Scilight

Pragmatic Cybersecurity
https://www.sciltp.com/journals/pc

Article

Check for updates

Assessing Latency of Lightweight AEAD Algorithms in CAN †

Joshua Copeland [1,*,‡], Leonie Simpson [1,‡] and Geoffrey Walker [2,‡]

[1] School of Computer Science, Faculty of Science, Queensland University of Technology, Brisbane, QLD 4000, Australia
[2] School of Electrical Engineering and Robotics, Faculty of Engineering, Queensland University of Technology, Brisbane, QLD 4000, Australia
* Correspondence: joshua.copeland@connect.qut.edu.au

† Extended from: Securing CAN Bus Transmissions with Lightweight AEAD Ciphers. In Proceedings of AISC 2026, Melbourne, VIC, Australia, 11–12 February 2026.
‡ These authors contributed equally to this work.

How To Cite: Copeland, J.; Simpson, L.; Walker, G. Assessing Latency of Lightweight AEAD Algorithms in CAN. *Pragmatic Cybersecurity* **2026**, *1*(1), 7. https://doi.org/10.53941/pc.2026.100007

Received: 16 April 2026
Revised: 23 June 2026
Accepted: 29 June 2026
Published: 30 June 2026

Abstract: The CAN protocol does not include cryptographic security mechanisms. While older vehicles had physically isolated CAN networks; modern vehicles' external connectivity can result in remote attack vectors as demonstrated by Miller and Valasek's Jeep Hacking in 2015. Some proposals for cryptographically securing CAN transmissions use specialised hardware and/or use non-lightweight cryptographic algorithms. In this paper we consider the suitability of lightweight Authenticated Encryption with Associated Data algorithms for providing security to the CAN protocol. NIST held a lightweight cryptography standardisation process, announcing ten finalists in 2021, and selecting ASCON from the finalists in 2023. This work aims to evaluate software-implementations of these algorithms for securing the very short payload lengths of standard CAN frames, without requiring specific CAN variants and/or specialised hardware for longer payloads or cryptographic acceleration. This would enable implementation of CAN bus security on low-end existing vehicle MCUs via firmware, without hardware modification, by using such algorithms with any existing CAN protocols that use encryption and/or MACs. We tested by re-purposing Weatherly's benchmarking software via modifying the test procedure to test shorter 4-byte payloads with authentication data in existing CAN frames. Low-end 8-bit 16 MHz and 20 MHz AVR processors are used for benchmarking, and results are used to rank finalists with respect to their time-efficiency advantage over AES-GCM. We also compare to encrypting the payload without including the ID and control bits as associated data to assess the performance impact of the associated data, and also compare to authentication-only use of the AEAD cipher where both the ID and payload are all processed as associated data to authenticate without encrypting. We find that several NIST finalists outperform AES-GCM in our testing of AEAD in the CAN data field. ASCON and TinyJAMBU are particularly time-efficient among nonce-misuse resilient finalists. If nonce-misuse resilience is not required, Schwaemm-128-128 has the best time-efficiency even against ASCON and TinyJAMBU variants, with this advantage being to an even greater extent in AD-based authentication-only results. The finalists' varying results against AES-GCM, and some discussion about other aspects such as nonce-misuse resilience and code-size are discussed. We also found that the relative overhead of processing the ID as associated data ranges from 50% to less than 1% across the tested cipher variants. We also found that while most cipher variants are more efficient, and some have negligible difference, when processing the payload as associated data; there are cipher variants that are less efficient.

Keywords: CAN bus; lightweight cryptography; authenticated encryption; vehicular communications

1. Introduction

1.1. Background of Automotive Security

Various tasks within vehicles can be digitally controlled by automotive electronic control units (ECUs) equipped with microprocessors [1], which are ubiquitous in real-world embedded systems. ECUs coordinate vehicle operations by communicating via industry-standardised protocols, examples including CAN and Local Interconnect Network (LIN) [1]. Such protocols have increasingly replaced the non-standard supplementary wiring previously used [2].

Addressing safety via the provision of mechanisms for fault management and environmental electromagnetic interference mitigation have been the typical focus of internal vehicular system standards. Due to such systems typically being isolated from external cyber environments, cyber security was usually not considered. However, this isolation is often not present in newer vehicles which increasingly feature external connectivity in their ECUs and/or infotainment systems; which introduce new potential external attack vectors. Remote manipulation of internal communication field buses may be performed by attackers that compromise such externally connected ECUs [3]. Miller and Valasek [4] notably showcased this in 2015 with their "Jeep hacking" demonstration. Physically connected vehicle controls unaffected by such attacks have been increasingly replaced with "by-wire" digital vehicle controls that can be commandeered in these attacks [1]. Furthermore, vehicles with multiple communications networks/protocols typically bridge them to carry out certain functions involving ECUs across them, so isolating critical components on a separate network/protocol is not viable as this impairs vehicle functionality [2].

1.2. Cryptography and Automotive Security

Cryptographic mechanisms can be used to detect alterations of transmitted data. However, CAN bus did not incorporate such mechanisms since ECUs did not communicate externally when CAN bus was designed. Since then, vehicular technologies have further integrated external cyber communications, presenting a possible vector for remote cyber intrusion of vehicles. Current research on vehicular communications has increasingly focused on intrusion detection systems (IDS), examples include [5–7]; use of artificial intelligence techniques are also increasingly being incorporated into IDS approaches for CAN such as [8–18]. However, this is complementary to the use of cryptographic mechanisms. This paper focuses on cryptographic mechanisms rather than network security.

Authenticated Encryption with Associated Data (AEAD) ciphers can be implemented to provide confidentiality and integrity for transmitted data, but such algorithms will require additional resources such as processing time, program storage space, etc. Latency is the time taken from the sending device starting to prepare the CAN frame to the receiving device having finalised the processing the receipt of the CAN frame. When cryptography is involved, this will include the cryptographic delay, as the latency begins with the sender processing the encryption and/or authentication of frame data before preparing and sending the frame and the recipient needing to decrypt and/or verify the data received from the frame. Any cryptographic algorithm used must have sufficient time-efficiency on typical low-end automotive-application microprocessors, to ensure adherence to maximum latency requirements for transmissions that vehicle operations depend on for real-time operation.

The CAN frame format has small field sizes, some of which cannot be encrypted such as the CAN ID. Furthermore, some CAN bus environments have nodes with limited computing resources. Thus, lightweight Authenticated Encryption with Associated Data (AEAD) ciphers may be suitable for provision of confidentiality and integrity for this environment. The encrypted payload along with the authentication tag and any other necessary data (e.g., nonce) can be placed in the CAN frame's data field. Frame fields that must not be encrypted to allow CAN hardware to operate as intended, such as the CAN ID and control bits, can be processed as associated data. Any spoofing or modification attempts made by an attacker against the encrypted payload or the associated data processed frame fields will be detected by a mismatched authentication tag. Prior works have applied approaches to implementation cryptography, including AEAD, for CAN messages, and have typically provided timings for the overhead incurred.

1.3. Our Contributions

NIST ran a lightweight AEAD cryptography standardisation process, in which they announced the ten finalists in 2021. ASCON was chosen by NIST for standardisation in 2023, then standardised in 2025 [19]. Our motivation is to evaluate the ten NIST lightweight AEAD finalists' performance and suitability when applied to the particularly short payload lengths in the existing CAN protocol. Despite the conclusion of the standardisation process, all ten finalists were assessed. While NIST's evaluation was general-purpose, we particularly focus on time-efficiency for

the very short payloads in standard CAN frames. This is due to our objective: determining suitable algorithms for use with CAN, when utilising typical low-end automotive microcontrollers limited in processing capability.

This paper extends the work presented at the 17th Australasian Information Security Conference (AISC) 2026 [20]. Contributions in that paper included:

(1) Modifying Weatherly's available benchmarking code to include 4-byte payloads in testing and to make the implementations utilise a shorter three-byte tag length.
(2) Recording timings of encryption, decryption, and overall transmission times for CAN frames when secured using Weatherly's implementation of the finalist algorithms on 8-bit AVR based Arduino boards using MCP2515 chips for CAN communication.
(3) Discussion of various aspects such as nonce-misuse resilience and comparative code size, along with the time-efficiency in CAN bus security measured by our experimental data, for an overview which finalist is best suited to which use case.
(4) Comparing the NIST lightweight finalists' execution times against previous CAN cryptography works' experimental timing results from their utilised algorithms.

The extension in this paper is:

(1) Explicit description of the contents of the associated data, and the byte-lengths of both associated data and encrypted message content in our testing.
(2) Additional experimentation to obtain data to enable comparisons between conventional AEAD (included in the original paper), authenticated encryption of the payload without including the ID along with the Data Length Code (DLC) and Remote Transmit Request (RTR) values as associated data (new in this paper), and processing ID and payload together all as associated data for authentication without encryption (new in this paper). This provides a means for determining the overhead of associated data processing and comparison of the efficiency of employing AEAD as a pure authenticator using just associated data processing.

1.4. Paper Structure

The remainder of this paper is structured as follows. Section 2 provides background on CAN bus, related works in automotive pentesting, and existing CAN security approaches. Section 3 describes our application of AEAD to CAN bus transmissions, given our security assumptions and requirements. Section 4 describes our experimentation approach. Section 5 presents the results of our experimentation along with Section 6 providing relevant discussion and comparison of results. Section 7 concludes this paper.

2. Background and Related Work

2.1. The CAN Protocol

BOSCH [21] designed the CAN bus protocol [22] in the 1980s. In the 1990s, CAN was standardised for automotive ECU communications [23]. Vehicle production costs can be reduced by utilising a single bus rather than a multitude of wires for in-vehicle data transmission. CAN bus is a protocol that remains in widespread use; often used as the backbone of ECU networks, even when used in tandem with other protocols such as LIN [24] for completing component connections [25]. Despite the development of successor variants CAN-FD [26] and CAN XL [27] which extend the capabilities of CAN, including a significant increase in maximum data payload, the original CAN specification is still widely used.

2.1.1. Hardware, Differential Transmission, and CSMA/CD

CAN bus utilises a CAN-H (High) line and CAN-L (Low) line to provide differential signalling resistant to electromagnetic interference. CAN-H and CAN-L remain at their bias voltages to represent a one, and are actively driven high and low respectively to represent a zero, as seen in Figure 1. CAN networks operate in a broadcast manner where each device has simultaneous access to these shared lines. The active driving of the zero state is an essential aspect of the Carrier Sense Multiple Access with Collision Detection (CSMA/CD) feature that allows multiple messages to attempt to transmit at the same time. Nodes not driving the lines to transmit a one can see when another node is driving the lines to send a zero, and are able to stop transmission (avoiding a collision), giving lower CAN IDs higher priority. External physical access to the CAN bus lines of a vehicle is typically provided by a standard On-Board Diagnostics II (OBD-II) port [3].

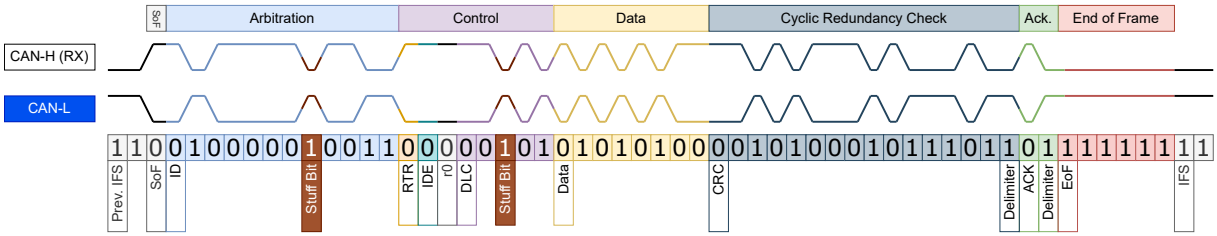

Figure 1. Graphic of a CAN A frame depicting how CAN-H and CAN-L signals work. Stuff bits ensure no more than five consecutive zeros are transmitted. Frame has ID of 515, data is ASCII char 'T'. Cyclic Redundancy Check value calculated using CRC RevEng 3.0.6 [28]. Ack./ACK—Acknowledgement; IFS—Inter-Frame Space; SoF—Start of Frame; RTR—Remote Transmission Request; IDE—Identifier Extension; DLC—Data Length Code; EoF End of Frame, IFS—Interframe space.

2.1.2. Frame Format

To be compatible with existing CAN hardware, an implementation of cryptographic security must respect the existing CAN frame format. Standard CAN frames have two variants, A and B, respectively utilising 11-bit and 29-bit ID lengths; Figure 2 shows the fields of these frame variants. An ID identifying the message, a 4-bit DLC value describing the data field length, followed by the data field which is 1 to 8 bytes long (actual length specified by the DLC value) for the intended payload, a 15-bit CRC for error detection, and an acknowledgement bit which all recipient nodes drive to zero to acknowledge the message. All authentication data is added to the data field alongside the encrypted payload when we later apply AEAD to the CAN frame, avoiding frame format interference.

	SOF	ID	RTR/ SRR	IDE	Ext ID	RTR (B)	r1	r0	DLC	Data	CRC	ACK	ACK Delim.	EOF	IFS
CAN A	0	11-bit	0	0				0	4-bit Value	1-8 Data Bytes	15-bit Checksum	1	0	1	1s Until Next SOF
CAN B		11 Bits	1	1	18 Bits	0	0								

Figure 2. Comparison of CAN bus frame variants A and B. Remote frames set RTR to 1 and omit data field. SRR—Substitute Remote Request.

2.2. CAN Bus Security and Automotive Pentesting

External cyber connectivity has been demonstrated by existing automotive pentesting research, such as Miller and Valasek's work [4], to be a practical vulnerability in modern vehicles making them susceptible to cyber attacks. Exposure of and/or access to CAN bus enables trivial snooping of messages due to its broadcast network structure, trivial blocking of legitimate messages via message priority arbitration abuse, and spoofing messages since no message-source identification or authentication is present in standard CAN [2].

Commandeering a compromised ECU to interface with the CAN network is a typical attack approach . Additional possible attack vectors are malicious devices inserted into the diagnostic port and compromise of wireless devices [2]. Miller and Valasek's [4] work is an example of wireless device compromise. CAN's broadcast design has been identified by existing works as making both single-node monitoring of all traffic [29], as well as sending malicious messages to any node, trivial attacks [2]. Furthermore, standard CAN lacks authentication [29] and lacks any sender or authentication fields [2]. While use of multiple buses to physically isolate ECUs is an alternative security methodology, this may not be fully effective when gateways are used to bridge those buses [29]. Consequently, cryptographic approaches have been explored by existing works for securing CAN communications.

2.3. Existing Approaches to CAN Frame Cryptography

2.3.1. Early Cryptography Applications

In 2005, Chavez et al. [30] utilised RC4 for CAN data field payload encryption. This prevents predictable alteration of the decrypted plaintext payload by an attacker. While noticing random/erroneous decrypted payloads may allow detection of alterations, a lack of a cryptographic authentication mechanism allows alterations that incidentally decrypt to alternative valid values to still be accepted. In 2009, Groll and Ruland [29] proposed that ECUs can belong to groups refereed to as Trusted Communication Groups where each group's messages are

encrypted by a group-scope symmetric key. Although their discussion primarily focuses on symmetric ciphers providing confidentiality, this concept can be used with modern AEAD symmetric ciphers to also provide integrity.

2.3.2. A Shift Toward Authentication

A shift toward cryptographic mechanisms that provide authenticity-checking began in the early 2010s. CANAuth [31], inserted a particularly large 112-bit Hash-Based Message Authentication Code (HMAC) in the extra bits provided by CAN+ [32]. Subsequent approaches worked with standard CAN frames, these include: MaCAN [33]; LCAP [34]; CaCAN [35]; VeCure [36] which had groups of nodes using group-scope keys, similar to Groll and Ruland's [29] concept; and Wu et al. [37] inserting an 8-bit MAC generated using digital watermarking methodology into the CAN frame.

2.3.3. Employing AEAD in the Original CAN Frame

Other works apply AEAD to CAN messages for provision of both confidentiality and authenticity. These include: Bruton's [38] approach which inserts a 32-bit payload, 16-bit authentication tag, and 16-bits of the counter for synchronisation purposes; TOUCAN [39], which uses AES and Chaskey to encrypt a 48-bit payload and generate a 24-bit MAC respectively; and LaaCAN [40] which employs a time-efficiency-optimised software implementation of ChaCha20-Poly1305 [41] to encrypt the data field, but replaces the CAN frame CRC with a 16-bit MAC, making this incompatible with standard CAN hardware.

2.3.4. Using Multiple Frames

Some works send multiple standard CAN frames, typically allowing the original full 8-byte payload in one frame and a longer authentication tag in another frame, at the expense of increased throughput on the bus; LeiA [42] allows one 64-bit MAC transmitted in its own frame to authenticate multiple other CAN messages to use less bandwidth compared to providing a MAC per message; vatiCAN [43] sends an additional 8-byte authentication message; Wei et al. [44] also fit the MAC in its own CAN frame; VulCAN [45] uses the CAN ID, data field, and a counter value as inputs to construct a 64-bit MAC; CryptoCAN [46] constructs a 60-bit AES-CMAC using the ID, DLC, 64-bit payload, and 31-bit freshness value as inputs, then proceeds to encrypt the 8-byte payload, 60-bit MAC, and 4 DLC bits in one 128-bit AES block split across two CAN frames in a MAC-then-encrypt approach; and CANcrypt [47] uses a preamble CAN frame preceding the payload CAN frame to send relevant authentication data.

2.3.5. Using CAN-FD

Some proposals make use of CAN-FD's longer maximum payload length which allows larger payload sizes along with full non-truncated authentication tag lengths. Woo et al. [48] fit an AEAD approach to CAN-FD [26], Agrawal et al. [49] use CAN-FD [26] capable ECUs that communicate through a Gateway ECU which manages security, and Montilla [50] use an encrypt-then-MAC approac where SHA256 and AES are used to generate MACs and encrypt payloads placed in CAN-FD frames.

2.3.6. Flexibility in Protocol Use and Advanced Hardware

LiBrA-CAN [51] highlights multiple means of authentication data transmission, deliberately supporting use of multiple CAN messages, CAN-FD [26], or CAN+ [32]. Some works instead present novel hardware mechanisms similar to CAN+, such as the proposal of CAN-MM by Oberti et al. [52] where On-Off Keying digital modulation transmits 64-bits of authentication data along with the standard CAN frame's 64-bit data field. This demonstrates that the means of transmitting authentication data and cryptographic algorithm choice are independent but related by needing to match transmission capcity with the authetication data length.

2.3.7. NIST Finalists

Some existing works propose the use of ASCON and TinyJAMBU; two of the finalists further discussed in Section 3. Wiemer and Zeh [53] propose the use of ASCON option for the CANsec protocol in CAN XL, due to ASCON's efficiency and its nonce-misuse resilience. Rasheed et al. [54] incorporated TinyJAMBU-128 (choosing this cipher due to its small state and block sizes) in a process that also utilises IDs of recently transmitted CAN messages to prevent trivial replay attacks.

3. AEAD Application to the CAN Frame

3.1. Lightweight AEAD Symmetric Ciphers

In 2018, the U.S. National Institute of Standards and Technology (NIST) published a call for submissions [55] for its Lightweight Cryptography (LWC) standardisation process [56]. The design criteria [55] specified a lightweight AEAD cipher with reduced implementation complexity and increased time efficiency, to better suit constrained environments compared to conventional algorithms such as Advanced Encryption Standard (AES). Of 57 submissions, 56 progressed to Round 1, 32 progressed to Round 2, and 10 were selected as finalists [56]. Ultimately, in 2023, ASCON [57] was selected for standardisation [58].

3.2. Rationale for Software Implementations of Ciphers

This paper's aim is the evaluation of lightweight AEAD algorithms for potential use in securing CAN bus transmissions efficiently. We only consider software implementations of cryptographic algorithms to avoid requiring hardware acceleration of the algorithms. There are variants of the NIST finalist cipher families deliberately designed to be highly efficient when implemented in software; our experiments apply these to the CAN protocol.

Our approach inserts authentication data and counter bits alongside the intended payload, within each CAN frame's existing data field, to avoid requiring non-standard CAN hardware. Advanced hardware approaches such as CANAuth's [31] use of CAN+ or the method of bit modulation proposed in CAN-MM [52] are not considered.

3.3. The CAN Bus Security Environment

3.3.1. Security Requirements and Limitations/Restrictions of CAN Bus

Requirements for authenticated cryptography in CAN bus, according to Herrwege et al. [31], include message authentication, replay attack resistance, group keys, and backward compatibility. Herrwege et al. [31] also identify real-time operation, short data length, limited ID space, and broadcast design as fundamental CAN bus constraints affecting authenticated encryption use in CAN bus. Hridoy and Zulkernine [40] highlight bandwidth usage, implementation expense, and protocol compatibility as important aspects regarding CAN cryptography efficiency.

3.3.2. CAN Bus Attack Model

Our attack model we consider as motivation for authenticating CAN frames is built on reasonable assumptions made by prior works. The assumptions adopted include: all traffic can be listened to since CAN is a broadcast network [31,40]; attackers are capable of sending any CAN message into the network (also possible this can be done via compromised ECU [34,36]), which enables spoofing and replay attacks [40]; attackers are able to physically access the CAN bus or any of the nodes [31]; and that compromised ECUs are a possible attack vector [34,36].

3.4. Suitability of NIST Finalists for Securing the CAN Bus Environment

3.4.1. Fulfilment of Security Requirements/Objectives

NIST lightweight AEAD candidates fulfil Message Authentication and Replay Attack Resistance by generating authentication tags that involve per-message nonces. These tags can be truncated to fit CAN frames for Backwards Compatibility and to work with the Short Data Length limitation. The Group Keys requirement pertains to key management, which is out-of-scope for this work. Finalists aim to be time-efficient to address the Real-Time Operation limitation. The Limited Message ID Space and Broadcast Design are out-of-scope, since this must be addressed by the key-management methodology and CAN network design.

3.4.2. Adherence to Limitations of CAN Bus

Backwards Compatibility requires adherence to the existing standard CAN frame format, which involves one CAN frame containing both payload and authentication with reduced lengths [33–35,37–39,59]. Alternatively, payload and authentication distributed across multiple frames [42–46,51]. This is a trade-off between payload and authentication data lengths per message, and the throughput required to send a single message.

3.5. Application of AEAD CAN Frame Layout

3.5.1. The Message Authentication Code

Every CAN frame carries a half-size 32-bit authenticated ciphertext (Auth CT) payload along with its authentication data, similar to Bruton's approach [38]. The least significant byte (LSB) of the counter (Ctr.) used as

the nonce is also included, leaving 3 bytes for the authentication tag; similar to the approach by TOUCAN [39]. Authentication value lengths of 2 bytes [34] and 1 byte [37] are also utilised in existing works.

In the general case, the probability of randomly guessing the correct authentication tag is $1/2^{tag\_length}$. Thus a 24-bit tag has a blind guess probability of $1/16777216$, which is 0.000006%. Utilising nonces makes the valid tag for a given CAN frame ID/payload a shifting target, so the attacker is effectively continually blind guessing; thus denying the use of brute-force guessing of the correct tag for the given key and target CAN frame content.

Note that AES-GCM has a known weakness: the tag's security level is only $tag\_length - 2^{message\_blocks}$ [60]. In our experiments, we test payloads of length less than one AES block, thus the security level reduction is less than one bit. There is a planned revision [61] to NIST Special Publication 800-38D [62] with a proposal to remove support for tags with lengths less than 96 bits, such as this use case. In any case, we do not recommend use of Advanced Encryption Standard in Galois/Counter Mode (AES-GCM) for our target 8-bit microcontroller units (MCUs) since they lack AES-GCM hardware acceleration, and creating an optimised software of it is non-trivial if possible [63]. AES-GCM is used in our testing purely to provide a benchmarking reference point by which the lightweight ciphers can be compared to when assessing the benchmarking results.

3.5.2. The Overall Application of AEAD to the CAN Frame

Full 64-bit payloads only incur a requirement of two CAN frames, and 32-bit or less payloads only require one frame. Authentication is straightforward as each frame holds its own authentication data. We acknowledge this does not adhere to TinyJAMBU's [64] security claim's assumption of an 8-byte minimum message length; but we retain all finalists for completeness of testing results and leave security-proof issues of specific finalists in very short messages to future work.

The ID, DLC value, and RTR value are included as associated data (AD), to protect them from modification/spoofing. Encrypting CAN IDs would prevent the CSMA/CD process from working in CAN controller hardware as cryptography is handled by the microcontroller. Table 1 shows the layout of this application to relevant CAN frame fields.

Table 1. Our CAN bus frame AEAD application. Only relevant CAN fields are shown.

Application	**Frame Layout**						
	ID	**Control Bits**			**Data Field Bytes**		
		RTR	**IDE**	**DLC**	**0–3**	**4–6**	**7**
Standard	Unauthenticated ID				Unauthenticated Payload		
Our AEAD	ID as Associated Data				Payload	Authentication Tag	Counter's Least-Significant Byte

3.5.3. The Counter and Nonce Values

In our experiments, we used a counter to obtain the nonce values, to avoid requiring time synchronisation. Just the least significant byte (LSB) of the nonce value was included in the CAN message data field for synchronisation purposes, as shown in Table 2. This is similar to the approaches taken by other researchers [35,36] to allow nodes to adjust their counter values correspondingly. This AEAD application could be performed in the experiments without including the LSB (e.g., 4-byte tag instead); it still works provided the nodes using that key reliably increment their counters for every successfully validated frame sent in the network, or where an alternate secure means of nonce calibration/re-synchronisation is utilised.

Table 2. Incorporation of the least-significant counter byte. CT—Ciphertext.

Value/Frame	**Layout of Value/Frame**		
Counter Value	Most-Significant Bytes of Counter		LSB of Ctr.
CAN Frame	4B Auth. CT Payload	3B Tag	LSB of Ctr.

The same nonce should not be repeated for a given key. If a counter has progressed through all possible values for the nonce, then the key must be replaced with a fresh key value. Deployments of AEAD in CAN networks that make use of multiple keys (for example, multiple distinct groups of nodes each with their own group key) can use a separate nonce for each key. When a counter associated with a particular key has exhausted all possible nonce values; that specific key is replaced with a new key. Key management activities, while important for security, are out-of-scope for this paper, as the focus is on determining the time overhead and practical application of adding

AEAD functionality to CAN frames. In our experiments, we measured times for transmission between just one sender and receiver and therefore used one counter.

4. Hardware Testing of Our AEAD Application to CAN Bus Using NIST Finalist Ciphers

We compared the time-efficiency overhead of applying the lightweight AEAD finalist candidates to CAN bus. A testing procedure was performed, where A direct CAN connection was formed between two Arduino boards acting as sender and receiver via MCP2515 [65] modules. To allow the receiver to signal to the sender the success or failure of each message decryption, two jumper wires are directly connected across the devices.

Although NIST already have benchmarking data available [66] for finalist cipher performance on Arduino boards using Weatherly's implementation (and other implementations); we use Weatherly's finalist cipher implementations [67] along with Mistry's Arduino CAN library [68] to obtain benchmarking results that pertain specifically to our application of AEAD to the CAN frames.

The ATMEGA4809 [69] and ATmega328P [70], respectively used in the Arduino Nano Every [71] and Arduino Uno R3 [72], have automotive-grade variants (shown in Table 3). Our benchmarking results should be representative of cryptographic delays that two ECUs using a similar 8-bit AVR microprocessors would experience if securing CAN frames with our approach.

Table 3. Main attributes of the Arduino boards. Automotive standard variants of Atmel MCUs have the VAO suffix in the part number. Not Recommended for New Designs (NRND) is a status assigned to MCUs by Microchip Technology meaning while the MCU is still produced, is not recommended for use in new designs.

Attribute	Arduino Board		
	Mega [73]	**Nano Every [71]**	**Uno [72]**
ATmega CPU	2560 [74]	4809 [69]	328P [70]
MCU Automotive Grade?	No	VAO Variants Exist	Yes
MCU Production Status?	Yes	Yes	NRND
Clock (MHz)/MIPS	16 / 16	20 / 20	16 / 16
Flash Memory (KB)	256	48	32

4.1. Hardware Utilised in Our Experiments

We ran our experiment on the Mega 2560 Rev3 [73], the Nano Every [71], and the Uno R3 [72]. Benchmarking on these three different Arduino boards allowed us to obtain time-efficiency data for various 8-bit AVR MCUs, representative of what may be used in low-performance automotive equipment. Available memory and CPU capabilities differ between the MCUs on the Arduino models, these differences are shown in Table 3. Notably the ATmega4809 [69] on the Nano Every [71], a MCU still in production as of 2026, has 20 MIPS while the others have 16 MIPS, and is also recommended by Microchip for automotive applications [75].

For each model of Arduino board, two boards are connected together as sender and receiver. Each pair of boards used MCP2515 CAN modules, connected to their respective boards via SPI (Serial Peripheral Interface), and were directly connected to each other via the CAN-H and CAN-L pins to form the CAN connection. Between the sender and receiver boards, two jumper wires are directly connected, allowing the receiver to signal the success or failure of decryption/verification to the sender, which waits for this signal in the process of determining overall transmission time (from sender starting encryption to receiver ending decryption).

4.2. Software Modified for Our Experiments

We modified the existing in-memory testing procedure performed by Weatherly's benchmarking code for the NIST LWC finalists [67] to include benchmarking of 4-byte messages in addition to 128-byte and 16-byte messages. We also incorporated our CAN testing procedure to run after the in-memory benchmarking, utilising Mistry's Arduino CAN library [68] (version 0.3.1). Although enabling use of shortened tag lengths required changing a defined tag length value the cipher code references, the cipher implementations are otherwise unmodified.

Regarding Weatherly's benchmarking, NIST's Status Report on the Final Round of the NIST Lightweight Cryptography Standardization Process [76] states that "Code in this effort was optimized for size, not speed". In the code's documentation discussing performance on 32-bit platforms [77], Weatherly states "The code was optimised for size rather than speed, which is the default optimisation option for the Arduino IDE". Due to Weatherly's benchmarking focusing of size efficiency rather than speed efficiency, we do not change the Arduino IDE's default use of the -oS compiler flag that optimises code size rather than speed. This ensures our results

are as directly comparable as possible, and are applicable to automotive ECUs that try to minimise the code size overhead of including authentication. Use of a speed-optimised compilation may potentially improve on our benchmarking results, hence suitability of finalists in our results should be likely to mean suitability when using any code-optimisation compiler flag option.

4.3. The Existing Benchmarking Procedure

Upon being run, the program will first perform the existing message benchmarking including both the original 128-byte and 16-byte message benchmarking as well as our added 4-byte message benchmarking. We compare the resulting factor values from our runs of the 128-byte and 16-byte benchmarking with the values submitted by Weatherly to NIST [76], to verify that our use of his implementation does not problematically deviate. Estimates of CAN encryption delay (without associated data) can be provided by our 4-byte benchmarking, since multiplication of the us/byte rate by 4 obtains an estimate of encrypting our 4-byte payload length we use in CAN testing. We left the sanity checking procedure already present in Weatherly's code [67] as is. ISAP sanity checks no longer passed when we used truncated tags, but the tag verifications still succeeded in our testing, suggesting the sanity check failures are an artefact of our modification of the code to use truncated tags.

4.4. CAN Testing

The 8-byte CAN data field of each sent CAN message carried the payload which was the message 'Test' (four single-byte chars) encrypted, a truncated 3-byte authentication tag, and the value of the least significant byte of the counter. The nonce used for each cipher was an counter of bit length equal cipher's maximum nonce size, with each message send/receipt respectively incrementing the sender's/receiver's local copy of the counter. The least significant byte value of the counter in the message allows for a node that has an out-of-sync counter due to missed messages to detect the lack of synchronisation and update their counter to match (up to 254 missed messages can be handled with just the least significant byte of the counter).

Note that management of nonces in CAN networks of realistic complexity is out-of-scope, our use of nonces is just to facilitate our CAN latency testing procedure with proper nonce use. The key used in the original software for benchmarking [67] was also utilised for this testing, since key management is out-of-scope.

The receiver will attempt to decrypt and verify received messages. CAN messages are sent with varying combinations of different settings/factors:

- CAN Rate: Tested 1000 Kbps through to 40 Kbps. Did not test 20 Kbps and less since these rates failed to transmit unencrypted messages; these rates were assumed to not work with the CAN modules. Testing these rates was not necessary as it is not typical for automotive networks to employ such low bitrates.
- Choice of Cipher: Each cipher was tested with the various combinations of other parameter values. Also tested unencrypted messages used to obtain reference CAN transmission times and check that CAN is working.
- CAN ID Type: Both types of CAN ID messages were tested, the associated data length is different for each.
- CAN ID as AD: Tested both with CAN ID, DLC, and RTR values included as associated data and without.

To obtain robust results, times are averaged across 200 runs (CAN messages) for each set of parameters. The sender records encryption time for each CAN message sent, the receiver likewise records decryption time for each received CAN message. Upon completing the decryption/verification process, the receiver signals completion via a high signal on the respective board-to-board connected jumper wire for success/failure. This allows the sender to record the overall transmission time and the success/failure state. PuTTY was used for saving results to a text file from which the data was further formatted/processed, since the Arduino boards output the results via serial. Versions 0.76, 0.77, and/or 0.78 were used during the original testing as these were the latest versions available in that timeframe. Version 0.83 of PuTTY was used in the more recent authentication-only testing. Note that our Arduino serial logging was not affected by any version differences.

4.5. Authentication-Only Testing

A further alteration of our CAN testing procedure in the software was made since publication of our earlier testing [20] to benchmark authentication-only time-efficiency. This modified procedure provides the payload to the cipher as more associated data rather than plaintext to be encrypted, thus skipping the encryption process. This approach is mentioned in Weatherly's technical report [78] stating that "AEAD modes are by their nature keyed message authenticators already, generating a tag to authenticate the associated data and plaintext. A simple keyed authentication mode can be constructed by setting the plaintext to empty and authenticating the message as associated data under a key and nonce."

This is relevant as there are many messages that may not require confidentiality. The flexibility to encrypt messages that do benefit from confidentiality and systems where implementing alternative authentication (e.g., HMAC) is a development cost that can be avoided if AEAD is already implemented, are both use cases for using the AEAD associated-data processing as an authenticator.

In this version all CAN rates, both ID types, and all AEAD ciphers were tested as before, only this time, rather than testing authenticated encryption either with or without including the ID as associated data, this versions always includes the ID as associated data while processing the plaintext payload as associated data as well, rather than running authenticated encryption on the payload.

4.5.1. The Associated Data

In all testing the standard ID is included as a 16-bit value (containing the 11 ID bits) or the extended ID instead as a 32-bit value (containing the 29 ID bits), alongside the DLC included in its own byte (containing the 4-bit number) and the RTR in its own byte (least significant bit denoting 0 or 1). This results in 4 bytes of associated data for the standard ID case and 6 bytes for the extended ID case, as shown in Table 4. While bit-packing could be employed to reduce the byte length, we kept the longer associated data lengths to assume a situation where entire variables are used in a straight forward manner. More comprehensive testing of algorithm performance at various combinations of AD and plaintext input lengths is potential future work.

Table 4. How CAN values are inserted into associated data. B — Bytes.

	Associated Data Bytes									
	Byte 9	**Byte 8**	**Byte 7**	**Byte 6**	**Byte 5**	**Byte 4**	**Byte 3**	**Byte 2**	**Byte 1**	**Byte 0**
Extended ID	Bytes 9 to 6 N/A				29 ID Bits (4 Bytes)				DLC	RTR
Standard ID	Bytes 9 to 4 N/A						11 ID Bits (2B)		DLC	RTR
Auth-Only Ext.	29 ID Bits (4 Bytes)				DLC	RTR	4-Byte Payload			
Auth-Only Std.	Bytes 9 & 8 N/A		11 ID Bits (2B)		DLC	RTR	4-Byte Payload			

The authentication-only testing adds 4 bytes to the associated data lengths for the respective ID type cases, as shown in Table 5, since we are processing the 4-byte payload as associated data rather than plaintext to authenticate it without encrypting it.

Table 5. Payload lengths tested.

Testing Type	**AD Lengths**		**Plaintext Length**
	Extended ID	**Standard ID**	
Original Testing - ID, DLC, and RTR as AD	6	4	4
Original Testing - Without AD	0	0	4
Authentication-Only Testing	10	8	0

This will allow comparison of the results to see what the overhead of the associated data is when using authenticated encryption and to see what the effect on performance is if instead employing an associated-data approach to only authenticating a message.

5. Results

We discuss results for the in-memory benchmarking and the CAN testing.

In-memory benchmarking tests processing of 128-byte, 16-byte, and 4-byte messages using Weatherly's existing benchmarking procedure. This is used so we can generate factors for 4-byte payloads to see how the relative performance of the finalists vs AES-GCM holds for this very small payload size.

The CAN results, generated as per the previously discussed CAN testing procedure, are displayed for 500 Kbps CAN rate with extended IDs for the three Arduino devices. We also, just on the Mega 2560, compare to two variations, a ciphertext-only variation where we do not authenticate the ID/DLC/RTE, and an authentication-only variation where we do authenticate the ID/DLC/RTR but do not encrypt the payload (authenticating the payload as associated data instead).

The memory benchmarking allows for protocol-independent inspection of the effect of the short payload size, while the CAN testing allows for comparison of efficiency when applying the AEAD cipher to the CAN frame in different ways (regular AEAD, ciphertext-only, and authentication-only).

5.1. AEAD Standard Benchmarking

5.1.1. 128-Byte and 16-Byte Memory Testing

Figure A1 in Appendix A shows that in the standard 128-byte and 16-byte tests, most finalists are more time-efficient than AES-GCM. Conversely, the finalists less time-efficient than AES-GCM are: hardware-optimised variants of Elephant, Grain-128AEAD, all ISAP variants, and Romulus-T.

Note that Weatherly's ChaCha20-Poly1305 implementation is a "32-bit non-vectorised implementation" intended just as a reference for comparative benchmarking on various MCUs with differing capabilities [77]; this implementation is not optimal for the 8-bit AVR MCUs we test with. Thus our results are not necessarily representative of this algorithm's potential performance on these MCUs and are only used as a reference point in cipher ranking. An algorithm can be fast but a software implementation that is not optimised for the processor it is being run on will result in reduced time-efficiency, arising from the implementation mismatch with the processing hardware, rather than from the algorithm design. Evaluation/development of an 8-bit AVR optimised ChaCha20-Poly1305 implementation MCUs is out-of-scope, since our focus is the finalists vs AES-GCM.

5.1.2. Factors Based on the ChaCha20-Poly1305 Implementation

Another use of this ChaCha20-Poly1305 implementation is the production of factors: how many times faster/slower a finalist (or AES-GCM) is compared to this ChaCha20-Poly1305 implementation. Weatherly's benchmarking code produces overall speed factors present in Figure A1 in Appendix A made by adding together a total of the four average times (encrypting 128 bytes, decrypting 128 bytes, encrypting 16 bytes, and decrypting 16 bytes) for each finalist (and AES-GCM), which is then divided by a likewise total for ChaCha20-Poly1305 to obtain a time factor. We also obtained the inverse speed factor by performing a basic vice-versa division in our modified version of the code, this being also present in Figure A1 in Appendix A. Note that this is different from the individual factors discussed in Section 6.3.

5.1.3. 4-Byte Memory Testing

Figure A2 in Appendix B shows less efficiency across the board in the 4-byte test results, suggesting much higher significance of initialisation time cost per encryption/decryption at this message length. Likely due to decreasing message length resulting in the initialisation processing becoming a greater proportion of the computation time spent on encryption/decryption of the message. The bytes-per-second performance of Schwaemm, TinyJAMBU, and ASCON are particularly efficient at these short message sizes.

5.1.4. Estimated CAN Delays from 4-Byte Testing

To obtain estimated CAN delays in Figure A2, the average encryption/decryption rate values are multiplied by 4 (since a 4-byte payload is used). Although reasonably close, these estimates were slightly smaller than our CAN testing results, since the CAN testing delay was slightly increased by the processing of certain CAN frame parameters as Associated Data. An additional rationale was to test the CAN sized payload within the same benchmarking procedure already present in Weatherly's code, to produce individual factors (discussed in Section 6.3) for the 4-byte payloads, allowing comparison with the 128-byte and 16-byte factors both from our testing and from the NIST LWC final round report [76] in Section 6.3.

5.2. AEAD CAN Testing

In Figure 3, we present results for 500 Kbps CAN rate, with extended CAN ID and use of associated data for authentication of ID, RTR, and IDE. This result subset reflects the most common automotive CAN speed (according to [79] obtained from [80]) and that some CAN networks require extended IDs to have sufficient ID space. The associated data as assembled in memory for processing by the AEAD cipher is 6 bytes consisting of a 1-byte value indicating the RTR value, and a 1-byte value in memory containing the DLC value, along with 4 bytes containing the 29-bit ID.

One notable observation relates to the performance of the T and M variants of Romulus. In our experiments, these variants crashed on the Ardunio Nano when attempting to perform CAN testing. Note in Figure 3, there is a blank space for Romulus-T in the Nano Every column (Romulus-M is absent as it was slower that the reference implementation of ChaCha20-Poly1305). However, testing was successful for the implementations on the Mega and Uno boards. Note that the authors of the Romulus cipher state that tags should not be truncated for these variants. This may be an artefact of our truncation of tags to three bytes in length, which may also be why

Romulus-T decryption failed and Romulus-M exhibited extremely poor time-efficiency on the Mega and Uno in our experimentation.

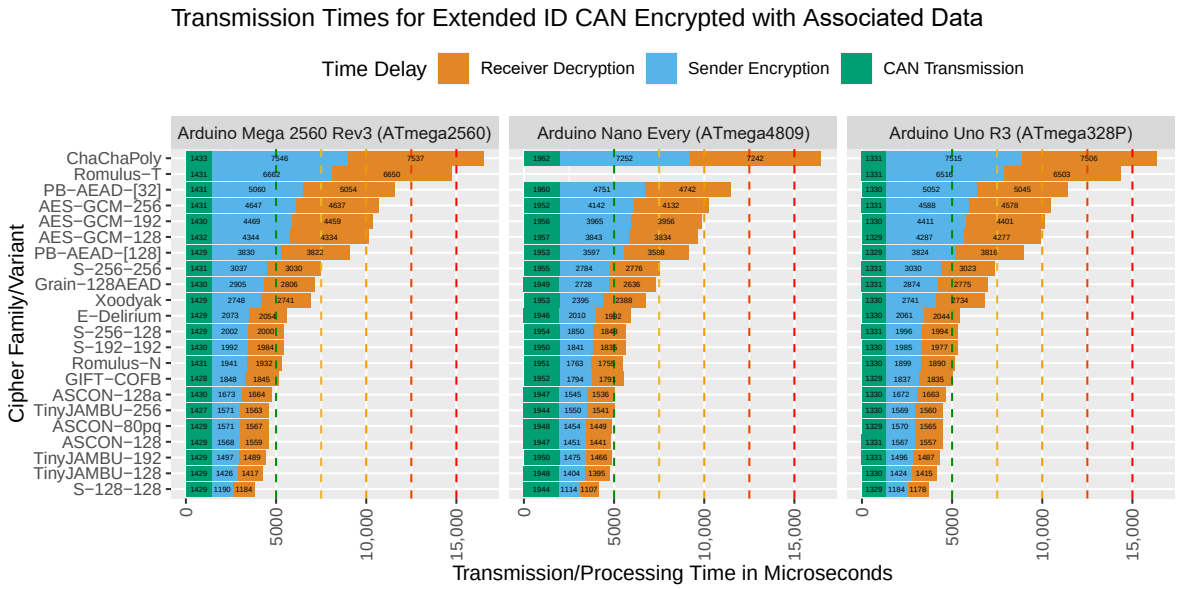

Figure 3. Results from the CAN testing. Mean times for encryption, decryption and CAN transmission stacked to visualise total latency. To maintain reasonable bar chart scale, ciphers slower than the reference implementation of ChaCha20-Poly1305 omitted. Note that the overall delay for Very Advantageous ciphers is less than 5 ms.

5.2.1. Measuring CAN Latency

The sender experimentally measures the encryption time and overall time (assisted by the receiver's signalling), while the receiver measures the decryption time. The non-cryptography related CAN bus transmission timing is inferred by subtracting the measured encryption and decryption times from the measured overall delay. Figure 3 shows the average across the 200 runs for each cipher of the measured encryption and decryption times along with the inferred non-cryptography transmission time stacked together to represent a typical overall delay for that cipher.

5.2.2. Statistical Variation of Results

Tables of statistical data for our CAN testing results are presented in Appendix C, Appendix D, and Appendix E for the Mega, Nano, and Uno, respectively. For each cipher family and variant, the table includes the mean, minimum, maximum, range, standard deviation, and coefficient of variation of the measured time across the 200 trials performed in each case: for the measured encryption, decryption, and overall time delay, as well as the calculated cryptographic delay and send time. These table entries clearly show the coefficient of variation (CV) is less than 0.5% across most values, with the send time being the exception at 0.5% to 1%. This is due to the non-crypto send time being calculated rather than directly measured, resulting in a higher CV from the compounding variability that results. The range is also shown to be less than 100 microseconds for all values. We note that the range for measured encryption and decryption delays are all 12 microseconds or less, sufficiently consistent for the purpose of ranking the ciphers this variability is much less than the typical difference between cipher variants (except for certain variants of ASCON and TinyJAMBU with near-identical results and are thus given the same rank).

5.3. Authentication-Only CAN Testing

Our results for authentication-only testing show most cipher families are more efficient at processing the payload along with the ID as associated data then running full authenticated encryption. The most efficient AEAD option; Sparkle's 128-128 variant, shows an approximately 33% improvement when used to process all message content as associated data for authentication without encryption.

However, this is not true for all cipher families, with some cipher families showing a disadvantage. There are three notable examples of this: ASCON-80pq, Grain-128AEAD, and ISAP-K-128A. These are anomalous results as intuitively, performing AD rather than AEAD should be faster due to omiting the encryption. This performance degradation may be due to cipher design aspects, such as performing extra work per byte of processed AD as opposed to encrypted plaintext. While interesting and warranting further investigation, this is out-of-scope of this work.

5.3.1. Statistical Variation of Results

Statistical variation for both authentication-only results (ID and payload all processed as associated data) and ciphertext-only results (ID and payload not processed as associated data) are respectively in Appendix F and Appendix G.

6. Discussion

6.1. Our Ranking of the Finalists for CAN Bus

The results are used to rank the finalists with respect to time-efficiency when used as software implementations for very short payloads in our CAN testing. Our evaluation table (Table 6) shows results in the Figure 3 dataset (our main data set with the most common CAN speed and choice of payload with inclusion of ID as AD).

Very Advantageous ciphers at least rival if not surpass ASCON. ASCON and TinyJAMBU have nonce-misuse resilience and high time-efficiency, as well as larger key variants while also being respectively used/recommended in two existing works [54] and CAN XL [53]. Schwaemm is particularly noteworthy for its 128-128 variant significantly outperforming ASCON and TinyJAMBU, so if nonce-misuse and longer key length are not required, this Schwaemm variant is the best time-efficiency alternative.

Table 6. Ranking of finalists' AE results (with AD), ordered by Mega/Uno results. Nano ranking positions differ slightly, but are in correct ranking categories since category differences are greater than device differences. The M and T variants of Romulus-M and Romulus-T do not support tag truncation and crashed in CAN testing on the Nano.

Category	Finalist	Mega 2560		Nano Every		Uno R3	
		Encrypt	Decrypt	Encrypt	Decrypt	Encrypt	Decrypt
Very Adv. faster or rivals ASCON	Sparkle-128-128	1184	1190	1107	1114	1178	1184
	TinyJAMBU-128	1417	1426	1395	1404	1415	1424
	TinyJAMBU-192	1489	1497	1466	1475	1487	1496
	ASCON-128	1559	1568	1441	1451	1557	1567
	TinyJAMBU-256	1563	1571	1541	1550	1560	1569
	ASCON-80pq	1567	1571	1449	1454	1565	1570
	ASCON-128a	1664	1673	1536	1545	1663	1672
Mod. Adv. <∼2 ms	GIFT-COFB	1845	1848	1791	1794	1835	1837
	Romulus-N	1932	1941	1755	1763	1890	1899
	Sparkle-192-192	1984	1992	1835	1841	1977	1985
	Sparkle-256-128	2000	2002	1848	1850	1994	1996
	Elephant-Delirium	2054	2073	1992	2010	2044	2061
Mar. Adv. <∼3 ms	Xoodyak	2741	2748	2388	2395	2734	2741
	Grain-128AEAD	2806	2905	2636	2728	2775	2874
	Romulus-M	2865	2880	Crashed	Crashed	2734	2741
	Sparkle-256-256	3030	3037	2776	2784	3023	3030
Bar. Adv.	PB-AEAD[128]	3822	3830	3588	3597	3816	3824
AES-GCM	AES-GCM-128	4334	4334	3834	3843	4277	4287
	AES-GCM-192	4459	4469	3956	3965	4401	4411
	AES-GCM-256	4637	4647	4132	4142	4578	4588
Disadv. >AES	PB-AEAD[32]	5054	5060	4742	4751	5045	5052
	Romulus-T	6650	6662	Crashed	Crashed	6503	6516
Reference	ChaCha20-Poly1305	7537	7546	7242	7252	7506	7515
V. Disadv. >Ref.	Elephant-Dumbo	8977	8993	8949	8966	8972	8988
	Elephant-Jumbo	11,440	11,458	11,414	11,430	11,435	11,452
	ISAP-K-128A	14,126	14,135	12,847	12,856	14,104	14,113
	ISAP-A-128A	20,647	20,657	18,418	18,427	20,592	20,601
	ISAP-K-128	93,058	93,066	85,777	85,786	93,025	93,034
	ISAP-A-128	141,515	141,524	126,457	126,467	141,233	141,242

Moderately Advantageous ciphers are less than or around 2 ms per operation, despite not rivalling ASCON. This still provides approx. 2 ms advantage over AES-GCM. Marginally Advantageous ciphers range from Xoodyak's approx. 2.7 ms on Mega/Uno to Schwaemm-256-256's approx. 3 ms on Mega/Uno, still retaining at least 1ms advantage over AES-GCM. PHOTON-Beetle[128] is Barely Advantageous, with only about a half-a-millisecond advantage at best. PHOTON-Beetle-AEAD-32 and Romulus-T are Disadvantageous since they are less time-efficient than AES-GCM.

Very Disadvantageous ciphers are less time-efficient than Weatherly's reference implementation of ChaCha20-Poly1305, despite it lacking optimisation for the 8-bit AVR architecture we ran our experiments on. Elephant hardware-optimised variants were inefficient as they are not intended to operate as software implementations.

ISAP's cipher design has a focus on protecting against various implementation attacks [76], the comparative time-inefficiency may be the resultant deliberate trade-off made for enhanced attack-resilience. Romulus-T nonce-misuse resilience may contribute to its relative time-inefficiency.

6.2. Authentication Only Results and Comparison to AEAD for ASCON and Sparkle

In Figure 4, we can see that Sparkle has a significant improvement, particularly the 128-128 variant at about a third faster. Notably, ASCON, the standardised finalist, has a minor disadvantage. While purpose-made mechanisms intended for implementing pure-authentication (e.g., HMAC, KMAC (KECCAK Message Authentication Code)) , etc.) do exist, for the use cases of leveraging an existing AEAD primitive as is, this further distinguishes Sparkle and ASCON. If a set of CAN messages require authentication but only a subset of those require confidentiality, one AEAD algorithm can use the AD-only and AEAD approaches accordingly on each frame to save implementation complexity and program space. In this example, Sparkle would provide improved time-efficiency for the authentication of non-confidential frames.

Comparison of Crypto Delays on Mega2560, 500Kbps CAN, Extended IDs

Cipher Family	Variant	Encrypt with ID as AD — Mean of Enc + Dec	Encrypt Without ID as AD — Mean of Enc + Dec	Relative Overhead of Including ID as AD	Auth-Only (ID + Data as AD) — Mean of Gen + Verify	Comparison to Authenticated Encryption	Auth-Only Relatively Advantegous to AE?
Sparkle	128-128	2373	1587	33.1226%	1597	-32.7012%	Moderate Advantage
TinyJAMBU	128	2843	2319	18.4312%	2693	-5.2761%	Minor Advantage
	192	2986	2464	17.4816%	2790	-6.5640%	Minor Advantage
ASCON	128	3126	2506	19.8337%	3331	6.5579%	Minor Disadvantage
TinyJAMBU	256	3134	2611	16.6879%	2891	-7.7537%	Minor Advantage
ASCON	80pq	3137	2520	19.6685%	8188	161.0137%	Extreme Disadvantage
	128a	3337	2525	24.3332%	3716	11.3575%	Minor Disadvantage
GIFT-COFB	N/A	3692	3690	0.0542%	5520	49.5125%	Major Disadvantage
Romulus	N	3872	3849	0.5940%	3894	0.5682%	Negligible Difference
Sparkle	192-192	3975	2650	33.3333%	2677	-32.6541%	Moderate Advantage
	256-128	4001	2676	33.1167%	2673	-33.1917%	Moderate Advantage
Elephant	Delirium	4126	4125	0.0242%	4115	-0.2666%	Negligible Difference
Xoodyak	N/A	5488	5473	0.2733%	5482	-0.1093%	Negligible Difference
Grain-128AEAD	N/A	5711	4328	24.2164%	186111	3158.8163%	Extreme Disadvantage
Romulus	M	5745	5730	0.2611%	3838	-33.1941%	Moderate Advantage
Sparkle	256-256	6066	4046	33.3004%	4067	-32.9542%	Moderate Advantage
PHOTON-Beetle-AEAD	[128]	7651	5131	32.9369%	5085	-33.5381%	Moderate Advantage
AES-GCM	128	8678	6309	27.2989%	6077	-29.9723%	Moderate Advantage
	192	8928	6559	26.5345%	6354	-28.8306%	Moderate Advantage
	256	9284	6915	25.5170%	3706	-60.0819%	Major Advantage
PHOTON-Beetle-AEAD	[32]	10113	5103	49.5402%	10064	-0.4845%	Negligible Difference
Romulus	T	13312	13256	0.4207%	9552	-28.2452%	Major Advantage
ChaChaPoly	N/A	15082	14184	5.9541%	5906	-60.8407%	Major Advantage
Elephant	Dumbo	17969	17968	0.0056%	22386	24.5812%	Moderate Disadvantage
	Jumbo	22897	22895	0.0087%	2704	-88.1906%	Major Advantage
ISAP	A-128A	28261	28247	0.0495%	28821	1.9815%	Negligible Difference
	K-128A	41304	41289	0.0363%	281285	581.0115%	Extreme Disadvantage
	A-128	186123	186107	0.0086%	40123	-78.4428%	Major Advantage
	K-128	283038	283024	0.0049%	28541	-89.9162%	Major Advantage

Figure 4. Results from the authenticated encryption CAN testing with ID as associated data as shown in Figure 3 along side AE without ID as AD and Authentication-Only results for the Mega running 500 Kbps CAN. This table also shows results for Elephant hardware variants and ISAP not present in Figure 3. Enc—Encryption; Dec—Decryption.

6.3. Comparison with Weatherly's Testing

In NIST's status report for the LWC final round [76], it is stated that Weatherly's benchmarking results indicate that "ASCON, GIFT-COFB, PHOTON-Beetle, Romulus, Schwaemm, TinyJAMBU, and Xoodyak were faster than ChaChaPoly". Our results are consistent with this since in each of these cipher families, at least one variant shows this advantage when processing our very small 4-byte CAN bus payloads. Since Hridoy and Zulkernine [40] use an optimised implementation of ChaCha20-Poly1305 [41] in their testing, this is not consistent with their results.

6.3.1. Individual Factors

Table 7 shows individual factor results from our testing and from Weatherly's testing presented in NIST's report [76]. Rather than adding together average encryption and decryption times for the 128-byte and 16-byte payloads and then dividing that, each combination of payload length and operation has its average individually divided. For example, the average encryption time for 128-byte payloads for each given finalist (and AES-GCM) is

divided by the corresponding average time for the reference implementation of ChaCha20-Poly1305, to get the individual speed factors for 128-byte encryption benchmarking. This is likewise done for both encryption and decryption of 128-byte, 16-byte, and 4-byte payload lengths in our testing.

Our obtained individual factors for 128-byte and 16-byte testing are reasonably similar to Weatherly's obtained factors. Furthermore, our 4-byte testing obtained new factors showing some finalists have significantly better factor improvement (against the reference implementation of ChaCha20-Poly1305) when processing 4-byte messages compared to 16-byte messages, while other finalists and AES-GCM have insignificant variation in this respect. Hence, some finalists obtain greater time-efficiency advantage when dealing with very small payloads, such as those in our CAN AEAD approach.

Table 7. Speed factors from memory testing compared against Weatherly's factors in NIST's LWC final round status report [76]. We determined factors for all variants and AES-GCM, not just the primary variants. A diverging colour map is used. The midpoint is a factor of 1 and uses Green-to-White-to-Orange for our factors and Blue-to-White-to-Red for Weatherly's factors. Green/Blue are factors greater than 1; Orange/Red are factors less than 1.

Cipher	Speed Factors									
	Mega 2560 Rev3—Sender						Weatherly's AVR Factors in NIST Report			
	4-Byte		16-Byte		128-Byte		16-Byte		128-Byte	
	Encrypt	Decrypt	Encrypt	Decrypt	Encrypt	Decrypt	Encrypt	Decrypt	Encrypt	Decrypt
AES-GCM-128	2.25	2.25	2.24	2.23	1.02	1.02				
AES-GCM-192	2.17	2.16	2.16	2.15	0.99	0.99				
AES-GCM-256	2.05	2.05	2.04	2.04	0.96	0.96				
ASCON-128	5.69	5.66	3.84	3.8	2.18	2.16	3.83	3.79	2.18	2.16
ASCON-128a	5.65	5.61	4.28	4.24	2.95	2.9				
ASCON-80pq	5.66	5.64	3.82	3.8	2.18	2.16				
Elephant-Delirium	3.46	3.43	3.44	3.4	1.72	1.7				
Elephant-Dumbo	0.79	0.79	0.79	0.79	0.37	0.37	0.79	0.79	0.37	0.37
Elephant-Jumbo	0.62	0.62	0.62	0.62	0.33	0.33				
GIFT-COFB	3.85	3.85	3.92	3.88	2.62	2.54	3.88	3.7	2.61	2.41
Grain-128AEAD	3.36	3.21	1.95	1.77	0.82	0.7	1.96	1.78	0.83	0.7
ISAP-A-128	0.08	0.08	0.07	0.07	0.12	0.12				
ISAP-A-128A	0.5	0.5	0.45	0.46	0.47	0.48				
ISAP-K-128	0.05	0.05	0.05	0.05	0.08	0.09				
ISAP-K-128A	0.34	0.34	0.34	0.34	0.4	0.41	0.31	0.32	0.37	0.38
PB-AEAD[128]	2.77	2.76	2.75	2.74	1.14	1.14	2.82	2.76	1.16	1.15
PB-AEAD[32]	2.78	2.78	1.12	1.12	0.32	0.32				
Romulus-M	2.48	2.47	2.47	2.46	1.08	1.08				
Romulus-N	3.7	3.68	3.66	3.64	1.56	1.55	3.66	3.62	1.56	1.55
Romulus-T	1.07	1.07	0.94	0.94	0.48	0.48				
Sparkle-128-128	9.01	8.96	8.73	8.63	4.43	4.38	5.07	4.89	4.45	4
Sparkle-192-192	5.38	5.35	5.27	5.22	3.4	3.54				
Sparkle-256-128	5.31	5.31	5.17	5.18	4.45	4.42				
Sparkle-256-256	3.51	3.51	3.47	3.46	3.03	3.01				
TinyJAMBU-128	6.15	6.12	4.03	3.99	1.79	1.77	4.02	3.99	1.79	1.78
TinyJAMBU-192	5.79	5.76	3.73	3.7	1.61	1.61				
TinyJAMBU-256	5.46	5.43	3.46	3.44	1.47	1.46				
Xoodyak	2.6	2.59	2.58	2.57	1.81	1.8	2.58	2.57	1.81	1.8

6.4. Comparison with NIST's Testing Results

According to testing performed by NIST [76]; ASCON, GIFT-COFB, PHOTON-Beetle, Romulus, Schwaemm, TinyJAMBU, and Xoodyak had better time-efficiency than AES-GCM on ATmega MCUs. Perhaps due to overtaking ChaCha20-Poly1305 in terms of efficiency as message sizes got shorter, Grain-128AEAD was better in some inputs but not others [76]. In our CAN testing, which processed a 4-byte payload and included the ID, DLC, and RTR values as associated data; our results aligned with NIST's results [76] in that all these ciphers, except for PHOTON-Beetle-AEAD[32], were more time-efficient than AES-GCM.

6.5. Comparison with Existing CAN Works

Groll and Ruland [29] benchmarked AES on a 32 Mhz ARM7 processor, taking 1.5–2 ms. The Very and Moderately Advantageous lightweight ciphers achieved similar performance on 16 Mhz AVR processors.

Computing a vatiCAN [43] HMAC took 2.95 ms at the same clock speed of 16 Mhz. Various other works also include efficiency results [34,38,39,45,46,48,59], but utilise hardware significantly more capable than the MCUs on the Arduinos we utilised in terms of processing capabilities/speed, making direct comparison irrelevant.

Wu et al. [37] state that according to theoretical calculations regarding their watermarking method "the authentication process can be completed in about 1 ms". Furthermore, their time-efficiency results for their watermark decoding process in their experiments using the STM32F103ZET6 and STM32F103VET6 development boards with 72 MHz processors showed 8 bytes decoded in less than 1 ms through to 64 bytes decoded in 5.66 ms. The use of more capable MCUs than our Arduino boards mean these results are not directly comparable however. Furthermore, the watermarking method only provides authentication, and the protection against replay attacks suffers from detection accuracy degradation the more frequently replay attack CAN messages are injected.

Hridoy and Zulkernine [40] tested with three Arduino boards: the Uno, Zero, and Due. Their test results for AES-GCM on the Uno were similar to ours at approximately 4–4.5 ms. Direct comparison is irrelevant for the Zero and Due which use more capable 32-bit ARM Cortex MCUs. They used an implementation of ChaCha20-Poly1305 by De Santis et al. [41] that carefully optimises the execution of the algorithm, thus obtaining better time-efficiency results. It should be emphasised that the purpose of Weatherly's implementation, rather than speed, is results that are "...a known quanitity to compare with other algorithms side by side." [77].

Poudyal and Morozov [81] tested Weatherly's ASCON implementation in his main Crypto library against AES-GCM, obtaining a result of 1106 microseconds (1.1 ms). This is notably close to our CAN delay estimate of approximately 1.2 ms obtained from the 4-byte benchmarking; although our actual CAN testing result for ASCON was longer at approximately 1.5 ms. This difference may be due to the overhead of our AEAD approach processing the CAN ID and control bits as associated data, if they did not perform such associated data processing.

6.6. Considering Compiled Code Size vs Time-Efficiency

Existing NIST benchmarking data [66] has data for the code size of various software implementations of the finalists compiled on various platforms. Figure 5 shows the sizes for Weatherly's implementations when compiled on the Arduino Uno (thus compiled for the ATmega328P). Available program space can be limited on certain microprocessors (especially low-end AVRs). Thus, if considering ASCON alternatives, any possible trade-offs between time-efficiency and program size will matter.

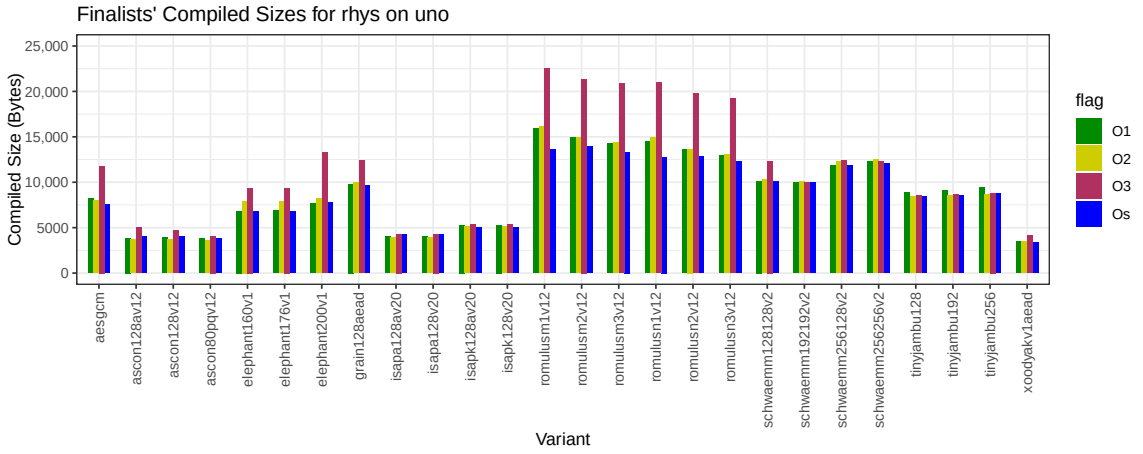

Figure 5. Compiled program sizes of Weatherly's implementations from NIST's benchmarking repository [66].

Amongst Weatherly's implementations, TinyJAMBU and Sparkle-128-128 are the best time-efficiency choices if a larger code size than ASCON can be tolerated. Sparkle lacks formal nonce-misuse resilience, so TinyJAMBU is preferred if this resilience is desired. If small program size is more important, Xoodyak is next-best to ASCON in terms of time-efficiency while being comparable to ASCON in terms of code size.

Note that different implementations of the same algorithm can differ in code size, for example, the same NIST dataset for the reference implementations, as shown in Figure 6, demonstrates the standard reference implementation of ASCON is larger than the standard reference implementation TinyJAMBU. Our discussion of time-efficiency vs code size is specific to Weatherly's implementations, as comparing multiple implementations is out-of-scope for this paper. Further exploration of implementations' code sizes and how the cipher designs influence this is potential future work. Although code-size is not the primary focus of this work, we highlight this point to showcase there are multiple aspects that may be considered when selecting an algorithm for MCUs which may or may not only be constrained in processing performance, but also in program memory.

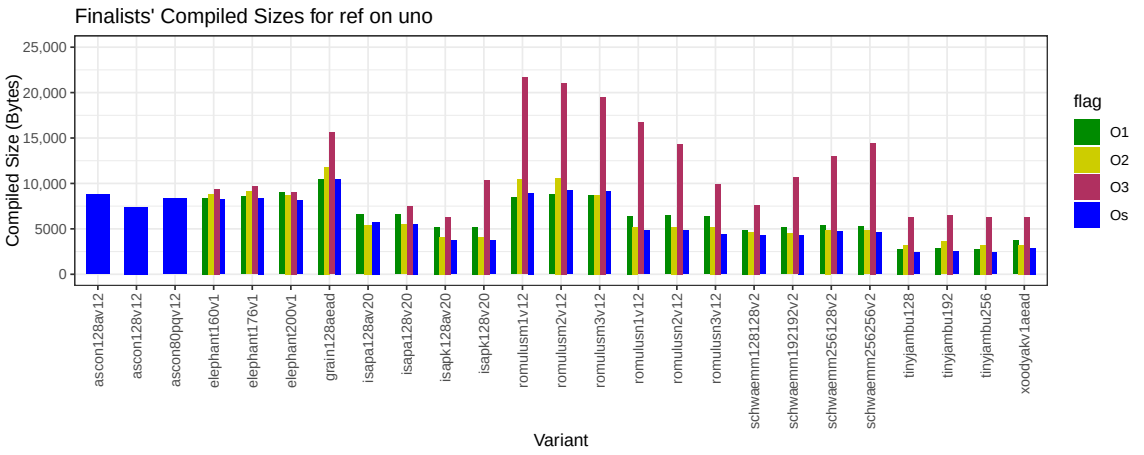

Figure 6. Compiled program sizes of the reference implementations from NIST's benchmarking repository [66].

7. Conclusions

This paper evaluates latency of NIST finalists with actual CAN transmissions of our AEAD approach for the standard CAN frame using a 4-byte payload, 3-byte tag, counter bits, and including CAN IDs (along with the DLC and RTR values) as associated data. Weatherly's implementation of the NIST finalists were utilised, and his testing procedure was modified to include 4-byte message lengths, so we can observe how the very short payloads of CAN frames affect the time-efficiency of the finalists.

Our results show that when using software implementations on low-end 16 MHz/20 MHz 8-bit AVRs with no cryptographic hardware acceleration, representative of current low-end automotive processors; some finalists meaningfully outperform AES-GCM. Our additional in-memory 4-byte tests and individual factors also demonstrate that some finalists become more relatively efficient than AES-GCM as the messages lengths reduce (from 128 to 16 to 4 bytes), indicating these finalists may have less severe initialisation cost in their message processing overhead, relevant to short message payloads.

We categorise the finalists into efficiency tiers for securing CANBUS transmissions with our AEAD approach by comparing finalists' encryption and decryption times against AES-GCM and the reference ChaCha20-Poly1305 implementation. While ASCON is standardised and has high-performance, TinyJAMBU is a similarly efficient alternative, usable provided the security claim limitation is sufficiently addressed for the use case of CAN bus. Schwaemm-128-128 is a faster alternative to ASCON if nonce-misuse resilience is not required. In the case that the AEAD algorithm is also used to just authenticate all or some CAN frames without encrypting the payload, Schwaemm-128-128 has an even greater relative advantage, obtaining a more significant improvement than TinyJAMBU and ASCON. The Moderately and Marginally Advantageous finalists still exhibit a clear time-efficiency advantage against AES-GCM and are worth considering where Very Advantageous finalists are not used. We recommend use of AES-GCM rather than any finalists ranking beneath Marginally Advantageous.

Overall, this work presents a time-efficiency evaluation of the finalists from NIST's LWC standardisation process in the CAN environment. While standardisation will likely result in ASCON being preferred in most cases, TinyJAMBU is a similar time-efficiency alternative, and Schwaemm-128-128 is a significantly more time-efficient alternative when nonce-misuse resilience is not required, especially so if encrypting the payload for confidentiality is not required. This evaluation was performed to help inform of alternative finalists that, in addition to ASCON, are viable future choices of algorithms with respect to software-implementation time-efficiency on low-end hardware for implementation of CAN bus security.

Future work in this research field includes:

- Further exploration for some of the AEAD ciphers to better understand why they are less efficient when processing all the message content as associated data than when encrypting the main message.
- Testing NIST finalist lightweight hash functions in an appropriate mode such as HMAC for authentication-only scenarios.
- Considering other frame formats with different payload lengths (such as a 4-byte tag with no nonce LSB and/or using two frames where one has intended payload and the other contains a larger authentication tag).
- Similar benchmarking on a wider variety of MCU capabilities/architectures.

Author Contributions

J.C.: Writing—original draft, Conceptualisation, Methodology, Software; L.S.: Writing—review & editing, Supervision; G.W.: X.X.: Writing—review & editing, Supervision. All authors have read and agreed to the published version of the manuscript.

Funding

This research was supported by the Commonwealth through an Australian Government Research Training Program Scholarship [DOI: https://doi.org/10.82133/C42F-K220].

Institutional Review Board Statement

Not applicable.

Informed Consent Statement

Not applicable.

Data Availability Statement

Data from the experiments can be requested from the contact author via email.

Conflicts of Interest

The authors declare no conflict of interest.

Use of AI and AI-Assisted Technologies

No AI tools were utilised for this paper.

Appendix A. Results of Standard Testing of 128-Byte and 16-Byte Payloads

Results from 128-Byte and 16-Byte Standard Testing (Microseconds Per Byte)

Cipher	Mega 2560 Rev3							Nano Every							Uno R3						
	Sanity	Factor		16-Byte		128-Byte		Sanity	Factor		16-Byte		128-Byte		Sanity	Factor		16-Byte		128-Byte	
	Check	Speed	Time	Enc	Dec	Enc	Dec	Check	Speed	Time	Enc	Dec	Enc	Dec	Check	Speed	Time	Enc	Dec	Enc	Dec
AES-GCM-128	Pass	1.26	0.79	197.79	199.06	101.67	102.56	Pass	1.36	0.74	175.89	177.08	90.18	91.01	Pass	1.27	0.79	195.25	196.52	100.36	101.25
AES-GCM-192	Pass	1.22	0.82	205.6	206.88	104.81	105.7	Pass	1.31	0.76	183.53	184.72	93.28	94.11	Pass	1.23	0.81	203.01	204.28	103.48	104.37
AES-GCM-256	Pass	1.18	0.85	216.71	217.99	108.34	109.24	Pass	1.26	0.79	194.55	195.74	96.79	97.62	Pass	1.19	0.84	214.07	215.34	106.99	107.89
ASCON-128	Pass	2.55	0.39	115.54	116.82	47.79	48.7	Pass	2.65	0.38	106.52	107.73	43.93	44.79	Pass	2.54	0.39	115.48	116.75	47.76	48.67
ASCON-128a	Pass	3.28	0.3	103.56	104.8	35.33	36.24	Pass	3.41	0.29	95.39	96.54	32.49	33.34	Pass	3.26	0.31	103.51	104.75	35.31	36.22
ASCON-80pq	Pass	2.55	0.39	116.02	116.97	47.85	48.72	Pass	2.65	0.38	107.03	107.9	44	44.81	Pass	2.54	0.39	115.96	116.9	47.82	48.69
ChaChaPoly	N/A	Refrence		443.13	444.38	104.15	105.06	N/A	Refrence		426.86	428.04	99.4	100.25	N/A	Refrence		441.84	443.08	103.48	104.39
Elephant-Delirium	Pass	2.07	0.48	128.93	130.68	60.49	61.74	Pass	2.05	0.49	124.87	126.5	58.33	59.59	Pass	2.07	0.48	128.26	129.99	60.2	61.46
Elephant-Dumbo	Pass	0.45	2.22	561.55	563.22	284.59	285.8	Pass	0.43	2.31	559.7	561.25	283.31	284.53	Pass	0.45	2.23	561.27	562.92	284.47	285.67
Elephant-Jumbo	Pass	0.39	2.54	715.52	717.23	316.67	317.87	Pass	0.38	2.65	713.72	715.3	315.55	316.74	Pass	0.39	2.55	715.22	716.91	316.55	317.75
GIFT-COFB	Pass	2.92	0.34	113.13	114.47	39.73	41.41	Pass	2.87	0.35	109.86	111.02	38.86	40.37	ChaChaPoly Not Compiled			112.65	113.93	39.55	41.27
Grain-128AEAD	Pass	0.95	1.05	227.34	251.38	126.52	150.31	Pass	0.97	1.03	213.5	235.49	118.84	140.6	Pass	0.96	1.04	224.61	248.64	124.28	148.07
ISAP-A-128	FAIL	0.47	2.14	5925.93	5927.15	869.72	870.63	FAIL	0.49	2.04	5462.73	5463.88	801.73	802.57	FAIL	0.47	2.14	5923.9	5925.12	869.43	870.33
ISAP-A-128A	FAIL	0.1	10.06	974.89	976.13	219.57	220.47	FAIL	0.1	9.69	888.08	889.23	200.99	201.84	FAIL	0.1	10.11	973.48	974.7	219.35	220.26
ISAP-K-128	FAIL	0.38	2.62	8845.49	8846.71	1233.86	1234.77	FAIL	0.41	2.45	7904.8	7905.94	1102.66	1103.51	FAIL	0.38	2.63	8827.88	8829.1	1231.41	1232.32
ISAP-K-128A	FAIL	0.07	14.62	1291.27	1292.5	258.23	259.14	FAIL	0.07	13.65	1152.31	1153.45	230.57	231.42	FAIL	0.07	14.67	1287.84	1289.06	257.61	258.52
PB-AEAD[128]	Pass	1.43	0.7	161.17	162.11	91.14	92.37	Pass	1.46	0.69	151.23	152.08	85.54	86.58	Pass	1.42	0.7	160.92	161.85	91	92.23
PB-AEAD[32]	Pass	0.43	2.35	395.71	397.18	325.64	327.03	Pass	0.43	2.3	371.36	372.61	305.63	306.78	Pass	0.42	2.36	395.08	396.53	325.11	326.51
Romulus-M	Pass	1.94	0.52	179.49	180.91	96.37	97.32	Pass	1.41	0.71	162.87	164.25	87.61	89.11	Pass	1.97	0.51	175.58	176.99	94.25	95.2
Romulus-N	Pass	1.34	0.75	121.08	122.25	66.94	67.79	Pass	2.04	0.49	110.04	111.11	60.88	61.67	Pass	1.36	0.73	118.45	119.61	65.48	66.32
Romulus-T	Pass	0.57	1.74	471.91	473.3	219.21	220.11	Standard Test Not Run			N/A	N/A	N/A	N/A	Pass	0.58	1.71	461.5	462.88	214.35	215.26
SPARKLE-128-128	Pass	4.67	0.21	50.74	51.51	23.51	24	Pass	4.85	0.21	47.4	48.15	21.8	22.27	Pass	4.65	0.21	50.55	51.32	23.43	23.92
SPARKLE-192-192	Pass	3.93	0.25	84.09	85.14	30.64	29.68	Pass	4.08	0.25	77.75	78.64	28.29	27.32	Pass	3.92	0.26	83.87	84.93	30.57	29.61
SPARKLE-256-128	Pass	5.31	0.19	85.75	85.76	23.39	23.78	Pass	5.47	0.18	79.21	79.2	21.52	21.88	Pass	5.3	0.19	85.53	85.55	23.34	23.72
SPARKLE-256-256	Pass	3.16	0.32	127.76	128.45	34.36	34.89	Pass	3.3	0.3	117.13	117.82	31.43	31.95	Pass	3.15	0.32	127.52	128.21	34.3	34.84
TinyJAMBU-128	Pass	2.21	0.45	110.09	111.31	58.3	59.21	Pass	2.14	0.47	108.39	109.52	57.53	58.37	Pass	2.2	0.46	109.97	111.18	58.25	59.16
TinyJAMBU-192	Pass	2	0.5	118.94	120.16	64.5	65.4	Pass	1.94	0.51	117.26	118.39	63.74	64.58	Pass	1.99	0.5	118.83	120.03	64.45	65.36
TinyJAMBU-256	Pass	1.83	0.55	128.01	129.22	70.81	71.72	Pass	1.77	0.56	126.34	127.47	70.07	70.91	Pass	1.82	0.55	127.89	129.09	70.77	71.67
Xoodyak	Pass	2.02	0.5	171.68	172.84	57.4	58.36	Pass	2.22	0.45	149.44	150.53	49.95	50.85	Pass	2.01	0.5	171.33	172.49	57.29	58.25

Figure A1. Factors from the 128-byte and 16-byte byte testing. Romulus-T crashed during memory testing on the Nano. GIFT-COFB could not be compiled with ChaCha20-Poly1305 on the Uno due to lack of flash memory so it has no factors for the Uno. ISAP failed the existing sanity checks when using a shorter tag length, but still operated for benchmarking with CAN, with results retained for completeness.

Appendix B. Results of Testing 4 Byte Payloads (Incl. CAN Delay Estimates)

Results from 4-Byte Standard Testing (Microseconds Per Byte)												
Cipher	Mega 2560 Rev3				Nano Every				Uno R3			
	4-Byte		CAN Delay Est.		4-Byte		CAN Delay Est.		4-Byte		CAN Delay Est.	
	Enc	Dec	Enc	Dec	Enc	Dec	Enc	Dec	Enc	Dec	Enc	Dec
AES-GCM-128	786.35	788.92	3145.4	3155.68	699.07	701.47	2796.28	2805.86	776.19	778.73	3104.76	3114.92
AES-GCM-192	817.63	820.18	3270.5	3280.72	729.65	732.04	2918.58	2928.16	807.27	809.8	3229.08	3239.2
AES-GCM-256	862.04	864.6	3448.16	3458.4	773.72	776.11	3094.9	3104.44	851.5	854.02	3406	3416.06
ASCON-128	311.05	313.54	1244.2	1254.16	287.35	289.69	1149.38	1158.74	310.89	313.33	1243.58	1253.34
ASCON-128a	313.48	315.82	1253.9	1263.26	288.95	291.14	1155.82	1164.58	313.33	315.66	1253.34	1262.62
ASCON-80pq	312.98	314.14	1251.9	1256.58	289.36	290.4	1157.42	1161.6	312.8	313.95	1251.2	1255.8
ChaChaPoly	1770.72	1773.11	7082.9	7092.44	1705.82	1708.06	6823.28	6832.24	1765.54	1767.89	7062.16	7071.54
Elephant-Delirium	512.3	516.86	2049.22	2067.46	496.28	500.53	1985.12	2002.12	509.63	514.13	2038.52	2056.52
Elephant-Dumbo	2242.79	2247.03	8971.16	8988.12	2235.58	2239.52	8942.32	8958.08	2241.66	2245.85	8966.64	8983.4
Elephant-Jumbo	2858.7	2863.07	11434.82	11452.3	2851.66	2855.72	11406.64	11422.88	2857.5	2861.8	11429.98	11447.22
GIFT-COFB	459.8	460.71	1839.18	1842.84	445.77	446.47	1783.06	1785.86	457.67	458.17	1830.66	1832.68
Grain-128AEAD	527.49	552.34	2109.96	2209.34	495.1	517.86	1980.38	2071.44	523.22	548.04	2092.9	2192.18
ISAP-A-128	23261.26	23263.53	93045.03	93054.1	21443.09	21445.21	85772.35	85780.85	23253.31	23255.54	93013.22	93022.14
ISAP-A-128A	3528.62	3530.89	14114.48	14123.58	3210.54	3212.66	12842.18	12850.66	3523.1	3525.33	14092.42	14101.34
ISAP-K-128	35375.73	35377.98	141502.9	141511.93	31613.32	31615.44	126453.28	126461.76	35305.3	35307.53	141221.18	141230.12
ISAP-K-128A	5158.85	5161.12	20635.42	20644.46	4603.39	4605.5	18413.56	18422.02	5145.13	5147.38	20580.54	20589.52
PHOTON-Beetle-AEAD[128]	639.35	641.36	2557.4	2565.46	599.92	601.79	2399.68	2407.16	638.34	640.32	2553.38	2561.26
PHOTON-Beetle-AEAD[32]	636.01	637.68	2544.02	2550.72	596.78	598.34	2387.14	2393.34	635.02	636.65	2540.06	2546.58
Romulus-M	713.34	716.56	2853.34	2866.24	647.09	650.12	2588.34	2600.48	697.62	700.8	2790.48	2803.22
Romulus-N	479.01	481.26	1916.02	1925.02	434.53	436.58	1738.12	1746.32	468.51	470.73	1874.04	1882.92
Romulus-T	1654.52	1657.55	6618.06	6630.18	Romulus-T Standard Testing Did Not Run				1618.06	1621.06	6472.22	6484.26
SPARKLE-128-128	196.53	197.97	786.12	791.86	183.67	185.07	734.68	740.28	195.76	197.19	783.02	788.74
SPARKLE-192-192	329.2	331.14	1316.78	1324.56	304.38	306.08	1217.52	1224.3	328.33	330.29	1313.32	1321.14
SPARKLE-256-128	333.19	333.63	1332.76	1334.5	307.98	308.32	1231.9	1233.26	332.33	332.77	1329.3	1331.08
SPARKLE-256-256	503.86	505.48	2015.42	2021.94	461.94	463.52	1847.74	1854.1	502.92	504.54	2011.68	2018.14
TinyJAMBU-128	287.71	289.92	1150.84	1159.7	282.83	284.92	1131.32	1139.66	287.36	289.55	1149.46	1158.18
TinyJAMBU-192	305.66	307.88	1222.64	1231.52	300.82	302.91	1203.28	1211.62	305.31	307.49	1221.24	1229.96
TinyJAMBU-256	324.13	326.35	1296.52	1305.4	319.33	321.39	1277.32	1285.56	323.77	325.97	1295.1	1303.86
Xoodyak	682.17	684.19	2728.7	2736.74	593.6	595.52	2374.42	2382.06	680.8	682.78	2723.2	2731.12

Figure A2. Performance of the lightweight ciphers from standard testing. Estimated CAN delay is the 4-byte message encryption rate multiplied by four.

Appendix C. CAN Testing Results Statistical Table - Mega 2560 Rev3

CAN (Authenticated Encryption) Results' Statistical Variation for Mega 2560 Rev3

Cipher Family	Variant	Overall Delay						Encryption Time						Decryption Time						Cryptography Delay (Enc + Dec)						Send Time (Overall - Crypto)					
		Mean	Min	Max	Range	SD	CV	Mean	Min	Max	Range	SD	CV	Mean	Min	Max	Range	SD	CV	Mean	Min	Max	Range	SD	CV	Mean	Min	Max	Range	SD	CV
ChaChaPoly		16514	16488	16540	60	10	0.061	7537	7532	7544	12	4	0.053	7546	7540	7552	12	4	0.053	15082	15072	15092	20	5	0.033	1431	1400	1468	68	10	0.698
AES-GCM	128	10110	10076	10140	64	12	0.119	4334	4332	4340	8	3	0.069	4344	4340	4352	12	4	0.092	8678	8672	8692	20	5	0.058	1432	1409	1468	68	12	0.838
	192	10358	10332	10388	56	12	0.116	4459	4456	4464	8	4	0.090	4469	4464	4476	12	4	0.090	8928	8920	8948	28	5	0.056	1430	1404	1460	56	11	0.769
	256	10714	10692	10732	40	10	0.093	4637	4632	4640	8	4	0.086	4647	4640	4652	12	4	0.086	9284	9272	9292	20	5	0.054	1431	1408	1452	44	11	0.769
ASCON	128	4555	4532	4580	48	10	0.220	1559	1552	1564	12	4	0.257	1568	1564	1572	8	4	0.255	3126	3116	3136	20	6	0.192	1429	1408	1460	52	11	0.770
	128a	4766	4748	4796	56	10	0.210	1664	1656	1668	12	4	0.240	1673	1668	1676	8	4	0.239	3337	3324	3344	20	5	0.150	1436	1400	1464	64	11	0.769
	80pq	4565	4544	4592	48	12	0.263	1567	1568	1572	12	4	0.255	1571	1564	1576	12	4	0.255	3137	3124	3148	24	5	0.159	1429	1404	1460	56	13	0.910
Elephant	Delirium	5555	5536	5580	44	10	0.180	2054	2052	2056	4	3	0.146	2073	2068	2080	12	2	0.096	4126	4120	4136	16	3	0.073	1429	1408	1456	48	10	0.700
	Dumbo	19481	19384	19436	52	12	0.062	8977	8968	8980	12	4	0.045	8993	8984	8996	12	4	0.044	17969	17956	17976	20	5	0.028	1433	1412	1464	52	12	0.837
	Jumbo	24331	24296	24364	68	13	0.053	11440	11436	11448	12	4	0.035	11458	11452	11464	12	4	0.035	22897	22888	22912	24	5	0.022	1435	1400	1472	72	13	0.906
GIFT-COFB		5129	5092	5152	60	13	0.254	1845	1836	1848	12	3	0.163	1848	1840	1852	12	3	0.162	3692	3680	3700	20	4	0.108	1428	1409	1464	64	13	0.910
Grain-128AEAD		7140	7129	7168	48	13	0.182	2806	2800	2808	8	4	0.143	2905	2896	2908	12	3	0.103	5711	5696	5716	20	5	0.088	1430	1408	1460	52	13	0.909
ISAP	A-128A	29697	29668	29728	60	11	0.037	14126	14120	14128	8	3	0.021	14135	14128	14148	12	3	0.021	28261	28248	28268	20	4	0.014	1436	1404	1468	64	11	0.766
	K-128A	42744	42712	42776	64	12	0.028	20647	20644	20656	12	3	0.015	20657	20652	20664	12	4	0.019	41304	41296	41316	20	5	0.012	1441	1412	1472	60	12	0.833
	A-128	187601	187576	187636	60	13	0.007	93858	93848	93868	12	3	0.003	93866	93856	93868	12	3	0.003	186123	186112	186128	16	5	0.003	1479	1452	1508	56	13	0.879
	K-128	284541	284529	284568	48	11	0.004	141515	141512	141529	8	3	0.002	141524	141520	141532	12	4	0.003	283038	283032	283052	20	5	0.002	1503	1476	1532	56	12	0.798
PHOTON-Beetle-AEAD	[128]	9080	9064	9112	48	11	0.121	3822	3816	3824	8	4	0.105	3838	3824	3832	8	4	0.104	7651	7640	7656	16	5	0.065	1429	1408	1460	52	12	0.840
	[32]	11544	11512	11576	64	10	0.087	5054	5044	5056	12	3	0.059	5060	5052	5060	8	2	0.040	10113	10100	10116	16	3	0.030	1431	1400	1464	64	11	0.769
Romulus	N	5303	5272	5336	64	12	0.226	1932	1924	1936	12	3	0.155	1941	1932	1944	12	3	0.156	3872	3860	3880	20	4	0.103	1431	1400	1464	64	12	0.839
	M	4174784492	4173540988	4176027476	2486488	722575	0.017	2865	2856	2868	12	4	0.140	2880	2880	2880	0	0	0.000	5745	5736	5748	12	4	0.070	4174778748	4173535240	4176021732	2486492	722576	0.017
	T	14743	14729	14768	48	11	0.075	6650	6644	6656	12	4	0.060	6662	6656	6668	12	4	0.060	13312	13300	13324	24	6	0.045	1431	1404	1460	56	12	0.839
Sparkle	128-128	3802	3776	3828	52	13	0.342	1184	1180	1192	12	3	0.253	1190	1188	1196	8	3	0.252	2373	2368	2380	12	4	0.169	1429	1404	1460	56	13	0.910
	256-128	5430	5404	5460	56	11	0.203	2000	1992	2004	12	2	0.100	2082	1992	2004	12	3	0.150	4001	3992	4004	12	3	0.075	1429	1404	1460	56	11	0.770
	192-192	5484	5376	5436	60	13	0.241	1848	1976	1984	8	3	0.151	1992	1984	1992	8	2	0.100	3975	3964	3976	12	3	0.075	1436	1400	1460	60	13	0.909
	256-256	7497	7472	7524	52	13	0.173	3030	3024	3032	8	3	0.099	3037	3028	3040	12	2	0.066	6066	6056	6072	16	3	0.049	1431	1400	1460	60	13	0.908
TinyJAMBU	128	4272	4232	4304	72	11	0.257	1417	1412	1424	12	4	0.282	1426	1420	1432	12	4	0.281	2843	2832	2856	24	6	0.211	1429	1396	1464	68	11	0.770
	192	4414	4388	4448	52	13	0.295	1489	1484	1492	8	4	0.269	1497	1492	1504	12	4	0.267	2986	2976	2996	20	5	0.167	1429	1404	1452	48	13	0.910
	256	4568	4536	4588	52	11	0.241	1563	1556	1568	12	4	0.256	1571	1564	1576	12	4	0.255	3134	3120	3144	24	6	0.191	1427	1400	1452	52	11	0.771
Xoodyak		6917	6888	6948	60	11	0.159	2741	2732	2744	12	4	0.146	2748	2740	2752	12	4	0.146	5488	5476	5496	20	6	0.109	1429	1400	1468	68	12	0.840

Figure A3. Results on the Arduino Mega 2560 Rev3 from the CAN testing for 500 Kbps baud rate, with associated data, and extended IDs. Shown in table format with statistical parameters. Measured "Overall Delay" can differ by a few microseconds from sum of the mean values of "Encryption Time", "Decryption Time", and "Send Time". However, Figure 3 depiction of this summation is reasonably representative of the overall delay, since a few microseconds is insignificant compared to the practical differences in performance between finalist variants.

Appendix D. CAN Testing Results Statistical Table—Nano Every

CAN (Authenticated Encryption) Results' Statistical Variation for Nano Every

Cipher Family	Variant	Overall Delay						Encryption Time						Decryption Time						Cryptography Delay (Enc + Dec)						Send Time (Overall - Crypto)					
		Mean	Min	Max	Range	SD	CV	Mean	Min	Max	Range	SD	CV	Mean	Min	Max	Range	SD	CV	Mean	Min	Max	Range	SD	CV	Mean	Min	Max	Range	SD	CV
ChaChaPoly		16455	16408	16496	88	15	0.091	7242	7240	7252	12	3	0.041	7252	7248	7260	12	3	0.041	14493	14488	14508	20	4	0.028	1962	1916	2000	84	16	0.815
AES-GCM	128	9634	9592	9676	84	15	0.156	3834	3824	3836	12	4	0.104	3843	3836	3848	12	4	0.104	7677	7660	7684	24	6	0.078	1957	1912	1992	80	15	0.766
	192	9876	9836	9920	84	17	0.172	3956	3948	3960	12	3	0.076	3965	3956	3968	12	4	0.101	7921	7904	7928	24	5	0.063	1956	1916	2000	84	17	0.869
	256	10225	10204	10260	56	13	0.127	4132	4128	4140	12	3	0.073	4142	4140	4152	12	3	0.072	8273	8268	8284	16	4	0.048	1952	1924	1992	68	13	0.666
ASCON	128	4837	4812	4876	64	14	0.289	1441	1436	1448	12	5	0.347	1451	1444	1456	12	5	0.345	2891	2880	2904	24	6	0.208	1947	1920	1984	64	13	0.770
	128a	5028	4996	5064	68	18	0.358	1536	1528	1540	12	5	0.326	1545	1540	1552	12	5	0.324	3081	3068	3092	24	6	0.195	1947	1916	1992	76	19	0.976
	80pq	4850	4820	4880	60	16	0.330	1449	1444	1456	12	5	0.345	1454	1448	1460	12	5	0.344	2902	2892	2916	24	6	0.207	1948	1916	1984	68	18	0.924
Elephant	Delirium	5947	5920	5988	68	14	0.235	1992	1984	1996	12	2	0.100	2010	2000	2012	12	3	0.149	4002	3992	4008	16	4	0.100	1946	1916	1988	72	14	0.719
	Dumbo	19876	19848	19924	84	13	0.065	8949	8940	8952	12	5	0.056	8966	8956	8968	12	5	0.056	17914	17896	17928	24	5	0.033	1962	1928	2004	76	13	0.663
	Jumbo	24888	24776	24844	68	17	0.069	11414	11408	11424	16	3	0.026	11430	11428	11440	12	4	0.035	22843	22836	22864	28	5	0.022	1965	1932	2004	72	17	0.865
GIFT-COFB		5535	5500	5576	76	18	0.325	1791	1784	1796	12	4	0.223	1794	1784	1796	12	4	0.223	3584	3568	3592	24	6	0.167	1952	1912	1988	76	18	0.922
Grain-128AEAD		7312	7288	7356	68	16	0.219	2636	2628	2640	12	5	0.190	2728	2720	2732	12	5	0.183	5364	5348	5372	24	7	0.130	1949	1920	1988	68	16	0.821
ISAP	A-128A	27669	27636	27724	88	16	0.058	12847	12840	12852	12	5	0.039	12856	12848	12860	12	5	0.039	25762	25688	25712	24	7	0.027	1967	1928	2012	84	16	0.813
	K-128A	38835	38796	38872	76	16	0.041	18418	18408	18420	12	3	0.016	18427	18420	18428	8	2	0.011	36844	36836	36848	12	3	0.008	1991	1952	2024	72	16	0.804
	A-128	173677	173604	173744	140	26	0.015	85777	85768	85780	12	5	0.006	85786	85776	85788	12	4	0.005	171562	171544	171568	24	6	0.003	2115	2048	2188	140	26	1.229
	K-128	255153	255068	255248	180	35	0.014	126457	126452	126464	12	5	0.004	126467	126460	126472	12	5	0.004	252924	252912	252936	24	7	0.003	2236	2136	2332	196	35	1.570
PHOTON-Beetle-AEAD	[128]	9137	9100	9168	68	12	0.131	3588	3584	3596	12	5	0.139	3597	3588	3600	12	5	0.139	7185	7172	7196	24	6	0.084	1953	1908	1992	84	12	0.614
	[32]	11452	11420	11488	68	19	0.166	4742	4736	4748	12	5	0.105	4751	4744	4756	12	5	0.105	9493	9480	9504	24	6	0.063	1960	1924	1996	72	19	0.969
Romulus	N	5468	5428	5596	72	16	0.293	1755	1748	1768	12	5	0.285	1763	1756	1768	12	4	0.227	3517	3504	3528	24	6	0.171	1951	1912	1988	68	17	0.871
	M	N/A (Crashed on Nano)																													
	T	N/A (Crashed on Nano)																													
Sparkle	128-128	4164	4132	4204	72	17	0.408	1107	1104	1116	12	4	0.361	1114	1112	1124	12	3	0.269	2221	2216	2236	20	5	0.225	1944	1908	1984	76	17	0.874
	256-128	5650	5612	5688	68	16	0.283	1848	1840	1852	12	4	0.216	1856	1848	1852	12	4	0.216	3697	3680	3704	24	6	0.162	1954	1912	1988	68	16	0.819
	192-192	5625	5588	5664	84	18	0.320	1835	1824	1836	12	4	0.218	1841	1832	1844	12	4	0.217	3676	3656	3688	24	6	0.163	1950	1908	1988	88	18	0.923
	256-256	7514	7476	7540	64	12	0.160	2776	2768	2780	12	5	0.180	2784	2776	2788	12	4	0.144	5559	5544	5568	24	7	0.126	1955	1916	1988	72	13	0.665
TinyJAMBU	128	4746	4704	4776	72	12	0.253	1395	1388	1400	12	5	0.358	1404	1400	1412	12	5	0.356	2798	2788	2812	24	6	0.214	1948	1908	1976	68	13	0.667
	192	4890	4864	4924	60	17	0.348	1466	1460	1472	12	5	0.341	1475	1468	1480	12	5	0.339	2941	2928	2952	24	7	0.238	1950	1912	1988	76	13	0.974
	256	5034	5004	5080	76	15	0.298	1541	1536	1548	12	5	0.324	1550	1544	1556	12	5	0.323	3090	3080	3104	24	7	0.227	1944	1920	1988	68	15	0.772
Xoodyak		6736	6704	6768	64	17	0.252	2388	2384	2396	12	5	0.209	2395	2392	2404	12	4	0.167	4783	4776	4796	20	6	0.125	1953	1916	1984	68	18	0.922

Figure A4. Results for the Arduino Nano Every from the CAN testing for 500 Kbps baud rate, with associated data, and extended IDs. Measured "Overall Delay" can differ by a few microseconds from sum of the mean values of "Encryption Time", "Decryption Time", and "Send Time". However, Figure 3 depiction of this summation is reasonably representative of the overall delay, since a few microseconds is insignificant compared to the practical differences in performance between finalist variants.

Appendix E. CAN Testing Results Statistical Table—Uno R3

CAN (Authenticated Encryption) Results' Statistical Variation for Uno R3

Cipher Family	Variant	Overall Delay						Encryption Time						Decryption Time						Cryptography Delay (Enc + Dec)						Send Time (Overall - Crypto)					
		Mean	Min	Max	Range	SD	CV	Mean	Min	Max	Range	SD	CV	Mean	Min	Max	Range	SD	CV	Mean	Min	Max	Range	SD	CV	Mean	Min	Max	Range	SD	CV
ChaChaPoly		16352	16328	16380	52	9	0.055	7596	7500	7512	12	4	0.053	7515	7512	7520	8	4	0.053	15021	15012	15032	20	6	0.040	1331	1300	1360	60	10	0.751
AES-GCM	128	9892	9864	9920	56	11	0.111	4277	4272	4284	12	4	0.094	4287	4284	4292	8	4	0.093	8563	8556	8576	20	5	0.058	1329	1304	1368	56	12	0.903
	192	10141	10116	10164	48	10	0.099	4401	4396	4408	12	4	0.091	4411	4408	4416	8	4	0.091	8812	8804	8824	20	6	0.068	1330	1304	1356	52	11	0.827
	256	10496	10468	10528	60	9	0.089	4578	4572	4584	12	4	0.087	4588	4584	4592	8	4	0.087	9166	9156	9176	20	6	0.065	1331	1308	1360	52	12	0.751
ASCON	128	4455	4432	4488	48	9	0.202	1557	1552	1560	8	4	0.257	1567	1560	1572	12	4	0.255	3124	3112	3132	20	7	0.224	1331	1304	1364	60	11	0.826
	128a	4665	4632	4692	60	11	0.236	1663	1656	1668	12	4	0.241	1672	1668	1676	8	4	0.239	3335	3324	3344	20	5	0.150	1330	1296	1356	60	11	0.827
	80pq	4463	4440	4488	48	10	0.224	1565	1560	1568	8	4	0.256	1578	1564	1576	12	4	0.255	3135	3124	3144	20	5	0.159	1329	1312	1356	44	10	0.752
Elephant	Delirium	5434	5412	5456	44	9	0.166	2044	2036	2044	8	2	0.096	2061	2060	2068	8	2	0.097	4105	4096	4112	16	3	0.073	1330	1308	1348	40	9	0.677
	Dumbo	19298	19264	19328	56	10	0.052	8972	8964	8976	12	4	0.045	8988	8980	8992	12	4	0.045	17959	17944	17968	24	6	0.033	1331	1300	1356	56	11	0.826
	Jumbo	24216	24188	24244	56	11	0.045	11435	11432	11440	8	4	0.035	11452	11448	11460	12	4	0.035	22887	22880	22896	16	4	0.017	1330	1300	1356	56	11	0.827
GIFT-COFB		5000	4976	5028	52	12	0.240	1835	1828	1836	8	3	0.163	1837	1828	1840	12	3	0.163	3671	3664	3676	12	4	0.109	1329	1308	1356	56	12	0.903
Grain-128AEAD		6979	6952	7004	52	10	0.143	2775	2768	2776	8	3	0.108	2874	2868	2876	8	4	0.139	5648	5640	5652	12	4	0.071	1331	1308	1356	48	10	0.751
ISAP	A-128A	29546	29520	29568	48	11	0.037	14104	14096	14108	12	3	0.021	14113	14104	14116	12	3	0.021	28216	28208	28224	16	5	0.018	1330	1304	1348	44	10	0.752
	K-128A	42522	42496	42548	52	11	0.026	20592	20588	20600	12	3	0.015	20601	20596	20608	12	3	0.015	41193	41184	41204	20	4	0.010	1329	1304	1360	56	11	0.828
	A-128	187386	187352	187416	64	11	0.006	93025	93016	93028	12	3	0.003	93034	93028	93036	8	3	0.003	186058	186044	186064	20	5	0.003	1328	1300	1356	56	11	0.828
	K-128	283861	283788	283828	40	10	0.004	141233	141228	141236	8	2	0.002	141242	141232	141244	12	3	0.002	282474	282468	282480	12	4	0.001	1329	1304	1356	52	11	0.828
PHOTON-Beetle AEAD	[128]	8967	8948	8996	48	12	0.134	3816	3808	3820	12	4	0.105	3824	3816	3828	12	4	0.105	7639	7624	7648	24	5	0.065	1329	1304	1360	56	13	0.978
	[32]	11426	11400	11452	52	10	0.088	5045	5036	5048	12	3	0.059	5052	5044	5056	12	2	0.040	10097	10088	10100	12	3	0.030	1330	1300	1356	56	10	0.752
Romulus	N	5118	5092	5148	56	12	0.234	1890	1880	1892	12	3	0.159	1899	1892	1900	8	3	0.158	3788	3780	3792	12	4	0.106	1330	1304	1364	60	12	0.902
	M	4176186024	4174966852	4177404496	2437644	707880	0.017	2802	2796	2808	12	4	0.143	2816	2808	2816	8	4	0.200	5618	5612	5628	16	4	0.071	4176181306	4174960932	4177398876	2437944	707880	0.017
	T	14349	14312	14376	64	11	0.077	6503	6500	6508	8	4	0.062	6516	6512	6520	8	4	0.061	13018	13012	13028	16	5	0.038	1331	1300	1356	56	10	0.751
Sparkle	128-128	3691	3664	3724	60	12	0.325	1178	1176	1184	8	3	0.255	1184	1180	1192	12	4	0.338	2362	2356	2372	16	4	0.169	1329	1300	1356	56	11	0.828
	256-128	5320	5296	5344	48	10	0.188	1994	1984	1996	12	3	0.150	1996	1988	1996	8	3	0.150	3989	3980	3992	12	4	0.100	1331	1304	1368	56	10	0.751
	192-192	5292	5268	5320	52	9	0.170	1977	1968	1980	12	3	0.152	1985	1976	1988	12	3	0.151	3962	3952	3968	16	4	0.101	1330	1304	1360	56	9	0.677
	256-256	7384	7360	7412	52	10	0.135	3022	3020	3024	4	3	0.099	3030	3020	3032	12	3	0.099	6053	6044	6060	16	4	0.066	1331	1304	1368	56	10	0.751
TinyJAMBU	128	4168	4144	4208	56	10	0.240	1415	1412	1420	8	4	0.283	1424	1420	1428	8	4	0.281	2838	2832	2848	16	6	0.211	1330	1304	1368	64	11	0.827
	192	4312	4288	4332	44	9	0.209	1487	1480	1492	12	4	0.269	1496	1492	1500	8	4	0.267	2982	2972	2992	20	7	0.235	1331	1304	1356	52	10	0.751
	256	4458	4432	4484	52	10	0.224	1568	1556	1568	8	4	0.256	1569	1564	1572	8	4	0.255	3129	3120	3136	16	7	0.224	1330	1304	1364	60	10	0.752
Xoodyak		6884	6784	6832	48	10	0.147	2734	2728	2736	8	4	0.146	2741	2736	2744	8	4	0.146	5474	5468	5480	12	4	0.073	1330	1308	1352	44	11	0.827

Figure A5. Results for the Arduino Uno R3 from the CAN testing for 500 Kpbs baud rate, with associated data, and extended IDs. Measured "Overall Delay" can differ by a few microseconds from sum of the mean values of "Encryption Time", "Decryption Time", and "Send Time". However, Figure 3 depiction of this summation is reasonably representative of the overall delay, since a few microseconds is insignificant compared to the practical differences in performance between finalist variants.

Appendix F. CAN Testing Results Statistical Table—Mega 2560 Rev3—No Associated Data

CAN (No AD / Ciphertext-Only) Results' Statistical Variation for Mega 2560 Rev3

Cipher Family	Variant	Overall Delay						Encryption Time						Decryption Time						Cryptography Delay (Enc + Dec)						Send Time (Overall - Crypto)					
		Mean	Min	Max	Range	SD	CV	Mean	Min	Max	Range	SD	CV	Mean	Min	Max	Range	SD	CV	Mean	Min	Max	Range	SD	CV	Mean	Min	Max	Range	SD	CV
ChaChaPoly		15517	15584	15648	52	11	0.070	7088	7080	7088	8	3	0.042	7097	7088	7100	12	3	0.042	14184	14168	14188	20	3	0.021	1434	1400	1456	56	12	0.837
AES-GCM	128	7738	7712	7760	48	9	0.116	3150	3148	3156	8	3	0.095	3160	3156	3168	12	3	0.095	6309	6304	6316	12	4	0.063	1429	1400	1452	52	9	0.630
	192	7967	7968	8012	44	12	0.150	3275	3272	3288	8	3	0.092	3285	3280	3292	12	3	0.091	6559	6552	6568	16	5	0.076	1428	1404	1456	52	13	0.910
	256	8345	8316	8368	52	11	0.132	3453	3448	3460	12	4	0.116	3463	3456	3468	12	4	0.116	6915	6904	6928	24	5	0.072	1430	1400	1452	52	11	0.769
ASCON	128	3935	3904	3968	64	11	0.280	1249	1244	1256	12	3	0.240	1258	1256	1264	8	3	0.238	2506	2500	2516	16	4	0.160	1429	1400	1456	56	11	0.770
	128a	3951	3928	3976	48	11	0.278	1258	1256	1264	8	3	0.238	1267	1264	1272	8	3	0.237	2525	2520	2536	16	4	0.158	1427	1408	1456	48	11	0.771
	80pq	3943	3916	3972	56	11	0.279	1257	1252	1264	12	4	0.318	1263	1256	1268	12	4	0.317	2520	2508	2528	20	5	0.198	1424	1400	1456	56	11	0.772
Elephant	Delirium	5552	5524	5580	56	10	0.180	2054	2052	2060	8	3	0.146	2072	2068	2080	12	3	0.145	4125	4120	4132	12	3	0.073	1427	1396	1456	60	10	0.701
	Dumbo	19399	19376	19424	48	11	0.057	8976	8968	8980	12	3	0.033	8992	8984	8996	12	4	0.044	17968	17956	17976	20	5	0.028	1432	1408	1464	56	12	0.838
	Jumbo	24329	24296	24348	52	12	0.049	11439	11436	11448	12	4	0.035	11456	11452	11464	12	4	0.035	22895	22888	22908	20	5	0.022	1434	1400	1456	56	12	0.837
GIFT-COFB		5118	5100	5148	48	11	0.215	1844	1836	1848	12	3	0.163	1847	1840	1848	8	3	0.162	3690	3676	3696	20	5	0.136	1428	1408	1460	52	12	0.840
Grain-128AEAD		5755	5728	5784	56	12	0.209	2115	2112	2120	8	3	0.142	2213	2212	2220	8	3	0.136	4328	4324	4336	12	4	0.092	1428	1404	1456	52	12	0.840
ISAP	A-128A	29682	29652	29716	64	11	0.037	14119	14112	14124	12	3	0.021	14128	14120	14132	12	3	0.021	28247	28236	28252	16	5	0.018	1436	1400	1476	76	12	0.836
	K-128A	42728	42700	42772	72	13	0.030	20648	20636	20648	12	3	0.015	20649	20644	20656	12	3	0.015	41289	41280	41300	20	5	0.012	1441	1408	1484	76	14	0.972
	A-128	187583	187560	187612	52	13	0.007	93049	93040	93052	12	4	0.004	93058	93052	93060	8	3	0.003	186107	186096	186112	16	4	0.002	1476	1452	1508	56	12	0.813
	K-128	284526	284496	284552	56	9	0.003	141508	141504	141512	8	3	0.002	141517	141512	141524	12	4	0.003	283824	283816	283836	20	5	0.002	1503	1476	1524	48	9	0.599
PHOTON-Beetle AEAD	[128]	6559	6532	6588	56	12	0.183	2562	2556	2568	12	4	0.156	2570	2564	2572	8	4	0.156	5131	5120	5140	20	5	0.097	1428	1396	1456	60	13	0.910
	[32]	8531	8492	8568	76	11	0.168	2548	2544	2552	8	4	0.157	2565	2548	2568	12	4	0.157	5103	5092	5112	20	5	0.098	1428	1396	1468	72	11	0.770
Romulus	N	5277	5252	5308	56	11	0.208	1921	1912	1924	12	3	0.156	1929	1928	1932	12	3	0.156	3849	3836	3856	20	5	0.130	1428	1408	1460	52	12	0.840
	M	4253277417	4252086736	4254467204	2380468	691762	0.016	2858	2852	2860	8	4	0.140	2872	2872	2872	0	0	0.000	5730	5724	5732	8	4	0.070	4253271687	4252081012	4254461472	2380460	691762	0.016
	T	14688	14652	14716	64	12	0.082	6623	6616	6628	12	4	0.060	6634	6628	6640	12	4	0.060	13256	13244	13268	24	5	0.038	1432	1396	1468	64	12	0.838
Sparkle	128-128	3012	2984	3048	64	12	0.298	791	784	798	12	3	0.379	796	788	808	12	3	0.377	1587	1572	1592	20	5	0.315	1426	1404	1456	52	12	0.842
	256-128	4103	4076	4132	56	11	0.268	1338	1332	1344	12	4	0.299	1339	1336	1344	8	4	0.299	2676	2668	2688	20	5	0.187	1428	1400	1456	56	11	0.770
	192-192	4078	4056	4112	56	14	0.343	1321	1316	1328	12	4	0.303	1329	1324	1336	12	4	0.301	2650	2648	2664	24	6	0.226	1429	1404	1456	52	14	0.980
	256-256	5475	5448	5508	52	11	0.201	2020	2012	2020	8	2	0.099	2026	2020	2028	8	3	0.148	4046	4036	4048	12	3	0.074	1430	1400	1456	56	11	0.769
TinyJAMBU	128	3747	3720	3772	52	13	0.341	1156	1152	1164	12	3	0.260	1164	1160	1172	12	3	0.260	2319	2312	2332	20	4	0.172	1428	1396	1456	60	13	0.910
	192	3898	3864	3928	56	9	0.231	1228	1224	1232	8	4	0.326	1236	1232	1240	8	3	0.243	2464	2456	2472	16	5	0.203	1427	1404	1460	56	10	0.701
	256	4039	4012	4056	44	11	0.272	1301	1296	1308	12	4	0.307	1318	1304	1316	12	4	0.305	2611	2608	2624	24	5	0.191	1429	1400	1448	48	11	0.770
Xoodyak		6903	6876	6932	56	10	0.145	2733	2728	2736	8	4	0.146	2741	2736	2744	8	4	0.146	5473	5464	5480	16	5	0.091	1430	1404	1456	52	11	0.769

Figure A6. Results for the Arduino Mega 2560 from the CAN testing for 500 Kbps, without including ID as associated data, using extended IDs. Shown in table format with statistical parameters. Measured "Overall Delay" can differ by a few microseconds from sum of the mean values of "Encryption Time", "Decryption Time", and "Send Time". However, Figure 3 depiction of this summation is reasonably representative of the overall delay, since a few microseconds is insignificant compared to the practical differences in performance between finalist variants.

Appendix G. CAN Testing Results Statistical Table—Mega 2560 Rev3—All as Associated Data

CAN (Authentication-Only) Results' Statistical Variation for Mega 2560 Rev3

Cipher Family	Variant	Overall Delay						Encryption Time						Decryption Time						Cryptography Delay (Enc + Dec)						Send Time (Overall - Crypto)					
		Mean	Min	Max	Range	SD	CV	Mean	Min	Max	Range	SD	CV	Mean	Min	Max	Range	SD	CV	Mean	Min	Max	Range	SD	CV	Mean	Min	Max	Range	SD	CV
ChaChaPoly		7336	7308	7368	60	10	0.136	2950	2944	2952	8	3	0.102	2957	2948	2960	12	3	0.101	5906	5892	5912	20	4	0.068	1430	1400	1460	60	10	0.699
AES-GCM	128	7567	7484	7549	56	12	0.160	3036	3028	3036	8	2	0.066	3042	3036	3044	8	3	0.099	6077	6068	6080	12	3	0.049	1430	1404	1464	60	12	0.839
	192	7783	7756	7808	52	12	0.154	3174	3172	3180	8	3	0.095	3181	3176	3188	12	3	0.094	6354	6348	6368	20	4	0.063	1429	1404	1468	56	12	0.840
	256	5135	5108	5164	56	12	0.234	1856	1844	1864	20	4	0.216	1866	1848	1880	32	5	0.262	3706	3692	3732	40	5	0.135	1429	1408	1456	48	12	0.840
ASCON	128	4761	4732	4780	48	11	0.231	1663	1656	1668	12	4	0.241	1668	1664	1672	8	4	0.240	3331	3320	3348	28	6	0.180	1431	1408	1452	52	12	0.839
	128a	5146	5124	5172	48	12	0.233	1858	1852	1868	8	4	0.215	1859	1852	1860	8	3	0.161	3716	3704	3720	16	5	0.135	1431	1408	1460	52	12	0.839
	80pq	9618	9592	9644	52	12	0.125	4092	4088	4092	4	2	0.049	4097	4096	4100	4	2	0.049	8188	8184	8192	8	3	0.037	1430	1404	1456	52	12	0.839
Elephant	Delirium	5545	5516	5580	64	12	0.216	2058	2048	2068	20	2	0.098	2065	2064	2072	8	2	0.097	4115	4112	4124	12	3	0.073	1431	1400	1464	64	12	0.839
	Dumbo	23811	23788	23844	56	13	0.055	11187	11180	11188	8	3	0.027	11200	11192	11200	8	2	0.018	22386	22376	22388	12	4	0.018	1426	1396	1456	60	13	0.912
	Jumbo	4135	4108	4168	60	11	0.266	1350	1344	1356	12	4	0.296	1355	1352	1360	8	4	0.295	2704	2696	2716	20	5	0.185	1432	1408	1460	52	11	0.768
GIFT-COFB		6959	6924	6972	48	12	0.173	2759	2752	2769	8	3	0.109	2762	2756	2764	8	4	0.145	5520	5508	5524	16	5	0.091	1430	1400	1460	60	12	0.839
Grain-128AEAD		187493	187464	187520	56	12	0.006	93053	93044	93056	12	3	0.003	93058	93052	93060	8	3	0.003	186111	186100	186116	16	4	0.002	1383	1352	1408	56	12	0.868
ISAP	A-128A	30245	30216	30276	60	12	0.043	14408	14404	14416	12	3	0.021	14414	14412	14428	16	3	0.021	28821	28816	28836	20	3	0.010	1425	1388	1448	60	13	0.912
	K-128A	282644	282616	282668	52	12	0.004	140640	140636	140644	8	4	0.003	140646	140640	140652	12	4	0.003	281285	281276	281296	20	5	0.002	1368	1328	1384	56	13	0.956
	A-128	41544	41520	41568	48	10	0.024	20059	20052	20064	12	4	0.020	20064	20060	20068	8	4	0.020	40123	40112	40132	20	6	0.015	1422	1396	1448	52	11	0.774
	K-128	29964	29940	29992	52	11	0.037	14264	14256	14264	8	2	0.014	14278	14272	14280	8	3	0.021	28541	28532	28544	12	4	0.014	1424	1396	1448	52	11	0.772
PHOTON-Beetle AEAD	[128]	6515	6488	6548	60	12	0.169	2541	2536	2544	8	4	0.157	2544	2540	2548	8	4	0.157	5085	5076	5092	16	5	0.098	1431	1404	1456	52	12	0.839
	[32]	11493	11468	11528	60	12	0.104	5031	5024	5032	8	3	0.060	5034	5028	5036	8	3	0.060	10064	10052	10068	16	4	0.040	1429	1400	1460	60	12	0.840
Romulus	N	5325	5296	5348	52	11	0.207	1944	1936	1944	8	2	0.103	1951	1944	1952	8	3	0.154	3894	3880	3896	16	4	0.103	1431	1400	1456	56	11	0.769
	M	5269	5244	5292	48	10	0.190	1917	1908	1920	12	3	0.156	1922	1916	1924	8	3	0.156	3838	3828	3844	16	4	0.104	1431	1404	1452	48	10	0.699
	T	10981	10956	11016	60	12	0.109	4773	4768	4776	8	4	0.084	4780	4772	4784	12	4	0.084	9552	9540	9568	28	5	0.052	1429	1400	1460	60	13	0.910
Sparkle	128-128	3029	3004	3048	44	11	0.363	796	788	808	12	3	0.377	801	796	804	8	4	0.499	1597	1584	1604	20	5	0.313	1433	1404	1456	52	12	0.837
	256-128	4103	4072	4124	52	11	0.268	1334	1332	1340	8	3	0.225	1340	1336	1344	8	4	0.299	2673	2668	2684	16	5	0.187	1430	1400	1456	56	12	0.839
	192-192	4108	4084	4132	48	13	0.316	1337	1332	1344	12	4	0.299	1341	1336	1348	12	4	0.298	2677	2668	2688	20	5	0.187	1431	1404	1456	52	13	0.908
	256-256	5498	5468	5532	64	11	0.200	2038	2024	2032	8	3	0.148	2037	2032	2040	8	2	0.098	4067	4060	4072	12	3	0.074	1432	1408	1460	52	11	0.768
TinyJAMBU	128	4122	4092	4152	60	11	0.267	1344	1340	1348	8	4	0.298	1349	1344	1356	12	4	0.297	2693	2684	2704	20	5	0.186	1430	1404	1464	60	12	0.839
	192	4220	4196	4252	56	11	0.261	1393	1388	1396	8	4	0.287	1398	1392	1404	12	4	0.286	2790	2780	2800	20	5	0.179	1430	1404	1460	56	11	0.769
	256	4322	4300	4344	44	11	0.255	1443	1436	1448	12	4	0.277	1448	1444	1452	8	4	0.276	2891	2880	2900	20	6	0.208	1432	1400	1460	60	12	0.838
Xoodyak		6911	6888	6932	44	11	0.159	2739	2732	2744	12	4	0.146	2743	2736	2748	12	4	0.146	5482	5468	5492	24	5	0.091	1429	1404	1460	56	12	0.840

Figure A7. Results for the Arduino Mega 2560 from the additional CAN testing for 500 Kbps, when processing ID and data field payload all as associated data, when using extended IDs. Shown in table format with statistical parameters. Measured "Overall Delay" can differ by a few microseconds from sum of the mean values of "Encryption Time", "Decryption Time", and "Send Time". However, Figure 3 depiction of this summation is reasonably representative of the overall delay, since a few microseconds is insignificant compared to the practical differences in performance between finalist variants.

References

1. Nolte, T.; Hansson, H.; Bello, L. Automotive Communications-Past, Current and Future. In Proceedings of the 2005 IEEE Conference on Emerging Technologies and Factory Automation, Catania, Italy, 19–22 September 2005; Volume 1, pp. 8–992.
2. Koscher, K.; Czeskis, A.; Roesner, F.; et al. Experimental Security Analysis of a Modern Automobile. In Proceedings of the 2010 IEEE Symposium on Security and Privacy, Oakland, CA, USA, 16–19 May 2010; pp. 447–462.
3. Checkoway, S.; McCoy, D.; Kantor, B.; et al. Comprehensive Experimental Analyses of Automotive Attack Surfaces. In Proceedings of the 20th USENIX Conference on Security, San Francisco, CA, USA, 10–12 August 2011; p. 6.
4. Valasek, C.; Miller, C. Remote Exploitation of an Unaltered Passenger Vehicle. In Proceedings of the Black Hat USA, Las Vegas, NV, USA, 1–6 August 2015.
5. Al-Mekhlafi, Z.G.; Alfhaid, S.A. Innovative Security Measures: A Comprehensive Framework for Safeguarding the Internet of Things. In *AI-Driven: Social Media Analytics and Cybersecurity*; Yafooz, W.M., Al-Gumaei, Y., Eds.; Springer Nature: Cham, Switzerland, 2025; pp. 175–185.
6. Al-Mekhlafi, Z.G. Software-Defined Vehicular Networks (SDVN). *Int. J. Comput. Sci. Netw. Secur.* **2022**, *22*, 231–243. https://doi.org/10.22937/IJCSNS.2022.22.9.33.
7. Lokman, S.F.; Othman, A.T.; Abu-Bakar, M.H. Intrusion Detection System for Automotive Controller Area Network (CAN) Bus System: A Review. *EURASIP J. Wirel. Commun. Netw.* **2019**, *2019*, 184. https://doi.org/10.1186/s13638-019-1484-3.
8. Patil, S.; Jain, S.; Kelkar, S. Enhancing Automotive Security: A Comparative Study of Machine Learning Model for Anomaly Detection in CAN Bus System. In Proceedings of the 2025 IEEE 6th India Council International Subsections Conference (INDISCON), Rourkela, India, 21–23 August 2025; pp. 1–6.
9. Purohit, S.; Govindarasu, M. HAVEN: A Hybrid Anomaly Detection System for Intra-Vehicular CAN-Bus Communication Using Rule-Based and Neural Networks. *IEEE Trans. Dependable Secur. Comput.* **2026**, *23*, 7315–7328. https://doi.org/10.1109/TDSC.2026.3673151.
10. Kumara, S.; Cyril, H.P. Deep Learning-Based Intrusion Detection and Cybersecurity Framework for Connected Vehicle CAN Bus Communication Networks. In Proceedings of the 2026 14th International Symposium on Digital Forensics and Security (ISDFS), Beverly, MA, USA, 19–20 March 2026; pp. 1–6.
11. Tóth, B.M.; Bánáti, A. Hybrid AI-Driven Intrusion Detection System for CAN Bus Networks. In Proceedings of the 2026 IEEE 24th World Symposium on Applied Machine Intelligence and Informatics (SAMI), Stara Lesna, Slovakia, 29–31 January 2026; pp. 000515–000520.
12. Le, T.T.H.; Adiputra, A.A.; Dharmawangsa, A.A.N.; et al. Lightweight CNN-Based Intrusion Detection for CAN Bus Networks. *IEEE Access* **2026**, *14*, 14870–14891. https://doi.org/10.1109/ACCESS.2026.3654521.
13. Frenken, R.; Bhatti, S.G.; Zhang, H.; et al. KD-GAT: Combining Knowledge Distillation and Graph Attention Transformer for a Controller Area Network Intrusion Detection System. In Proceedings of the 2025 IEEE 28th International Conference on Intelligent Transportation Systems (ITSC), Gold Coast, QLD, Australia, 18–21 November 2025; pp. 3721–3726.
14. Ahmed, N.; Babu, M.A.; Mollah, M.M.H.; et al. LRAE: A Low-Rank Autoencoder for Real-Time Efficient CAN Bus Intrusion Detection. In Proceedings of the 2025 12th International Conference on Wireless Networks and Mobile Communications (WINCOM), Riyadh, Saudi Arabia, 25–27 November 2025; pp. 1–6.
15. Amer, K.; Bahaa-Eldin, A.; Sobh, M. Autoencoder-Powered Anomaly Detection: Securing CAN Bus Networks Against Cyber Attacks. In Proceedings of the 2025 12th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI), Semarang, Indonesia, 25–26 September 2025; pp. 764–769.
16. Du, L.; Cheng, J.; Yun, L.; et al. StageDistill-CAN: Lightweight Multi-Stage Distillation for CAN Bus Intrusion Detection. In Proceedings of the 2025 6th International Conference on Internet of Things, Artificial Intelligence and Mechanical Automation (IoTAIMA), Dalian, China, 22–24 August 2025; pp. 215–224.
17. Azeez, B.A.; Dheyaa Radhi, A.; Mahmood, H.; et al. Zero-Day Threat Detection in Autonomous Vehicles Using Few-Shot LSTM Autoencoders on CAN Bus Data. In Proceedings of the 2025 3rd International Conference on Cyber Resilience (ICCR), Dubai, United Arab Emirates, 3–4 July 2025; pp. 1–8.
18. Yu, Z.; Liu, Y.; Li, R.; et al. LIDS: A Lightweight Intrusion Detection System for Controller Area Network. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2025**, *44*, 3303–3312. https://doi.org/10.1109/TCAD.2025.3543431.
19. Turan, M.S.; McKay, K.; Kang, J.; et al. *Ascon-Based Lightweight Cryptography Standards for Constrained Devices: Authenticated Encryption, Hash, and Extendable Output Functions*; Technical Report NIST Special Publication (SP) 800-232; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2025.
20. Copeland, J.; Simpson, L.; Walker, G. Securing CAN Bus Transmissions with Lightweight AEAD Ciphers. In Proceedings of the 2026 Australasian Information Security Conference, Melbourne, VIC, Australia, 11–12 February 2026; pp. 73–85.
21. Robert Bosch GmbH. Company Overview. 2023. Available online: https://www.bosch.com/company/ (accessed on 8 February 2023).
22. Robert Bosch GmbH. *CAN Specification*; Robert Bosch GmbH: Stuttgart, Germany, 1991.
23. CAN in Automation (CiA). History of CAN Technology. 2023. Available online: https://www.can-cia.org/can-knowledge/history-of-can-technology (accessed on 27 March 2023).

24. *LIN Specification Package*; Revision 2.2A; LIN Consortium: Ulm, Germany, 2010.
25. CSS Electronics. LIN Bus Explained—A Simple Intro. 2022. Available online: https://www.csselectronics.com/pages/lin-bus-protocol-intro-basics (accessed on 13 April 2022).
26. Robert Bosch GmbH. *CAN with Flexible Data-Rate*; Robert Bosch GmbH: Stuttgart, Germany, 2012.
27. Robert Bosch GmbH. CAN_XL, CAN XL, CAN, Bosch_CAN, IP-Modules. 2022. Available online: https://www.bosch-semiconductors.com/ip-modules/can-protocols/can-xl/ (accessed on 10 July 2025).
28. Greg Cook. CRC RevEng: Arbitrary-Precision CRC Calculator and Algorithm Finder. 2024. Available online: https://reveng.sourceforge.io/ (accessed on 17 April 2025).
29. Groll, A.; Ruland, C. Secure and Authentic Communication on Existing In-Vehicle Networks. In Proceedings of the 2009 IEEE Intelligent Vehicles Symposium, Xi'an, China, 3–5 June 2009; pp. 1093–1097.
30. Chavez, M.; Rosete, C.; Henriquez, F. Achieving Confidentiality Security Service for CAN. In Proceedings of the 15th International Conference on Electronics, Communications and Computers (CONIELECOMP'05), Puebla, Mexico, 28 February–2 March 2005; pp. 166–170.
31. Herrewege, A.; Singelée, D.; Verbauwhede, I. CANAuth—A Simple, Backward Compatible Broadcast Authentication Protocol for CAN Bus. In Proceedings of the ECRYPT Workshop on Lightweight Cryptography, Louvain-la-Neuve, Belgium, 28–29 November 2011.
32. Ziermann, T.; Wildermann, S.; Teich, J. CAN+: A New Backward-Compatible Controller Area Network (CAN) Protocol with up to 16× Higher Data Rates. In Proceedings of the 2009 Design, Automation & Test in Europe Conference & Exhibition, Nice, France, 20–24 April 2009; pp. 1088–1093.
33. Hartkopp, O.; Reuber, C.; Schilling, R. Full Paper MaCAN—Message Authenticated CAN. In Proceedings of the 10th ESCAR Europe, Berlin, Germany, 28–29 November 2012.
34. Hazem, A.; Fahmy, H.M.A. LCAP—A Lightweight CAN Authentication Protocol for Securing in-Vehicle Networks. In Proceedings of the 10th ESCAR Embedded Security in Cars Conference, Berlin, Germany, 28–29 November 2012.
35. Kurachi, R.; Matsubara, Y.; Takada, H.; et al. CaCAN—Centralized Authentication System in CAN. In Proceedings of the Embedded Security in Cars(ESCAR) Europe 2014, Braunschweig, Germany, 18–19 November 2014.
36. Wang, Q.; Sawhney, S. VeCure: A Practical Security Framework to Protect the CAN Bus of Vehicles. In Proceedings of the 2014 International Conference on the Internet of Things, IOT, Cambridge, MA, USA, 6–8 October 2014; pp. 13–18. https://doi.org/10.1109/IOT.2014.7030108.
37. Wu, W.; Dai, J.; Huang, H.; et al. A Digital Watermark Method for In-Vehicle Network Security Enhancement. *IEEE Trans. Veh. Technol.* **2023**, *72*, 8398–8408. https://doi.org/10.1109/TVT.2023.3247180.
38. Bruton, J. Securing CAN Bus Communication: An Analysis of Cryptographic Approaches. Ph.D. Thesis, National University of Ireland, Galway, Galway, Ireland, 2014.
39. Bella, G.; Biondi, P.; Costantino, G.; et al. Toucan: A protocol to secure controller area network. In Proceedings of the ACM Workshop on Automotive Cybersecurity (AutoSec'19), Richardson, TX, USA, 27 March 2019; Association for Computing Machinery: New York, NY, USA, 2019; pp. 3–8.
40. Hridoy, S.A.A.; Zulkernine, M. LaaCan: A Lightweight Authentication Architecture for Vehicle Controller Area Network. In *Security and Privacy in Communication Networks*; Park, N., Sun, K., Foresti, S., et al., Eds.; Springer International Publishing: Cham, Switzerland, 2020; pp. 215–234.
41. De Santis, F.; Schauer, A.; Sigl, G. ChaCha20-Poly1305 Authenticated Encryption for High-Speed Embedded IoT Applications. In Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE), Lausanne, Switzerland, 27–31 March 2017; pp. 692–697.
42. Radu, A.I.; Garcia, F.D. LeiA: A Lightweight Authentication Protocol for CAN. In *Computer Security—ESORICS 2016*; Askoxylakis, I., Ioannidis, S., Katsikas, S., et al., Eds.; Springer International Publishing: Cham, Switzerland, 2016; pp. 283–300.
43. Nürnberger, S.; Rossow, C. vatiCAN—Vetted, Authenticated CAN Bus. In *Cryptographic Hardware and Embedded Systems—CHES 2016*; Gierlichs, B., Poschmann, A.Y., Eds.; Springer: Berlin, Heidelberg, 2016; pp. 106–124.
44. Wei, Z.; Yang, Y.; Li, T. Authenticated CAN Communications Using Standardized Cryptographic Techniques. In *Information Security Practice and Experience*; Bao, F., Chen, L., Deng, R.H., et al., Eds.; Springer International Publishing: Cham, Switzerland, 2016; pp. 330–343.
45. Van Bulck, J.; Mühlberg, J.T.; Piessens, F. VulCAN: Efficient Component Authentication and Software Isolation for Automotive Control Networks. In Proceedings of the 33rd Annual Computer Security Applications Conference (ACSAC '17), Orlando, FL, USA, 4–8 December 2017; Association for Computing Machinery: New York, NY, USA, 2017; pp. 225–237.
46. CANIS Automotive Labs. *Encryption on CAN Bus: Overview of CryptoCAN*; CANIS Automotive Labs: Cambridge, UK, 2022.
47. Embedded Systems Academy. CANcrypt—Home. 2025. Available online: https://www.cancrypt.net/v1/index.html (accessed on 10 July 2025).
48. Woo, S.; Jo, H.J.; Kim, I.S.; et al. A Practical Security Architecture for In-Vehicle CAN-FD. *IEEE Trans. Intell. Transp. Syst.* **2016**, *17*, 2248–2261. https://doi.org/10.1109/TITS.2016.2519464.

49. Agrawal, M.; Huang, T.; Zhou, J.; et al. CAN-FD-Sec: Improving Security of CAN-FD Protocol. In *Security and Safety Interplay of Intelligent Software Systems*; Hamid, B., Gallina, B., Shabtai, A., et al., Eds.; Springer International Publishing: Cham, Switzerland, 2019; pp. 77–93.
50. Montilla, A.A. *Encrypted and Secure Messages with CAN FD*; Technical Report; Universitat Politècnica de Catalunya: Barcelona, Spain, 2021.
51. Groza, B.; Murvay, S.; Herrewege, A.V.; et al. LiBrA-CAN: Lightweight Broadcast Authentication for Controller Area Networks. *ACM Trans. Embed. Comput. Syst.* **2017**, *16*, 90. https://doi.org/10.1145/3056506.
52. Oberti, F.; Savino, A.; Sanchez, E.; et al. CAN-MM: Multiplexed Message Authentication Code for Controller Area Network Message Authentication in Road Vehicles. *IEEE Trans. Veh. Technol.* **2024**, *73*, 14661–14673. https://doi.org/10.1109/TVT.2024.3402986.
53. Wiemer, F.; Zeh, A. Enabling Secure Communication for Automotive Endpoint-ECUs through Lightweight-Cryptography. In Proceedings of the 7th ACM Computer Science in Cars Symposium (CSCS '23), Darmstadt, Germany, 5 December 2023; Association for Computing Machinery: New York, NY, USA, 2023.
54. Rasheed, A.; Baza, M.; Badr, M.; et al. Efficient Crypto Engine for Authenticated Encryption, Data Traceability, and Replay Attack Detection over CAN Bus Network. *IEEE Trans. Netw. Sci. Eng.* **2023**, *11*, 1008–1025. https://doi.org/10.1109/TNSE.2023.3312545.
55. National Institute of Standards and Technology. *Submission Requirements and Evaluation Criteria for the Lightweight Cryptography Standardization Process*; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2018.
56. Information Technology Laboratory—Computer Security Division—National Institute of Standards and Technology. Lightweight Cryptography. 2022. Available online: https://csrc.nist.gov/Projects/lightweight-cryptography (accessed on 8 February 2022).
57. Dobraunig, C.; Eichlseder, M.; Mendel, F.; et al. Ascon v1.2. 2021. Available online: https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/ascon-spec-final.pdf (accessed on 18 April 2022).
58. *NIST Selects 'Lightweight Cryptography' Algorithms to Protect Small Devices*; NIST: Gaithersburg, MD, USA, 2023.
59. Siddiqui, A.S.; Lee, C.C.; Che, W.; et al. Secure Intra-Vehicular Communication over CANFD. In Proceedings of the 2017 Asian Hardware Oriented Security and Trust Symposium (AsianHOST), Beijing, China, 19–20 October 2017; pp. 97–102.
60. Ferguson, N. Authentication Weaknesses in GCM. 20 May 2005. Available online: https://csrc.nist.gov/CSRC/media/Projects/Block-Cipher-Techniques/documents/BCM/Comments/CWC-GCM/Ferguson2.pdf (accessed on 16 June 2026).
61. Computer Security Division, Information Technology Laboratory. NIST to Revise Special Publication 800-38D | Galois/Counter Mode (GCM) and GMAC Block Cipher Modes 2024. Available online: https://csrc.nist.gov/News/2024/nist-to-revise-sp-80038d-gcm-and-gmac-modes (accessed on 16 June 2026).
62. Dworkin, M. *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*; Technical Report NIST Special Publication (SP) 800-38D; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2007.
63. Kim, H.; Seo, H. Optimizing AES-GCM on ARM Cortex-M4: A Fixslicing and FACE-Based Approach. Cryptology ePrint Archive 2025. Paper 2025/512. Available online: https://eprint.iacr.org/2025/512 (accessed on 10 July 2025).
64. Wu, H.; Huang, T. TinyJAMBU: A Family of Lightweight Authenticated Encryption Algorithms (Version 2). 17 May 2021. Available online: https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/tinyjambu-spec-final.pdf (accessed on 18 April 2022).
65. Microchip Technology. MCP2515. 2023. Available online: https://www.microchip.com/en-us/product/MCP2515# (accessed on 17 February 2023).
66. Lightweight-Cryptography-Benchmarking/Benchmarks at Main · Usnistgov/Lightweight-Cryptography-Benchmarking. 2025. Available online: https://github.com/usnistgov/Lightweight-Cryptography-Benchmarking/tree/main/benchmarks (accessed on 7 Feburary 2025).
67. Weatherly, R. LWC Finalists. 2022. Available online: https://github.com/rweather/lwc-finalists (accessed on 16 February 2023).
68. Mistry, S. CAN. 2023. Available online: https://docs.arduino.cc/libraries/can/ (accessed on 16 February 2023).
69. ATMEGA4809. 2026. Available online: https://www.microchip.com/en-us/product/ATMEGA4809 (accessed on 19 June 2026).
70. ATMEGA328P. 2023. Available online: https://www.microchip.com/en-us/product/atmega328p (accessed on 25 July 2023).
71. Nano Every. 2023. Available online: https://docs.arduino.cc/hardware/nano-every (accessed on 16 February 2023).
72. UNO R3. 2023. Available online: https://docs.arduino.cc/hardware/uno-rev3 (accessed on 16 February 2023).
73. Mega 2560 Rev3. 2023. Available online: https://docs.arduino.cc/hardware/mega-2560 (accessed on 16 February 2023).
74. ATmega2560. 2023. Available online: https://www.microchip.com/en-us/product/atmega2560 (accessed on 25 July 2023).
75. Recommended for Automotive Designs. 2025. Available online: https://www.microchip.com/en-us/solutions/automotive-and-transportation/recommended-for-automotive (accessed on 9 February 2025).
76. Turan, M.S. *Status Report on the Final Round of the NIST Lightweight Cryptography Standardization Process*; Technical Report NIST IR 8454; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2023.
77. Rhys Weatherly. Performance on 32-Bit Platforms. 2025. Available online: https://rweather.github.io/lwc-finalists/performance.html (accessed on 7 February 2025).

78. Weatherley, R. Additional Modes for LWC Finalists Technical Report, Version 1.0. 2021. Available online: https://github.com/rweather/lwc-finalists/blob/master/doc/lwc-modes-v1-0.pdf (accessed on 21 June 2026).
79. Garnatz, O.; Decker, P. Change in Automotive Communication Systems. *CAN Newsletter*, December 2020.
80. December 2020: CAN XL. 2025. Available online: https://www.can-cia.org/services/publications/can-newsletter-magazine/december-2020-can-xl (accessed on 10 July 2025).
81. Poudyal, S.; Morozov, K. In-Vehicle Communication Security: Testing Real-Life Data. In Proceedings of the 2024 IEEE International Conference on Mobility, Operations, Services and Technologies (MOST), Dallas, TX, USA, 1–3 May 2024; pp. 235–241.