Low-Latency Cryptographic Protection for SCADA Communications

Andrew K. Wright[1], John A. Kinast[2], and Joe McCarty[2]

[1] Cisco Systems, 12515 Research Blvd., Austin, TX USA 78759
akwright@acm.org
[2] Gas Technology Institute, 1700 S. Mount Prospect Rd.,
Des Plaines, IL USA 60018
{john.kinast,joe.mccarty}@gastechnology.org

Abstract. Supervisory Control And Data Acquisition (SCADA) systems are real-time process control systems that are widely deployed throughout critical infrastructure sectors including power, gas, oil, and water. However, SCADA networks generally have little protection from the rising danger of cyber attack. A retrofit solution to protect existing SCADA communications links must assure the integrity of commands and responses that are typically transmitted over serial lines at speeds from 300 to 19200 bits per second, while introducing minimal additional latency into the real-time SCADA traffic.

This paper describes the key aspects of a cryptographic protocol for retrofit SCADA link protection that leverages the Cyclic Redundancy Checks (CRC) transmitted by existing SCADA equipment to achieve strong integrity while introducing minimal latency. The protocol is based on a new *position embedding* encryption mode which, for a b-bit block cipher, ensures that any unauthentic message an adversary can construct (i) includes at least b randomly chosen bits, and therefore, by a new result proved for error detection by systematic shortened cyclic codes, (ii) contains a correct h-bit CRC with probability 2^{-h}. The low speed of the communications channel limits the rate at which an adversary can make trials, enabling detection of potential attacks before enough trials can be made to achieve any significant likelihood of success. The protocol avoids the need for a decrypting link protection module to buffer decrypted data until an end-of-message integrity check is verified, which would otherwise add significant latency.

1 Introduction

Supervisory Control And Data Acquisition (SCADA) systems are real-time process control systems that monitor and control local or geographically remote devices. They are in wide use throughout a variety of critical infrastructure sectors, including power, gas, oil, and water, and are a critical component of operations. As illustrated in Fig. 1, a typical SCADA system consists of an *operations console*, a SCADA *master*, and one or more *remote units* that share a communications link. The SCADA master runs a program that polls the remote

M. Jakobsson, M. Yung, J. Zhou (Eds.): ACNS 2004, LNCS 3089, pp. 263–277, 2004.

units, receives and interprets responses, reports system status on the operations console, and issues commands automatically or in response to operator actions. The communications link is commonly a dedicated serial line, a dialup serial line, or a radio link, and operates at low speeds such as 300 to 19200 bits per second. Remote units read temperatures, pressures, flows, voltages, currents, frequencies, or other physical quantities, and control valves, circuit breakers, or other devices that influence physical processes. SCADA devices are environmentally hardened to withstand extremes in temperature, humidity, electromagnetic interference, etc. and typically have service lifetimes measured in decades. They are considerably more expensive than comparable commodity devices, and utilities throughout the world have extensive investments in serial-based SCADA hardware.

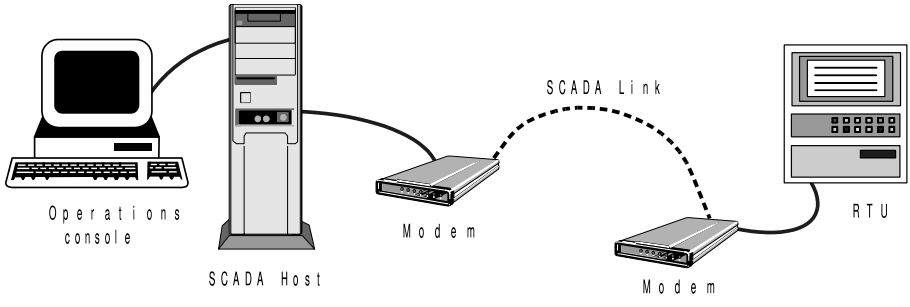

Fig. 1. Typical SCADA System

Due to the nature of the physical processes that SCADA systems control, malicious attacks directed against SCADA systems have the potential to cause significant disruption and damage to critical infrastructures and the markets they supply. SCADA masters and operations consoles are generally well-protected by physical security measures such as perimeter fences and armed guards. Operations consoles generally require at least password authentication. Historically, SCADA masters have not been connected to other computer networks that may have a path to the Internet.[1] SCADA remotes have some physical security, being located at such sites as the tops of telephone poles and transmission towers, and in unmanned stations secured by barbed wire and padlocks. However, SCADA communications links are particularly vulnerable to cyber attack. An adversary with no physical access to any part of a SCADA system can easily compromise dialup links. Compromising radio links requires only proximity and an appropriate transmitter. Leased lines are easily tapped from various points in the telephone network. While most of the over 150 widely deployed SCADA

[1] This is beginning to change with the introduction of SCADA systems that communicate using IP (Internet Protocol), and is a very serious but different problem.

protocols use a Cyclic Redundancy Code (CRC) to detect communications errors caused by noise, CRCs provide no protection against a malicious adversary. Some SCADA protocols require device passwords be transmitted along with commands, but these passwords are usually transmitted in the clear where they are easily snooped. We are aware of no SCADA protocols that include strong provisions to assure the integrity of SCADA traffic against a malicious adversary.

Recognizing the vulnerability of SCADA communications, the American Gas Association (AGA) is preparing a series of recommendations for protecting those communications [1]. The extensive investments utilities have made in existing equipment necessitate a retrofit solution to protect these systems. The diversity of deployed equipment, the ages of deployed hardware, and the limited computational power of deployed devices preclude building protection directly into existing systems. Thus one of AGA's key recommendations will be a standard for cryptographically protecting existing serial-based SCADA communications. This standard will be implemented in the form of a *SCADA Cryptographic Module* (SCM) with two serial ports. A SCADA message received from a SCADA master or remote on a SCM's *plaintext* port will be protected and sent out the SCM's *ciphertext* port, and vice versa. SCMs will be deployed between SCADA devices and the modems for the communications links, as shown in Fig. 2. The key property these devices must assure is *data integrity*: that commands and responses are not forged or altered during transmission.

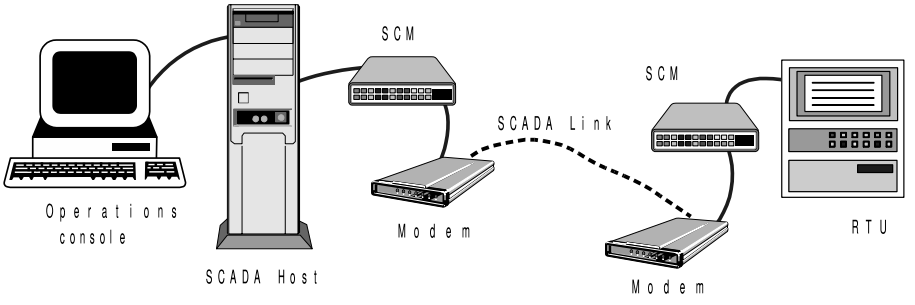

Fig. 2. SCADA System with SCMs Deployed

The constraints imposed by retrofit requirements make designing a protocol to assure integrity more complex than it might first appear. The protocol must introduce minimal additional latency between the SCADA master and remote to avoid impacting the real-time nature of the traffic carried over this low speed channel. The obvious solution of appending to each message some form of integrity check value such as a Message Authentication Code (MAC) would require the receiving SCM to buffer the entire message and check its MAC before forwarding the message out the SCM's plaintext port. Forwarding the buffered message would take as long as receiving it, since both SCM ports will generally

operate at the same baud rate, and would thus double the communications latency. Many SCADA environments are unable to tolerate this much additional latency.

The solution we propose leverages the CRC check performed by the receiving SCADA device. Both sending and receiving SCMs buffer only enough data to fill one block of a block cipher, and forward the encrypted or decrypted block as soon as the block is complete. Thus for a 128-bit block cipher such as AES [2], our protocol introduces 16 characters of latency at each of the sender and the receiver, regardless of message length. Our protocol encrypts a SCADA message using a new encryption mode we call *position embedding* (PE) mode. PE-mode encryption ensures that an any attempt to modify ciphertext blocks or to splice together a new message from ciphertext blocks taken from older messages will result in at least one ciphertext block decrypting to random bits. By a new result we prove for error detection by systematic shortened cyclic codes, this in turn ensures that the unauthentic message contains a valid h-bit CRC with probability 2^{-h}. The low speed of the communications channel limits the rate at which the adversary can make trials, and a MAC checked *after* the message has been forwarded permits SCMs to detect potential attacks before enough trials can be made to achieve any significant likelihood of success. Our protocol avoids the need for the decrypting SCM to buffer decrypted data until an end-of-message integrity check is verified.

The remainder of the paper is organized as follows. Section 2 discusses requirements for a satisfactory solution. Section 3 presents our protocol, establishes a security theorem, and discusses its impact on latency. We discuss implementation considerations in Section 4, and review related work in Section 5. We conclude with our expectations for deployment of this protocol in the field.

2 Requirements for SCADA Communications Protection

To be effective, a protocol for retrofit protection of SCADA communications must address the three classical security properties of confidentiality, integrity, and availability. Since SCADA systems measure and control physical processes that are generally of a continuous nature, and since SCADA systems are simple and repetitive, SCADA commands and responses are relatively easy to predict. Thus confidentiality is secondary in importance to data integrity. To assure data integrity, the protocol must prevent an adversary from constructing unauthentic messages, modifying messages that are in transit, reordering messages, replaying old messages, or destroying messages without detection. Given the predictable nature of SCADA commands and responses, the protocol must be designed to address these issues with the recognition that known plaintext attacks are not only possible but likely. Guaranteeing availability of the communications link is more difficult. Unlike the Internet, SCADA communications networks seldom have redundant communications paths. Thus an adversary with access to the communications link can flood the link to deny communications, or even selectively jam specific messages. However, most SCADA masters monitor link

quality and will report excessive errors to the operator. The protocol should either ensure that this link monitoring facility continues to function, or should provide an alternative.

Many SCADA systems communicate at rates as low as 19200, 1200, or even 300 bits per second. At these speeds, the time required to transmit a single character is significant. Character overhead in message formatting must be kept to a minimum and full message buffering must be avoided if at all possible to limit impact on message latency. Message buffering at the receiving SCM particularly impacts latency since the plaintext port of the SCM will generally operate at the same speed as its ciphertext port, and thus forwarding the decrypted message will take as long as receiving it. Since many SCADA installations continuously cycle amongst devices, initiating a new status poll as soon as a response is received, any increase in latency directly affects the rate at which system state is updated. On the other hand, most embedded CPUs suitable for use in SCMs have more than adequate computation power for cryptographic operations at these speeds. The low communication rate also works to our advantage in limiting the rate at which an adversary can make online trials.

Finally, the retrofit communications protection system must be easy to deploy and manage, and must not adversely impact safe operation of the SCADA system.

3 Retrofit Protection for SCADA Communications

In this section we present an overview of our protocol designed for retrofit protection of SCADA communications, describe the encryption method, including our new position-embedding encryption mode, establish its security, and analyze the latency the protocol introduces.

3.1 Protocol Overview

We consider a simple point-to-point scenario where two SCMs are deployed to protect the communications between a SCADA master and a single remote device. The two SCMs initially share session establishment keys and use these to negotiate shared session keys. The session negotiation procedure is fairly standard and not our focus, so we will not describe it further. The result of session negotiation is that the two SCMs share an encryption key and an authentication key.

A SCM has two communication ports. A SCM receives and transmits *SCADA messages* on its plaintext port. SCADA messages comprise commands, responses, acknowledgments, negative acknowledgments, keep-alive messages, etc. generated by the SCADA system, and are all treated by the SCM in the same manner. A SCM must be able to recognize the beginning and end of a SCADA message, but for this scenario needs no other knowledge of the format of a SCADA mes-

sage.[2] We assume the SCADA message contains a CRC that is checked by the receiving SCADA device.

A SCM transmits and receives *ciphertext messages* on its ciphertext port. Once a session has been negotiated, a SCM sends a ciphertext message to its peer SCM only when it receives a SCADA message on its plaintext port. If a ciphertext message is damaged or lost in transit, our protocol does not attempt to retry it. In this way, whatever methods the SCADA system uses to recover from communication errors and to avoid collisions will continue to operate as usual.

When the first characters of a SCADA message are received on a SCM's plaintext port, the SCM immediately begins transmitting a ciphertext message header that includes a sequence number to its peer. Each time enough characters are received on the plaintext port to fill a cipher block, the SCM encrypts and transmits a block of ciphertext. Finally the SCM transmits a trailer that includes a message authentication code (MAC).

At the receiving SCM, an incoming ciphertext message header signals the start of a new message. The receiving SCM checks that the sequence number in the header is greater than the last sequence number it received. If this comparison fails, the SCM ignores the remainder of the ciphertext message. Otherwise, each time enough characters are received on the ciphertext port to fill a cipher block, the SCM decrypts the block and immediately begins forwarding the decrypted characters via its plaintext port to the receiving SCADA unit. When the trailer of the ciphertext message is received, the SCM computes and checks the MAC. By this time, the decrypted SCADA message may have already been forwarded in its entirety to the receiving SCADA unit. If the authentication check fails, it is too late to prevent forwarding the unauthentic message. Thus the authentication code only alerts the SCM to a possible failure of data integrity. The crux of our design is to encrypt in such a way that an adversary attempting to modify or inject an unauthorized ciphertext message can at best hope to construct one in which no fewer than one cipher block will decrypt to random bits, and thus the h-bit SCADA CRC will be correct with probability 2^{-h}.

3.2 Encryption

Our protocol uses a block cipher encryption algorithm that operates on b-bit blocks, such as AES for which b is 128. We require this cipher to have *real-or-random indistinguishability* [3]: modification of any of the bits of a ciphertext block makes the result of decryption appear uniformly random. Typical block ciphers have this property [4, p. 228]. We denote the single-block encryption and decryption functions for key k by $\mathrm{E}_k[\cdot]$ and $\mathrm{D}_k[\cdot]$ respectively. We also use a message authentication algorithm such as HmacSHA-1 or CBC-MAC. We denote the authentication function for key k' by $\mathrm{MAC}_{k'}[\cdot]$. We assume the sender and receiver have previously negotiated the shared session keys k and k'.

[2] In a multidrop scenario where several remote devices share the communications link, a sending SCM will need to parse the header of the SCADA message in order to select the appropriate encryption key for the receiver.

A SCM maintains a *send sequence* state variable in order to assign a *sequence number* to each ciphertext message it sends. The send sequence variable is initialized to one at session negotiation, and is incremented with every ciphertext message sent. Let S be a SCADA message containing an h-bit CRC. The SCM prepends a sequence number i to the SCADA message S to form a *plaintext message* $P = i\ S$, where juxtaposition denotes concatenation. Let a *padding sequence* z be a sequence of bits beginning with a '1' bit and followed by from 0 to $b-1$ '0' bits [5]. The SCM appends to $P = i\ S$ a padding sequence z such that the concatenation $S\ z$ is a multiple of b bits long. The SCM formats $S\ z$ into $n = \lceil (|S|+1)/b \rceil$ plaintext blocks $p_1, \ldots, p_n$, where $|S|$ denotes the length of S in bits, as follows:

$$P\ z \quad = \quad i\ S\ z \quad = \quad i\ p_1\ p_2\ \ldots\ p_n$$

The resulting padded plaintext message is thus $nb + |i_{\max}|$ bits long, where $|i_{\max}|$ is the fixed number of bits used to represent a sequence number.

The sending SCM enciphers $P\ z$ to the ciphertext message C as follows:

$$\begin{aligned} C &= i\ c_1\ c_2\ \ldots\ c_n\ a \\ \text{where} & \\ c_j &= \mathrm{E}_k\left[p_j \oplus \mathrm{E}_k\left[i\ j\ 0\ldots\right]\right] \\ a &= \mathrm{MAC}_{k'}\left[i\ p_1\ p_2\ \ldots\ p_n\right] \end{aligned}$$

Here $\mathrm{E}_k\left[i\ j\ 0\ldots\right]$ denotes the encryption of i concatenated with j concatenated with enough zeros to fill b bits, and $\oplus$ denotes *exclusive or* (bitwise addition mod 2). The SCM outputs each ciphertext block c_j as soon as it is available. Thus the SCM transmits on its ciphertext port the sequence number i, followed by a sequence of cipher blocks, followed by the MAC a. A simple character escaping mechanism, the details of which are not important here, enables the receiver to parse this message into its header i, body $c_1\ c_2\ \ldots\ c_n$, and trailer a. Including the sequence number in the message's header allows the receiver to decrypt the message regardless of whether any preceding messages were damaged by either line noise or an adversary's actions.

Let $\overline{C}$ be the ciphertext message that the receiving SCM sees. If $\overline{C}$ differs from C, this may be due to line noise or malicious actions of an adversary. A SCM receiving $\overline{C}$ formats the message into a sequence number, a sequence of cipher blocks, and a MAC as follows:

$$\overline{C} \quad = \quad \overline{i}\ \overline{c}_1\ \overline{c}_2\ \ldots\ \overline{c}_n\ \overline{a}$$

The SCM maintains a *receive sequence* state variable in order to record the sequence number of the last authenticated message that it received. The receive sequence variable is initialized to zero at session negotiation. Before decrypting the following ciphertext blocks, the SCM checks that the sequence number $\overline{i}$ contained in the message is greater than the SCM's receive sequence variable. If it isn't, the SCM discards the remainder of the message. This check ensures that an adversary cannot replay old messages. Provided the sequence number check

succeeds, the SCM decrypts the message as follows:

$$\begin{aligned} \overline{P}\ \overline{z} &= \overline{i}\ \overline{S}\ \overline{z} = \overline{i}\ \overline{p}_1\ \overline{p}_2\ \ldots\ \overline{p}_n \\ &\text{where} \\ \overline{p}_j &= \mathrm{D}_k\left[\overline{c}_j\right] \oplus \mathrm{E}_k\left[\overline{i}\ j\ 0 \ldots\right] \end{aligned}$$

The SCM forwards the decrypted plaintext blocks $\overline{p}_j$ to the SCADA system as soon as they are available, stripping the padding from the last block. Finally, the SCM computes the MAC for the message as follows:

$$\widetilde{a} = \mathrm{MAC}_{k'}\left[\overline{i}\ \overline{p}_1\ \overline{p}_2\ \ldots\ \overline{p}_n\right]$$

and compares it to the MAC $\overline{a}$ received with the message. If the two match, the SCM updates its receive sequence variable to the sequence number $\overline{i}$ of the received message, and otherwise it logs an error.

Our encryption algorithm is essentially a cascade cipher composed of two block ciphers, each using a different NIST-approved encryption mode [5]. The plaintext is first encrypted using *counter* (CTR) mode with a counter that depends on both the message sequence number and the block position within the message. The result is reencrypted using *electronic codebook* (ECB) mode. We call this combination *position embedding* (PE) mode since it embeds the position of a plaintext block into its corresponding cipher block. The properties of PE mode allow us to leverage the underlying SCADA CRC to assure data integrity.

3.3 SCADA Model

We model the operation of a SCADA unit receiving a decrypted SCADA message $\overline{S}$ with the total function PAYLOAD. This function checks the format of $\overline{S}$, calculates and verifies the CRC, and either returns a substring of $\overline{S}$ that represents the actual SCADA data with header and formatting stripped, or the distinguished token **error** indicating that the message was either incorrectly formatted or the CRC check failed. We assume that for all $\overline{S}$ for which PAYLOAD returns a non-**error** message substring, PAYLOAD returns either the same substring or **error** for all prefixes of $\overline{S}$. That is, PAYLOAD finds the shortest prefix of $\overline{S}$ that can be interpreted as a valid SCADA message. Extra bits beyond the end of that substring are either ignored or result in **error**. This assumption is realistic as shortest-prefix decoding corresponds to what most SCADA systems implement. Furthermore, any SCADA system in which a prefix of a valid message was also a valid message would be susceptible to the longer message being transformed into the shorter one by line errors.

3.4 Security

We consider the security of our protocol in the face of known plaintext attacks. We do not consider chosen plaintext attacks because the adversary's principal goal is to disrupt the operation of the SCADA system and the physical processes

it controls. If the adversary is able to inject chosen plaintext messages into the plaintext port of a SCM, the adversary has physical access to the SCADA system and can likely perform far greater disruption and damage by other means.

To assure the integrity of SCADA messages, our protocol must guard against an adversary injecting an unauthentic ciphertext message into the communications link, modifying a ciphertext message during its transmission, reordering messages, or replaying an old message. Forging and alteration are prevented by ensuring that an unauthentic ciphertext has a low probability of decrypting to a SCADA message containing a valid CRC. Reordering and replay are prevented by ensuring that an alteration of the sequence number will likewise result in a low probability that the ciphertext decrypts to a SCADA message containing a valid CRC. The following theorem captures this security property more precisely.

Theorem 1. *Let* **CP** *be a collection of corresponding ciphertext, plaintext message pairs defined as in Section 3.2. Let PAYLOAD, as defined in Section 3.3, return non-***error** *for each plaintext message, and utilize an h-bit CRC whose generating polynomial has the form* $g(x) = g_h x^h + g_{h-1}x^{h-1} + \cdots + g_1 x + g_0$ *where* $g_h = 1$ *and* $g_0 = 1$. *Let* $\mathrm{E}_k[\cdot]$ *and* $\mathrm{D}_k[\cdot]$ *be the encryption and decryption functions of the b-bit block cipher used to form the ciphertexts in* **CP**, *with* $b \geq h$. *Let* $\overline{C} = \overline{i}\ \overline{c}_1\ \ldots\ \overline{c}_{\overline{n}}$ *be a ciphertext message, different from any of the ciphertexts in* **CP**, *constructed by an adversary who knows* **CP** *but not the cipher key k. Decrypt* $\overline{C}$ *to* $\overline{P}\ \overline{z} = \overline{i}\ \overline{S}\ \overline{z} = \overline{i}\ \overline{p}_1\ \ldots\ \overline{p}_{\overline{n}}$ *where* $\overline{p}_j = \mathrm{D}_k[\overline{c}_j] \oplus \mathrm{E}_k\left[\overline{i}\ j\ 0 \ldots\right]$. *If* $\overline{P}$ *is not one of the plaintexts in* **CP**, *then PAYLOAD*$(\overline{S})$ *returns non-***error** *with probability at most* 2^{-h}.

The proof of this theorem relies on a new result on error detection by systematic shortened cyclic codes that differs from any of the results that we have found in the literature.

Lemma 1. *Let* **H** *be a systematic shortened cyclic binary code with generator polynomial* $g(x) = g_h x^h + g_{h-1}x^{h-1} + \cdots + g_1 x + g_0$ *where* $g_h = 1$ *and* $g_0 = 1$. *Let w be any bit string of length at least h. If any h consecutive bits of w are selected uniformly and independently at random, then w is a codeword of* **H** *with probability* 2^{-h}.

Proof of Lemma 1. First, consider the long division process that is used to encode or decode a message [6]. The function $\mathcal{M}_1(\cdot\,,\cdot)$ represents one intermediate step of the division process:

$$\begin{aligned}
&\mathcal{M}_1(b_{n-1}\cdots b_0\,,\ g_h\cdots g_0) = b_{n-1}\cdots b_0 && \text{if } n = h\\
&\mathcal{M}_1(b_{n-1}\cdots b_0\,,\ g_h\cdots g_0) = b_{n-2}\cdots b_0 && \text{if } n > h \text{ and } b_{n-1} = 0\\
&\mathcal{M}_1(b_{n-1}\cdots b_0\,,\ g_h\cdots g_0) = (b_{n-2} - g_{h-1})\cdots(b_{n-h-1} - g_0)\, b_{n-h-2}\cdots b_0 && \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{if } n > h \text{ and } b_{n-1} = 1
\end{aligned}$$

To encode a message bit string m, h zero bits are first appended to it, and then $\mathcal{M}_1$ is repeatedly applied until the result is h bits wide. Let $\mathcal{M}$ denote the

repeated application of $\mathcal{M}_1$ until the result is h bits wide (transitive closure). The concatenation $m\,\mathcal{M}(m\,0_{h-1}\cdots 0_0)$ is a codeword. A received bit string w is a codeword if and only if $\mathcal{M}(w) = 0_{h-1}\cdots 0_0$.

Fix l and consider the set of all bit strings $\{b_{h-1}\cdots b_0\,0_{l-1}\cdots 0_0\}$. This set has cardinality 2^h. We will show by induction on l that the set of remainders after repeated application of $\mathcal{M}_1$ has cardinality 2^h. The base case where $l = 0$ is straightforward. For the induction step, we need to show that the set $\{\mathcal{M}_1(b_{h-1}\cdots b_0\,0_{l-1}\cdots 0_0\,,\ g_h\cdots g_0)\}$ has cardinality 2^h. To see this, consider the two cases for the values of b_{h-1}. If b_{h-1} is 0, we have

$$\begin{aligned}\mathcal{X}_0 &= \{\mathcal{M}_1(b_{h-1}\cdots b_0\,0_{l-1}\cdots 0_0\,,\ g_h\cdots g_0)\}\\ &= \{b_{h-2}\cdots b_0\,0\,0_{l-2}\cdots 0_0\}\end{aligned}$$

and $\mathcal{X}_0$ has cardinality 2^{h-1}. If b_{h-1} is 1, we have

$$\begin{aligned}\mathcal{X}_1 &= \{\mathcal{M}_1(b_{h-1}\cdots b_0\,0_{l-1}\cdots 0_0\,,\ g_h\cdots g_0)\}\\ &= \{(b_{h-2}-g_{h-1})\cdots(b_0-g_1)\,(0-g_0)\,0_{l-2}\cdots 0_0\}\\ &= \{(b_{h-2}-g_{h-1})\cdots(b_0-g_1)\,1\,0_{l-2}\cdots 0_0\} \qquad \text{since } g_0 = 1\end{aligned}$$

and $\mathcal{X}_1$ has cardinality 2^{h-1}. Since $\mathcal{X}_0$ and $\mathcal{X}_1$ do not intersect, their union has cardinality 2^h.

Consider a bit string $w = \overline{w} + (b_{h-1}\cdots b_0\,0_{l-1}\cdots 0_0)$ where $|w| = |\overline{w}| \geq h+l$ and bits $b_{h-1},\ldots,b_0$ are selected uniformly and independently at random. Then

$$\begin{aligned}\mathcal{M}(w) &= \mathcal{M}(\overline{w} + (b_{h-1}\cdots b_0\,0_{l-1}\cdots 0_0))\\ &= \mathcal{M}(\overline{w}) + \mathcal{M}(b_{h-1}\cdots b_0\,0_{l-1}\cdots 0_0).\end{aligned}$$

Now, $\mathcal{M}$ takes each element of $\{b_{h-1}\cdots b_0\,0_{l-1}\cdots 0_0\}$ to a different image, by the cardinality argument above. Hence $\mathcal{M}(b_{h-1}\cdots b_0\,0_{l-1}\cdots 0_0)$ is uniformly distributed. Hence $\mathcal{M}(w)$ is uniformly distributed. Since $\mathcal{M}(w)$ has h bits, the probability that w is a valid codeword is 2^{-h}. □

CRCs are systematic shortened cyclic codes. Figure 3 gives the generator polynomials for a number of CRC codes in widespread use, and all of those we have encountered satisfy the prerequisites of Lemma 1. With Lemma 1 in hand, we return to the proof of our security theorem.

Proof of Theorem 1. Not knowing the encryption key, an adversary is limited to constructing an unauthentic message $\overline{P}$ by choosing ciphertext blocks at random, or by using ciphertext blocks from **CP** and (*i*) modifying bits in ciphertext blocks, (*ii*) changing the sequence number, and/or (*iii*) splicing together ciphertext blocks from messages in **CP**, including reordering ciphertext blocks, deleting ciphertext blocks, and inserting ciphertext blocks. We first show that some of these cases lead directly to PAYLOAD returning **error** for the decrypted message, while the remaining cases lead to at least one plaintext block of the decrypted message being randomized.

Choosing ciphertext blocks at random or modifying bits in a ciphertext block randomizes the corresponding decrypted plaintext block, due to real-or-random

CRC Code	generator polynomial
CRC-4	$g(x) = x^4 + x^3 + x^2 + x + 1$
CRC-7	$g(x) = x^7 + x^6 + x^4 + 1$
CRC-8_A	$g(x) = x^8 + x^7 + x^6 + x^4 + x^2 + 1$
CRC-8_B	$g(x) = x^8 + x^5 + x^4 + 1$
CRC-12	$g(x) = x^{12} + x^{11} + x^3 + x^2 + x + 1$
CRC-ANSI	$g(x) = x^{16} + x^{15} + x^2 + 1$
CRC-CCITT	$g(x) = x^{16} + x^{12} + x^5 + 1$
CRC-SLDC	$g(x) = x^{16} + x^{15} + x^{13} + x^7 + x^4 + x^2 + x + 1$
CRC-24	$g(x) = x^{24} + x^{23} + x^{14} + x^{12} + x^8 + 1$
CRC-32 (IEEE 802.3)	$g(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

Fig. 3. Generator Polynomials for Popular CRC codes [6]

indistinguishability of the cipher used in the term $\mathrm{D}_k\,[\overline{c}_j]$ in the decryption formula.

Changing the sequence number $\overline{i}$ randomizes the term $\mathrm{E}_k\left[\overline{i}\ j\ 0\ldots\right]$ in the decryption formula, due to the security of the cipher used in this term, and thus randomizes every block of decrypted plaintext.

Using blocks from different messages or moving blocks to different positions within a message again randomizes the term $\mathrm{E}_k\left[\overline{i}\ j\ 0\ldots\right]$ in the decryption formula, and thus randomizes the decrypted plaintext of any ciphertext blocks from messages other than $\overline{i}$ or ciphertext blocks from $\overline{i}$ that do not occupy their original message positions. Deleting ciphertext blocks from the end of a message shortens the decrypted SCADA message, but by the assumptions in the definition of PAYLOAD, PAYLOAD returns **error** on such a shortened message. Adding ciphertext blocks to the end of a message lengthens the decrypted SCADA message, but again by definition PAYLOAD returns either the same result or **error** for such a lengthened message. Modifying the last ciphertext block can shorten or lengthen the message by changing the padding bits, but again PAYLOAD returns either the same result or **error**. Modifying the last ciphertext block which contains the padding bits could also result in fewer than h bits of the message being randomized. If this is the only portion of the message that is randomized, we can consider this as a burst error of width less than h. An h-bit CRC detects all such cases [6], and thus PAYLOAD returns **error**.

Now, if one or more plaintext blocks of $\overline{S}$ are randomized, we consider the randomized blocks as errors introduced into an originally correct plaintext message. Since $b \geq h$, there is at least one span of h consecutive bits that are randomized. Thus, provided PAYLOAD finds $\overline{S}$ to be correctly formatted, the CRC is correct with probability 2^{-h} by Lemma 1. □

3.5 Latency

We discuss latency in terms of the time required to transmit a character over the long-haul SCADA communications line. For a typical SCADA configuration of 1200 baud, 8 data bits, 1 start bit, and 1 stop bit, transmitting one character takes 8.3 milliseconds. We assume that the plaintext and ciphertext ports of both SCMs all use the same data rate, and that the SCADA system delivers SCADA messages that are free of gaps. We also assume that SCMs have enough computing power that encryption and decryption have no significant impact on latency.

On the sending side, the SCM must wait to receive an entire block of plaintext from the sending SCADA device before it can encrypt and begin transmitting ciphertext. This requirement introduces a delay of $\frac{b}{8}$ character times, *e.g.* 16 character times for AES with $b = 128$. However, the SCM can begin transmitting the SCM message header, containing the sequence number i, as soon as it receives the first character of the SCADA message. Provided this header is shorter than the cipher block length, the SCM will complete transmitting the header before it receives enough plaintext characters to encrypt the first block. The transmission time for the header is thereby entirely masked by the time required to receive the first block of plaintext. Thus the latency introduced at the sender is exactly $\frac{b}{8}$ character times.

On the receiving side, the SCM must wait to receive an entire block of ciphertext before it can decrypt and begin forwarding the corresponding cleartext. Again, this requirement introduces a delay of $\frac{b}{8}$ character times. Receiving and checking the MAC in the trailer is performed after (or perhaps during) the forwarding of the decrypted SCADA message to the SCADA system, and hence introduce no additional latency. The total delay introduced at the receiver is thus $\frac{b}{8}$ character times.

In sum, our protocol introduces a fixed latency of $2 \cdot \frac{b}{8}$ character times, regardless of the length of the SCADA message. For AES, this is 32 character times.

4 Implementation

As observed earlier, our position embedding encryption mode is essentially a cascade cipher composed of an ECB-mode cipher and a CTR-mode cipher. Since these block ciphers use the same algorithm and key, during encryption they can share the same special-purpose cipher unit in a hardware-based implementation, or the same state variables and key expansion in a software implementation. During decryption, however, the CTR-mode cipher is used in encryption mode while the ECB-mode cipher is used in decryption mode, and this may preclude sharing hardware units or software modules.

It is possible for both the sender and receiver to optimistically perform CTR-mode encryptions for several blocks of several future messages in advance of the receipt of those messages. While we expect even the least capable of the current

generation of embedded processors to provide more than adequate performance for cryptographic operations at SCADA communications rates, this optimization could be useful in other applications.

A Java implementation that codifies many of the details we expect to appear in the AGA recommendation is available as open source [7]. This implementation supports several *cipher suites*, of which one uses the position embedding mode described in this paper. The implementation also includes a cipher suite that relies on the MAC rather than the SCADA CRC to assure integrity, at the cost of requiring the receiver to verify the MAC before delivering the deciphered SCADA message. This cipher suite provides stronger security at the expense of latency, and should be used in deployments where the additional latency can be tolerated. In the future, additional cipher suites may be defined to support different key lengths, MAC lengths, and encryption algorithms.

5 Related Work

Stream ciphers are particularly susceptible to known-plaintext active attacks, and are thus unsuitable for protecting SCADA communications without the additional protection of a MAC. This includes block ciphers used with CTR mode alone. An adversary who knows the plaintext corresponding to an encrypted message can recover the key stream, and then replace the message with a different one that decrypts to a plaintext of his choosing. Even with only partially-known plaintext, the linear nature of CRCs allows an adversary to patch up a CRC underneath a stream cipher by performing operations on the encrypted stream. These problems are well known vulnerabilities in the WEP protocol [8].

Stubblebine and Gligor [9] show how blocks from messages encrypted with a block cipher in CBC mode may be spliced together to form unauthentic messages. Their attack applies even when the plaintext includes a CRC. Thus block encryption with CBC mode alone provides inadequate integrity protection.

Our work has a similar goal to that of *non-malleable cryptography*, which seeks to ensure that given a ciphertext, it is impossible to generate a different ciphertext so that the respective plaintexts are related [10]. Our system achieves a weaker property. While the plaintexts may be related, they will be sufficiently unrelated that the CRC is likely to fail.

Beaver *et al.* [11] describe an encryption scheme that uses the internal state of the cipher to obtain authentication. Authenticated encryption solves a different problem than we are concerned with, namely that of computing authentication cheaply in parallel with encryption. However, the internal properties of their cipher appear to provide similar randomization of the plaintext when the ciphertext is modified, and thus their cipher may be viable alternative to our scheme. Ours has some advantage in being built entirely from NIST approved primitives.

Gligor and Donescu [12] propose an encryption and authentication mode called XCBC that requires only a non-cryptographic integrity check such as a CRC to assure message integrity. Their method may also be a viable alternative

to our scheme, however the fact that it is patented could deter broad acceptance of a standard that employed it.

A good deal is known about the properties of cyclic and shortened cyclic codes for detecting various kinds of errors, including burst errors [13] and errors over a binary symmetric channel [14]. However, neither of these types of errors precisely matches the types of errors that an adversary's actions can introduce with our encryption scheme. In particular, the classic results on burst errors, which are sometimes stated imprecisely, apply to errors that occur as one single consecutive string of randomized bits. In our situation, an adversary's actions can produce bursts of error bits separated by segments of non-error bits, to which the classic results do not apply. Our Lemma 1 appears to represent a new result on error detection.

6 Conclusion

The American Gas Association (AGA) develops and publishes standards for the gas industry. In February 2004 the AGA 12 task group distributed for ballot a draft of the first AGA 12 recommendation for protecting SCADA communications [15]. This draft describes general requirements for a solution, and a subsequent recommendation will specify a protocol in detail. Several vendors are planning to build and market SCM devices that implement the final standard. These devices will likely be targeted for use not only in the gas industry but in other industries such as power, oil, and water. It is our hope and expectation that these devices will be widely deployed before a significant cyberterrorism incident makes their need all too evident.

Acknowledgments. We would like to acknowledge the leadership of Bill Rush of the Gas Technology Institute in the AGA-12 effort, the many individuals who have contributed to the AGA-12 standards effort, and Mark Torgerson and Erik Anderson at Sandia National Labs who provided valuable advice and critique.

References

1. Gas Technology Institute: http://www.gtiservices.org/security. (2004)
2. National Institute of Standards and Technology: Federal Information Processing Standards Publication 197 (FIPS PUB 197), Advanced Encryption Standard (AES). (2001)
3. Bellare, M., Desai, A., Jokipii, E., Rogaway, P.: A concrete security treatment of symmetric encryption: Analysis of the DES modes of operation. Proc. 38th Annual Symposium on Foundations of Computer Science (1997)
4. Menezes, A.J., Oorschot, P.C.V., Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press (1997)
5. National Institute of Standards and Technology: NIST SP 800-38A 2001 ED, Recommendation for Block Cipher Modes of Operation. (2001)
6. Wicker, S.B.: Error Control Systems for Digital Communication and Storage. Prentice Hall (1995)

7. Wright, A.K.: http://scadasafe.sourceforge.net. (2004)
8. Borisov, N., Goldberg, I., Wagner, D.: Intercepting mobile communications: The insecurity of 802.11. Proc. MOBICOM (2001)
9. Stubblebine, S.G., Gligor, V.D.: On message integrity in cryptographic protocols. Proc. 1992 IEEE Symposium on Research in Security and Privacy (1992) 85–104
10. Dolev, O., Dwork, C., Naor, M.: Non-malleable cryptography. Proc. 23rd ACM Symposium on Theory of Computing (1991)
11. Beaver, C., Draelos, T., Schroeppel, R., Torgerson, M.: ManTiCore: Encryption with joint cipher-state authentication. IACR Preprint, 2003/154, www.iacr.org (2003)
12. Gligor, V.D., Donescu, P.: Fast encryption and authentication: XCBC encryption and XECB authentication modes. Presented at the 2nd NIST Workshop on AES Modes of Operation, Santa Barbara, CA (2001)
13. Peterson, W.W., Weldon, E.J.: Error correcting codes. MIT Press, Cambridge, MA (1972)
14. Witzke, K.A., Leung, C.: A comparison of some error-detecting CRC code Standards. IEEE Trans. Commun. COM-33, No. 9, 996 (1985)
15. AGA 12 Task Group: Cryptographic protection of SCADA communications: General recommendations (2004)