Chapter 1

PREDICTIVE YASIR: HIGH SECURITY WITH LOWER LATENCY IN LEGACY SCADA

Rouslan V. Solomakhin, Patrick P. Tsang and Sean W. Smith

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. We 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, our results demonstrate a $15.32 \pm 0.27\%$ improvement in latency.

1. Introduction

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, we develop an approach to optimize the performance of the previous fastest security solutions.

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 [3].

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) [5].

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.

Such attacks violate the FEP's and the DA's assumption about authenticity of the messages. The messages must be protected, or *authenticated*. Although a utility can upgrade its FEPs and DAs to more secure versions, representatives of the power industry tell our lab that these devices are prohibitively expensive, and the upgraded devices would still have to communicate over the slow legacy network. A network upgrade is also expensive. The cheaper and faster option is a bump-in-the-wire device (BitW) [7, 13, 14, 16] that secures all messages that it intercepts on a legacy network.

Two BitWs work in concert to authenticate a message (Figure 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

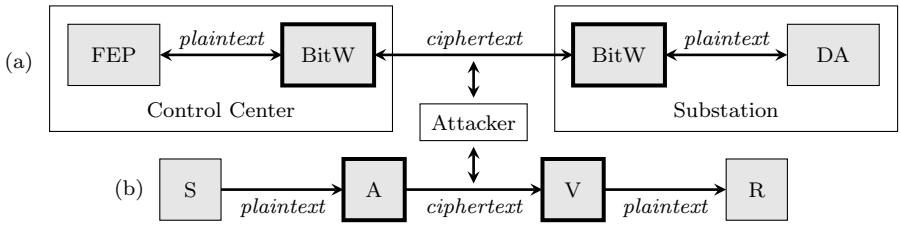

Figure 1. BitWs. (a) A typical setup. (b) A useful abstraction.

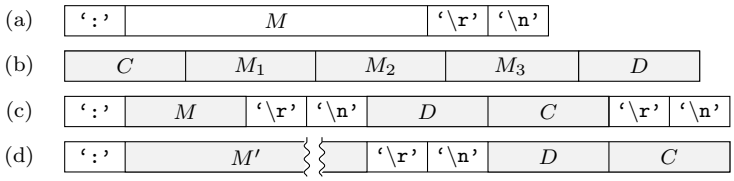

Figure 2. Message formats for Modbus/ASCII. (a) Plaintext. (b) PE ciphertext in blocks. (c) YASIR ciphertext. (d) Predictive YASIR ciphertext. The shaded areas are modified or generated by a BitW. The symbols C and D denote the counter and digest.

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, 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 prior to the device.

When a DA sends a message to a FEP, the authenticator and the verifier switch roles. Due to this symmetry, we prefer to use the words sender and receiver instead (Figure 1b). We may use the word device to refer to any sender, authenticator, verifier, or receiver. In the figures, we 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. For example, suppose this bound is 300 ms (Figure 3; we use the diagram style from the YASIR paper [13]). 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, if a byte has 10 bits. Because of the low bandwidth, a BitW should not wait to receive the whole message before processing it,

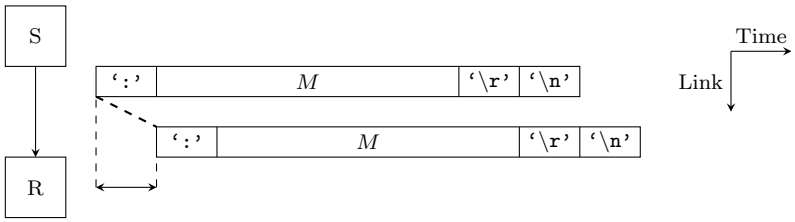

Figure 3. Transmission latency without authentication.

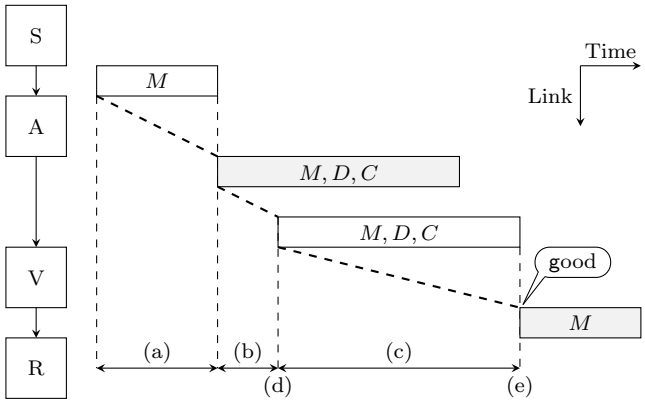

Figure 4. Latency with hold-back. Both BitWs hold-back the whole message before forwarding it. The shaded areas are modified or generated by a BitW. Hold-back delays a message by (a) and (c). The transmission delay is (b). The verifier starts receiving the message at time (d), but starts to forward it 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.

a practice known as *hold-back* (Figure 4). If both BitWs in a pair hold-back a message that is longer than 144 bytes, the delay would exceed 300 ms. 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.

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

Tsang and Smith [13] 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. [16] 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 5). We use a similar technique to let the verifier process the message online (Figure 9).

When an authenticator appends the digest to the message, it must ensure that the verifier can distinguish between them. We have two options. The first option is to prepend the message length to the ciphertext, but a common protocol like Modbus/ASCII (Figure 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 our work, we aim to eliminate these inefficiencies in online processing.

We do not try to eliminate the overhead due to the digest by compressing or pre-sending it. A BitW cannot compress the digest, because a strong digest does not have a pattern. Also, a BitW cannot pre-send a part of the digest before the device receives the full plaintext message, because this would weaken digest strength. If an authenticator pre-sends a part of the digest, the pre-sent part would contain no information about the part of the message that the authenticator has not yet received.

Organization. The rest of the paper comprises six sections. In Section 1.2, we list previous approaches to build BitWs that authenticate messages on legacy slow networks. Then our approach is briefly explained in Section 1.3. In Section 1.4, we describe our approach in detail and outline the test methodology. We show the evaluation results in Section 1.5. In Section 1.6, we list future research directions. Finally, we draw conclusions in Section 1.7.

2. Related Work

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, we consider related BitW solutions only if they authenticate messages: SEL-3021-2, AGA SCM, and YASIR. (Thus we ignore such solutions as SEL-3021-1, because it does not protect integrity, or PNNL SSCP embedded device, because it is not a bump-in-the-wire device.)

SEL-3021-2. SEL-3021-2 [7] is a commercial off-the-shelf BitW from Schweitzer Engineering Laboratories. The device uses the Message Authentication Protocol (MAP) [6] 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.

AGA SCM. AGA SCM (American Gas Association SCADA Cryptographic Module) [14] is a BitW proposed by AGA 12 Task Group. The group members have developed a reference implementation [15], 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 4). The online mode is Position Embedding (PE) [16], which is a modified version of AES in counter mode (AES-CTR) followed by a standard version of AES in electronic code book

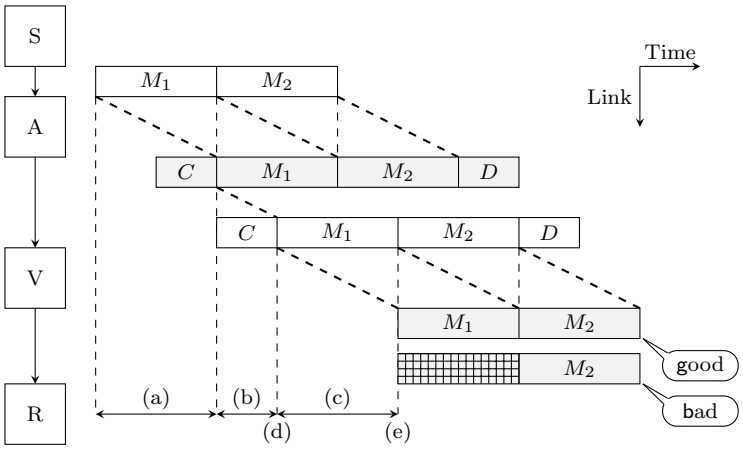

Figure 5. Latency in PE mode. The parts M_1 and M_2 are blocks of message M. Both BitWs buffer a 16-byte block of a message before forwarding it, delaying the message by 32 byte-times, denoted (a) and (c). The verifier starts receiving the message at time (d) and starts forwarding it to the receiver 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.

mode (AES-ECB) (Figure 6). 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 5).

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 [2, 9], i.e., an adversary can modify the ciphertext with predictable changes to the CRC, even without learning the encryption key.

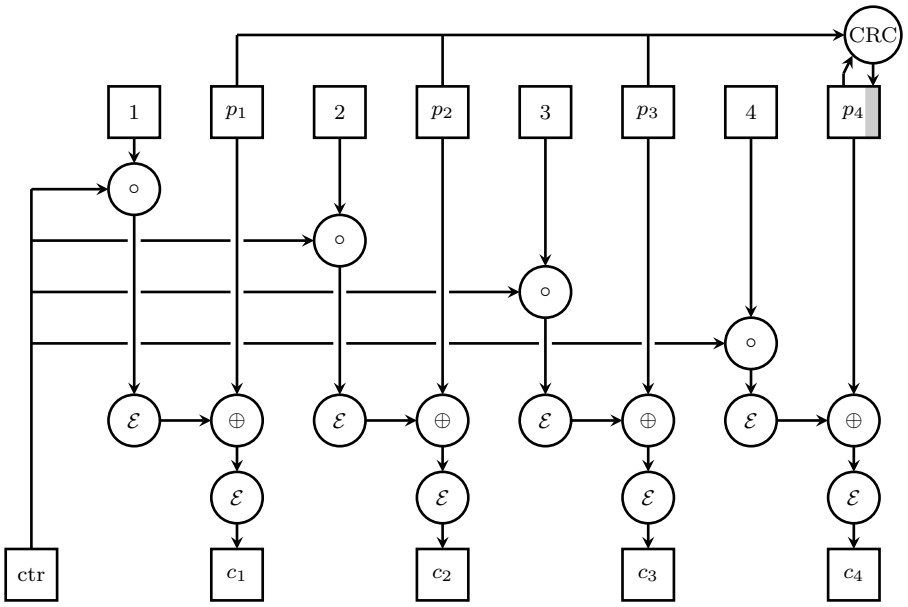

Figure 6. PE mode is AES-CTR, followed by AES-ECB with the same key. This mode relies on CRC to authenticate messages. The symbols $\circ$ and $\oplus$ denote concatenation and exclusive-or. The symbol $\mathcal{E}$ is the encryption function. The plaintext is $p_1 \circ p_2 \circ p_3 \circ p_4$, and the ciphertext is $\mathrm{ctr} \circ c_1 \circ c_2 \circ c_3 \circ c_4$. 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.

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 weary 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, who exploited predictability of CRC to create undetectable bogus messages for Kerberos and Remote Procedure Calls (RPC) [12].

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

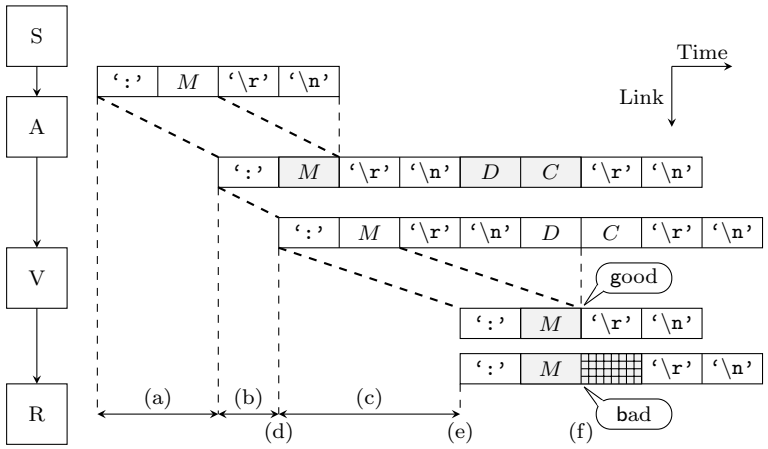

Figure 7. Latency with YASIR. The authenticator buffers 2 bytes of the message to detect its end. The verifier buffers 14 bytes of the message to verify its digest. Total latency is 16 bytes-times, denoted (a) and (c). The verifier starts receiving the message at time (d) and starts forwarding the message 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.

adversary modifies a message in PE mode, the verifier BitW scrambles the plaintext in 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.

YASIR. Our lab's YASIR [13] is a BitW that authenticates messages with $\leq$18 bytes-times of overhead (Figure 2c). The actual overhead depends on the underlying protocol. With Modbus/ASCII, YASIR delays a message by $\sim$16 byte-times (Figure 7). 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 the PE mode [14], 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 [17].

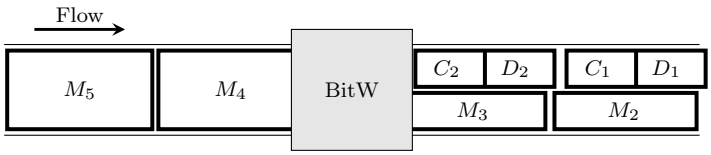

Figure 8. A BitW can overlap compressed messages with digests and counters to avoid overloading a channel that is close to its capacity. Messages M_2 and M_3 are compressed to be overlapped with digests and counters for messages M_1 and M_2.

3. Approach

Previous work [7, 13, 14, 16] 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, we look at broader message patterns.

Our 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 10b). In effect, we use prediction to take advantage of the higher bandwidth for ciphertext that is provided by this optimistic, *a priori* compression of the plaintext. Intuitively, my solution is YASIR that predicts plaintext messages to eliminate encoding inefficiencies (Figure 9). Note that a BitW can also use compression to avoid overloading a channel that is close to its capacity (Figure 8).

Similar to YASIR, Predictive YASIR causes the receiver to drop the message if an attacker modifies the ciphertext. The verifier forwards the message without the last byte to the receiver, which must have the last byte before it acts 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, the verifier forwards the last byte of the digest to the receiver (Figure 10e). 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 10c). When the authenticator receives the whole message, it sends the digest and the counter for this message (Figure 10d–e). The device then updates the

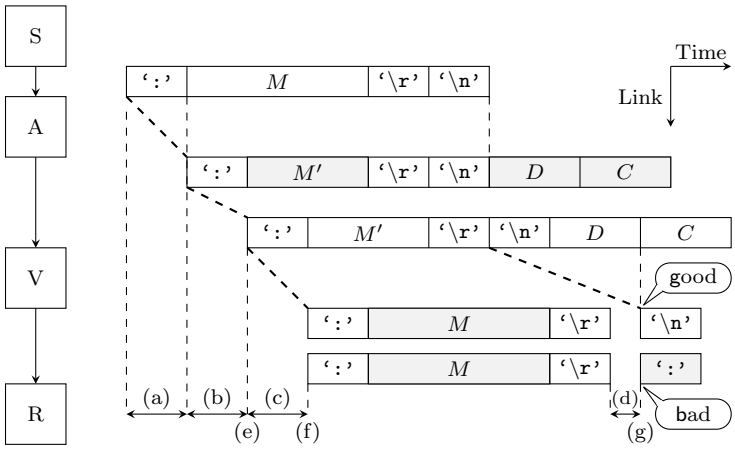

Figure 9. Latency with Predictive YASIR. The authenticator does not buffer the message, but must delay it by 1 byte-time, denoted (a). The verifier also does not buffer the message and also must delay it by 1 byte-time, denoted (c). In addition, the verifier delays the message by $|D| - 1$ byte-times, 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.

weights in the Bayesian network. We elaborate on the Bayesian network in Section 1.4.

Contribution. The solution we provide is a non-intrusive way to "steal" bandwidth for security needs via data coding techniques and utilize this bandwidth with help from message prediction. The coding is effective because data being sent is sufficiently low entropy and can thus be compressed and predicted to some extent. (If a BitW compresses without predicting, it would have to wait more of the message, incurring more latency.)

4. Methods

(For more details, see the upcoming technical report [18].)

Modbus. Control centers often communicate with substations in Modbus/ASCII [10] protocol. (This protocol is also widely used in oil and gas, manufacturing, and water treatment control networks.) The sender begins to transmit a message with the *reset* symbol `':'` (colon). If a device receives this symbol, it must drop any incompletely received

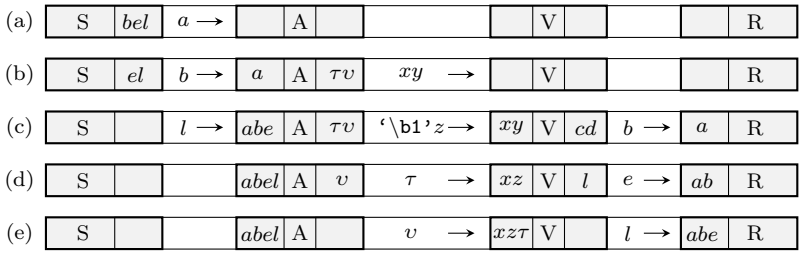

Figure 10. Example of Predictive YASIR operation. (a) The sender begins message transmission. (b) The authenticator receives prefix a, predicts that the message is *abcd*, compresses and encrypts the prediction into ciphertext xy, and sends out xy. (c) The authenticator receives prefix *abe*, changes 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, i.e., reset itself. The sender encodes every byte in the message 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 and 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 11).

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 we use it to compress messages.

Scalable Simulation Framework. We use the Scalable Simulation Framework (SSF) [1, 11] to construct an experiment and measure the overhead of our approach. 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, we 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.

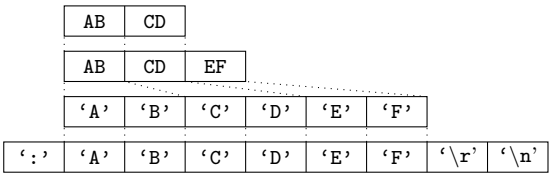

Figure 11. Encoding of an example message into Modbus/ASCII.

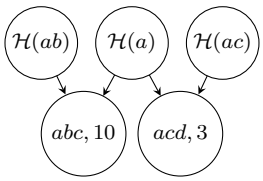

Figure 12. Bayesian network as a bipartite graph.

Device. Our 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 while processing plaintext input, the device deals with the two inputs independently in order.

Bayesian Network. To predict the incoming plaintext, the authenticator models the network traffic with a Bayesian network (Figure 12). 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.

We implement the Bayesian network with a hash-table of prefixes and a table of tuples (m, f)—messages and their frequencies. Figure 12 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.

Bayes' Theorem. To prevent incorrect predictions, the authenticator calculates the probability $\Pr(H|D)$ 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 $\Pr(H|D) = \Pr(D|H) \cdot \Pr(H) \,/\, \Pr(D)$.

1 $\Pr(D|H)$ is the conditional probability of the current data observation given our 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 this hypothesis to the total number t of messages of same or greater length that passed through the device. Therefore, we have $\Pr(H) = h/t$.

3 $\Pr(D)$ is the prior probability of data occurrence. This is a ratio of the number d of occurrences of this data over the total number o of all pieces of data of the same length that passed through the device. Therefore, we have $\Pr(D) = d/o$.

Substituting these terms into the equation yields $\Pr(H|D) = (h \cdot o)/(t \cdot d)$.

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 10c). 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 z, then it sends the back-away signal `‘\b1’`z. The verifier computes the digest on the final version of the message, after it discards all incorrect predictions.

Cipher Format. Because Modbus/ASCII uses only half of the available bandwidth, our 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 2d).

Experiment. The simulation contains four components: one FEP, two BitWs, and one DA (Figure 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. 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.

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. We are grateful to Paul Myrda of Electric Power Research Institute for providing this trace. These devices use DNP3 protocol to communicate and record the trace. Before the simulation, we convert the trace into Modbus/ASCII format suitable for input into SSF. We do not have Modbus/ASCII traces because it is difficult to obtain traces from the real-world settings.

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. We 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. We 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. We measure the average byte-time latency in each test and calculate the improvement percentage from YASIR to Predictive YASIR. We do not compare performance of our solution to previous approaches, because YASIR has the lowest latency.

5. Results

We present the results of the simulation in Figure 13. These results demonstrate that Predictive YASIR has 15.32% less average latency than the original YASIR with a 95% confidence interval of 0.27 percentage points. Recall that we 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. We find that prediction performance does not degrade when the number of unique messages increases. Predictive YASIR latency is 13.55 byte-times with a 95% confidence interval of 0.04. In contrast, original YASIR latency is always 16 byte-times. When the authenticator makes a prediction mistakes in the experiment, it successfully recovers with a back-away. The

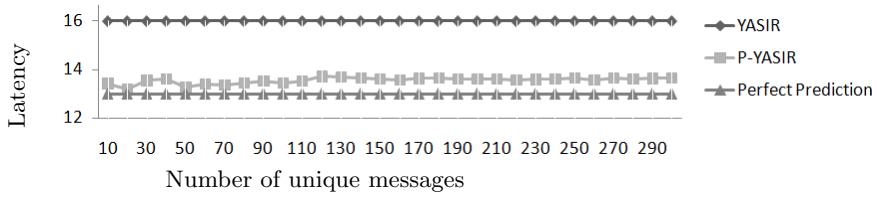

Figure 13. Average end-to-end latency of YASIR and Predictive YASIR simulation. We include for reference the byte-time latency for perfect prediction with a 12-byte HMAC digest.

verifier determines all messages to be valid, because we do not introduce errors into the ciphertext stream.

6. Future Work

We may further modify Predictive YASIR to improve prediction, to increase the number of protocols that the system can secure, to reduce the amount of space that the algorithm uses, to manage keys, and to validate the results in a real world setting.

Historical Data. An authenticator can 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. A historical system would adjust its predictions even though the long-term majority of the temperature reports is still at 10°C.

Protocols. We use Modbus/ASCII protocol, but we conjecture that the technique scales well to other industrial control network protocols, e.g., DNP3 [5]. 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.

Space. From a theoretical perspective, Predictive YASIR computes statistics about the data stream to predict the next message. Our implementation uses space that is linear in the number of unique mes-

sages in the stream. Many prefer to use more efficient stream statistics algorithms, such as that of Indyk and Woodruff [8] or Ganguly et al. [4].

Key Management. We do not address key distribution, but concentrate on the BitW algorithm. Other works in this area have addressed this key distribution issue. For example, the AGA SCM design [14] specifies how devices negotiate the keys and ScadaSafe [15] 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.

Validation. Finally, collecting real industrial network data traces from substations and control centers is an important task to verify correctness of this simulation and test future hypotheses. Unfortunately, vendors hesitate to share data, because it may reveal proprietary information or trade secrets.

7. Conclusions

We demonstrate how message prediction and coding techniques can be used to decrease latency due to encoding inefficiencies. We apply this idea to message authentication in slow legacy power grid networks. We hypothesize that this method is effective because the data is sufficiently low entropy and thus a BitW can predict and compress it. Our evaluation demonstrates a $15.32 \pm 0.27\%$ 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, we propose a range of research directions to further the knowledge of this area.

Acknowledgment: Work supported under DOE Award Number DE-OE0000097.

References

[1] J. Banks, J. Carson II, B. Nelson and D. Nicol, Discrete-Event System Simulation, Prentice Hall, New Jersey, 4th edition, 2005.

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

[3] T. Fleury, H. Khurana and V. Welch, Towards a Taxonomy of Attacks Against Energy Control Systems, *Proceedings of the IFIP International Conference on Critical Infrastructure Protection*, 2008.

[4] S. Ganguly, A. Singh and S. Shankar, Finding Frequent Items Over General Update Streams, *Scientific and Statistical Database Management*, 2008.

[5] DNP Users Group, DNP—Overview of the DNP3 Protocol (www.dnp.org/About).

[6] Schweitzer Engineering Laboratories Inc., SEL-3021-2 Serial Encrypting Transceiver Data Sheet (www.selinc.com/WorkArea/DownloadAsset.aspx?id=2855).

[7] Schweitzer Engineering Laboratories Inc., SEL-3021-2 Serial Encrypting Transceiver (www.selinc.com/SEL-3021-2).

[8] P. Indyk and D. Woodruff, Optimal Approximations of the Frequency Moments of Data Streams, *Symposium on Theory of Computing*, 2009.

[9] A. Menezes, P. van Oorschot and S. Vanstone, Handbook of Applied Cryptography, CRC Press, 1st edition, 2001.

[10] Modbus-IDA, MODBUS Application Protocol 1.1b (www.modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf).

[11] UIUC Modeling and Networking Systems Research Group, PRIME: Parallel Real-time Immersive network Modeling environment (www.primessf.net/bin/view/Public/PRIMEProject).

[12] S. Stubblebine and V. Gligor, On Message Integrity in Cryptographic Protocols, *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, pp. 85–104, 1992.

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

[14] A. Wright, AGA 12 Part 2-akw Proposed SCADA Encryption Protocol (scadasafe.sourceforge.net/Protocol).

[15] A. Wright, ScadaSafe (scadasafe.sourceforge.net).

[16] A. Wright, J. Kinast and J. McCarty, Low-Latency Cryptographic Protection for SCADA Communications, *Proceedings of the 2nd International Conference on Applied Cryptography and Network Security*, Springer, pp. 263–277, 2004.

[17] U.S. Department of Commerce, NIST, Information Technology Laboratory, Secure Hash Standard, FIPS PUB 180-3 (csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf).

[18] R. Solomakhin, Predictive YASIR: High Security with Lower Latency in Legacy SCADA, Dartmouth Computer Science Technical Report TR2010-665.