Highlights

Robust Authentication for Automotive Control Networks through Covert Channels[1]

Stien Vanderhallen, Jo Van Bulck, Frank Piessens, Jan Tobias Mühlberg

- *We explore novel defensive uses of covert-channel communication in automotive control networks, specifically in Controller Area Networks (CAN):*
- We argue for the use of timing-based covert communication in automotive control networks, and evaluate qualitative characteristics of a practical timing channel implementation on CAN.
- We present a timing-based nonce synchronisation scheme for message authentication in VulCAN, improving robustness against message loss.
- We evaluate the security properties of our design and develop an argument showing that our scheme does not harm VulCAN's security guarantees in the presence of a powerful network-level attacker.
- We conduct the first comprehensive exploration of covert, and covert-like, bandwidth sources applicable to CAN.

[1]The published version of this article is available at `https://doi.org/10.1016/j.comnet.2021.108079`.

Robust Authentication for Automotive Control Networks through Covert Channels

Stien Vanderhallen[a], Jo Van Bulck[a], Frank Piessens[a], Jan Tobias Mühlberg[a]

[a]*KU Leuven, Dept. Computer Science, imec-DistriNet, Celestijnenlaan 200a, Leuven, 3001, Belgium,*

Abstract

Automotive control networks offer little resistance against security threats that come with the long-range connectivity in modern cars. Remote attacks that undermine the safety of vehicles have been shown to be practical. A range of security mechanisms have been proposed to harden resource-constrained embedded microcontrollers against malicious interference, including cryptographic protocols that establish the authenticity of in-vehicle message exchange. However, authenticated communication comes with repercussions on deployability and vehicle safety in terms of reliability, real-time compliance, backwards compatibility, and bandwidth and resource use.

In this article we investigate benign, defensive uses of *covert channels* to implement and support vehicular message authentication mechanisms as a transparent, resource-conserving approach to automotive network security. We provide the first comprehensive evaluation of covert channels in Controller Area Networks (CAN) with respect to the attainable bandwidth and reliability of covert communication. Our analysis identifies timing-based covert channels as candidates to design a *complementary* nonce synchronisation channel that can enhance robustness against message loss in existing authentication schemes. We practically implement and evaluate this design on top of an open-source authenticated CAN communication library, showing that covert timing channels can improve communication robustness in benign circumstances, while not reducing the security guarantees of the underlying

Email address: jantobias.muehlberg@cs.kuleuven.be (Jan Tobias Mühlberg)

The published version of this article is available at https://doi.org/10.1016/j.comnet.2021.108079.

authentication primitives when under attack.

Keywords: covert channels, message authentication, trusted computing, Controller Area Network, automotive security

---

1. Introduction

Information leakage through side channels is long known to be exploitable in practical attacks [1, 2] across all kinds of systems. In the context of connected vehicles, this information leakage has been identified as a threat to user privacy [3]. Experimental exploits aim at tracking vehicles [4], attack key exchange protocols [5], or compromise the security of commercial immobiliser systems [6]. In this paper, however, we develop a novel *defensive* use case for the "hidden" bandwidth provided by side channels, which we use to support authentication schemes for automotive control networks.

Automotive Control Networks. Modern-day vehicles are equipped with an unprecedented amount of embedded Electronic Control Units (ECUs), that regulate and enhance the experience of both drivers and passengers. Controller Area Networks (CANs) serve to connect such in-vehicle components and are widely used by the automotive industry, as well as in building automation, factory control and agriculture. Given those application domains, CAN often plays a major role in safety-critical, real-time sensitive functions, such as emergency braking and parking assistance.

Therefore, CAN network security in the form of, e.g., access control and message authentication, was sacrificed for performance and timeliness. This is a sensible approach when assuming CAN networks to be inaccessible beyond vehicle boundaries. However, the rise of on-board systems for entertainment, communication and navigation broke that assumption, by connecting a subset of embedded vehicle components not only to CAN, but also to wider-reaching networks, including the Internet. As such, remotely accessible vectors for penetrating CAN networks were introduced, which resulted in a wide range of practical attack demonstrations against CAN networks and the connected ECUs [7, 8, 4, 9]. Those findings show that, given the safety-critical functions performed by ECUs, major harm can follow from these devices' CAN communication being tampered with.

Authentication in CAN. The unique constraints of the automotive industry, in terms of resource utilisation and backwards-compatibility requirements,

have prompted researchers to propose a multitude of lightweight cryptographic protocols for transmitting customised Message Authentication Codes (MACs) on the CAN bus [10, 11, 12]. Moreover, the use of embedded Trusted Execution Environments (TEEs) has recently been put forward as a promising solution to protect software and cryptographic key material on participating ECUs [13, 14]. For the purpose of our research, VulCAN [13] is particularly interesting because it presents a light-weight open-source library implementation of multiple CAN authentication protocols for a readily available IoT TEE [15]. Despite these strong security guarantees, however, existing message authentication measures remain impractical due to increased CAN bus activity and the possibility of message loss, thereby weakening the real-time and reliability properties of safety-critical in-vehicle functionality.

A challenging trade-off thus exists for CAN applications, which have to consider automotive safety requirements from largely incompatible performance and security angles. TACAN (Transmitter Authentication in CAN, [16]) recently suggested to somewhat alleviate resource demands by transmitting authentication messages via *covert channels*. Such non-conventional communication forms exploit behavioural properties of regular, bandwidth-consuming traffic for their information carrier, and consequently incur no extra network load for transmitting data. As such, when employed for defensive purposes, covert channels hold the compelling potential of transparently implementing security mechanisms without sacrificing scarcely available bandwidth. TACAN's immediate use of covert authentication suffers from several major limitations, however. Most importantly, TACAN does not aspire to be a full authentication solution, but only pursues a significantly weaker form of continuous "transmitter authentication". That is, in the face of severe restrictions regarding the available bandwidth and reliability of covert communication, TACAN cannot guarantee the authenticity of individual message payloads and only enables a trusted monitor node to periodically establish that sender nodes have access to a certain key by means of heavily truncated cryptographic digests.

Our Contributions. In this paper, we refine the challenges and opportunities for using covert communication in benign applications. Specifically, we move beyond using covert channels as a mere drop-in replacement for existing CAN authentication schemes, and we develop a more innovative and suitable *composite* usage where covert bandwidth is leveraged not to replace but rather to complement existing authentication protocols. After comprehensively sur-

veying covert-channel opportunities in CAN, we conclude that inter-arrival time manipulation provides the most tempting characteristics for software-defined and application-independent covert transmission. We then design such a timing-based covert channel that enhances existing MAC-based authentication schemes, and we ensure the use of that covert channel to be *(1) secure*, in the sense that authentication cannot be bypassed even when the covert channel is completely controlled by the adversary, *(2) optional*, in the sense that authentication remains functional even under circumstances where the covert channel completely fails, but *(3) advantageous*, in the sense that quality-of-service improves by reducing the number of authentication failures caused by (possibly non-malicious) message loss.

Concretely, we use a timing-based covert channel to extend and improve nonce synchronisation in the previously published vatiCAN [11] message authentication protocol. Our practical implementation is conceived as a pluggable backend extension to the open-source VulCAN [13] authentication library. Crucially, by using covert bandwidth to encode partial nonce bits, we can entirely dispose of vatiCAN's global nonce generator component, which has previously been shown to be vulnerable to practical replay attacks based on the birthday paradox [13]. To that end, we first present and evaluate a covert channel based on packet inter-arrival time manipulation on the CAN bus. Next, we migrate that timing channel to the specific application context of nonce synchronisation in VulCAN's vatiCAN backend, and we argue that our approach improves robustness without compromising the pursued security guarantees. In summary, we make the following contributions:[2]

1. We argue for the use of timing-based covert communication in automotive control networks, and evaluate qualitative characteristics of a practical timing channel implementation on CAN.

2. We present a timing-based nonce synchronisation scheme for message authentication in VulCAN, improving robustness against message loss.

3. We evaluate the security properties of our design and develop an argument showing that our scheme does not harm VulCAN's security guarantees in the presence of a powerful network-level attacker.

[2]All experiments will be made open-source available upon acceptance. The code is available for reviewers on request through the program chairs.

4. We conduct the first comprehensive exploration of covert, and covert-like, bandwidth sources applicable to CAN.

Outline. Section 2 provides background, and Section 3 defines the problem, goals, and requirements. We develop a practical timing channel instance in Section 4. Section 5 presents our design for timing-based nonce synchronisation in VulCAN, which is analysed from a performance and security perspective in respectively Sections 6 and 7, and discuss limitations and future work in Section 8. Finally, in Section 9, we present an overview of covert-channel opportunities for CAN, and draw conclusions in Section 10.

2. Background and Related Work

This section positions our work in its context of previous research on CAN security threats, message authentication, nonce synchronisation, and covert channels. From the combination of those fields emerges the purpose of our work, *i.e.*, improving CAN application security by repurposing side channels, traditionally known as attack vectors, as software-controlled, supplementary bandwidth for benign use.

2.1. Controller Area Network

CAN protocol design. The Controller Area Network (CAN) protocol [17] is used by, amongst others, the automotive industry for in-vehicle communication. It serves to connect the electrical components regulating, e.g., a vehicle's brakes to its parking sensors, or its dashboard velocity meter to its wheels. Participating microcontrollers, typically referred to as Electrical Control Units (ECUs), interface the CAN bus by means of dedicated CAN transceiver hardware chips.

To fit this context of mostly embedded computing devices taking part in CAN communication, CAN is built upon a broadcast medium without an addressing mechanism, thus relieving from any overhead related to connection management and/or network joining and leaving. Moreover, ECUs are not synchronised in their access to a CAN bus, meaning all connected ECUs can both transmit on and read from their CAN bus at any time. Preventing from unstable bus behaviour due to this design, a 0-bit is deemed dominant, which means its transmission overwrites a (recessive) 1-bit. To accommodate such design, CAN nodes halt frame transmission on detecting unwanted overwriting of their own communication.

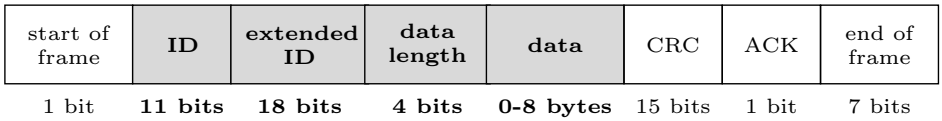


Figure 1: CAN 2.0 data frame fields. Software-controlled fields are indicated in grey, others are controlled in the transceiver hardware.

A simplified overview of a CAN frame is given in Figure 1. Importantly, the data payload field has a fixed maximal length of 8 bytes (64-bit), severely restricting the amount of information that can be transmitted in a single CAN frame. As indicated, an *ID field* is further leveraged as an addressing mechanism for associating messages to their application and/or sender. It measures 11 bits in original CAN, and is extended to 29 bits in CAN 2.0 [17]. Given the CAN arbitration scheme of 0-bit transmissions being dominant, the ID field moreover serves to denote its message's priority. Indeed, the lower a frame's ID, the higher chance it has of retaining bus occupation until transmission of its data payload.

CAN security threats. Multiple network attacker capabilities were described to be enabled by both the nature of the CAN design, and its increasingly remote accessibility [7, 18, 19]. Their exploitation requires gaining control over the transmission logic of an ECU connected to the targeted CAN bus, either through the physical attachment of a rogue node [8], or a remote code injection attack [4, 20, 21] that leverages external vehicle accessibility. Below, the main capabilities enabled by such malicious ECU control are listed.

Eavesdropping. As CAN uses a broadcast medium that is not equipped with access control, ECUs are free to record and inspect all traffic on the CAN bus they are connected to. CAN moreover provides no confidentiality mechanism in its design, which means malicious nodes receive the same plaintext data as the benign nodes they are eavesdropping on.

Message insertion. As CAN provides no mechanisms for bus access control and/or transmitter authentication, a malicious ECU can inject a frame carrying any ID and data payload on its bus, and have it be successfully received by other nodes on that bus, if that message conforms to the CAN frame format and its CRC field carries a suitable checksum.

Message deletion. Any 1-bit transmission on a CAN bus can be overwritten by a 0-bit, and CAN nodes stop message transmission whenever they detect such overwriting. Therefore, an appropriately timed 0-bit transmission by a malicious ECU can cause deletion of any CAN message.

2.2. Message Authentication in CAN

CAN continues to be heavily used in the automotive industry. In the face of the CAN security threats discussed above, several proposals have been made to weaken network attackers in their capabilities, and harden software modules against compromise. Cryptographic message authentication protocols form a subset of those efforts, and are of specific interest for this work.

Message authentication is performed on each frame transmission, and takes the frame at stake as a parameter. Concretely, when authenticating a message, its sender calculates a Message Authentication Code (MAC) over *(1)* the message itself, *(2)* a secret, and *(3)* a nonce value providing freshness, and transmits that MAC along with the message at hand. Receiver parties then verify the identity of a packet's origin through validating such MACs to be calculated over an appropriate message, secret and nonce.

Authentication Protocols. Several message authentication protocols were developed in academia specifically for CAN, including vatiCAN [11], LiBrA-CAN [22], CANAuth [12] and LeiA [10]. The main common factor amongst those protocols is their aim for backwards compatibility, *i.e.*, CAN nodes not supporting their form of message authentication should not break when other components on their CAN bus enable authenticated communication. The industry standard body AUTOSAR [23] formulates guidelines on backwards compatible in-vehicle message authentication, which are complied to by vatiCAN [11] and LeiA [10]. There are additional protocols with focus on establishing cryptographic (session) keys between ECUs, e.g., [24, 25].

VulCAN [13] is an open-source CAN authentication suite, including practical implementations of both the vatiCAN and LeiA protocols. Message authentication in VulCAN is transparently performed by accompanying each application message with a subsequently sent authentication message that carries a truncated 64-bit MAC, calculated with a 128-bit symmetric cryptographic key as recommended by AUTOSAR [23]. MAC-calculation and -transmission details are governed by the respective vatiCAN or LeiA backends. VulCAN furthermore supports efficient compilation to embedded Sancus enclaves [26, 15] so as to shield application logic and cryptographic key

material from potentially compromised firmware on the target ECU. For this, VulCAN leverages the strong, hardware-level cryptographic primitives and memory isolation guarantees provided by the Sancus TEE in an automotive context to efficiently enable authenticated CAN communication, as well as software attestation and isolation. Notably, the VulCAN design explicitly excludes the operating system, networking, and I/O interaction software from its Trusted Computing Base (TCB), thus allowing for both CAN traffic and unprotected ECU application logic to be taken over by an attacker while retaining its security guarantees.

Nonce Synchronisation. In message authentication, *nonces* (or counters) are crucial to provide message freshness. More specifically, a nonce, is associated with each authenticated message, and used as a parameter in both the calculation and validation of that message's MAC. As opposed to the secret key used as another parameter in MAC calculation, nonces can safely be disclosed. Indeed, obtaining the nonce associated with a frame does not suffice for an attacker to construct a valid corresponding MAC, as it is assumed not to have access to the associated secret.

By using a *unique* nonce for each message, any two valid MACs are calculated using a different nonce, meaning they are unlikely to be equal, depending on MAC algorithm strength. As such, valid MACs are guaranteed to correspond to *fresh* messages, *i.e.*, messages that were not yet authenticated and/or processed as such. Therefore, attackers incapable of producing valid MACs themselves cannot successfully resort to replaying authenticated traffic on a CAN bus and trick receivers into accepting it as such.

Since large nonces cannot be simply sent along with the corresponding MAC in the limited-sized CAN frame layout, nonces pose a non-trivial challenge in synchronising sending and receiving ECUs. While some authentication protocols [11] rely on *implicit* nonce synchronisation, where the sender and receiver ECUs each increment their own local nonce counter on respectively every message transmission or arrival, such an approach cannot recover from the likely scenario of message loss. Indeed, in the case of message loss, sender and receiver nonces get out-of-sync and all subsequent MAC verifications will fail. Alternative solutions anticipate packet loss by including (the lower bits of) the nonce in some of the fields of the authentication frame itself. However, such solutions invariably have to sacrifice MAC strength [16] or backwards compatibility [10], since the standard CAN frame does not leave any remaining software-controlled bits free.

2.3. Covert Channels

This work defines a covert channel as *data transmission that (1) for its payload carrier leverages some behavioural property of an underlying form of non-covert communication (here, CAN traffic) and (2) in exploiting that carrier does not require write access to non-covert traffic data objects (here, CAN frames).* We consider only network-level covert channels, deeming hardware-level behaviours out of scope.

Most literature on covert channels presents the concept as hiding transmission from external parties by "covering" it in regular traffic, as stated in our definition. Beyond that common ground however, little consensus exists on a detailed covertness definition, in that certain channels are considered covert by some authors, and non-covert by others. For instance, TACAN [16] proposes to overwrite CAN frame Least Significant Bits (LSBs) for transmitting "covert" data. A subset of existing research allows for covert channels to overwrite overt traffic as long as it goes unnoticed in application logic [27, 28, 29], thus categorising LSB manipulation as a covert channel in applications that ignore a non-zero number of LSBs. Our definition however prohibits packet manipulation regardless of the application logic at hand, which follows a different group of authors [30, 31], amongst which Lampson [31] in the original definition of covert channels.

Timing/storage covert channels. Covert channels are typically divided into two subclasses. *Timing channels* use packet timings for their covert payload carrier, whereas *storage channels* use memory locations, or data objects, that all participating nodes can access. Our definition does not explicitly mention those categories, yet allows for similar subclasses in covert communication. Manipulation of a frame's timing can be done regardless of its contents, which adequately qualifies packet timing as a covert payload carrier. Therefore, all timing channels applicable to CAN are considered covert here, such as the packet Inter-Arrival Time (IAT) channel proposed by TACAN [16]. In contrast, no network-level storage channels introduced in previous work meet our covertness requirements, such as the LSB manipulation channel discussed earlier. Timing channels have been discussed in the context of the Internet-of-Things before [32], where the authors show that the detectability of the channel is proportional to the channel's reliability. Since our approach aims at using covert bandwidth to transfer information that is not confidential detectability of the channel is not a concern in our work.

Security implications. Most known covert channels are leveraged for malicious purposes, more specifically for leaking confidential information. Some authors [31, 29] explicitly require the sending party in a covert channel to have a privilege level different from its receiver's, to enable security policy bypass through covert communication. We pose no requirements on the security privileges of participants, however, as we innovatively leverage covert communication to enforce defensive mechanisms, rather than to bypass them.

3. Problem Statement

From the previous section, we conclude that nonce synchronisation poses an important and reoccurring challenge for CAN authentication solutions, where message sizes are strictly limited, network bandwidth is scarce, and packet loss is likely to occur in realistic deployments. We first argue below why authentication solutions recently proposed in the literature do *not* provide satisfactory solutions to the nonce synchronisation challenge. Next, we formulate requirements for practical and secure nonce synchronisation, and we define the system model and attacker capabilities.

3.1. Motivation

vatiCAN limitations. Nonces cannot be included in the limited 64-bit vatiCAN authentication packets. All sender and receiver ECUs are instead expected to maintain a local 32-bit nonce counter [11], which is incremented for every successful message transmission or arrival, respectively. However, in case of packet loss, sender and receiver nonces may get out-of-sync, leading all subsequent MAC verifications to fail after being computed with the wrong nonce. To somewhat mitigate this scenario and allow authenticated communication to eventually recover after message loss, vatiCAN proposes the use of a trusted global Nonce Generator (NG) component that periodically synchronises nonces in the entire network. Particularly, NG periodically broadcasts a randomly generated value that should be used as the new initial nonce value by all participants on the CAN bus for that era.

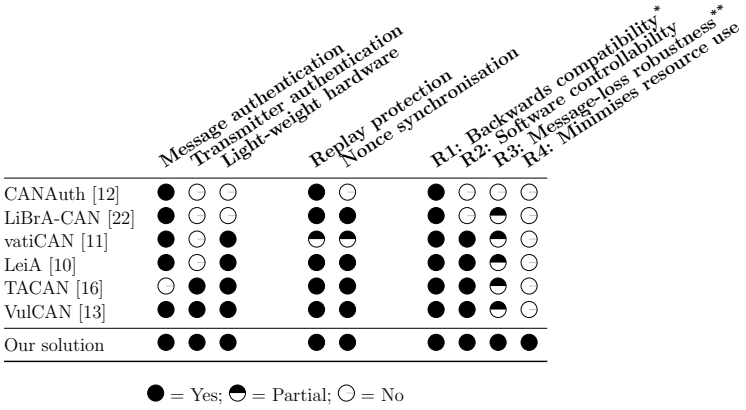
However, NG's periodic nonce reset scheme has been shown to render vatiCAN susceptible to advanced replay attacks based on the birthday paradox [13]. Depending on the application and NG configuration, as little as 30 minutes of recorded CAN traffic may suffice to successfully replay a safety-critical authenticated message [13]. VulCAN therefore excludes NG from its vatiCAN backend, trading robustness against message loss for security.

LeiA limitations. LeiA [10] offers a more tempting solution to message loss recovery by directly embedding nonce bits in the 18-bit extended CAN identifier field. To compensate for the relatively small nonce size, LeiA includes larger 64-bit epoch counters that can be used to refresh session keys on nonce counter overflow. As every authentication message now carries the associated nonce in its identifier field, receivers can straightforwardly recover after loosing one or more sender messages. Furthermore, to recover from longer-term network outages, LeiA includes an elaborate error-frame protocol to resynchronise sender and receiver epoch counters.

Crucially, however, sacrificing extended CAN identifiers to encode nonce bits breaks backwards compatibility with real-world applications relying on those. Even apart from backwards compatibility concerns, extended CAN identifiers also come with a considerable performance cost. Namely, in the first actual implementation of the LeiA specification, VulCAN [13] reports considerable runtime overheads (in the order of 18 %) for sending a legacy CAN message vs. one with an extended CAN 2.0 identifier. This is due to the increased interaction with the CAN transceiver hardware and may likely also affect other considerations such as power consumption.

TACAN limitations. The TACAN [16] protocol does not provide strong message authentication, but rather pursues a significantly weaker form of "transmitter authentication". Particularly, successful authentication with TACAN only entails that the sender ECU has access to a certain (TPM-backed) key, which does not exclude that the adversary has manipulated the transmitted messages at creation time or in-flight, or even has code execution on the target ECU. TACAN relies on a global trusted monitor node to act as the receiving end of a covert channel and periodically extract a covert authentication payload from the aggregated stream of sender messages (e.g., by means of the relative timing of the individual messages). Every TACAN authentication payload consists of a nonce counter, which is increased for every covered transmission era, followed by a MAC over that nonce to prove possession of a symmetric cryptographic key. Due to the limited covert bandwidth available, the MACs have to be heavily truncated at the cost of the resulting security level (e.g., TACAN's evaluation only considers 36-bit authentication messages, of which only 12 bits are derived from a cryptographic digest).

Table 1: A summary of our proposal in comparison with related work, reflecting system properties discussed in Section 2 and Section 3. We have grouped system properties *(1)* general protocol features, *(2)* the presence of replay protection, and *(3)*, our requirements from Section 3. — *Footnotes* — *: Backwards compatibility means that ECUs that support authentication can still interact with legacy ECUs. However, bus congestion and the use of arbitration IDs may render the scheme incompatible with existing deployments. **: Systems with partial message-loss robustness recover from individual message loss through nonce-resynchronisation steps that degrade performance.



3.2. System Requirements

Motivated by limitations in existing work on nonce synchronisation and benign uses of covert channels, we identified system requirements listed below. In Table 1 we provide an overview of related work and our requirements.

R1: Backwards compatibility. We require our approach to not affect regular vatiCAN functionality, in that ECUs implementing our design can participate in vatiCAN-authenticated communication with devices that do not. As we aim for an extension to VulCAN, we require it not to break backwards compatibility with applications using original VulCAN.

R2: Software controllability. To ensure flexibility and portability of our design, we require it to affect software-controlled properties of CAN only.

R3: Message-loss robustness. Both benign and malicious message loss are inevitable in CAN. We therefore aim to propose a solution that increases vatiCAN's availability guarantees in the event of message loss.

R4: Minimised resource use. In our context of embedded computing, resources are scarce. Therefore, we aim for our design to require as little ECU energy consumption and transmission latency as possible, and do not allow it to incur supplementary CAN bus congestion.

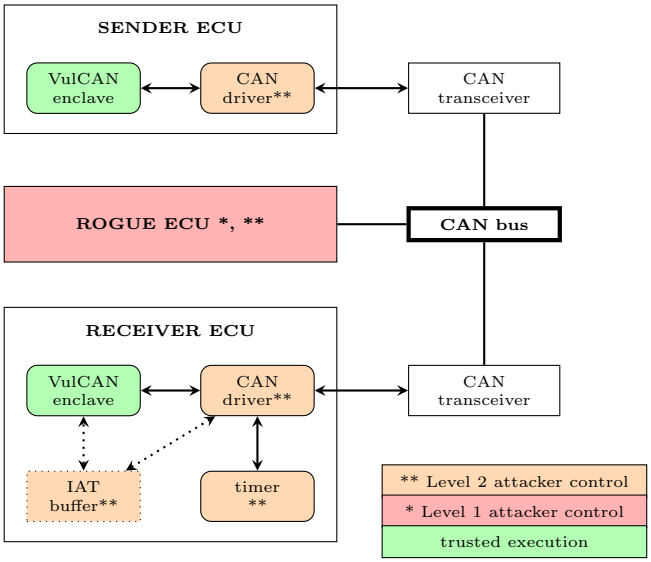


Figure 2: System model for covert nonce synchronisation in VulCAN.

3.3. System Model

The system model for our our design is graphically presented in Figure 2. It consists of "vulcanised" sender and receiver nodes, *i.e.*, Sancus-enabled ECUs that implement VulCAN. Communication proceeds over a vatiCAN [11] authenticated connection, which is built upon a CAN bus that is also accessible to rogue ECUs. As our design is formulated as an extension to VulCAN [13], both the sender and the receiver are assumed to govern authenticated CAN communication through an appropriately extended VulCAN library that executes in a trusted Sancus enclave [15]. Their respective CAN driver software and CAN transceiver hardware lie outside that trusted environment, and are thus deemed under attacker control. To accommodate timing-based communication, the receiver ECU furthermore provides an untrusted timer peripheral, which serves to measure authentication frame timings, and an IAT buffer in which those measurements are stored.

3.4. Attacker Profiles and Security Objective

Attacker Level 1: Network attacker. Our Level 1 attacker model considers a malicious party that, either via physical access or remote code injection [4, 20], can take over a non-vulcanised ECU for deleting and/or inserting CAN frames on the common bus. This is the traditional adversary considered

by CAN message authentication solutions [10, 11] and is presented as a rogue ECU in Figure 2.

Attacker Level 2: Software attacker. Going beyond the Level 1 profile, VulCAN [13] protects even against a more powerful attacker that is capable of arbitrary code execution in all software modules that are not Sancus-protected. Consequently, our Level 2 attacker controls both non-vulcanised ECUs like the Level 1 attacker does, and all non-enclave software on vulcanised ECUs. Notably, the VulCAN design explicitly excludes CAN driver software from its trusted computing base, meaning that CAN driver operations are to be considered untrusted and under Level 2 attacker control. With such driver logic propagating all CAN traffic to and from VulCAN software, overtaking it yields this attacker an ideal man-in-the-middle position in CAN communication. The IAT buffer, and timer mechanism provided at receiver side moreover lie outside Sancus protection as well, rendering the timing values they store, respectively produce, under Level 2 attacker control.

Security objective. In formulating our design, we seek to arrive at a practical solution which can provide strong authentication guarantees for individual messages (vs. TACAN), while additionally being able to detect and recover from incidental message loss without requiring to sacrifice extended CAN identifiers (vs. LeiA) or to introduce a vulnerable global nonce generator component (vs. vatiCAN). For this, we extend the vatiCAN backend in the VulCAN library with an innovative covert nonce synchronisation mechanism which offers security guarantees similar to LeiA [10]. Moreover, we will argue for our design not harming VulCAN's security guarantees in the face of both attacker profiles listed above.

4. Timing-Based Covert Communication in CAN

As emerges from Section 9, the opportunities for truly covert transmission in CAN that conform to system requirements **R2** and **R4** reside in timing-based communication. Therefore, and as a prelude to our proposed design, this section provides with an in-depth analysis of the **T1** timing channel.

4.1. Channel Design

The timing channel considered here corresponds to the **T1** channel introduced in Section 9. Concretely, it transmits covert data by adjusting the originally equal-sized time intervals between non-covert, periodic CAN

messages, similar to a covert channel considered by TACAN [16]. At sender side, packet inter-transmission times are modified accordingly, and receiver-side logic monitors message inter-arrival times before decoding them to the appropriate covert payload. This section discusses the core concepts constituting this channel and its implementation in more detail.

Sender side. The sender-side encoding of a covert bit b_{send} to an appropriate inter-transmission time t_{ITT} here is done using a deviation from the underlying message stream period T with one δ offset parameter, as encoded in Equation (1). Note that a deviation of multiple δ offsets could be used as well. Such an approach allows for more than one covert bit to be encoded in every inter-transmission time, as a trade-off for larger real-time effects.

$$t_{ITT} = \begin{cases} T + \delta & if\ b_{send} = 1 \\ T - \delta & if\ b_{send} = 0 \\ T & otherwise \end{cases} \tag{1}$$

Receiver side. The receiver-side decoding of an observed packet inter-arrival time t_{IAT} to a corresponding covert payload $b_{receive}$, here is implemented to follow the decoding of Equation (2), which matches the encoding of Equation (1). To maximise accuracy, IAT value registration at receiver side can be done by configuring the CAN transceiver chip to fire an interrupt on packet arrival and measuring the timing interval elapsed since the last message arrival in the corresponding Interrupt Service Routine (ISR).

$$b_{receive} = \begin{cases} 1 & if\ \ t_{IAT} > T + \delta/2 \\ 0 & if\ \ t_{IAT} < T - \delta/2 \\ \_ & otherwise \end{cases} \tag{2}$$

4.2. Channel Evaluation

Implementation technology. An implementation of this timing channel was done using two 16-bit MSP430 microcontrollers [33], both connected over an SPI-interface to their own MCP2515 CAN transceiver [34]. One microcontroller implements receiver-side functionality, the other takes on the role of a sender. Both connect to the same CAN bus via their respective CAN controllers. The bus is configured to run at 500 kbps in all experiments. When introducing bus noise for performance assessment purposes, USBtin [35] hardware is used and attached to that same CAN bus.

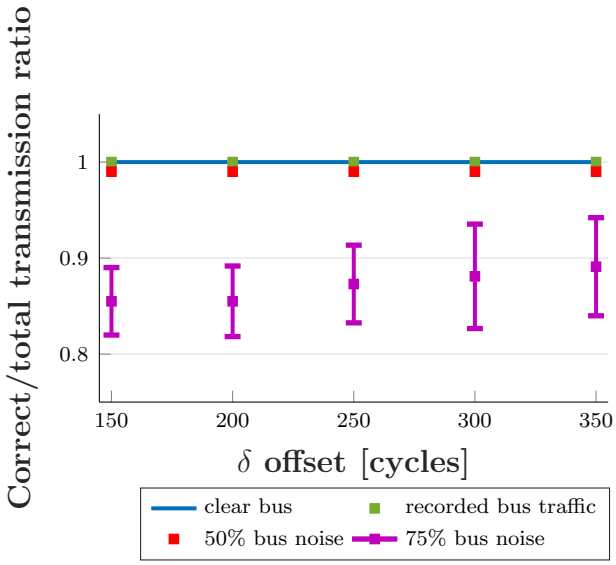


Figure 3: Fraction of covert payloads transmitted correctly for varying values of δ in a low-level IAT channel implementation on MSP430 hardware, in four bus configurations (clear bus, 50% and 75% random bus congestion, pre-recorded background traffic)

Experimental setup. The scenario used to assess our timing channel progresses as follows. Some covert payload is transmitted 100 consecutive times over this channel, in 10,000 repetitions. That sequence is executed using δ offset values varying from 150 to 350 CPU cycles, and in four different CAN bus configurations: on a clear CAN bus, on a CAN bus with 50% randomised congestion, on a bus with pre-recorded background traffic, and on a CAN bus with 75% random bus congestion. Our pre-recorded traffic stems from the 500 kbps bus of a 2005 Volkswagen vehicle driving slowly on a test track. Thus, this part of the evaluation shows how our channel performs when embedded in a realistic vehicular network. The experiments with the final heavily congested bus configuration is meant to showcase the limits of the approach, as our interrupt-driven channel exhibits near-perfect performance in all other bus conditions.

Results. Our experimental results are depicted in Figure 3. In the first three bus configurations on which our timing channel was assessed (clear bus, 50% bus congestion, pre-recorded traffic), it transmits 100% of covert payloads correctly, using δ offsets as low as 150 CPU clock cycles, which corresponds to 7.5 μs at a 20 MHz clocking frequency. Such accuracy was mainly enabled

by our precise, interrupt-driven approach to IAT value registration.

When raising bus congestion to 75%, the amount of correctly transmitted covert payloads drops to an average of 85-90% when varying δ from 150 to 350 clock cycles. Software-level noise sources for our IAT channel have been eliminated as much as deemed possible, so this performance degradation mainly stems from bus occupancy by noise traffic which impedes timely arrival of the non-covert traffic constituting our IAT channel.

5. Timing-Based Nonce Synchronisation

Given the promising results of IAT-exploitation on MSP430-hardware, that mechanism can sensibly be applied to a more specific, practical context. This section motivates and discusses the timing between application- and corresponding authentication-traffic in VulCAN as a concrete source of such covert bandwidth and proposes its use for nonce synchronisation purposes, in an effort to alleviate performance- and security-related issues in the current VulCAN design. This case-study shows that covert transmission, despite its drawbacks, can considerably aid existing security solutions, even when offering only little supplementary bandwidth.

5.1. Overview of the Approach

The following high-level overview of our proposed design for nonce synchronisation in VulCAN is illustrated in Figure 4.

In the original VulCAN design, transmission of a message from a sender (**S**) to a receiver (**R**) over a vatiCAN-authenticated connection progresses as follows. First, **S** transmits the message at hand, or the *application message*. Then, **S** calculates a MAC for the latter, using its local nonce. Finally, **S** embeds that MAC in a second message, referred to as the *authentication message*. Upon receiving an application message, **R** calculates its own MAC using its local nonce, which **R** then compares to the MAC carried by the authentication message it subsequently receives. Only if those two MACs are equal, authentication passes and the application message at hand is accepted by **R**, whereas it is discarded otherwise.

In our proposed covert VulCAN extension, **S** introduces a delay before sending its authentication message, which is an encoding of the N least significant bits of its local nonce value. **R** in turn proceeds as normal, yet monitors the message pair's inter-arrival time. Should authentication fail using its local nonce, **R** does not immediately discard the application message

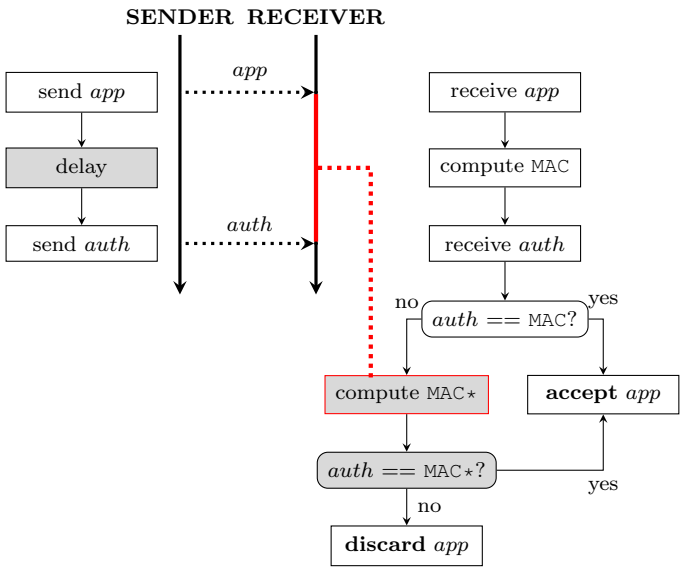


Figure 4: High-level overview of timing-based nonce synchronisation in VulCAN's vatiCAN backend, with additions to the original VulCAN design indicated in grey

at hand. Instead, it replaces the N least significant bits of its local nonce with the bits it decodes from the obtained inter-arrival time. **R** may then retry and possibly correct authentication by calculating a different MAC using that new nonce (if it meets certain criteria).

Our approach hence migrates the timing channel discussed and evaluated in Section 4 to the practical context of vatiCAN nonce synchronisation in VulCAN. Notably, while the original IAT channel carries its covert payload in the timings between application messages, this nonce synchronisation scenario exploits the relative timing difference between an application message and its corresponding authentication frame. As such, authenticated traffic can fully transparently carry covert information without exhibiting application-level periodicity, which was not possible for the timing channel discussed in Section 4.

5.2. Motivation

This section motivates our approach as presented in Section 5.1, more specifically its VulCAN-embedded use of a timing channel on the one hand, and the use of that covert bandwidth in nonce synchronisation for VulCAN's vatiCAN backend on the other hand. That distinction is explicitly made to

emphasise how either could benefit VulCAN independent of the other, when moving beyond the context of our work.

Moreover, we show that our design fulfils all system requirements listed in Section 3.2. Concretely, it inherits **R2** and **R4** from using a timing channel, and enables **R1** and **R3** as elaborated on below.

Timing channel embedding in VulCAN. Our proposed design embeds timing channel logic, *i.e.*, manipulation and decoding of authentication frame timings, in the VulCAN library itself, by extending the original code governing vatiCAN-authenticated communication accordingly.

Transparency. The VulCAN library has full control over both the sending and receiving of application messages, as well as authentication messages. As such, all timing-related operations needed for exploiting the covert bandwidth considered here can be embedded in the VulCAN library itself, without affecting its user interface.

*Backwards compatibility **(R1)**.* As both the current VulCAN library and the vatiCAN message authentication protocol do not rely on any authentication frame timing management in sending and receiving authenticated traffic, the covert transmission described here is a mere extension of its existing functionality, and no compatibility with legacy or "vulcanised" ECUs is broken when it is enabled.

Application independence. Covert data here is transmitted via timings not between application messages, but between each single application message and its corresponding authentication frame. Our covert channel therefore relieves from the communication periodicity prerequisite posed on application traffic by the channel discussed in Section 4.

Real-time effects. The VulCAN library at receiver side propagates application messages to the receiver's higher-level logic only after they have been authenticated. The proposed delay of authentication frames therefore delays application frame processing as well, which may somewhat attenuate previous claims of transparency, backwards compatibility and application independence in real-time scenarios. Indeed, applications that in their functionality rely on CAN packet timings, depending on their sensitivity to deviation in those timings, might break when their vulcanised CAN nodes transition to using this extension.

Application to nonce synchronisation. Our proposed design transmits the N least significant bits of the nonce used in constructing the MAC carried by an authentication frame in the timing of that frame.

*2^N-length message loss burst recovery **(R3)**.* The explicit, although covert, transmission of N least significant local nonce bits allows for a gap between the local nonce values of a sender and a receiver as large as 2^N to be bridged successfully on authentication message arrival. For security purposes, as elaborated on in Section 7, only receiver nonces getting behind on sender nonces can be recovered from, which in vatiCAN corresponds to recovery from message loss.

Indeed, vatiCAN guarantees monotonically increasing local nonce values, that are updated on successful message transmission (arrival) at sender (receiver) side, which means a higher local nonce at a sender implies some transmissions were left without a corresponding arrival. As vatiCAN moreover increments local nonce values by one on each successful transmission/arrival, a nonce gap of 2^N corresponds to as many subsequent messages getting lost. In comparison, current VulCAN design sacrifices message loss recovery completely for replay attack resistance in its vatiCAN backend, as discussed in Section 2.

Benefits over LeiA. LeiA [10] uses extended CAN IDs, *i.e.*, the **ID1** covert channel as discussed in Section 9, for transmitting partial nonce values, which *(1)* breaks applications relying on extended IDs, and *(2)* incurs extra power and time consumption in applications that were not previously using extended CAN frames. Our approach inherits neither of these drawbacks, while partially implementing LeiA's functionality of transmitting nonce bits for synchronisation.

5.3. Sender-Side Design

We summarise sender-side operations for our covert nonce synchronisation protocol in Figure 5.

In original VulCAN design, a sending node first transmits the message at hand, with no modifications, then calculates the MAC value corresponding to the message, and without delay transmits the MAC as authentication payload in a second message.

In difference, in our proposed VulCAN extension, the sender additionally encodes the N least significant bits of the nonce that is used during

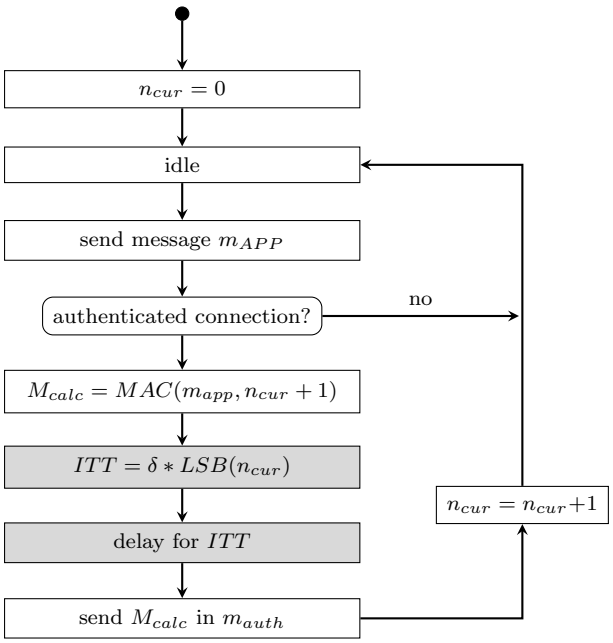


Figure 5: Sender-side message transmission flowchart when leveraging authentication frame timings for nonce synchronisation in VulCAN's vatiCAN backend. Additions to the original VulCAN design are indicated in grey. N denotes the amount of nonce bits transmitted covertly, δ the IAT value granularity used.

MAC computation by carefully delaying transmission of the authentication payload. In our implementation, a cycle-accurate delay can be caused by executing a busy-waiting assembly loop of appropriate length. Covert payload encoding is performed by multiplying the N least-significant nonce bits with a timing interval δ.

5.4. Receiver-Side Design

We summarise receiver-side operations for our covert nonce synchronisation protocol in Figure 6.

Receiving nodes on an authenticated communication channel in the original VulCAN design first receive an application message, then calculate an expected MAC value using their local nonce value, and then receive the corresponding authentication frame. If the latter carries a payload equal to the expected MAC, authentication succeeds and the application message is proceeded to the receiver's higher-level logic. Otherwise, that application message is discarded.

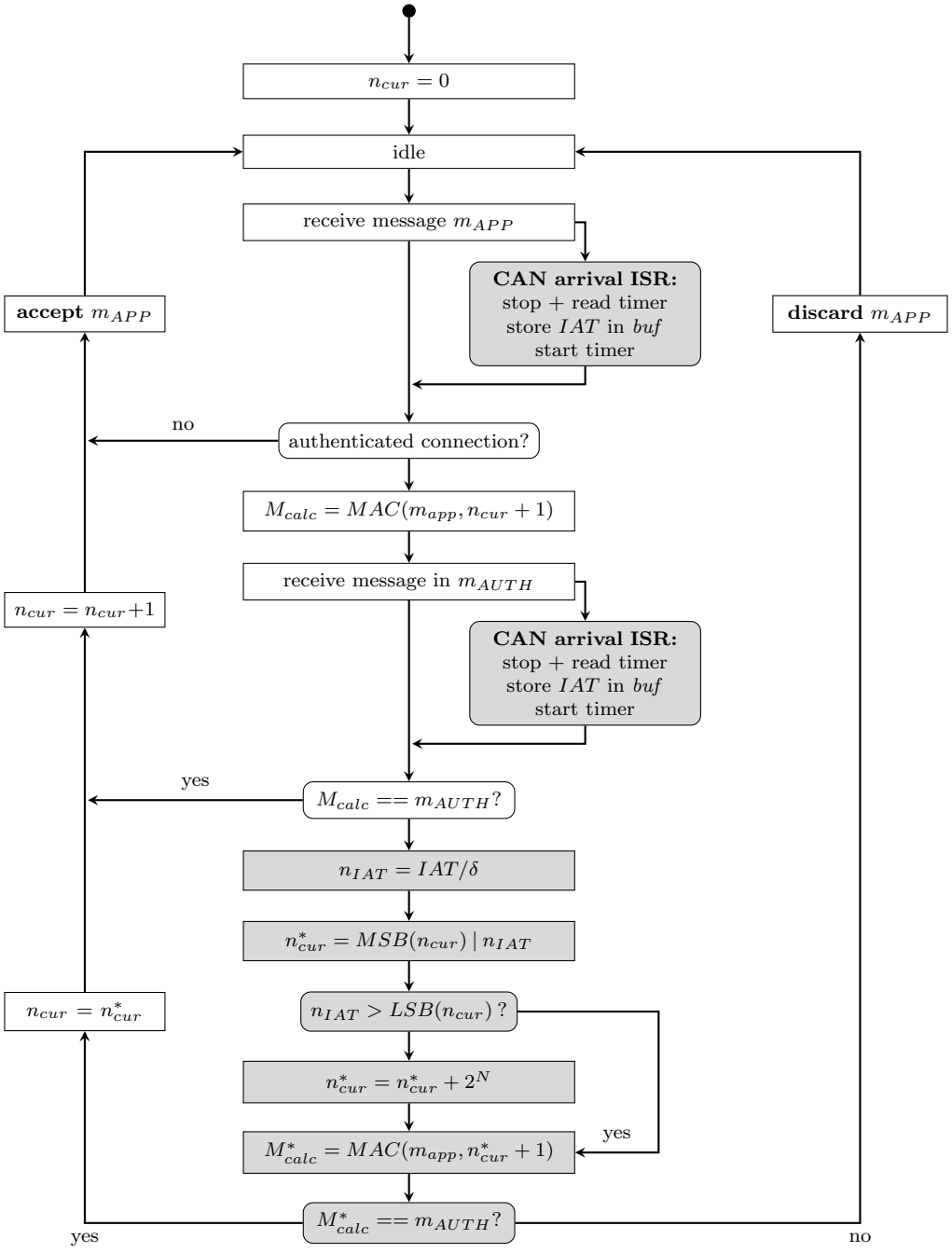


Figure 6: Receiver-side message receiving flowchart when leveraging authentication frame timings for nonce synchronisation in VulCAN's vatiCAN backend. Additions to the original VulCAN design are indicated in grey. N denotes the amount of nonce bits transmitted covertly, δ the IAT value granularity used. *buf* refers to a buffer holding IAT values.

The first extension to the original design accommodates the collection of IAT values. For this, a circular *IAT buffer* is added to the receiver's software, which holds inter-arrival timings of both application messages and authentication messages, and which is accessible to untrusted CAN driver software as well as the trusted in-enclave VulCAN library. Registration of IAT values is embedded in CAN driver software, using an interrupt mechanism similar to the timing channel implementation of Section 4. On each CAN message arrival, an interrupt is fired, whose ISR measures its timing relative to the preceding message arrival and stores that value in the IAT buffer. Furthermore, a global *IAT index* is added to the receiver's software to denote the position of the most recently registered value in the shared IAT buffer. That index, like the IAT buffer, is accessible by both the trusted in-enclave VulCAN library and the untrusted CAN driver.

Second, the original vatiCAN backend of the VulCAN library is extended with additional operations to recover from message authentication failure. In such an event, a new tentative nonce is first constructed based on the inter-arrival time of the most recent authentication frame. Recall that this timing is recorded by the untrusted CAN driver and can be fetched from shared memory by adding the current IAT index to the start address of the circular IAT buffer. In construction of the new tentative nonce, N covert nonce bits N_{dec} are first obtained by dividing the IAT value at hand by the δ parameter used for encoding at the sender side (see Section 5.3). To protect against malicious nonce downgrading attacks, our VulCAN extension subsequently compares N_{dec} to the N least significant bits N_{rec} of the receiver's local nonce. Only when N_{dec} is strictly larger than N_{rec}, the receiver's local nonce least significant bits are replaced with N_{dec}. Otherwise, we assume an overflow of the covert nonce bits and first increment the upper part ($32 - N$ bits) of the receiver's local nonce value before replacing the lower bits with N_{dec}.

Based on its new nonce, a second MAC value can now be calculated at the receiver, which is then compared to the received MAC value. When those are equal, authentication succeeds on this second attempt and the covert nonce is committed at the receiver side. In this case, the receiver has fully transparently recovered from the message loss, and the application message at hand can be securely propagated to the application logic. In the case where authentication fails again, on the other hand, the receiver's nonce remains at its original local value, on which the first authentication failure occurred, and the application message at stake is discarded. Such repeated authentication failure may for instance occur in case of a more

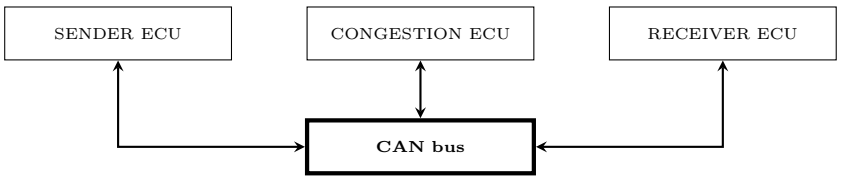


Figure 7: Experimental set-up used for performance evaluation of timing-based nonce synchronisation, as reported on in Table 3.

severe network outage, where more than 2^N subsequent messages have been lost. At this point, the receiver may opt to initiate a more involved and expensive resynchronisation protocol with several messages going back and forth to reset nonces and securely re-establish a shared session key with the sender, as in the LeiA [10] specification.

6. Performance Evaluation

Experimental scenario. To evaluate the reliability our approach, we implemented and assessed it on Sancus-enabled MSP430 hardware [33] equipped with off-the-shelf MCP2515 CAN transceivers [34]. Our microcontroller setup is identical to VulCAN [13] and the MSP430s are clocked at 20 MHz; the CAN is configured to 500 kbps. As illustrated in 7, we configured a vulcanised pair of nodes to use timing-based nonce synchronisation ($N = 2$ nonce bits, $\delta = 1000$ CPU cycles) in a transmission sequence of 10.000 authenticated messages, each fifth of which is explicitly suppressed to simulate incidental message loss, totalling no less than 2000 dropped messages for each experiment. We evaluate practical reliability by measuring the total amount of authenticated messages that were successfully accepted at the receiver side, which reveals the average amount of message losses that were successfully recovered from by our design. This experiment is repeated for the four different CAN bus configurations used for our **T1** timing channel evaluation in Section 4: a clear bus, a bus replaying real-world CAN traffic, a bus with 50% randomised congestion, and a bus with 75% randomised congestion. With the simulated congestion, our setup is technically equivalent to having many communicating ECUs on the same network, as in an actual vehicle. All relevant experimental parameters are listed in 2.

Results. Table 3 summarises experimental results. It shows that our approach offers near-perfect $> 99\%$ reliability in recovering from message loss

Table 2: Simulation parameters used in a performance evaluation of timing-based nonce synchronisation, as reported on in Table 3.

Parameter	**Description**	**Value**
Application dependent		
T	Time interval between packets when unmodified for covert channel	100 ms
Drop rate	Measure for artificial message loss	Every fifth message
Total packet count	Total amount of packets transmitted	10.000
Timing channel dependent		
N	Amount of nonce bits sent through packet timings	2
δ	Packet timing deviation used for nonce bit encoding	1000 CPU cycles
CAN bus dependent		
Congestion	Measure for background traffic on the CAN bus used	0,50,75% / pre-recorded
CAN bus speed	Bitrate on the CAN bus used	500 kbit/s

Table 3: Average fraction of message losses recovered from through timing-based nonce synchronisation

	% message loss recovered from
clear bus	99,68
recorded bus traffic	99,04
50% bus noise	95,35
75% bus noise	89,35

on a bus with real-world background traffic. Even when heavily increasing CAN bus congestion to 75%, the nonce recovery rate only slightly degrades to around 90%. These results are very similar to our **T1** timing channel evaluation in Section 4, as was expected due to the close analogy between **T1** and the VulCAN-specific timing channel that is used in our design. Moreover, these reliability results confirm the objective of our design offering message loss robustness as imposed by **R3**. Figure 8 illustrates the throughput of our channel for $N = 2$ and $N = 3$ nonce bits, and $\delta = 1000$ and $\delta = 2000$ CPU cycles (without bus congestion). Our results show that the preferable channel configuration remains application specific and depends on the acceptable latency. While VulCAN [13] reports round-trip times for authenticated communication of 1.82 ms to 2.18 ms, which is well below many automotive safety deadlines, our solution adds a modest 0.2 ms latency in the worst case. However, this comes with increased robustness against message loss, which

would trigger a more expensive nonce re-synchronisation.

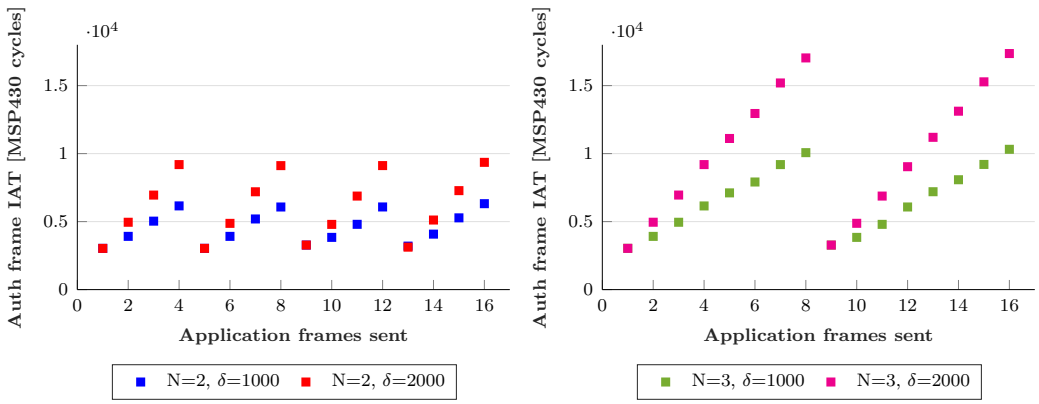


Figure 8: Time series of authentication frame inter-arrival times for timing-based nonce synchronisation in VulCAN's vatiCAN backend. Four times two nonce bits *(left)* or two times three nonce bits *(right)* are transmitted over 16 application frames.

7. Security Analysis

As nonce mechanisms were put in place for security purposes, our nonce synchronisation approach is carefully designed to harm neither vatiCAN's nor VulCAN's security guarantees. We discuss the attacker capabilities that affect our specific design, as well as the security measures put into place to prevent those from being more effective in our proposed extensions, compared to the original VulCAN design.

7.1. Attacker Capabilities

Below, we list the attacker capabilities that pertain to our specific use of a timing channel, as enabled by the Level 1 and Level 2 attacker profiles introduced in Section 3.4.

Attacker level 1: Network attacker. As demonstrated by existing CAN network attacks [7, 8, 18], CAN inherently offers little security. By extension, a timing channel built upon CAN communication, such as the one proposed here, is vulnerable to multiple security threats. A Level 1 attacker can successfully affect the timing channel used in our approach as listed below. Note that these actions pertain to timing-based transmission itself, rather than its

underlying CAN traffic. Nevertheless, both are evidently interdependent, which leads to most IAT channel attacker capabilities following from CAN network-level attacker capabilities as listed in Section 2.1. We consider side-channel attacks or probing [5] against our covert channel as out of scope since we use the channel to communicate non-confidential information only. These analyses could, however, be applied against the underlying key management and authentication protocols.

A1: Eavesdropping. As CAN employs broadcast communication without enforcing bus access control, this attacker can eavesdrop on, as well as record, packet inter-arrival times and thus all IAT channel payloads. As nonce values however can be disclosed without affecting security guarantees, this attacker capability does not cause any harm and thus will not be focused on hereafter. This disclosing of all CAN traffic does however provide the attacker with the scheduling and/or CAN network architecture knowledge that could be a prerequisite for other attacker capabilities [7].

A2: Message manipulation. In multiple ways, an attacker is capable of delaying the CAN traffic constituting IAT values. For example, injection of high-priority CAN frames, or well-timed 0-bit transmissions, can cause legitimate CAN transmission to be delayed due to the nature of its arbitration mechanism, or targeted nodes can temporarily be forced into bus-off mode [7, 8]. Consequently, all IAT values are vulnerable to modification. Note how attackers can both increase and lower IAT values, through respectively delaying authentication and application frames, which means arbitrary IAT values can be constituted on an attacker-controlled CAN bus.

A3: Message deletion. In order to delete IAT payloads from a timing channel, the corresponding CAN packets are to be deleted, through for example CAN bus flooding or a selective DoS attack [7, 18]. There is no way for an attacker to delete IAT payloads from a timing channel without suppressing the corresponding CAN frames, as their presence inevitably can be detected, and their timing thus registered.

A4: Message insertion. Analogous to **A3**, an IAT value can be inserted into this IAT channel through inserting two corresponding CAN frames on the targeted CAN bus. As a Level 1 attacker has free access to that bus, they can place any two messages with any inter-arrival time on it.

Note that such inserted messages need to have the proper IDs in order for their inter-arrival time to be registered by a vulcanised receiving node, as VulCAN incorporates a low-level mechanism for message ID masking and filtering. More specifically, VulCAN dictates the use of a message ID equal to

an application message's incremented by one, for its corresponding authentication frame. For IAT value insertion to work, the second inserted message's ID thus has to be equal to the first message's ID incremented by one, and the latter moreover has to be equal to the ID used by the targeted VulCAN communication channel.

Attacker level 2: Software attacker. Our Level 2 attacker model gives rise to supplementary attacker capabilities affecting our design, due to its unrestricted access to the untrusted non-enclave application logic on the participating ECUs handling both regular and covert traffic. Note that a Level 2 attacker also has **A1**-**A4** abilities, as it is strictly more powerful than a Level 1 attacker (see Section 3.4).

A5: Stealthy message manipulation. As opposed to the message manipulation capability **A2** discussed for the Level 1 profile, a Level 2 attacker is capable of altering IAT values much more directly, without even having to interfere with the CAN bus. With arbitrary unprotected code execution at the receiver side, Level 2 attackers can take over either CAN driver software, IAT buffer and index contents, or even the timing peripheral measuring IAT values. They can thus trivially cause arbitrary, attacker-chosen IAT values to be registered and used as alleged authentication frame timings in the trusted in-enclave VulCAN logic.

A6: IAT buffer location manipulation. CAN driver software is deemed untrusted in our system model and requires write access to the receiver-side IAT buffer. Therefore, that buffer is placed in unprotected memory, which is subsequently dereferenced as an untrusted pointer by the trusted, in-enclave VulCAN library. However, such pointer passing in the shared address space is known to come with subtle security implications [36]. Particularly, in a confused-deputy attack scenario [37], Level 2 adversaries can maliciously craft untrusted pointers such that the shared IAT buffer unexpectedly falls *inside* the VulCAN enclave. As Sancus grants enclaves with access to both private and public memory locations [15, 38], execution will continue in such a case and the secret enclave data referenced by the poisoned buffer pointer will be wrongly interpreted as alleged IAT values. Any attacker-observable properties of VulCAN's execution, e.g., the passing or failing of authentication, can subsequently be used as a side channel to leak enclave secrets [36].

A7: IAT index manipulation. An index into the receiver's IAT buffer, that denotes the position of the most recently collected IAT value, is updated

by CAN driver software in our proposed design. Therefore, that index value and its location in unprotected memory is under Level 2 attacker control. Hence, attackers can attempt to make trusted in-enclave VulCAN code dereference arbitrary addresses, possibly lying in the protected memory range, by supplying a malicious, out-of-bounds index [36]. Moreover, similar to the **A6** capability, this attacker could position the IAT index at a protected location in memory, and deduce private information from the trusted VulCAN library dereferencing that location.

7.2. Security Measures

Our proposed nonce synchronisation approach was designed not to harm VulCAN security properties, *i.e.*, not to enlarge its vulnerability in the face of both Level 1 and Level 2 attackers. The security measures listed below were designed for that purpose, and are therefore executed in the trusted Sancus enclaves hosting sender- and receiver-side VulCAN logic.

M1: IAT encoding. The encoding and decoding between nonce bits and authentication frame delays in the proposed scheme is a multiplication with, respectively division by, a parameter δ. When faced with a weaker attacker profile, this design aspect could have been designed to provide additional security guarantees, but since attackers can eavesdrop on, and arbitrarily manipulate, all IAT values (see Section 7.1), it is justified to be as algorithmically simple as possible.

M2: Monotonically increasing nonces. The proposed design entails that whenever authentication fails using a receiver's local nonce value, a second try is done using a different nonce, which is constructed based on the corresponding authentication frame timing (see Section 5.4). That nonce construction is explicitly designed to only yield nonces higher than the original local nonce value. Combined with vatiCAN's guarantee of monotonically increasing local nonces, this measure guarantees that whenever a receiver changes its local nonce value after a successful authentication attempt, that nonce is guaranteed to become strictly higher.

M3: Delayed nonce commit. Should an authentication retry using a timing-based nonce fail, the local nonce of the receiver involved is reset to its original value, which was used for its first authentication attempt. Consequently, local nonce value incrementation only occurs when the corresponding authentication frame is calculated using either the receiver's original, or IAT-constructed nonce value.

The combination of **M2** and **M3** requires an attacker targeting this nonce mechanism to possess a valid pair of application and authentication messages, that is moreover calculated using a nonce strictly higher than the locally stored value of a receiver in the targeted communication channel. Without such a pair, an attacker's capabilities for modification, deletion and insertion in the proposed timing channel are insufficient to launch a nonce-based attack against this VulCAN extension.

A valid message pair is available only when the attacker involved is either capable of constructing its own valid authentication frame for an application message, or can replay a recorded pair. On vulcanised nodes, MAC calculation is done in a protected Sancus enclave, using a cryptographic key to which an attacker (of either Level 1 or Level 2) has no access. This renders computation of forged authentication frames computationally infeasible. The second approach of replaying recorded traffic is prohibited by the proper use of monotonically increasing nonces for all authenticated traffic in the original VulCAN design [13]. Likewise, measures **M2** and **M3** above ensure that nonce monotonicity is strictly maintained at all times in our covert nonce synchronisation extension, such that any attack directed at our nonce synchronisation approach would coincide with an opportunity for launching a replay attack against original VulCAN communication.

Consequently, **M1**, **M2** and **M3** together offer resilience of our nonce synchronisation scheme to a Level 1 attacker. Indeed, none of its capabilities render it capable of generating effective authentication frames. A Level 2 attacker does not have that ability either, but its control over the IAT buffer and IAT buffer index read by trusted vatiCAN code leads to the additional need for the following security measures.

M4: IAT buffer validation. Attacker capability **A6** describes how the considered system model (see Section 3.3) leads to trusted code dereferencing Level 2 attacker-controlled memory locations, which is a known attack vector for leaking enclave memory [36]. Validation of that buffer is therefore done at VulCAN initialisation. More specifically, it is checked to completely lie outside enclave memory.

M5: IAT buffer index validation. Attacker capability **A7** poses the same threat of trusted code dereferencing untrusted memory locations as described in **M4**, which is mitigated by validating the IAT index to have a value between 0 and the length of the IAT buffer at the time it is dereferenced by trusted VulCAN code. Moreover, that index itself is validated to lie outside of protected memory.

In conclusion, the security measures **M1**-**M5** do not prevent a malicious party from executing its capabilities **A1**-**A7**, but mitigate harmful repercussions. E.g., a Level 2 attacker in **A7** is still capable of changing the receiver-side IAT buffer index to an arbitrary value, yet they cannot leverage that ability to leak trusted memory due to **M5**. Similarly, IAT values can still be manipulated due to both **A2** and **A5**, but an effective replay attack is disabled by the combination of measures **M2** and **M3**. A complete overview of the security provided by our approach in comparison with related work is provided in Table 1 in Section 3.

8. Limitations and Future Work

Guaranteed availability. Our explicit security objective, in line with previous CAN authentication solutions [13, 11, 10], is to ensure that only genuine messages are ever processed at the receiver side. This does not entail, however, that all such messages will also arrive at the receiver side. That is, in the presence of strong Level 1 or Level 2 attackers who can arbitrarily manipulate traffic on the CAN bus, or even hijack untrusted system software on the participating ECUs, guaranteed availability falls explicitly out of the scope of our approach. Guaranteeing network bandwidth and availability is an orthogonal question and would likely require, amongst other, replacing CAN altogether with a non-broadcast medium that can guarantee quality-of-service at the network level, as well as making extensive changes in the TEE design on the participating ECUs [39].

Our solution, therefore, is more pragmatic in that we showcase the use of limited covert bandwidth to improve the robustness of an existing authentication scheme (vatiCAN) for an existing industry-standard communication medium (CAN). We have shown that our solution can improve quality-of-service under benign circumstances, while at the same time not creating insecurities when under attack.

Message suppression. The very motivation for our approach enables malicious suppression of authenticated application traffic, which was not possible in VulCAN's original vatiCAN [11] backend. Concretely, when our nonce synchronisation strategy is enabled, an attacker is capable of selectively suppressing as much as 2^N subsequent application messages at a time, while other traffic is processed normally at receiver side. Depending on the application scenario, such suppression could harm vehicle safety, e.g., when a

message warning for malfunctioning components is suppressed and the receiver continues regular execution instead of initiating a reactive operation. In contrast, the use of VulCAN's current vatiCAN backend implies discarding all CAN traffic subsequent to such message suppression.

As our security objective states, we aim to provide the same security guarantees as LeiA [10]. Notably, this message suppression scenario succeeds in LeiA as well and hence does not harm the compliance of our design to its security objective. However, to accommodate application-level measures that handle (malicious) message loss, future work could extend the VulCAN API to include status information about message loss recovery through timing-based nonce synchronisation.

Limitation to CAN. We identified covert channels in the specific context of CAN, whereas other interesting opportunities could also exist for alternative network protocols. One such protocol is CAN+ [40], which is a physical extension to CAN based on exploiting the higher clock frequency of most embedded devices relative to their CAN bus' speed, for extra CAN transmission during their over-clocked cycles. This context could allow for more intricate covert channels, for example based on the timing properties of transmission within such over-clocked time intervals. That approach could allow for a timing channel similar to the one presented in Section 4, without the prerequisite of periodicity in underlying traffic, as timing information of this specific kind can be contained within the transmission of each individual message.

Limitation to VulCAN. This work discussed a practical application of covert transmission in VulCAN's approach to message authentication [13]. However, the covert channels discussed in Section 9 were formulated for CAN communication in general, which means they could benefit application domains beyond VulCAN as well. They moreover are not restricted to be leveraged for message authentication, or other security mechanisms, as they offer supplementary bandwidth usable for any purpose. As such, they could even be implemented as an extension to CAN communication itself when embedded in CAN driver software, instead of incorporated in existing applications and designs. They could therefore enable a backwards compatible, transparent mitigation for bandwidth scarcity in CAN applications.

Limitation to vatiCAN. Our design was formulated specifically for VulCAN's vatiCAN backend [11], yet could benefit other message authentication schemes similarly. The challenges faced in nonce synchronisation apply to any such

scheme, and thus could be relieved by use of (timing-based) covert channels. For instance, future work could explore the use of our timing channel for synchronising implicit nonce parts in VulCAN's LeiA backend [10].

Scalability in Realistic Scenarios. In [13], Van Bulck et al. extensively discuss scalability issues of authenticated communication in automotive CAN networks that are compatible with legacy infrastructure that does not support the authentication protocol. Generally, adding AUTOSAR-compliant authentication to the CAN communication of two ECUs will at least double the bandwidth requirement and latency of a message exchange. That is because, instead of one message, two messages are to be transferred and the authenticity of the payload is only established after both messages are received and processed. Cryptographic operations add to this increased latency and also increase the energy consumption and general costs of the system. To which extent message loss and nonce synchronisation increase the general overhead depends a lot on the actual operating conditions such as bus saturation and (electrical) noise levels, and to the best of our knowledge, there is no accessible database with this kind of data for modern cars or other large-scale control systems. We therefore leave a comprehensive evaluation of the scalability of our approach under real vehicular conditions for future work.

9. An Overview of Covert Channels in CAN

This section overviews covert, and covert-like, bandwidth opportunities in CAN. Most of them are storage channels, and do not satisfy our covertness definition as formulated in Section 2.3, due to (partial) overwriting of CAN frame fields. We do however include such channels for completeness, as they comply to the covertness definition used in a subset of existing research on covert channels [27, 29, 28]. Channels that satisfy our covertness definition are italicised in this overview, which is moreover structured following the CAN frame format as presented in Figure 1, listing channels leveraging respectively the ID field (**ID**), the data field (**D**), the CRC field (**E**) and frame timings (**T**). For each channel, we indicate the system requirements (see Section 3.2) it fulfils when incorporated in our design.

We refer to Table 4 at the end of this section for an overview matrix summarising the properties of the channels discussed below.

9.1. Storage Channels

ID1: Dedicating ID bits (R2). Some applications rely on fewer distinct IDs than representable with 11 or 29 bits. Those can dedicate bits in the (extended) ID field of their frames to covert data. LeiA [10] uses this channel for nonce synchronisation by embedding nonce bits in extended IDs.

ID2: Manipulating arbitration collision frequency. Existing CAN attacks [7, 8] describe how CAN's arbitration scheme enables selective DoS attacks. Repurposing that technique to a covert channel, nodes can cause selective arbitration collisions to transmit covert data to their targeted node. The collision frequency then can encode some covert payload. Government of arbitration collisions is not generally possible from CAN driver software, restricting this channel's applicability for our work to discussion purposes.

D1: Dedicating LSBs (R2, R4). As described by TACAN [16], some applications are resilient to lowered accuracy in transmitted values. Some amount of LSBs can then carry covert data, as long as no more bits than discarded by the underlying application are repurposed. As only manipulation of bits is done by this channel, its use incurs little computing overhead and no real-time effects.

D2: Manipulating packet size (R2). Another approach to leveraging the CAN frame data field uses frame sizes as a covert payload carrier, by varying data field lengths, and data length values accordingly. The applications this channel is built on must tolerate the associated variations in accuracy. Also, as this channel affects the amount of bits sent over its bus, its influence on real-time compliance has to be considered.

D3: Data field padding (R2). Should an application use a predictable (in extremis, fixed) data field length in its communication, the data field of its packets can be padded with covert data. Naturally, this channel only yields non-zero bandwidth when that predictable data field length is lower than 8 bytes. Moreover, it increases the amount of traffic generated, which endangers real-time compliance.

E1: Dedicating the CRC field. As CAN provides with automatic retransmission of frames with an invalid CRC field, that field can carry covert data on initial transmission, which a receiver can monitor before it is deemed erroneous and retried. The CRC field however is not software-controllable, rendering this channel not of interest for our work.

E2: Pass/fail of error detection check. In a less invasive manner than proposed by **E1**, it is possible to embed covert information in the passing or

failing of a packet's CRC check. More specifically, a sender can purposely manipulate its CRC field for it to yield an error when checked at receiver side. A receiver monitoring this channel can then decode traces of its CRC check outcomes, without needing the corresponding CRC fields themselves, to the intended hidden payload.

9.2. Timing Channels

T1: Manipulating packet inter-arrival times (R2, R4). Even when not in control of multiple message IDs, data can be encoded in packet timings, more specifically in their inter-arrival times. If an application makes use of periodic communication of period T, deviations from T can encode some covert payload, as proposed by TACAN [16]. Making a receiving party monitor packet inter-arrival times then enables it to decode those to the corresponding covert data. This approach obviously endangers real-time deadlines, and furthermore is only useful in applications using periodic communication.

T2: Packet reordering (R2, R4). Not only the timing of a packet, but also its ordering relative to other packets can be an encoding of covert data. More specifically, when an application is in control of packet transmissions with different ID fields, it can adjust the order of its packets to a sequence from which covert information can be decoded. That practice obviously impacts real-time behaviour and thus cannot be considered generally applicable, even less because of its prerequisite of the underlying application controlling multiple packet IDs.

T3: Combining T1 and T2 (R2, R4). As the ordering of payloads is a property orthogonal to the timings between them, **T1** and **T2** can be combined into one hybrid timing channel. More specifically, information can be encoded at sender side in the ordering of the messages on the one hand, and in their inter-transmission times on the other hand. A receiver can monitor both from software to then decode them to the corresponding covert data. Note that this approach also combines the underlying application prerequisites of both channels, which are governance of multiple message IDs and periodic communication in this case.

9.3. Hybrid Channels

As already illustrated by the **T3** channel, several of the channels described in this section can be combined into one hybrid channel. Two conditions should be fulfilled when committing to such a combination:

Orthogonality. The channels combined must be pairwise orthogonal, *i.e.*, the use of one cannot interfere with the mechanisms enabling any of the others. To illustrate, the use of N ID bits (**ID1**) can seamlessly be combined with the use of M LSBs (**D1**). In contrast, overwriting those M least significant bits serves no purpose if some bits in the data field are dropped for covert communication via packet size (**D2**).

Prerequisite compatibility. The applications for which a hybrid channel is suitable, must be in the common subset of those appropriate for its constituting channels. These application prerequisites thus should not contradict each other.

9.4. Overview of (Covert) Channels in CAN

Table 4 summarises the properties of our selected channels in a matrix structure. Its rows correspond to the channels discussed above, and its columns evaluate several channel characteristics deemed useful in comparing different forms of (covert) communication: *(1) Channel ID*: The identifier used for a channel as introduced in Section 9; *(2) Stack level*: The level from which a channel can be controlled (hardware/software); *(3) Real-time compliance*: Whether or not enabling a channel affects real-time properties of underlying transmission; *(4) Application dependence*: Dependence on the nature of the application a channel is built on for it to expose non-zero hidden/covert bandwidth; *(5) Bandwidth parameters*: Channel parameters constituting a channel's bandwidth formula; *(6) Bandwidth formula*: Expresses the maximum bandwidth of a channel in bit/s as a function of its already listed bandwidth parameters; *(7) Covertness*: Denotes whether a channel satisfies the covertness definition of Section 2.3.

10. Conclusion

In-vehicle networks like CAN nowadays are faced with powerful remote attackers. Lightweight cryptographic protocols for message authentication have already been proposed to mitigate such threats, yet bring new challenges in terms of deployability in a safety-critical context. Our work explored the benign use of *covert channels* to complement and improve existing restrained authentication solutions. To that end, we contributed a comprehensive overview of covert bandwidth opportunities in CAN. Motivated by those results, we proposed a design that incorporates a timing channel for

covert nonce synchronisation in an existing message authentication scheme. Our approach showcases the use of limited covert bandwidth to improve quality-of-service in benign circumstances, while not jeopardising the strong security guarantees provided by the underlying authentication solution when under attack. As such, we improved quality-of-service by increasing robustness against message loss, while incurring little extra bandwidth, latency and energy usage, which traditional message authentication strategies do not allow. Moreover, we showed that our design does not harm the underlying authentication scheme's security guarantees.

In a wider perspective, our nonce synchronisation case-study results encourage the benign, defensive use of covert channels in attacker-sensitive, resource-limited scenarios beyond automotive applications or CAN.

Acknowledgments. This research is partially funded by the Research Fund KU Leuven, and by the Flemish Research Programme Cybersecurity. We thank Fritz Alder (imec-DistriNet, KU Leuven) for his constructive comments on an earlier version of this paper.

Table 4: Matrix of the CAN channels described in Section 9, with: **ID1**: Dedicating ID bits - **ID2**: Manipulating arbitration collision frequency - **D1**: Dedicating least significant bits - **D2**: Manipulating packet size - **D3**: Data field padding - **E1**: Dedicating the CRC field - **E2**: Pass/fail of error detection check - **T1**: Manipulating packet inter-arrival times - **T2**: Packet reordering - **T3**: Combination of **T1** and **T2**

Channel ID	**Stack level**	**Real-time compliance**	**Application dependence**	**Bandwidth parameters**	**Bandwidth formula** (m = *host channel message rate*)	**Covertness**
A. Storage channels						
ID1	SW	Yes	Yes	hidden bits inserted N	Nm	No
ID2	HW	No	No	collisions allowed/message c	$\lfloor \log_2(c) \rfloor m$	No
D1	SW	Yes	Yes	hidden bits inserted N	Nm	No
D2	SW	No	Yes	max. # bits inserted N_i, max. # bits dropped N_d	$\lfloor \log_2(N_i + N_d) \rfloor m$	No
D3	SW	No	Yes	amount of bits padded N	Nm	No
E1	HW	No	No		$15m$	No
E2	HW	No	No		m	No
B. Timing channels						
T1	SW	No	Yes	application message period T, SW clock granularity g, CAN bus clock frequency f	$\lfloor \log_2(a_{eff}) \rfloor m$ with $a_{eff} = \lfloor 2T/LCM(g, 1/f) \rfloor$	Yes
T2	SW	No	Yes	amount of ID's controlled i	$\lfloor \log_2(i) \rfloor m$	Yes
T3	SW	No	Yes	cf. **T1** and **T2**	$(\lfloor \log_2(i) \rfloor + \lfloor \log_2(a_{eff}) \rfloor)m$ with $a_{eff} = \lfloor 2T/LCM(g, 1/f) \rfloor$	Yes

References

[1] P. C. Kocher, Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems, in: Annual International Cryptology Conference, Springer, 1996, pp. 104–113.

[2] D. Brumley, D. Boneh, Remote timing attacks are practical, Computer Networks 48 (5) (2005) 701–716.

[3] P. Kleberger, T. Olovsson, E. Jonsson, Security aspects of the in-vehicle network in the connected car, in: 2011 IEEE Intelligent Vehicles Symposium (IV), IEEE, 2011, pp. 528–533.

[4] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, T. Kohno, et al., Comprehensive experimental analyses of automotive attack surfaces, in: Proceedings of the 20th USENIX Conference on Security, Vol. 4, USENIX Association, 2011, pp. 447–462.

[5] S. Jain, Q. Wang, M. T. Arafin, J. Guajardo, Probing attacks on physical layer key agreement for automotive controller area networks, in: 2018 Asian Hardware Oriented Security and Trust Symposium (AsianHOST), IEEE, 2018, pp. 7–12.

[6] L. Wouters, J. Van den Herrewegen, F. D Garcia, D. Oswald, B. Gierlichs, B. Preneel, Dismantling dst80-based immobiliser systems, IACR Transactions on Cryptographic Hardware and Embedded Systems 2020 (2) (2020) 99–127.

[7] S. Fröschle, A. Stühring, Analyzing the capabilities of the CAN attacker, in: Computer Security – ESORICS 2017, Springer, 2017, pp. 464–482.

[8] A. Palanca, E. Evenchick, F. Maggi, S. Zanero, A stealth, selective, link-layer denial-of-service attack against automotive networks, in: International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Springer, 2017, pp. 185–206.

[9] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, et al., Experimental security analysis of a modern automobile, in: 2010 IEEE Symposium on Security and Privacy (SP), IEEE, 2010, pp. 447–462.

[10] A. I. Radu, F. D. Garcia, LeiA: A Lightweight Authentication Protocol for CAN, in: Computer Security – ESORICS 2016, Springer, 2016, pp. 283–300.

[11] S. Nürnberger, C. Rossow, – vatiCAN – vetted, authenticated CAN bus, in: Cryptographic Hardware and Embedded Systems – CHES 2016, Springer, 2016.

[12] A. Van Herrewege, D. Singelee, I. Verbauwhede, CANAuth - a simple, backward compatible broadcast authentication protocol for CAN bus, in: ECRYPT Workshop on Lightweight Cryptography, 2011.

[13] J. Van Bulck, J. T. Mühlberg, F. Piessens, VulCAN: Efficient component authentication and software isolation for automotive control networks, in: Proceedings of the 33rd Annual Computer Security Applications Conference, ACM, 2017, pp. 225—-237.

[14] T. Mishra, T. Chantem, R. Gerdes, Teecheck: Securing intra-vehicular communication using trusted execution, in: Proceedings of the 28th International Conference on Real-Time Networks and Systems, 2020, pp. 128–138.

[15] J. Noorman, J. V. Bulck, J. T. Mühlberg, F. Piessens, P. Maene, B. Preneel, I. Verbauwhede, J. Götzfried, T. Müller, F. Freiling, Sancus 2.0: A low-cost security architecture for IoT devices, ACM Transactions on Privacy and Security (TOPS) 20 (3) (2017) 1–33.

[16] X. Ying, G. Bernieri, M. Conti, R. Poovendran, TACAN: Transmitter authentication through covert channels in controller area networks, in: Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems, ACM, 2019, pp. 23–34.

[17] Texas Instruments, Introduction to the Controller Area Network (CAN), Application Report SLOA101 (2002) 1–17.

[18] T. Hoppe, S. Kiltz, J. Dittmann, Security threats to automotive CAN networks – practical examples and selected short-term countermeasures, in: International Conference on Computer Safety, Reliability, and Security, Springer, 2008, pp. 235–248.

[19] M. Wolf, A. Weimerskirch, C. Paar, Security in automotive bus systems, in: Workshop on Embedded Security in Cars, Bochum, 2004.

[20] C. Miller, C. Valasek, Remote exploitation of an unaltered passenger vehicle, Black Hat USA (2015).

[21] C. Miller, C. Valasek, A survey of remote automotive attack surfaces, Black Hat USA (2014).

[22] B. Groza, S. Murvay, A. van Herrewege, I. Verbauwhede, LiBrA-CAN: A Lightweight Broadcast Authentication Protocol for Controller Area Networks, in: International Conference on Cryptology and Network Security, Springer, 2012, pp. 185–200.

[23] AUTOSAR Classic Platform Specification 4.3, Specification of secure onboard communication, `https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_SWS_SecureOnboardCommunication.pdf` (2016).

[24] S. Jain, J. Guajardo, Physical layer group key agreement for automotive controller area networks, in: International Conference on Cryptographic Hardware and Embedded Systems (CHES), Springer, 2016, pp. 85–105.

[25] Y. Xiao, S. Shi, N. Zhang, W. Lou, Y. T. Hou, Session key distribution made practical for can and can-fd message authentication, in: Annual Computer Security Applications Conference (ACSAC), ACM, 2020, pp. 681–693.

[26] J. Noorman, P. Agten, W. Daniels, R. Strackx, A. Van Herrewege, C. Huygens, B. Preneel, I. Verbauwhede, F. Piessens, Sancus: Low-cost trustworthy extensible networked devices with a zero-software trusted computing base, in: 22nd USENIX Security Symposium (USENIX Security '13), USENIX Association, 2013, pp. 479–498.

[27] V. Berk, A. Giani, G. Cybenko, N. Hanover, Detection of covert channel encoding in network packet delays, Dartmouth College Technical Report TR2005-536 (2005).

[28] S. Zander, G. Armitage, P. Branch, A survey of covert channels and countermeasures in computer network protocols, IEEE Communications Surveys and Tutorials 9 (3) (2007) 44–57.

[29] US Department of Defense, Trusted computer system evaluation criteria, DoD 5200.28-STD (1986).

[30] R. A. Kemmerer, A practical approach to identifying storage and timing channels: Twenty years later, in: Proceedings of 18th Annual Computer Security Applications Conference, IEEE, 2002, pp. 109–118.

[31] B. W. Lampson, A note on the confinement problem, Communications of the ACM 16 (10) (1973) 613–615.

[32] S. Al-Eidi, O. Darwish, Y. Chen, Covert timing channel analysis either as cyber attacks or confidential applications, Sensors 20 (8) (2020) 2417.

[33] T. Instruments, MSP430x1xx family: User's guide, http://www.ti.com/lit/ug/slau049f/slau049f.pdf (2006).

[34] Microchip Technology Inc., MCP2515: Stand-alone CAN controller with SPI interface, http://ww1.microchip.com/downloads/en/DeviceDoc/MCP2515-Stand-Alone-CAN-Controller-with-SPI-20001801J.pdf (2019).

[35] T. Fischl, USBtin - USB to CAN interface, https://www.fischl.de/usbtin/ (2016).

[36] J. Van Bulck, D. Oswald, E. Marin, A. Aldoseri, F. D. Garcia, F. Piessens, A tale of two worlds: Assessing the vulnerability of enclave shielding runtimes, in: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, ACM, 2019, pp. 1741–1758.

[37] N. Hardy, The confused deputy (or why capabilities might have been invented), ACM SIGOPS Operating Systems Review 22 (4) (1988) 36–38.

[38] R. Strackx, F. Piessens, B. Preneel, Efficient isolation of trusted subsystems in embedded systems, in: International Conference on Security and Privacy in Communication Systems, Springer, 2010, pp. 344–361.

[39] J. Van Bulck, J. Noorman, J. T. Mühlberg, F. Piessens, Towards availability and real-time guarantees for protected module architectures, in:

Companion Proceedings of the 15th International Conference on Modularity (MASS'16), ACM, 2016, pp. 146–151.

[40] T. Ziermann, S. Wildermann, J. Teich, CAN+: A new backward-compatible Controller Area Network (CAN) protocol with up to 16x higher data rates, in: Proceedings of the Conference on Design, Automation and Test in Europe, European Design and Automation Association, 2009, pp. 1088—-1093.