Data Distribution and Redistribution - A formal and practical Analysis of the DDS Security Standard

Timm Lauser
timm.lauser@h-da.de
Darmstadt University of Applied Sciences
Darmstadt, Germany

Maximilian Müller
m.mueller@fzi.de
FZI Research Center for Information Technology
Karlsruhe, Germany

Ingmar Baumgart
baumgart@fzi.de
FZI Research Center for Information Technology
Karlsruhe, Germany

Christoph Krauß
christoph.krauss@h-da.de
Darmstadt University of Applied Sciences
Darmstadt, Germany

Abstract

The Data Distribution Service (DDS) is a popular communication middleware for the Internet of Things (IoT), providing its own security mechanisms specified in the DDS Security standard. In this work, we formally analyze the authentication handshake protocol and the encryption algorithm used in DDS. We discover a replay vulnerability in the encryption algorithm, implement a proof-of-concept attack on an open-source implementation of DDS, and review security-relevant changes in the recently published version 1.2.

CCS Concepts

• **Security and privacy** → **Security protocols**; **Authentication**; **Formal security models**; **Embedded systems security**.

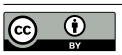
Keywords

security; formal analysis; data distribution service; IoT; automotive

ACM Reference Format:
Timm Lauser, Maximilian Müller, Ingmar Baumgart, and Christoph Krauß. 2025. Data Distribution and Redistribution - A formal and practical Analysis of the DDS Security Standard. In *The 40th ACM/SIGAPP Symposium on Applied Computing (SAC '25), March 31-April 4, 2025, Catania, Italy.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3672608.3707869

1 Introduction

The Data Distribution Service (DDS) is a popular communication middleware standard for industrial Internet of Things (IoT) used in many application domains like transportation, smart energy, and industrial automation, among others. Moreover, in automotive, it is sometimes considered a secure alternative to Scalable Service-Oriented Middleware over IP (SOME/IP) as middleware for Ethernet-based communication in modern vehicles. Contrary to SOME/IP (cf. [20, 8]), DDS has a dedicated security standard providing authentication and access control capabilities.



SAC '25, March 31-April 4, 2025, Catania, Italy

ACM ISBN 979-8-4007-0629-5/25/03
https://doi.org/10.1145/3672608.3707869

Formal verification has been proven to be a powerful tool for verifying the security of communication protocols. A first formal analysis of DDS Security was recently published by Wang, Li and Guan [18]. They provide the first formal description of the protocol flow, formalize security goals, and analyze the defined protocols with the ProVerif [2] prover. In parallel to their work, we independently extracted a formal description of the authentication and encrypted communication protocol of DDS Security and conducted a formal verification with the Tamarin prover [1].

While the formal model of DDS Security in [18] describes one overall protocol flow, we concentrated on modeling the authentication and encryption flow in two separate, more in-depth models. As a result, we discovered a novel replay attack in the encrypted communication protocol. In contrast to [18], who only analyzed DDS Security 1.1 [14], we also consider version 1.2 [15], which was released in 2024. In addition, we provide a Proof of Concept (POC) implementation for our discovered attack in an open-source implementation of DDS.

In summary, we provide the following contributions:

(1) A formal description of the authentication protocol and encryption algorithm used in the built-in DDS plugins mandated by DDS Security.
(2) A formal analysis of these algorithms in the symbolic model with the Tamarin prover, unveiling a novel replay attack.
(3) We implement our discovered attack in OpenDDS [16], and evaluate the practicability of the attack.
(4) We discuss the changes in DDS Security 1.2 and provide a Tamarin model for the newly introduced pre-shared key protection mechanism.
(5) We publish our Tamarin models and attack implementation as open source.

Roadmap. The paper is structured as follows. In Section 2, we present related work. Next, we introduce the basic concepts of DDS in Section 3, followed by a discussion on DDS Security 1.1 and its built-in plugins in Section 4. Our formal analysis of the authentication and encryption plugin is presented in Section 5. In Section 6, we present a novel replay attack, its implementation and mitigation. Section 7 discusses the changes in DDS Security 1.2 and our formal model of it. We conclude the paper in Section 8.

2 Related Work

Before the DDS Security standard was released, He and Liang [7] analyzed the DDS specification for security issues and put forward a scenario where unauthorized publishers or subscribers may be able to inject data into the DDS network or receive data not intended for the legitimate recipient. They present a high-level overview of theoretical attacks on DDS and it is these types of attacks that DDS Security has been designed to mitigate.

White et al. [19] demonstrate that unauthenticated clients can enumerate connected devices within a DDS Domain protected by DDS Security.

Friesen et al. [6] compare the DDS Security standard to the previous state of the art to use TLS/DTLS to secure DDS communication. They conclude that DDS Security employs the same cryptographic primitives to protect authenticity, integrity, and confidentiality as (D)TLS, however, in addition, it provides an access control mechanism. Moreover, via Quality of Service (QoS) policies, DDS can additionally protect the availability by enabling redundancy through QoS policies. They highlight the possibility of using different protection mechanisms based on the DDS Topic.

Kim et al. [9] propose to replace the data-based access control mechanism in DDS Security with an ABAC-based approach.

Maggi et al. [12] analyze the security of six common DDS implementations, identifying several vulnerabilities. While most of these are implementation-specific, they also identified an amplification vulnerability in the DDS Standard, where a single, maliciously crafted discovery message could be used to initiate a network flooding attack.

Recently, Wang et al. [18] published the first formal analysis of DDS Security 1.1. They extract a formal protocol description from the DDS Standard and modeled the protocol with ProVerif [2], a well-established automated verification tool for security protocols. They distinguish between the classical Dolev-Yao attacker model, where the attacker controls the communication channel but cannot compromise legitimate protocol participants, and the malicious participant model, which enables further attacks. Their analysis identified the following attacks:

(1) *Permission File Impersonation Attack:* The protocol fails to verify that the user of a valid permission document is the actual owner of the document.
(2) *Denial of Service (DoS) Attack:* Attackers can forge messages to initiate a high number of authentication attempts, resulting in a high workload for performing cryptographic operations on the victim.
(3) *Discovery Service Disruption Attack and Degradation Attack:* The participant discovery mechanism in DDS is unprotected, and thus, can be disturbed by an attacker.
(4) *Privacy Leakage Attack:* Privacy goals are not explicitly defined. Some of the information transferred in plain, especially permission files and topic identifiers, may contain private data.

While their work is very similar to ours, our work has a narrower scope, focusing on the authentication and encryption mechanism in DDS Security and discovering a novel replay attack. Additionally, we provide a POC implementation of our discovered attack, while [18] only provides a theoretical analysis. For the malicious participant model, which corresponds to the model we use in this work, they claim that all their security goals could not be met, however, they do not describe their attacks in detail. This contradicts our results as described in Section 5.

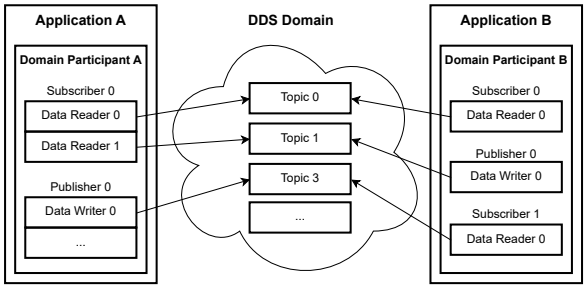


Figure 1: DDS domain with two participants

In automotive, DDS is considered an alternative to the SOME/IP middleware. In contrast to DDS, the SOME/IP standard does not provide its own security mechanisms. A formal verification of SOME/IP with the Tamarin prover can be found in [20]. In addition, [20] and [8] both proposed a formally verified security extension for SOME/IP.

3 DDS

DDS is a publish-subscribe communication middleware. Each self-contained communication is called a DDS Domain. In a DDS Domain, there are six types of entities, namely *Domain Participant (DP)*, *Publisher*, *Subscriber*, *DataWriter*, *DataReader* and *Topic* (Figure 1). Typically, there is one DP per application. Each DP may contain an arbitrary number of Publishers and Subscribers. A Publisher comprises one or more DataWriters to send data. Each DataWriter is bound to a single Topic that describes the data, its type, and QoS. The data is received by DataReaders that are, like DataWriters, bound to a single Topic. DataWriters only receive messages that are bound to the same topic they are bound to. A DataReader is managed by a Subscriber which may possess several DataReaders. Publishers send data directly to Subscribers without any middle-broker required.

DataReaders and DataWriters can also be configured by QoS, which can be described as configuration parameters. Through QoS, important message attributes like reliability or data durability are guaranteed. Only DataReaders and DataWriters with compatible QoS (and Topic) can communicate.

For communication, the Real-Time Publish-Subscribe (RTPS) on-the-wire protocol [13] is used. Hereby, an *RTPS Message* is composed of a *RTPS Header* and potentially multiple *RTPS SubMessages* (Figure 2). Each SubMessage consists of a *SubMsg Header* and multiple *SubMsg Elements*. The actual application data is contained in a SubMessage of type *Data* (or *DataFrag* if the data has to be fragmented into several SubMessages). In the *Data* SubMessage, there are several SubMessage elements, such as sequence numbers, inline QoS, and the *SerializedPayload*, containing the actual application data.

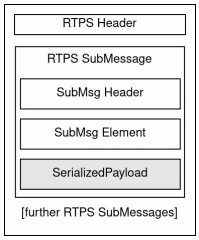


Figure 2: RTPS message format [15]

4 DDS Security

This section discusses the DDS Security standard, focusing on Version 1.1. For information on changes introduced in Version 1.2, please refer to Section 7. DDS Security provides a plugin architecture that augments DDS with security-related functionality. These plugins allow authenticating applications and enforce access control decisions for actions such as joining a DDS Domain or reading or writing a *Topic*. DDS Security names the following security goals:

- S_1 *Confidentiality of data samples:* Only authorized domain participants can obtain the exchanged data values.
- S_2 *Integrity of data samples and messages:* Only authorized modifications of data values are possible. DDS messages cannot be modified in transit.
- S_3 *Authentication of DDS writers and readers:* The identity of DataWriters and DataReaders is verified before they can participate in the communication.
- S_4 *Authorization of DDS writers and readers:* Fine-grained permission management for reading and writing data is possible and enforced.
- S_5 *Message-origin authentication:* Received messages are only accepted if they can be verified to originate from the claimed DP.
- S_6 *Data-origin authentication:* It can be verified that specified data originated from the claimed DataWriter or DataReader.
- S_7 *Non-repudiation of selected data (optional):* Neither sender nor recipient can deny their processing of specific data. Therefore, the sender of the data obtains a proof of delivery, and the recipient a proof of the sender's identity.

DDS Security provides fine-grained control over which elements of a message should be protected. It is possible to protect only the SerializedPayload, whole SubMessages, or whole RTPS Messages.

The functionality of DDS Security is split into 5 different plugins (where the last two are optional):

Authentication Authenticated key exchange based on a preconfigured Public Key Infrastructure (PKI).

AccessControl Access control based on a permissions document signed by the Permissions Certificate Authority (CA).

Cryptography Encryption and Message Authentication Code (MAC) calculation for messages.

DataTagging Allows to add tags to data samples.

Logging Logging of security events to a dedicated DDS topic.

While it is possible to define own implementations of these plugins, in this paper we analyze the security of the so-called *built-in* plugins, which are specified in the standard and mandatory for all implementations. In addition, we focus on the mandated plugins and do not consider the DataTagging and Logging plugins here.

For a better understanding of the interoperation of these plugins, we exemplary describe their application: Assume a DDS Domain that uses DDS Security and includes participant A. Participant B wants to join the domain and, thus, runs the discovery protocol. B then discovers A and queries the AccessControl plugin, if authentication is required to join the domain. If yes, B tries to validate the identity of A. At the same time, A discovers B and initiates the validation of B's identity. A and B then perform a three-way handshake protocol as defined by the authentication plugin. During the handshake, A and B mutually authenticate each other, as well as their respective permissions within the domain, and exchange a shared secret. If the authentication fails, A and B will not consider each other as part of the same domain.

If B now creates a new Topic within the DDS domain, A will use the AccessControl plugin to verify that B is allowed to create a topic with this name. All following publish/subscribe communication is then protected using the Cryptography plugin, which is used to transform unsecured RTPS messages into secured messages based on access control policies specified in a domain governance document.

4.1 Authentication Plugin

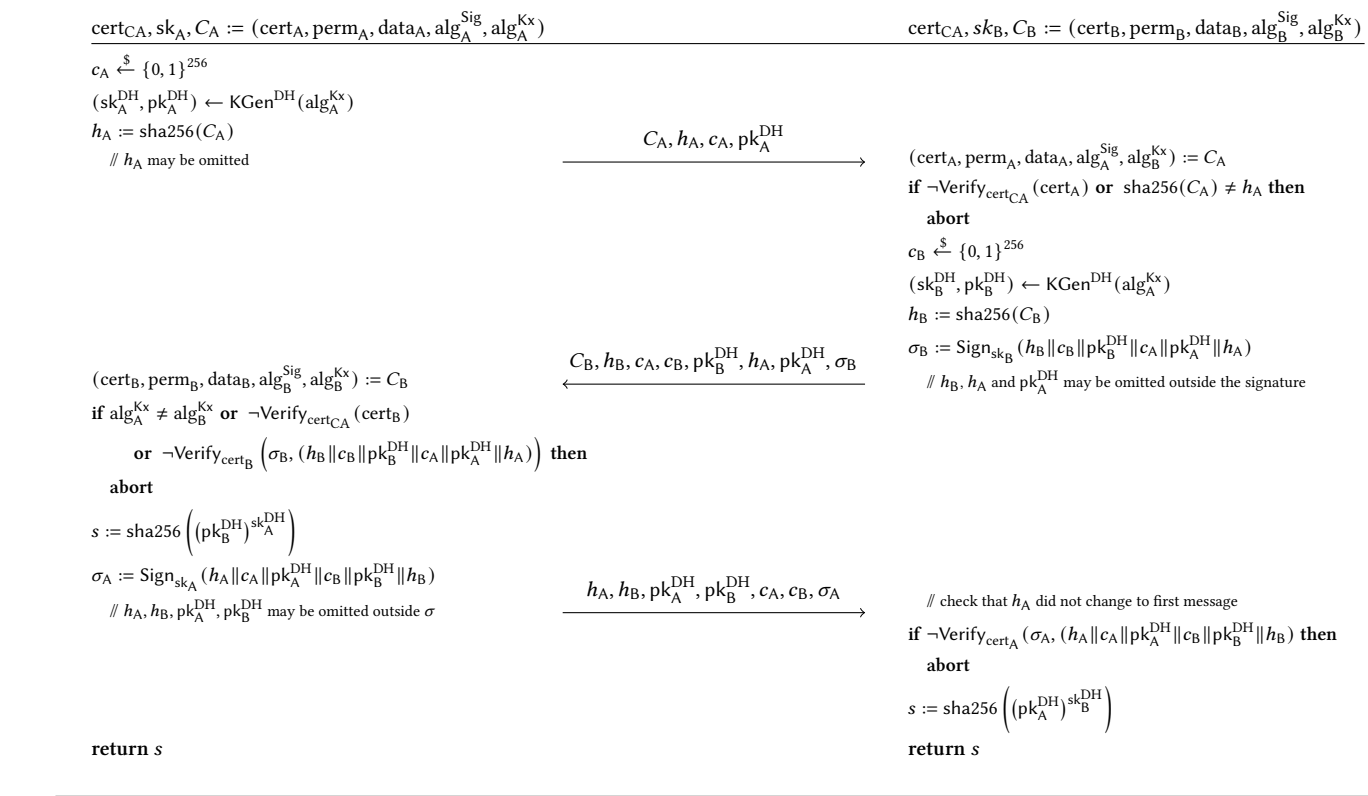
The Authentication plugin enables mutual authentication between a participant joining a DDS domain and domain members and establishes a shared secret between them.

This is achieved by a handshake protocol executed between the principal joining the DDS domain and a principal within the domain. The handshake protocol for the built-in Authentication plugin is shown in Figure 3.

The bidirectional authentication within the handshake protocol uses a shared identity certificate authority CA as trust anchor. CA's certificate is known to all domain members. In addition, each domain participant P has a long-term private key sk_P and a corresponding certificate cert_P, containing P's public key, signed by CA. At the start of the protocol, the participant A with the lowest Globally Unique Identifier (GUID) randomly chooses a new challenge c_A, computes a new Diffie Hellman (DH) key pair, and sends the challenge c_A, the public DH key $\mathrm{pk}_\mathrm{A}^\mathrm{DH}$, and C_A to the other participant B. Hereby, $C_\mathrm{A} := (\mathrm{cert}_\mathrm{A}, \mathrm{perm}_\mathrm{A}, \mathrm{data}_\mathrm{A}, \mathrm{alg}_\mathrm{A}^\mathrm{Sig}, \mathrm{alg}_\mathrm{A}^\mathrm{Kx})$ contains A's certificate cert_A, permission document perm_A, topic data data_A, and selected signature and key exchange algorithms. B then verifies cert_A, chooses a new challenge c_B, computes a new DH key pair and sends this information to A together with C_B and a signature σ_B over C_B, C_A and both challenges and public DH keys. A verifies σ_B together with cert_B and computes her own signature over this information and sends it together with c_A and c_B to B. Now, both parties compute a shared DH key and derive the shared secret s.

4.2 AccessControl Plugin

The built-in AccessControl plugin consists of a Permissions CA, a domain governance document (*governance file*), and a *permissions file* for each domain participant. The permissions CA is a shared CA which may be the same as the identity CA. Moreover, the plugin defines several protection kinds, which can be used to secure



- cert_{CA} X.509 certificate of the certificate authority CA, containing the public key pk_{CA}
- sk_A Private key of participant A, corresponding to pk_A. In storage, it is encrypted with AES128-CBC under a base64-encoded password.
- cert_A X.509 certificate of participant A, it contains the public key pk_A and is signed by the corresponding certificate authority
- perm_A Permission document of participant A
- alg_A^{Sig} Signature algorithm used by participant A. Either 2048 Bit RSA or 256 Bit ECDSA with the curve prime256v1 (NIST P-256).
- data_A The so-called ParticipantBuiltinTopicData contains, among others, an identity token, a permissions token, information about the cryptographic algorithms supported by the participant, and supported built-in endpoints.
- alg_A^{Kx} The key agreement algorithm selected by A. B shall set alg_B^{Kx} to the same algorithm.
- c_A A challenge randomly selected by participant A
- $(\text{sk}_A^{DH}, \text{pk}_A^{DH})$ Diffie-Hellman secret and public key of participant A, respectively.
- s Shared secret that is established between the participants A and B

Figure 3: DDS Security handshake protocol (cf. [15])

data. These protection kinds are referenced in the governance and permissions files.

4.2.1 Permissions CA. The Permissions CA is represented by an X.509 certificate which contains the public key of the CA. The corresponding private key is used to sign the governance and permissions files.

4.2.2 Protection Kinds. There are 3 basic and 2 advanced protection kinds in the AccessControl plugin. The basic ones are `NONE`, `SIGN`, and `ENCRYPT`. `NONE` applies no cryptographic transformation on the data, `SIGN` appends a MAC (AES-128-GMAC or AES-256-GMAC) to the data and `ENCRYPT` first encrypts the data and then computes a MAC on the encrypted data. The latter is also known as Encrypt-then-MAC and the algorithms AES-128-GCM or AES-256-GCM are used for this operation. The advanced protection kinds are `SIGN_WITH_ORIGIN_AUTHENTICATION` and `ENCRYPT_WITH_ORIGIN_AUTHENTICATION`. These protection kinds guarantee that a message stems from the supposed sender, preventing other receivers from impersonating the sender.

How these protection kinds are applied can be seen in the description of the AES-GCM-GMAC algorithm in Section 4.3.2.

4.2.3 Domain Governance Document. The Domain Governance Document is an XML file that specifies how the domain shall be secured. This file is signed by the Permissions CA using S/MIME (version 3.2) and the signer certificate is included in the signature.

In this file, the following settings can be configured. The options are either true or false (T/F) or a protection kind.

domains The IDs of the domains to be secured.
allow_unauthenticated_participants (T/F) Indicates whether unauthenticated participants can join the domain and see unprotected topics and discovery data.
enable_join_access_control (T/F) Indicates whether an authenticated user may join the domain and see the discovery data without checking the access control policies.
discovery_protection_kind Protection kind for discovery messages.
liveliness_protection_kind Protection kind for liveliness messages.
rtps_protection_kind Protection kind for the whole RTPS message. This is in additon to any protection applied on single RTPS SubMessages, for example, the ones defined in the topic access conditions.

Additionally, there are topic access rules:

topic_expression Identifies the topics to which the rules apply.
enable_discovery_protection (T/F) Indicates whether discovery messages shall be protected.
enable_read_access_control (T/F) Indicates whether all participants may read the topic or access control shall be considered.
enable_write_access_control (T/F) Indicates whether all participants may write to the topic or access control shall be considered.
metadata_protection_kind Protection kind for the topic metadata, e.g., sequence numbers, heartbeats, etc.
data_protection_kind Protection kind of the payload data (RTPS *SerializedPayload* SubMessage) sent to the topic.

A rule only applies on a DP if the DP belongs to a Domain with an ID protected by the governance file. If multiple rules match, only the first one is evaluated.

4.2.4 Permission Document. Permissions for domain participants are encoded within an XML document referencing their distinguished name. Permissions include, e.g., publication, subscription, and/or relay of topics, partitions, or data tags. The document is signed with S/MIME (version 3.2) by the Permissions CA and the signer certificate is included within the signature.

To validate permissions, the Permissions plugin first checks that the subject name in the permission document correctly matches the remote participant's identity certificate. Then it validates the signature of the permissions document with the permissions CA's public key. If both succeed, the permissions defined in the permission document are locally cached. The cache will then be referred to in future access decisions.

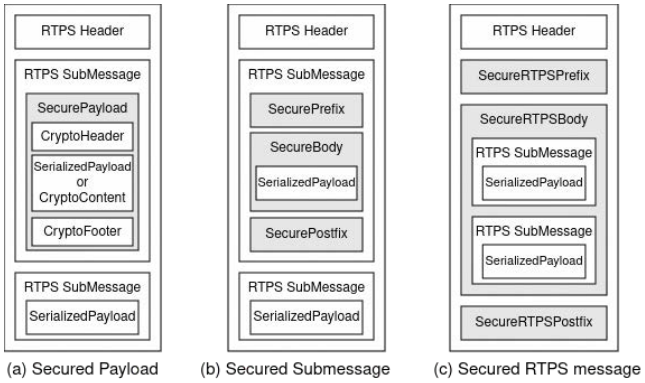


Figure 4: RTPS message structure with different security configurations [15]

4.3 Cryptography Plugin

The built-in Cryptographic plugin is referred to as *DDS:Crypto:AES-GCM-GMAC plugin*. It introduces types and operations necessary to support encryption, digest, MACs, and key exchange for DPs, DataWriters, and DataReaders.

4.3.1 Message formats. DDS Security provides a high level of flexibility for the scope of the applied protection mechanisms. Some applications may choose to only authenticate (or encrypt) the *SerializedPayload*, while other applications may secure additional metadata, such as sequence numbers, or the whole *RTSP Message*. An overview of message formats is given in Figure 4.

This fine-granular control over which elements of a message should be protected is executed via the governance document (see Section 4.2.3).

Depending on the specified protection scope, different transformations are applied to the RTPS message. In general, those transformations add a prefix and a postfix to the element under protection. The prefix contains cryptographic keys or identifies the applied cryptographic algorithm, while the postfix contains one or multiple authentication tags.

Securing the *SerializedPayload* (authentication or encryption) is handled by the operation *encode_serialized_payload* (Figure 4a). This operation obtains the payload serialized in a *SerializedPayload* submessage element and outputs an RTPS *SecurePayload*[1] SubMessage element. This element comprises a *CryptoHeader*, the (optionally encrypted) *SerializedPayload*, and a *CryptoFooter* element. If the SerializedPayload is encrypted, the element containing the encrypted data is called *CryptoContent*.

A RTPS submessage can be secured by the operations *encode_datawriter_submessage* and *encode_datareader_submessage* (Figure 4b). The output of such functions typically contains a *SecurePrefix*, followed by a *SecureBody* and a *SecurePostfix*.

Finally, a whole RTPS message can be protected by the function *encode_rtps_message* (Figure 4c). The functions return an RTPS

[1]In the specification this submessage element is called CryptoContent. However, this is ambiguous since in the specification, the SubMessage element, which only contains the ciphertext of the SerializedPayload is also called CryptoContent. Therefore, we rename this SubMessage element to SecurePayload.

Table 1: Notations used for AES-GCM-GMAC

mk	Master sender key
msalt	Master salt
$\mathsf{ID}_{\mathsf{mk}}$	Master key ID
IV	96-bit initialization vector (session ID + suffix)
IV^s	Initialization vector suffix (incremented with each encryption or MAC calculation)
$\mathsf{ID}^{\text{session}}$	Session ID (initially random, incremented when a new session key is needed)
$\mathsf{k}^{\text{session}}$	Session key (derived from master key, salt, and session ID using HMAC-SHA256)
sbc	Session block counter (counts blocks ciphered with the current session key)
$\mathsf{ID}_{\mathcal{R}}$	Receiver ID
$\mathsf{mk}_{\mathcal{R}}$	Master receiver-specific key
$\mathsf{k}_{\mathcal{R}}^{\text{session}}$	Receiver-specific session key (derived from master receiver key, salt, and session ID using HMAC-SHA256)
$\mathbb{R}$	Set of receivers
t, $t_{\mathcal{R}}$	Common and receiver-specific MAC tags (computed with master or receiver-specific keys)
$\vec{t}_{\mathbb{R}}$	Set of tuples containing receiver IDs and corresponding MAC tags

message, beginning with an *RTPS Header*. In the authentication-only case, the *RTPS SubMessages* follow in plaintext. In the case of encrypted authentication, all SubMessages are encoded in one element called *SecureRTPSBody* which contains the ciphertext. At the end of the RTPS message, there is a *SecureRTPSPostfix* element, containing the MACs.

4.3.2 AES-GCM-GMAC. The built-in Cryptographic plugin supports authenticated encryption using Advanced Encryption Standard (AES) in Galois Counter Mode (AES-GCM). It also supports creating MACs using Galois MAC (GMAC).

The AES-GCM encryption can be described as follows: $(c, t) := \text{aes-gcm}_{\mathsf{k}}^{\mathsf{IV}}(m, \text{AAD})$, where a plaintext m is encrypted with a symmetric key k and an Initialization Vector (IV), denoted by IV, to form a ciphertext c. Moreover, the authentication tag t is computed by a MAC over the ciphertext and the Additional Authenticated Data (AAD) which is only authenticated and not encrypted. AES-GCM can also be used to only authenticate data. In this case, m is supposed to be empty and the operation is denoted by $\text{aes-gmac}_{\mathsf{k}}^{\mathsf{IV}}(\text{AAD}) = \text{aes-gcm}_{\mathsf{k}}^{\mathsf{IV}}(\emptyset, \text{AAD})$. Further notations are defined in Table 1.

Encryption. The DDS:Crypto:AES-GCM-GMAC encryption algorithm, which is described in Figure 5 works as follows:

First, the IV is built by concatinating the session ID and the initialization vector suffix. This IV will be used for all the following operations associated with a specific sender. There are different session keys used in the operations (the session key $\mathsf{k}^{\text{session}}$ and the receiver-specific session keys $\mathsf{k}_{\mathcal{R}}^{\text{session}}$).

To ensure that at most *max_blocks_per_session* blocks are encrypted with the same session key, an internal counter *sbc* is used. If *sbc* plus the block size of the (possibly padded) plaintext to be encrypted exceeds *max_blocks_per_session*, new session keys are

Encrypt(m, $\mathbb{R}$)

// Calculate new session keys if exceeding session block counter
if $sbc + block\_size(m) > max\_blocks\_per\_session$ **then**
  $\mathsf{ID}^{\text{session}} := \mathsf{ID}^{\text{session}} + 1$
  $sbc := 0$
  $\mathsf{k}^{\text{session}} := \text{hmac}_{\mathsf{mk}}^{\text{sha256}}(\texttt{"SessionKey"} \| \mathsf{msalt} \| \mathsf{ID}^{\text{session}})$
  foreach $\mathcal{R}_i \in \mathbb{R}$ **do**
    $\mathsf{k}_{\mathcal{R}_i}^{\text{session}} := \text{hmac}_{\mathsf{mk}_{\mathcal{R}_i}}^{\text{sha256}}(\texttt{"SessionReceiverKey"} \| \mathsf{msalt} \| \mathsf{ID}^{\text{session}})$
  endforeach
endif
$\mathsf{IV} := \mathsf{ID}^{\text{session}} \| \mathsf{IV}^s$ // Construct 96-bit IV
$(c, t) := \text{aes-gcm}_{\mathsf{k}^{\text{session}}}^{\mathsf{IV}}(m, \emptyset)$ // Encrypt and mac m under the master key
$sbc := sbc + block\_size(m)$ // increase session block counter
// compute receiver macs from their session keys and IV (only use auth tags from AES-GCM)
$\vec{t}_{\mathbb{R}} := \left\{ \left(\mathsf{ID}_{\mathcal{R}_i}, t_{\mathcal{R}_i} := \text{aes-gmac}_{\mathsf{k}_{\mathcal{R}_i}^{\text{session}}}^{\mathsf{IV}}(t) \right) \middle| \mathcal{R}_i \in \mathbb{R} \right\}$
$\mathsf{IV}^s := \mathsf{IV}^s + 1$ // increase IV suffix for the next message
return $(IV \| c \| (t, \vec{t}_{\mathbb{R}}))$ // header = IV, content = c, footer = $(t, \vec{t}_{\mathbb{R}})$

Figure 5: DDS:Crypto:AES-GCM-GMAC encryption algorithm (c.f. [15])

generated by incrementing $\mathsf{ID}^{\text{session}}$ and calculating new key material. Moreover, *sbc* is then reset to 0.[2]

The ciphertext c and authentication tag t are then computed as $(c, t) := \text{aes-gcm}_{\mathsf{k}^{\text{session}}}^{\mathsf{IV}}(m, \emptyset)$, where m is the message to be encrypted. Additionally, *sbc* is incremented by the block size of m.

If origin authentication is desired, receiver specific MACs are computed by applying a GMAC on the tag t, that is calculating $t_{\mathcal{R}_i} := \text{aes-gmac}_{\mathsf{k}_{\mathcal{R}_i}^{\text{session}}}^{\mathsf{IV}}(t)$, for each receiver $\mathcal{R}_i$.

Finally, the IV is incremented and the algorithm outputs the ciphertext. The final ciphertext consists of the IV in the *CryptoHeader*, c in the *CryptoContent*, and t (and optionally $\vec{t}_{\mathbb{R}}$) in the *CryptoFooter*.

Decryption. The receiver $\mathcal{R}$ looks up the master key mk and master salt msalt from header information (the *transformation_key_id*). It then parses the IV from the CryptoHeader to obtain the session ID $\mathsf{ID}^{\text{session}}$. With the obtained session ID, the receiver computes the session key $\mathsf{k}^{\text{session}}$ and optionally its session receiver-specific key $\mathsf{k}_{\mathcal{R}}^{\text{session}}$. It then decrypts the ciphertext c and verifies the common MAC with $\mathsf{k}^{\text{session}}$ and, if received, verifies its receiver-specific MAC $t_{\mathcal{R}}$ with $\mathsf{k}_{\mathcal{R}}^{\text{session}}$.

There are no checks whether the $\mathsf{IV} = (\mathsf{ID}^{\text{session}}, \mathsf{IV}^s)$ is valid. For example, an IV suffix may be reused and not incremented after an encryption operation.

[2]Theoretically, IVs would be reused if *max_blocks_per_session* could exceed the possible values of the 64 Bit IV^s. However, this is not possible as *max_blocks_per_session* is defined as a 64 Bit integer.

Key Exchange. To exchange topic-specific master sender keys and receiver-specific keys, required for encryption and authentication as described above, DDS Security defines a special pair of Readers/Writers, the *BuiltinParticipantVolatileMessageSecureWriter* and *-Reader*. These are used to establish a secure channel between two participants to exchange key material, such as the master sender key and receiver-specific key for a topic between the publisher and the subscriber. Therefore, they derive their master key mk and master salt msalt directly from the shared secret s and the challenges c_A and c_B exchanged during the authentication handshake. In this case, no (additional) receiver-specific key is used.

Hereby, the master sender key mk and master salt msalt are computed as follows:

$$\mathsf{mk} := \mathsf{hmac}^{\mathsf{sha256}}_{\mathsf{key}}(s),$$

where $\mathsf{hmac}^{\mathsf{sha256}}_{\mathsf{key}}$ denotes an Hash-based Message Authentication Code (HMAC) using SHA256 as hash function and key = sha256 ($c_B \| \texttt{"key exchange key"} \| c_A$) as key.

$$\mathsf{msalt} := \mathsf{hmac}^{\mathsf{sha256}}_{\mathsf{key2}}(s),$$

where key2 = sha256($c_A \| \texttt{"keyexchange salt"} \| c_B$),

Please note that A and B refer to the roles in the authentication handshake. Thus, mk and msalt are equal, independent of wheater A or B sends the message.

5 Formal Security Analysis

We conduct our formal analysis of DDS Security with the Tamarin prover [1], a state-of-the-art tool for automated formal verification in the symbolic model. All our Tamarin models are available in an online repository.[3]

In the symbolic model, cryptographic primitives such as encryption or MAC algorithms are assumed to be perfectly secure, abstracting from their computational security properties. Instead, the verification focuses on the composition of these primitives, their input and output, and the general protocol flow. This is combined with a powerful attacker that has complete control over the network as well as the ability to corrupt entities in the system. See [10] for a discussion and overview of formal verification tools and security properties relevant to automotive protocols.

In the following, first, we shortly introduce our attacker model and define formal security properties, before we describe our formal analysis of the authentication handshake protocol and the encryption protocol specified by DDS Security.

5.1 Attacker Model

We consider a Dolev-Yao-style attacker [5] with complete control over the network. The attacker can read and modify any message that is exchanged and inject forged messages. In addition, the attacker can corrupt an arbitrary number of domain participants. In our model, this means that the attacker obtains access to the participant's long-term secrets. Please note that the attacker can trivially impersonate any corrupted party, thus, security requirements are only required to hold if the parties in question remain *honest*. Honest parties follow the protocol specification and have not been corrupted by the attacker.

[3] https://code.fbi.h-da.de/seacop/dds-tamarin

5.2 Security Properties

We use well-established security notions for authentication and secrecy [11, 3], formalized in Tamarin [17].

Definition 5.1 (Non-injective agreement). Two agents A and B *non-injectively agree* on a protocol run defined by (transferred) data items $\vec{d}$, if whenever an agent A completed a protocol run, apparently with agent B, and believes the data items $\vec{d}$ were exchanged during this protocol run, then B was previously running the protocol, apparently with A, and with the same data items $\vec{d}$.

Definition 5.2 (Injective agreement). Two agents A and B *injectively agree* on a protocol run defined by (transferred) data items $\vec{d}$, if they non-injectively agree on the protocol run and the same protocol run will never be agreed on by the same or other agents at a different point in time.

In other words, *injective* and *non-injective agreement* are both strong authentication properties covering the authenticity of the identity of communication partners as well as the transferred information. However, *injective agreement* also requires the absence of replay attacks, which are not covered by *non-injective agreement*.

Definition 5.3 (Syntactic Secrecy). A data item s known to one or more honest agents $\mathbb{A}$ is *syntactically secret*, if there is no valid execution trace of the protocol where the attacker obtains knowledge of s without compromising a long-term secret of an agent $A \in \mathbb{A}$.

Definition 5.4 (Perfect forward secrecy). A data item s belonging to one or more honest agents $\mathbb{A}$ involved in an (authentication) process p is *forward secret* if there is no valid execution trace of the protocol where the attacker obtains knowledge of s without compromising a long-term secret of an agent $A \in \mathbb{A}$ before p finished.

We use these security properties to analyze whether the DDS Security standard meets its requirements as summarized in Section 4. We use injective-agreement as a formal notion of authentication (S_3, S_5, S_6). Please note that this notion of authenticity implies integrity (S_2). Furthermore, we use syntactic secrecy, or perfect forward secrecy, if applicable, for confidentiality requirements (S_1). As we do not analyze the *AccessControl* plugin in detail in this study, we do not consider S_4 here. Also, please note that S_7 is considered optional and is not addressed by the built-in plugins.

5.3 Analysis of the Authentication Plugin's Handshake

Our Tamarin model captures the handshake protocol as shown in Figure 3, excluding the optional parts. To avoid unnecessary complexity of our model, we only consider a single CA instead of a complete PKI. Verification of PKI certificate chains is not in the scope of this paper.

The authentication handshake is used for the initial authentication of domain participants and to establish a shared secret, which is later used to derive keys for payload encryption and authentication of DataReaders and DataWriters. Thus, apart from item S_5, the authenticity and confidentiality of the established shared secret is fundamental for enabling all other security requirements. We expect the handshake protocol to satisfy the following security lemmas:

Lemma 5.5 (Secrecy of long term keys). *The long-term keys of the participants,* sk_A *and* sk_B *are not revealed to the attacker and, thus, satisfy* secrecy *within the protocol.*

Lemma 5.6 (Perfect forward secrecy of shared secret). *The shared secret s, established between A and B during the handshake, satisfies* perfect forward secrecy.

Lemma 5.7 (Bidirectional authentication). *A successful protocol run between A and B satisfies* injective agreement *for both the initiator A and the correspondent B on the exchanged certificates, permissions, and cryptographic algorithms* C_A, C_B, *the exchanged challenges* c_A *and* c_B *and the established shared secret s.*

All lemmas have been successfully proven with Tamarin and the proofs can be reconstructed and verified with our provided model. Thus, we conclude that the handshake protocol achieves its security goals of bidirectional authentication and establishment of a shared secret with perfect forward secrecy.

5.4 Analysis of the Encryption Algorithm

We modeled the communication between the VolatileMessageSecureWriter and VolatileMessageSecureReader, which are used to establish a secure channel between two domain participants based on the shared key established during authentication. Over this channel, topic-specific keys can then be exchanged. We use a separate Tamarin model, assuming a symmetric key has been previously shared during authentication. We use the Authenticated Encryption with Associated Data (AEAD) model from [4] for AES-GCM. Please note that security-wise, the communication of topic data is equivalent.

Based on the security requirements, we would expect the following lemmas to hold:

Lemma 5.8 (Message secrecy). *When a participant* B *receives and accepts a message m, apparently coming from participant* A*, m is* syntactically secret.

Lemma 5.9 (Message authentication and replay protection). *When a participant* B *receives and accepts a message m, apparently coming from participant* A*,* A *and* B injectively agree *on m.*

We successfully verified the first lemma with Tamarin, showing that the encryption algorithm succeeds in protecting the secrecy of transferred data. However, the verification of the second lemma fails with the attacker being able to replay old messages. While the initialization vector is constructed as a counter, this is only used to guarantee the uniqueness of the IV. The receiver only uses the IV from the received message without verifying its freshness. Thus, the protocol is vulnerable to replay attacks. Nonetheless, the attacker is not able to forge arbitrary messages, i.e., the following lemma holds:

Lemma 5.10 (Message authentication without replay protection). *When a participant* B *receives and accepts a message m, apparently coming from participant* A*,* A *and* B non-injectively agree *on m.*

Thus, this replay attack could be detected if the replay is evident from the message content, i.e., if a counter is included and messages with repeated counters are considered duplicates and dropped. As DDS messages contain a sequence number, we determine the feasibility of such a replay attack by developing a proof of concept implementation for a common DDS implementation in the following section.

6 Practical Evaluation of the Replay Attack

As stated in section 5.4, the DDS:Crypto:AES-GCM-GMAC encryption on its own is prone to a replay attack.

The root of this issue stems from the way the encryption algorithm handles the decryption of ciphertexts ([14, Section 9.5.3.3.5]). The details of the decryption algorithm are stated in the *Decryption* paragraph in section 4.3.2. Since the IV is taken from the CryptoHeader without further checks, there is no check whether the session ID and the IV suffix are reused. From this session ID, the session key is obtained. Thus, replaying the same message leads to the same decryption key and, therefore, to a valid decryption of the same message again. To prevent the attack, the receiver could maintain a state that verifies that the combination $\mathsf{IV} = (\mathsf{ID}^{\text{session}}, \mathsf{IV}^s)$ is not reused. However, this solution would involve additional development work and result in decreased performance. The replay attack can also be prevented by existing built-in functionalities within the AccessControl plugin, which are discussed in section 6.2. Nevertheless, the existence and mitigation of this replay attack is not mentioned in the specification, and we propose adding a note to the specification.

6.1 Proof of Concept (POC)

We implement a POC for the replay attack on the encryption algorithm. For it, we use the open source implementation *OpenDDS* [16]. For reproducibility, we built a Docker container with a modified version of the Messenger Example.[4] The source code of our POC can be found on GitHub.[5]

In our scenario, there are two domain participants, one creates a publisher and a data writer and the other one creates a subscriber and a data reader on the topic *Movie Discussion List*. The publisher sends seven messages to the subscriber with a delay of 2 seconds each. After the third message is sent, the subscriber sleeps for 10 seconds. During this time, the last message sent is replayed.

We use the python library scapy[6] for package capturing and manipulation. In our case, RTPS packets are sent via UDP/IP. RTPS packets can be detected by the magic bytes *RTPS* (0x52545053). Moreover, messages are send to the multicast address 239.255.0.2[7] and can be filtered by this IP address. We then replay the last captured RTPS packet p. A comparison between the sent RTPS packets conducted that in one protocol run they only differ in a timestamp (part of the SubMessage INFO_TS), a writer sequence number and the serialized data field. Manual tests show that taking p, incrementing its sequence number, and then resending it via UDP via the kernel network stack was sufficient for the attack to work. We chose to use the kernel network stack for sending the

[4] https://opendds.readthedocs.io/en/latest-release/devguide/quickstart/docker.html
[5] https://github.com/fzi-forschungszentrum-informatik/DDS-SECURITY-Replay-Attack-POC
[6] https://github.com/secdev/scapy
[7] https://github.com/OpenDDS/OpenDDS/blob/DDS-3.29.1/dds/DCPS/transport/rtps_udp/RtpsUdpInst.cpp#L338

Table 2: Maximum protection settings for the replay attack

Property	Value
allow_unauthenticated_participants	false
enable_join_access_control	true
discovery_protection_kind	ENCRYPT
liveliness_protection_kind	ENCRYPT
rtps_protection_kind	NONE
enable_discovery_protection	true
enable_liveliness_protection	true
enable_read_access_control	true
enable_write_access_control	true
metadata_protection_kind	NONE
data_protection_kind	ENCRYPT

packet because it automatically handles the checksum computation. The destination IP and port are also adopted from p.

The library scapy was able to parse unprotected RTPS packets, but not secured RTPS packets like in Figure 4. We therefore extended the scapy library to parse RTPS packets with encrypted *SerializedPayload*.

Note that it is mandatory to increase the writer sequence number. If the subscriber receives several messages with the same sequence number, all except the first one are ignored. This can lead to a DoS attack if an attacker repeatedly replays a message with increasing sequence number. An example of this can also be found in our GitHub repository.

We offer different configurations for the governance file by different docker compose files. The maximum protection that can be set, such that the attack in our case still succeeds can be found in Table 2. Consider that *data_protection_kind* is set to ENCRYPT, that is the actual payload in encrypted. Please note that at the time of writing, OpenDDS does not support advanced protection kinds. Since we do not alter discovery or liveliness messages and just replay the *SerializedPayload*, the values *discovery_protection_kind* and *liveliness_protection_kind* could also be set to an advanced protection kind and the replay attack should still be successful.

6.2 Mitigation

As described in the last section 6.1 and in Table 2, not all protection kinds in the governance file can be set for the replay attack to succeed. To prevent the replay attack, the writer sequence number has to be protected. This can be achieved by either encrypting this sequence number or protecting it with a MAC. To do so, at least one of the following settings in the governance file must be set to SIGN or ENCRYPT or one of the advanced protection kinds from Section 4.2.3:

rtps_protection_kind: If this value is set, then the whole RTPS message is protected. Therefore, the sequence number is also protected. This corresponds to the packet 'Secured RTPS message' in Figure 4.

metadata_protection_kind: The sequence number belongs to the metadata. If the metadata are not to be altered, then the sequence number is safe from modification.

7 DDS Security Version 1.2

In this section, we discuss relevant changes in DDS Security 1.2 and their impact on security. Please note that the authentication handshake protocol and encryption algorithm analyzed in Section 5 remain unaltered. The following major security-relevant changes are introduced:

New cryptographic algorithms: DDS Security 1.2 allows more cryptographic algorithms, including SHA384 for the Digital Signature Algorithm (DSA) signatures and the P-384 elliptic curve for Diffie-Hellman and DSA.

Protected discovery phase: The initial discovery of domain participants, prior to authentication, can be protected using a Pre-shared Key (PSK) between all domain participants.

New builtin PSK security plugins: New builtin Authentication, AccessControl, and Cryptography plugins are defined that only rely on the PSK and do not require digital certificates.

Key revision mechanism: Previously authorized access can be revoked by replacing the shared cryptographic keys.

In the following, we describe and analyze the protection of the discovery phase, as well as briefly discuss the new built-in PSK plugins based on security-critical use cases such as automotive. An analysis of the key revision mechanisms will be left for future work.

7.1 Protected Discovery Phase

In DDS Security 1.1, the initial discovery of domain participants, which has to be performed before authentication, remained unprotected. This enabled DoS attacks that relied on the high resource consumption for computing a DH key pair during the initialization phase ([18]) and led to information leakage ([18, 19]).

To address this issue, DDS Security 1.2 introduced PSKs to secure the communication between domain participants before the authentication was performed. This requires a pre-shared key between all domain participants. PSKs can be a passphrase of up to 512 printable ASCII characters. They can be enabled via the additional setting *rtps_psk_protection_kind* in the domain governance document. It can be set to one of the basic protection kinds NONE, SIGN, and ENCRYPT and will always affect the complete RTPS message (see Section 4.2.3).

For PSK protection, the Cryptographic plugin is used (see Section 4.3.2). However, the master sender key mk and master salt msalt used by the algorithm are not defined based on the exchanged challenges and secret from the not yet completed authentication process. Instead, they are computed based on the pre-shared key psk as follows:

$$\mathsf{mk} := \mathsf{hmac}^{\mathsf{sha256}}_{\mathsf{key}}(\text{"master sender key derivation"} \| 0x01)$$

where $\mathsf{key} = \mathsf{hmac}^{\mathsf{sha256}}_{\mathsf{key}_2}(\mathsf{psk})$ and

$$\mathsf{key}_2 = \text{"PSK-SKEY"} \| \mathit{SenderKeyId} \| \text{"RTPS"} \| \mathit{ProtocolVersion} \| \mathit{VendorId} \| \mathit{GuidPrefix}$$

Similar, msalt is computed as:

$$\mathsf{msalt} := \mathsf{hmac}^{\mathsf{sha256}}_{\mathsf{key}}(\text{"master salt derivation"} \| 0x01)$$

with key defined as above with the exception that the fixed string "PSK-SKEY" in key of the inner HMAC function is replaced with "PSK-SALT". No receiver-specific keys are used.

We provide an additional Tamarin model for this configuration of the Cryptographic plugin. Our analysis shows that the resulting encryption protocol does not guarantee *non-injective* or *injective agreement*. First, replay attacks are possible as shown in Section 5.4. Second, the encryption protocol relies on receiver-specific keys for origin authentication, which are not used here. Thus, the specific origin of a message is not authenticated. However, adversaries cannot forge arbitrary messages.

While unique identifiers contained within the message payload might prevent redirection attacks, the reliance on a single PSK means that this protection mechanism can be circumvented by compromising a single domain participant.

7.2 Builtin PSK Plugins

The new builtin plugins *DDS:Auth:PSK*, *DDS:Access:PSK* and *DDS:Crypto:PSK* enable the usage of PSK only, without further certificate-based authentication. Hereby, DDS:Auth:PSK and DDS:Access:PSK only operate as dummy plugins, authenticating and authorizing all participants with knowledge of the PSK. DDS:Crypto:PSK works as described above, using the PSK to secure all communication, including the communication subsequent to authentication.

As shown in the section above, this may enable an attacker to redirect messages between participants. In addition, compromising a single participant or the PSK completely breaks the entire system's security. Therefore, these plugins are not suited for security-critical scenarios including automotive. Otherwise, similar attacks as the de-association attack on SOME/IP [20] would be possible.

8 Conclusion

In general, DDS Security provides a well-thought-out protection mechanism based on well-established cryptographic algorithms to secure DDS Domains. Our analysis showed that the authentication handshake protocol satisfies strong authentication and secrecy properties in the symbolic model. However, our analysis showed that the prevention of replay attacks, which are among the main threats considered in the DDS Security Specification, is not guaranteed with the used encryption/authentication algorithm. In practice, replay attacks might be prevented by the packet's sequence number, which, however, is not protected in all possible configurations of DDS Security. Moreover, this side-effect is not documented within the standard. Additional weaknesses in DDS Security 1.1, especially regarding DoS attacks, have been shown by previous work.

DDS Security 1.2 tries to mitigate the DoS vulnerability during the initial participant discovery. Even though we could not analyze the key revision mechanism introduced in DDS Security 1.2 in this work, we consider the possibility of revising keys a huge improvement. The possibility to revoke access for compromised participants or certificates and to dynamically revoke permissions is very important for critical systems.

Future work includes developing a POC implementation of our discovered replay attack for DDS Security 1.2. Although this attack is theoretically feasible, OpenDDS does not implement DDS Security 1.2 at the time of writing. Moreover, it could be interesting to analyze if the replay attack could be used to circumvent the PSK-based DoS protection. Furthermore, a formal analysis of the security of the new key revision mechanism should be performed.

Acknowledgments

This work was partly funded by the Topic Engineering Secure Systems of the Helmholtz Association (HGF) and supported by KASTEL Security Research Labs, Karlsruhe. Co-funded by the European Union under project European Digital Innovation Hub applied Artificial Intelligence and Cybersecurity EDIH-AICS, (Grant no. 101083994).

We made limited use of an AI-based coding assistant (Codeium) during development of the formal models and proof-of-concept implementation.

References

[1] David Basin, Cas Cremers, Jannik Dreier, Simon Meier, Ralf Sasse, and Benidikt Schmidt. 2024. *Tamarin Prover*. https://tamarin-prover.com/.

[2] Bruno Blanchet and Vincent Cheval. 2024. *ProVerif: Cryptographic protocol verifier in the formal model*. https://bblanche.gitlabpages.inria.fr/proverif/.

[3] Véronique Cortier, Michaël Rusinowitch, and Eugen Zălinescu. 2006. Relating two standard notions of secrecy. In *Computer Science Logic*. Zoltán Ésik, editor. Springer Berlin Heidelberg, 303–318. ISBN: 978-3-540-45459-5.

[4] Cas Cremers, Alexander Dax, Charlie Jacomme, and Mang Zhao. 2023. Automated Analysis of Protocols that use Authenticated Encryption: How Subtle AEAD Differences can impact Protocol Security. Cryptology ePrint Archive, Paper 2023/1246. (2023). https://eprint.iacr.org/2023/1246.

[5] Danny Dolev and Andrew Yao. 1983. On the security of public key protocols. *IEEE Transactions on information theory*, 29, 2, 198–208.

[6] Maxim Friesen, Gajasri Karthikeyan, Stefan Heiss, Lukasz Wisniewski, and Henning Trsek. 2020. A comparative evaluation of security mechanisms in DDS, TLS and DTLS. In *Kommunikation und Bildverarbeitung in der Automation: Ausgewählte Beiträge der Jahreskolloquien KommA und BVAu 2018*. Springer Berlin Heidelberg, 201–216.

[7] Zhi Yong He and Yue Liang. 2015. Study on the DDS Network Information Security Technology. *Applied Mechanics and Materials*, 738, 1213–1216.

[8] M. Iorio, A. Buttiglieri, M. Reineri, F. Risso, R. Sisto, and F. VALENZA. 2020. Protecting In-Vehicle Services : Security-Enabled SOME/IP Middleware. *IEEE Vehicular Technology Magazine*.

[9] Hwimin Kim, Dae-Kyoo Kim, and Alaa Alaerjan. 2022. ABAC-Based Security Model for DDS. *IEEE Transactions on Dependable and Secure Computing*, 19, 5, 3113–3124. DOI: 10.1109/TDSC.2021.3085475.

[10] Timm Lauser, Daniel Zelle, and Christoph Krauß. 2020. Security analysis of automotive protocols. In *Computer Science in Cars Symposium* (CSCS '20) Article 11. Association for Computing Machinery, Feldkirchen, Germany, 12 pages. DOI: 10.1145/3385958.3430482.

[11] Gavin Lowe. 1997. A hierarchy of authentication specifications. In *Proceedings 10th Computer Security Foundations Workshop*. IEEE, 31–43.

[12] Federico Maggi, Rainer Vosseler, Mars Cheng, Patrick Kuo, Chizuru Toyama, T Yen, and E Boasson V Vilches. 2022. A Security Analysis of the Data Distribution Service (DDS) Protocol. *Trend Micro Research, Inc., Japan*, 15–20.

[13] Object Management Group. 2022. DDS Interoperability Wire Protocol. Version 2.5. (Apr. 2022). https://www.omg.org/spec/DDSI-RTPS.

[14] Object Management Group. 2018. DDS Security. Version 1.1. (2018). https://www.omg.org/spec/DDS-SECURITY/1.1/PDF.

[15] Object Management Group. 2024. DDS Security. Version 1.2. (June 2024). https://www.omg.org/spec/DDS-SECURITY/1.2/PDF.

[16] OpenDDS Foundation. 2024. *OpenDDS*. https://opendds.org/.

[17] The Tamarin Team. 2024. *Tamarin-Prover Manual - Security Protocol Analysis in the Symbolic Model*. https://tamarin-prover.com/manual/develop/tex/tamarin-manual.pdf.

[18] Binghan Wang, Hui Li, and Jingjing Guan. 2024. A Formal Analysis of Data Distribution Service Security. In *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security* (ASIA CCS '24). Association for Computing Machinery, Singapore, Singapore, 716–727. ISBN: 9798400704826. DOI: 10.1145/3634737.3656288.

[19] Thomas White, Michael N. Johnstone, and Matthew Peacock. 2017. An investigation into some security issues in the DDS messaging protocol. In *The Proceedings of 15th Australian Information Security Management Conference*. C. Valli, editor. Edith Cowan University, Perth, Western Australia, (Dec. 2017), 132–139.

[20] Daniel Zelle, Timm Lauser, Dustin Kern, and Christoph Krauß. 2021. Analyzing and Securing SOME/IP Automotive Services with Formal and Practical Methods. In *The 16th International Conference on Availability, Reliability and Security* (ARES 2021) Article 8. Association for Computing Machinery, Vienna, Austria, 20 pages. DOI: 10.1145/3465481.3465748.