \leftarrow \texttt{FAMAC.AVer}(K_i^{\texttt{FAMAC}}, \boldsymbol{C}_{i^*}, \sigma_{i^*, i^*+b-1})$.). If $b = 1$, $\mathcal{A}^{\texttt{FAMAC}}$ decrypts each ciphertext as (i.e., $\boldsymbol{M}_{i^*} \leftarrow \{\texttt{FSE.Dec}(K_j^{\texttt{FSE}}, C_j)\}_{j=i^*}^{i^*+b-1}$.) Finally, $\mathcal{A}^{\texttt{FAMAC}}$ outputs success if $\mathcal{A}$'s forgery passes verification and is non-trivial (i.e., not a recombination of previously authenticated ciphertexts).

$\mathcal{A}^{\texttt{FAMAC}}$ wins the FA-EU-CMA experiment of FAMAC whenever $\mathcal{A}$ forges a valid ciphertext–tag pair under Diamond. Hence, we obtain the security bound which concludes the proof.

$$\texttt{Adv}^{\texttt{FA-EU-CMA}}_{\texttt{Diamond}}(t, q) \leq \texttt{Adv}^{\texttt{FA-EU-CMA}}_{\texttt{FAMAC}}(t_{\texttt{FAMAC}}, q_{\texttt{FAMAC}}).$$

□

7 CONCLUSION

We presented Diamond, a provable secure symmetric-key Forward-secure and Aggregate Authenticated Encryption (FAAE) framework tailored for the stringent constraints of contemporary IoT platforms. Diamond addresses foundational limitations in existing AE and MAC constructions by synergizing efficient forward-secure key evolution, OO cryptographic optimization, and compact tag aggregation. Diamond guarantees breach-resilient confidentiality and authenticity, while ensuring near-optimal online latency across heterogeneous hardware classes. Through a comprehensive experimental evaluation on 64-bit ARM Cortex-A72, 32-bit ARM Cortex-M4, and 8-bit AVR microcontrollers, we demonstrated that Diamond's instantiations deliver substantial reductions in offline preprocessing, order-of-magnitude improvements in online AuthEnc and verification throughput, and significantly lower end-to-end latency for large telemetry batches compared to FAAE baselines and NIST lightweight AE candidates. Diamond's modular design, extensibility to diverse universal AMAC constructions, and compatibility with commodity toolchains make it a practical, deployment-ready FAAE primitive for mission-critical IoT domains. To encourage reproducibility and adoption, we release our open-source full-fledged implementation.

ACKNOWLEDGMENTS

This work was supported by both Army Research Laboratory W911NF-24-2-0078 and National Science Foundation NSF-SNSF 2444615. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government

REFERENCES

[1] Mihir Bellare and Chanathip Namprempre. 2000. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *International Conference on the Theory and Application of Cryptology and Information Security*. 531–545.

[2] Mihir Bellare and Bennet Yee. 2003. Forward-security in private-key cryptography. In *Topics in Cryptology—CT-RSA 2003: The Cryptographers' Track at the RSA Conference*. 1–18.

[3] Daniel J. Bernstein. 2005. The Poly1305-AES Message-Authentication Code. In *Fast Software Encryption: 12th International Workshop, Revised Selected Papers*. 32–49.

[4] Colin Boyd and Kai Gellert. 2021. A modern view on forward security. *Comput. J.* 64, 4 (2021), 639–652.

[5] Luan Cardoso dos Santos, Johann Großschädl, and Alex Biryukov. 2019. FELICS-AEAD: benchmarking of lightweight authenticated encryption algorithms. In *Inter. Conf. on Smart Card Research and Advanced Applications*. 216–233.

[6] Eric Cronin, Sugih Jamin, Tal Malkin, and Patrick McDaniel. 2003. On the performance, feasibility, and use of forward-secure signatures. In *Proc. of the 10th ACM conference on Computer and communications security*. 131–144.

[7] Jean Paul Degabriele, Jan Gilcher, Jérôme Govinden, and Kenneth G Paterson. 2024. SoK: Efficient Design and Implementation of Polynomial Hash Functions over Prime Fields. In *IEEE Sym. on Security and Privacy (SP)*.

[8] Yevgeniy Dodis, Daniel Jost, and Harish Karthikeyan. 2022. Forward-secure encryption with fast forwarding. In *Theory of Cryptography Conference*. Springer, 3–32.

[9] Oliver Eikemeier, Marc Fischlin, Jens-Fabian Gotzmann, Anja Lehmann, Dominique Schroder, Peter Schroder, and Daniel Wagner. 2010. History-free aggregate message authentication codes. In *Security and Cryptography for Networks: 7th Int. Conf. SCN, September 13-15, 2010. Proceedings 7*. 309–328.

[10] Mark Etzel, Sarvar Patel, and Zulfikar Ramzan. 1999. Square hash: Fast message authentication via optimized universal hash functions. In *Annual International Cryptology Conference*. 234–251.

[11] Matthew D Green and Ian Miers. 2015. Forward secure asynchronous messaging from puncturable encryption. In *2015 IEEE Symposium on Security and Privacy*. IEEE, 305–320.

[12] Shibo He, Kun Shi, Chen Liu, Bicheng Guo, Jiming Chen, and Zhiguo Shi. 2022. Collaborative sensing in Internet of Things: A comprehensive survey. *IEEE Communications Surveys & Tutorials* 24, 3 (2022), 1435–1474.

[13] Jens Hiller, Martin Henze, Martin Serror, Eric Wagner, Jan Niklas Richter, and Klaus Wehrle. 2018. Secure low latency communication for constrained industrial IoT scenarios. In *2018 IEEE 43rd Conf on Local Computer Networks*. 614–622.

[14] Shoichi Hirose and Hidenori Kuwakado. 2014. Forward-secure sequential aggregate message authentication revisited. In *International Conference on Provable Security*. Springer, 87–102.

[15] Gael Hofemeier and Robert Chesebrough. 2012. Introduction to intel aes-ni and intel secure key instructions. *Intel, White Paper* 62 (2012), 6.

[16] Michael Hutter and Peter Schwabe. 2013. NaCl on 8-bit AVR microcontrollers. In *International Conference on Cryptology in Africa*. 156–172.

[17] Jonathan Katz and Andrew Y Lindell. 2008. Aggregate message authentication codes. In *Cryptographers' Track at RSA Conf.* 155–169.

[18] Safiullah Khan, Wai-Kong Lee, and Seong Oun Hwang. 2021. Scalable and efficient hardware architectures for authenticated encryption in IoT applications. *IEEE Internet of Things Journal* 8, 14 (2021), 11260–11275.

[19] Adam Langley and Yael Nir. 2021. rfc7539: ChaCha20 and Poly1305 for IETF Protocols.

[20] Huina Li, Le He, Shiyao Chen, Jian Guo, and Weidong Qiu. 2023. Automatic preimage attack framework on Ascon using a linearize-and-guess approach. *IACR Transactions on Symmetric Cryptology* (2023).

[21] He Li, Vireshwar Kumar, Jung-Min Park, and Yaling Yang. 2021. Cumulative message authentication codes for resource-constrained IoT networks. *IEEE Internet of Things Journal* 8, 15 (2021), 11847–11859.

[22] Ying Liu, Weiting Zhang, Letian Li, Juqin Wu, Yu Xia, Shuai Gao, and Hongke Zhang. 2024. Toward autonomous trusted networks-from digital twin perspective. *IEEE Network* 38, 3 (2024), 84–91.

[23] Di Ma and Gene Tsudik. 2007. Forward-secure sequential aggregate authentication. In *2007 IEEE Symposium on Security and Privacy (SP'07)*. IEEE, 86–91.

[24] D. Ma and G. Tsudik. 2007. Forward-Secure Sequential Aggregate Authentication. In *2007 IEEE Symposium on Security and Privacy*.

[25] Giorgia Azzurra Marson and Bertram Poettering. 2014. Even more practical secure logging: Tree-based seekable sequential key generators. In *European Symposium on Research in Computer Security*. Springer, 37–54.

[26] David A McGrew and John Viega. 2004. The security and performance of the Galois/Counter Mode (GCM) of operation. In *International Conference on Cryptology in India*. Springer, 343–355.

[27] Yoav Nir and Adam Langley. 2018. *ChaCha20 and Poly1305 for IETF Protocols*. Technical Report.
[28] Saif E Nouma and Attila A Yavuz. 2024. Trustworthy and efficient digital twins in post-quantum era with hybrid hardware-assisted signatures. *ACM Transactions on Multimedia Computing, Communications and Applications* 20, 6 (2024), 1–30.
[29] Saif E Nouma and Attila A Yavuz. 2025. Lightweight and Breach-Resilient Authenticated Encryption Framework for Internet of Things. *IEEE Military Communications Conference (MILCOM)* (2025).
[30] Krzysztof Piotrowski, Peter Langendoerfer, and Steffen Peter. 2006. How public key cryptography influences wireless sensor node lifetime. In *Proceedings of the fourth ACM workshop on Security of ad hoc and sensor networks*. 169–176.
[31] Paulo HL Rettore, Jannis Mast, Thorsten Aurisch, Aline Carneiro Viana, Peter Sevenich, and Bruno P Santos. 2025. Military IoT from Management to Perception: Challenges and Opportunities Across Layers. *IEEE Internet of Things Magazine* 8, 2 (2025), 25–31.
[32] Joseph Salowey, Abhijit Choudhury, and David McGrew. 2008. *AES Galois Counter Mode (GCM) cipher suites for TLS*. Technical Report.
[33] Juliet Samandari and Clémentine Gritti. 2025. Online/Offline Digital Signatures: A Systematic Literature Review. *IEEE Access* (2025).
[34] Izaz Ali Shah, Muhammad Zada, Abdul Basir, Syed Ahson Ali Shah, Usman Rizqi Iman, Young-Hyo Lim, and Hyoungsuk Yoo. 2024. Efficient wirelessly-powered biotelemetric system for IoMT-enabled leadless pacemakers in dynamic cardiac environments. *IEEE Internet of Things Journal* (2024).
[35] Yuba Raj Siwakoti, Manish Bhurtel, Danda B Rawat, Adam Oest, and RC Johnson. 2023. Advances in IoT security: Vulnerabilities, enabled criminal services, attacks, and countermeasures. *IEEE Internet of Things Journal* 10, 13 (2023), 11224–11239.
[36] David Trilla, Carles Hernandez, Jaume Abella, and Francisco J Cazorla. 2019. Worst-case energy consumption: A new challenge for battery-powered critical devices. *IEEE Transactions on Sustainable Computing* 6, 3 (2019), 522–530.
[37] Meltem Sönmez Turan, Kerry McKay, Donghoon Chang, Jinkeon Kang, and John Kelsey. 2024. Ascon-based lightweight cryptography standards for constrained devices. In *NIST Special Publication (SP) NIST SP 800–232 ipd*.
[38] Venkatasubramanian Viswanathan, Alan H Epstein, Yet-Ming Chiang, Esther Takeuchi, Marty Bradley, John Langford, and Michael Winter. 2022. The challenges and opportunities of battery-powered flight. *Nature* 601, 7894 (2022), 519–525.
[39] Eric Wagner, Jan Bauer, and Martin Henze. 2022. Take a bite of the reality sandwich: revisiting the security of progressive message authentication codes. In *Proceedings of the 15th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. 207–221.
[40] Eric Wagner, David Heye, Jan Bauer, Klaus Wehrle, and Martin Serror. 2025. MAC Aggregation over Lossy Channels in DTLS 1.3. In *2025 IEEE 33rd International Conference on Network Protocols (ICNP)*. IEEE, 1–11.
[41] Eric Wagner, Martin Serror, Klaus Wehrle, and Martin Henze. 2022. BP-MAC: fast authentication for short messages. In *15th ACM conf. on security and privacy in wireless and mobile networks*. 201–206.
[42] Eric Wagner, Martin Serror, Klaus Wehrle, and Martin Henze. 2024. When and How to Aggregate Message Authentication Codes on Lossy Channels?. In *Int. Conf. on Applied Cryptography and Network Security*. 241–264.
[43] Xiaojie Wang, Zhonghui Zhao, Ling Yi, Zhaolong Ning, Lei Guo, F Richard Yu, and Song Guo. 2024. A survey on security of UAV swarm networks: attacks and countermeasures. *Comput. Surveys* 57, 3 (2024), 1–37.
[44] Zhenning Wang, Yue Cao, Kai Jiang, Huan Zhou, Jiawen Kang, Yuan Zhuang, Daxin Tian, and Victor CM Leung. 2024. When crowdsensing meets smart cities: A comprehensive survey and new perspectives. *IEEE Communications Surveys & Tutorials* 27, 2 (2024), 1101–1151.
[45] Jianghong Wei, Xiaofeng Chen, Jianfeng Ma, Xuexian Hu, and Kui Ren. 2021. Communication-efficient and fine-grained forward-secure asynchronous messaging. *IEEE/ACM Transactions on Networking* 29, 5 (2021), 2242–2253.
[46] Tahreem Yaqoob, Haider Abbas, and Mohammed Atiquzzaman. 2019. Security vulnerabilities, attacks, countermeasures, and regulations of networked medical devices—A review. *IEEE Com. Surveys & Tutorials* 21, 4 (2019).
[47] Attila A Yavuz, FATİH ALAGÖZ, and Emin Anarim. 2010. A new multi-tier adaptive military MANET security protocol using hybrid cryptography and signcryption. *Turkish Journal of Electrical Engineering and Computer Sciences* 18, 1 (2010), 1–22.
[48] Attila A. Yavuz and Peng Ning. 2012. Self-sustaining, efficient and forward-secure cryptographic constructions for Unattended Wireless Sensor Networks. *Ad Hoc Networks* 10, 7 (2012), 1204–1220.