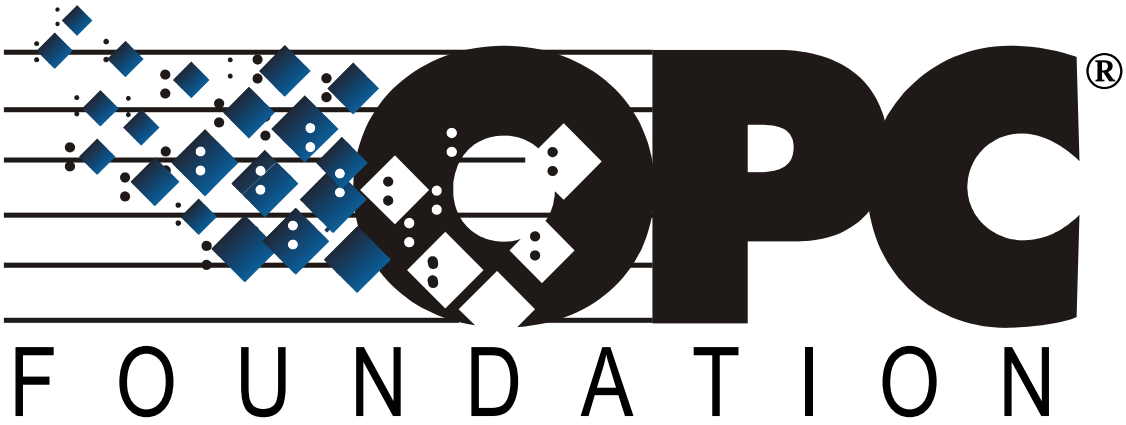

OPC Unified Architecture

Specification

Part 2: Security Model

Release 1.04

August 3, 2018

Specification Type:	Industry Standard Specification	Comments:	
Title:	OPC Unified Architecture Part 2 :Security Model	Date:	August 3, 2018
Version:	Release 1.04	Software:	MS-Word
		Source:	OPC UA Part 2 - Security Model Release 1.04 Specification.docx
Author:	OPC Foundation	Status:	Release

CONTENTS

Figures

Tables

No table of figures entries found.

OPC FOUNDATION

UNIFIED ARCHITECTURE –

FOREWORD

This specification is the specification for developers of OPC UA applications. The specification is a result of an analysis and design process to develop a standard interface to facilitate the development of applications by multiple vendors that shall inter-operate seamlessly together.

AGREEMENT OF USE

GENERAL PROVISIONS

Should any provision of this Agreement be held to be void, invalid, unenforceable or illegal by a court, the validity and enforceability of the other provisions shall not be affected thereby.

This Agreement shall be governed by and construed under the laws of the State of Minnesota, excluding its choice or law rules.

This Agreement embodies the entire understanding between the parties with respect to, and supersedes any prior understanding or agreement (oral or written) relating to, this specification.

ISSUE REPORTING

The OPC Foundation strives to maintain the highest quality standards for its published specifications, hence they undergo constant review and refinement. Readers are encouraged to report any issues and view any existing errata here: http://www.opcfoundation.org/errata

Revision 1.04 Highlights

The following table includes the Mantis issues resolved with this revision.

Mantis ID	Summary	Resolution
3314	5.2.2.1 Wording Needs Clarification	Fixed wording to indicate private key
3315	5.2.2.2 Wording is Incorrect	Fixed to indicate ActivateSession
3316	5.2.2.4 and 5.2.2.5 Wording Needs Clarification	Fixed text as required
3317	6.11(4) Text Needs Clarification	Updated text to indicate OPC UA installation
3318	6.11(2) Text Needs Clarification	Fixed text
3345	Protection-targets Definition change	Fixed all issues (multiple places and issues in the text)
3346	Threat type clarifications/fixes	Fixed all issues (multiple places and issues in the text)
3366	Expand Best-Practices Descriptions.	Fixed all issues (multiple places and issues in the text)
3389	When using HTTPS certificates may be shared by multiple applications. SSLv2 must be disabled for all.	Added text explain HTTPS certificates
3488	Sessionless calls need to be documented	Added text related to Sessionless calls
3684	Remove definition of "OPC UA Application"	Removed definition – it is now defined in Part 1
3933	OAuth2 needs to be added	Added description of OAuth2
3934	User roles	Add description of User roles
3935	Websockets	Added text to describe Websockets transport
3936	Reverse connect	Added text to describe issues with reverse connect

Mantis ID	Summary	Resolution
3968	Support for Pub/Sub	Added text to describe *PubSub* .
3969	Profile updates	Updated specification to point to on-line profiles
4186	Kerberos support	Kerberos support pull from document to match other specification – replaced by OAuth2
4187	Deprecate all text related to WS secure conversation	WS-SecureConversation is no longer used in any specification, it has been removed from all specifications.

OPC Unified Architecture Specification

Part 2: Security Model

1 Scope

This specification describes the OPC Unified Architecture (OPC UA) security model. It describes the security threats of the physical, hardware, and software environments in which OPC UA is expected to run. It describes how OPC UA relies upon other standards for security. It provides definition of common security terms that are used in this and other parts of the OPC UA specification. It gives an overview of the security features that are specified in other parts of the OPC UA specification. It references services, mappings, and *Profile*s that are specified normatively in other parts of this multi-part specification. It provides suggestions or best practice guidelines on implementing security. Any seeming ambiguity between this part and one of the other normative parts does not remove or reduce the requirement specified in the other normative part.

Note that there are many different aspects of security that have to be addressed when developing applications. However, since OPC UA specifies a communication protocol, the focus is on securing the data exchanged between applications. This does not mean that an application developer can ignore the other aspects of security like protecting persistent data against tampering. It is important that the developers look into all aspects of security and decide how they can be addressed in the application.

This part is directed to readers who will develop OPC UA *Client* or *Server* applications or implement the OPC UA services layer. It is also for end Users that wish to understand the various security features and functionality provided by OPC UA. It also offers some suggestions that can be applied when deploying systems. These suggestions are generic in nature since the details would depend on the actual implementation of the *OPC UA Application*s and the choices made for the site security.

2 Reference documents

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

Part 1: OPC UA Specification: Part 1 – Overview and Concepts

http://www.opcfoundation.org/UA/Part1/

Part 3: OPC UA Specification: Part 3 – Address Space Model

http://www.opcfoundation.org/UA/Part3/

Part 4: OPC UA Specification: Part 4 – Services

http://www.opcfoundation.org/UA/Part4/

Part 5: OPC UA Specification: Part 5 – Information Model

http://www.opcfoundation.org/UA/Part5/

Part 6: OPC UA Specification: Part 6 – Mappings

http://www.opcfoundation.org/UA/Part6/

Part 7: OPC UA Specification: Part 7 – Profiles

http://www.opcfoundation.org/UA/Part7/

Part 12: OPC UA Specification: Part 12 – Discovery

http://www.opcfoundation.org/UA/Part12/

Part 14: OPC UA Specification: Part 14 – *PubSub*

http://www.opcfoundation.org/UA/Part14/

SOAP Part 1: SOAP Version 1.2 Part 1: Messaging Framework
http://www.w3.org/TR/soap12-part1/

SOAP Part 2: SOAP Version 1.2 Part 2: Adjuncts
http://www.w3.org/TR/soap12-part2/

SSL/TLS: RFC 2246: The TLS Protocol Version 1.0
http://www.ietf.org/rfc/rfc2246.txt

X509: X.509 Public Key Certificate Infrastructure
http://www.itu.int/rec/T-REC-X.509-200003-I/e

HTTP: RFC 2616: Hypertext Transfer Protocol - HTTP/1.1
http://www.ietf.org/rfc/rfc2616.txt

HTTPS: RFC 2818: HTTP Over TLS
http://www.ietf.org/rfc/rfc2818.txt

IS Glossary: Internet Security Glossary
http://www.ietf.org/rfc/rfc2828.txt

NIST 800-12: Introduction to Computer Security
http://csrc.nist.gov/publications/nistpubs/800-12/

NIST 800-57: Part 3: Application-Specific Key Management Guidance
http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57_PART3_key-management_Dec2009.pdf

NERC CIP: CIP 002-1 through CIP 009-1, by North-American Electric Reliability Council
http://www.nerc.com/page.php?cid=2|20

SPP-ICS: Guide to Industrial Control Systems (ICS) Security
http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-82r2.pdf

SHA-1: Secure Hash Algorithm RFC
http://tools.ietf.org/html/rfc3174

PKI: Public Key Infrastructure article in Wikipedia
https://en.wikipedia.org/wiki/Public_key_infrastructure

X509 PKI: Internet X.509 Public Key Infrastructure
http://www.ietf.org/rfc/rfc3280.txt

RFC 5958: Asymmetric Key Packages
http://tools.ietf.org/search/rfc5208

PKCS #10: Certification Request Syntax Specification
http://tools.ietf.org/html/rfc2986

OAuth2: The OAuth 2.0 Authorization Framework
https://tools.ietf.org/html/rfc6749

JWT: JSON Web Token (JWT)
https://tools.ietf.org/html/rfc7519

OpenID: OpenID Connect Discovery 1.0

https://openid.net/specs/openid-connect-discovery-1_0.html

3 Terms, definitions, and abbreviations

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in Part 1 as well as the following apply.

3.1.1 Access Restriction

A limit on the circumstances where an operation, such as a read, write or a call, can be performed on a *Node*.

Note 1 to entry: Operations can only be performed on a *Node* if the *Client* has the necessary *Permissions* and has satisfied all of the *Access Restrictions*.

3.1.2 Access Token

A digitally signed document that asserts that the subject is entitled to access a *Resource*.

Note 1 to entry: The document includes the name of the subject and the *Resource* being accessed.

3.1.3 Application Instance

individual installation of a program running on one computer.

Note 1 to entry: There can be several *Application Instances* of the same application running at the same time on several computers or possibly the same computer.

3.1.4 Application Instance Certificate

Certificate of an individual *Application Instance* that has been installed in an individual host.

Note 1 to entry: Different installations of one software product would have different *Application Instance Certificates*. The use of an *Application Instance Certificate* for uses outside of what is described in the specification would could greatly reduce the security provided by the *Application Instance Certificate* and should be discouraged.

3.1.5 Asymmetric Cryptography

Cryptography method that uses a pair of keys, one that is designated the *Private Key* and kept secret, the other called the *Public Key* that is generally made available.

Note 1 to entry: 'Asymmetric Cryptography, also known as "public-key cryptography". In an Asymmetric Encryption algorithm when an entity "A" requires *Confidentiality* for data sent to entity "B", then entity "A" encrypts the data with a Public Key provided by entity "B". Only entity "B" has the matching Private Key that is needed to decrypt the data. In an asymmetric Digital Signature algorithm when an entity "A" requires message Integrity or to provide *Authentication* for data sent to entity "B", entity A uses its Private Key to sign the data. To verify the signature, entity B uses the matching Public Key that entity A has provided. In an asymmetric key agreement algorithm, entity A and entity B each send their own Public Key to the other entity. Then each uses their own Private Key and the other's Public Key to compute the new key value.' according to IS Glossary.

3.1.6 Asymmetric Encryption

the mechanism used by *Asymmetric Cryptography* for encrypting data with the *Public Key* of an entity and for decrypting data with the associated *Private Key*

3.1.7 Asymmetric Signature

the mechanism used by *Asymmetric Cryptography* for signing data with the *Private Key* of an entity and for verifying the data's signature with the associated *Public Key*

3.1.8 Auditability

security objective that assures that any actions or activities in a system can be recorded

3.1.9 Auditing

the tracking of actions and activities in the system, including security related activities where *Audit* records can be used to review and verify system operations

3.1.10 **Authentication**

security objective that assures that the identity of an entity such as a *Client*, *Server*, or user can be verified

3.1.11 **Authorization**

the ability to grant access to a system resource

Note 1 to entry: *Authorization* of access to resources should be based on the need-to-know principle. It is important that access is restricted in a system.

3.1.12 **AuthorizationService**

A *Server* which validates a request to access a *Resource* returns an *Access Token* that grants access to the *Resource*.

Note 1 to entry: The *AuthorizationService* is also called STS (Security Token Service) in other standards.

3.1.13 **Availability**

security objective that assures that the system is running normally. That is, no services have been compromised in such a way to become unavailable or severely degraded

3.1.14 **Certificate Authority**

entity that can issue *Certificate*s, also known as a CA

Note 1 to entry: The *Certificate* certifies the ownership of a *Public Key* by the named subject of the *Certificate*. This allows others (relying parties) to rely upon signatures or assertions made by the *Private Key* that corresponds to the *Public Key* that is certified. In this model of trust relationships, a CA is a trusted third party that is trusted by both the subject (owner) of the *Certificate* and the party relying upon the *Certificate*. CA s are characteristic of many *Public Key infrastructure* (PKI) schemes

3.1.15 **CertificateStore**

persistent location where *Certificate*s and *Certificate* revocation lists (CRLs) are stored

Note 1 to entry: It may be a disk resident file structure or on Windows platforms it may be a Windows registry location.

3.1.16 **Claim**

A statement in an *Access Token* that asserts information about the subject which the *Authorization Service* knows to be true.

Note 1 to entry: *Claims* can include username, email, and *Roles* granted to the subject.

3.1.17 **Confidentiality**

security objective that assures the protection of data from being read by unintended parties

3.1.18 **Cryptography**

transforming clear, meaningful information into an enciphered, unintelligible form using an algorithm and a key

3.1.19 **Cyber Security Management System**

program designed by an organization to maintain the security of the entire organization’s assets to an established level of *Confidentiality*, *Integrity*, and *Availability*, whether they are on the business side or the industrial automation and control systems side of the organization

3.1.20 **Digital Signature**

value computed with a cryptographic algorithm and appended to data in such a way that any recipient of the data can use the signature to verify the data’s origin and *Integrity*

3.1.21 **Hash Function**

algorithm such as SHA-1 for which it is computationally infeasible to find either a data object that maps to a given hash result (the "one-way" property) or two data objects that map to the same hash result (the "collision-free" property) , see IS Glossary

3.1.22 Hashed Message Authentication Code

MAC that has been generated using an iterative *Hash Function*

3.1.23 Integrity

security objective that assures that information has not been modified or destroyed in an unauthorized manner, see IS Glossary

3.1.24 Identity Provider

A *Server* which verifies credentials provided by a *Security Principal* and returns a token which can be passed to an associated *Authorization Service*.

3.1.25 Key Exchange Algorithm

protocol used for establishing a secure communication path between two entities in an unsecured environment whereby both entities apply a specific algorithm to securely exchange secret keys that are used for securing the communication between them

Note 1 to entry: A typical example of a *Key Exchange Algorithm* is the SSL Handshake Protocol specified in SSL/TLS.

3.1.26 Message Authentication Code

short piece of data that results from an algorithm that uses a secret key (see *Symmetric Cryptography*) to hash a *Message* whereby the receiver of the *Message* can check against alteration of the *Message* by computing a *MAC* that should be identical using the same *Message* and secret key

3.1.27 Message Signature

Digital Signature used to ensure the *Integrity* of *Message*s that are sent between two entities

Note 1 to entry: There are several ways to generate and verify *Message Signatures* however they can be categorized as symmetric (See Clause 3.1.40) and asymmetric (See Clause 3.1.5) approaches.

3.1.28 Non-Repudiation

strong and substantial evidence of the identity of the signer of a *Message* and of *Message Integrity*, sufficient to prevent a party from successfully denying the original submission or delivery of the *Message* and the *Integrity* of its contents

3.1.29 Nonce

random number that is used once typically by algorithms that generate security keys

3.1.30 Permission

The right to execute an operation, such as a read, write or a call, on a *Node*.

3.1.31 Private Key

the secret component of a pair of cryptographic keys used for *Asymmetric Cryptography*

Note 1 to entry: *Public Key* and *Private Key* are always generated as a pair. If either is updated the other must also be updated

3.1.32 Public Key

the publicly-disclosed component of a pair of cryptographic keys used for *Asymmetric Cryptography*, see IS Glossary

Note 1 to entry: *Public Key* and *Private Key* are always generated as a pair. If either is updated the other must also be updated

3.1.33 Public Key Infrastructure

the set of hardware, software, people, policies, and procedures needed to create, manage, store, distribute, and revoke *Certificates* based on *Asymmetric Cryptography*

Note 1 to entry: The core PKI functions are to register users and issue their public-key *Certificates*, to revoke *Certificates* when required, and to archive data needed to validate *Certificates* at a much later time. Key pairs for data *Confidentiality* may be generated by a *Certificate* authority (CA); it is a good idea to require a *Private Key* owner to generate their own key pair as it improves security because the *Private Key* would never be transmitted according to IS Glossary. See PKI and X509 PKI for more details on *Public Key* Infrastructures.

3.1.34 **Resource**

A secured entity which an application needs to access.

Note 1 to entry: A *Resource* is usually a *Server*.

3.1.35 **Rivest-Shamir-Adleman**

algorithm for *Asymmetric Cryptography*, invented in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman, see IS Glossary

3.1.36 **Role**

A function assumed by a *Client* when it accesses a *Server*.

Note 1 to entry: A *Role* may refer to a specific job function such as operator or engineer.

3.1.37 **Scope**

A *Claim* representing a subset of a *Resource*.

Note 1 to entry: A *Scope* may indicate a set *Nodes* managed by a *Server*.

3.1.38 **Security Key Service**

A *Server* that accepts *Access Tokens* issued by the *Authorization Service* and returns security keys that can be used to access the specified *Resource.*

Note 1 to entry: The keys are typically used for cryptography operations such as encrypting or decrypting messages sent on a *PubSub* stream.

3.1.39 **Secure Channel**

in OPC UA, a communication path established between an OPC UA *Client* and *Server* that have authenticated each other using certain OPC UA services and for which security parameters have been negotiated and applied

3.1.40 **Symmetric Cryptography**

branch of cryptography involving algorithms that use the same key for two different steps of the algorithm (such as encryption and decryption, or signature creation and signature verification), see IS Glossary

3.1.41 **Symmetric Encryption**

the mechanism used by *Symmetric Cryptography* for encrypting and decrypting data with a cryptographic key shared by two entities

3.1.42 **SecurityGroup**

publisher and subscribers that utilize a shared security context

3.1.43 **Symmetric Signature**

the mechanism used by *Symmetric Cryptography* for signing data with a cryptographic key shared by two entities

Note 1 to entry: The signature is then validated by generating the signature for the data again and comparing these two signatures. If they are the same then the signature is valid, otherwise either the key or the data is different from the two entities.

3.1.44 **TrustList**

list of *Certificate*s that an OPC UA Application has been configured to trust

3.1.45 **Transport Layer Security**

standard protocol for creating *Secure Channel*s over IP based networks

3.1.46 **X.509 Certificate**

Certificate in one of the formats defined by X.509 v1, 2, or 3

Note 1 to entry: An *X.509 Certificate* contains a sequence of data items and has a *Digital Signature* computed on that sequence. OPC UA only uses V3.

3.2 Abbreviations

AES	Advanced Encryption Standard
CA	Certificate Authority
CRL	Certificate Revocation List
CSMS	Cyber Security Management System
DNS	Domain Name System
DSA	Digital Signature Algorithm
ECDH	Elliptic Curve Diffie-Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm
HMAC	Hash-based Message Authentication Code
JSON	JavaScript Object Notation
JWT	JSON Web Token
NIST	National Institute of Standard and Technology
PKI	Public Key Infrastructure
RSA	public key algorithm for signing or encryption, Rivest, Shamir, Adleman
SHA	Secure Hash Algorithm (Multiple versions exist SHA1, SHA256,…)
SKS	Security Key Server
SOAP	Simple Object Access Protocol
SSL	Secure Sockets Layer
TLS	Transport Layer Security
UA	Unified Architecture
UACP	Unified Architecture Connection Protocol
UADP	Unified Architecture Datagram Protocol
URI	Uniform Resource Identifier
XML	Extensible Mark-up Language

3.3 Conventions for security model figures

The figures in this document do not use any special conventions. Any conventions used in a particular figure are explained for that figure.

4 OPC UA security architecture

4.1 OPC UA security environment

OPC UA is a protocol used between components in the operation of an industrial facility at multiple levels: from high-level enterprise management to low-level direct process control of a device. The use of OPC UA for enterprise management involves dealings with customers and suppliers. It may be an attractive target for industrial espionage or sabotage and may also be exposed to threats through untargeted malware, such as worms, circulating on public networks. Disruption of communications at the process control could result in financial losses, affect employee and public safety or cause environmental damage.

OPC UA will be deployed in a diverse range of operational environments with varying assumptions about threats and accessibility, and with a variety of security policies and enforcement regimes. OPC UA, therefore, provides a flexible set of security mechanisms. Figure 1 is a composite that shows a combination of such environments. Some OPC UA *Applications* are on the same host and can be easily protected from external attack. Some OPC UA *Applications* are on different hosts in the same operations network and might be protected by the security boundary protections that separate the operations network from external connections. Some OPC UA *Applications* run in relatively open environments where users and applications might be difficult to control. Other OPC UA *Applications* are embedded in control systems that have no direct electronic connection to external systems.

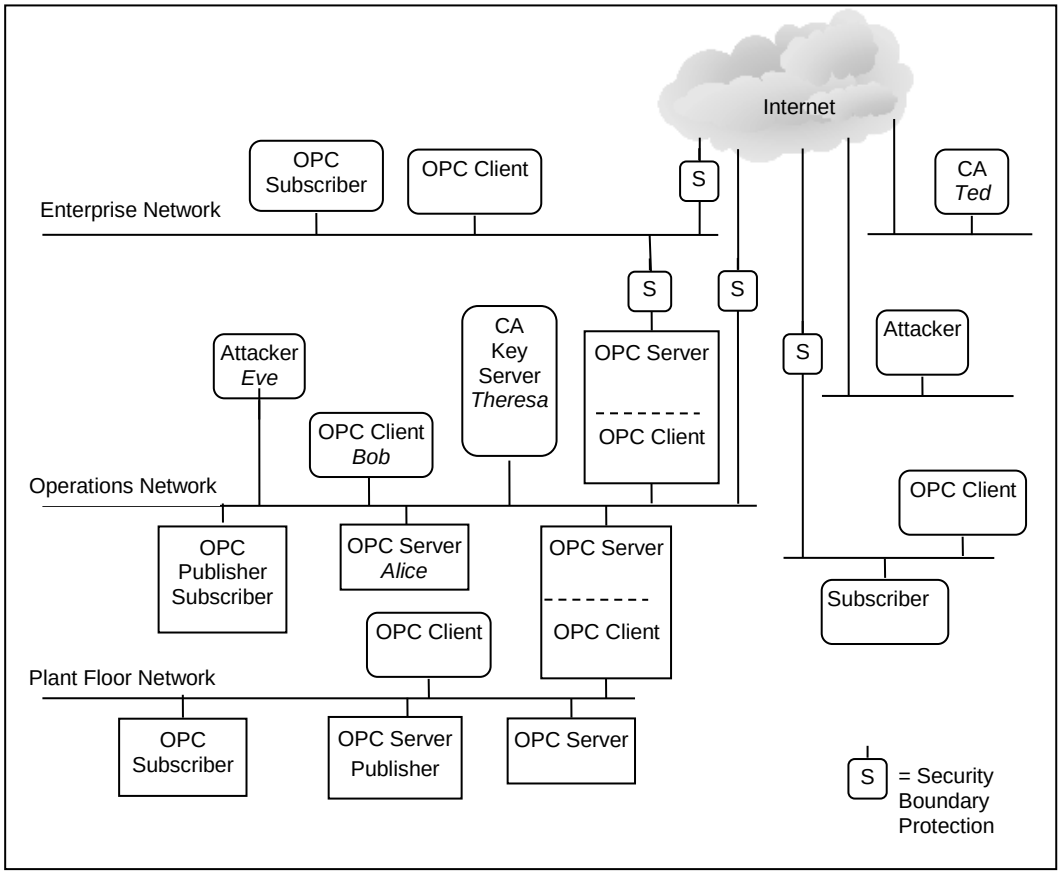

Figure 1 - OPC UA network example

OPC UA also supports multiple protocols and communication technologies, that might require different levels of security and different security infrastructure. For example, both Client - Server and Publisher - Subscriber communication is shown in Figure 1

4.2 Security objectives

4.2.1 Overview

Fundamentally, information system security reduces the risk of damage from attacks. It does this by identifying the threats to the system, identifying the system's vulnerabilities to these threats, and providing countermeasures. The countermeasures reduce vulnerabilities directly, counteract threats, or recover from successful attacks.

Industrial automation system security is achieved by meeting a set of objectives. These objectives have been refined through many years of experience in providing security for information systems in general and they remain quite constant despite the ever-changing set of threats to systems. They are described in the sub clause 5.1 and sub clause 5.2 reconciles these objectives against the OPC UA functions. Clause 6 offers additional best practice guidelines to *Client* and *Server* developers or those that deploy *OPC UA Applications*.

4.2.2 Authentication

Entities such as clients, *Servers*, and users should prove their identities. *Authentication* can be based on something the entity is, has, or knows.

4.2.3 Authorization

The access to read, write, or execute resources should be authorized for only those entities that have a need for that access within the requirements of the system. *Authorization* can be as coarse-grained as allowing or disallowing a *Client* to access a *Server* or it could be much finer grained such as allowing specific actions on specific information items by specific users. The granularity of a system depends in part on the functionality supported by the *Server*, but in general *Authorization* should be

given based on the need-to-know principle i.e. a user should be granted access only to information they require for the function they are performing.

4.2.4 Confidentiality

Data is protected from passive attacks such as eavesdropping, whether the data is being transmitted, in memory, or being stored. To provide *Confidentiality,* data encryption algorithms using special secrets for securing data are used along with *Authentication* and *Authorization* mechanisms for accessing that secret.

4.2.5 Integrity

Receivers receive the same information that the original sender sent, without the data being changed during transmission.

4.2.6 Non- Repudiation

Repudiation is the rejection or denial of something as valid or true. *Non-Repudiation* is assuring that something that actually occurred cannot be claimed as having not occurred. A security service that provides this protection can be one of two types:

- One in which the recipient of the data gets and stores information proving that the data came from the originator. This blocks the originator from claiming they never sent the data.
- One in which the sender of the data gets confirmation that the data was received by the recipient as intended.

4.2.7 Auditability

Actions taken by a system must be recorded in order to provide evidence to stakeholders:

- that this system works as intended (successful actions are tracked).
- that identify the initiator of certain actions (user activity is tracked).
- that attempts to compromise the system were denied (unsuccessful actions are tracked).

4.2.8 Availability

Availability is impaired when the execution of software that needs to run is turned off or when the software or communication system is overwhelmed by processing input. Impaired *Availability* in OPC UA can appear as slowing down of *Subscription* performance or the inability to add *Sessions* for example.

4.3 Security threats to OPC UA systems

4.3.1 Overview

OPC UA provides countermeasures to resist threats to the security of the information that is communicated. The sub clause 4.3 list the currently known threats to environments in which OPC UA will be deployed, and Sub-clause 5.1 reconciles these threats against the OPC UA functions.

4.3.2 Denial of Service

4.3.2.1 Overview

The prevention of authorized access to a system resource or the delaying of system operations and functions. This can occur from a number of different attacks vectors including message flooding, resource exhaustion and application crashes. Each of these are described separately.

Denial of Service impacts *Availability*.

See 5.1.2 for the reconciliation of this threat.

4.3.2.2 Message flooding

For *Client-Server*, an attacker can send a large volume of *Message*s, or a single *Message* that contains a large number of requests, with the goal of overwhelming the OPC UA *Server* or dependent components such as CPU, TCP/IP stack, operating system, or the file system. Flooding attacks can be conducted at multiple layers including OPC UA, SOAP, [HTTP] or TCP.

Message flooding attacks can use both well-formed and malformed *Message*s. In the first scenario, the attacker could be a malicious person using a legitimate *Client* to flood the *Server* with requests. Two cases exist, one in which the *Client* does not have a *Session* with the *Server* and one in which it does. *Message* flooding may impair the ability to establish OPC UA *Sessions* or terminate an existing *Session*. In the second scenario, an attacker could use a malicious *Client* that floods an OPC UA *Server* with malformed *Message*s in order to exhaust the *Server*'s resources.

For *PubSub*, an attacker can send a large volume of dataset messages with the goal of overwhelming the subscriber, the middleware or dependent components such as CPU, TCP/IP stack, operating system, or the file system. Flooding attacks can be conducted at multiple layers including OPC UA, UDP, AMQP, MQTT.

As in *Client-Server, PubSub* message flooding attacks can use both well-formed and malformed *Messages*. For well-formed *Messages*, the attacker could be one in which the publisher is not a member of the *SecurityGroup* and one in which it is a member. For malformed *Messages,* an attacker could use a malicious *Publisher* that floods a network with malformed *Messages* in order to exhaust the system's resources.

In general, *Message* flooding may impair the ability to communicate with an OPC UA entity and result in denial of service.

4.3.2.3 Resource Exhaustion

An attacker can send a limited number of messages that obtain a resource on the system. The commands are typically valid, but they each use up a resource resulting in a single *Client* obtaining all resources blocking valid *Clients* from accessing the *Server*. For example, on a *Server* in which only 10 *Sessions* are available a malicious person using a legitimate *Client,* might obtain all 10 *Sessions*. Or a malicious *Client* might try to open 10 secure channels, without actually completing the process.

Resource exhaustion attacks do not occur in the same manner for *PubSub* communications since no session or resources are allocated. For *PubSub* communication, the *Publisher* is not susceptible. In broker-less *PubSub* communication, the *Subscriber* can, with the use of filters, bypass any resource exhaustion issues. In broker case, both the *Publisher* and *Subscriber* must connect to the broker. Although the *Publisher* and *Subscriber* are not directly susceptible (as in the broker-less case), the broker is susceptible. The details for broker communication is not part of OPC UA but is defined by the broker protocol.

4.3.2.4 Application Crashes

An attacker can send special message that will cause an application to crash. This is usually the result of a known problem in a stack or application. These system bugs can allow a *Client* to issue a command that would cause the *Server* to crash, as an alternate it might be a *Server* that can respond to a legitimate message with a response that would cause the *Client* to crash. The attacker could also be a *Publisher* that issues a *Message* that would cause *Subscribers* to crash.

4.3.3 Eavesdropping

Eavesdropping is the unauthorized disclosure of sensitive information that might result directly in a critical security breach or be used in follow-on attacks.

If an attacker has compromised the underlying operating system or the network infrastructure, then the attacker might be able to record and capture *Message*s. It may be beyond the capability of a *Client* or *Server* to recover from a compromised operating system.

Eavesdropping impacts *Confidentiality* directly and if session establishment is not secured *Authentication* and *Authorization*. It also indirectly threatens all other security objectives.

See 5.1.3 for the reconciliation of this threat.

4.3.4 **Message spoofing**

This includes feigning identities (user, application, process etc.). An attacker may forge *Message*s from a *Client* or a *Server* or a *Publisher* where the messages are forged to attempt to appear to be from an application other that the sending application or process. Spoofing may occur at multiple layers in the protocol stack.

By spoofing *Message*s from a *Client,* a *Server* or *Publisher*, attackers may perform unauthorized operations and avoid detection of their activities.

Message spoofing impacts *Integrity* and *Authorization*.

See 5.1.4 for the reconciliation of this threat.

4.3.5 **Message alteration**

Network traffic and application layer *Message*s may be captured or modified and forwarded to OPC UA *Client*s, *Servers,* and *Subscribers*. *Message* alteration may allow illegitimate access to a system.

Message alteration impacts *Integrity, Authorization, Auditability, Non-Repudiation* and during session / secure channel establishment *Authentication*.

See 5.1.5 for the reconciliation of this threat.

4.3.6 **Message replay**

Network traffic and valid application layer *Message*s may be captured and resent to OPC UA *Client*s, *Servers* and *Subscribers* at a later stage without modification. An attacker could misinform the user or send a valid command such as opening a valve but at an improper time, so as to cause damage or property loss. An attacker may attempt to establish a *Session* using a recorded *Session*.

Message replay impacts *Authorization* and during *Session* / secure channel establishment *Authentication.* See 5.1.6 for the reconciliation of this threat.

4.3.7 **Malformed Messages**

An attacker can craft a variety of *Message*s with invalid *Message* structure (malformed XML, SOAP, UA Binary, etc.) or data values, and send them to OPC UA *Client*s, *Servers* or *Subscribers*.

The OPC UA *Client*, *Server* or *Subscriber* may incorrectly handle certain malformed *Message*s by performing unauthorized operations or processing unnecessary information. It might result in a denial or degradation of service including termination of the application or, in the case of embedded devices, a complete crash. In a worst-case scenario an attacker could use malformed *Message*s as a pre-step for a multi-level attack to gain access to the underlying system of an *OPC UA Application*.

Malformed *Message*s impacts *Integrity* and *Availability*.

See 5.1.7 for the reconciliation of this threat.

4.3.8 **Server profiling**

An attacker tries to deduce the identity, type, software version, or vendor of the *Server* or *Client* in order to apply knowledge about specific vulnerabilities of that product to mount a more intrusive or damaging attack. The attacker might profile the target by sending valid or invalid formatted *Message*s to the target and try to recognize the type of target by the pattern of its normal and error responses.

Server profiling impacts all of the security objectives indirectly.

See 5.1.8 for the reconciliation of this threat.

4.3.9 **Session hijacking**

An attacker may use information (retrieved by sniffing the communication or by guessing) about a running *Session* established between two applications to inject manipulated *Message*s (with valid session information) that allow him or her to take over the *Session* from the authorized user.

An attacker may gain unauthorized access to data or perform unauthorized operations.

Session hijacking impacts all of the security objectives.

See 5.1.9 for the reconciliation of this threat.

4.3.10 **Rogue Server**

An attacker builds a malicious OPC UA *Server* or installs an unauthorized instance of a genuine OPC UA *Server* in a system. The rouge *Server* may attempt to masquerade as a legitimate UA *Server* or it may simply appear as a new *Server* in the system.

The OPC *Client* may disclose necessary information.

A rogue *Server* impacts all of the security objectives except *Integrity* and *Non-Repudiation*.

See 5.1.10 for the reconciliation of this threat.

4.3.11 **Rogue Publisher**

An attacker who builds a malicious OPC UA *Publisher* or installs an unauthorized instance of a genuine OPC UA *Publisher* in a system. The rouge *Publisher* may attempt to masquerade as a legitimate UA *Publisher* or it may simply appear as a new *Publisher* in the system.

A rogue *Publisher* impacts all of the security objectives except *Integrity* and *Non-Repudiation*.

See 5.1.10 for the reconciliation of this threat.

4.3.12 **Compromising user credentials**

An attacker obtains user credentials such as usernames, passwords, *Certificate*s, or keys by observing them on papers, on screens, or in electronic communications, or by cracking them through guessing or the use of automated tools such as password crackers.

An unauthorized user could launch and access the system to obtain all information and make control and data changes that harm plant operation or information. Once compromised credentials are used, subsequent activities may all appear legitimate.

Compromised user credentials impact *Authentication*, *Authorization* and *Confidentiality*.

See 5.1.11 for the reconciliation of this threat.

4.3.13 **Repudiation**

This is not a direct attack, since it is not about communication, but it is the trust following the communication. *Repudiation* causes trust issues with either the sender or the receiver of the data.

Repudiation impacts *Non-Repudiation*.

See 5.1.12 for the reconciliation of this threat.

4.4 OPC UA relationship to site security

OPC UA security works within the overall *Cyber Security Management System* (*CSMS*) of a site. Sites often have a *CSMS* that addresses security policy and procedures, personnel, responsibilities, audits, and physical security. A *CSMS* typically addresses threats that include those that were described in 4.3. They also analyse the security risks and determine what security controls the site needs.

Resulting security controls commonly implement a “defence-in-depth” strategy that provides multiple layers of protection and recognizes that no single layer can protect against all attacks. Boundary protections, shown as abstract examples in Figure 1, may include firewalls, intrusion detection and prevention systems, controls on dial-in connections, and controls on media and computers that are brought into the system. Protections in components of the system may include hardened configuration of the operating systems, security patch management, anti-virus programs, and not allowing email in

the control network. Standards that may be followed by a site include [NERC CIP] and [IEC 62351] which are referenced in Clause 2.

The security requirements of a site *CSMS* apply to its OPC UA interfaces. That is, the security requirements of the OPC UA interfaces that are deployed at a site are specified by the site, not by the OPC UA specification. OPC UA specifies features that are intended so that conformant OPC UA *Applications* can meet the security requirements that are expected to be made by sites where they will be deployed. Those who are responsible for the security at the site should determine how to meet the site requirements with OPC UA conformant products.

The system owner that installs OPC UA *Applications* should analyse its security risks and provide appropriate mechanisms to mitigate those risks to achieve an acceptable level of security. OPC UA meets the wide variety of security needs that might result from such individual analyses. OPC UA *Applications* are required to be implemented with certain security features which are available for the system owner's optional use. Each system owner should be able to tailor a security solution that meets its security and economic requirements using a combination of mechanisms available within the OPC UA specification and external to OPC UA.

The security requirements placed on the OPC UA *Applications* deployed at a site are specified by the site *CSMS*, not by the OPC UA specification. The OPC UA security specifications, however, are requirements placed upon OPC UA *Applications*, and recommendations of how OPC UA should be deployed at a site in order to meet the security requirements that are anticipated to be specified at the site.

OPC UA addresses some threats as described in 4.3. The OPC Foundation recommends that OPC UA *Application* developers address the remaining threats, as detailed in Clause 6. Threats to infrastructure components that might result in the compromise of operating systems, where OPC UA *Applications* are running, are not addressed by OPC UA.

4.5 OPC UA security architecture

4.5.1 Overview

The OPC UA security architecture is a generic solution that allows implementation of the required security features at various places in the *OPC UA Application* architecture. Depending on the different mappings described in Part 6, the security objectives are addressed at different levels. The OPC UA security architecture, for *Client* / *Server* communication is structured in an Application Layer and a Communication Layer atop the Transport Layer as shown in Figure 2.

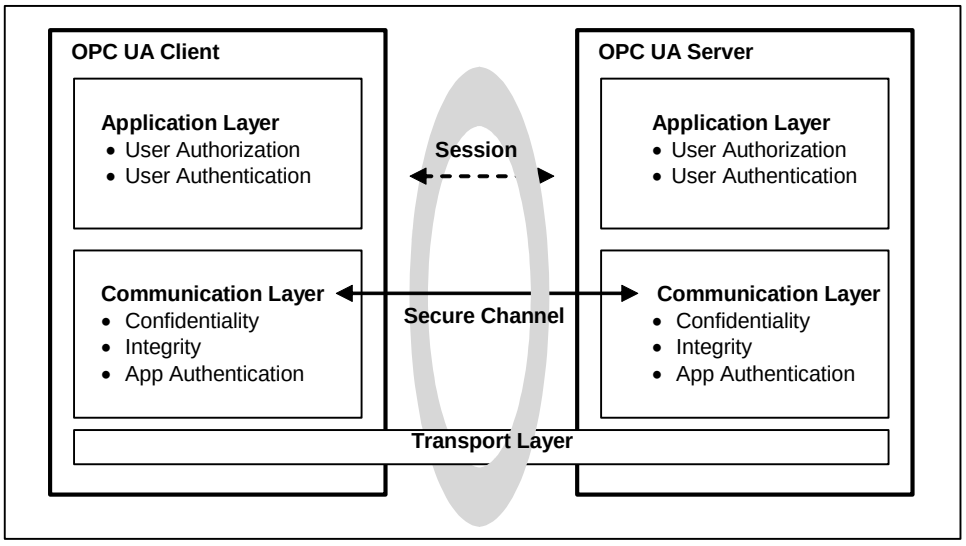

Figure 2 – OPC UA security architecture – Client / Server

OPC UA also supports a Publish - Subscribe communications architecture (*PubSub*) and the security architecture for that communication is illustrated in Figure 3.

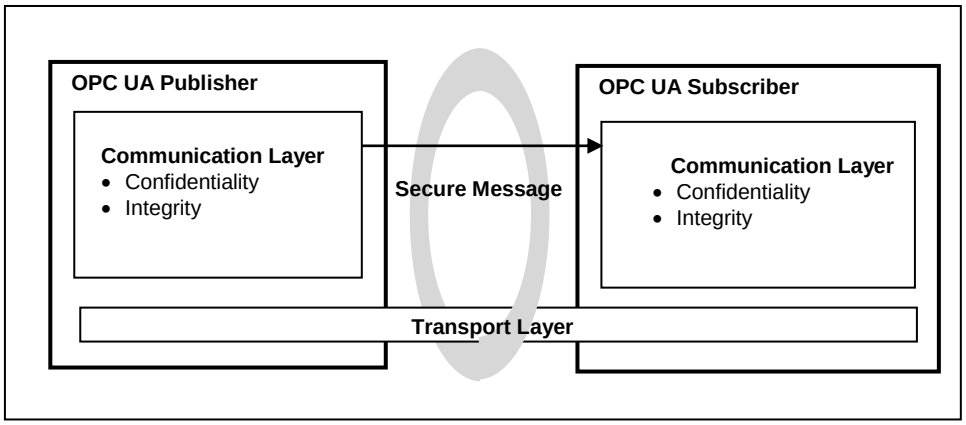

Figure 3 - OPC UA security architecture- Publisher - Subscriber

4.5.2 **Client / Server**

Client / *Server* communication can include both *Session* and session-less communication.

The routine work of a *Client* application and a *Server* application to transmit information, settings, and commands is done in a *Session* in the Application Layer. The Application Layer also manages the security objectives user *Authentication* and user *Authorization*. The security objectives that are managed by the Application Layer are addressed by the *Session Services* that are specified in Part 4. A *Session* in the Application Layer communicates over a *Secure Channel* that is created in the Communication Layer and relies upon it for secure communication. All of the *Session* data is passed to the Communication Layer for further processing.

Although a *Session* communicates over a *Secure Channel* and has to be activated before it can be used, the binding of users, *Sessions*, and *Secure Channels* is flexible.

Impersonation allows a user to take ownership of an existing *Session*.

If a *Secure Channel* breaks, the *Session* will remain valid for a period of time allowing the *Client* to re-establish the connection to the *Session* via a new *Secure Channel.* Otherwise, the *Session* closes after its lifetime expires.

The Communication Layer provides security mechanisms to meet *Confidentiality*, *Integrity* and application *Authentication* as security objectives. One essential mechanism to meet these security objectives is to establish a *Secure Channel* (see 4.13) that is used to secure the communication between a *Client* and a *Server*. The *Secure Channel* provides encryption to maintain *Confidentiality*, *Message Signatures* to maintain *Integrity* and *Certificates* to provide application *Authentication.* The data that comes from the Application Layer is secured and passes the “secured” data to the Transport Layer. The security mechanisms that are managed by the Communication Layer are provided by the *Secure Channel* Services that are specified in Part 4.

The security mechanisms provided by the *Secure Channel* services are implemented by a protocol stack that is chosen for the implementation. Mappings of the services to some of the protocol stack options are specified in Part 6 which define how functions in the protocol stack are used to meet the OPC UA security objectives.

The Communication Layer can represent an OPC UA connection protocol stack. OPC UA specifies alternative stack mappings that can be used as the Communication Layer. These mappings are described in Part 6.

If the OPC UA Connection Protocol (UACP) is used, then functionality for *Confidentiality*, *Integrity*, application *Authentication*, and the *Secure Channel* are similar to the SSL/TLS specifications, as described in Part 6.

The Transport Layer handles the transmission, reception, and the transport of data that is provided by the Communication Layer.

To survive the loss of the Transport Layer connections (e.g. TCP connections) and resume with a new connection, the Communication Layer is responsible for re-establishing the Transport Layer connection without interrupting the logical *Secure Channel*.

The transport layer can also be used to implement *Confidentiality* and *Integrity* by using HTTPS as described in Part 6. It is important to note that HTTPS certificates can be (and often are) shared by multiple applications on a platform and that they can be compromised outside of the OPC UA usage of them. All applications on the platform that use the same shared certificate have the same settings, such as disabling of SSLv2.

OPC UA provides a session-less *Service* invocation (see Part 4 overview and see Part 6 for details). The session-less communication provides *User Authentication.* The communication channel provides *Confidentiality* and *Integrity.* The communication channel might be an OPC UA Secure channel (without a session). It might be a communication channel, such as HTTPS, which relies on transport protocols to provide security. In addition, *User Authentication* and/or *Application Authentication* can also be established by the use of an *Access Token* which is obtained from an *AuthorizationService* (see Part 6 for details).

Additional communication mappings are described in Part 6. These mappings may rely on transport protocols to provide *Confidentiality* and *Integrity*. One example is Websockets, which utilizes HTTPS transport layer security to provide *Confidentiality* and *Integrity*.

4.5.3 Publish-Subscribe

4.5.3.1 Overview

The *PubSub* can be deployed in two environments, one in which a broker exists and one which is broker less. For a detailed describe of this model see Part 14The two environments have different security considerations associated with them, and each will be described separately.

4.5.3.2 Broker-less

The broker-less *PubSub* communication model provides *Confidentiality* and *Integrity.* This is accomplished using *Symmetric Encryption* and signature algorithms. The required symmetric keys are distributed by a Security Key Server (SKS) (see Part 14 for additional details). The SKS makes use of the standard *Client*/*Server* security described in the previous section to establish application *Authentication* as well as user *Authentication*. This approach allows all applications (*Publishers* and/or *Subscribers*) in a *SecurityGroup* to share information

A benefit of using shared symmetric keys is the high performance they offer, but a drawback is that for a group of applications that use a shared symmetric key, all of the applications in the group have the same rights. All applications must trust all other applications in the group. Any application (*Publisher* or *Subscriber*) in the group can publish a message and any application (*Publisher* or *Subscriber*) in the group can decode the message.

For example, a system might be composed of a shared symmetric group that is composed of a controller (*Publisher*) and three *Subscribers* (say HMI's). The controller is publishing messages and the HMIs are receiving the messages. If one of the HMIs is compromised, it might start publishing messages also. The other two HMIs will not be able to tell that the message was not sent from the controller. One possible solution to this situation could be if the shared symmetric group is composed of just the controller and one HMI. Additional groups would be created for each HMI, then no HMI could affect the other HMIs. Other possible solutions could also involve the network architecture and services, such as unicast restricted network communication, but these are outside the scope of the of OPC UA specification. The configuration of *SecurityGroups* requires careful consideration when deploying systems to ensure security.

4.5.3.3 Broker

When using a *Broker* in the *PubSub* model, the same shared symmetric key concepts as defined in 4.5.3.2 can be used to provide *Confidentiality* and *Integrity.* Furthermore, communication to the *Broker* can be secured according the rules defined for the *Broker*. These rules are not defined in the OPC Foundation specification but are defined by the *Middleware*. In many cases the *Middleware*

requires the authorization of both the *Publishers* and the *Subscribers* before they can interact with the *Broker*. The *Broker* interactions can provide security mechanisms to meet *Confidentiality*, *Integrity* and application or user *Authentication* as security objectives. If the published message is not secured using the shared symmetric key concepts, the message content is visible to the *Broker* which creates some risk of man-in-the-middle attacks. The use of the shared symmetric keys eliminates this risk.

4.6 SecurityPolicies

A *SecurityPolicy* specifies which security mechanisms are to be used and are derived from a Security *Profile* (see 4.7 for details). Security policies are used by the *Server* to announce which mechanisms it supports and by the *Client* to select one to use with the *Secure Channel* it wishes to open or for the session-less connection it wishes to make. *SecurityPolicies* are also used with *PubSub* communication. *SecurityPolicies* include the following information:

- algorithms for signing and encryption
- algorithm for key derivation

The choice of allowed *SecurityPolicies* is normally made by the administrator typically when the OPC UA *Applications* are installed. The available security policies are specified in Part 7. The Administrator can at a later time also change or modify the selection of allowed *SecurityPolicies* as circumstances dictate.

The announcement of security policies is handled by special discovery services specified in Part 4. More details about the discovery mechanisms and policy announcement strategies can be found in Part 12.

In the *Client Server* communications pattern, each *Client* can select a policy independent of the policy selected by other *Clients*.

For the *Publish Subscribe* communications pattern, the *SecurityPolicy* is associated with a published *DataSet* and all *Subscribers* must utilize the same *SecurityPolicy*.

Since computing power increases every year, specific algorithms that are considered as secure today can become insecure in the future, therefore, it makes sense to support different security policies in an *OPC UA Application* and to be able to adopt more as they become available. NIST or other agencies even make predictions about the expected lifetime of algorithms (see NIST 800-57). The list of supported security policies will be updated based on recommendation such as those published by NIST. From a deployment point of view it is important that the periodic site-review checks that the currently selected list of security profiles still fulfil the required security objectives and if they do not, then a newer selection of *Security Profiles* is selected

There is also the case that new security policies are composed to support new algorithms that improve the level of security of OPC UA products. The application architecture of OPC UA *Application* should be designed in a way that it is possible to update or add additional cryptographic algorithms to the application with little or no coding changes.

Part 7 specifies several policies which are identified by a specific unique URI. To improve interoperability among vendors' products, *Server* and *Publisher* products implement these policies rather than define their own. *Clients* and *Subscribers* support the same policies.

4.7 Security Profiles

OPC UA *Client* and *Server* products are certified against *Profiles* that are defined in Part 7. Some of the *Profiles* specify security functions and others specify other functionality that is not related to security. The *Profiles* impose requirements on the certified products but they do not impose requirements on how the products are used. A consistent minimum level of security is required by the various *Profiles*. However, different *Profiles* specify different details such as which encryption algorithms are required for which OPC UA functions. If a problem is found in one encryption algorithm then the OPC Foundation can define a new *Profile* that is similar, but that specifies a different encryption algorithm that does not have a known problem. Part 7 is the normative specification of the *Profiles,* but *Profiles* are maintained in an on-line application (http://opcfoundation-onlineapplications.org/profilereporting/) allowing for updating of *Profiles*, especially security related profiles, in a more timely manner than allowed by documentation publication cycles.

Policies refer to many of the same security choices as *Profiles*; however the policy specifies which of those choices to use in the *Session*. The policy does not specify the range of choices that the product offers, they are described in the *Profiles* that it supports.

These policies are included in Certification Testing associated with OPC UA *Applications*. The Certification Testing ensures that the standard is followed and that the appropriate security algorithms are supported.

Each security mechanism in OPC UA is provided in OPC UA *Applications* in accordance with the *Profiles* with which the OPC UA *Application* complies. At the site, however, the security mechanisms may be deployed optionally. In this way each individual site has all of the OPC UA security functions available and can choose which of them to use to meet its security objectives.

Security *Profiles* describe a *Profile* “None” that is used for testing, but if any other more secure *Profiles* are available this *Profile* is disabled by default. *Profile* “None” provides no security.

4.8 Security Mode Settings

OPC UA supports the selection of several security modes: “None”, “Sign”, “SignAndEncrypt”. Security mode “None” can only be used with security Profile None. It is disabled for all other security *Profiles*. The choice of “Sign” or “SignAndEncrypt” is dependent on the CSMS, in some applications where data confidentiality is not required, “Sign” might be sufficient.

4.9 User Authentication

User *Authentication* is achieved when the *Client* passes user credentials to the *Server* as specified via *Session Services* (described in Part 4). The *Server* can authenticate the user with these credentials.

The owner (user) of a *Session* can be changed using the *ActivateSession Service* in order to meet needs of the application.

User *Authentication* is not directly part of the *Publish-Subscribe* communication pattern but is used as part of the SKS associated with this communication pattern.

4.10 Application Authentication

OPC UA uses a concept conveying Application *Authentication* to allow applications that intend to communicate to identify each other. Each *OPC UA Application Instance* has a *Certificate* (*Application Instance Certificate*) assigned that is exchanged during *Secure Channel* establishment. The receiver of the *Certificate* checks whether it trusts the *Certificate* and based on this check it accepts or rejects the request or response *Message* from the sender. This trust check is accomplished using the concept of *TrustLists*. *TrustLists* are implemented as a *CertificateStore* designated by an administrator. An administrator determines if the *Certificate* is signed, validated and trustworthy before placing it in a *TrustList*. A TrustList also stores Certificate Authorities (CA). *TrustLists* that include CAs, also include *Certificate* Revocation Lists (CRLs). OPC UA makes use of these industry standard concepts as defined by other organizations.

In OPC UA, HTTPS can be used to create *Secure Channels*, however, these channels do not provide *Application Authentication*. If *Authentication* is required, it is based on user credentials (User Authentication see 4.9). More details on *Application Authentication* can be found in Part 4.

4.11 User Authorization

OPC UA provides user authorization based on the authenticated user (see 4.9). OPC UA *Applications* may determine in their own way what data is accessible and what operations are authorized or they may use Roles (see 4.12). *Profiles* exist to indicate the support of user credentials to restrict or control access to the address space.

4.12 Roles

OPC UA provides standard approach for implementing role based security. *Servers* may choose to implement none, part or all of mechanisms defined in Part 5. The OPC UA approach assigns *Permissions* to *Roles*. *Clients* are then granted *Roles* based on connection information. *Roles* might be restricted by User Authentication, Application Authentication, Security Modes, or Transports. The

assignment of *Roles* and restrictions is application specific. The interactions are illustrated in Figure 4.

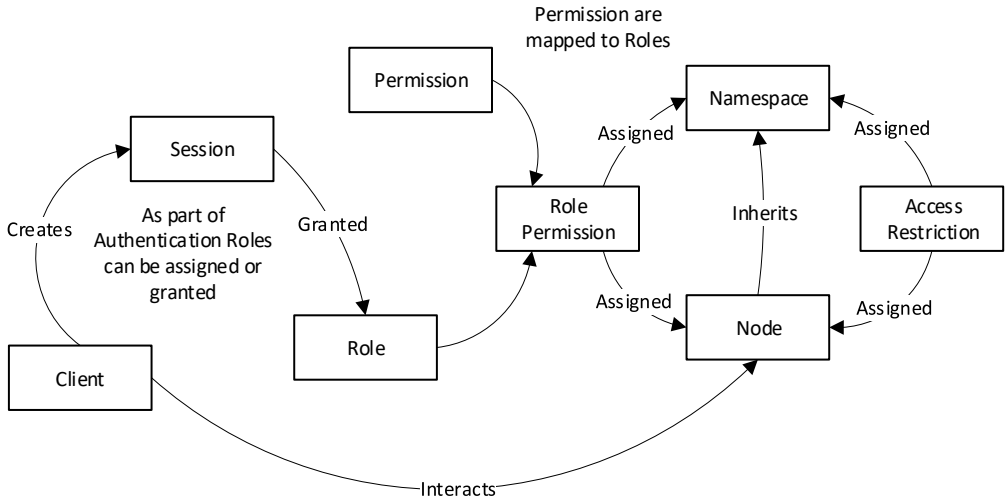

Figure 4 - Role overview

For additional description of roles see in in Part 5

4.13 OPC UA security related Services

The OPC UA Security Services are a group of abstract service definitions specified in Part 4 that are used for applying various security mechanisms to communication between OPC UA *Client*s and *Server*s.

The Discovery Service Set (specified in Part 4) defines services used by an OPC UA *Client* to obtain information about the security policies (see 4.6) and the *Certificate*s of specific OPC UA *Server*s.

The services of the *Secure Channel* Service Set (specified in Part 4) are used to establish a *Secure Channel* which is responsible for securing *Message*s sent between a *Client* and a *Server*. The challenge of the *Secure Channel* establishment is that it requires the *Client* and the *Server* to securely exchange cryptographic keys and secret information in an insecure environment, therefore a specific *Key Exchange Algorithm* (similar to SSL Handshake protocol defined in SSL/TLS) is applied by the communication participants.

The OPC UA *Client* retrieves the security policies and *Certificate*s of the OPC UA *Server* by the above mentioned discovery services. These *Certificates* contain the *Public Keys* of the OPC UA *Server*.

The OPC UA *Client* sends its *Public Key* in a *Certificate* and secret information with the OpenSecureChannel service *Message* to the *Server*. This *Message* is secured by applying *Asymmetric Encryption* with the *Server's Public Key* and by generating *Asymmetric Signatures* with the *Client*'s *Private Key.* However, the *Certificate* is sent unencrypted so that the receiver can use it to verify the *Asymmetric Signature*.

The *Server* decrypts the *Message* with its *Private Key* and verifies the *Asymmetric Signature* with the *Client*'s *Public Key*. The secret information of the OPC UA *Client* together with the secret information of the OPC UA *Server* is used to derive a set of cryptographic keys that are used for securing all further *Message*s. Furthermore, all other service *Message*s are secured with *Symmetric Encryption* and *Symmetric Signature*s instead of the asymmetric equivalents.

The *Server* sends its secret information in the service response to the *Client* so that the *Client* can derive the same set of cryptographic keys.

Since *Clients* and *Servers* have the same set of cryptographic keys they can communicate securely with each other.

These derived cryptographic keys are changed periodically so that attackers do not have unlimited time and unrestricted sequences of *Message*s to use to determine what the keys are.

For *PubSub* communications, the security related definitions are specified in Part 14 and provide a description of how to secure messages and also how to obtain the security keys required for message security.

The *Publisher* will utilize the keys provided to secure the message. It will encrypt the body of the message and sign the entire message. *Subscribers* will utilize the keys to decrypt and verify the signature of the messages.

To obtain the required keys, the *Publisher* or *Subscriber* make use of *Client* – *Server* communication. The keys may also be obtained using session-less method calls.

4.14 Auditing

4.14.1 General

*Client*s and *Servers* generate audit records of successful and unsuccessful connection attempts, results of security option negotiations, configuration changes, system changes, user interactions and *Session* rejections.

OPC UA provides support for security audit trails through two mechanisms.

First, it provides for traceability between *Client* and *Server* audit logs. The *Client* generates an audit log entry for an operation that includes a request. When the *Client* issues a service request, it generates an audit log entry and includes the local identifier of the log entry in the request sent to the *Server*. The *Server* logs requests that it receives and includes the *Client*'s entry id in its audit log entry. In this fashion, if a security-related problem is detected at the *Server*, the associated *Client* audit log entry can be located and examined. OPC UA does not require the audit entries to be written to disk, but it does require that they be available. OPC UA provides the capability for *Servers* to generate *Event Notifications* that report auditable *Events* to *Client*s capable of processing and logging them. See Part 4 for more details on how services in OPC UA are audited.

Second, OPC UA defines audit parameters to be included in audit records. This promotes consistency across audit logs and in *Audit Events*. Part 5 defines the data types for these parameters. Other information models may extend the audit definitions. Part 7 defines *Profile*s which include the ability to generate *Audit Events* and use these parameters, including the *Client* audit record id.

Because the audit logs are used to prove that the system is operating securely, the audit logs themselves should also be secured from unauthorized tampering. If someone without authorization were able to alter or delete log records, this could hide an actual or attempted security breach. Because there are many different ways to generate and store audit logs (e.g. files or database), the mechanisms to secure audit logs are outside the scope of this specification.

In addition, the information in an audit record may contain sensitive or private information, thus the ability to subscribe for *Audit Events* is restricted to appropriate users and/or applications. As an alternative, the fields with sensitive or private information can instead contain an error code indicating access denied for users that do not have appropriate rights.

The clauses 4.14.2, 4.14.3, 4.14.4 and 4.14.5 illustrate the behaviour of OPC UA *Servers* and *Client*s that support *Auditing*.

4.14.2 **Single Client and Server**

Figure 5 illustrates the simple case of a *Client* communicating with a *Server*.

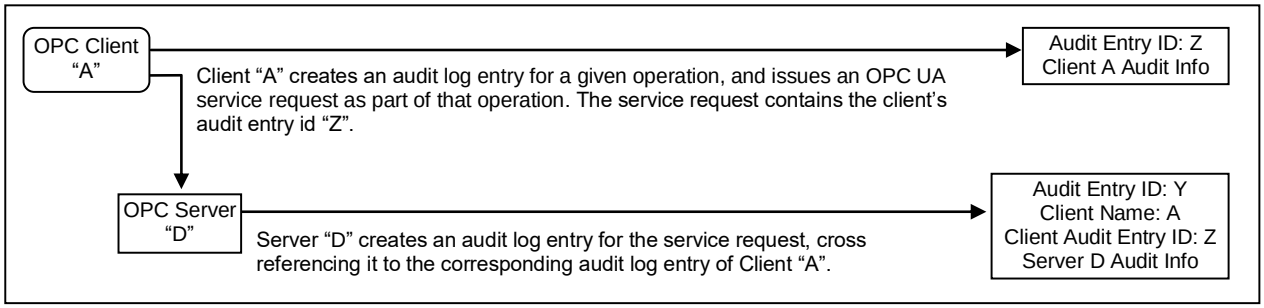

Figure 5 – Simple Servers

In this case, OPC Client “A” executes some auditable operation that includes the invocation of an OPC UA service in *Server* “D”. It writes its own audit log entry, and includes the identifier of that entry in the service request that it submits to the *Server*.

The *Server* receives the request and creates its own audit log entry for it. This entry is identified by its own audit id and contains its own *Auditing* information. It also includes the name of the *Client* that issued the service request and the *Client* audit entry id received in the request.

Using this information, an auditor can inspect the collection of log entries of the *Server* and relate them back to their associated *Client* entries.

4.14.3 **Aggregating Server**

Figure 6 illustrates the case of a *Client* accessing services from an aggregating *Server*. An aggregating *Server* is a *Server* that provides its services by accessing services of other OPC UA *Servers*, referred to as lower layer-*Servers*.

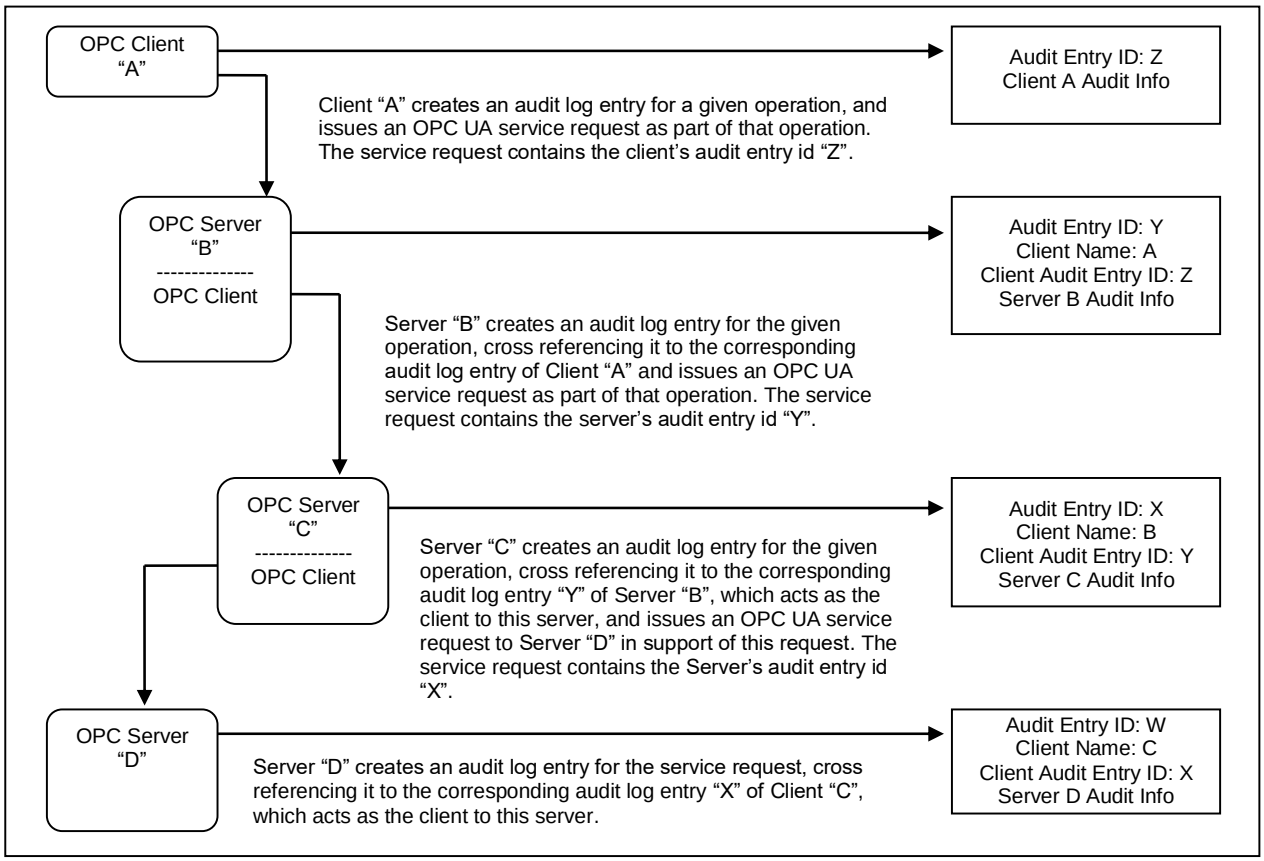

Figure 6 – Aggregating Servers

In this case, each of the *Servers* receives requests and creates its own audit log entry for them. Each entry is identified by its own audit id and contains its own *Auditing* information. It also includes the name of the *Client* that issued the service request and the *Client* audit entry id received in the request. The *Server* then passes the audit id of the entry it just created to the next *Server* in the chain.

Using this information, an auditor can inspect the *Server*'s log entries and relate them back to their associated *Client* entries.

In most cases, the *Servers* will only generate *Audit Events*, but these *Audit Events* will still contain the same information as the audit log records. In the case of aggregating *Servers*, a *Server* would also be required to subscribe for *Audit Events* from the *Servers* it is aggregating. In this manner, *Server* "B" would be able to provide all of the *Audit Events* to *Client* "A", including the *Events* generated by *Server* "C" and *Server* "D".

4.14.4 Aggregation through a non-auditing Server

Figure 7 illustrates the case of a *Client* accessing services from an aggregating *Server* that does not support *Auditing*.

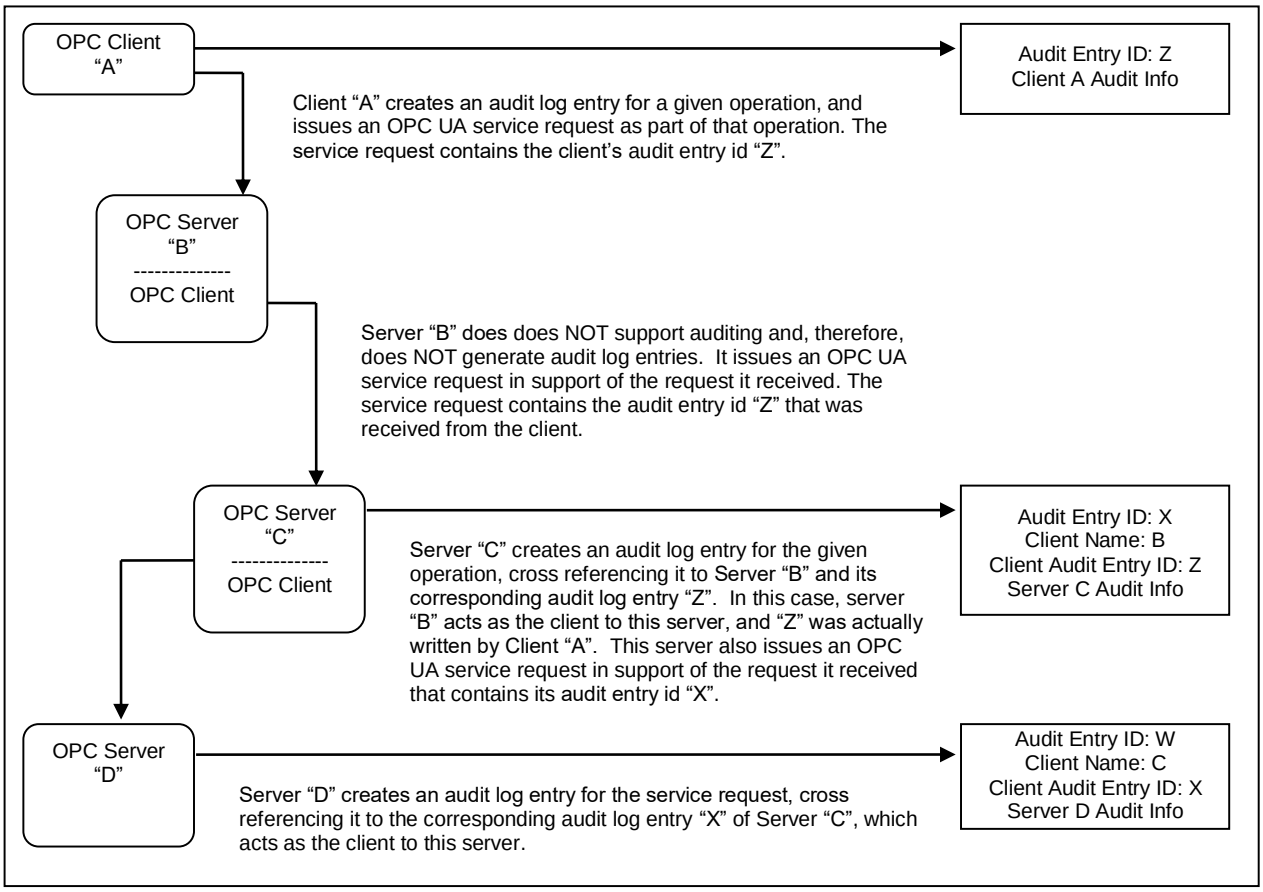

Figure 7 – Aggregation with a non-auditing Server

In this case, each of the *Servers* receives requests and creates their own audit log entry for them, with the exception of *Server* "B", which does not support *Auditing*. In this case, *Server* "B" passes the audit id it receives from its *Client* "A" to the next *Server*. This creates the required audit chain. *Server* "B" is not listed as supporting *Auditing*. In a case where a *Server* does not support writing audit entries, the entire system may be considered as not supporting *Auditing*.

In the case of an aggregating *Server* that does not support *Auditing*, the *Server* would still be required to subscribe for *Audit Events* from the *Servers* it is aggregating. In this manner, *Server* "B" would be able to provide all of the *Audit Events* to *Client* "A", including the event generated by *Server* "C" and *Server* "D", even though it did not generate an *Audit* event.

4.14.5 **Aggregating Server with service distribution**

Figure 8 illustrates the case of a *Client* that submits a service request to an aggregating *Server*, and the aggregating service supports that service by submitting multiple service requests to its underlying *Servers*.

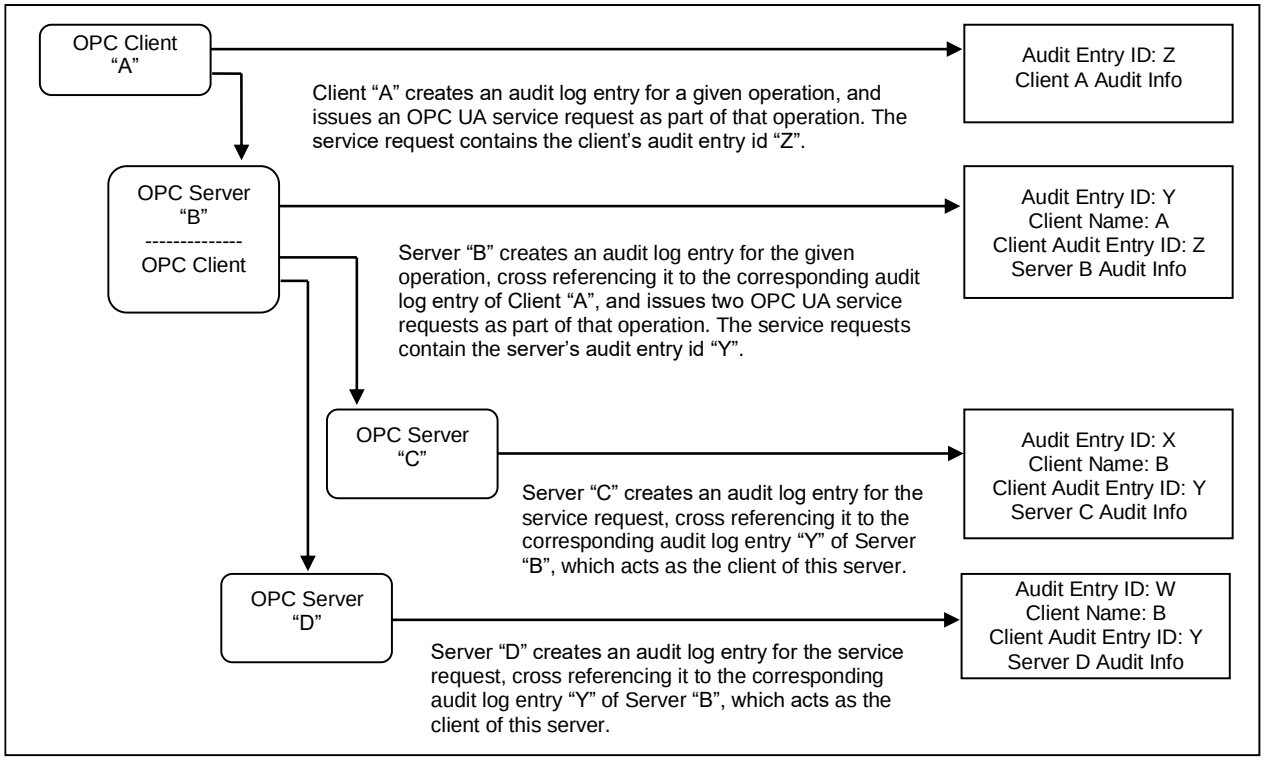

Figure 8 – Aggregate Server with service distribution

In the case of aggregating *Servers*, a *Server* would be required to subscribe for *Audit Events* from the *Servers* it is aggregating. In this manner, *Server* "B" would be able to provide all of the *Audit Events* to *Client* "A", including the event generated by *Server* "C" and *Server* "D".

5 Security reconciliation

5.1 Reconciliation of threats with OPC UA security mechanisms

5.1.1 Overview

The following sub-clauses reconcile the threats that were described in 4.3 against the OPC UA functions. Compared to the reconciliation with the objectives that will be given in 5.2, this is a more specific reconciliation that relates OPC UA security functions to specific threats. A summary of the reconciliation is available in Figure 9.

Attacks	Authentication	Authorization	Confidentiality	Integrity	Auditability	Availability	Non-Repudiation
Denial of Service						X	
Eaves Dropping	X	X	X				
Message Spoofing		X					
Message Alteration	X	X		X	X		X
Message Replay	X	X					
Malformed Messages						X	
Server Profiling	(X)	(X)	(X)	(X)	(X)	(X)	(X)
System Hijacking	X	X	X	X	X	X	X
Rogue Server	X	X	X		X	X	
Compromising User Credentials	X	X	X				
Repudiation							X

Figure 9 - Security Reconciliation Threats Summary

5.1.2 Denial of Service

5.1.2.1 Overview

See 4.3.2 for a description of this threat. For discussion purposes denial of service is broken into three major categories message flooding, resource exhaustion and application crashes.

5.1.2.2 Message flooding

OPC UA minimizes the loss of *Availability* caused by *Message* flooding by minimizing the amount of processing done with a *Message* before the *Message* is authenticated. This prevents an attacker from leveraging a small amount of effort to cause the legitimate *OPC UA Application* to spend a large amount of time responding, thus taking away processing resources from legitimate activities.

GetEndpoints (specified in Part 4) and OpenSecureChannel (specified in Part 4) are the only services that the *Server* handles before the *Client* is authenticated. The response to GetEndpoints is only a set of static information so the *Server* does not need to do much processing. The response to OpenSecureChannel consumes significant *Server* resources because of the signature and encryption processing. OPC UA has minimized this processing, but it cannot be eliminated.

The *Server* implementation could protect itself from floods of OpenSecureChannel *Message*s in two ways.

First, the *Server* could intentionally delay its processing of OpenSecureChannel requests once it receives more than some minimum number of bad OpenSecureChannel requests. It should also issue an alarm to alert plant personnel that an attack is underway that could be blocking new legitimate OpenSecureChannel calls.

Second, when an OpenSecureChannel request attempts to exceed the *Server*'s specified maximum number of concurrent channels the *Server* replies with an error response without performing the

signature and encryption processing. Certified OPC UA *Servers* are required to specify their maximum number of concurrent channels in their product documentation as specified in Part 7.

OPC UA user and *Client Authentication* reduce the risk of a legitimate *Client* being used to mount a flooding attack. See the reconciliation of *Authentication* in 5.2.3.

In *PubSub*, the *Subscriber* filters messages that it processes based on header information, allowing it to quickly discard any messages that do not conform to its required filter. In addition, the message signature is checked to eliminate any message that is well formed, but not from the desired *SecurityGroup*. *PubSub* can also be configured for unicast instead of multicast, which allows the network infrastructure to block multicast flooding attacks.

OPC UA *Auditing* functionality provides the site with evidence that can help the site discover that flooding attacks are being mounted and find ways to prevent similar future attacks (see 4.14). As a best practice, *Audit Events* should be monitored for excessive connection requests.

OPC UA relies upon the site *CSMS* to prevent attacks such as *Message* flooding at protocol layers and systems that support OPC UA.

5.1.2.3 Resource exhaustion

OPC UA user and *Client Authentication* reduce the risk of a legitimate *Client* being used to mount a resource exhaustion attack. Additionally, *Server Auditing* allows the detection of the *Client* if a resource exhaustion attack was carried out by a legitimate *Client*. *Servers* are also required to recycle *OpenSecureChannel* request that have not been completed (specified in Part 4), this will eliminate attacks from non-legitimate *Clients.* Resource exhaustion attacks do not apply to *PubSub* Systems, since no sessions or resources are allocated.

5.1.2.4 Application Crashes

OPC UA provides certification of OPC UA *Applications*. The lab testing and certification includes testing by injecting error and junk commands which might discover common faults. OPC Foundation stacks are also fuzz tested to ensure they are resilient to errors. Although a certified OPC UA *Application* does not guarantee fault free operation, the certified OPC UA *Application* is more likely to be resilient to application crashes caused by denial of service attacks.

5.1.3 Eavesdropping

See 4.3.3 for a description of this threat.

OPC UA provides encryption to protect against eavesdropping as described in 5.2.5.

5.1.4 Message spoofing

See 4.3.4 for a description of this threat.

As specified in Part 4 and Part 6, OPC UA counters *Message* spoofing threats by providing the ability to sign *Message*s. Additionally, *Message*s will always contain a valid *SessionId*, *SecureChannelId*, *RequestId* and *Timestamp* as well as the correct sequence number. OPC UA when operating as part of a *Session*, restricts user spoofing in the same manner since the user information is provided as part of the *Session* establishment. It is important that when a device starts up that the *SessionId* that is initially assigned to the first *Session* is a random number or a continuation of the last *Session* number used and is not always reset to 0 or a predictable number.

As specified in Part 14, OPC UA *PubSub* counters *Message* spoofing threats by providing the ability to sign messages. *Messages* can also contain a valid *PublisherId*, *DataSetClassId*, timestamp information, network message number and sequence number, which further restricts *Message* spoofing.

5.1.5 Message alteration

See 4.3.5 for a description of this threat.

OPC UA counters *Message* alteration by the signing of *Message*s that are specified in Part 4 and Part 14. If *Message*s are altered, checking the signature will reveal any changes and allow the

recipient to discard the *Message*. This check can also prevent unintentional *Message* alteration due to communication transport errors.

5.1.6 Message replay

See 4.3.6 for a description of this threat.

OPC UA uses *SessionIds*, *SecureChannelIds*, *Timestamps*, sequence numbers and *RequestIds* for every request and response *Message*. *Message*s are signed and cannot be changed without detection therefore it would be very hard to replay a *Message*, such that the *Message* would have a valid Session ID, *Secure Channel* ID, Timestamp, Sequence Numbers and Request ID. (All of which are specified in Part 4 and Part 6). The establishment of a secure channel / *Session* includes the same signature, timestamps and sequence number that are part of all messages and thus cannot be replayed.

OPC UA *PubSub* uses *PublishId*, *DataSetId*, and can use *Timestamps*, network message numbers, sequence numbers for published messages. *Messages* can be signed and cannot be changed without detection therefore it would be very hard to replay a message that has all of the fields enabled. It is worth noting that *PubSub* does allow the disabling of fields in a message. The disabling of the *Timestamp*, network message number and sequence number, would allow replay attacks. If a replay attack is of concern in a CSMS, then these field should be enabled.

5.1.7 Malformed Messages

See 4.3.7 for a description of this threat.

Implementations of OPC UA *Applications* counter threats of malformed *Message*s by checking that *Message*s have the proper form and that parameters of *Message*s are within their legal range. Invalid *Message*s are discarded. This is specified in Part 4, Part 6 and Part 14.

5.1.8 Server profiling

See 4.3.8 for a description of this threat.

OPC UA limits the amount of information that *Servers* provide to *Client*s that have not yet been identified. This information is the response to the GetEndpoints service specified in Part 4.

5.1.9 Session hijacking

See 4.3.9 for a description of this threat.

OPC UA counters *Session* hijacking by assigning a security context (i.e. *Secure Channel*) with each *Session* as specified in the CreateSession *Service* in Part 4. Hijacking a *Session* would thus first require compromising the security context.

5.1.10 Rogue Server or Publisher

See 4.3.10 and 4.3.11 for a description of this threat.

OPC UA *Client* applications counter the use of rogue *Servers* by validating *Server Application Instance Certificates*. There would still be the possibility that a rogue *Server* provides a *Certificate* from a certified OPC UA *Server*, but since it does not possess the appropriate *Private Key* (because this will never be distributed) to decrypt *Message*s secured with the correct *Public Key* the rogue *Server* would never be able to read and misuse secured data sent by a *Client*. Also, without the *Private Key* the *Server* would never be able to sign a response message to a *Client.*

OPC UA *Subscriber* applications counter the effect of a rogue *Publisher* by validating the signature on the published messages.

5.1.11 Compromising user credentials

See 4.3.11 for a description of this threat.

OPC UA protects user credentials sent over the network by encryption as described in 5.2.5.

OPC UA depends upon the site *CSMS* to protect against other attacks to gain user credentials, such as password guessing or social engineering.

5.1.12 **Repudiation**

See 4.3.13 for a description of this threat.

OPC UA *Client* and *Server* applications counter *Repudiation* by the signing of *Message*s that are specified in Part 4. A signed message indicates that the message originated from the owner of the private key. During *OpenSecureChannel* and *Session* establishment the communicating parties are clearly identified and confirmed. Lastly *Auditing* as described in Part 4 will track the information associated with the message.

5.2 Reconciliation of objectives with OPC UA security mechanisms

5.2.1 **Overview**

The following sub clauses reconcile the objectives that were described in 4.2 with the OPC UA functions. Compared to the reconciliation against the threats of 5.1, this reconciliation justifies the completeness of the OPC UA security architecture.

5.2.2 **Application Authentication**

OPC UA Applications support *Authentication* of the entities with which they are communicating. As specified in the *GetEndpoints* and *OpenSecureChannel* services in Part 4, OPC UA *Client* and *Server* applications identify and authenticate themselves with *X.509 v3 Certificate*s and associated private keys (see [X509]). Some choices of the communication stack require these *Certificate*s to represent the machine or user instead of the application.

For publish subscribe communications *Client Server* communications is required to obtain the shared keys from a *Security Key Service* (SKS). Although the application authentication is not directly between the Subscriber and the Publisher, the SKS ensures that only authenticated applications can obtain the keys used by the *Publisher* and *Subscriber*.

5.2.3 **User Authentication**

OPC UA Applications support *Authentication* of users by providing the necessary *Authentication* credentials to the other entities. As described in the ActivateSession service in Part 4, the OPC UA *Client* accepts a UserIdentityToken from the user and passes it to the OPC UA *Server*. The OPC UA *Server* authenticates the user token. *OPC UA Applications* accept tokens in any of the following forms: username/password, X.509 v3 *Certificate* (see [X509]), or JSON Web Token (JWT).

As specified in the CreateSession and ActivateSession *Services* in Part 4, if the UserIdentityToken is a *Certificate* then this token is validated with a challenge-response process. The *Server* provides a *Nonce* and signing algorithm as the challenge in its CreateSession response. The *Client* responds to the challenge by signing the *Server*'s *Nonce* and providing it as an argument in its subsequent ActivateSession call.

5.2.4 **Authorization**

OPC UA does not specify how user or *Client Authorization* is to be provided. *OPC UA Applications* that are part of a larger industrial automation product may manage *Authorizations* consistent with the *Authorization* management of that product. Identification and *Authentication* of users is specified in OPC UA so that *Client* and *Server* applications can recognize the user in order to determine the *Authorization* level of the user.

OPC UA *Servers* respond with the *Bad_UserAccessDenied* error code to indicate an *Authorization or Authentication* error as specified in the status codes defined in Part 4.

In *PubSub* interactions user *Authorization* can be used as part of the key distribution (SKS). This allows the *Publisher* and SKS to restrict access to specific users

5.2.5 **Confidentiality**

OPC UA uses *Symmetric* and *Asymmetric Encryption* to protect *Confidentiality as a security objective*. Thereby *Asymmetric Encryption* is used for key agreement and *Symmetric Encryption* for securing all other *Message*s sent between *OPC UA Application*s. Encryption mechanisms are specified in Part 6 and Part 14.

OPC UA relies upon the site *CSMS* to protect *Confidentiality* on the network and system infrastructure. OPC UA relies upon the *PKI* to manage keys used for *Asymmetric Encryption* which is then used to establish symmetric session keys.

5.2.6 **Integrity**

OPC UA uses *Symmetric* and *Asymmetric Signatures* to address *Integrity* as a security objective. The *Asymmetric Signatures* are used in the key agreement phase during the *Secure Channel* establishment. The *Symmetric Signatures* are applied to all other *Message*s including *PubSub* messages.

OPC UA relies upon the site *CSMS* to protect *Integrity* on the network and system infrastructure. OPC UA relies upon the *PKI* to manage keys used for *Asymmetric Signatures* which is then used to establish symmetric session keys.

5.2.7 **Auditability**

As specified in the UA *Auditing* description in Part 4, OPC UA supports *Audit* logging by providing traceability of activities through the log entries of the multiple *Client*s and *Servers* that initiate, forward, and handle the activity. OPC UA depends upon *OPC UA Application* products to provide an effective *Audit* logging scheme or an efficient manner of collecting the *Audit Event*s of all nodes. This scheme may be part of a larger industrial automation product of which the *OPC UA Applications* are a part.

5.2.8 **Availability**

OPC UA minimizes the impact of *Message* flooding as described in 5.1.2.

Some attacks on *Availability* involve opening more *Sessions* than a *Server* can handle thereby causing the *Server* to fail or operate poorly. *Servers* reject *Sessions* that exceed their specified maximum number. Other aspects of OPC UA such as OPC UA Secure Conversation can also affect availability and are discussed in Part 6

6 Implementation and deployment considerations

6.1 Overview

Clause 6 provides guidance to vendors that implement *OPC UA Application*s. Since many of the countermeasures required to address the threats described above fall outside the scope of the OPC UA specification, the advice in Clause 6 suggests how some of those countermeasures should be provided.

For each of the following areas, Clause 6 defines the problem space, identifies consequences if appropriate countermeasures are not implemented and recommends best practices.

6.2 Appropriate timeouts:

Timeouts, the time that the implementation waits (usually for an event such as *Message* arrival), play a very significant role in influencing the security of an implementation. Potential consequences include

- Denial of service: Denial of service conditions may exist when a *Client* does not reset a *Session*, if the timeouts are very large.
- Resource consumption: When a *Client* is idle for long periods of time, the *Server* keeps the *Client*'s buffered *Message* or information for that period, leading to resource exhaustion.

The implementer should use reasonable timeouts for each connection stage.

6.3 Strict Message processing

The specifications often specify the format of the correct *Message*s and are silent on what the implementation should do for *Message*s that deviate from the specification. Typically, the implementations continue to parse such packets, leading to vulnerabilities.

- The implementer should do strict checking of the *Message* format and should either drop the packets or send an error *Message* as described below.
    - Error handling uses the error code, defined in Part 4, which most precisely fits the condition and only when returning an error code is appropriate. Error codes can be used as an attack vector, thus their uses should be limited as described in Part 4. Part 4 describes that a single generic error is returned before and during the establishment of a secure channel. Once the secure channel has been established then appropriate specific error codes are returned.
    - Another attack vector that can be used is timing variations; this is minimized by the description in Part 4 that requires the closing of the socket for any errors when establishing a secure channel. Vendors should be careful in their implementation to ensure that all paths that result in the closure of the socket do not provide a timing hint indicating which failure path was encountered. This can be accomplished by having a random delay before closing the socket or before returning a generic error code.
- All arrays lengths, string lengths and recursion depth should be strictly enforced and processed.

6.4 Random number generation

Random numbers that meet security needs can be generated by suitable functions that are provided by cryptography libraries. Common random functions such as using rand() provided by the "C" standard library do not generate enough entropy. As an alternative, implementers could use the random number generators provided by the Microsoft Windows Crypto library (WinCrypt library) or by OpenSSL. Even the random functions provided in cryptography libraries require a source of entropy to initialize and the required entropy is not always available on embedded devices. PCs can use several individual pieces of information (hardware ids like CPU, Mac, addresses, USB devices, screen resolution, installed software ...) to generate entropy, but embedded devices are built completely

identically. Often only the time and maybe a MAC address is left for entropy. These sources of entropy can be guessed or discovered. This makes the embedded devices very vulnerable.

A common mistake is to generate cryptographic keys during the first boot. Thus even the time information is predictable (creation time is stored e.g. in a certificate). Some alternate solutions a vendor might want to consider:

- Add specific entropy generator hardware when designing embedded devices.
- Do not generate certificates on embedded devices. Use an external tool or the GDS to generate the certificate and load it onto the device. A problem could still remain for the symmetric keys, as these are normally not created directly during the boot phase; rather they are created when a client connects.
- Wait long enough until enough entropy information is available. Some operating systems provide hints when they have reached this point.
- For embedded systems without a good entropy source it may help to store the cryptographic pseudo-random number generator (CPRNG) state, so that it will not produce the same random numbers after every boot.

Vendor should ensure that cryptographic functions they use are initialized with suitable entropy and that the generated certificates are not created in a predictable manner.

6.5 Special and reserved packets

The implementation understands and correctly interprets any *Message* types that are reserved as special (such as broadcast and multicast addresses in IP specification). Failing to understand and interpret those special packets may lead to vulnerabilities.

6.6 Rate limiting and flow control

OPC UA does not provide rate control mechanisms, however an implementation can incorporate rate control.

6.7 Administrative access

OPC UA describes that certain functionality, such as the management of *CertificateStore*s, should be restricted to administrators. This Multi-part standard does not describe the details associated with administrative access. The nature of administrative access varies from platform to platform. Some platforms only have a single administrator. Other platforms provide multiple levels of administrative access such as backup administrator, network administrator, configuration administrator etc. The deployment site should make appropriate selections for administrator access and the implementer should allow for the configuration of appropriate administrator account access.

Administrative access restrictions include items such as configuration files for *Servers* and *Clients*. For example, configuration files might contain paths to certificate stores or exposed endpoints both of which if changed could cause major issues.

Administrative access should also be used to control *Audit Events*, see 4.14 for additional details.

6.8 Cryptographic Keys

Security Profiles defined in Part 7 describe required algorithms and required key lengths. Key length requirements may be specified as a range, i.e. 1024-2048. It is important that an OPC UA *Application* supports the entire range for its *Application Instance Certificate*. This allows an end user to generate a key (*Application Instance Certificate*) that meets their security requirements. This may extend the period of time for which the given Security profile can be used. For example, key lengths less than 2048 are already considered insecure, but if an end user generates certificates for the high end of the range (2048), the application might still be considered secure (depending on the other algorithms).

6.9 Alarm related guidance

OPC UA supports a robust *Alarm* and *Condition* information model which includes the ability to disable alarms, shelve alarms, and to generally manage alarms. Alarm processing and management is an important part of maintaining efficient control of a plant. From a security point of view it is important that this avenue be adequately protected, to ensure that a rogue agent does not create a dangerous or financial situation. OPC UA provides the tools required for this protection, but the implementer

needs to ensure that they are exercised correctly. All functions that allow changes to the running environment are able to generate *Audit Events* and are to be restricted to appropriate users.

The disabling of Alarms is one such function that should be restricted to personnel with appropriate access rights. Furthermore, any action that disables an alarm, whether it be initiated by personnel or some automated system, should generate an *Audit* Event indicating the action.

The shelving of alarms should follow similar guideline as the disabling of alarms with regard to access and *Auditing*, although it may be available to a wider range of users (operators, engineers). Also, the implementer should ensure that appropriate timeouts are configured for Alarm Shelving. These timeouts should ensure that an Alarm cannot be shelved for a period of time that could cause safety concerns.

Dialog *Events* could also be used to overload a *Client*. It would be a best practice for *Servers* that support dialogs to restrict the number of concurrent dialogs that could be active. Also, Dialogs should include some timeout period to ensure that they are not used to create a DOS. *Client* implementers should also ensure that any dialog processing cannot be used to overwhelm an operator. The maximum number of open dialogs should be restricted and dialogs should be able to be ignored (i.e. other processing should still be available).

6.10 Program access

OPC UA describes functionality that allows for programs to be executed as part of the OPC UA *Server*. These programs can be used to perform advanced control algorithms or other actions. The use of these actions should be restricted to personnel with appropriate access rights. Furthermore, the definition of Programs should be carefully monitored. It is recommended that statistics be maintained regarding the number of defined programs in addition to their execution frequency. This information is available to administrative personnel. In no case should an unlimited number of program executions be allowed.

6.11 Audit event management

The OPC UA specification describes *Audit Events* that are to be generated and the information that these *Audit Events* include as a minimum, however, the specification does not describe how these *Audit Events* are handled once they are generated. *Audit Events* can be subscribed to by multiple *Audit* tracking systems or logging systems. The OPC UA specification does not describe these systems. It is assumed that any number of vendor provided systems could provide this functionality. As a best practice whatever system is used to store and manage, *Audit Events* should ensure the following:

- That *Audit Events* are not tampered with once they are received.
- The *Subscription* for *Audit Events* should be via a *Secure Channel* to ensure they are not tampered with while in transition.
- For *Clients* that log audit events; it is recommended that the logged audit events be persisted in such a manner that the audit events can be authenticated and linked to the original transaction.

An *Audit* event management system could have additional requirements based on the site *CSMS.*

6.12 OAuth2, JWT and User roles

OAuth2 defines a standard for *Authorization Services* that produce JSON Web Tokens (JWT), also known as *Access Tokens*. These JWTs are passed as an *Issued Token* to an OPC UA *Server* which uses the signature contained in the JWT to validate the token. JWT can also provide information to the *Server* regarding the roles associated with the *Authenticated* user. The enforcement of the roles is the responsibility of the *Server.* Part 4, Part 5 and Part 6 describes OAuth2 and JWTs in more detail. Sites should ensure that they follow the best practices defined in the site *CSMS* for *OAuth2.*

6.13 HTTPs, SSL/TLS & Websockets

HTTPs defines a standard transport security. This transport security does not always ensure end to end security. Proxy servers or other intermediaries may exist. If end to end security is required then additional step such as a VPN should be taken.

If SSL/TLS communication is support, the keys used for TLS must be different then the keys for TCP communication. Reusing the keys introduces security issues. Only TLS 1.2 should be enabled, other versions of TLS have security issues and should not be enabled.

SLL version 2 has security issues and should be disabled. It is important that it is disabled for all applications on the machine not just for the UA application.

Websockets is just another protocol that is secured using HTTPS. If using Websockets all of the security guideline for HTTPs and TLS should be followed.

6.14 Reverse Connect

Reverse connect allows a *Server* to initiate the connection to a *Client* (open the socket sending a HEL message). This results in an additional security concern for the *Client*, in that the *Client* needs to validate that the connection is from an appropriate *Server* and not a denial of service attack. If the *Server* does not respond in a timely manner to the open *SecureChannel* request the *Client* should close the channel.

7 Unsecured Services

7.1 Overview

OPC UA provides a number of services that do not require security to access. These services require special consideration from a security point of view. These services provide capabilities that allow clients to discover servers and connect to them. The Discovery services are available as local services or global services and can be multicast.

7.2 Multi Cast Discovery

OPC UA can be configured to support discovery in multiple manners. One of the options is a multi-cast discovery. In this type of Discovery, *Servers* announce themselves on a subnet when they start. Application machines or an actual application can listen and build a list of the available servers.

Multicast DNS operations are insecure because of their very nature; they allow rogue servers to broadcast their presence or impersonate another host or server. Risks from Rogue *Servers* can be minimized if OPC UA security is enabled and all applications use certificate trust lists to control access. Also *Clients* should cache connection information, minimizing the lookup of *Server* information. However, even if you use UA security, multicast DNS should be disabled in environments where an attacker can easily access the network.

Applications (or discovery servers) are built to ensure that they cannot be overloaded or brought down by high broadcast rates on the multi-cast discovery channel or by too large a list of server applications.

7.3 Global Discovery Server Security

7.3.1 Overview

The Global Discovery Server (GDS) is a special OPC UA *Server* that provides Discovery services for a plant or entire system. In addition it can provide certificate management functionality (See Part 12)

There are multiple methods of accessing a GDS:

1) *Servers* can register with the *Discovery Server*
2) *Clients* can query the GDS for available *Servers*
3) *Clients* can pull certificates from the GDS
4) *Servers* can pull certificates from the GDS
5) The GDS can push certificates to a *Server*
6) The GDS can access other discovery *Servers* to build a list of available *Servers*.

Several types of threats need to be discussed with regard to the available access methods:

1. Threats where a rogue GDS is in a system.

2. Threats against the GDS, including the presence of rogue *Clients* or *Servers*

3. Threats against the certificate management functionality provided by a GDS.

7.3.2 **Rogue GDS**

The following guidelines are important to remember when dealing with a GDS:

- It is important that *Servers* register with the *Discovery Server* they are configured to register with and that *Servers* do not blindly register with a GDS that it has not been configured to register with. *Servers* have to be aware that a *Discovery Server* might be a rogue *Server*.

- A *Server* registers all endpoints that it provides, ensuring that the list provided by the *Discovery Server* and the *Server* match. This ensures that *Clients* can determine if the *Discovery Server* provided valid information.

- *Clients* should be aware of rogue *Discovery Servers* that might direct them to rogue *Servers*. *Clients* can use the SSL/TLS server certificate (if available) to verify that the *Discovery Server* is a *Server* that they trust and/or ensure that they trust any *Server* provided by the *Discovery Server*.

- As described in Part 4*, Clients* always verify that they trusts the *Server* certificate and that the EndpointUrl matches the HostNames specified in the certificate before it creates a *Session* with a *Server*. After it creates a *Session* it looks at the *EndpointDescriptions* returned by the *Server* and verifies that it used the best security possible and that the *Server's* Certificate matches the one that the *Client* used to connect. The *EndpointDescription* provided by the *Server* includes a relative *SecurityLevel* that is used to determine if the most secure endpoint was used.

7.3.3 **Threats against a GDS**

As described in Part 4, the *FindServersOnNetwork Service* can be used without security and is therefore vulnerable to denial of service (DOS) attacks. A *Discovery Server* should minimize the amount of processing required to send the response for this *Service*. This can be achieved by preparing the result in advance.

The GDS only accept *Server* registrations from *Servers* that are trusted or have appropriate administrative access rights. This will help ensure that a rogue *Server* does not become registered with a GDS.

7.3.4 **Certificate management threats**

A GDS, that also provides certificate management, supports User Access security as described in Part 12. This includes restricting all certificate management functionality to administrators. Furthermore, the list of *Clients* that are allowed to access management functionality may be limited.

Certificate management includes a provisioning phase and run time phase. The provisioning phase is when the GDS is providing initial certificate(s) to *Clients* or *Servers* that are just entering the system. The runtime phase is the day to day operation of system and includes providing updated CRLs, certificate renewals and updated trust lists.

The provisioning of systems is inherently not secure, but can be very useful in providing a greatly simplified deployment of a complex system. Provisioning in a GDS is not enabled by default, but requires an administrative action to enable. It is also recommended that the provisioning feature, when enabled, will only stay enabled for a limited time.

The runtime phase of GDS certificate operations can be performed in a very secure manner, since all *Servers* and *Clients* already have certificates to ensure a secure connection. For the push model of certificate management, the GDS establishes a secure channel using the highest security level available in the target *Server*. It does not provide updated CRLs, Certificates or TrustLists via an endpoint that has a lower security level than the security level of the updates. For example if a 4096 certificate is to be updated it cannot be updated using a 2048 channel, but a 2048 certificate can be

updated using a 4096 channel. If a new higher level certificate needs to be deployed, it is handled in the same manner as the provisioning of a new server.

8 Certificate management

8.1.1 Overview

OPC UA *Applications* typically have *Application Instance Certificates* to provide application level security. They are used for establishing a secure connection using *Asymmetric Cryptography.* These *Application Instance Certificates* are *Certificates* which are *X.509 v3 Certificates* and contain a list of data items that are defined in Part 4 and completely described in Part 6. These data items describe the *Application Instance* that the *Certificate* is assigned to.

The *Certificates* include a *Digital Signature* by the generator of the *Certificate*. This *Digital Signature* can be self-signed (The signature is generated by the *Private Key* associated with *X.509 v3 Certificate* that is the *Application Instance Certificate*) or can be signed by a *Certificate Authority* (The signature is generated by the *Private Key* associated the *X.509 v3 Certificate* of the CA). Both types of *Certificates* provide the same level of security and can be used in *Asymmetric Cryptography.* The *Signatures* can be generated using a variety of algorithms, where the algorithms provide different levels of security (128 bit, 256 bit, 512 bit ...). The algorithm that is required for signing a certificate is specified as part of the *Security Policy*. *Servers* and *Clients* should be able to support more than one certificate since more than one certificate may be required depending on the *Security Profiles* that are being supported.

Asymmetric Cryptography makes use of two keys – a *Private Key* and a *Public Key*. An OPC UA *Application* will have a list of trusted *Public Keys* that represent the applications it trusts. This list of trusted *Public Keys* is stored either in the Windows Registry or a file folder. It will also have a *Private Key* that corresponds to its *Application Instance Certificate*. The OPC UA *Application* can use a *Public Key*, from its list, to validate that the signature on a received connection request was generated by the corresponding *Private Key*. An application can also use the *Public Key* of the target application to encrypt data, which can only be decrypted using the *Private Key* of the target application.

8.1.2 Self signed certificate management

The major difference between CA signed and self-signed *Certificate* in an OPC UA installation is the effort required to deploy and maintain the *Certificates.* The choice of when to use a CA issued *Certificate* versus a self-signed *Certificate* depends on the installation and site requirements.

Figure 10 illustrates the work that is required to maintain the trust list for self-signed *Certificates*.

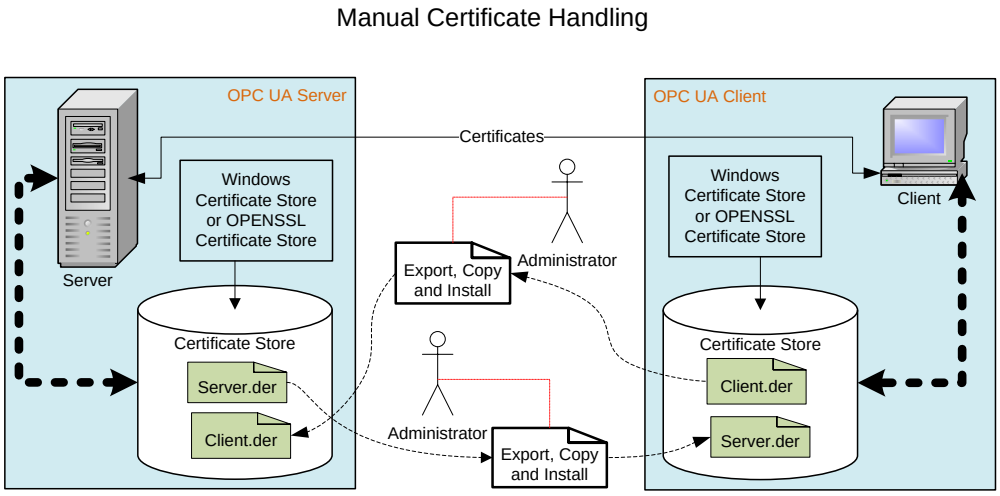

Figure 10 - Manual Certificate handling

An administrator would be required to copy the *Public Key* associated with all *Client* applications to all *Server* applications that they may need to communicate with. In addition, the administrator would

be required to copy the *Public Key* associated with all *Server* applications to all *Client* applications that may need to communicate with them. As the number of *Servers* and *Clients* grows, the administration effort can become too burdensome. In addition, a *Certificate* has a lifetime and will need to be replaced with an updated *Certificate* at some point in time. This will require that new *Private Keys* and *Public Keys* be generated and all of the *Public Keys* to be copied again. In very small installations, explicitly listing what *Clients* a *Server* trusts by installing the *Public Key* of the *Client Application Instance Certificate* in the Trusted *Certificate* store of the *Server* may be acceptable.

8.1.3 CA Signed Certificate management

In systems with multiple *Servers* and *Clients* the installation of *Public Keys* in Trust Lists can very quickly become cumbersome. In these instances, the use of a company specific CA can greatly simplify the installation/configuration issues. The CA can also provide additional benefits such as management of *Certificate* expiration and *Certificate Revocation Lists* (CRL). Figure 11 provides an illustration of this activity.

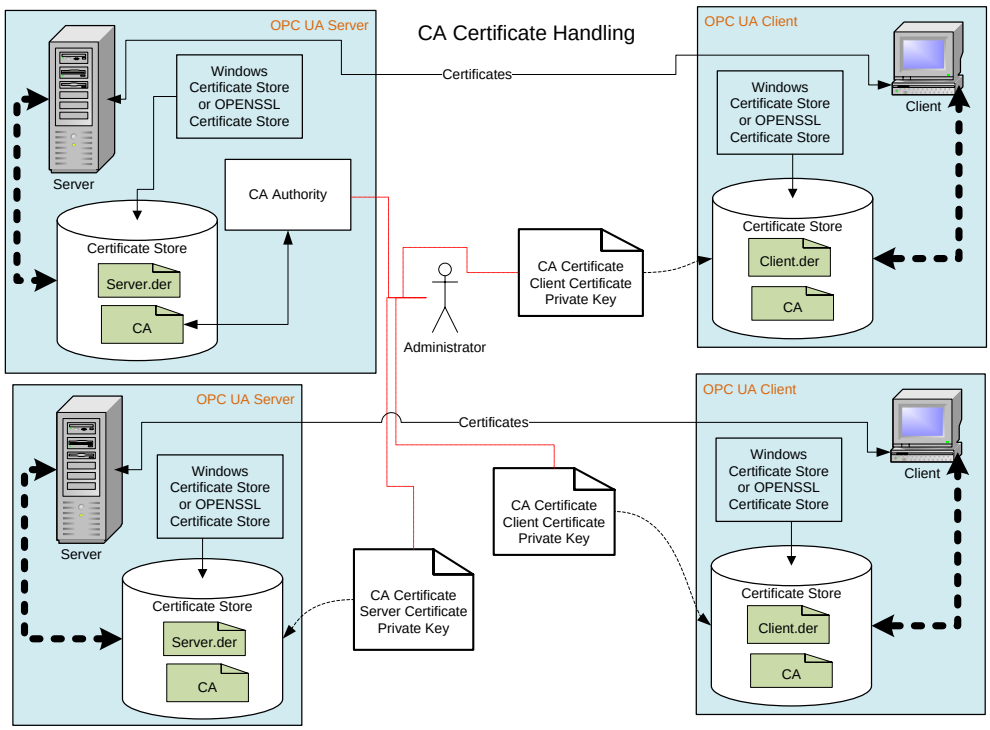

Figure 11 - CA Certificate handling

The administrator will need to generate a CA signed *Application Instance Certificate* for all *Clients* and *Servers* that are installed in a system, but he will only need to install the CA *Public Key* on all machines. When a *Certificate* expires and is replaced, the administrator will only need to replace the expired *Certificate* (*Public Keys* and *Private Keys*), there will be no need to copy a *Public Key* to any locations.

The company specific CA allows the company to control the issuing of *Certificates*. The use of a commercial CA (such as VeriSign) would not be recommended in most cases. An OPC UA Application typically is configured to trust only the other applications determined by the Company as trusted. If all *Certificates* issued by a commercial CA were to be trusted then the commercial CA would be controlling which applications are to be trusted, not the company.

Certificate management needs to be addressed by all application developers. Some applications may make use of *Certificate* management that is provided as part of a system wide infrastructure, others will generate self-signed *Certificates* as part of an installation. See Part 12 for additional details on system wide infrastructures for *Certificate* management.

8.1.4 GDS Certificate Management

8.1.4.1 Overview

In some systems, a *GlobalDiscoveryServer* with Certificate Management may be deployed. The *GlobalDiscoverServer* will either push certificates to *Clients* and *Servers* or allow *Servers* and *Clients* to pull certificates. The *GlobalDiscoveryServer* certificate management can manage all certificate deployments; this includes TrustLists, CAs and CRLs.

8.1.4.2 Developers Certificate management

From a developer point of view, it is a best practice, if your OPC UA *Application* supports *Certificates*, that it automatically provides a self-signed *Application Instance Certificate* on installation. In addition, the OPC UA *Application* is able to easily replace the self-signed *Application Instance Certificate* with a CA issued *Application Instance Certificate* or have the self-signed certificate signed by a CA. The configuration of a Trust List should also be easily accomplished. Typically, Trust Lists for *Public Keys* of Application Instances are kept in a separate list than those of a CA. Also, an OPC UA *Application* should be able to handle Certificate Revocation Lists (CRL). These are lists of *Public Keys* that are associated with a given CA that have been revoked. This allows a CA to remove a *Certificate* that it had signed from circulation. CRLs are provided by a CA and usually distributed in some automatic manner; see Part 12 for additional details.

From a security point of view, it is essential that the *Certificate* stores used to store *Private Keys* are protected and secured only allowing read/write access by an appropriate administrator and /or by the OPC UA *Application*. Trust lists, CRLs, and trusted CA lists are secured allowing only write access by an appropriate administrator and in the case of pull configuration by the application. Read access may be granted to other valid users, but the list of users allowed read access would be a site decision.

From an Installation point of view, it is a best practice that a standard tool to generate an *Application Instance Certificate* is provided. This tool could be one provided by an OPC UA SDK vendor or by the OPC Foundation. The standard tool ensures that the *Application Instance Certificates* that are generated include all of the required fields and settings. A particular OPC UA *Application* should be able to accept and install any valid *Application Instance Certificates* generated by external tools. The choice of the actual tool is site specific. Figure 9 provides an overview of some of the key points of Certificate handling.

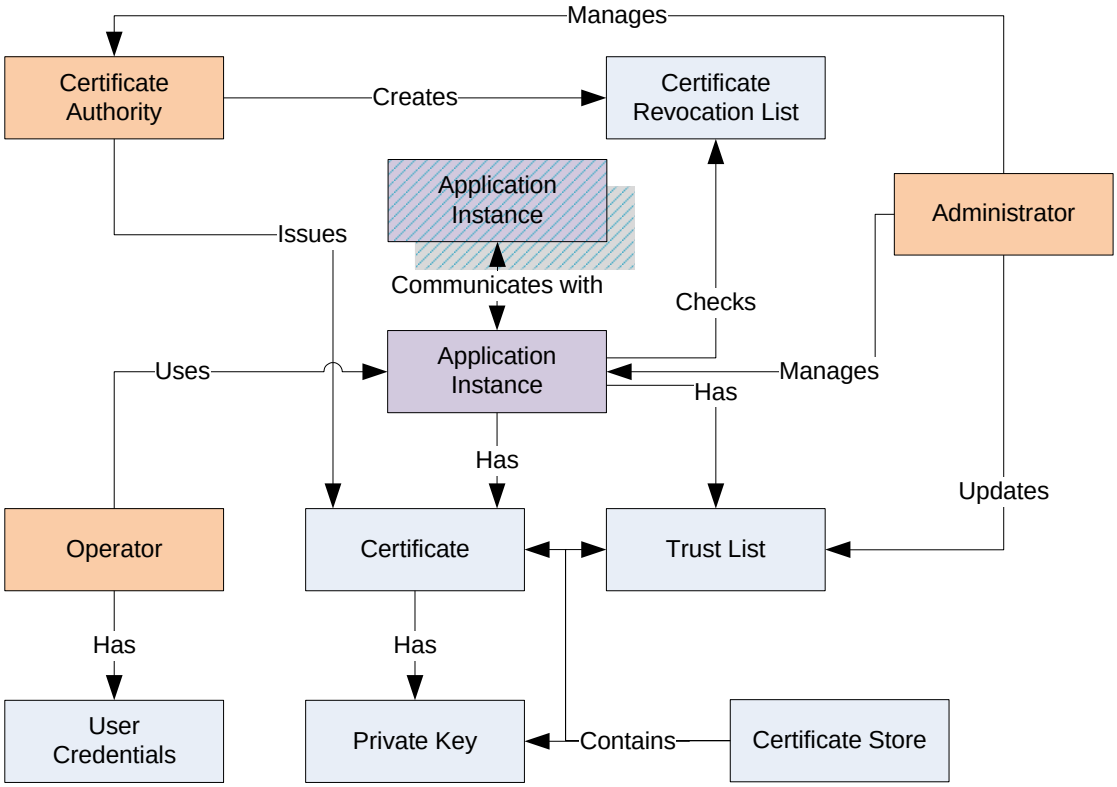

Figure 12 – Certificate handling

The following is a summary of these key points when a CA based, security required system is deployed:

Application Instance – An OPC UA *Application* installed on a single machine is called an Application Instance. Each instance has its own *Application Instance Certificate* which it uses to identify itself when connecting to other OPC UA *Applications* (the *Public Key* and *Private Key*). Each Application Instance has a globally unique URI which identifies it. The OPC UA *Application* will also check trust lists and CRL's to determine if access should be granted. The OPC UA *Application* will communicate using a secure channel established using *Asymmetric Cryptography* with other applications.

Administrator – The person or persons that administer the *Certificate* handling associated with a UA system and manage the security settings for Application Instances. This includes setting the contents of trust lists and managing any activities performed by a CA.

Operator – An Operator is person who uses the Application Instance. More than one Operator may exist for any given OPC UA *Application*. An Operator may have User Credentials which are used to determine access rights and to track activities within the Application Instance.

User Credential – A User Credential is a generic term for an electronic ID which identifies an Operator/User. It may be passed to a Server after the *Application Instance Certificate* is used to create a secure channel. It can be used to determine access rights and to track activities (auditing).

Certificate Authority (CA) – A *Certificate Authority* (CA) is an administrator or organization which is responsible for creating and managing *Certificates* (it is usually a partially automated software product). The *Certificate Authority* verifies that information placed in the *Application Instance Certificate* is correct and adds a *Digital Signature* to the *Certificate* that is used to verify that the information has not been changed. Each CA has its own *Certificate* which is used to create the *Digital Signatures*. A CA is also responsible for maintaining CRLs. In most cases it is a software package that an administrator periodically reviews or accesses, usually when the software package generates an alarm or notification that some review action is required.

Certificate – A *Certificate* is an electronic ID that can be held by an OPC UA *Application*. The ID includes information that identifies the holder, the issuer, and a unique key that is used to verify *Digital Signatures* created with the associated *Private Key*. The syntax of these *Certificates* conforms to the X.509 specification and as a result these *Certificates* are also called "*X.509 Certificates*".

Self-Signed Certificate – A self-signed *Certificate* is a *Certificate* which has no *Certificate Authority*. These *Certificates* can be created by anyone and can be used in situations where the administrators of UA Applications are able to verify the claims by reviewing the contents themselves. A system that uses only self-signed *Certificates* would not have CA or CRL.

Private Key – A *Private Key* is a secret number known only to the holder of a *Certificate*. This secret allows the holder to create *Digital Signatures* and decrypt data. If this secret is revealed to unauthorized parties then the associated *Certificate* can no longer be trusted or used. It is replaced or in the case of a CA generated *Certificate* it is revoked.

Trust List – A Trust List is a list of *Certificates* which are trusted by an Application Instance. When security is enabled, UA Applications reject connections from peers whose *Certificates* are not in the trusted list or if the *Certificate* is issued by a CA that is not in the Trust List.

Certificate Store – A Certificate Store is a place where *Certificates* and *Private Keys* can be stored on a file system. All Windows systems provide a registry based store called the Windows Certificate Store. All UA systems can also support a directory containing the *Certificates* stored in a file which is also called an OpenSSL Certificate Store. In all cases the Certificate Store needs to be secured, in that only administrators are allowed to write new entries. The security should follow the 'least privileged' principle, in that read or write access is only allowed to those who really need the data. This means that an administrator for example can store a *Private Key* but is not allowed to read them, and conversely an UA application can read such *Private Keys*, but cannot write them.

Revocation List – A Revocation List is a list of *Certificates* which have been revoked by a CA and are not be accepted by an Application Instance.

_____