Hamed Soroush, Philip M. Irey IV, Gerardo Pardo-Castellote, Steve Canup

Next-generation Cybersecurity for Advanced Real-time Distributed Systems

March 2015

ABSTRACT

The Data Distribution Service (DDS) is a common information technology standard mandated for use by the Department of Defense (DoD) and heavily used by industry. DDS usage is particularly pervasive in Navy surface combatants, especially because of its reliability, robustness, and provisioning of quality of service. Until recently, DDS has offered limited support for security. For instance, existing DDS systems support isolating DDS applications into a security enclave running at "system high." Inside the "protected domain," applications are authorized to publish and subscribe to any data in the DDS Global Data Space. However, applications are not authenticated unless done at the application layer, and metadata is sent unencrypted and unprotected against tampering. Typical DDS infrastructure provides no guarantees (other than those provided by the physical protection of the system) related to information confidentiality, pedigree, or integrity.

With naval cybersecurity requirements being pushed out into current and future programs of record, steps are being taken to provide a more secure environment that is less susceptible to cyber-attacks. In order to help meet these requirements in distributed systems, the U.S. Navy has recently funded the prototype development of security extensions to DDS. Documentation from this effort contributed significantly to the development of the Object Management Group (OMG) DDS Security Specification standard. This paper will describe the design of the security extensions to DDS that provide the necessary support for authentication, authorization, confidentiality, integrity, and auditing. In addition, it discusses the relevance of the extensions to the Navy's cybersecurity strategy moving forward.

Introduction

In the 1990s, the architecture and technologies used by surface Navy combat systems were limiting performance and scalability. Architectures and technologies that could overcome these limitations were evaluated, resulting in recommendations to use distributed network architectures and standards-based, commercial off-the-shelf (COTS) technologies [1]. DDS was one of the key standards-based COTS technologies recommended for publish-subscribe data distribution. Key DDS features that drove this recommendation include support for the development of scalable distributed computing systems, support for loose coupling between components, the capability to provide high-performance data transfer, and capabilities such as highly flexible quality of service (QoS) control needed to support a wide range of real-time combat system requirements. DDS was widely adopted by surface Navy combat systems for publish-subscribe data transfer.

The introduction of publish-subscribe to combat systems completely changed the way interfaces were developed. Rather than defining strictly point-to-point interface design specifications to describe data exchanges that occur between two endpoints, interfaces were now described as topics, which specified the type of data to be sent, and the QoS with which to distribute that data. While the technology significantly reduced the complexity of making interface changes and increased flexibility, it presented challenges for addressing security requirements that would be desired in future combat system iterations.

DoD cybersecurity requirements for Navy afloat platforms, along with all DoD systems, originate from DoD Instruction 8500.1, "Cybersecurity," and eventually trace down to specific platform requirements. For example, a series of data-in-transit (DIT) controls are defined to protect data as it flows through networks. Navy afloat platforms employ a defense-in-depth strategy applying these cybersecurity requirements at a variety of critical points on a platform.

In 2010, recognizing the need to address cybersecurity requirements more broadly in its combat systems, the Navy initiated a Small Business Innovative Research (SBIR) topic [2] to "develop the capability to authenticate, authorize, encrypt, key manage and audit publishers and subscribers in a real-time deadline scheduled pub/sub software environment on a per middleware message basis." Through the execution of this SBIR, DDS security extensions to support these cybersecurity capabilities were prototyped and then documented in the OMG DDS Security Specification standard. Today, multiple vendors are adding these cybersecurity capabilities to their DDS implementations.

The expansion of the DDS standard to include mechanisms for authentication, authorization and access control, confidentiality, integrity, and auditing provides capabilities that can be used to address a variety of cybersecurity requirements of interest to the Navy. For example, DDS extensions for authentication, authorization and access control, confidentiality, and integrity can directly support DIT requirements. The provided auditing capabilities can be used by SIEM [System Incident and Event Monitoring] mechanisms to record security events. DDS Security improves the fidelity of event reconstruction by adding a degree of certainty: identifying the participants in the event and their actions.

The DDS standard defines a common set of service plugin interfaces (SPIs) to achieve interoperability and allows the implementation of the plug-ins to be customized depending on end user security requirements. The flexibility provided by this pluggable architecture will allow the Navy to integrate DDS Security capabilities more easily into its information systems architecture.

To support typical use cases, DDS Security also comes with a standardized built-in implementation of the security plugins.

Navy Cybersecurity Strategy

The cybersecurity strategy for Navy afloat systems is similar to any other system of systems strategy. A Navy afloat platform is divided into multiple functional domains, as shown in Figure 1. Specific cybersecurity requirements are defined for each of those domains and the systems within those domains. A subset of those requirements will address communication within and between those domains. Each domain and system in the domain will have multiple DIT requirements driven by various functional and cybersecurity requirements.

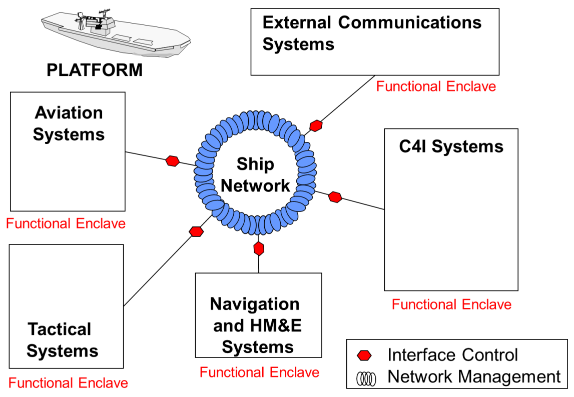

Figure 1: Notional Shipboard Enclaves

For domains where DDS is used, plans must be developed to introduce the new security mechanisms provided by the DDS Security standard. Prior to this, analysis must be done to determine which of the mechanisms should be used and to what level they should be used (i.e., choice of algorithms and key sizes). The analysis is critical since many of the DDS-based data flows have strict timing requirements and any unnecessary overhead (i.e., encrypting an entire message where only a portion really needs to be encrypted) could impact the ability to deliver the data in the required time interval. DDS Security was designed to be highly flexible, protecting exactly what needs protection and minimizing overhead.

Data Distribution Service (DDS)

Today, DDS is widely used by numerous surface Navy combat systems. In these systems, topics are defined to distribute specified messages. For example, systems using Product Line Architecture (PLA) components publish data conforming to the PLA Common Data Model on DDS topics. Subscribers to those topics then consume the data.

To accommodate the large number of topics and scalability needed by some systems, topics can be partitioned into separate DDS domains.

Background on Data Distribution Service

The OMG DDS is a communication application program interface (API) and interoperability standard. DDS provides a data-centric publish-subscribe model for a middleware that integrates loosely coupled real-time distributed systems. A key DDS feature is that it is data-centric in the sense it separates state management and data distribution from application logic and supports discoverable data models. This exposes the data model to the communication middleware, enabling the DDS middleware to reason about and optimize the performance of data movement in the system. In order to customize run-time behavior and achieve a desired performance profile, DDS allows publishing and subscribing entities to express several QoS parameters, such as data durability, reliability, delivery deadline, ownership, liveliness, and resource limits. Offered versus requested QoS requirements of the participating entities are matched before any communication can proceed. In case of a mismatch, corresponding applications are notified by the middleware. The QoS capabilities provided by DDS are used to tailor the distribution of data so that critical system requirements can be met.

DDS Terminology

A *domain* is a concept used to bind individual applications together for communication. To communicate with each other, *DataWriters* and *DataReaders* must have the same *Topic* of the same data type and be members of the same *domain.* Applications in one domain cannot subscribe to data published in a different domain.

DomainParticipant objects enable an application to exchange messages within domains. *DomainParticipants* are used to create and use *Topics, Publishers, DataWriters, Subscribers,* and *DataReaders* in the corresponding *domain*. An application uses a *DataWriter* to publish data into a domain. A *DataReader* is the point through which a subscribing application accesses data received over the network. A *Publisher* is used to group individual *DataWriters,* and a *Subscriber* is used to group *DataReaders.* Default QoS behavior can be specified for a *Publisher* and have it apply to all *DataWriters* in the *Publisher's* group. A similar relationship holds between a subscriber and its group of *DataReaders. Topics* provide the basic connection points between *DataWriters* and *DataReaders.* To communicate, the *Topic* of a *DataWriter* on one node must match the *Topic* of a *DataReader* on any other node.

There are several approaches for defining and using data types in a DDS-based solution, including using built-in types, defining types at compile time, and programmatically defining dynamic types. In general, DDS types are specified in the standard Interface Definition Language before being mapped to a desired target language such as C++ or Java. Once defined, middleware automatically discovers data objects and manages the QoS.

DDS objects *(DomainParticipants, DataWriters,* and *DataReaders)* that may reside on different nodes find out about each other through a mechanism called DDS *Discovery.* This mechanism is used to detect when participants enter or leave the DDS domain, and allows them to learn about each other's identifiers, transport locators, and requested or offered QoS.

The unique values of data passed over DDS are called *samples.* A sample is a combination of a topic (distinguished by a topic name), an *instance* (distinguished by a key value), and the actual user data of a certain type.

DDS Security

The DDS Security extension is comprised of a security model, a pluggable architecture and associated SPIs, and specification of built-in implementations of these SPIs. In this section, the authors provide a general description of each of these high-level components.

DDS Security Model

In general, a security model defines the security principals, associated threats, the objects being secured, and the operations on the objects to be restricted according to a security policy. In the DDS Security model, what is being secured is a specific DDS domain and, in the domain, the ability to read or write information (e.g., specific topic or even data-object instances in the topic) in the domain.

To provide such secure access, domain participants must first be authenticated so their identity can be established. Once authentication has been obtained, access control policies are enforced that determine whether the participant is allowed to perform specific actions. Examples of such actions are joining a DDS domain, defining a new topic, and reading or writing a specific topic.

Threat Model

Specific threats that DDS Security is designed to protect against are as follows.

Unauthorized subscription: An unauthorized eavesdropper who is connected to the same network should not be able to observe the data sent over DDS by legitimate participants, for example, by tapping into the communication channel or simply subscribing to a multicast address used by authorized subscribers.

DDS Security protects against unauthorized subscription attacks by having senders securely share a secret key with authenticated and authorized receivers and using the key to encrypt what they write.

Unauthorized publication: Unauthorized participants should not be able to inject packets into the DDS data space and have them processed as valid packets by legitimate participants.

DDS Security protects against unauthorized publication attacks by having senders authenticate messages using digital signatures or hash-based message authentication codes (MACs). The required keys for computing MACs are distributed securely.

Tampering and Replay: Authorized yet compromised participants may use their knowledge of shared secret keys for malicious behavior. For example, a malicious participant, M, who is authorized to subscribe to data on Topic T but not authorized to publish on the topic, may use information gained by subscribing to the data to attempt to publish tampered information in the network and to convince other subscribers of the legitimacy of the tampered information.

DDS Security provides mechanisms to enforce the use of digital signatures by all participants to protect against this attack. In cases where the performance penalty of using digital signatures is deemed too high, the sender keeps different pair-wise keyed-hash message authentication code (HMAC) keys with each receiver, and DDS Security validates them before data is delivered to the receiver.

DDS Security Plugins Interfaces

DDS Security presents a pluggable architecture with plugins for authentication, access control, cryptographic operations, logging, and data tagging. The interfaces of these plugins are part of the standard, allowing for different implementations of security protocols and algorithms in each plugin. The pluggable architecture of DDS Security and how it fits in the overall ecosystem of DDS applications is depicted in Figure 2.

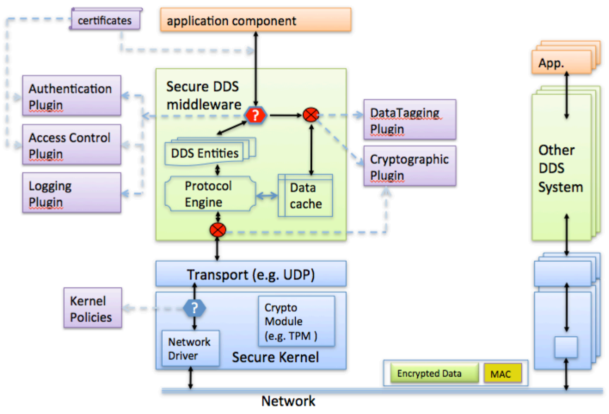

Figure 2: Architectural View of DDS Security

Without going into the details of the standard interface for each plugin, their general characteristics are described. Details of the SPIs can be found in [3].

Authentication Service Plugin

The authentication plugin SPI defines the types and operations necessary to authenticate DDS domain participants. With DDS Security enabled, every DDS participant will be required to authenticate prior to joining to a DDS domain. Furthermore, DDS Security enhances the discovery mechanism that registers participants with the DDS middleware by making authentication a requirement. For protected DDS domains, a participant enabling the authentication plugin can only communicate with participants who have the authentication plugin enabled.

Once authenticated to the middleware, two participants that have learned about each other (but not yet trust each other) exchange messages to mutually authenticate through a challenge-response protocol. SPI is designed to oblige multiple implementations with varying numbers of message exchanges; therefore, a variety of such protocols could be implemented in the plugin. Often a shared secret is also derived from a successful authentication message exchange. The shared secret can be used to exchange cryptographic material for encryption and message authentication. Authentication SPI supports data structures for shared secret establishment and use.

Access Control Service Plugin

Once a domain participant is authenticated, its permissions are validated and enforced. The Access control plugin API defines the types and operations necessary to support an access control mechanism for DDS domain participants.

Traditionally, access rights are described by using matrices with rows representing subjects (users) and columns representing objects (resources). A cell would therefore define the access rights a given subject has over an object. Such matrices are implemented in two ways. The first approach is row-centric, keeping a set of *capabilities* stored with each subject. The second approach is column-centric, keeping access control lists per object. DDS Security supports both of these approaches.

In the case of DDS Security, the permissions associated with a domain participant include the permission to join to a certain domain, create a topic, publish through its *DataWriters* into certain topics, and subscribe via its *DataReaders* to certain topics.

Cryptographic Service Plugin

The cryptographic plugin defines the types and operations necessary for encryption, digest creation, generation of message authentication codes, and the key exchange for *DomainParticipants, DataWriters,* and *DataReaders.*

DDS users may have specific cryptographic libraries they use for encryption as well as specific requirements regarding the algorithms for digests, message authentication, and signing. Applications may also require having only some of those functions performed, or performed only for certain DDS *Topics* and not for others. This plugin API is general enough to use in such different deployment scenarios.

Logging Service Plugin

The logging service plugin API defines the types and operations necessary to log security events for DDS *DomainParticipants.* It provides the capability to log all security events, including expected behavior and all security violations or errors. These security logs can be used for audits. Other security plugins use this plugin to log events.

The logging API has two options for collecting log data. The first is to log all events to a local file for collection and storage. The second is to distribute log events securely over the DDS.

Data Tagging Service Plugin

Data tagging adds a security label or tag to data. An example use case for such tags is specifying data classification levels. The tags could be used in making access control decisions (i.e., in conjunction with the access control plugin), message prioritization, or consumption by applications other than the middleware.

Four different possible approaches were identified for data tagging in DDS Security. These approaches include: data writer tagging, in which data received from a certain DataWriter has the tag of the DataWriter; data instance tagging, in which each instance of the data has a tag; individual sample tagging, in which every DDS sample has its own tag attached; and per-field sample tagging, in which each field in the sample has its associated tag.

DDS Security supports *DataWriter* tagging, as this was considered the best choice among the four. This solution does not require the tag to be added to each individual sample and is more aligned with the general approach in DDS wherein the metadata for all *DataWriter* samples is the same. It also leads to the highest performance, as the tags only need to be exchanged once when the *DataWriter* is discovered and not sent with each sample. This approach is directly used for typical use cases where each application or *DomainParticipant* writes data on a Topic with a common set of tags (i.e., all at the same specified security level). For use cases where an application creates data at different classifications, the application can create multiple *DataWriters* with different tags. The other identified approaches have a higher overhead and more complex management of tags.

DDS Security does not currently provide a built-in plugin for data tagging SPI.

Built-in Plugins

DDS Security defines the behavior and implementation of at least one built-in plugin for each kind of the discussed SPIs. The built-in plugins provide out-of-the-box interoperability between implementations of the specification. In this section, the authors briefly discuss the requirements that drove the design choices as well as the structure of each of the built-in plugins.

Requirements and Priorities

The major functional requirements considered when designing built-in plugins include: authentication of applications joining a domain; access control publishers and subscribers at the domain and topic levels; message integrity and authentication; encryption of data samples using different keys for different topics; and the capability of securely sending data over multicast.

The nonfunctional requirements include high performance and scalability, robustness and availability, fitness to DDS support for data-centricity, and ease of use.

Performance and Scalability

DDS is commonly deployed in systems that demand high performance and need to scale to large numbers of computers, processes, topics, and data-objects belonging to each topic. Therefore, plugin operations such as cryptographic or policy enforcement operations should have minimal impact on the performance and scalability of the system. These considerations translate into practical design decisions made for the built-in plugins as discussed below.

Limited Use of Asymmetric Cryptography: Because of its high computation costs, the use of asymmetric key cryptography should be limited to discovery, authentication, and shared secret establishment phase, and not within the critical path of data distribution. Symmetric ciphers should be used when encrypting application data.

Support for Secure Multicast: Since multicasting is crucial to achieve high performance in many DDS deployments, built-in plugins should support it even for ciphered data.

Topic-level Security: The use of ciphers, HMACs, or digital signatures shall be selectable on a per stream (Topic) basis. Furthermore, the built-in plugins should support authentication-only modes, providing data integrity of data even when it is not encrypted.

Robustness and Availability

DDS has originally been designed to meet high uptime requirements of mission-critical systems, having robustness and reliability as main features. DDS communication model and protocols are defined and commonly implemented in a peer-to-peer fashion without relying on any centralized services; thus avoiding single points of failure. It is required that the built-in security plugins do not negate these properties of the middleware. This means that centralized policy decision points or services should be avoided in the plugin implementation. Furthermore, each domain participant should stay self-contained, having all the necessary components they need to operate securely in the presence of system partitions. Multi-party key agreement protocols should be avoided in the built-in plugins as the disruption of one party easily disrupts them. Last but not the least, the impact of a possible component compromise should be kept to a minimum, preferably to that component itself. This requirement translates into security token and key compartmentalization of as much as possible. Having a system-wide key for the whole DDS domain, for example, would negate this requirement and should be avoided. The keys used for encrypting data written to a topic are picked by the *DataWriter* and securely distributed to the discovered readers through the use of pairwise exchange keys derived from shared secrets established with each *DataReader.*

Fitness to DDS Data-centric Model

Data-centricity is among the main features that attract application developers to DDS. DDS developers architect their systems by defining domains to which DDS applications join and the topics that they need to read and write. Therefore, the access control mechanism provided by the built-in plugins should support this level of granularity.

It is of course possible to provide access control at a finer granularity, including the keyed instances that the applications read or write, content filters, and QoS policies. However, this level of access control potentially conflicts with the goal of ease of configurability and maintainability and was considered of lower priority for the built-in plugins.

Another important requirement also rises from the semantics of DDS communication, according to which individual samples can be consumed independent of each other. Depending on the QoS policy settings, samples written by a single *DataWriter* may be received and processed out of order relative to the order sent; may be received with intermediate gaps resulting from best-effort communication (if selected); or may be filtered by content, time, or history. Consequently, cryptographic transformations of samples (e.g., decryption, signature verification) should not require reconstruction of a specific context using previous samples.

Leveraging Existing Security Technologies

Our approach in the design of built-in plugins has been to leverage existing technologies and tools for security. The benefit of reuse is twofold: First and foremost, use of already proven approaches provides an overall better security solution. Second, it reduces the barrier of entry for implementers. Consequently, built-in plugin use established cryptographic algorithms for creating ciphers, signatures, and digests as well as standard approaches to provision public key infrastructures. Existing approaches for key management and secure multicast have been leveraged as much as possible; yet they have been adapted for use in DDS's data-centric model when needed.

Ease of Use

One of the major requirements in designing built-in plugins has been achieving a balance between rich functionality and ease of use for most common deployment scenarios. The importance of this requirement becomes clear when one notes the broad categories of the anticipated adopters of DDS Security. Developers of specialized applications would likely develop their own security plugins to meet special requirements or to integrate with their exiting security infrastructure. On the other hand, users who want to secure their systems but do not have complex security requirements are more likely to use the built-in plugins out of the box. As a result, they care a lot about the ease of plugin configurability and maintainability. Hard-to-configure security solutions often lead to incorrect configurations that put the entire system at risk.

Built-in Authentication Plugin

The built-in authentication plugin is implemented using a trusted certificate authority (CA). It performs mutual authentication among discovered participants using the Digital Signature Algorithm (DSA) [4] and establishes a shared secret using the Diffie-Hellman (D-H) Key Agreement [5].

The CA could be preexisting or created for deploying applications on a DDS domain. Prior to a *domain participant* being enabled, the built-in authentication plugin associated with it must be configured with: the X.509 certificate that defines the shared CA and contains the 2048-bit RSA public key; the 2048-bit RSA private key of the domain participant; and an X.509 certificate that chains up to the shared CA, which binds the 2048-bit RSA key of the *DomainParticipant* to the subject name for the *DomainParticipant* and any intermediate CA certificates required to build the chain. The configuration API for the built-in authentication plug-in is left outside of the specification, accommodating different security concerns without breaking interoperability.

Once discovered and authenticated to the middleware, domain participants are mutually authenticated to each other by running a point-to-point public key-based challenge-response handshaking protocol [3]. Upon successfully completing the handshake process, the participants learn about each other's identities and granted access permissions as part of the secure discovery process. They would also establish a shared secret used to derive symmetric keys that enable message exchange security.

Built-in Access Control Plugin

The built-in access control plugin implements the access control plugin API using a permissions document signed by a shared certificate authority (permissions CA). The built-in access control plugin is configured with three documents: the permissions CA certificate; a domain governance document signed by the permissions CA; and the domain participant's permissions signed by the permissions CA.

The permissions CA certificate is a self-signed x.509 certificate containing the CA's public key used to sign the domain governance and participants' permissions document.

The domain governance document is written in XML [eXtensible Markup Language], specifying which DDS domains shall be protected and the details of the protection. The domain governance document is signed by the permissions CA, and configures the following security aspects of the DDS domain: whether the discovery information should be protected and the kind of protection (MAC or ENCRYPT_THEN_MAC); whether liveliness messages should be protected; whether a discovered participant that cannot authenticate or fails authentication should be allowed to join the domain and see any data configured as unprotected; whether discovery data on a specific topic should be protected; whether metadata (e.g., sequence numbers, heartbeats) should be protected and how; whether the payload should be protected and how; and whether read/write access to the topics should be open to all or restricted to the participants with proper permissions.

The XML permissions document contains the permissions of the domain participant. The permissions document binds the permissions to the *DomainParticipant's* distinguishable name as defined in the built-in authentication plug-in.

Built-in Cryptographic Plugin

The built-in cryptographic plugin provides data encryption services using Advanced Encryption Standard (AES) in counter (CTR) mode. It supports two AES key sizes: 128 bits and 256 bits. It also provides HMAC services with two different hashing functions: SHA256 and SHA1.

The approach followed is conceptually similar to that used for a Secure Real-time Transport Protocol (SRTP) [6]. However, it has been enhanced to support additional scenarios, such as the presence of services like a DDS persistence service or a data relay service, which are present in DDS real-time publish-subscribe (RTPS) systems and not supported by SRTP.

AES in CTR mode is the algorithm used for data confidentiality. While AES is a block cipher, the use of counter mode effectively turns it into a stream cipher. The algorithm generates key-stream blocks that are XORed with the plaintext blocks to get the ciphertext. Since the XOR operation is symmetric, the decryption operation is exactly the same.

Counter mode use allows decryption of blocks in arbitrary order. This is paramount for the DDS because a *DataReader* may not receive all the samples written by a matched *DataWriter* as a result of QoS-based filtering.

Built-in Logging Plugin

The built-in logging plugin publishes the logging information to a specific built-in DDS topic. The access control for this topic is set such that any domain participant with the permission necessary to join to the domain is allowed to write to this topic. However, to read the topic, the *DomainParticipant* needs a grant for it in its permissions document.

CONCLUSIONS

This paper described the requirements and design of the security extensions to the DDS standard. The DDS Security extension is comprised of a security model, a pluggable architecture and associated SPIs, and specification of built-in implementations of these SPIs.

The expansion of the DDS standard to include mechanisms to support authentication, authorization and access control, confidentiality, integrity, and auditing provides capabilities that can be used to address a variety of cybersecurity requirements of interest to the Navy; however, prior to employing these mechanisms, analysis must be conducted to determine exactly where and how these new capabilities will be applied.

REFERENCES

[1] Michael W. Masters, Philip Irey, Leslie Madden, Antonio Samuel, "An Open Technical Architecture for the U.S. Navy", Proceedings of ASNE Day 2004.

[2] Integrity and Authentication of Real-Time Data in Navy Combat Systems, Navy SBIR 2010.2-Topic N102-156.

[3] DDS Security Specification, http://www.omg.org/spec/DDS-SECURITY/

[4] DSA, FIPS PUB 186-3 Digital Signature Standard (DSS). http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf

[5] Diffie-Hellman (D-H) Key Agreement Method. IETF RFC 2631. http://tools.ietf.org/html/rfc2631

[6] M. Baugher, D. McGrew, M. Naslund, E. Carrara, K. Norrman, "The Secure Real-time Transport Protocol (SRTP)" IETF RFC 3711, http://tools.ietf.org/html/rfc3711

ACKNOWLEDGEMENTS

The authors would like to thank Adam Simonoff for his leadership as Technical Point of Contact for the Small Business Innovative Research topics that supported the development of Secure DDS. The authors would also like to thank Paul Pazandak from Real-Time Innovations and NSWCDD employees Keith Meyers, Danica Leninsky, Lorna Lockert, and Jim Cunningham for their thoughtful review of the paper as well as the NSWCDD Naval Innovative Science and Engineering program that sponsored experiments conducted with DDS Security.

AUTHOR BIOGRAPHIES

Hamed Soroush, PhD, is a Research Security Engineer at Real-Time Innovations (RTI). His expertise spans security, privacy, forensic, networking, and embedded systems. Dr. Soroush has sought to improve the performance and trustworthiness of networked systems. He leads the RTI security efforts in the Security Working Group of the Industrial Internet Consortium. He holds a Ph.D. in computer science from the University of Massachusetts Amherst and a master's degree in information networking from Carnegie Mellon University. Prior to joining RTI in 2014, Dr. Soroush was a University of Virginia faculty member for one year.

Philip M. Irey IV *earned his master's degree in computer science from the Virginia Polytechnic Institute and State University (Virginia Tech). The focus of his 26 years with the U.S. Navy has been on computing infrastructure for surface ship systems. For the last two years, that focus has shifted to cybersecurity for those systems.*

Gerardo Pardo-Castellote, is the Chief Technology Officer at RTI and an expert in secure real-time software architectures and networking. His professional experience includes real- time distributed middleware, distributed systems and software, control-system software, distributed system software security and software-system design. He was the main author of the OMG DDS Specification and leads the development of the current OMG DDS Security Specification. He currently chairs the Data Distribution Group at the OMG. Dr. Pardo-Castellote received his PhD in electrical engineering from Stanford University. He also holds an MS in computer science, an MSEE from Stanford University, and a BS in physics from the University of Granada, Spain.

Steve Canup, *is a graduate of the Virginia Polytechnic Institute and State University (Virginia Tech) with a master's degree in computer science. He has worked for the U.S. Navy for the past 27 years, focusing the last 8 years on cybersecurity for surface ship systems.*