Security concepts with CAN XL

Peter Decker (Vector Informatik)

Given the rise of automotive Ethernet and in view of the growing number of communication systems, a consolidation appears reasonable to limit complexity and costs. With Ethernet now covering infotainment, ADAS, telematics and connectivity at 100...1000 Mbit/s, CAN and CAN FD operate in the range of 0.5...5 Mbit/s and are responsible for engine management, and body control. CAN XL or 10BASE-T1S can potentially be used in the future for control systems.
Considering that about 90 % of all network nodes communicate at speeds below or up to 10 Mbit/s, the 10 Mbit/s domain covers a wide field of application.

Defense in Depth

A trending concept in the IT industry in the last decades was the application of multiple independent layers of security controls in an information technology system with the intention to provide redundancy in case one of these layers fails. This principal is called Defense in Depth.
Therefore, it is not surprising that the industry demands a security layer for both competitors in the 10Mbit/s domain that works in an automotive environment to satisfy this trend.
This paper deals with two problems you encounter when you want to establish a security layer on a multi-drop Data Layer. Namely developing a fast Key Agreement Protocol and enabling End-to-End Encryption when connecting two networks with a Bridge Device.
Following the general idea of Defense in Depth the security controls mentioned in this article are by no means intended to replace any of the already existing security controls within the OSI layers (e.g., SecOC, IPSec or TLS) but to supplement them.

MACsec and CANsec

While Ethernet has MACsec and its accompanying Key Agreement protocol MKA already defined for a decade, CiA is in progress of specifying an CANsec for CAN XL with CiA 613-2. Both MACsec's and CANsec's goals are to provide Confidentiality, Integrity, and Authenticity[1] with line-speed performance on top of their respective Data Layer.

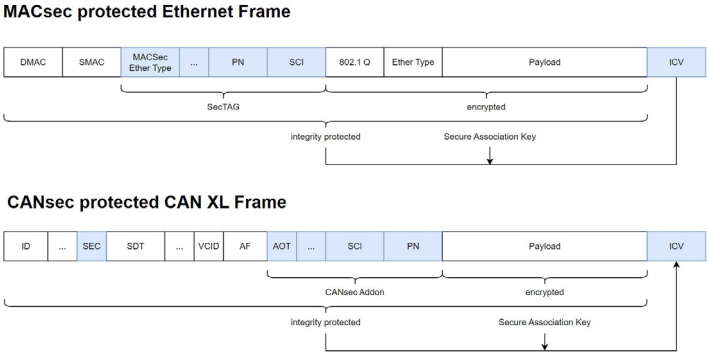

Figure 1: MACsec / CANsec[2]

To meet these requirements both technologies use block ciphers with a mode of operation that provides Authenticated Encryption and Associated Data (AEAD). While MACsec only supports AES in various flavors, CANsec plans to support lightweight cryptography in form of ASCON [3].

The protected frames carry a monotonically increasing Packet Number (PN) to protect against Replay Attacks. This Packet Number is used to derive a Nonce that is required as input of the respective AEAD algorithm.

[1] Authenticity in a strict sense is not achievable between multiple participants and just one key. Authenticity means that you can prove that a message was sent by someone in possession of the key.
[2] Details omitted.

With these cornerstones set it is not possible to use a permanently valid encryption key for entire lifecycle of the network because the reuse of a nonce with the same key effectively makes the encryption ineffective.

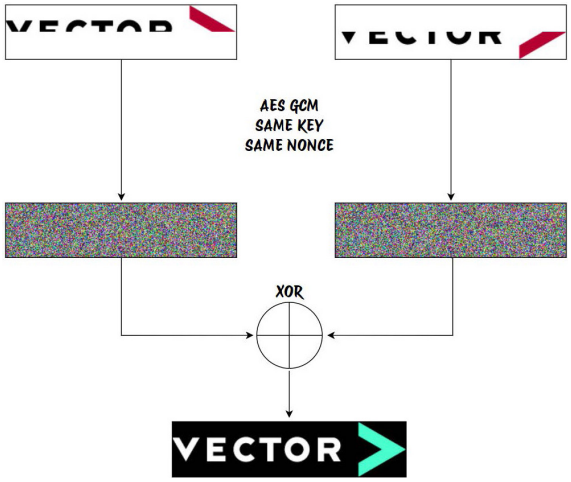

Figure 2: GCM Nonce Reuse

Therefore, a mechanism is required to agree on a temporarily valid session key - or Secure Association Key (SAK) according to the MKA specification - and replace this key regularly.

MKA in the 10Mbit/s domain

Ethernet has the well-proven, feature rich, but also quite complex MACsec Key Agreement specified to solve this challenge, so it feels quite natural to choose MKA also as Key Agreement solution for both CANsec (CAN XL) and MACsec (10BASE-T1S).

MKA is designed around a zero knowledge and zero assumptions approach, and each participant only relies on the quality of its own implementation. It does not need to know how many participants are, will be or were online and the participants are not required to send data with a certain frequency as MKA uses dedicated MACsec Key Agreement Protocol Data Units (MKPDU). Furthermore, replay protection is ensured by the random Member Identifier each participant generates during start-up. If the random number generator of the participant in question has a high quality this participant is safe against replay attacks of MKA control messages.

There are though two decisive differences between the typical MKA application and the intended application in the 10 Mbit/s domain.

Firstly, while the MACsec Key Agreement is specified for an arbitrary number of participants (or Peers according to IEEE 802.1) it was hardly every used with more than two.

Secondly, the MKA Hello time is two seconds. Consequently, a key cannot be agreed upon in less than two seconds[3] and will take three seconds on average[1].

While this Key Agreement time is acceptable in the typical Ethernet environment like a Data Center, it is not in an automotive or other more timing sensitive environments. There is no consensus how much communication delay caused by the addition of a security layer is acceptable. Opinions ranges from a couple of milliseconds to over one hundred milliseconds. It all depends on the specific use case, but the three seconds MKA has to offer on average are too slow for the majority of (automotive) use cases.

Faster MKA

All following optimizations have the goal to improve the Key Agreement time (i.e. the time span between all participants being online and all participants can communicate with each other) without violating the existing specification [2]. So, the MKPDU and especially all included Parameter Sets are not modified in any way and no “shall” and “can” rules of the IEEE specification are violated.

In 2021 Dr. Völker proposed [1] a couple of modifications to optimize the Key Agreement time. The basic idea behind them:
Classify MKPDUs based on their content and modify the send frequency accordingly.

The first intuitive rule set you can produce, is that each participant sends an updated MKPDU whenever it has news to send that helps to achieve a key agreement. These rules are:

[3] Exclusive lower bound.

(1) The Potential Peer or Live Peer List changed.
(2) A new Secure Association Key needs to be distributed.
(3) A new Secure Association Key has been installed and needs confirmation.
(4) A new participant wants to join the Connectivity Association.

We call this set of rules the **Basic Profile.**

As shown in [1] these modifications yield substantial results, reducing the time required for a Key Agreement below 30 milliseconds.

Welcome to multi-drop

Let us apply these modifications to MKA, scale up the number of participants to n and do a theoretical analysis of how many messages need to be exchange and how many messages each participant[4] must process.
If all participants come online simultaneously, each one will generate a MKA Hello message that includes its own randomly generated Member Identifier. Each participant receives n - 1 of these messages, process one at a time, add n - 1 entries to its Potential Peer List and send a response n – 1 times. In total n^2 messages are sent and n^2-n messages must be processes by each participant. As consequence this initial step of Key Agreement does not scale linearly with the number of participants.

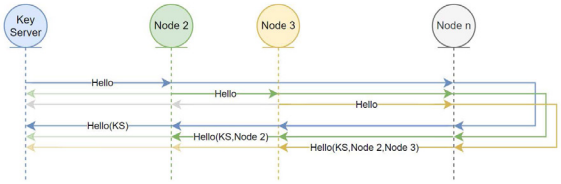

Figure 3: Basic Profile – Participant Discovery[5]

This problem can be solved if we modify our rules slightly to:

(1) A Key Server Peer[6] was added to the Potential Peer or Live Peer List.

We call this modified set of rules the Optimized Profile.

In a typical setup you only have one or two Key Server capable participants. This brings the number of messages that must be processed down to $\mathcal{O}(n)$.

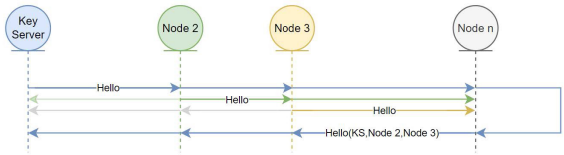

Figure 4: Optimized Profile – Participant Discovery

A similar phenomenon can be observed during key distribution. According to the MKA specification a Key Server shall distribute a new SAK if a participant is added to its Live Peer List. In our system start-up scenario, the Key Server therefore issues a new SAK when it gets the first MKA Hello response. And then another one, when it gets the second response. This continues until the Live Peer List finally contains the Member Identifiers of all participants. In total n – 1 SAK are distributed and only the last one is confirmed by all participants. Unfortunately, all SAKs are acknowledged by the participant whose MKA Hello was the first one to be processed by the Key Server. All SAKs but the first one, are acknowledged by the participant whose MKA Hello was second to be processed. This also continues until the last distributed SAK is finally accepted and acknowledged by all participants. In total each participant must process a $\mathcal{O}(n^2)$ number of messages to complete the Key Agreement.

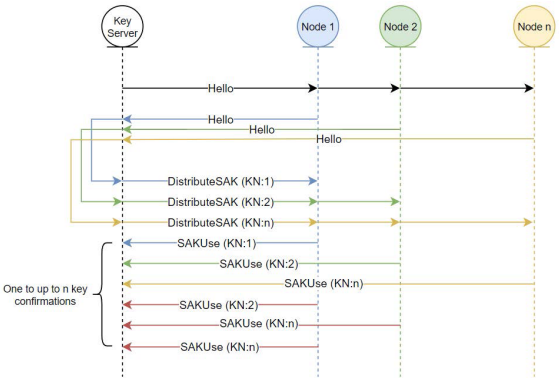

Figure 5: Basic Profile – Key Distribution

[4] Typically, eight to ten.
[5] Only Node n related MKA Hello Messages are shown.
[6] A Peer claiming to be able to perform the tasks of a Key Server.

To combat that problem the number of Key Distribution messages by the Key Server must be reduced. One option to realize that is to give the Key Server knowledge about the number of participants. With that knowledge the Key Server can intentionally delay the distribution of the SAK until it received the MKA Hello messages of all expected participants. This requires a completely static network setup and is for this reason not applicable for all use cases. Consider the case where one participant has a significantly[7] slower start-up behaviour preventing the rest of the network from communicating.
A second approach that conserves the MKA property of being network topology unaware is to throttle the issuing of new SAKs. After processing an incoming MKPDU the Key Server reaches a state that requires it to issue a new SAK. Instead of emitting the corresponding MKPDU immediately it delays this emission for certain time span[8]. This gives the Key Server time to process other incoming MKA Hello messages (s. Figure 6) and issue a SAK that can be used by more or even all participants. Let us add this behaviour to the Optimized Profile.

In simulations with sixteen participants this optimization reduced the number of distributed SAKs down to two and therefore eliminated the quadratic increase of runtime. The approach offers quite a bit of flexibility. Depending on the network setup you can choose the throttle time to optimize the Key Agreement times.

Examples:
(1) All participants are identically fast. You set the throttle time to a small value but big enough for the Key Server to process all MKA Hello responses.
(2) One participant is much slower than the rest but provides vital information. You set the throttle time to a value that gives the slow participant enough time to send its MKA Hello response.

Even better this configuration must only be applied to the Key Server(s) in case the attached participants do not offer this option, or you do not have access to its configuration.

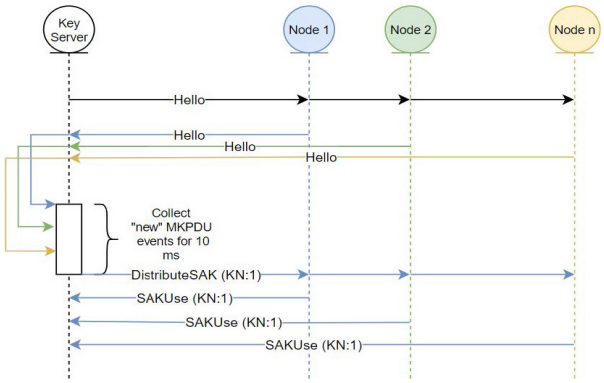

Figure 6: Optimized Profile – Key Distribution CANoe Simulation

We at Vector used our well-known tool CANoe to evaluate the effectiveness of the described tuning options.

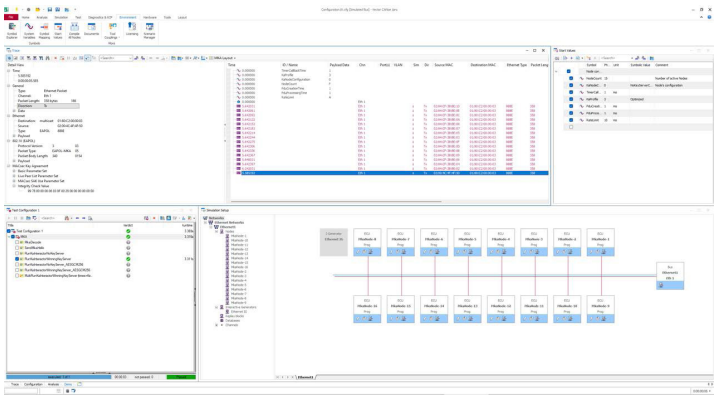

Figure 7: MKA Simulation in CANoe

The current simulation is capable of simulating MKA over both CAN XL and Ethernet with up to seventeen participants.

Various parameters are adjustable (e.g., processing time for one MKPDU and the throttle time).

The following table shows the results of running the simulation with both profiles and eight and sixteen participants with Ethernet as Data Layer.

Time denotes the timespan between the first sent message of the Key Server and the Key Server having received all participants' key acknowledgement message[9].

[7] More than forty milliseconds slower.
[8] Ten milliseconds yield appealing results in a simulation.
[9] MACsec SAK Use parameter set.

Messages denote the number of messages received by the Key Server within two seconds after it started the Key Agreement.

Participants	Basic Profile		Optimized Profile	
	Time	Messages[10]	Time	Messages[10]
8	47 ms	42	19 ms	14
16	162 ms	165	44 ms	53

Table 1: Simulation Results[11]

The numbers show that the Optimized Profile can reduce the number of needed messages significantly and because of this the Key Agreement time. Even more importantly it eliminates the quadratic runtime behaviour.


1. RunKaInteractorWinningKeyServer ()

Pass

Test Case Start: 16.02.2024 13:31:44 +01:00
Test Case Ende: 16.02.2024 13:31:48 +01:00
Ausführungsdauer: 00:00:04

Main Part

Zeitstempel	Testschritt	Titel	Verdict
3.698162		KaProfile used: Optimized	
3.698162		PduProcessingTime used: 1 ms	
3.698162		PduCreationTime used: 1 ms	
3.698162		RateLimit used: 10 ms	
5.698162		Waited for 2000 ms.	
5.698162		waitResult: 0	
5.698162		Time of Key distribution: 00: 00:03.7011 Time to Key distribution: 3 ms Time of Key agreement: 00:00:03.7421 Time to Key agreement: 44 ms ProtocolDataUnitsReceived: 53 ProtocolDataUnitsCreated: 17	
5.698162		IsKeyServer: true	Pass
5.698162		Has ActiveAssoziation	Pass
5.698162		Has no LatestAssoziation	Pass
5.698162		CipherSuite is CipherSuite.Default	Pass
5.698162		Has SecureAssoziationKey	Pass
5.698162		SecureAssoziationKey is 16 bytes long	Pass
5.698162		Key Number: 15, AssoziationNumber: Three	Pass
5.698162		LivePeers.Count is 15	Pass
5.698162		AllPeersAreLive is true	Pass

Figure 8: CANoe Test Report

MKA Alternatives

It depends on the requirements one has regarding a Key Agreement on top of what the name implies. If you have the protection goal that a Key Agreement is not attackable with valid recorded traffic from an isolated participant, the Key Agreement protocol requires some challenge response protocol part that enforces the inclusion of a value (the challenge) that is out of control of a potential attacker and has a good quality (is as random as possible).
There are two ways to achieve that. The first one requires a well synchronized common time source. This is a challenge itself and not discussed in this paper. The second one requires each participant to generate a random number, transmitting it to all other participants, which in turn must include all these random numbers (named Member Identifier in MKA) in their responses. This is exactly what MKA does. Every alternative must do something similar and inevitably requires at least one message round trip and therefore transmission plus processing time. Key Agreements with protection against this kind of replay attacks are not achievable without a setup time.

Conclusion

Simulation results clearly indicate that MKA can be optimized to a level that a Key Agreement in a typical network setup with up to sixteen participants can be realized under fifty milliseconds. This is a huge improvement compared to standard MKA considering that only sharper timings and an analysis of what is important has been applied.
The approach keeps all the positive properties of MKA (many features, well-proven, network agnostic) and makes it feasible for a lot of use cases that require faster communication ramp-up times.

End-to-End CANsec in Bridging Scenarios

There are use cases in which a Bridge Device is part of two or more CAN XL networks to enable the forwarding of messages from CAN Bus A to CAN Bus B and vice versa.

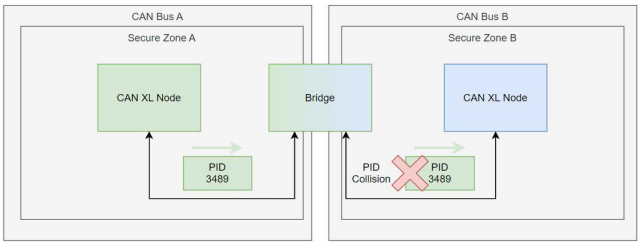

Figure 9: Bridged CAN XL networks

It can happen that a Priority Identifier of a message from CAN Bus A is already used by a CAN XL Node in CAN Bus B, so it cannot be forwarded unmodified without causing Priority Identifier collisions.

[10] Incoming messages processed by Key Server within two seconds after the Key Server sent its first message.
[11] Parameters of the simulation are documented in Appendix A - Simulation Parameters

If you want to secure such a system with CANsec, you can do this with distinct Connectivity Associations for Bus A and Bus B. The Bridge Device decrypts incoming messages, modifies the Priority Identifier as needed and re-encrypts the message. With Defense in Depth in mind this may be not the best option as this means that the Bridge Device must be in possession of the long-term secrets of both Associations which makes it a prime target for an attacker.

From a security point of view, it is desirable to have an End-to-End encryption, i.e., the Bridge Device does no cryptography and therefore needs no access to any keys. To enable such scenarios with CANsec the current CiA 613-2 draft specifies options to exclude the Priority Identifier and the Virtual CAN Identifier (VCID) from the integrity protection provided by CANsec.
While these options indeed make it possible to realize such scenarios, you should use these with extreme caution as it is of utmost importance that neither the Priority Identifier nor the VCID contain semantic information.

Never use Priority Identifier as an Identifier

One of the improvements of CAN XL over classic CAN and CAN FD that it breaks the semantically unfortunate double purpose of the Identifier field. Before CAN XL the Identifier field was both a priority instruction to the physical layer and a message identifier[12]. Therefore, you cannot alter the Identifier field without changing the meaning of the message. With CAN XL the priority instruction is the Priority Identifier, and the Acceptance Field (AF) takes responsibility to identify the message's meaning.
In a perfect world, system engineers designing a CAN XL network have this in mind und do not mix up the independent purpose of the two fields. If this is the case, CANsec supports bridging with End-to-End encryption with its exclude options. But what happens if at least one CAN XL Node is a simple migration of an existing CAN FD implementation? It is likely that the Priority Identifier is just the CAN Identifier of the legacy project because it was the easiest way to migrate the project to CAN XL[13]. The Priority Identifier has more meaning than desired, and you cannot bridge the CAN XL messages without losing the CANsec protection.
Sadly, the current CANsec draft does not offer a secure solution for this scenario.
Another disadvantage of CANsec's exclude options is that all Nodes in the network must be configured in advance that certain messages are intended to be forwarded and exclude the Priority Identifier from its ICV for that reason. It is not possible to attach a second CAN Bus via a Bridge Device without altering the configuration which may be not possible because you do not have access to all Nodes. This limits extension options.

CAN-in-CAN

If you have a scenario like this where the Priority Identifier[14] conveys semantic information and/or want to have the flexibility to add a Bridge Device optionally or temporarily, you need a solution that satisfies the following requirements:
(1) Nodes do not need to know that their message may be cross the boundary of their CAN XL network.
(2) All fields of the CAN XL Header are protected by CANsec.

A solution that meets above requirements and does not break the end-to-end encryption is CAN-in-CAN[15]:
The configuration of the originator network stays unmodified, so every Node transmits CANsec protected Frames. The Bridge Device identifies Frames to be forwarded and uses the whole CAN XL Frame including its Header data as payload for a new "wrapped" Frame with a new CAN XL Header[16]. Figure 10 illustrates the concept.

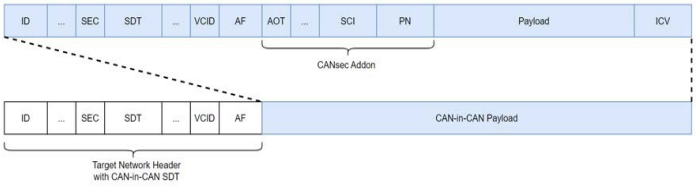

Figure 10: CAN-in-CAN

[12] Defining what the meaning of the message is.
[13] Simply use the CAN Identifier for both Priority Identifier and Acceptance Field to avoid that problem.
[14] Or the Virtual CAN Identifier.
[15] No CiA specification available.
[16] This target network Frame can be CANsec protected if desired.

The Nodes in the receiving network identify a forwarded frame by its special Service Data Type[17], remove the Target Network Header and can now validate the unmodified CANsec protected Frame. None of its content (incl. Priority Identifier and VCID) can be manipulated without invalidating the Frame.

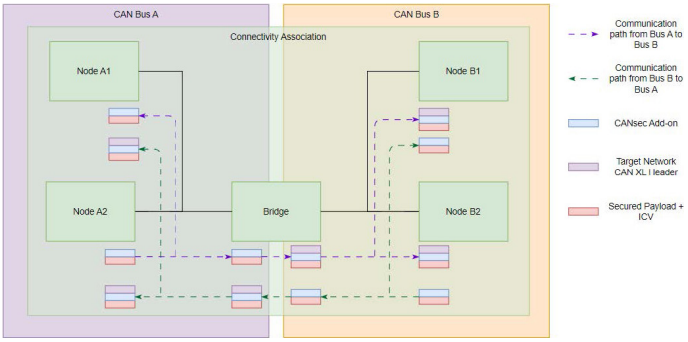

Figure 11: Communication Example

As MKA control messages must pass through the bridge the same way, all participating Nodes must be capable of the CAN-in-CAN concept.

Conclusion

While the CAN-in-CAN concept requires twelve extra bytes in the CAN XL payload it offers more flexibility and a possibility to securely bridge CAN XL networks with legacy components.

Appendix A - Simulation Parameters

PduProcessingTime	1 ms
PduCreationTime	1 ms
RateLimit	10 ms

References

[1] Starting Up MACsec for Automotive Ethernet, Dr. Lars Völker
[2] IEEE802.1X-2020
[3] Ascon - Lightweight Authenticated Encryption & Hashing

[17] Which is not standardized.

Peter Decker
Vector Informatik
Holderäckerstr. 36
DE-70499 Stuttgart
0711/80670-3615
Peter.Decker@vector.com
www.vector.com

Sven Hoffmann (Dipl.-Math.) has been employed at Vector Informatik GmbH since 2021 and serves as Senior Software Engineer in the Research and Development department.