Dartmouth Computer Science Technical Report TR2008-628

Key Management for Secure Power SCADA

Manya Sleeper
manya.k.sleeper@alum.dartmouth.org

Senior Honors Thesis

June 2008

This research was sponsored in part by the NSF, under grant CNS-0524695. The views and conclusions do not necessarily reflect the views of the sponsors

Key Management for Secure Power SCADA

By: Manya Sleeper
Advisor: Sean Smith
manya.k.sleeper@alum.dartmouth.org

Abstract: This thesis proposes a key management protocol for secure power SCADA systems that seeks to take advantage of the full security capacity of a given network by allowing devices to use public key cryptography for key management if they are capable of doing so and reverting to symmetric key cryptography only when such use is necessitated by the weakness of a given device. Allowing devices to obtain different levels of security permits SCADA networks to maximize their security in the decades before such networks are capable of implementing fully public key-based key management protocols. Such a system is obtained through the use of a protocol based on a modified version of SSL using X.509 certificates containing encrypted symmetric keys that allow master devices the option of using the symmetric keys for encrypting the shared secret used to create keying material, instead of using a slave device's public key. This thesis presents the protocol and uses proof-of-concept code to carry out a performance evaluation of the key management scheme.

1. Introduction

Securing the power grid must include providing encryption and authentication capabilities to the devices that control the transmission of data across grid communication lines. In order for encryption and authentication to occur, an effective key management system must be in place to provide the keys needed for the encryption and authentication protocols. This need for encryption and authentication capabilities, and with it, the need for key management, is especially applicable to Supervisory Control and Data Acquisition (SCADA) networks, one type of network often used within the power grid to collect data and transmit it back to a central unit for processing. SCADA networks' architecture and the inability of some devices within SCADA networks to perform public key cryptography produce unique challenges for securing SCADA networks, and, more particularly, for developing key management schemes for SCADA systems.

1.1 Goals

The goal of this thesis is to develop a key management protocol for power SCADA networks, with a focus on creating a scheme that takes advantage of the full security capacity of the system, despite the mixed capabilities of the devices within the network, and that allows for ease of future upgrade.

Because SCADA networks often include some devices that are unable to perform public key cryptography and some devices that are able to perform public key operations, using a key management protocol that fails to take advantage of those devices that are able to use public key cryptography wastes the resources available in the system and makes the system less secure than it could be if the all devices were used to their full potential. Thus, this thesis seeks to develop a key management system that will utilize the full capacity of all devices within the system, allowing those devices that can perform public key operations to use public key cryptography for key management and allowing those devices unable to use public key cryptography to use symmetric key operations.

SCADA networks are also in the process of being upgraded to computationally superior devices that are able to perform public key encryption and that could be integrated into a normal Public Key Infrastructure (PKI). This thesis, therefore, seeks to create a protocol that uses a PKI and Secure Sockets Layer (SSL)-based handshake that differs only slightly from the normal PKI and SSL handshake commonly in use. This would allow for ease of upgrade to a normal PKI system, likely using SSL for key establishment, once all the machines in a given SCADA network developed the capacity for public key operations at some point in the distant future.

1.2 Proposal

To meet the previously described goals, this thesis proposes a key management system based on PKI, using a slightly modified version of SSL. SCADA systems are made up of central master devices, which play the role of the client in the SSL handshake, and they also include potentially remote slave devices, which each play the role of the server in the handshake. The proposed system relies on X.509 certificates with an encrypted symmetric key included in an extension. These augmented certificates allow the SSL handshake to be modified so that when a slave device is unable to perform public key cryptography, a master device can use the symmetric key, which is removed from the slave device's certificate and decrypted, instead of the slave device's public key, when encrypting a value to be used to develop keying material. This allows the master device to use the slave device's public key for encryption with slave devices that are able to perform public key cryptography and to only use symmetric key cryptography with those devices unable to perform asymmetric operations, thus taking advantage of the full capacity of the network. It also allows for easy upgrade to normal PKI, as devices with public key abilities can be phased in with no change to the key management protocol, until an entirely PKI/SSL-based system with devices able to perform public key cryptography is in place.

1.3 Outline

This thesis will first, in Section 2, provide a summary of the issue's background, outlining the various characteristics of power SCADA networks, elaborating on the need

to provide security for such systems, and describing the various aspects of key management. Section 3 will then outline six key assumptions being made about power SCADA networks and the implications these assumptions have for the development of an effective key management protocol. Next, Section 4 will describe various work that relates to developing a key management protocol specific to SCADA networks, focusing on PKI, Kerberos, and two papers that have been published specifically addressing the issue of key management in SCADA systems. Section 5 will describe the proposed key management scheme, and Section 6 will describe the relative performance of the various devices in the different handshakes, offering further justification for the chosen protocol. Finally, Section 7 will suggest direction for future work, and Section 8 will offer conclusions.

2. Background

A typical power SCADA network consists of a central control master device and slave Remote Terminal Unit (RTU) devices that gather data (referred to for the remainder of this thesis as master and slave devices respectively). Connected to the slave devices by serial lines, the master device polls them, one at a time, to retrieve the gathered data for processing. Because the slave devices are constantly gathering data and the master device is able to get steady updates, such devices are used in systems in which real time updates on remote processes and conditions are necessary. Within the power grid, SCADA networks are used "to monitor and manage the electric distribution, transmission

and generation environments."[1] Thus, SCADA networks have a unique purpose that makes them vital to power grid function.

Because SCADA networks control critical data in systems in the power grid, ensuring the authenticity and integrity of the data as it travels over the SCADA lines is very important. *The National Strategy to Secure Cyberspace* identifies "securing […SCADA as] a national priority […as] disruption of these systems can have significant consequences for public health and safety."[2] According to the Center for Scada Security, however, "studies and assessments have found a lack of security in SCADA systems,"[3] because, when SCADA networks were implemented in the power grid, data security was not the key concern; rather, they were designed, for "availability and personal safety." [4] Consequently, many SCADA systems currently in the power grid do not possess the security features necessary to protect the data traveling over SCADA serial lines and, therefore, lack the ability to protect the critical power grid functions dependent on that data.

Because SCADA networks are widely used in the power grid, it is not feasible to replace all the SCADA devices currently in use with devices designed to be used in a secure manner. While devices are slowly being replaced, for decades there will continue

---

[1]Hadley, M., Huston, K., (2006), *AGA 12: Part 2 Performance Test Plan*, National SCADA Test Bed, Retrieved April 5, 2008 from the World Wide Web: http://www.oe.energy.gov/DocumentsandMedia/AGA_12_Part_2_Performance_Test_Plan.pdf, 1. This same article also describes how polling takes place on a SCADA device (1-2).

[2]The White House, (February 2003) *The National Strategy to Secure Cyberspace*, Retrieved May 15, 2008 from the World Wide Web: http://www.whitehouse.gov/pcipb/cyberspace_strategy.pdf, 32.

[3] *Frequently Asked Questions: Why the sudden concern about SCADA systems?*, Sandia National Laboratories: The Center for SCADA Security, Retrieved May 15, 2008 from the World Wide Web: http://www.sandia.gov/scada/faq.htm.

[4] Hadley et al. 1.

to be SCADA devices in the power grid that were not designed to be used in a secure network. Thus, SCADA devices transmit data that needs to be secured; yet, currently they are not designed to implement secure data processing or transmission. To increase the security of the power grid, it is necessary to find ways to add security to existing SCADA devices and networks, while simultaneously securing new SCADA devices as they are added to the network.

One of the main ways in which security can be increased in SCADA networks is through the implementation of symmetric key authentication (e.g. through HMACs) and encryption of data traveling over SCADA links. However, for symmetric key authentication and encryption to be an effective means of security, proper key management is necessary. As Bruce Schneier puts it, "Designing secure cryptographic algorithms and protocols isn't easy, but […k]eeping the keys secret is much harder."[5] Key management, the practice of keeping the symmetric keys used for encryption and authentication secure, secret, and provably attached to the appropriate users, encompasses several major issues.

First, the key itself must be large and random enough to be secure and, therefore, must be created in a secure manner. For a key to be securely generated, it must be as random as possible and, thus, must be created using a good pseudo random number generator seeded with a random value (or be obtained from another truly random source). While the issue of choosing a good random key is important, this thesis does not address this problem. To prevent the use of keys with poor randomness in implementing the key

[5] Schneier, B. (1996), *Applied Cryptography*. New York: John Wiley & Sons, 169.

management scheme described here, one should do as Schneier recommends and use a good source for the random number and avoid known weak keys.[6]

Key management also encompasses the important issue of delivering the keys to the devices that will be using them. Beyond being generated in a secure manner, keys must also be transmitted securely, so that they will not be intercepted in transit by an adversary and later used to decrypt data or authenticate false data. This issue of key transmission differs for the two major types of cryptography – public key cryptography and private key cryptography (also known as asymmetric and symmetric key cryptography, respectively). In private key cryptography, two or more parties share a secret key, and it is this shared secret that allows them to perform encryption and authentication. The same secret key is used for encryption and decryption, with one key, for example, used for such tasks as calculating and checking a HMAC for authentication or encrypting and decrypting data (although a different key would be used for each of these tasks). Thus, it is necessary for this shared key to be secretly transported from one party to another or established in a joint manner that will allow the proper parties to have knowledge of the secret key but will prevent an adversary from learning its contents.

Instead of using a single symmetric key, public key cryptography relies on an asymmetric key pair consisting of a public key and a private key. The private key is used for decrypting ciphertexts and computing signatures, while only the public key is required for encrypting plaintexts or checking a signature. Thus, only the party to whom the key pair belongs needs to know the private key and only the private key needs to be kept a secret. However, public key cryptography still faces the challenge of ensuring that

[6] Requirements for secure key generation summarized from Schneier, 170-175. Requirements for good implementation found in Schneier, 173-174.

other devices can trust that a public key actually belongs to the party for which they wish to perform encryption or authentication.[7]

There are three major ways to transmit symmetric keys from one party to another securely. First, it is possible to secure the transmission of a key by encrypting it with a previously distributed key. This ensures that an adversary will not be able to intercept the new key, as long as they do not possess the old key. Such a method requires only that the previous key was managed securely and that a secure encryption algorithm is used. Second, it is possible to physically transmit a key over a trusted physical channel. Using a trusted physical channel means ensuring that the means of delivery are truly trustworthy – for example, making sure that personnel can be trusted to keep the key secret en route to the destination device.[8] While ensuring that physical channels of transmission are secure is important, this thesis does not address that issue, and instead assumes that such means are secure. Establishing such secure channels (including hiring trustworthy personnel, preventing them from accidentally, or purposely, revealing the key, and ensuring that devices used to store keys are not stolen or lost) is left to those implementing the protocol.

The third means of transmission, is not strictly the transmission of a key. Instead, it is the establishment of a key through the use of a protocol, like Diffie-Hellman, in which a key is created through coordinated steps taken by each party and without either

[7] Smith, S. & Marchesini, J., (2008), *The Craft of System Security*, Upper Saddle River: Addison-Wesley, 163-180, provides a description of public and symmetric key cryptography.
[8] Summarized from Schneier 176-178.

party ever sending the entire key.[9] Both sides end up with the key, but the key itself is never actually transmitted. SSL, the handshake on which this thesis's protocol is based, does not directly transmit a key across a connection, but instead sends a master secret across that can be used by each side to develop material for various symmetric keys.[10] While, in the SSL protocol, this master secret is encrypted using a previously known key, the session key itself is not transmitted across the connection. This thesis, therefore, in addressing means of key transmission, will focus on distributing material used for developing keys by encrypting it with previously known keys and by physically transmitting actual keys.

To address the issue faced by public key cryptography, it is possible to use a trusted Certification Authority that signs a certificate containing a device's public key and information about the device's identity. This ties the key to the identity, and certifies that the key belongs to the device. Other parties that trust the Certification Authority can then trust that the public key belongs to the party claiming it. This is the approach taken by PKI, as will be further described in the related works section and adopted by this thesis.

A third issue addressed under the broad category of key management is ensuring key freshness. If a key is used for too long of a period of time, such use increases the likelihood that an adversary will be able to gather enough outputted messages, or will have the time to run the computations necessary, to figure out the key. Long-term use

---

[9] Smith, S. & Marchesini, J., 178-179 provides a description of the Diffie-Hellman algorithm for establishing a shared secret without either party creating and transmitting the secret itself.

[10] Rescorla, E. (2001), *SSL and TLS: Designing and Building Secure Systems*, Boston: Addison-Wesley, 83-88 describes the process of deriving various keys from the secret material in the SSL protocol.

also makes it more likely that the key will be accidentally or purposely revealed in some other manner, and, as the key is used to encrypt more and more messages, the potential cost of the key's compromise rises. It is necessary, therefore, to regularly change the key being used for encryption over a particular channel, with the specific lifetime dependent on what the key is being used to encrypt and how often it is being used. [11]

As Schneier summarizes, "Different keys may have different lifetimes."[12] In the case of this thesis, keys are used for encrypting other keys or material to develop keys and for encrypting data sent over the SCADA data channels. The keys used for encrypting and authenticating the data sent over SCADA communication lines "should have relatively short lifetimes[,…while the…] Key-encryption keys don't have to be replaced as frequently [as t]hey are used only occasionally […] for key exchange."[13] As will be outlined in Section 5, the key management protocol proposed in this thesis addresses freshness by using frequently-replaced keying material to create session keys for encrypting data and by using less frequently-replaced, but still updated, long-term keys for encrypting the master secret used to create the keying material (equivalent to "key-encryption keys") and for encrypting the keys found within the certificates of the slave devices unable to perform public key cryptography.

Finally, key management also involves dealing with keys that have been compromised. If a key has been compromised, it should no longer be used for encryption or authentication, and all devices using that key for cryptographic operations need to be

[11] Summarized from Schneier, 183-184.
[12] Schneier, 184.
[13] Schneier,184.

informed of the key's compromised status.[14] In the power SCADA system model used in this thesis, notification of key compromise is important only at the master device's level. As will be explained in Section 3, if the security of the master device inside the substation is compromised, the security issues go beyond those involved in cryptography, and notifying the slave devices of the compromise of a master device for cryptographic purposes would be ineffective.

This thesis seeks to contribute to efforts to increase the security of the existing SCADA systems within the power grid by developing a key management system for power SCADA that allows for effective key distribution, freshness, and revocation after compromise.

3. Assumptions and Implications

SCADA networks represent a unique network topology and function and, therefore, have a variety of unique characteristics that must be taken into account when attempting to implement security measures for them. Because it was not possible to gain access to a power facility over the course of developing this work, this thesis makes several assumptions about characteristics of SCADA networks based on information from current literature and from a phone conversation with David Whitehead, Vice President of R&D at Schweitzer Engineering Laboratories, Inc. This section provides an outline of those assumptions (briefly summarized in Table 1) and their implications, noting that it is possible that, with further research, such assumptions could prove to be false, which would necessitate a reevaluation of the affected portions of this thesis.

[14] Summarized from Schneier 182-183.

Table 1: Table of Assumptions Made

1. It is assumed that, within a SCADA network, there commonly exist slave devices that lack the computational capacity necessary to perform public key cryptography.
2. It is assumed that SCADA slave devices are in the process of being upgraded to machines with greater processing power that are able to perform public key operations.
3. It is assumed that the serial lines over which master and slave devices communicate are used to near-capacity, leaving little bandwidth for overhead added by key management communications.
4. It is assumed that substations, and the devices within substations, are secure.
5. It is assumed that slave devices within the typical power SCADA network can be located remotely from the substation, but are visited regularly, although not necessarily frequently.
6. It is assumed that slave devices rarely communicate with other slave devices.

First, this thesis assumes that, while the SCADA master device has computing capabilities typical of a modern computing system[15], some SCADA slave devices lack the capacity to perform asymmetric encryption, at least in a reasonably efficient manner. David Whitehead suggested that slave devices lacked such capabilities, and Dawson et al. state that "RTUs have limited memory and processing power. There are RTUs running

[15] Dawson, R., Boyd, C., Dawson, E., & Nieto J. (2006), SKMA- A Key Management Architecture for SCADA Systems, In *Conference in Research and Practice in Information Technology, Vol. 54, Fourth Australasian Information Security Workshop Hobart, Australia (AISW-NetSec 2006)*, Australian Computer Society, Inc, suggests that "Master stations and sub-master stations are computers with resources at least as plentiful as a modern desktop computer" (Section 2.2).

industry standard protocols on 16 bit Microprocessors with 8 kilobytes of RAM (working memory), and 64 kilobytes of EPROM (persistent memory)."[16] Beaver et al. also support this assumption, suggesting that "many critical SCADA communications […] have maximum delay times on the order of 2- 4 milliseconds"[17], and that "The time to sign and then verify using the public protocols such as DSA, RSA run on the order of 2-11 ms for moderate security [and…] may be many times lower than these values if a less powerful processor is used. At worst, these times may be on the order of 1000 times slower if an 8-bit processor is used. Further, these times do not account for any sort of message queuing."[18] While Beaver et al.'s argument applies to how public key cryptography would add a large delay if it were used to encrypt messages, it also holds for key management – on weak devices, public key cryptography, even if only used for infrequent key management messages, would be prohibitively slow, if not impossible. This thesis assumes, therefore, that there commonly exist slave devices in SCADA networks that cannot carry out public key operations for key management.

This assumption presents a challenge to key management. Several key management protocols, like using RSA or elliptic curve cryptography to implement PKI, require the capacity for public key operations. If any SCADA devices within the network lack the ability to perform asymmetric cryptography, then such key management protocols cannot be used effectively without modifying the protocol. As PKI, and with it SSL, represents one of the primary means of establishing secure communication

[16] Dawson et al., Section 2.1
[17] Beaver, C., Gallup D., NeuMann, W. & Torgerson, M, (March 2002), Key Management for SCADA, *Sand Report*, New Mexico: Sandia National Laboratories, 10
[18] Beaver et al., 11.

pathways and of providing at least partial authentication, this limitation is one of the main issues that this thesis seeks to overcome.

Second, while the assumption is made that many slave devices are unable to perform asymmetric encryption, at least in a reasonably efficient manner, such devices are in the process of being upgraded to more advanced machines with the capacity to perform public key operations.[19] When combined with the previous assumption, this presents another challenge to creating an effective key management system for SCADA devices. Because, as previously described, there are often devices in the SCADA network that are unable to perform asymmetric cryptography, key management must be carried out using symmetric cryptography. However, if key management is performed solely using symmetric key cryptography, it does not take advantage of the superior capability of those devices within the system that are able to perform public key operations. Key management protocols that use public key cryptography are more secure than symmetric key-based systems, as only one party, the device performing the decryption and producing the signatures, needs to know the private key. Public key cryptography is also much more scalable than symmetric key cryptography, as it allows all messages encrypted for or authenticated by one party to use one key; whereas, symmetric key cryptography requires one key per communicating pair. Not utilizing the superior computing power of devices able to perform public key operations would lead the network to have an increasingly suboptimal level of security as the number of computationally stronger devices in the network increased over time. Thus, in order to take advantage of the increasing numbers of computationally strong devices in SCADA

[19] This assumption is supported by Dawson et al, Table 1, and Beaver et al., 7.

networks, this thesis outlines a key management system that allows for public key-based management for those devices capable of asymmetric cryptography and symmetric key-based key management for those devices that lack the capacity to perform public key operations.

Third, this thesis assumes that, considering the current data transmission rates without added security, there is not a large amount of extra bandwidth available on the connections between slave and master devices that can be used for key management. This assumption is based on the conversation with David Whitehead, in which he suggested that SCADA lines were currently used to near-capacity, leaving little bandwidth for the overhead added by the addition of security measures. Hadley et al. also support this assumption, stating that "It is common to find 75 to 80 percent of the bandwidth utilized."[20] However, based on the conversation with David Whitehead and Hadley et al.'s work,[21] this thesis also assumes that not all bandwidth is being used at once. In Whitehead and Hadley et al.'s descriptions of the SCADA topology, the network is set up such that the master device is at the center of several slave devices and queries the slave devices one at a time. The slave devices then respond with large amounts of data. This topology seems to indicate that traffic along the SCADA lines would be sporadic and might follow set patterns of use, depending on when the master device needed to query the various slave devices. This thesis, therefore, does not address the issue of ensuring that key management can take place within such a reduced-bandwidth environment, beyond ensuring that any data that it would be necessary to send in the course of the key management processes would only need to be sent at short,

[20] Hadley et al., 2.
[21] Hadley et al., 2.

schedulable intervals, which would allow it to be sent during lag times in the patterns of use. This thesis does not address how such lag times would be found or how such transmissions would be scheduled, leaving such implementation to future work.

Fourth, this thesis assumes that devices within substations (the master devices) are secure. Beaver et al. also make this assumption,[22] and David Whitehead suggested that it would be acceptable to assume that devices within a substation were secure because, if the substation were to be compromised, the security breach would go beyond cryptographic security. This assumption affects the network model used in the development of key management protocols. Because devices within substations are assumed to be secure, the channels that are assumed to require encryption and authentication and, therefore, keys in order to perform encryption and authentication, are those between master devices and the slave devices and between master devices. Thus, key management only needs to take place between these devices, not within a substation between the master device and other substation devices.

Fifth, this thesis assumes that the slave devices in power SCADA networks can be located remotely from the substation and, thus, from the master device, but that even remotely located slave devices are physically visited by personnel on a regular basis. David Whitehead mentioned that remotely located SCADA devices were visited by personnel and suggested the possibility that the devices were visited monthly. This assumption indicates that keys used for encrypting communications cannot be kept fresh through transmission over physical channels, as devices are not visited often enough to

[22] Beaver et al., 10.

replace the keys with the necessary frequency. However, because visits do occur, there is a reliable means of physically updating keys on a longer-term time schedule.

Sixth, and finally, this thesis assumes that slave devices rarely communicate directly with one another. Although Dawson et al. suggests that such communication does take place and should, therefore, be allowed for in any SCADA key management scheme,[23] Beaver et al. present a model of SCADA communication in which slave devices that wish to communicate with one another must do so through the master device to which they are connected, arguing that such a model limits the damage that one compromised slave device can inflict upon the network, simplifies the process, and reduces the scope and cost of the protocol. They conclude that in some special cases slave devices should be allowed to communicate directly with each other without going through the master device; however, these cases could be dealt with on an individual basis and would not have to be part of the generalized protocol.[24] Based on the strength of Beaver et al.'s arguments, this thesis assumes that slave devices do not communicate directly with each other but, rather, communicate through a master device. This thesis will, however, outline how special cases of slave device to slave device communication could be added without significantly changing the protocol.

These assumptions, based on literature and on the conversation with David Whitehead, shape the concept of SCADA devices used in this thesis. In such a network, a master device that can perform asymmetric cryptography queries slave devices, some of which that can perform asymmetric cryptography and some of which that cannot, with the number of those which that can perform asymmetric cryptography increasing over

[23] Dawson et al., Section 2.5.2.
[24] Beaver et al. 13-14.

time. The assumptions also shape the challenge addressed by this thesis – the need to create a key management system in which public key cryptography can be used for those devices that are able to perform such operations, while still allowing for symmetric key cryptography for those devices that lack the capacity to carry out asymmetric key functions.

4. Related Work

Two major categories of work relate to this thesis – work related to key management for general cryptography (more specifically, related to using Public Key Infrastructures (PKI) and Kerberos for key management) and work related to securing SCADA networks and providing key management for securing SCADA networks.

4.1 Work Related to Key Management

There has been a great deal of work done on the general issue of providing key management for cryptographic systems. Schneier provides a general outline of work related to the issues involved in key management.[25] However, this thesis concentrates on addressing the idea of using X.509 certificates, with symmetric keys included in their extension fields for use in establishing keying material that can be used to create symmetric session keys. Therefore, it draws primarily upon two key management schemes that can be used to establish symmetric session keys – Kerberos and PKI.

Kerberos is a symmetric key-based key management protocol that performs key management through the use of a central server that issues symmetric key-encrypted

[25] Schneier, 169-187.

tickets that include a symmetric session key. The encrypted tickets tie a client's identity to the symmetric session key in such a way that a server the client wishes to use can trust that the client is the one who shares the session key. For this to work, each client on the network has a secret symmetric key that it shares with the Kerberos server. When a user wishes to use a server on the network, the user first authenticates himself or herself to the client using his or her password. Then, the client contacts Kerberos to obtain a key to use to communicate with the Kerberos Ticket-Granting Service, and Kerberos responds with a key encrypted with the client's password. Using the key it received from the Kerberos server, the client then contacts the Kerberos server's Ticket-Granting Service with a request for a ticket for a session with the desired service. Kerberos's Ticket-Granting Service responds to the request by issuing a ticket for the desired session that identifies the client and the server and includes a symmetric session key produced by Kerberos for the requested session. The Ticket-Granting Service sends the client both the key encrypted under the client's symmetric key and a ticket encrypted under the symmetric key shared between Kerberos and the server with whom the client wishes to communicate. This allows the client to present the ticket to the server to prove its identity and establish a session key, as the server trusts the Kerberos server, and only the Kerberos server could have encrypted the ticket that contains both the client's identity and the session key.[26]

---

[26]Neuman, B., & Ts'o, T, (September 1994), Kerberos: An Authentication Service for Computer Networks, *IEEE Communications, 32(9)*, 33-38, Retrieved May 28, 2008 from the World Wide Web: http://gost.isi.edu/publications/kerberos-neuman-tso.html, originally presented the Kerberos protocol.

Schneier 566-571 also provides a summary of the Kerberos protocol.

While Kerberos allows for the establishment of symmetric session keys through symmetric key cryptography, the use of Public Key Infrastructures (PKI) helps provide key management for public key cryptography. Whereas Kerberos is designed for a system in which two parties – the client and the Kerberos server or the server and the Kerberos server – need to have a shared secret (the shared symmetric key), PKI only requires that one party know the secret (the private key) while all other parties can use the public key. This leads to a different system, with a focus on using a public/private key pair, rather than a shared secret value, to tie a public key to a party's identity. Whereas Kerberos relies on a central Kerberos server that shares keys with all the clients and servers on the network, PKI relies on a Certification Authority (CA) with a publicly known public key. Instead of tickets that tie a client's identity to a session key, PKI uses public key certificates that can include an entity's identity and its public key and that are digitally signed by a trusted CA using the CA's private key. The CA's signatures ties the entity's name and public key together in the same manner that the Kerberos server's encryption of the ticket with the symmetric key it shares with the server does in the Kerberos scheme – it allows other entities to verify that the CA is certifying that the public key belongs to the entity whose name is in the certificate, and it prevents untrustworthy entities from asserting bindings between attributes and clients (because they would be unable to sign the certificate with the CA's private key).

Because of the properties of public key cryptography, a single public key certificate can be used more universally than a Kerberos ticket can be used. While a Kerberos ticket can only be presented to the server under whose key it is encrypted, a public key certificate can be presented to any other entity who also trusts the CA that

issued and signed the certificate. Because the CA's public key is commonly known among those entities that trust its signature, trusting parties can check the signature and verify that the user and the user's public key have been bound by the CA. Thus, when two entities wish to establish symmetric session keys while using PKI, they can each present their respective certificates, verify the certificates to gain each others' public keys, and then use the public keys to encrypt and send or, in some protocols, establish, a session key that they can use for later encryption.[27] This process is often carried out using Secure Sockets Layer (SSL), a handshake process that can, among other capabilities, provide authentication of at least one of the devices' public keys through verification of its certificate (or, in some cases, chain of certificates) and establish shared secret material for use in creating the symmetric keys for authentication and encryption using the authenticated public keys.[28]

For SCADA networks, neither symmetric key management, nor PKI alone, is sufficient. PKI alone would not work because many SCADA slave devices lack the capacity to perform public key operations, at least in a reasonably efficient manner. Alternatively, while symmetric key management alone could work in the short term, concentrating on creating a specialized key management system made efficient for symmetric key use could be counterproductive in the long term, as it would neglect to take advantage of newer slave devices (with the ability to perform public key operations)

---

[27] Housley, R., & Polk, T., (2001), *Planning for PKI: Best Practice Guide for Deploying Public Key Infrastructure*. New York: John Wiley & Sons, Inc, 17-27 describes the basic functioning of PKI certificates. Polk 43-68 describes the various constituents of PKI and PKI architectures. Polk 69-105 describes X.509 certificates.

Smith & Marchesini, 249-267. also describes PKI.

[28] Rescorla, 57-94 describes the basic SSL procedure.

as they were inserted into the network. Setting up a system with a concentration on symmetric key management would not only waste capacity for security, but could also make an eventual transition to PKI more difficult. Therefore, this thesis looks for a solution to SCADA key management that can implement a modified version of PKI that would allow for symmetric key management for devices that lack the computational capacity to carry out the asymmetric key operations.

There have been some efforts to combine PKI and Kerberos. KX.509, produced by the University of Michigan, allows one to use a Kerberos ticket to create an X.509 certificate,[29] and Kerberos Public Key Initialization uses PKI to authenticate clients to the Kerberos server.[30] However, neither of these schemes allow for a PKI structure in which a phased withdrawal of symmetric key management could occur.

4.2 Work Related to Secure SCADA and SCADA Key Management:

The work related to implementing cryptography in SCADA networks from which this thesis arises is entitled "YASIR: A Low-Latency, High-Integrity Security Retrofit for Legacy SCADA Systems."[31] The YASIR scheme provides a bump in the wire device that allows for encryption and authentication of data using symmetric key cryptography. Such cryptographic operations would require the use of a secure key management protocol to maintain the symmetric keys required for such operations; however, the YASIR scheme currently lacks a means for key management. This thesis seeks to

[29] Information Technology Central Services at the University of Michigan (Oct. 10, 2005), *KX.509: X.509 certificates via Kerberos*, Retrieved May 30, 2008 from the World Wide Web: http://www.kx509.org/ provides details.
[30] Housley & Polk, 38-39 describes Kerberos Public Key Initialization.
[31] Tsang, P. & Smith, S., (To appear August 2008), YASIR: A Low-Latency, High-Integrity Security Retrofit for Legacy SCADA Systems. *23rd International Information Security Conference (SEC 2008)*.Springer-Verlag LNCS.

provide such a key management protocol that could be used to make encryption and authentication schemes like YASIR effective.

Because this thesis is designed to address a need for symmetric key management in power SCADA systems, a variety of closely-related key management work bears only slight relevance. A great deal of research has been done on key management for sensor networks. While such networks have some similarities to SCADA networks in that their abilities to perform cryptographic operations are severely constrained by a lack of computational resources, they also face challenges that are not faced by SCADA networks. Sensor networks are not of a fixed size and must deal with nodes entering and leaving the network and with the network growing to large sizes at various times.[32] For SCADA systems, on the other hand, "The structure of the network and its communication channels will be well defined. Ad hoc communications are not required."[33] Thus, while sensor network key management protocols, like that presented in "SMOCK: A Self-contained Public Key Management Scheme for Mission-critical Wireless Ad Hoc Networks,"[34] address the issue of implementing key management in systems with resource limitations, they primarily address issues relating to the unknown nature of sensor networks that, because of the differences between the two types of systems, are not relevant to SCADA.

More relevant to SCADA systems are two papers that describe key management schemes specific to SCADA. Beaver et al. outline a multi-level key management

---

[32] He, W., Huang Y., Nahrstedt, K., & Lee, W., (2007), SMOCK: A Self-Contained Public Key Management Scheme for Mission-critical Wireless Ad Hoc Networks *Proceedings of the Fifth Annual IEEE International Conference on Pervasive Computing and Communications*, IEEE Computer Society, 1.
[33] Dawson et al., Table 1.
[34] He et al.

architecture with a certification authority. Between substations, this scheme uses PKI with specialized certificates, and between a substation and its slave devices it uses a hash function applied to shared long-term keys that creates symmetric session keys shared between the substation and the slave devices. Dawson et al. present a symmetric key-based scheme that uses a key distribution center.

Both of these protocols offer SCADA network-specific key management protocols. They do not, however, allow for easy upgrade to PKI as SCADA systems are upgraded. While either of the schemes suggested in these papers could provide a viable short-term solution, implementing such a solution might be detrimental in the long run, as it would prevent the use of the increasing resources in the network as the devices were slowly upgraded. Both of these schemes also neglect to take advantage of the full capabilities of the network, if the network includes devices that are able to perform public key operations (Diagram 1 illustrates this wasted capacity over time), because both protocols use shared keys for key management, even with devices capable of public key cryptography. This thesis presents a scheme that would allow key management in the current network while also allowing for future upgrade and while taking advantage of the full capabilities of all the devices within the network.

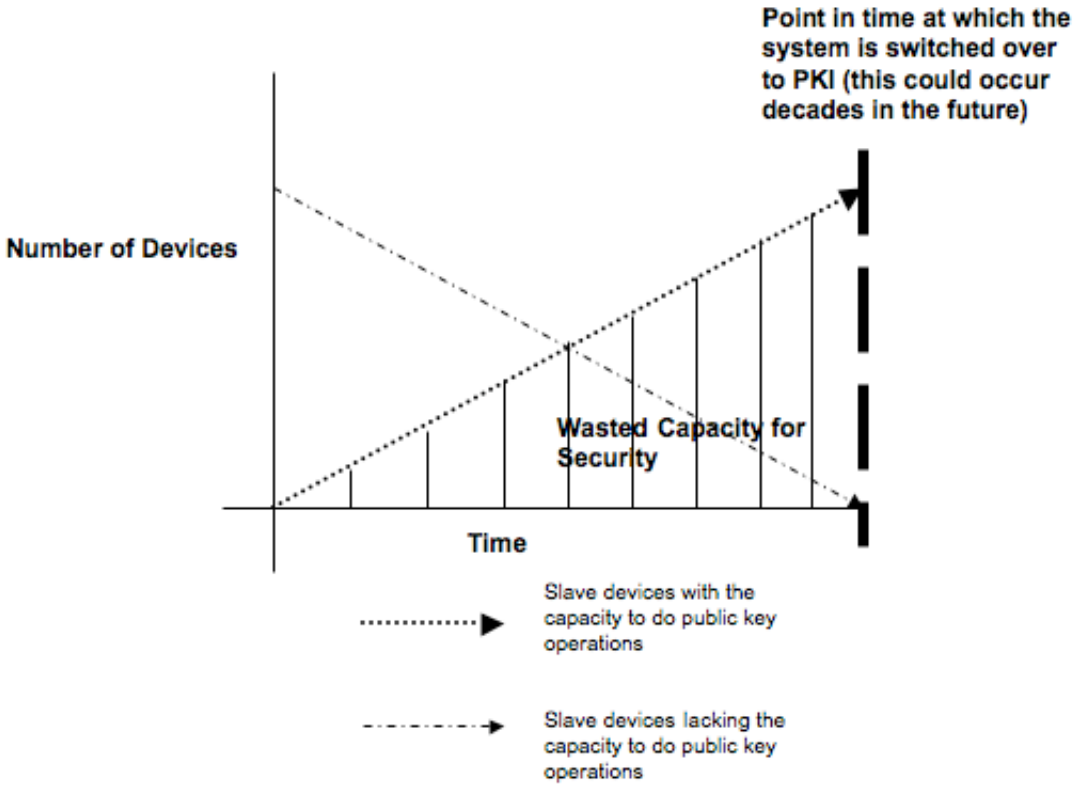

Diagram 1: This diagram illustrates how, if a symmetric key management protocol is used over time, more and more slave devices with public key capacity will be added to the SCADA network, leading to more and more wasted capacity for security within the network until the point in time at which the system is switched over to PKI (or another system that takes advantage of public key capabilities). As shown above, to optimize the amount of the security in the system, it is necessary to allow devices that are capable of public key cryptography to use public key cryptography for key management.

5. Key Management Scheme

The key management scheme presented here is based on the typical PKI key management scheme, with some modifications to allow for its use with slave devices that lack the capacity to perform asymmetric cryptography. PKI uses X.509 certificates signed by a trusted CA to tie a public key to a device's identity and to allow that device to participate in an SSL handshake process – that is, presenting its certificate and using its public key to establish a random value known as the master secret that can be used to derive keying material, including symmetric session keys for use in authentication and

encryption of data. To modify this scheme, the CA must add a long-term symmetric key, known to the CA and the certificate-holding device, to the certificates of those slave devices unable to perform public key cryptography. In order to prevent these keys from being intercepted, these long-term symmetric keys in the slave devices' certificates must be encrypted with a symmetric key sent to the master device by the CA for the purpose of decrypting these encrypted keys. If the slave device is unable to perform the asymmetric key operations necessary to decrypt the master secret encrypted under a public key, this allows the master device another option – it can decrypt the symmetric key in the slave device's certificate and encrypt the master secret using the symmetric key from within the slave device's certificate instead of using the public key in the slave device's certificate.

This section will go into the details of this scheme, describing the various actors in the protocol and how they interact, outlining the details of how the extra symmetric key can be inserted into the X.509 certificates, and explaining how a modified SSL handshake (using the decrypted symmetric key rather than the slave device's public key to encrypt the master secret) would work.

5.1 Actors

There are several actors in this protocol. The first is the master device. As outlined in the previous section, the master device is assumed to have the computational power of a modern computer and is, thus, able to perform both public and symmetric key cryptography. As described previously, the master device and its keys are also assumed to be secure, as they are located within a substation or other such facility. While a

SCADA network topology consists of one master polling several slave devices, there can be multiple master devices (with their own sets of slave devices) arranged hierarchically on the network, with lines running from master stations to sub-master stations.[35] While such links do exist within the network, they will not be explicitly addressed in the protocol portion of this thesis, as all master devices are able to perform public key operations; thus, normal SSL with PKI can be used between master devices.

The second actor is the slave device. Slave devices can be located remotely, in locations physically distant from the substation. As described in Section 3, slave devices vary in their computational power; some are able to perform public key cryptography, while others are limited to symmetric key operations because their computational capacities render public key operations inefficient or impossible.

The final actor is the Certification Authority (CA). The CA certifies the public keys belonging to the master devices and slave devices capable of public key cryptography by producing and signing the devices' certificates with its private key. The CA also produces the symmetric keys stored in the certificates and the symmetric key used for encrypting/decrypting the symmetric keys stored in the certificates. Thus, the CA must be reliable, and its private key must be kept very secure. The CA should be similar to that suggested by Beaver et al., in that it should have "sufficient computational resources, be physically secure, and [have] a high quality random number generator,"[36] and it should be located in a secured, supervised area, where the its private key can be protected.[37]

---

[35] Dawson et al., Section 2.
[36] Beaver et al., 8.
[37] Franklin, M., Mitcham, K, Smith, S., Stabiner, J., & Wild, O., (2005), CA-in-a-Box, *Public Key Infrastructure: EuroPKI 2005*, Springer-Verlag LNCS. describes the various

These separate actors all communicate with each other over several general communication links. First, master stations can communicate with each other over the data links that run between them. Second, master stations can communicate with the slave devices to which they are connected over the serial lines connecting the SCADA network. Third, slave devices can communicate with the master device to which they are connected using these same serial lines. As outlined in the description of assumptions, slave devices are assumed not to communicate directly with each other except in special cases outside of the normal protocol. Finally, the CA can communicate with the master device and the slave devices. Unlike the other communication channels, the CA communicates with the devices over physical, out-of-band channels rather than over data lines. The CA is responsible for generating and signing certificates based on public keys generated by the devices and distributing the certificates among the devices. The CA is also responsible for producing the symmetric keys used in the protocol. Along with creating these keys, the CA must ensure that the symmetric keys stored in the certificates are distributed to the appropriate devices along with the certificates, and that the key used to encrypt the symmetric key in the certificates is transmitted to the master device. To accomplish all of these tasks, public keys that need to be certified from the various devices and the material the CA generates for the master and slave devices are transported using the physical visits to the various devices – that is, the monthly personnel visits to the slave devices suggested by David Whitehead or a physical visit to the master or slave device by CA personnel.

---

requirements for a CA, the options for building a CA, and how an organization can develop its own CA from commercial software.

These various actors and communication channels produce a simple network model: various slave devices communicate with one master device, which communicates with other master devices and the various slave devices while the Certification Authority simultaneously communicates with all devices intermittently over physical, out-of-band channels. This thesis will primarily address a subset of this model – the interactions between slave devices, the CA, and one master device. As described previously in this section, it will not address master device to master device key management.

5.2 Keys and Certificates

To begin to secure these communication channels, the different actors need to create and possess various keys (as outlined in Table 2) and certificates.

Table 2: Summary of Keys

Key	Key Created By:	Key Known By:
Symmetric key found, encrypted, in the certificate of a slave device unable to perform public key cryptography	-The CA	-The CA -The slave device to which the certificate belongs
Symmetric key used to encrypt the symmetric keys found in the certificates of slave devices unable to perform public key cryptography	-The CA	-The CA -The master device
Slave device (capable of public key cryptography)'s private key	-The slave device to which it belongs	-The slave device to which it belongs
Slave device (capable of public key cryptography)'s public key	-Created by the slave device to which it belongs -Certified by the CA in the slave device's X.509 certificate	-The CA -Presented, through the use of the certificate, during the handshake procedure to allow the master device to gain knowledge of it
Master device's private key	-The master device to which it belongs	-The master device to which it belongs
Master device's public key	-Created by the master device to which it belongs -Certified by the CA in the master device's X.509 certificate	-The CA -Presented, through the use of the certificate, during the handshake procedure to allow slave devices capable of public key cryptography to gain knowledge of it
CA's private key	-The CA	-The CA
CA's public key	-The CA	-All master devices and slave devices capable of public key operations

First, the Certification Authority must have a public and private key pair, with the public key known to all the master devices and all the slave devices that are able to perform asymmetric key cryptography and with the private key absolutely secure and unknown to any other devices besides the CA itself. This key pair is used in a manner typical of PKI – the CA uses it to sign certificates, tying other public keys to device names and attributes, with the security of the signature dependent on the security of the CA's private key.

Second, the master device must have a symmetric key for decrypting the keys encrypted within the certificates of its slave devices that are unable to perform public key operations, which will allow for the establishment of keying material if the slave device is incapable of public key cryptography. This symmetric decrypting key is generated by the CA and physically distributed to the master device through an out-of-band, physical channel.

Third, each master and slave device needs an X.509 certificate signed by the CA. All of these certificates must include the typical information found in an X.509 certificate – identifying information, including the device's common name, public key, and length of time of validity (with the public key value containing a dummy number in a slave device's certificate if the slave device is incapable of public key cryptography). Those certificates distributed to the slave devices, or at least those distributed to those slave devices that are unable to perform public key operations (although for ease of implementation it may be simpler to standardize all certificates), must also include a symmetric key encrypted with the symmetric key the master device receives from the CA. The slave devices that are incapable of public key cryptography also need to know the unencrypted form of the symmetric key contained, in encrypted form, in their certificates. The symmetric keys within the certificates would, therefore, be distributed to the slave device along with the certificate over the out-of-band channel. As will be described in more detail in the following section, this allows the master device, when it receives a certificate from a slave device unable to perform public key operations, to decrypt the symmetric key in the certificate the slave device presents and use that symmetric key (rather than the slave device's public key) to encrypt the master secret for

establishing keying material for a session key. The symmetric key in the certificate is able to take the place of the slave device's public key because, after decryption takes place, both the master device and the slave device have knowledge of the key and no adversary should know it.

X.509 allows for information outside of the basic information required for certificate generation to be included in a certificate through the use of extensions. Extensions can either be standard and defined by X.509 (like the BasicConstraints extension which appears as the first extension under X509v3 extensions in Diagram 2), or they can be non-standard and user defined[38]. While it is generally better to use standard extensions as they are "widely supported by commercial products,"[39] because the extension containing the encrypted symmetric key is used by only a subset of slave devices within the specific SCADA network implementing this slightly modified PKI, it is possible to add a private extension to the certificates to hold the symmetric key. An example of such an added private extension can be seen as the second extension under X509v3 extensions in Diagram 2. In this example, a Netscape Comments-type private extension was added (Netscape Comments allow for a variety of data types to be included in an extension, including an encrypted key value), and it contains an encrypted symmetric key (which is not shown in the example, because the encrypted key appears as unprintable binary data).

To allow for such certificates to be potentially integrated with other commercial systems, a case that could arise, for example, if a master device were to communicate with a non-SCADA system using the same CA, it is necessary to mark the added private

[38] Summarized from Housley & Polk 79-98.
[39] Housley & Polk, 80.

extension as non-critical. If a certificate has a critical extension that a device does not recognize, the device will automatically reject the certificate; however, if the extension is marked non-critical and the device does not recognized the extension the device will simply ignore the extension and not necessarily reject it.[40]

```
Certificate:

    Data:
        Version: 3 (0x2)
        Serial Number: 13 (0xd)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: CN=ManyasThesis, ST=New Hampshire,
C=US/emailAddress=manya@dartmouth.edu, O=Dartmouth
        Validity
            Not Before: May 29 10:43:41 2008 GMT
            Not After : May 29 10:43:41 2009 GMT
        Subject: C=US, O=Dartmouth, OU=CS,
CN=SlaveDevice/emailAddress=manyas@dartmouth.edu
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:b8:d4:a7:7d:78:3c:10:84:e9:5a:b7:55:bd:5c:
                    5b:0e:46:8f:41:cd:f9:82:f3:04:23:d2:d2:26:50:
                    c4:e2:04:9d:e8:16:96:04:d7:6a:d1:04:66:cc:bd:
                    09:1f:c4:77:6c:39:65:aa:1a:79:c4:43:e7:de:0d:
                    d3:09:d6:ca:d9:2b:d8:92:73:a3:ab:08:69:0d:cc:
                    f1:dc:26:8a:b4:96:28:3c:b3:70:49:52:a9:b3:df:
                    54:a6:a3:c6:04:2c:e0:82:b7:15:21:93:e8:45:89:
                    85:69:73:b8:47:b9:f7:d6:5e:d5:b0:eb:ba:fb:ee:
                    44:8d:37:b6:5a:9e:c1:a0:3b
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                      ENCRYPTED SYMMETRIC KEY (UNPRINTABLE BINARY)
    Signature Algorithm: md5WithRSAEncryption
        63:7b:59:84:0c:19:93:4c:a7:4a:e3:86:5e:a3:38:bf:e3:fb:
        e4:36:46:9d:72:5c:60:d5:1d:bb:89:ca:4b:b1:5f:7d:d1:f3:
        8a:23:ab:79:a8:52:3a:e5:11:ff:74:b0:c4:c8:93:3f:42:98:
        f3:b8:24:34:ce:e1:81:ff:ae:57:23:4e:13:4a:1a:87:39:f6:
        b8:ed:2e:75:9d:62:99:53:09:93:18:90:b2:f2:e7:6a:b3:d7:
        29:04:bb:97:11:3c:02:e2:91:4d:61:64:8c:3d:1c:10:11:f6:
        bc:2c:0f:de:f7:90:48:7d:bc:1e:c0:17:31:57:6d:a4:a2:b2:
        9a:8e
```

Diagram 2: Example of an X.509 certificate with an added encrypted symmetric key (seen under X509v3 extensions as the Netscape comment type field – the encrypted symmetric key would appear as a line of binary)

[40] Housley & Polk, 80 describe the criticality flag in X.509 certificates.

This is the desired behavior in this system. If the certificates are used in the specialized system, the devices will use the encrypted symmetric key extension as necessary. If the certificates are used outside of the system, the other devices will ignore that extension field, but, assuming the slave or master device is capable of public key operations, they will be able to perform a normal SSL handshake or other PKI-based operation with the device. This allows for both the utilization of the maximal capacity within the SCADA network and potential ease of integration with other networks and future upgrades.

Master devices and slave devices capable of public key cryptography also require the public/private key pair that goes along with the certificate. The master device and slave devices with the proper computational capacities create their own public/private key pairs in order to prevent any other device from knowing the private keys. The devices' public keys are then transported to the CA using physical means of transportation. The CA authenticates the public keys as belonging to the master or slave devices through physical, out-of-band measures (e.g. authenticating personnel bringing the public keys using ID cards). For each public key the CA authenticates, it creates and signs a certificate that ties the public key to the master or slave device from which it arrived. Certificates are then distributed to the devices using physical, out-of-band channels.

Diagram 3 illustrates the final network model, showing the master device, the CA, the two types of slave devices, the various communication paths, and the resources, keys, and capabilities each device has.

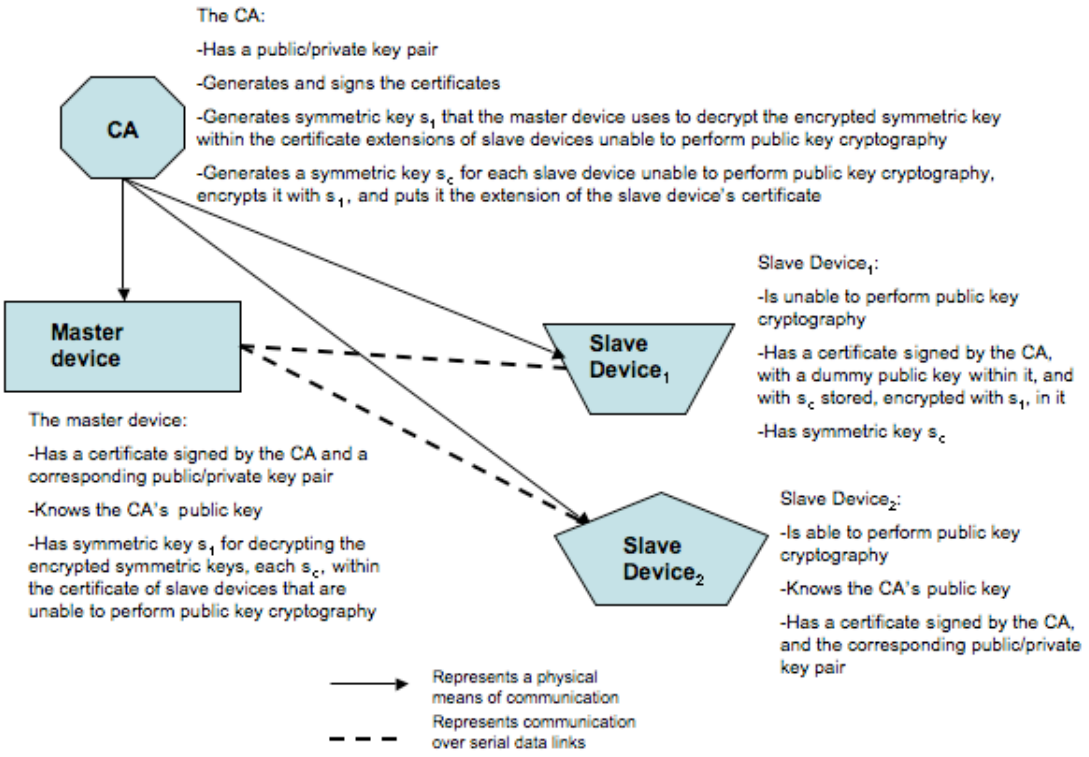

Diagram 3: This diagram shows the network configuration and the various capacities, keys, and certificates each device has.

This establishes the basic model that will be used in the key management protocol for key distribution and freshness, as outlined in the next section.

5.3 Key Distribution and Freshness

As described in Section 2, one of the main aspects of key management is key distribution in order to maintain key freshness. In the model described in the previous section, keys are distributed in two ways – physically and over data lines.

Symmetric keys from the CA, along with the certificates certifying that they are tied to a particular device, are physically moved from the CA's location to the various slave devices. The security of this move depends on the trustworthiness of personnel and the physical channel used for transportation and is not addressed in this thesis. These

physically distributed keys, however, cannot be changed on a short-term regular basis. As described in the assumptions section, remote slave devices are not necessary easily reachable and are only visited on a (potentially) monthly basis. Using the keys within the certificates to encrypt data, therefore, is not a viable option, as they would be used repeatedly and would rapidly lose their freshness. Instead, it is necessary to create, exchange, and use shorter-lived keying material to create symmetric session keys for use in encrypting data.

It is also necessary to use shorter-lived keying material with slave devices that are able to use public key cryptography. Public key encryption operations are much more computationally costly than symmetric key operations, so it is more efficient to use the public keys to exchange keying material that can be used to create a symmetric session key and then use that symmetric session key for encryption. The procedure often utilized to establish the material used to create symmetric session keys for encryption and authentication using public keys and PKI, is SSL (this, among other functionalities). SSL is a handshake process that takes place between a client, a device that is requesting a service, and a server, a device that is providing the service. In the case of the SCADA network, the master device plays the role of the client, and the slave device plays the role of the server. Diagram 4 shows a simplified version of the typical SSL process applied to a SCADA master device and a slave device that can perform public key operations.

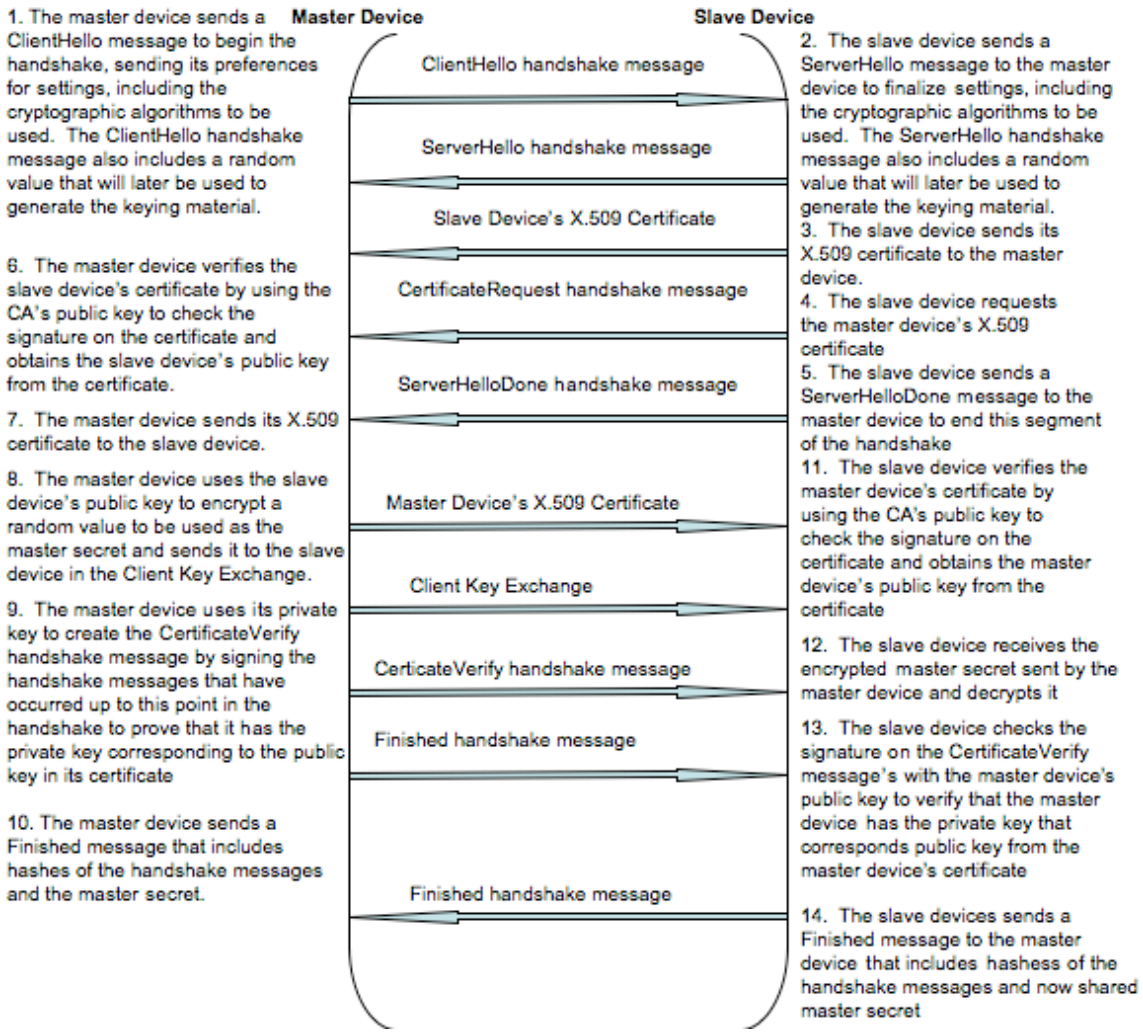

Diagram based on a diagrams of the simplified SSL handshake found in Rescorla, page 60 and page 98. Process based on the SSL handshake described in Rescorla pages 61-88 and pages 96-97, 108-112 .

Diagram 4: This diagram illustrates a simplified version of the SSL handshake between a SCADA master device and a SCADA slave device that is able to perform public key operations.

First, the master device sends the slave device the first handshake message, called the ClientHello message. This message alerts the slave device to the fact that the master device wishes to establish new keying material. The ClientHello message also allows the master device to send the slave device its preferences for the settings to be used during the handshake exchange, including the cryptographic algorithms that it prefers be used for encryption and authentication. Finally, the ClientHello message includes a random

value, chosen by the master device, that both devices will later use in creating session keys in order to prevent replay attacks (attacks in which an adversary, without knowledge of the keys involved, repeats a message used during an earlier exchange and passes it off as part of a later exchange).

Second, the slave device sends the master device the ServerHello message. The ServerHello message is similar to the ClientHello message. It establishes the settings that will be used for communication, including finalizing the encryption and authentication algorithms, and includes a random value that both the slave and master device will later use in computing session keys. After sending the ServerHello message, the slave device sends the X.509 certificate it received from the CA. Because the X509 certificate contains the slave device's public key and because, in this case, the slave device is capable of public key cryptography, sending the certificate to the master device provides the master device with the means to encrypt messages to the slave device using public key encryption. Once the slave device sends its certificate to the master device, it sends the master device the CertificateRequest handshake message to request that the master device send its certificate so that the slave device can verify the master device's identity. Finally, the slave device indicates that it is finished with this segment of the handshake by sending a ServerHelloDone handshake message.

When the master device receives the slave device's CertficateRequest message, the master device sends the slave device its X.509 certificate. Once the master device has received the slave device's certificate, it is able to verify that the slave device's public key actually belongs to that slave device by verifying the signature on the certificate using the CA's public key. When the master device has verified the authenticity of the

slave's public key, the master device encrypts the random value that will serve as the master secret with slave device's public key and sends it to the slave device in the ClientKeyExchange message (as described previously, the master secret is the secret material that, combined with the random values from the ClientHello message and ServerHello message, is run through a key derivation algorithm to create, among other things, the session key shared by the master and slave devices for use in encrypting and decrypting data and the shared session key used for authenticating messages). After the ClientKeyExchange message is sent, the master device sends the CertificateVerify handshake message, a message consisting of the handshake messages signed using the private key corresponding to the private key the master device included in its certificate, proving that the master device actually possesses the public/private key pair and that the master device is, in reality, the party bound to the public key in the certificate. The master device then creates hashes using the master secret and the handshake messages and sends the hashes to the slave device in the Finished message so that the slave device can confirm that the exchange was not tampered with.

After receiving the encrypted master secret from the master device in the ClientKeyExchange message, the slave device decrypts the master secret using the symmetric key contained within its certificate (that it received, unencrypted, from the CA). The slave device also verifies the master device's X.509 certificate, and uses the public key found in the certificate to check the signature in the CertificateVerify handshake message, in order to verify the master device's identity. The slave device then checks the hashes in the Finished message it received from the master to confirm that the handshake was not tampered with and ends the basic handshake by sending a final

Finished message containing hashes created using all the handshake messages it received (including the final Finished message from the master device) and the master secret to the master device to again confirm that the handshake was not tampered with.[41]

At the end of this handshake, both the master device and the slave device have a shared secret (the master secret), and potentially session keys (derived from that shared secret and the random values passed in the ServerHello and ClientHello messages) that they can use to encrypt data traveling over the serial line between them. An adversary listening in the middle of the exchange would be unable to create the same session key, because one component necessary to create the keying material (the master secret) only traveled over the link in encrypted form. As described previously, because of the random values, the adversary would also be unable to repeat previously sent handshake messages. When the handshake completes, each side has also authenticated the other side through the use of the X.509 public certificates. Thus, SSL allows for both sides to end up with secure symmetric session keys for encryption and authentication.

A modified version of the typical SSL handshake is necessary in order to produce the same results for slave devices that are unable to perform public key cryptography. There are two major areas in which the handshake differs. First, the master device is unable to encrypt the master secret using the slave device's public key, and, second, the slave device is unable to verify the master device's identity through the use of the master device's X.509 certificate and the VerifyCertificate handshake message.

The slave device cannot perform the asymmetric key operations. Therefore, when the master device sends the ClientKeyExchange message, encrypting the master

[41] Description of basic SSL exchange summarized from Rescorla, 57-88. Description of client authentication aspects summarized from Rescorla 96-98 and 108-112.

secret using the slave device's public key would not allow the slave device to decrypt the message and gain access to the master secret. Instead, the master device can extract the encrypted symmetric key from the extension in the slave device's certificate, decrypt it using the symmetric key the CA sent to the master device (with which the CA also encrypted the key in the certificate) and use the now-decrypted symmetric key to encrypt the master secret for the ClientKeyExchange. Diagram 5 illustrates a simplified version of the modified SSL process for slave devices that lack the capacity to perform public key operations. The changes from the normal SSL handshake are shown in bold and are highlighted by boxes.

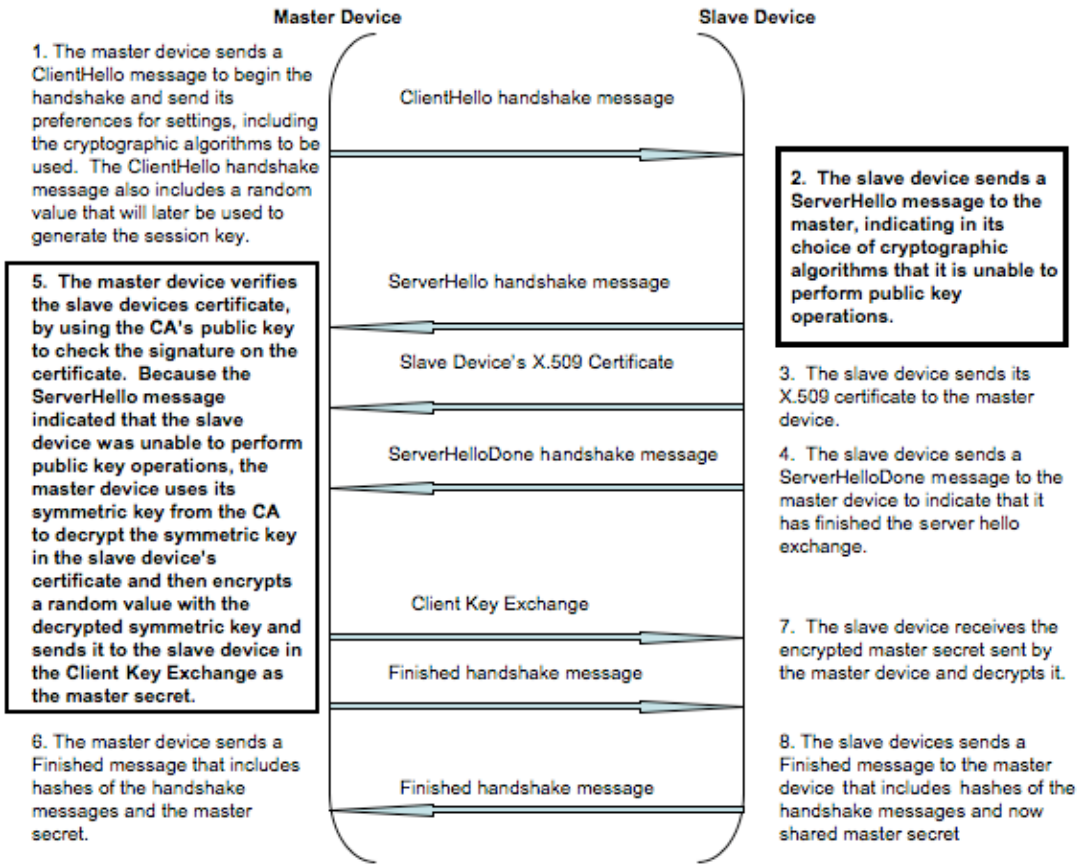

Diagram based on a diagram of the simplified SSL handshake found in Rescorla, page 60. Process based on the SSL handschake described in Rescorla pages 61-88.

Diagram 5: This diagram illustrates a simplified version of the SSL handshake between a SCADA master device and a SCADA slave device that is unable to perform public key operations (changes from the normal SSL handshake shown in bold and surrounded by boxes).

As it is possible to see in Diagram 5, the handshake is nearly the same with slave devices that are unable to perform public key operations as it is with devices that have the capacity for such operations. There are only three major differences. The first difference can be seen in Step 2 of the handshake in Diagram 5. Whereas slave devices that are able to perform public key operations include whatever preferences they have for cryptographic algorithms in the ServerHello message, the slave devices that are unable to

perform asymmetric key operations indicate in the ServerHello message that they are unable to use public key functions.

This message alerts the master device to the fact that the slave device is unable to perform public key cryptography. Therefore, when the master device sends the master secret to the slave device in the ClientKeyExchange message, the master device does not encrypt the value using the slave device's public key; instead, it reverts to symmetric key cryptography to maintain the value's secrecy. This second difference between the handshakes can be seen in Step 5 of the handshake in Diagram 5. Instead of using the slave device's public key to encrypt the master secret for use in the ClientKeyExchange message, the master device extracts the encrypted symmetric key from the extension in the slave's certificate, decrypts it using the symmetric key it received from the CA for decrypting the symmetric keys in the certificates, and uses this decrypted symmetric key to encrypt the master secret for the ClientKeyExchange. As in normal SSL, this allows both the master device and the slave device to gain knowledge of the master secret without an adversary being able to intercept the secret value. The master and slave devices are then able to use this master secret, along with the random values from the ClientHello and ServerHello messages, to derive session keys for encryption and authentication. The use of these random values again prevents an adversary from performing replay attacks, and the client and server are left with the ability to derive secure session keys from the shared master secret.

The third difference can be seen in how this modified handshake deals with the problem that arises in verifying the master device's identity. While the master device can still verify the slave device's identity using the slave device's certificate, the slave device

is unable to perform the public key operations necessary to verify the master device's identity using the master device's certificate and the CertificateVerify message. Therefore, as can be seen in Diagram 5 (compared to Diagram 4) a slave device that is unable to perform public key cryptography does not send the CertificateRequest handshake message to the master device and, therefore, does not receive the master device's X.509 certificate or the CertificateVerify handshake message. This is because the slave device would be unable to check the signature on the certificate or the handshake message, so receiving such data would only waste time and bandwidth. Instead, the slave device indirectly verifies the master device's identity through the master device's knowledge of the symmetric key, which is needed to decrypt the symmetric key in the slave device's certificate. Because only the master device should know the key needed for this decryption operation, any adversary would be unable to obtain the decrypted version of the symmetric key in the slave device's certificate and, therefore, would be unable to encrypt the master secret under the proper value. The encryption of the master secret using the key obtained from the certificate, combined with the Finished handshake message created from the master secret (and, therefore, proving knowledge of the encrypted value) proves the identity of the master device. Thus, at the end of this second type of handshake, both the master device and slave device have been able to authenticate each others' identities, and both have knowledge of the master secret that can be used to derive session keys for encrypting and authenticating data. Thus, the combination of these two handshakes allows both sides to authenticate each others' identities and to end up with secure symmetric session keys for encryption and

authentication, regardless of whether or not the slave device is able to perform public key cryptography.

5.4 Initializing Devices

To add a new device to the network, the device produces a public/private key pair, if it is able to do public key cryptography, and the CA issues a certificate for the device, certifying that the public key belongs to the device. The certificate includes an encrypted symmetric key that the master device can extract if the device is unable to do public key cryptography. The certificate and, if the device is a slave device that is unable to do public key cryptography, the symmetric key are then physically loaded onto the new device, either at the production facility or, if the system is being put into place using devices that have already been put into the field, by CA personnel.[42]

When the master device wishes to establish a session key with a new device, it goes about doing so in the same way it would with an established device, and the signed certificate vouches for the new device when it enters the network. When a new master device is added to the network, it is also issued a symmetric key for decrypting the symmetric keys in the certificates of the slave devices with which it communicates. Thus, new devices can be added to the network, or the proper materials to create this protocol can be added to devices already in an existing SCADA network

5.4 Revoking Keys/Certificates

[42] CA idea suggested by Patrick Tsang, PhD student in the Dartmouth Computer Science Department, personal communications, May 28, 2008.

As described in previous sections, it is only necessary to revoke keys and certificates for the slave devices. If a master device's key or certificate is compromised, this indicates that security problems are larger than those arising from cryptographic compromise. To revoke a slave device's certificate, the master device keeps a Certificate Revocation List that keeps track of those certificates that have been revoked, and does not accept certificates on that list. This will prevent the master device from accepting a certificate from a slave device whose long-term symmetric key (contained within its certificate) or private key has been compromised and whose certificate has been revoked.

A certificate should be revoked and added to the Certificate Revocation List as soon as the master device becomes aware of the compromise of a slave device. This leaves the risk that a slave device will become compromised without the master device being aware of it occurring. Such an issue would have to be handled by attempting to maximize detection of compromise, perhaps through the detection of unusual traffic patterns or of protocol disturbances. The details of such a system, however, will have to be left to future work.

5.5 Bandwidth Considerations

As outlined in Section 3, it is assumed that there is limited bandwidth available for the key management process; however, it is also assumed that there are certain periods in which it would be possible to send key management data. This protocol could fit this assumption by not having the master device establish keying material/session keys when it wishes to send data but, rather, at a pre-planned time that could be scheduled into the usage patterns of the network. Thus, it could be possible to use this protocol to get

around bandwidth limitations if such usage patterns were confirmed and pinpointed. Actually determining a means to find, confirm, and locate such patterns, however, is left to future work.

5.6 Slave Device to Slave Device Key Management

As mentioned previously, there might be some special cases in which two slave devices regularly have a need to communicate and, therefore, should be provided with a special instance of direct key management. If both devices are capable of public key cryptography, they can establish secret keying material in the same manner as the master device and a slave device capable of asymmetric cryptography. Because both devices would have fully functioning public key certificates, no additional material or modifications to the protocol would be necessary. This is one of the advantages of using a modified version of PKI; it would allow for increased scalability if it became necessary to add large numbers of slave to slave device links and if the slave devices were all able to perform public key cryptography (a case that might arise if the network topology changed as the devices were modernized).

If only one of the devices in the special case is incapable of asymmetric cryptography, the two devices would perform the same handshake performed between the master device and a slave device that is incapable of public key operations, with the slave device that can perform public key cryptography initiating the process and playing the role of the master device. To set up this special case, it would be necessary for the CA to create an additional symmetric key for decrypting keys within certificates and an additional certificate (for the slave device unable to perform public key cryptography)

with an encrypted symmetric key in the certificate's extension. This certificate, and the symmetric key that was encrypted and included within it, would be distributed to the slave device unable to perform asymmetric key cryptography, and the decrypting key would be distributed to the slave device able to carry out public key operations. The distribution of the keys and certificate would use the same out-of-band channels as normal key and certificate distribution; however, the symmetric keys used would be different from those used in master-slave device key management, as this would prevent the compromise of master-slave device communications if slave-slave device communications were compromised. Once the additional certificates and keys were established, the handshake would proceed normally, with the slave device capable of public key operations playing the role of the master device.

If, in the special case, both slave devices are unable to perform public key cryptography, it would be necessary for the CA to create an additional certificate for each of the two devices, with each certificate containing the same symmetric key, encrypted with the same, new, decryption key. Again, the symmetric keys would have to be different from those used for master-slave device communications to prevent the compromise of slave-slave device keys from compromising master-slave communications. The CA would then distribute the certificates, the symmetric key from within the certificate and the decryption key to each of the two devices. One of the devices would then be designated to play the role of the master device, and the two devices would carry out a variation on the normal SSL handshake. Because neither device would be able to perform public key operations, after receiving each others' certificates in the handshake, the devices would extract the encrypted symmetric key,

decrypt it, and verify it against the symmetric key they received from the CA. This would take the place of checking the signature on the certificate. To confirm that the slave device playing the role of the master device also knew the symmetric key, the CertificateVerify message would include a HMAC with the shared symmetric key, instead of a signature using the private key from of the public/private key pair. The slave device playing the role of the master device would also encrypt the master secret using the shared symmetric key from the certificate, instead of using the other slave device's public key. This would allow two slave devices that were both unable to perform public key operations to authenticate each others' identities and establish a shared secret when necessary.

As these three cases show, although this key exchange protocol does not include slave-slave communication channels, it would be possible to set up such channels in the special cases where they might be absolutely necessary.

6. Performance Differences

Looking at the two handshakes presented in the previous section, it is possible to identify the specific operations unique to each handshake. The performance differences associated with these different operations show how modifying the SSL handshake allows slave devices with more limited computational capabilities to participate in the key management process.

In proof-of-concept code created to illustrate the handshake between a master device and a slave device with a computational capacity that prevents it from performing public key operations, the slave device has to perform one necessary symmetric key

operation that a slave device able to perform public key operations would not have to perform; it has to decrypt the encrypted master secret it is sent by the master device.[43] In the case of the proof-of-concept code, this decryption is performed using DES in cipher feedback mode; however, in actual implementation, it could be done using whichever symmetric key operation best fit the specific SCADA system's use of authentication and encryption (as described in Section 6: Future Work). In this section, however, when this operation is referenced, it will refer to the DES cipher feedback decryption used in the proof-of-concept code. Alternatively, the normal SSL-style handshake, as seen in Diagram 4, necessitates that the slave device perform two public key verifications (one of the signature on the master device's certificate and one of the signature on the CertificateVerify handshake message) and an asymmetric key decryption (of the master secret encrypted under the slave device's public key) that do not take place in the handshake with a slave device unable to perform public key operations. The difference in the demands placed on slave devices with the capacity for public key cryptography and those without the capacity for public key cryptography are summarized in Table 3:

[43] To receive a copy of the proof-of-concept code, email manya.k.sleeper@alum.dartmouth.org. The proof-of-concept presents a simplified prototype of the handshake process, and includes one extra symmetric key operation on the slave device's side for ease of programming that would not be included in an actual handshake.

Table 3: Summary of Handshake Distinctions

Slave device capable of public key cryptography	Slave device incapable of public key cryptography
-Public key verification of the CA's signature on the master device's certificate -Public key verification of the CertificateVerify handshake message -Public key decryption of the master secret sent by the master device after being encrypted under the slave device's public key	-Symmetric key decryption of the master secret sent by the master device after being encrypted under the symmetric key contained in the slave device's certificate

It is possible to show why a computationally weak slave device would be unable to carry out the requirements for the public key-based handshake, yet would be able to carry out the requirements for the symmetric key-based handshake and to show that a computationally stronger slave device would be able to carry out the requirements for the public key-based handshake. To do so, it was necessary to calculate how long it would take weak and strong slave devices to carry out each of the two sets of unique handshake requirements.

Determining how long it would take for a computationally weak SCADA device to perform each of the sets of unique handshake operations was done by timing the slave device's symmetric key decryption of the master secret in the proof-of-concept code,

timing an RSA verification operation[44], and timing an implementation of a public key decryption of the same master secret that the slave device decrypts in the proof-of-concept code.[45] Because the proof-of-concept code was run on a computer with a 3391.654 MHz CPU, and because David Whitehead suggested that weak SCADA devices can have processors with 20 MHz CPUs[46], it was necessary to normalize the timing data, which was done by multiplying the time by the CPU ratio. After normalizing the data for weak SCADA devices, the timing results were as follows in Table 4.

[44] The exact operation timed was the verification of the slave device's certificate by the master device in the proof-of-concept code, as, in the proof-of-concept code, the slave device does not verify the master device's certificate. However, as both the master and slave device possess the same format of certificate in the protocol, and perform the same verification operation, timing this verification operation should have provided data that was representative of the verification that would occur if the slave device verified the master device's certificate.

[45] To obtain these results, the code was run on a 32-bit, Unix machine with a 3391.654 MHz CPU running Fedora. Timing was done using the C function getrusage(), with 20 repetitions for each test, and a loop of a thousand trials for each repetition for symmetric key decryption (in order to gain enough time to get a result), all of which were averaged afterwards. Repeated symmetric key decryption using OpenSSL functions also required two additional operations (the initialization of the decryption context and a malloc) each time a decryption was performed, that would not be required as part of a non-repeated decryption. To correct for this, the two operations were also repeated one thousand times and timed; that time was then subtracted from the total time of the thousand repetitions of decryption and the two operations. Results of 0 (or very close to 0) were assumed to be faulty data and were removed from the data set. Symmetric key decryption was performed using DES in cipher feedback mode, and public key verification and decryption was performed using RSA with a 1024 bit key.

[46] D. Whitehead, personal communications, May 6, 2008.

Table 4: Normalized Times for Unique Handshake Operations

	Symmetric Key Decryption of a 48-bit value (using DES in CFB mode)	Public Key Decryption of a 48-bit value (using RSA with a 1024 bit key)	Public Key Verification of an X.509 Certificate (Using RSA with a 1024 bit key)
Running time on a 3391.654 MHz (32 bit) computer	0.0017003 ms	3.74945 ms	0.94965 ms
Running time normalized for a 20 MHz weak SCADA device	0.2883415 ms	635.84185 ms	161.04421ms

As can be seen in Table 4, public key operations take significantly more time than symmetric key operations, and the overhead that would be added to a computationally weak SCADA device, if it attempted to take part in a public key based handshake, would be considerable. The total differences between a computationally weak SCADA device taking part in a symmetric key-based handshake and a public key-based handshake and a computationally strong SCADA device with the capacity for public key operations (assumed to be comparable to the machine the timing tests were run on) taking part in a public key based handshake, can be seen in Table 5.

Table 5: Total Time Differences for the Two Handshakes and Weak and Capable SCADA Devices

	Public key based handshake run on a slave device capable of public key cryptography (assumed to be comparable to machine timing code was originally run on)	Public key based handshake run on a slave device incapable of public key cryptography	Symmetric key based handshake run on a slave device incapable of public key cryptography
Time added by RSA decryption operation	3.74945 ms	635.84185 ms	
Time added by two RSA verification operations	1.8993 ms	322.08842 ms	
Time added by symmetric key decryption			0.2883415 ms
Total unique time	5.64875 ms	957.93027 ms	0.2883415 ms

As seen in Table 5, the public key operations that are prohibitive for weak SCADA devices (adding 957.93027ms of overhead) are possible for more capable SCADA devices (in this case, the device would be able to perform the handshake with only 5.64875 ms extra time). This shows how the modified SSL-based protocol allows for the maximum level of security by taking advantage of the capabilities of the computationally stronger devices in the network, while providing the computationally weaker devices with an option that is not prohibitively expensive in terms of time.

7. Future Work

There are three major directions future work on further developing this key management system could take. First, to make such a protocol applicable to actual SCADA networks, it would be necessary to develop an implementation of the handshake that could function on SCADA devices. This would also involve determining which symmetric cipher would be best for encrypting the key within the certificate and for encrypting the master secret encrypted with the key removed from the certificate. It would also necessitate figuring out how to deal with the cipher-specific issues necessary for implementation.

Second, to allow this key management protocol to be used in actual power SCADA systems, it would be necessary to determine how the various components of the protocol would fit into the human infrastructure of the power grid. Various roles would have to be defined, and power grid employees would have to be placed in charge of such tasks as transporting new symmetric keys to the slave devices and determining if slave devices had been compromised. All these roles would have to fit into the existing structure of the power grid, adding as little additional overhead as possible and allowing for as few mistakes as possible. Additional work would have to be done to develop the infrastructure needed and to determine how it could merge with the existing power grid organization.

Finally, as mentioned in Section 3, this thesis does not fully address how this key management protocol would overcome bandwidth limitations, although it assumes that there exist identifiable times in which key management data could be sent because of predictable use patterns. Additional work would have to be carried out to confirm that

such patterns exist in the use of SCADA serial lines and to determine how to identify the periods in which it would be best to send key management data to avoid overloading at-capacity data lines.

8. Conclusion

It is necessary to increase power grid security without replacing all the devices within the grid, and, therefore, it is necessary to develop a key management protocol that can allow for key management for SCADA slave devices that are unable to perform public key cryptography. However, in creating a key management system for SCADA networks, it is not necessary to sacrifice the capacity of those devices in the network that are able to perform public key operations. As Section 6 outlined, slave devices of different computational capacities can take part in different handshakes and, while doing so, achieve different levels of security. To take advantage of such differences in the capabilities of devices within SCADA networks, it is necessary to use a system, like the one outlined in this thesis, that allows for symmetric key-based key management while simultaneously putting in place a structure that permits devices that can perform public key cryptography to perform at their full capacities. Such a system would provide the highest degree of security that the system is capable of and, therefore, would provide an added benefit over systems that force devices with the capacity for public key operations to rely on symmetric key cryptography for key management.

Acknowledgements:

I would like to thank my advisor, Professor Sean Smith for helping me with research and writing and for reading my many drafts. I would also like to thank Massimiliano Pala, Patrick Tsang, Sergey Bratus, and Ashwin Ramaswamy for helping me with the issues that arose over the course of writing this thesis and creating my proof-of-concept code, and I would like to thank Jennifer Heinen for proof-reading services and grammatical support.