6-1-2010

Predictive YASIR: High Security with Lower Latency in Legacy SCADA

Rouslan V. Solomakhin
Dartmouth College

PREDICTIVE YASIR: HIGH SECURITY WITH LOWER LATENCY IN LEGACY SCADA

A Thesis

Submitted to the Faculty

in partial fulfillment of the requirements for the

degree of

Master of Science

in

Computer Science

by

Rouslan V. Solomakhin

DARTMOUTH COLLEGE

Hanover, New Hampshire

June 2010

Dartmouth Computer Science Technical Report TR2010-665

Abstract

Message authentication with low latency is necessary to ensure secure operations in legacy industrial control networks, such as power grid networks. Previous authentication solutions by our lab and others looked at single messages and incurred noticeable latency. To reduce this latency, we develop Predictive YASIR, a bump-in-the-wire device that looks at broader patterns of messages. The device (1) predicts the incoming plaintext based on previous observations; (2) compresses, encrypts, and authenticates data online; and (3) pre-sends a part of ciphertext before receiving the whole plaintext. I demonstrate the performance properties of this approach by implementing it in the Scalable Simulation Framework and testing it on Modbus/ASCII protocol, which is widely used in the power grid, oil and gas, manufacturing, and water treatment control networks. By looking at broader message patterns and using predictive analysis, my results demonstrate a $15.48 \pm 0.35\%$ improvement in latency over the previous most efficient solution. The simulation code is available from `http://www.cs.dartmouth.edu/~pyasir/`.

Acknowledgements

I adopted parts of the thesis from my submission to the Fourth Annual IFIP Working Group 11.10 International Conference on Critical Infrastructure Protection [25]. These materials are based upon work supported by the National Science Foundation under grant CNS-0524695 and by the Department of Energy under Award Number DE-OE0000097. I am grateful to Paul Myrda of Electric Power Research Institute for providing the SCADA trace and to Andrew K. Wright for suggesting semi-embedded devices. I appreciate the support of my thesis committee: Sergey L. Bratus, Andrew T. Campbell, and Sean W. Smith.

Disclaimer

Contents

List of Figures

Chapter 1

Introduction

Electric utilities use networking to control power generation and distribution. Because electricity cannot be stored efficiently and must be directed to recipients immediately, the power grid is in a delicate, near real time balance. A minor disturbance could cause severe repercussions. The United States built the power grid half a century ago, when network-based attacks were rare. New threats warrant retrofitting security into the legacy network of the power grid. Protecting a legacy network is difficult, however, because the critical infrastructure components must communicate fast, but security slows them down. In this work, I develop an approach to improve the performance of the previous fastest security solution.

Power utilities monitor and control the power grid through a partially unsecured slow legacy network. This network connects substations and control centers. In a control center, human operators ensure safe and continuous operation of the grid by monitoring data terminals. A terminal provides a visual representation of the data that it receives from a Front End Processor (FEP), which exchanges messages with Data Aggregators (DA) in substations. A FEP and a DA connect via a slow legacy point-to-point network, which is often unsecured [6].

For example, a nearby hydro-electric power station consists of a control center

on the bank and a substation on the dam. The substation communicates with other substations on the river via microwave radio in unsecured Distributed Network Protocol v3 (DNP3) [4].

Because of the unsecured communication, an adversary can insert messages into the traffic to impersonate any device on the network. For example, an adversary can impersonate a FEP and command a DA to perform tasks that it should not do in normal operation. The adversary can either replay an "increase power output" message from the FEP multiple times or increase the value in the message "set power output to 10 MW," which would overload the substation, possibly causing a rolling blackout in the power grid. If the adversary impersonates a DA, then the adversary can replay old DA status messages to the FEP, which would forward these messages to operator's terminal. Since the terminal would be receiving old status messages, it would not reflect the abnormalities in the power grid, and the operator would have a hard time detecting the attack. Even if the operator discovers abnormalities through an alternative channel, understanding the scope of the problem would be difficult in absence of correct data on the terminal, which would slow down the operator's reaction to the attack. This reaction delay would buy time for the adversary to subvert more substations. Because the substations can be manipulated remotely, authorities would struggle to find the attacker, and the security cameras would not record a trespasser.

FEPs and DAs send queries and replies to each other and act upon received messages with an assumption that everything they receive is authentic. If an attacker modifies a message in transition or sends an extra message to an end-point device, then this message is not authentic, but the legacy devices have no way of distinguishing bogus and authentic data. To prevent malicious attacks, FEPs and DAs must be able to make a distinction between bogus and authentic data and act only upon authentic messages, or *authenticate* their communications. Message authentication involves a modification of the message format, which typically makes the messages longer. The devices in

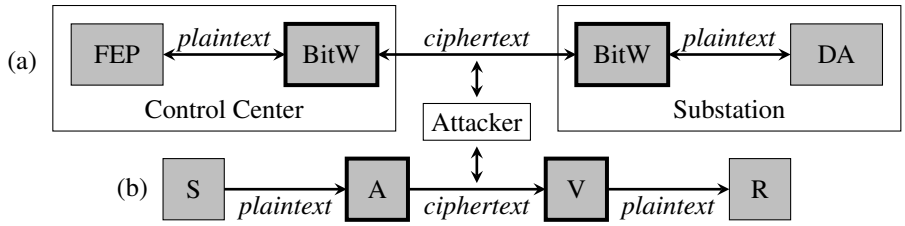

Figure 1.1: Bump-in-the-wire devices protect the end-point devices from a malicious attacker by converting plaintext into ciphertext. (a) A typical setup with a Front End Processor, a Data Aggregator and two bump-in-the-wire devices. (b) A useful abstraction with a Sender, an Authenticator, a Verifier, and a Receiver.

the control center and the substation must have some logic to authenticate messages. Although a utility company can upgrade its FEPs and DAs to more secure versions with built-in message authentication, representatives of the power industry tell our lab that these devices are prohibitively expensive and sometimes even custom-made. More importantly, the upgraded devices would have to communicate over the slow legacy network, because a network upgrade is often expensive and sometimes infeasible. Without thoughtful consideration of the constraints of a slow legacy network, new logic for handling messages and extended message formats can lead to longer delays in controlling critical infrastructure components, which is unacceptable for managing the power grid. The cheaper and more efficient option is an external bump-in-the-wire device (BitW) [23, 28, 31, 33] that authenticates all messages that it intercepts on the network, without the need to upgrade either the end-point devices or the slow legacy network.

Two BitWs work in concert to authenticate a message (Figure 1.1a). Before a message from the FEP leaves the control center, the authenticator BitW reformats the original *plaintext* message into a *ciphertext* message with added counter and a redundancy, or a *digest* of the message. The counter stops an attacker that attempts to replay an old message. The digest depends on the message, the counter, and a key shared by the pair of BitWs, a *digest key*. Due to the cryptographic properties of the mechanism to generate the digest, it is intractable for an adversary without the digest key to construct the correct digest for an altered ciphertext. When the ciphertext arrives in the substation,

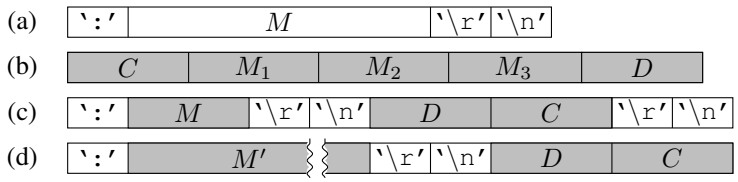

Figure 1.2: Message formats for Modbus/ASCII. (a) Plaintext. (b) PE ciphertext in blocks. (c) YASIR ciphertext. (d) Predictive YASIR ciphertext. The shaded areas are modified or generated by a BitW. The symbols C and D denote the counter and digest.

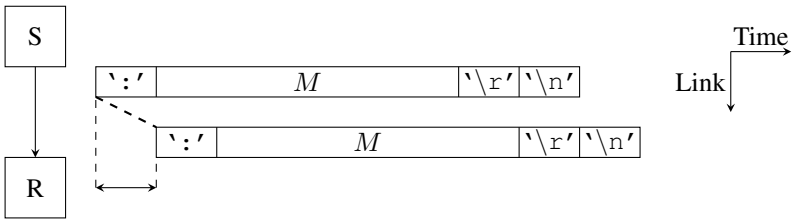

Figure 1.3: Transmission latency without authentication. Sender S sends a message to the receiver R over a slow, legacy, point-to-point link.

the verifier BitW compares its own calculation of the ciphertext digest with what it has received. If the two digests match, the verifier reformats the ciphertext into a plaintext message and forwards it to the DA.

Note that a BitW is still susceptible to attacks if the adversary gains access to a FEP or a DA prior to the BitW.

When a DA sends a message to a FEP, the authenticator and the verifier switch roles. Due to this symmetry, I prefer to use the words "sender" and "receiver" instead (Figure 1.1b). I may use the word "device" to refer to any sender, authenticator, verifier, or receiver. In the figures, I denote the devices with letters S, A, V, and R.

In the slow legacy power grid network, the end-to-end latency of a message typically should not exceed a certain bound to ensure the correct operation of the controlled devices. These devices are in a delicate balance, and introducing a significant delay into the network is not acceptable. Figure 1.3 shows the end-to-end latency of a message without security features. I use the diagram style from the YASIR paper [28]. Because of the low bandwidth, a BitW should not wait to receive the whole message before

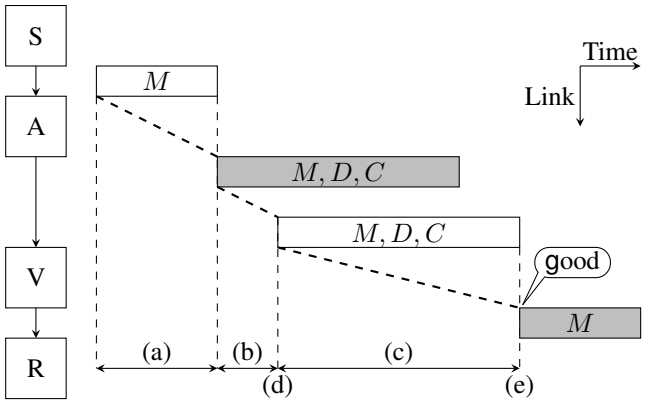

Figure 1.4: Latency with hold-back. Both BitWs hold-back the whole message before forwarding it. The shaded areas are modified or generated by a BitW. Hold-back delays a message by (a) and (c). The transmission delay is (b). The verifier starts receiving the message at time (d), but starts to forward it to the receiver only at time (e), when it has checked that the digest is correct. If an attacker modifies the message, the verifier drops it.

processing it, a practice known as *hold-back* (Figure 1.4). For example, suppose the latency bound is 300 ms. A millisecond equals the time in which a device transmits 0.9 bytes, or 0.9 *byte-times*, on a network with a bandwidth of 9600 baud with 10-bit bytes. If both BitWs in a pair hold-back a message that is longer than 144 bytes, the delay would exceed the 300 ms bound. In contrast, a BitW should forward each byte quickly, or process the message *online*. The online processing must thwart an attacker that attempts to either replay or modify ciphertext.

I consider how online processing handles each type of attack in turn.

Tsang and Smith [28] demonstrated that an online BitW can stop an attacker that attempts to replay an old message, a *replay-attack*, without message delay. The delay is absent because, although the authenticator transmits the counter at the end of the ciphertext, the verifier forwards the whole message to the receiver before checking the counter. Since the counter affects the digest, the verifier increments the counter from the previous message to calculate the new value. If the calculated value is larger than what the verifier receives, it ignores the received counter. If the calculated value is smaller than what the verifier receives, it sets its own counter to the received value to

synchronize with the authenticator. Before the counter overflows, the BitW pair resets its value to zero and changes the digest key.

To stop an attacker that attempts to modify a message, a BitW pair appends a digest after the message, but before the counter. This digest depends on the message, the counter, and the digest key. The verifier compares the received digest to what it calculates itself. If the two digests match, then the verifier forwards the message to the receiver. Intuitively, one might think that the verifier cannot process the ciphertext online, that it must wait until it receives the whole digest before it can forward the message to the receiver, thus incurring a byte-time of latency for each byte in the message. Although some prior approaches do this, Wright et al. [33] suggested to forward the message as soon as the verifier receives it, but to introduce a Cyclic Redundancy Check (CRC) error if an attacker modifies the message, thus exploiting the receiver's ability to detect random errors (Figure 2.1). I use a similar technique to let the verifier process the message online (Figure 3.2).

When an authenticator appends the digest to the message, it must ensure that the verifier can distinguish between them. It has two options. The first option is to prepend the message length to the ciphertext, but a common protocol like Modbus/ASCII (Figure 1.2a) has variable length messages and does not specify length in the message. To find the length of a message in this protocol, an authenticator must hold-back the full message. The second option is online: delimit the ciphertext parts with a *message-digest* separator. If the separator appears in the message data, then the authenticator marks it with a special symbol to avoid confusing the verifier, i.e., the authenticator *escapes* the separator within the message. Modbus/ASCII has a *message-end* indicator that can be used as the message-digest separator. In general, new separators and escapes delay a message, but do not improve its authenticity. At first glance, they are an encoding inefficiency that an online authenticator must include. As part of my work, I aim to eliminate these inefficiencies in online processing.

Organization. The rest of the paper comprises six chapters. In Chapter 2, I review related work. Then my approach is briefly explained in Chapter 3. In Chapter 4, I describe my approach in detail and outline the test methodology. I show the evaluation results in Chapter 5. In Chapter 6, I list future research directions. Finally, I draw conclusions in Chapter 7.

Chapter 2

Related Work

2.1 Legacy Network

Recall that power grid network upgrade is expensive and sometimes infeasible. If a utility does not upgrade the network, then the end-point devices cannot increase the network bandwidth due to the fundamental limits on the amount of information that a device can transmit on a communication line [18, 19, 24].

End-point devices send information by changing the power, frequency, or phase of the electric signal on a communication line with time. Assume, for the sake of discussion, that a SCADA device transmits information by varying levels of power and keeps the frequency and phase constant. For example, let 1 watt indicate bit 0 and 2 watts indicate bit 1. The receiver must distinguish the two different levels of power to decode either binary digit.

A communication line has maximum levels of power and frequency that it can handle. If a transmitter increases the power or frequency above the maximums, then the line may melt or exhibit corona discharge and arcing. Also, the receiver must be able to distinguish the different levels of power and frequency, and so the transmitter cannot subdivide the signal into infinitely many levels. Given these physical properties,

we can calculate the maximum amount of information that we can send over a communication line. Nyquist showed that the maximum bit rate achievable on a noiseless communication line is:

$$\text{noiseless maximum bit rate} = 2H \log V \text{ bits/second},$$

where H is the frequency of the signal and V is the number of power levels of the signal [18, 19, 27]. Unfortunately, it is difficult to approach the maximum bit rate due to noise. Shannon showed that the maximum bit rate achievable on a noisy communication line is:

$$\text{noisy maximum bit rate} = H \log \left(1 + \frac{S}{N}\right) \text{ bits/second},$$

where H is the frequency of the signal, S is the power of the signal, and N is the power of the noise [24, 27]. As the transmitter increases the bit rate, each bit becomes a less unique signal and indistinguishable under noise. This causes the receiver to make many mistakes when decoding bits, or have a high *bit error rate*. To reduce the bit error rate, the transmitter can use M-ary transmission to approach the maximum bit rate [21]. This method buffers multiple bits together and sends them out as one signal. In particular, suppose the transmitter buffers 8 bits, which can take any of $2^8 = 256$ possible values. Then the transmitter sends a signal at one of 256 possible levels of power. With constant frequency, such 256-ary transmission increases the distance between the signals 8 times, reducing the possibility of noise corrupting the signal on the line.

Thus it is important to conserve bandwidth on legacy power grid networks that have a built-in maximum bit rate and cannot be upgraded.

2.2 Compression

Performance is important on legacy, low-bandwidth communication lines. To avoid performance degradation due to new features, computer scientists have considered compression. For example, Jacobson describes how to compress TCP/IP headers to improve interactive responsiveness of slow point-to-point links [11]. A single character sent over a TCP/IP link results in a 41 byte packet, which is acknowledged by the receiver with another 41 byte packet. An 82 byte overhead can be prohibitive on a slow link, because an interactive system should have 0.1 to 0.2 response delay [15]. To reduce the TCP/IP header overhead, Jacobson devises a heuristic to compress TCP/IP headers into as few as 3 bytes. The heuristic relies on statefullness of TCP/IP connections and the slow rate of change of header information within a connection. These properties let Jacobson compress a TCP/IP header into a connection identifier, a bitmask of changed fields, and the delta of changed fields. The compressor is a driver in the networking stack, which is an embedded solution, not a bump-in-the-wire device.

Jacobson's scheme is programmed compression. It provides the best compression ratio at the cost of development time and inapplicability to other types of data. A more universal technique is described by Welch, Ziv, and Lempel [30, 34]. They rely on repeated patterns in data to build a mapping from data subsequences to the compression alphabet. The result is this mapping, or table, and the data expressed in compression alphabet. A useful property of the algorithm is that the table contains a prefix for each subsequence within the table. Whereas a naive decompressor would need the table to reverse look up data based on the compression alphabet, a more efficient approach is to build the reverse-look-up table on the fly. Both compression and decompression in this scheme are stream-wise operations, which is important for keeping the overhead low.

Colleagues in the power grid industry report that even two second delay is too long for a physical process to respond to an operator's command. To reduce the overhead of security in legacy power grid networks, we can use bump-in-the-wire devices.

2.3 Bump-in-the-Wire Devices

Colleagues in the power sector inform our lab that message integrity is more important than confidentiality, because an attacker may learn the state of the system from the physical world. For example, an adversary may see the open flood gates of a dam and deduce that the control station is sending an "open flood gates" message, and the substation is sending a "flood gates open" message. In contrast, without measures to ensure message integrity, an adversary may cause actions—such as opening the flood gates or shutting down a generator—with significant negative repercussions. If a utility needs to also hide the content of its messages, a BitW can provide zero-latency confidentiality through a stream cipher, such as AES in counter mode, which encrypts a plaintext stream by exclusive-or with a pseudo-random stream. Therefore, I consider related BitW solutions only if they authenticate messages: SEL-3021-2, AGA SCM, and YASIR. (I ignore such solutions as SEL-3021-1 from Schweitzer Engineering Laboratories [9] and SCADAsafe from RealTime Interactive Systems Corporation [3], because they do not protect message integrity. I also do not consider SSCP embedded device from Pacific Northwest National Laboratory [13], because it is not a bump-in-the-wire device.)

SEL-3021-2

SEL-3021-2 [23] is a commercial off-the-shelf BitW from Schweitzer Engineering Laboratories. The device uses the Message Authentication Protocol (MAP) [22] to provide integrity with HMAC-SHA-1 or HMAC-SHA-256 digest. The specifications omit the numbers on how much this device delays a message, instead recommending to not use SEL-3021-2 if low latency is desired.

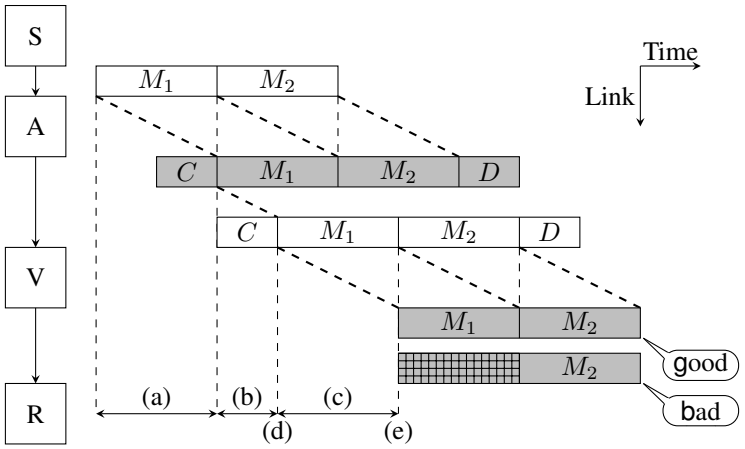

Figure 2.1: Latency in PE mode. The parts M_1 and M_2 are blocks of message M. Both BitWs buffer a 16-byte block of a message before forwarding it, delaying the message by 32 byte-times, denoted (a) and (c). The verifier starts receiving the message at time (d) and starts forwarding it to the receiver at time (e). If an attacker attempts to modify a block in the message, the verifier unconsciously scrambles the block (crossed out), which the receiver detects by checking the CRC at the end of the message.

AGA SCM

AGA SCM (American Gas Association SCADA Cryptographic Module) [31] is a BitW proposed by AGA 12 Task Group. The group members have developed a reference implementation [32], which can use several modes with hold-back and one online mode. The hold-back modes buffer the whole message before checking the digest and sending the message on, slowing down the message by the time that is linear in its size (Figure 1.4). The online mode is Position Embedding (PE) [33], which is a modified version of AES in counter mode (AES-CTR) followed by a standard version of AES in electronic code book mode (AES-ECB) (Figure 2.2). PE mode delays a message by 32 byte-times, because both BitWs in a pair buffer 16-byte blocks of data to apply AES-ECB (Figure 2.1).

AGA 12 modifies the way AES-CTR generates counters. First, the authenticator increments a 14-byte *session clock* by one every r microseconds, where r is termed *counter resolution*. Second, the authenticator concatenates the session clock with a 2-byte block counter. The authenticator sets the block counter to zero at the beginning

of each message and increments it by one for each block in the message. Finally, the authenticator encrypts the resulting 16-byte counter value and applies exclusive-or operation to the encrypted counter and a plain text block, as the standard AES-CTR does. To avoid using the same counter for two messages, AGA 12 requires to set counter resolution such that an authenticator can send at most one message in a single session clock tick.

For message integrity, PE mode relies on CRC. Note that a CRC in plaintext protects from random errors, but not from malicious attacks on message integrity. A BitW cannot protect message integrity with AES-CTR or AES-ECB alone, either. The counter mode is malleable [5, 14], i.e., an adversary can modify the ciphertext and keep the CRC valid, even without learning the encryption key [2].

The electronic code book mode is vulnerable to a *known-plaintext* attack, where an adversary that knows the plaintext of two messages can splice their parts into a third message, if the CRC of the new message equals the CRC of one of the original messages.

PE mode prevents splicing and predictable changes to ciphertext by combining the counter and electronic code book modes of encryption. The cryptographic community is wary of using such combinations with CRC for message integrity, however, because similar modes have been shown to be insecure before. One example is cipher block chaining mode (CBC), which was shown to not provide message integrity protection with a CRC. This result was demonstrated by Stubblebine and Gligor [26], who exploited predictability of CRC to create undetectable bogus messages for Kerberos V5 [12] and DCE RPC [20].

Thus, PE mode depends on the nonmalleability of its ciphertext: if an adversary changes the ciphertext, it should be impossible to predict what happens to the CRC. If an adversary inserts, removes, or reorders blocks, then the verifier BitW should scramble the plaintext in the CTR step. If an adversary flips a bit, then the verifier BitW

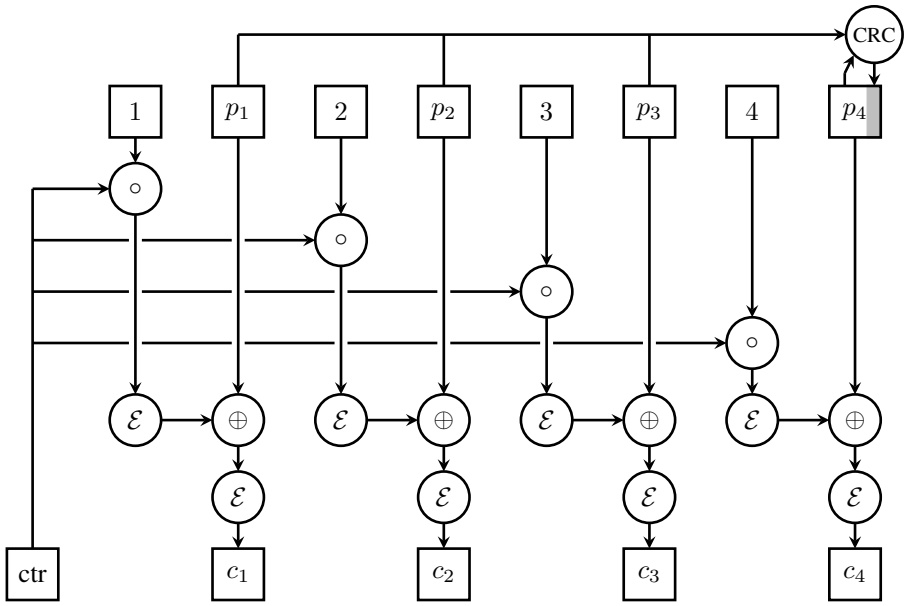

Figure 2.2: PE mode is AES-CTR, followed by AES-ECB with the same key. This mode relies on CRC to authenticate messages. The symbols $\circ$ and $\oplus$ denote concatenation and exclusive-or. The symbol $\mathcal{E}$ is the encryption function. The plaintext is $p_1 \circ p_2 \circ p_3 \circ p_4$, and the ciphertext is $\text{ctr} \circ c_1 \circ c_2 \circ c_3 \circ c_4$. Each block p_i and c_i is 16 bytes long. The message counter ctr is 14 bytes long, and the block counters (here 1–4) are 2 bytes long. Depending on the protocol, the CRC is from 2 to 4 bytes long.

should scramble the plaintext in the ECB step. Because of such scrambling, the receiver detects a CRC error. The probability of this error is 2^{-h}, where h is the length of the CRC. Different variants of CRC vary in length between 8 and 32 bits, but AGA specifies to use this mode when the CRC is at least 16 bits.

I developed a weak attack on integrity of messages in PE mode. (A history of attacks on encrypted CRCs suggests that a stronger attack may be possible.) Given PE mode ciphertext, an adversary can truncate the end of the ciphertext by an arbitrary number of blocks, and the verifier BitW would not scramble the message. Such truncation results in a valid plaintext for a protocol that does not specify message length and uses silence on the wire to delimit messages. One such protocol is Modbus/RTU. The receiving SCADA device accepts the plaintext message if it contains a valid CRC at the end of the remaining part, and so the adversary must know the plaintext contents of the message to make this attack work. Suppose an adversary truncates the ciphertext $c_1 \circ c_2 \circ c_3 \circ c_4$

into $c_1 \circ c_2 \circ c_3$. The receiving SCADA device would accept the plaintext message if the last bytes of the plaintext block p_3 contain a valid CRC for the plaintext $p_1 \circ p_2 \circ p_3$.

The simple way to prevent this attack is to use a cryptographic digest to verify message integrity. PE mode already uses a digest to detect an incorrect counter, but does not directly prevent the receiving SCADA device from acting upon a fabricated message. Instead of this behavior, PE mode could directly insert an incorrect CRC into the message when it detects an incorrect digest.

YASIR

Our lab's YASIR [28] is a BitW that authenticates messages with $\leq$18 bytes-times of overhead (Figure 1.2c). The actual overhead depends on the underlying protocol. With Modbus/ASCII, YASIR delays a message by $\sim$16 byte-times (Figure 2.3). The delay comprises the 12 bytes of HMAC-SHA-1-96 digest for data integrity, 2 bytes for the authenticator to detect the end of the message, and 2 bytes for the verifier to have an opportunity to control the message CRC. Building on the ideas from PE mode [31], YASIR turns malicious errors into random ones by sending an incorrect CRC to the receiver if the digest is invalid. In contrast to PE mode, YASIR delays a message by fewer byte-times and authenticates a message with a fully standard and accepted cryptographic technique [29].

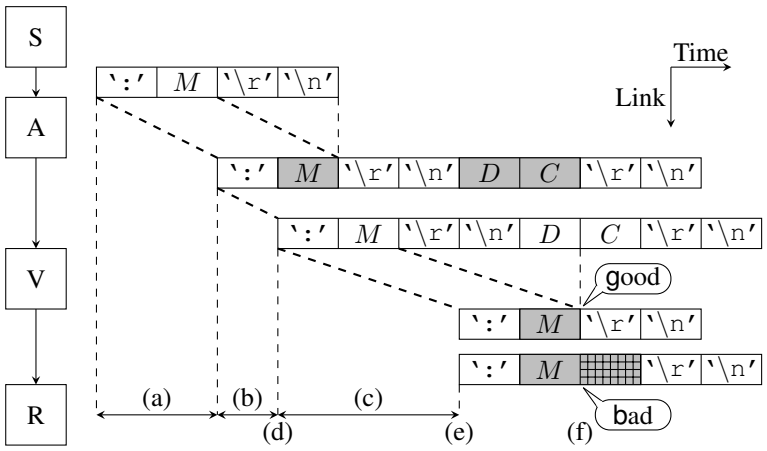

Figure 2.3: Latency with YASIR. The authenticator buffers 2 bytes of the message to detect its end. The verifier buffers 14 bytes of the message to verify its digest. Total latency is 16 bytes-times, denoted (a) and (c). The verifier starts receiving the message at time (d) and starts forwarding the message to the receiver at time (e). At time (f), the verifier knows whether the digest is correct. If an attacker attempts to modify a message, the verifier sends the wrong CRC (crossed out) to the receiver.

Chapter 3

Approach

Previous work [23,28,31,33] looked at authenticating individual messages with a digest and delayed messages because of encoding inefficiencies, namely searching for special symbols in the plaintext and escaping special symbols in the ciphertext. To eliminate these inefficiencies, I look at broader message patterns.

My solution is to use a Bayesian network to predict the incoming plaintext and pre-send the prediction. As each byte enters the authenticator, the device predicts the rest of the message based on its previous observations. It compresses and encrypts its hypothesis and pre-sends as much ciphertext as possible (Figure 3.3b). In effect, the authenticator uses prediction to take advantage of the higher bandwidth for ciphertext that is provided by this optimistic, *a priori* compression of the plaintext. If the authenticator compresses the plaintext without predicting it, then the BitW would not be able to pre-send the ciphertext. Thus, the authenticator would not be utilizing the higher bandwidth due to compression, and the latency would not improve. Intuitively, my solution is YASIR that predicts and compresses plaintext messages to eliminate encoding inefficiencies (Figure 3.2).

Note that a BitW can also use compression to avoid overloading a channel that is close to its capacity (Figure 3.1).

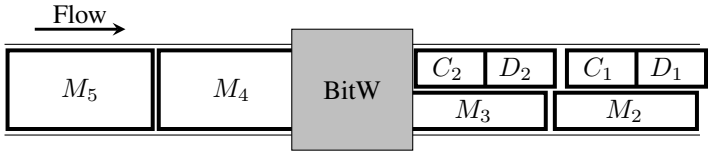

Figure 3.1: A BitW can overlap compressed messages with digests and counters to avoid overloading a channel that is close to its capacity. Messages M_2 and M_3 are compressed to be overlapped with digests and counters for messages M_1 and M_2.

Similar to YASIR, Predictive YASIR causes the receiver to drop the message if an adversary modifies the ciphertext. To prevent an attack on message integrity, the verifier forwards the message without the last byte to the receiver, which must have the last byte to act on the message. When the verifier receives the digest, it calculates its own to compare with what it has received. If the two digests match, then the verifier forwards the last byte of the message to the receiver (Figure 3.3e). The receiver now has the complete valid message. On the other hand, if the two digests differ, the verifier forwards the reset symbol to the receiver. Upon the reset symbol, the receiver must drop the incomplete message to adhere to the specifications in Modbus/ASCII protocol.

If the authenticator changes its hypothesis, the device sends a *back-away* signal to the verifier to indicate how much of the prediction is incorrect plus the delta for the correct ciphertext (Figure 3.3c). An authenticator may make and incorrect prediction for a rare message. Thus, rare messages are longer due to the back-away and the delta. This property resembles Huffman coding, which uses longer codes to represent rare characters [8].

When the authenticator receives the whole message, it sends the digest and the counter for this message (Figure 3.3d–e). The authenticator then updates the weights in its Bayesian network. I elaborate on the Bayesian network in Section 4.

Note that Predictive YASIR does not eliminate the overhead due to the digest. Although an authenticator can calculate the digest of the predicted message, the device cannot compress and pre-send this digest. Compressing the digest does not work because a strong digest does not have a pattern. Pre-sending the digest does not work

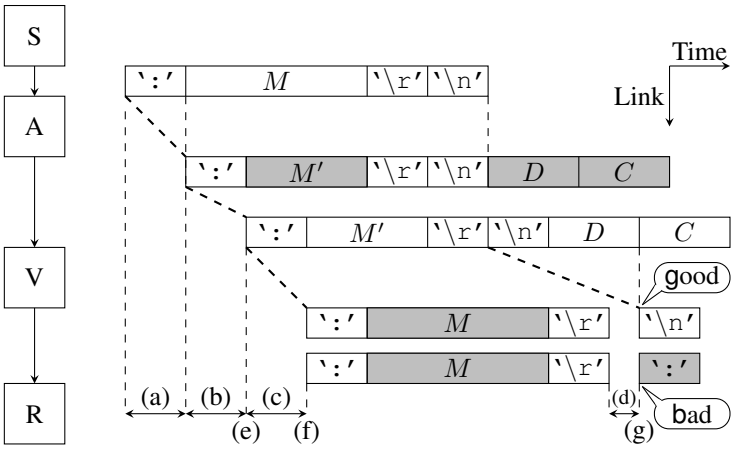

Figure 3.2: Latency with Predictive YASIR. The authenticator does not buffer the message, but must delay it by 1 byte-time, denoted (a). The verifier also does not buffer the message and also must delay it by 1 byte-time, denoted (c). In addition, the verifier delays the message by $|D| - 1$ byte-times, denoted (d). When prediction works well, the overall delay is $|D| + 1$, which is 13 byte-times. The verifier starts to receive the message at time (e) and starts forwarding it to the receiver almost immediately at time (f). At time (g), the verifier knows whether the digest is correct. If an attacker attempts to modify a message, the verifier resets the receiver with `':'` instead of forwarding the whole message.

because that would weaken the digest strength, effectively truncating the digest by the number of bytes that are pre-sent. For example, if the authenticator pre-sends 11 bytes of a 12 byte digest, then the digest is truncated down to 1 byte. An adversary would have to guess 1 byte of the digest to convince the verifier that the originally predicted message is correct. The adversary has an at most 1 in 8 chance of correctly guessing a single byte. If the authenticator makes a mistake in prediction and later corrects the message, the adversary can drop the correction to prevent the verifier from receiving the correct message. Such attack is dangerous when the originally predicted message is "everything is turned on," but the actual message is "everything is turned off." Whereas the first status message is more common, an operator would highly value receiving the second status message, which indicates an outage.

Contribution. The solution I provide is a non-intrusive way to "steal" bandwidth for security needs via data coding techniques and utilize this bandwidth with help from

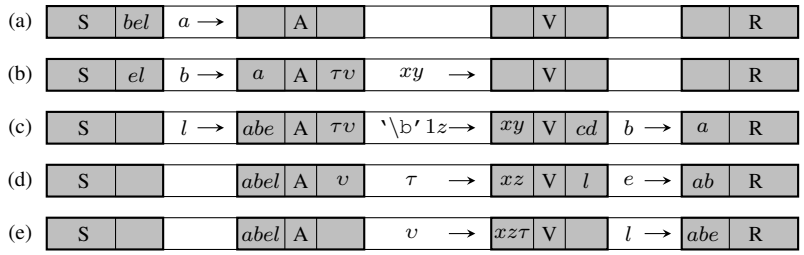

Figure 3.3: Example of Predictive YASIR operation. (a) The sender begins message transmission. (b) The authenticator receives prefix a, predicts that the message is $abcd$, compresses and encrypts the prediction into ciphertext xy, and sends out xy. (c) The authenticator receives prefix abe, changes its prediction to $abel$, compresses and encrypts the prediction into xz, and sends the back-away to replace y with z. (d) The authenticator receives the full message from the sender and transmits the digest τ to the verifier. (e) The authenticator transmits the counter υ to the verifier. The verifier compares the received digest τ to its own calculation. If the two digests match, the verifier forwards the last byte of the message to the receiver. Otherwise, the verifier resets the receiver.

message prediction. The coding is effective because data being sent is sufficiently low entropy and can thus be compressed and predicted to some extent.

Chapter 4

Methods

4.1 Modbus

Control centers often communicate with substations in Modbus/ASCII [16] protocol. This protocol is also widely used in oil and gas, manufacturing, and water treatment control networks. Modbus/ASCII dictates the sender to begin a message with the *reset* symbol `':'` (colon). If a device receives this symbol, it must drop any incompletely received message, i.e., reset itself. The sender encodes every message byte in ASCII, where this variation of the protocol takes its name. At the end of the message, the sender appends a CRC and the terminating symbols `'\r\n'` (a carriage return plus a newline).

For example, if the CRC of `0xABCD` is `0xEF`, then the sender encodes the hex message `0xABCD` into a Modbus/ASCII message `':ABCDEF\r\n'`, which is `0x3A4142434445460D0A` in hex (Figure 4.1).

Note that ASCII encoding is inefficient, because every byte of the message is two bytes in Modbus/ASCII. This inherent inefficiency allows for greater debugging capabilities in the field, but I use it to compress messages. I conjecture that a BitW can compress a SCADA message in any format, however, because SCADA communica-

tions are repetitive. SCADA messages also have a built-in redundancy that prevents random errors. A BitW can compress this redundancy, because Predictive YASIR provides integrity protection via the digest.

4.2 Scalable Simulation Framework

I use the Scalable Simulation Framework (SSF) to construct the experiment and measure the overhead of my approach [1, 17]. SSF simulates networked entities that exchange events. The framework automates collection of various statistics about the simulation. If the simulation is large and runs slowly on a single computer, then I can scale it up with minimal effort by distributing the workload over a set of machines. The device entities exchange single byte events to ensure they can process one byte at a time.

To synchronize the timing, a BitW outputs at most one byte for each byte that it receives, except after it has received the whole message. After a BitW receives the whole message, the device can output the rest of plaintext or ciphertext. Such behavior simulates enough silence on the wire to allow for any message expansion. Legacy slow networks may be saturated, however, leaving no space for message expansion. An interesting direction of further research is whether message authentication is possible with zero-byte message expansion. Note that Predictive YASIR does not achieve this goal, regardless of message compression ratio, because the authenticator must send the digest only after it receives the whole plaintext message from the sender.

4.3 Device

My design for a BitW entity has two ports: one for plaintext and one for ciphertext. The device continuously listens for input on both ports. The machinery for processing data on these ports is independent. If a device receives data on the ciphertext port

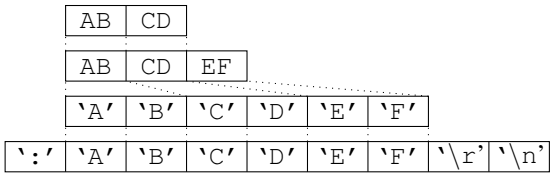

Figure 4.1: Encoding of an example message into Modbus/ASCII.

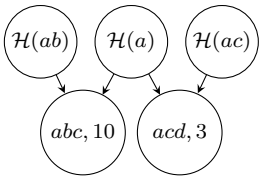

Figure 4.2: Bayesian network as a bipartite graph.

while processing plaintext input, or vice-versa, the device deals with the two inputs independently and asynchronously. Thus, I treat a physical link as two independent simplex links, one in each direction, similar to Jacobson [11].

When a BitW receives data on the plaintext port, it increments the message counter, generates a hypothesis about the rest of the plaintext message, then compresses, encrypts, authenticates this hypothesis, and finally sends the ciphertext on the ciphertext port or stays silent, depending on how much ciphertext it has sent. Once the device receives the whole plaintext, it generates the digest and sends the digest with the counter on the ciphertext port.

When a BitW receives data on the ciphertext port, it increments the message counter, decrypts and decompresses the data and starts to send out the result on the plaintext port or buffers the digest, depending on which part of the ciphertext it is receiving. Once the device has received the whole digest, it verifies it. If the digest is valid for this message, the device sends the final part of the plaintext message on the plaintext port; if the digest is not valid, then the device resets the receiver to drop the maliciously altered message. Identical to YASIR [28], the receiving device uses the actual counter value to synchronize its own counter with the sending device.

4.4 Bayesian Network

To predict the incoming plaintext, the authenticator models the network traffic with a Bayesian network (Figure 4.2). The model is a labeled directed acyclic graph. A

vertex label is either a message prefix or a full message and its frequency. All edges are directed from the prefixes to the full-length messages. A prefix vertex may have multiple out-edges. For instance, the prefix `':'` has an edge to all observed messages, because all Modbus/ASCII messages begin with this symbol. Note that a message vertex has in-edges from all of its prefixes.

I implement the Bayesian network with a hash-table of prefixes and a table of tuples (m, f)—messages and their frequencies. Figure 4.2 uses $\mathcal{H}$ to denote hashing. Each prefix object has a list of the message-frequency tuples. When a plaintext message passes through an authenticator, the frequency of this message increases by one. To predict the rest of the message from its prefix, the authenticator looks up the prefix hash in the Bayesian network. This prefix may have edges to multiple messages, out of which the authenticator predicts the most frequent one.

4.5 Bayes' Theorem

To prevent incorrect predictions, the authenticator calculates the probability of a hypothesis correctness under current data observation using Bayes' theorem. A hypothesis is the message prediction. A data observation is the prefix. If a hypothesis is less than 50% likely, then the device falls back to its non-predictive mode, which is similar to YASIR. Bayes' theorem states

$$\Pr(H|D) = \frac{\Pr(D|H) \cdot \Pr(H)}{\Pr(D)} \ .$$

1. $\Pr(D|H)$ is the conditional probability of the current data observation given a hypothesis. If the predicted message is correct, then the prefix must occur. Therefore, we have

$$\Pr(D|H) = 1 \ .$$

2. $\Pr(H)$ is the prior probability of a hypothesis. This is a ratio of the number h of occurrences of the predicted message to the total number t of messages that are at least as long as the prefix. Therefore, we have

$$\Pr(H) = \frac{h}{t} \ .$$

3. $\Pr(D)$ is the prior probability of data occurrence. This is a ratio of the number d of occurrences of this prefix over the total number o of all previously observed prefixes of the same length. Therefore, we have

$$\Pr(D) = \frac{d}{o} \ .$$

Substituting these terms into the equation yields

$$\Pr(H|D) = \frac{h \cdot o}{t \cdot d} \ .$$

For example, when the authenticator receives the reset character `':'`, which begins every message, the probability of this data occurrence equals one, because the number of occurrences of this prefix equals the total number of all previously observed prefixes of the same length. More formally, we would have

$$D = \texttt{':'} \ \Rightarrow \ d = o \ \Rightarrow \ Pr(D) = \frac{d}{o} = 1 \ .$$

The device predicts the most frequent message overall. The device is likely to be correct only if this message constitutes at least 50% of all observed messages.

On the other hand, a longer prefix improves prediction accuracy. As the prefix length approaches the message length, a device predicts the message with probability close to one, because the probability $\Pr(D)$ of the prefix approaches the probability

$\Pr(H)$ of the message.

4.6 Back-away

As an authenticator pre-sends a predicted message, it monitors the incoming plaintext to verify that the prediction is correct. If the authenticator discovers an error in its prediction, it sends the back-away signal `'\b'` (backspace) to the verifier, followed by the number of bytes to discard from the predicted message, and transmits the corrected part of the message (Figure 3.3c). The discarded bytes are always the last bytes that the device sends out, because Predictive YASIR uses stream compression and encryption algorithms. For example, if the authenticator needs to discard the last byte and replace it with `0xFF`, then it sends `'\b'` followed by `0x01FF`, which is `0x0801FF` all together in hex. The verifier computes the digest on the final version of the message, after it discards all incorrect predictions.

The verifier must receive the back-away symbol before it forwards the wrong part of plaintext to the receiver. That is, the verifier should send plaintext byte i only after the authenticator receives the actual byte i. The verifier maintains this property by outputting at most one byte of plaintext for every byte of ciphertext and by being silent when it receives data that increases the length of the ciphertext: the escapes and the back-aways.

4.7 Cipher Format

Because Modbus/ASCII uses only half of the available bandwidth, my compression reclaims this space. The authenticator converts each ASCII character (`'0'` to `'9'` and `'A'` to `'F'`) into its equivalent 4-bit representation: `0x0` to `0xF`. The authenticator appends the digest and the counter after the terminating symbol `'\r\n'`. Thus the

whole encrypted and authenticated message comprises `':'` symbol, followed by message data, followed by `'\r\n'`, followed by the digest and the counter (Figure 1.2d).

Because the ciphertext may contain the special symbols `'\r\n'` (carriage return plus newline) and `'\b'` (backspace), the authenticator replaces the special symbols in the ciphertext with `'\r\r\n'` and `'\b\b'`. This makes the message longer, but Predictive YASIR compensates for this inefficiency with compression.

4.8 Experiment

The simulation contains four components: one FEP, two BitWs, and one DA (Figure 1.1a). The FEP connects to the plaintext port of the first BitW. The two BitWs connect via their ciphertext ports. The plaintext port on the second BitW connects to the DA.

The FEP has a set of messages that it sends to the DA in random order. It sends each byte of a message individually, but without delays. Therefore, the authenticator can only act on information from a single byte, which simulates a slow legacy network. The authenticator sends at most one byte of ciphertext for every byte of plaintext it receives, except after it has received the whole message, when it sends the rest of the ciphertext to the verifier. This simulates enough computational power to query the Bayesian network on every byte of plaintext and enough silence on the wire to avoid congestion due to ciphertext being longer than plaintext. Because real world networks may be saturated, it would be interesting to look for ways to avoid message expansion. One possibility is to overlap the digest and counter for one message with the ciphertext for the next message (Figure 3.1).

The data for the experiment is a trace collected from GE XA/21™ SCADA / Energy Management System talking to a GE D400 Substation Data Manager in a lab setting. These devices use DNP3 protocol to communicate and record the trace. Before the

simulation, I convert the trace into Modbus/ASCII format suitable for input into SSF. I do not have Modbus/ASCII traces because obtaining traces from real-world settings is difficult.

Both YASIR and Predictive YASIR run 30 times. In the ith run of the simulation, the FEP has $10i$ unique messages to send to the DA. I vary the number of unique messages because prediction ability may deteriorate with many unique messages. Each run lasts for 200,000 SSF ticks, enough to send each message more than once. I reset the Bayesian network after each run.

The simulation assumes that the BitWs have enough computational power so that prediction, compression, encryption, and authentication operations do not affect latency. I measure the average byte-time latency in each test and calculate the improvement percentage from YASIR to Predictive YASIR. I do not compare performance of my solution to the approaches prior to YASIR, because YASIR has the lowest latency.

Chapter 5

Results

I present the results of the simulation in Figure 5.1. These results demonstrate that Predictive YASIR has 15.48% less average latency than the original YASIR with a 95% confidence interval of 0.35 percentage points. Recall that I do not compare Predictive YASIR to other bump-in-the-wire devices that provide message authenticity, because they have higher latency than the original YASIR. I find that prediction performance does not degrade when the number of unique messages increases. Predictive YASIR latency is 13.52 byte-times with a 95% confidence interval of 0.06. In contrast, original YASIR latency is always 16 byte-times. When the authenticator makes a prediction mistake in the experiment, it successfully recovers with a back-away. The verifier determines all messages to be valid, because I do not introduce errors into the ciphertext stream.

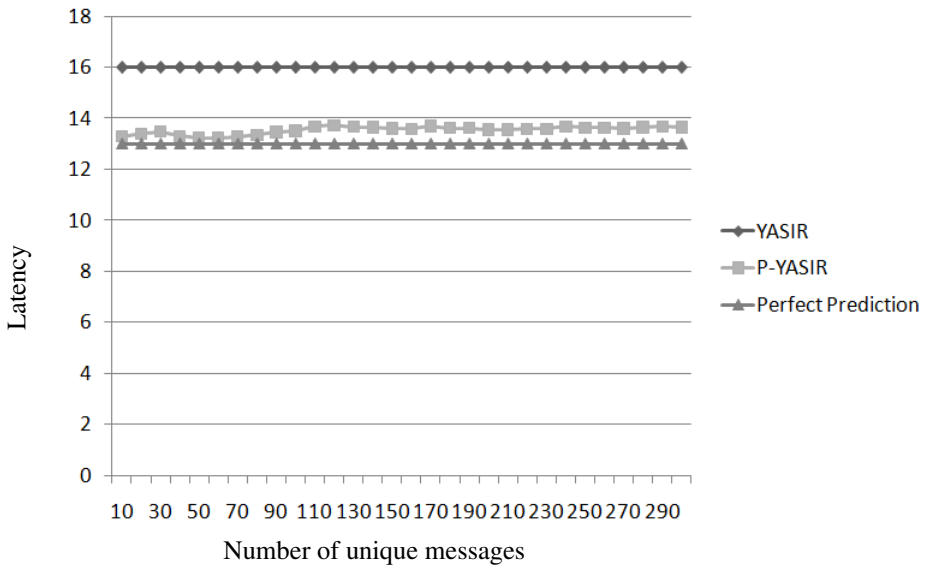

Figure 5.1: Average end-to-end latency of YASIR and Predictive YASIR simulation. I include for reference the byte-time latency for perfect prediction with a 12-byte HMAC digest.

Chapter 6

Future Work

6.1 Semi-embedded Device

A security device may gain a performance improvement if it plugs directly into the sending device. Because a PCI or ISA port has much higher bandwidth than a serial cable, the security device should receive a message virtually in an instant. If the security device compresses the message, than it should incur near-zero latency due to the appended digest and counter. Such semi-embedded device would not need prediction. Measuring performance of such a device may require building it on a PCI or ISA board, however, which requires hardware expertise and resources. One complication is that a utility would have to reconfigure or patch its software to send the message through a PCI port, instead of the serial port. Such modifications may be as expensive as patching the software to use cryptography. The advantage of a semi-embedded solution is a clear separation from the sending device, which is good engineering practice.

6.2 Historical Data

An BitW authenticator may be able to use historical data to predict plaintext, similar to a branch predictor in an instruction pipeline. Historical data can be useful in predicting

natural phenomena, such as temperature. Imagine a sensor to measure the temperature of water in a river. This temperature is 10°C in the majority of cases, but recently has increased to 11°C and remains at that level. An authenticator that uses only statistics would continue to mistakenly predict messages with 10°C temperature reports. In contrast, an historical predictor would adjust its predictions even though the long-term majority of the temperature reports is still at 10°C.

6.3 Protocols

I use Modbus/ASCII protocol, but I conjecture that the technique scales well to other industrial control network protocols, e.g., DNP3 [4]. Although one can easily compress a Modbus/ASCII message, all sensors should have a finite and small number of states. For instance, outdoor water temperature has only 100 integer states in Celsius and varies little. The compression scheme must be stream-wise to avoid incurring latency. The scheme by Welch, Ziv, and Lempel should work well [30, 34]. Their algorithm is stream-wise and efficient. The source code is publicly available.

6.4 Space

From a theoretical perspective, Predictive YASIR computes statistics about the data stream to predict the next message. My implementation uses space that is linear in the number of unique messages in the stream, because the authenticator stores each unique message in its Bayesian network for further predictions. Many computer scientists prefer to use more efficient stream statistics algorithms, such as that of Indyk and Woodruff [10] or Ganguly et al. [7]. Replacing the Bayesian network would require a new prediction algorithm, however, which may not be trivial.

6.5 Key Management

I do not address key distribution, but concentrate on the BitW algorithm. Other works in this area have addressed the key distribution issue. For example, the AGA SCM design [31] specifies how devices negotiate the keys and ScadaSafe [32] implements these specifications. Key management is easy to misconfigure or ignore when the cryptographic device resides in a locked up substation, creating a false sense of safety. Therefore, easily and correctly configurable security policies deserve our attention.

6.6 Validation

Collecting real network data traces from substations and control centers is an important task to verify correctness of this simulation and test future hypotheses. Because vendors and utilities hesitate to share data that may reveal proprietary information, it is important to build trust relationships with the industry.

Chapter 7

Conclusions

I demonstrate how message prediction and coding techniques can be used to decrease latency due to encoding inefficiencies. I apply this idea to message authentication in slow legacy power grid networks. I hypothesize that this method is effective because the data is sufficiently low entropy and thus a BitW can predict and compress it. My evaluation demonstrates a $15.48 \pm 0.35\%$ improvement in byte-time latency without compromising on security. Such savings can be significant on congested networks that require a fast response and in other applications with encoding inefficiencies. Finally, I propose a range of research directions to further our understanding of these issues and improve the state of art. The simulation code is available from `http://www.cs.dartmouth.edu/~pyasir/`.

Bibliography

[1] Jerry Banks, John S. Carson II, Barray L. Nelson, and David M. Nicol. *Discrete-Event System Simulation*. Prentice Hall, New Jersey, 4th edition, 2005.

[2] Nikita Borisov, Ian Goldberg, and David Wagner. Intercepting Mobile Communications: The Insecurity of 802.11. *Proceedings of the Seventh Annual International Conference on Mobile Computing and Networking*, Jul 2001.

[3] RealTime Interactive Systems Corporation. `http://www.rtiscorp.com/`, 2010.

[4] DNP Users Group. DNP—Overview of the DNP3 Protocol. `http://www.dnp.org/About/`, 2009.

[5] D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography. *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, pages 542–552, 1991.

[6] Terry Fleury, Himanshu Khurana, and Von Welch. Towards a Taxonomy of Attacks Against Energy Control Systems. *Proceedings of the IFIP International Conference on Critical Infrastructure Protection*, Mar 2008.

[7] Simit Ganguly, Abhaendra Singh, and Satyam Shankar. Finding Frequent Items Over General Update Streams. *Scientific and Statistical Database Management*, 2008.

[8] David A. Huffman. A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the I.R.E.*, 40(10):1098–1101, 1952.

[9] Schweitzer Engineering Laboratories Inc. `http://www.selinc.com/`, 2010.

[10] Poitr Indyk and David Woodruff. Optimal Approximations of the Frequency Moments of Data Streams. *Symposium on Theory of Computing*, 2009.

[11] V. Jacobson. Compressing TCP/IP Headers for Low-Speed Serial Links. *Request for Comments 1144*, 1990.

[12] J. Kohl and B. Clifford Neuman. Kerberos Version 5 RFC, draft #4. *Project Athena*, Dec 1990.

[13] Pacific Northwest National Laboratory. `http://www.pnl.gov/`, 2010.

[14] Alfred J. Menezes, van Paul C. Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*, page 311. CRC Press, 1st edition, Dec 2001.

[15] Robert B. Miller. Response time in man-computer conversational transactions. *Proceedings of the Fall AFIPS Joint Computer Conference, Part I*, pages 267–277, Dec 1968.

[16] Modbus-IDA. MODBUS Application Protocol 1.1b. `http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf`, 2006.

[17] UIUC Modeling and Networking Systems Research Group. PRIME: Parallel Real-time Immersive network Modeling Environemnt. `http://www.primessf.net/bin/view/Public/PRIMEProject`, 2009.

[18] Harry Nyquist. Certain Factors Affecting Telegraph Speed. *Bell System Techinal Journal*, page 324, Apr 1924.

[19] Harry Nyquist. Certain Topics in Telegraph Transmission Theory. *American Institute of Electrical Engineers Transactions*, 47:617, Apr 1928.

[20] Open Software Foundation, OSF, Distributed Computing Environment (DCE). Remote Procedure Call Mechanisms, Code Snapshot 3, Release 1.0. Mar 1991.

[21] Mischa Schwartz. *Information Transmission, Modulation, and Noise*, pages 432–436,594–595. McGraw-Hill Publishing Company, New York, 4th edition, 1990.

[22] Schweitzer Engineering Laboratories Inc. SEL-3021-2 Serial Encrypting Transceiver Data Sheet. `http://www.selinc.com/WorkArea/DownloadAsset.aspx?id=2855`, 2007.

[23] Schweitzer Engineering Laboratories Inc. SEL-3021-2 Serial Encrypting Transceiver. `http://www.selinc.com/SEL-3021-2/`, 2009.

[24] Claude E. Shannon. A Mathematical Theory of Communication. *Bell System Techinal Journal*, 27:623–656, Oct 1948.

[25] Rouslan Solomakhin, Patrick Tsang, and Sean Smith. High Security with Low Latency in Legacy SCADA Systems, to appear in *Critical Infrastructure Protection IV*. T. Moore and S. Shenoi (Eds.), Springer, Heidelberg, Germany, 2010.

[26] Stuart G. Stubblebine and Virgil D. Gligor. On Message Integrity in Cryptographic Protocols. *Proceedings of the 1992 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 85–104, May 1992.

[27] Andrew S. Tanenbaum. *Computer Networks*, pages 77–82. Prentice Hall, New Jersey, 3rd edition, 1996.

[28] Patrick P. Tsang and Sean W. Smith. YASIR: A Low-Latency, High-Integrity Security Retrofit for Legacy SCADA Systems. *Proceedings of the IFIP TC 11 23rd International Information Security Conference*, 278:445–459, 2008.

[29] U.S. Department of Commerce, National Institute of Standards and Technology, Information Technology Laboratory. Secure Hash Standard. *FIPS PUB 180-3*, Oct 2008.

[30] Terry A. Welch. A Technique for High-Performance Data Compression. *IEEE Computer*, pages 8–19, 1984.

[31] Andrew K. Wright. AGA 12 Part 2-akw Proposed SCADA Encryption Protocol. `http://scadasafe.sourceforge.net/Protocol`, 2006.

[32] Andrew K. Wright. ScadaSafe. `http://scadasafe.sourceforge.net/`, 2009.

[33] Andrew K. Wright, John A. Kinast, and Joe McCarty. Low-Latency Cryptographic Protection for SCADA Communications. *Proceedings of the 2nd International Conference on Applied Cryptography and Network Security*, pages 263–277, 2004.

[34] Jacob Ziv and Abraham Lempel. Compression of Individual Sequences via Variable-Rate Coding. *IEEE Transactions on Information Theory*, IT-24(5), 1978.