A Practical Guide for CIP Security Device Developers

Michael Mann
Senior Systems Engineer
Pyramid Solutions

Ron Floyd
Engineering Manager
Pyramid Solutions

Jack Visoky
Product Security Architect
Rockwell Automation

Joakim Wiberg
Manager Technology and Platforms
HMS Industrial Networks

Presented at the ODVA
2017 Industry Conference & 18th Annual Meeting
February 21-23, 2017
Palm Harbor, Florida, USA

Abstract

CIP Security (Profile 1) adds transport layer security to the CIP protocol, and has a large, system-wide impact on CIP based products. As such, there are many considerations that product developers must take into account when designing CIP Security enabled products. Many of these issues can have potential security implications and as a result require careful thought. Although the ODVA CIP Security specification provides sufficient information for the implementation of this protocol, it is still beneficial to product developers to have some additional guidance at their disposal. The aim of this paper is to provide non-normative guidance around many of the important considerations that have an impact on CIP Security implementations. This paper does not seek to replace or replicate the information within the CIP Security specification, but rather provides additional guidance and information. Furthermore, as this paper is non-normative, the information described within is not necessary for compliance (unless it is also stated within the official CIP Security specification).

Keywords

CIP Security, Cyber Security, Certificates, Authentication, Integrity, TLS

Definition of Terms

Acronym/Term	Description
AES: Advanced Encryption Standard	Symmetric Encryption Algorithm designed to be efficient with both hardware and software. Supports 128 bit data blocks and key sizes of 128/192/256 bits.
BSD: Berkeley Source Distribution	Derivative works from original source are not required to be distributed under the original terms, nor is the owner required to make source code freely available.
Certificate Authority	A trusted entity that issues electronic documents that verifies a digital entity's identity on the internet.
Cipher Suite	Examples supported in CIP Security Specification: RSA, ECC, PSK, NULL.
CoCo: Connection Configuration Object	CIP defined object that may be used to configure a device to receive both secure and non-secure communication.
CVE: Common Vulnerability and Exposure	Publically available list of security threats with unique identifiers (CVE names, numbers and ID's). Database maintained by Mitre corporation and the National Cybersecurity FFRDC.
(D)TLS: Datagram Transport Layer Security	Based on the TLS protocol, (D)TLS provides communication security over UDP. Reference IETF RFC 6347.
Digital Certificate	A digital certificate is an electronic "passport" that allows a person, computer or organization to exchange information securely over the Internet using the public key infrastructure (PKI).
Digital Certificate, Self-Signed	Certificate signed by the same entity whose identity it certifies. (i.e. signed with its own private key.) Used when parties know each other and trust to protect key. Hardware based key storage not required.
Digital Certificate, Vendor	A self-signed certificate that adds product level public keys and secure key storage.

Digital Signature	The digital equivalent of handwritten signature or stamped seal to validate identity. It is a mathematical technique to validate the authenticity and integrity of a message or digital content. It also ensures the authenticity of the source such that service cannot be denied (see non-repudiation.) Digital signatures are based on public key infrastructure (see PKI.)
ECC: Elliptic Curve Cryptography	public key cryptography based on algebraic structure of elliptic curves.
Encryption, Asymmetric	Form of encryption using paired keys. One key is known to everyone, the public key and the other is kept secret, known as the private key. One key encrypts a message and the opposite key decrypts the message. RSA, ECC, Diffie Hellman
Encryption, Symmetric	Form of encryption using single key to both encrypt and decrypt data. Examples: AES, SHA
Entropy Source	true random number generator, often hardware generated for cryptography purposes. NIST SP 800-90
GCC: GNU compiler collection	
GPL: General Public License	Software whose source code is available at no cost for anyone to use for any purpose.
Hash Function	Any function that maps data of arbitrary size to fixed length data. A cryptographic hash function is designed to be one way such that it is infeasible to derive the original number. Keys in public key encryption are based on hash number.
HMAC: Hash-Function Message Authentication Code	Authentication code based on two inputs: the message and the key. The output is a code that cannot be used by attackers to derive the source. HMAC is an integral piece of TLS security architecture.
Message Authentication	Confirmation that the message came from the stated sender (its authenticity) and has not been changed in transit (its integrity). In CIP Security message authentication is achieved via TLS and the HMAC function.
MAC: Message Authentication Code	A message authentication code (MAC) is a short piece of information (e.g. code corresponding to a specific product serial number) used to authenticate a message.

Message Confidentiality	Assurance that the messages between two entities cannot be monitored by untrusted entities. Message confidentiality is achieved in CIP Security via the TLS Algorithm.
Message Integrity	Assurance that the message passed between two trusted entities has not been corrupted or altered. Message integrity is achieved in CIP Security via the TLS Algorithm.
Non-Repudiation	Repudiation is the rejection of an agreement. Nonrepudiation in cyber security refers to the ability to ensure that the party to a communication cannot deny the authenticity of a digital signature.
NULL	NULL Encryption cipher suite. No encryption used, cipher used for authentication communication only, often during debug/testing process.
PKI: Public Key Infrastructure	A public key infrastructure (PKI) is a set of roles, policies, and procedures needed to create, manage, distribute, use, store, and revoke digital certificates and manage public-key encryption. In cryptography, a PKI is an arrangement that binds public keys with respective user identities by means of a certificate authority (CA).
PSK: Pre Shared Key Encryption	A shared secret key that was previously shared between two parties used for encryption and decryption of data.
RSA	Public key asymmetric encryption algorithm. RSA acronym based on names of authors Rivest, Shamir and Adleman.
SHA: Secure Hash Algorithm	Hash function maps arbitrary data to data of a fixed size. The SHA family of algorithms designed by the NSA are used in creating digital signatures.
Spoofing	In network security, an attacker masquerades as a legitimate entity on the network in order to gain access to the entity's system or information.
SSL: Secure Socket Layer	Standard for establishing a secure link between two entities on a network. Transport Layer Security (TLS) standards have superseded SSL. However, the term “SSL” is often used to refer to either the earlier SSL protocol as well as the newer TLS protocol and libraries.

TCP: Transport Control Protocol	Provides connection management and guaranteed end to end delivery of data between two network devices. Transport Layer Security (TLS) uses TCP services. Reference IETF RFC 5246.
TLS: Transport Layer Security	Transport Layer Security (TLS) is the successor cryptographic protocol to Secure Socket Layer (SSL), that provides secure connections over a computer network. Both are often referred to as SSL. Reference IETF RFC 5246.
UDP: User Datagram Protocol	Connectionless data transmission protocol. D(TLS) uses UDP services
X.509	Standard for a Digital security certificate using the PKI to verify that the public key belongs to the entity contained within the certificate. (Standards Organization: United Nations ITU-T)

Introduction

Adding CIP Security to a new or existing project is a serious undertaking that involves several important decisions and considerations. It is important to understand the key points in these decisions, and ramifications that a particular path may have with regard to both CIP Security and the overall product.

Within this paper, several considerations are raised and discussed, along with potential options being offered where appropriate. Although it is not possible to exhaustively list all possible decisions and impacts that CIP Security would have to a product, this paper aims to provide a reasonable overview to help product developers in adding CIP Security support to a new or existing product.

Library considerations

The (D)TLS library is one of the core components in CIP Security as it provides a secure transport mechanism using the standard Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) protocols. TLS and DTLS make use of the IETF-standard, RFC 5246 and RFC 6347 respectively, protocols in order to provide a secure transport for EtherNet/IP traffic. The (D)TLS library sits between the TCP/IP stack and the application protocol (EtherNet/IP over TLS and DTLS).

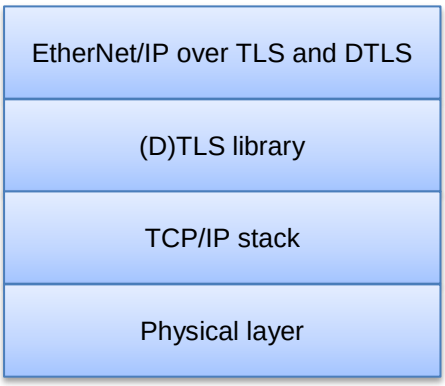

Figure 1 EtherNet/IP over TLS and DTLS Layering

The (D)TLS library is a large and complex piece of software and it is crucial that all parts implemented in this library are designed with security in mind. Even the smallest and simplest design flaws or bugs in the (D)TLS library might compromise the security the device and in the end the whole system. For those reasons it is not recommended to roll your own (D)TLS library. Instead it is recommended to obtain an existing and well known (D)TLS library.

There are many things, ranging from cost and license for the library to size and performance, to consider when evaluating and choosing a (D)TLS library. This section will touch some of the things that should be considered when choosing a (D)TLS library. Since there are many factors to consider and many of them are business related, it is not possible to give any specific recommendation. There are many (D)TLS libraries available from different vendors and sources all with their own respective pros and cons. Therefore, each vendor should select a (D)TLS library based on their needs.

In this paper four different (D)TLS libraries have been investigated. They have not been compared head to head but rather used as references giving directions and pointers regarding the different things looked at in the library consideration section. The libraries that have been of most focus are the ones targeting embedded systems running on a smaller microcontroller. The four libraries that have been looked at are:

- OpenSSL
- wolfSSL
- mbed TLS (formerly known as PolarSSL)
- MatrixSSL

Cost and license

There are both free and commercial (D)TLS libraries, some of the libraries are available under a dual-license model meaning there are both a free and open source version as well as a commercial license version that can be purchased.

One of the most well-known and used libraries, OpenSSL, is a free and open source library. Since OpenSSL is open source, anyone can view the code and for this reason security related issues with the library can possibly be discovered earlier. The dual-license libraries use the same idea, making the source code publically available so anyone can view it and thus possibly having security related issues found earlier.

The dual-license libraries are usually published under GPL and thus making the libraries free to use as along as the source code they're integrated in also are published publically. In most cases this is not an acceptable license for most companies building products using CIP Security. Instead the company behind the dual-license library provide a paid version of the library and in this case it comes under a different type of license model. The companies offer different types of the paid licenses, ranging from a per-unit cost to full buy-outs. The dual-license is attractive since it's possible to start out using the free version for testing, initial development, and prototyping and when getting closer to finalizing the product a paid license can be obtained making it possible to sell the product without having to disclose the full source code of the product.

Some of the free and open source libraries are available under different licenses that allow them to be used for free and without any more restrictions than maintaining a copyright notice in the written documentation. OpenSSL for example is distributed under a BSD like license.

Support

Choosing a commercial (D)TLS library generally provide some sort of professional support from the library vendor. And as all cases when purchasing software, the support of it might come directly from the vendor of the library or from the distributor. That of course can be a big difference especially in the case of (D)TLS libraries which are rather large and complicated pieces of software.

The non-commercial (D)TLS libraries generally have a rather large and active community behind them. And the support in those cases comes from the library developers and other users of the library. This doesn't necessarily mean that it's of less help than the support expected from a paid and commercial library. However, the amount of support and the time it would take to get help from the community behind the non-commercial libraries may vary between the libraries and also when the help is needed.

Since the of (D)TLS libraries can be configured in many different ways and also provide a lot of function calls it's important that they come with good documentation. The level of documentation differs a lot between libraries.

OpenSSL has a lot of well written documentation which is easy to access and use when already up and running. However, it does not provide a good introduction to get started and set up things, this on the other hand can be found at other places.

MatrixSSL, mbed TLS, and wolfSSL all have good and useful documentation on how to port, setup, and get started with the libraries. They also come with easy example applications for both clients and servers. The API documentation varies a lot between the libraries but are overall enough and easy to understand.

Reputation
That the (D)TLS library is well-known and widely used is important to provide assurance that it's possible to use the library as a building block in creating a secure product. OpenSSL is probably the most well-known library around and it's being used in Linux and Unix distributions, amongst others. It also has a good reputation of being stable and well tested.

MatrixSSL, mbed TLS, and wolfSSL have been around for many years and the companies behind the libraries have been successful selling the libraries. They all have references to well-known companies and projects using their libraries.

Vulnerability Management
Having a well-defined and working procedure for dealing with vulnerabilities is important for the makers of a (D)TLS library. When a vulnerability is discovered and reported the makers of the library must act in a timely manner to fix the vulnerability in the library.

The vulnerability management process should include procedures for how to deal with reported common known vulnerabilities, Common Vulnerability and Exposures or CVE for short. A CVE Identifier is a unique number that can be used over different security advisories by different vendors to refer to the same issue.

OpenSSL, mbed TLS, and wolfSSL all list the CVEs that affect certain versions of the library and in what versions they have been addressed. By doing this users of the library can see that the library makers actively update and correct issues and vulnerabilities in the library.

Another thing that's an important part to consider when it comes to a library's vulnerability management process is the ability to report possible issues. It is important that vulnerabilities that are discovered and how the makers of the library work with the reporter to find out and determine if the issue is real and a valid vulnerability for the library.

There should also be a way to subscribe to updates and changes in the library so potential vulnerability fixes are received quickly, instead of actively having to go out and check for updates at the library's website.

Footprint
Depending on the type of device that is implementing the library the memory footprint might be very important. In the smaller embedded devices that run EtherNet/IP today there might not be

enough memory, both non-volatile and RAM, to implement a (D)TLS library. Larger devices like connection originators, i.e. PLCs, generally have more memory and thus the memory footprint of the library is of less importance. Also, since the (D)TLS library is a large piece of software adding that to an existing product that doesn't support CIP Security over EtherNet/IP might be an issue. For that reason the footprint of the (D)TLS library is of high importance.

There are some (D)TLS libraries that are specifically designed for embedded devices. Those libraries generally consume less memory, both non-volatile memory and RAM, than libraries designed to run on a desktop computer. Also the libraries designed for embedded systems often can be configured to use its own heap. This can, in some cases, make the design easier if the existing design doesn't already have dynamic memory management. The libraries designed for desktop computers rely on existing dynamic memory system and system calls. However, the embedded libraries don't make use of this operating system infrastructure, and often can be configured to run in an environment with no operating system whatsoever.

Since embedded devices usually don't have a lot of memory it's important that the library is scalable so it can be configured to only include functionality required for CIP Security over EtherNet/IP. The (D)TLS libraries that are designed with the intention of being used in embedded systems generally provide an easy way to enable and disable functionality and thus allow them to be tailored and configured in the most effective way.

Libraries like OpenSSL that were designed to be used as a (D)TLS library for computers generally don't provide the developer with as many configuration options to fine-tune them to the lowest level. However, since those libraries generally are used in more capable and higher performant devices that might run Linux or similar, the memory footprint isn't likely an issue in those cases.

Capabilities

CIP Security over EtherNet/IP puts some requirements on the (D)TLS library. The key items that the (D)TLS library has to support in order to be able to use it for implementing CIP Security over EtherNet/IP are:

- TLS has to be at least version 1.2
- DTLS has to be at least version 1.2
- Cipher suite requirements as mandated by the specification
- Allow the use of pre-shared keys or X.509 certificates for endpoint authentication
- Allow use of either RSA or ECC public/private key pairs
- Provide data encryption (in addition to data integrity), or data integrity only (null encryption)

OpenSSL, mbed TLS, and wolfSSL all support the requirements listed above. However, MatrixSSL doesn't support cipher suites for data integrity only (NULL cipher suites). For this reason MatrixSSL can't be used as an alternative for CIP Security over EtherNet/IP.

Performance

In order to implement the TLS protocol the (D)TLS library needs to perform a number of supporting cryptography operations and message digest operations, such as the SHA-256 hash algorithm. Those operations are all computation and processing heavy. And in order to implement CIP Security over EtherNet/IP on an embedded device performance is extremely important.

The TLS protocol requires a lot off processing power. However, processing power might not be the first thing considered when choosing a microcontroller for an embedded system, so time might have to be spent on optimizing performance. It's been shown that it's possible to run CIP Security over EtherNet/IP on low end microcontrollers like Cortext-M3. But this generally requires that work is done to profile the system and analyze where the time is spent when performing the cryptography operations and message digest operations. The code where the most time is spent could potentially be optimized or placed in faster memory. Also some microcontrollers have

hardware accelerators for performing cryptography and message digest functions (see the section on hardware architecture for more information on this). The (D)TLS library can then be ported to use the hardware accelerators instead of code that executes on the micro controller. This can in many cases dramatically improve performance and at the same time decrease the memory footprint. Some (D)TLS libraries have porting functions to make the integration job easier.

(D)TLS libraries also make heavy use of a heap and a lot of the cryptography and message digest functions are performed against data on that heap. Thus it's important that the heap is placed in a memory with high bandwidth and latency against the microcontroller or CPU. Placing the heap in a specific memory location is naturally easier to do if the library makes use of a dedicated heap.

Non-scientific tests have been performed to compare the performance of mbed TLS on a microcontroller system, using Cortex-M3, and a system running an application processor, dual Cortext-A8. No efforts were made to optimize the code, i.e. the library was compiled out of the box using the standard configuration options. In both cases GCC was used to compile the code and the same optimization level was used. The application process system was running Linux and the embedded system ran a home grown RTOS. The tests showed that the raw processing power of the application processor did make a big difference in performance. On the application processor system, the initial TLS handshake took tenths of milliseconds compared to seconds on the microcontroller system. That said, it's possible to optimize performance on the microcontroller to achieve an initial TLS handshake in the range of 100 milliseconds.

Technology

The libraries reviewed in this section were all written in C and for that reason likely easy to integrate in the environment most EtherNet/IP products are developed in. Beyond the four (D)TLS libraries mentioned here there are many other libraries, some of them written in other languages. Based on the environment used to implement a certain product, research needs to be done to find a (D)TLS library suited for that specific environment.

Besides this, the smaller (D)TLS libraries that are designed to be used in embedded systems, like MatrixSSL, mbed TLS, and wolfSSL are all written in a way where it's possible to run them bare-metal, i.e. without an operating system underneath.

Depending on the runtime environment where the libraries are intended to be used, this is something that needs to be considered. If using an operating system that provides all of the APIs and libraries that OpenSSL expects, it would likely make it easy to integrate. If the product doesn't have an operating system or just a simple RTOS without standardized system calls and libraries, then it would likely make it easier to use the smaller libraries intended for embedded systems.

Summary of library considerations

The table below give a brief summary of the sections covered above related to the library considerations. The library consideration section and table will to help assist choosing a library. However since there are many parameters besides what is discussed in this paper that affect the choice of TLS library, each vendor should consider their needs and do a thorough investigation.

	OpenSSL	**wolfSSL**	**mbed TLS**	**MatrixSSL**
Cost and license	Free	Dual-license	Free	Dual-license
Support	Good	Good	Good	Good
Reputation	Good	Ok	Ok	Ok
Vulnerability Management	Good	Good	Good	Unknown
Footprint	High	Low	Low	Low
Capabilities	Good	Good	Good	Low
Performance	Unknown	Unknown	Unknown	Unknown
Technology	Good	Good	Good	Good

Table 1 TLS library summary

Key Management and Secure Identity

(D)TLS connections always start with at least one side having security credentials (which are either a PSK or certificate). The general model of CIP Security is for the user to provision a product with credentials that can be used to make and/or receive (D)TLS connections. However, this begs the question: how can security be applied to the connection that is used to provision the product with these credentials? A set of default credentials are necessary to bootstrap this secure connection. Using the default credentials, a secure connection is made to provision the device with its initial credentials (either a PSK or a certificate), as well as any other appropriate CIP Security configuration. Once this initial CIP Security configuration is completed, the default credentials are no longer used to create secure connections (unless the product is returned to a default out-of-box state).

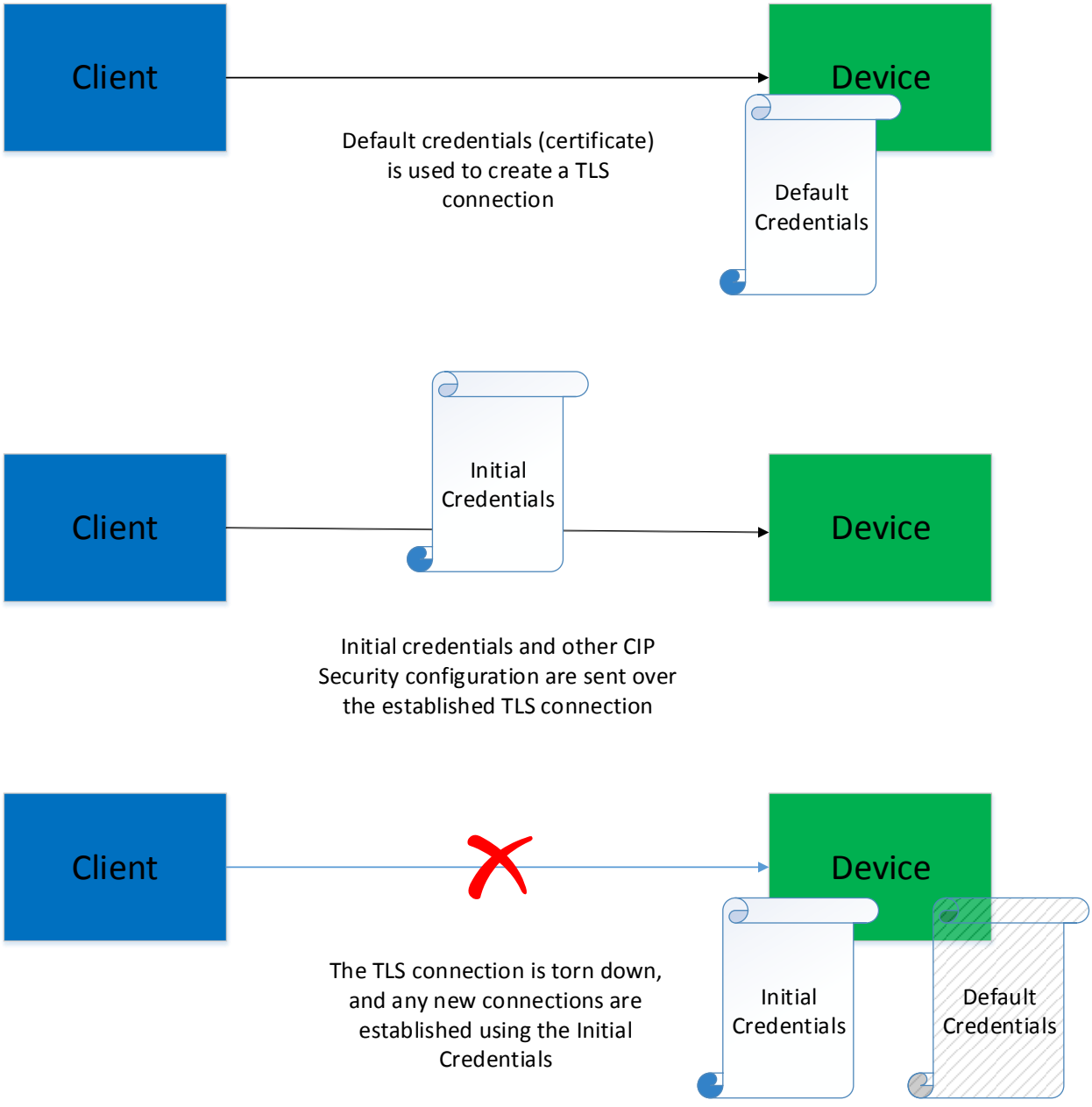

Figure 2 Commissioning credentials

There currently exist two options for the default identity. One is a Self-Signed Certificate, and the other is a Vendor Certificate. Each of these options has advantages and disadvantages which will be discussed in more detail. However, it is necessary first to understand the threats against initial commissioning.

There are two types of credentials that can be commissioned; a PSK or a certificate. The commissioning for each of these types of credentials has unique risks and threats. A certificate is public information, and a PSK is private. Therefore, when a PSK is commissioned there is a risk to the confidentiality of the communication, as an attacker who can discover the PSK value can both communicate on the system, as well as perform spoofing, data tampering, and information disclosure attacks on devices using the PSK. For certificate based credentials, there is no need for confidentiality during commissioning, as all of the information within a certificate is public (note that there may still be a marginal benefit to this confidential communications to prevent the attacker from knowing what type of configuration the system is using altogether). However, an attacker that can launch a data tampering or spoofing attack on the connection used to commission the initial credentials can certainly compromise the device being commissioned. The attacker would have the ability to tie the certificate to a key under his/her control, or to tamper with the device's credentials.

	Threat		
Credential Type	**Data Tampering**	**Information Disclosure**	**Spoofing**
PSK	Applies	Applies	Applies
Certificate	Applies	Doesn't Apply	Applies

Table 2 Credentials vs threats

With the threats enumerated, it is possible to briefly discuss the risk each poses. For data tampering, in the case of either a PSK or a certificate this would result in the endpoint's credentials being misconfigured. This might prevent the endpoint from communicating properly on the system, or might cause the endpoint to communicate with other endpoints that were not intended. Information disclosure represents no risk to the certificate, as all the information is public. However, in the case of configuring a PSK, learning the value of the PSK has the potential to compromise the confidentiality and integrity of all future communications. This applies not just for the endpoint in question, but for any other endpoints that are using the same PSK as credentials. Spoofing of the target allows an attacker to provision an endpoint under their control with the credentials intended for the original target endpoint.

The two possible categories for default credentials are a Self-Signed Certificate and a Vendor Certificate. Each of these options provide different guarantees for the level of risk. To truly analyze the risk mitigation provided by each of these options, one would need to understand actual implementation. However, a few general conclusions can be made. The mitigations each of these options provide, as well as other considerations, are detailed in the following sections.

Self-Signed Certificate

A Self-Signed Certificate is certainly the simpler of the two options, and in this simplicity lies the greatest benefit of the Self-Signed Certificate. No product PKI is necessary to be created and maintained, and no hardware-based secure key storage is necessary for the product. These are things that generally cannot be feasibly done in a field update. However, a product can relatively easily generate a Self-Signed Certificate via a field update. Furthermore, beyond the cost of storing data (on the order of a few kilobytes of memory), there is very little additional cost to implementing this beyond the general (D)TLS library.

Although the Self-Signed Certificate brings many benefits in the form of simplicity and low cost, it does have some drawbacks. The Self-Signed Certificate does little to protect against the spoofing case. A Self-Signed Certificate can be easily spoofed (as one can be generated by any attacker). Without out of band checking there is no guarantee of certificate authenticity, and therefore no guarantee that the connection is established with the intended device. However, if

the connection is indeed established using the intended device's Self-Signed Certificate then that connection does provide some security benefits. Assuming a cipher suite is chosen which includes confidentiality, then this would be an effective mitigation against information disclosure (as well as data tampering). Note that this is predicated on the successful establishment of a TLS connection using the intended Self-Signed Certificate. Put another way, the large weakness here is with the initial connection establishment, as an attacker could replace the intended Self-Signed Certificate with one under his/her control, in which case all security benefits on that connection are lost.

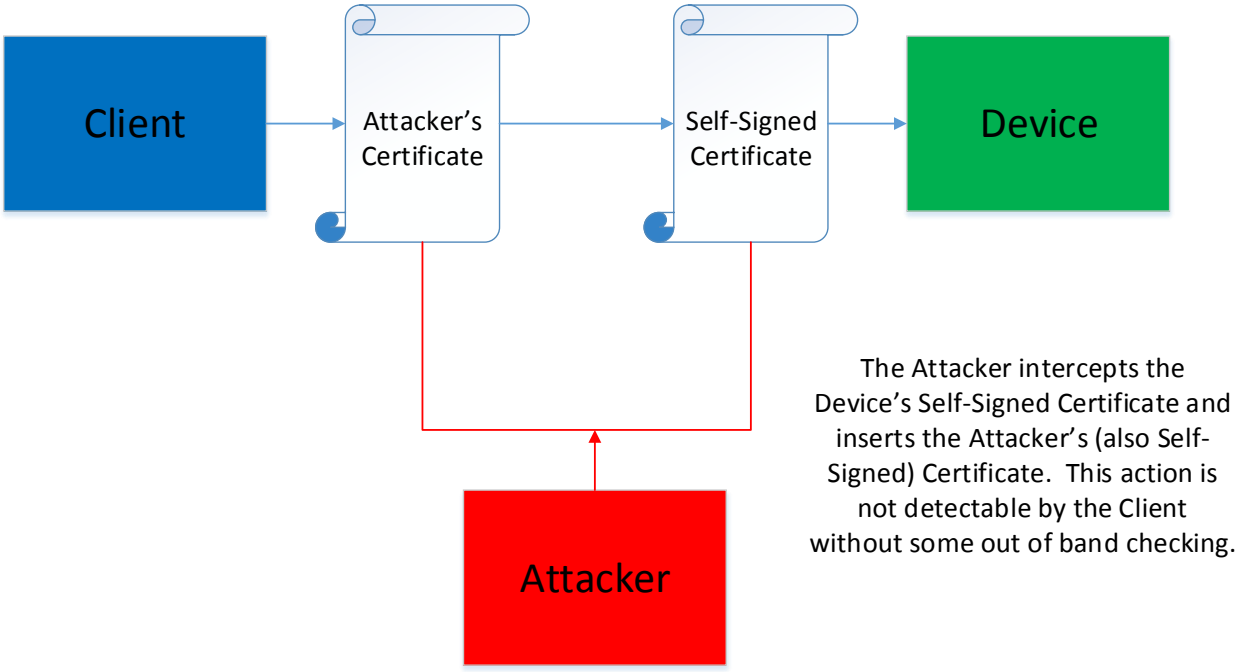

Figure 3 Certificate interception

Vendor Certificates

In contrast to a Self-Signed Certificate, the Vendor Certificate adds some level of complexity to the product implementing it. The two main areas of complexity introduced by Vendor Certificates are a product level PKI and secure key storage. Strictly speaking, neither of these is absolutely necessary for the implementation of Vendor Certificates. However, these aspects help significantly to realize the benefits of Vendor Certificates. A product level PKI is necessary for providing the signing services that sign the Vendor Certificates. There are many considerations for this, such as scale, availability, security, etc. Different products/organizations will have different needs in this area, therefore it is not possible to describe a product level PKI that would be appropriate for all use cases. However, in general setting up a PKI in a moderate to large organization is not trivial, in terms of cost, effort, and complexity. One important area of consideration is around the protection of the signing keys; if the signing key is compromised then the PKI essentially loses its value. Therefore, threat modeling and risk assessment activities should be undertaken to guide the level of protection necessary.

The other aspect of Vendor Certificates that merits discussion is the secure key storage. Whereas the product level PKI was around the keys and services for signing Vendor Certificates, the secure key storage is around the protection of each product's private key (which corresponds to the public key present in the Vendor Certificate). Protection of this key is important because the ability to use it allows an attacker to impersonate the product. Again, there are a wide range of solutions that can be applied here, from simple data obfuscation to robust hardware

mechanisms. What solution is implemented depends on many factors that are outside the scope of this document (but also should undergo threat modeling and risk assessment activities to drive the decision). In summary, implementing Vendor Certificates will certainly add cost and complexity to a product.

In many ways Vendor Certificates are susceptible to the same risks and threats as a Self-Signed Certificate. However, Vendor Certificates do offer a clear advantage in this area; which is the fact that their authenticity can be verified by anyone with the proper verification key. That is, the spoofing attack described with the Self-Signed Certificates changes. If the client verifies the authenticity of the device's Vendor Certificate, then it is no longer possible for this spoofing attack to occur. However, this benefit comes with some caveats:

1. The client must know *a priori* the public key used to verify Vendor Certificates. This means the client may need to maintain a list of several keys from several different vendors. These keys are all public information, so it is more a matter of building this knowledge into the client.
2. Compromising any Vendor Certificate breaks this scheme. If an attacker is able to compromise any Vendor Certificate of a trusted vendor, then it can be used to spoof the device's valid Vendor Certificate. That is, this scheme is only as good as the weakest Vendor Certificate trusted by the client. This implies that any vendors participating should implement a robust PKI as well as robust secure key storage, as both of these are likely compromise points.

Note that similar to the Self-Signed Certificate, once a Vendor Certificate has been used to establish a TLS session, then that session will benefit from all of the normal TLS protection mechanisms.

Although somewhat outside the scope of CIP Security, it is useful to note that there can be other benefits of Vendor Certificates. These certificates allow for authenticity checks on a given product and can be used to prevent cloned products (as a clone would presumably not have access to the PKI needed to create a valid Vendor Certificate). Furthermore, Vendor Certificates can be used to provide authenticity and non-repudiation of data produced by a given product (via cryptographic signing). Depending on the product's use cases this might be a useful feature to include, and could help to justify the cost of implementing Vendor Certificates.

Vendor Certificates and Self-Signed Certificates are both viable options for default credentials. Vendor Certificates do provide some additional security benefits, especially in terms of the added difficulty of spoofing the certificate used to establish an initial TLS connection, yet at the cost of increased complexity. Another important area of discussion revolves around a system that uses both Vendor Certificates and Self-Signed Certificates. In this case the "weakest link" of the system in the Self-Signed Certificates; therefore, the benefits of the Vendor Certificates are vastly reduced or even lost. As the system will need to accept a Self-Signed Certificate, then that is the lowest acceptable security level, and therefore can be used to launch spoofing attacks as described. Despite this, Vendor Certificates can bring additional benefits outside of CIP Security, and should still be considered for usage. Any decision on which implementation is used should be made through careful consideration of requirements, as well as threats and risks on a given product. The information here can and should be used as a basis to start the discussion around the costs and benefits to each solution, but ultimately vendors must decide what makes the most sense for their given products.

Connection Origination

An EtherNet/IP network will be more exposed to attack when non-secure devices are allowed to exist on the same network as secure devices. However, it is also recognized that the rate of adoption of CIP Security amongst device suppliers will vary over time. Further, end users may choose to continue with certain non-secure legacy devices after secure scanners are installed.

In Chapter 1 of CIP Security (Volume 8) it specifies the following: Devices that support CIP Security must still be able to interoperate with devices that do not support CIP Security, on the same network. It should be a matter of end user configuration to allow or disallow such a mix of devices on the network. When mixing devices with secure and non-secure communications, it is the end user's responsibility to manage the device and network configuration appropriately. The user may need to provide additional controls such as firewalls or physical security means.

In the hybrid world where non-secure devices will coexist on a network, the Originator of communications (Scanner) must "know" the security related communication types of its targets. Therefore, it will be incumbent upon the scanner device vendor to offer a mechanism to differentiate both secure and non-secure communications with devices on the EtherNet/IP network. This section of the paper is intended for scanner device implementers.

As a matter of course, vendors will develop configuration tools to accommodate the complexities of CIP security configuration. Using vendor specific configuration tool may be the preferred mechanism to identify connection types.

The CIP defined Connection Configuration Object (CoCo) provides a standardized method to create, configure and control CIP connections. To specify a "security" connection, include the (D)TLS port in the connection path attribute (6) of a CoCo instance. For example, an "unsecure" connection would have a connection path of "192.168.10.10", while a "secure" connection would have a connection path of "192.168.10.10:2221".

Debugging/Testing

Understanding TLS connection problems can sometimes be difficult, especially when it's not clear what messages are actually being sent and received. However, since CIP Security over EtherNet/IP uses an identical application layer and just minor deviations in the communication compared to EtherNet/IP, the messages communicated should be known. For the same reason it's unlikely that there will be any larger issues with the application layer data communicated. And in the cases where there are issues with the EtherNet/IP application data or communications it's better, if possible, to debug this without running EtherNet/IP on top of TLS.

When running EtherNet/IP the traffic can easily be captured and decoded with Wireshark using the correct infrastructure devices. When running CIP Security over EtherNet/IP it's still possible to capture and see the traffic in Wireshark, but naturally it's not possible decode the traffic and see the exact contents since it might be encrypted. There are however some things that should be done to help and ease debugging issues when running CIP Security over EtherNet/IP. One option is only use NULL encryption cipher suites, and thus only use authentication-only communication. Volume 8 defines three (D)TLS certificate cipher suites, one each for RAS, ECC, and PSK with NULL encryption. When using one of those cipher suites the data won't be encrypted, thus the data can be decoded using Wireshark.

In the cases when it's not possible to change the cipher suite, i.e. in a real installation, it's still possible to decode the actual traffic. This is accomplished by providing Wireshark with the private key, and naturally if this is in a real installation this isn't possible for security reasons. Also, it's worth mentioning that Wireshark cannot decrypt Diffie Helleman cipher suites. These are cipher suites with DH in their name; among the cipher suites that Volume 8 defines only three are non Diffie Helleman cipher suites.

The most likely issues that needs to be debugged are the initial communication and the TLS handshake. There are many different things that initially can go with the TLS handshake when starting a new development and porting the (D)TLS library for the first time. The TLS handshake can easily be captured and analyzed using Wireshark since it is not encrypted. One tool that's handy with doing initial tests and debugging is OpenSSL. OpenSSL comes with a command line client, this command line client can be used to perform just the TLS handshake. This is accomplish by issuing something like:

```
$ openssl s_client -connect <host ip>:2221
```

This command in conjunction with Wireshark can be really useful during initial development. It's also possible to provide the OpenSSL command line client with real certificates and keys doing something like:

```
$ openssl s_client -connect <host ip>:2221 –cert certandkey.pem –key certandkey.pem
```

Doing this it is possible to perform the full TLS handshake and test out that the certificate and key handling works correctly. This is useful when testing and verifying the CIP Security object and their interaction to the (D)TLS library.

The OpenSSL command line tool can also be used to test and debug other things. It provides options to test protocol support making it possible to verify that for example only TLS 1.2 is supported. There are also options to test out server-side cipher suite support. This is useful for testing out and verifying that the (D)TLS library has been correctly configured and setup to support the required cipher suites.

Another tool that is useful to verify the protocol support and implemented cipher suites in the server is nmap. This command can be used to list all cipher suites and supported protocols. This is accomplished with the following command line:

```
$ nmap --script ssl-enum-ciphers -p 2221 <host ip>
```

Performance Considerations

Adding CIP Security to a product is associated with a cost regarding high performance requirements on the processing unit of the product. Before actually implementing and testing CIP Security it's impossible to tell if an existing product can handle the performance degradation that TLS adds to CIP Security over EtherNet/IP. Running CIP Security over EtherNet/IP can be done on almost any processing unit but on smaller low end 8-bit microprocessors the product would likely end up being far too slow to be considered usable. However, many products (probably the majority) that have been developed in the last several years are built using 32-bit microprocessors. These are also most likely to be the products in which CIP Security is supported. Those devices are most likely capable of handling the performance degradation that comes with TLS and CIP Security over EtherNet/IP.

It's impossible to provide any rule of thumb if a certain processing unit will be capable of handling the addition of CIP Security. The reason being that there are many factors that vary and influence the overall performance requirements. Some of those factors are: the compiler being used and how well it can optimize the code, the performance and the bandwidth between the processing unit and the memory, how well the existing TCP/IP and EtherNet/IP stack perform, how well the TLS library used performs on the platform being used, if the processing unit has hardware accelerators for the cryptographic primitives, and if the compiler and TLS library are capable of making use of those. Besides this there are other factors that may affect the overall performance such as, which development tools and which specific platform were used. Many of those items can be overcome and worked around to make the overall product perform better.

There are some specific things that are of interest when considering the performance, the connection startup and the data flow during the connection. These are things that can be optimized in different ways and may need work to create a usable and well working product.

During the connection startup the TLS handshake takes place. This is when the two endpoints in the communication negotiate the details of which encryption algorithm and cryptographic keys to use before the first byte of data is transmitted. Also during this time the two endpoints are authenticated which is

done using public-key cryptography. This is a computationally heavy process that requires a lot of processing power from both sides in the communication.

During the data flow, i.e. primarily class 0/1 communication, two things impact the data latency: bulk encryption and message integrity. For the bulk data encryption this can be disabled by the end user by selecting one of the NULL encryption cipher suites, this however then provides no confidentiality but in many applications that's an acceptable tradeoff. For the message integrity part, which is a hash function, it can in many cases be optimized. The optimization can be done either in C or using an assembly-language optimized implementation.

In many processing units there are dedicated cryptographic hardware accelerators. Those hardware accelerators can provide functionality to assist the calculation of the different cryptographic primitives used for TLS. The hardware assisted support varies a lot and has to be looked at closely when choosing a processing unit for a new product. Some processing units provide hardware acceleration for all the cryptographic primitives, such as hash-functions, symmetric key algorithms, and public key algorithms – both Elliptic Curve and RSA. In those cases it's possible to offload the main processing unit a lot and achieve near line speed cryptography for both bulk data encryption and message integrity. Using hardware assisted cryptography naturally comes with a higher per unit cost. But considering the importance of CIP Security this is probably a good thing to consider when designing new products. Many silicon vendors offer pin compatible processing units with and without hardware assisted cryptography engines, thus it's possible to add the hardware assisted option later on and release this as a new updated product with higher and better capabilities. This without having to redesign the hardware, but just mounting a different part during manufacturing. This is discussed further in the next section, "Hardware Architecture Considerations".

Hardware Architecture Considerations

For CIP Security there are three main areas in which hardware may be particularly beneficial:

- Hardware that provides secure key storage
- Hardware that provides cryptographic acceleration
- Hardware that provides entropy generation

These hardware components are not required for CIP Security, but it is likely that many products would benefit from this additional hardware. Each of these is discussed in more detail within this section. Note that a single piece of hardware may perform more than one function; these functions would not necessarily need to be implemented as three separate hardware solutions. No matter what hardware is selected, there are some issues that need to be addressed regardless of the functionality included in the hardware:

- Trust boundaries: Some hardware is within a processor, some is on a printed circuit board (PCB), and some is easily removable (as in on a USB stick). Which is chosen depends on use case as well as the boundary of trust.
- Performance: Depending on the type of countermeasures employed and the underlying technology, hardware based secure key storage can reduce overall system performance. It is important to understand if the performance is acceptable for the given system.
- Capabilities: The key storage hardware may only support a limited number of algorithms and/or keys. The CIP Security spec limits what must be supported, but it might be desirable to have other algorithms for both future usage and for other uses within a product.
- Cost: Including extra hardware on a product is certainly not free, an organization will need to decide what cost they are willing to pay to have the extra capabilities given by secure storage hardware.

- Contention: More than one part of the system may need to use a given hardware resource. Mechanisms for dealing with this are an important consideration, whether they be hardware-based, software-based, or some combination thereof.

Beyond these general issues, there are considerations specific to each hardware function:

Secure Key Storage Hardware

As discussed in the section on secure identity, if a product implements a Vendor Certificate then secure key storage is necessary to protect the private key associated with this certificate. Although software based options are available for this, hardware generally provides a more robust protection. Hardware based key storage is useful beyond just protecting the key associated with the vendor certificate. Keys associated with the certificate sent to a device by a user (for CIP Security) can also be protected by hardware based secure key storage. Best practices denote that the key is generated by the device and the private portion of the key never leaves the device. Hardware based secure key storage allows a device to achieve this for the keys associated with (D)TLS communications. Furthermore, many devices have other uses for protecting data, of course hardware based secure key storage may be used to protect other data beyond that associated with (D)TLS communications.

Cryptographic Acceleration Hardware

CIP Security involves a significant amount of cryptographic operations on communications packets, which of course results in a performance impact. However, a product that includes specialized hardware for cryptographic acceleration can reduce or even eliminate the burden of these cryptographic operations on the main processor. There are two essential categories of cryptographic operations that are needed in CIP Security. One is the public key, or asymmetric operations. These are generally done during connection handshaking; examples include RSA, ECC, Diffie Hellman, etc. The other are symmetric operations, which are generally during the lifetime of the connection (once the handshaking is complete). Examples include AES, SHA, etc. Depending on the needs of the device, hardware can be used to assist either or both of these operations. Note that in general, the asymmetric operations take longer time and place a higher burden on the system. However, as these are mainly done as part of the connection handshake their impact is limited. The symmetric operations are generally faster, but occur much more often as there is at least one operation done on every packet that is sent or received. Throughput and latency targets are of course product specific, but hardware can be used to help achieve these targets. The following flowchart can aid in the decision-making process around what type of cryptographic accelerators to use:

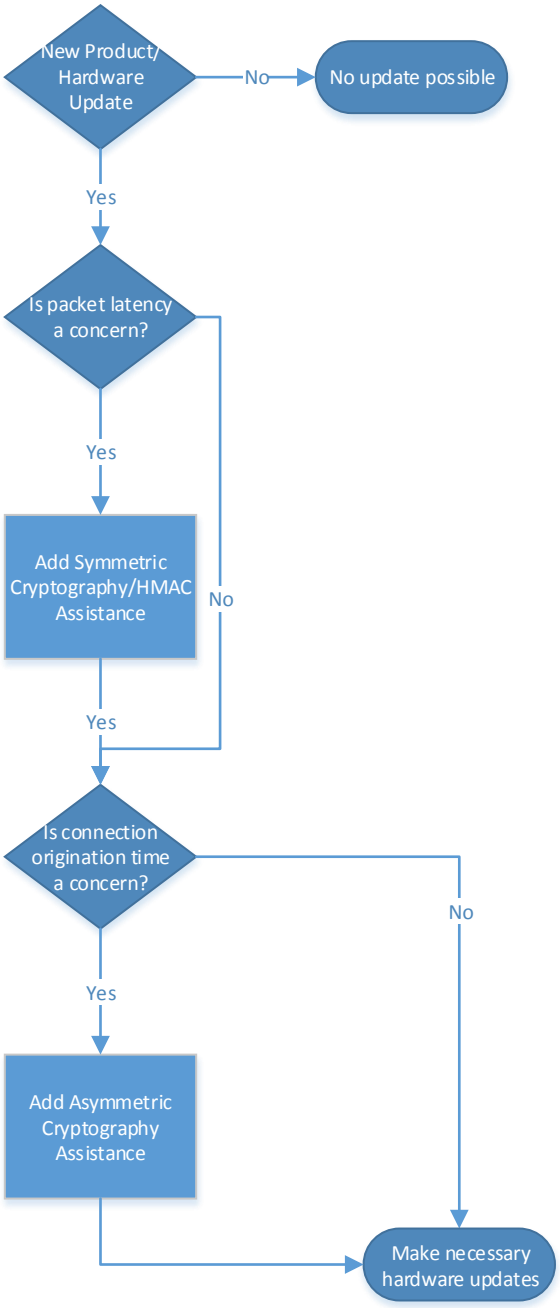

Figure 4 Hardware cryptography decision flow-chart

Entropy Generation Hardware

Another important area for hardware assistance is that of entropy generation. If possible, using a hardware based source for randomness is generally the best option. As such, when considering what hardware to include in a product a hardware based random number generator should certainly be kept in mind. More discussion on this is given below on the Entropy Sources section.

Entropy Sources

True Random Number Generators

Generation and management of cryptographically strong entropy is essential for the security of (D)TLS sessions. The CIP Security specification mentions that hardware based entropy sources, or True Random Number Generators, are to be preferred over any software based sources. Therefore, it is ideal if a dedicated piece of hardware for entropy generation can be placed on the product. Even more ideal is for that hardware to include countermeasure protections such that the entropy source is well protected, even inthe case of other parts of the system being compromised by an attacker. Keys generated by this hardware should stay within the hardware for the key lifetime. Furthermore, the hardware should be designed such that it is compliant with well-known standards for entropy generation (such as NIST SP 800-90). Although not generally necessary, more than one True Random Number Generator (TRNG) within a product may provide further security advantages through redundant mechanisms and diversity of sources, albeit at greater product cost.

Lack of Specialized Hardware

The above describes the ideal entropy generation support for CIP Security. However, it is recognized that this may not be feasible for all products. Some products may need to gather entropy from sources other than a True Random Number Generator. If this is the case it is still possible to gather entropy on a product (although maybe not at the same quality as a True Random Number Generator). This section will provide some general guidelines for gathering entropy on such a product.

- **Use Physical Sources** – All products run (at some point) on some sort of hardware. Hardware is necessarily a physical entity, subject to laws and constraints of physics. As such, there is a degree of randomness and entropy to hardware. Inspect the system for hardware which may exhibit some form of randomness. Likely areas of this include oscillator drift, RAM decay, and various monitoring/diagnostic circuitry (for example a temperature monitor). Generally, the higher the resolution these sources are, the more randomness they will exhibit (for example, a temperature monitor that reports in degrees Celsius to more than one decimal point).
- **Combine Sources** – The more physical sources that can be combined generally yields better entropy, unless a second source degrades the entropy of the other more high quality sources. This also provides a diversity of input; even if an attacker compromises one entropy source to behave in a predictable manner the other sources can still provide entropy such that the product functions properly and robustly. Furthermore, some sources may be slower to generate high quality entropy than others. With more than one source used, this allows entropy to be available without being subject to the limitations of the slowest source.
- **Test Sources** – It is important to run tests that give a measure of the quality of the entropy. NIST SP 800-90 B provides extensive tests for entropy sources. All sources should be tested to determine if there are enough suitable sources for the product. Of course what constitutes a suitable source is dependent on the product's security posture, although general guidance is given within the NIST publication.

Cipher suites

One integral part of the TLS handshake is in which the client provides the server with a list of its supported cipher suites. The server selects one of the offered cipher suites that will be used during this connection. If no acceptable cipher suites are presented to the server, the TLS handshake will fail and the connection will be closed. Once the cipher suite has been agreed on and the connection has been established, the cipher suite insures the privacy, authentication, and integrity for the data that passes between the client and server over the (D)TLS session (connection).

The cipher suite is what determines the level of security for the (D)TLS session. Some cipher suites even provide weak security and should not be used. For this reason it's essential that the correct cipher suites are used and it's of the highest importance to be careful in the cipher suites offered and accepted. The cipher suites defined by CIP Security for EtherNet/IP should be considered reasonable in terms of providing security.

Since the cipher suite determines the level of strength in the cryptography algorithms used, they also highly impact the performance of the data flow on the (D)TLS session. Thus higher cryptographic strength picked by using a different cipher suite might degrade the packets per second a device is capable of producing and consuming.

For authentication and key exchange the cipher suite defines the asymmetric algorithm used. From the cipher suites defined by CIP Security for EtherNet/IP this could be RSA or Elliptic Curve. RSA is generally more used and widely deployed then Elliptic Curve. Elliptic Curve on the other hand offer the same or better security with smaller key sizes. As a consequence of this the performance using Elliptic Curve can be higher.

The bulk data encryption in CIP Security for EtherNet/IP, when enabled, relies on AES. AES is a standard for encryption and it contains a lot of variations. Though this standard is what's adopted by the US government and now used worldwide. What's defined for CIP Security is considered providing reasonable security for confidentiality.

The data integrity is provided by HMAC and in the case for CIP Security for EtherNet/IP SHA-1 or SHA-2. Both SHA-1 and SHA-2 are accepted by NIST and considered providing enough security.

Table 3 is a summary of different cipher suites used by CIP Security for EtherNet/IP.

Cipher Suite	**Description**
TLS_RSA_WITH_NULL_SHA256	RSA for key exchange; null encryption; SHA256 for message integrity. Encryption is not provided.
TLS_RSA_WITH_AES_128_CBC_SHA256	RSA for key exchange. AES 128 for message encryption, SHA256 for message integrity.
TLS_RSA_WITH_AES_256_CBC_SHA256	RSA for key exchange. AES 256 for message encryption, SHA256 for message integrity.
TLS_ECDHE_ECDSA_WITH_NULL_SHA	ECDHE_ECDSA for key exchange; null encryption; SHA1 for message integrity. Encryption is not provided.
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	ECDHE_ECDSA for key exchange. AES 128 for message encryption, SHA256 for message integrity.
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	ECDHE_ECDSA for key exchange. AES 256 for message encryption, SHA256 for message integrity.

TLS_ECDHE_PSK_WITH_NULL_SHA256	ECDHE in conjunction with PSK for key exchange; null encryption; SHA256 for message integrity. Encryption is not provided.
TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA256	ECDHE in conjunction with PSK for key exchange. AES 128 for message encryption, SHA256 for message integrity.

Table 3 Example of cipher suites

The following table (included from NIST SP 800-57: Recommendation for Key Management) gives general guidance on the strength of cryptographic keys based on their key length. Obviously longer keys provide better security, yet what algorithm is used is also relevant to the overall strength.

Security Strength	Symmetric key algorithms	FFC (e.g., DSA, D-H)	IFC (e.g., RSA)	ECC (e.g., ECDSA)
≤ 80	2TDEA[21]	$L = 1024$ $N = 160$	$k = 1024$	$f = 160\text{-}223$
112	3TDEA	$L = 2048$ $N = 224$	$k = 2048$	$f = 224\text{-}255$
128	AES-128	$L = 3072$ $N = 256$	$k = 3072$	$f = 256\text{-}383$
192	AES-192	$L = 7680$ $N = 384$	$k = 7680$	$f = 384\text{-}511$
256	AES-256	$L = 15360$ $N = 512$	$k = 15360$	$f = 512+$

Figure 5 NIST Cryptography Strengths

Considerations for System Time

As a part of a X.509 v3 certificate there's a field named Validity which contains two dates, named notBefore and notAfter. The validity period for a certificate is the time from notBefore through notAfter, inclusive. This period is set when the certificate is generated and outside of this period of time, the certificate is invalid.

In many of the control applications and systems controlled by EtherNet/IP there's no need for a global system time. Generally the controller's notion of time is sufficient, and there's no need for a wider notion of time in the slave nodes. For this reason, most of the embedded systems on which EtherNet/IP products are build today have no notion of the global system time or a real time clock (RTC). Even though a CIP interface to IEEE 1588, Precision Clock Synchronization Protocol for Networked Measurement and Control Systems, is defined only niche EtherNet/IP products implement this for use in special applications. We can expect that the number of EtherNet/IP products implementing IEEE 1588 will increase over time and thus also the number of devices that will have a global notion of time.

To make use of the validity in a X.509 v3 certificate, the EtherNet/IP device verifying the peer needs to be aware of the system time. Since the majority of EtherNet/IP devices don't support this today there's an attribute in the EtherNet/IP Security Object that allows the user to enable and disable the certificate

expiration check. Furthermore, this attribute is by default set to disabled, meaning that the peer's certificate expiration shall not be checked.

An EtherNet/IP device that implements IEEE 1588 or NTP to obtain the system time can enable the option to allow for the certificate expiration check. That the certificate's validity is checked is preferable to making sure that old and expired certificates are used.

It should be noted that IEEE 1588 doesn't include any form of secure authentication mechanisms. Although recent versions of NTP provide for the possibility of authentication, in practice that's not used. Most systems trust unauthenticated NTP replies to set the system clock. This mean that a Man-In-The-Middle attacker can control a device's clock, and by doing so violate the security properties for TLS.

Lately, initiatives have been done to develop secure alternatives to NTP. Google is funding a project developing a secure time protocol called roughtime. The protocol aims to provide a rough time synchronization in a secure way. The "rough" time synchronization means that it's not providing a precise and perfect time synchronization, rather it provides a synchronization within 10 seconds of the correct time, which is more than enough for use with security. Roughtime is a simple and light weight protocol that potentially could become the open secure alternative to NTP.

In the future when there's a secure and widespread alternative to NTP, which possibly could be roughtime, CIP Security for EtherNet/IP should probably standardize on that. And at some point, we recommend that as a suggested supported protocol for all devices implementing CIP Security for EtherNet/IP.

References

[1] RFC5246, Transport Layer Security (TLS) Protocol Version 1.2, Aug 2008
[2] RFC6347, Datagram Transport Layer Security Version 1.2, Jan 2012
[3] ODVA, Inc. The CIP Networks Library, Volume 8: CIP Security™, PUB00299
[4] NIST, SP 800-57. Recommendation for Key Management, Part 1, Rev 4