Chapter 5

# HIGH SECURITY WITH LOW LATENCY IN LEGACY SCADA SYSTEMS

Rouslan Solomakhin, Patrick Tsang and Sean Smith

**Abstract** Message authentication with low latency is necessary to ensure secure operations in legacy industrial control networks such as those in the power grid. Previous authentication solutions that examine single messages incur noticeable latency. This paper describes Predictive YASIR, a bump-in-the-wire device that reduces the latency by considering broader patterns of messages. The device predicts the incoming plaintext based on previous observations; compresses, encrypts and authenticates data online; and pre-sends a portion of the ciphertext before receiving the entire plaintext. The performance of Predictive YASIR is evaluated using a simulation involving the Modbus/ASCII protocol. By considering broader message patterns and using predictive analysis, improvements in latency of $15.48 \pm 0.35\%$ are obtained.

**Keywords:** Legacy SCADA systems, security, low latency

## 1. Introduction

The United States power grid was built half a century ago, when network-based attacks were practically non-existent. New threats warrant retrofitting security in legacy networks in the power grid. Protecting a legacy network is difficult, however, because critical infrastructure components must communicate rapidly, but security slows the communications. This paper presents a predictive approach that optimizes the performance of the previous fastest security solutions.

## 2. Background

A power utility typically monitors and controls operations using a partially-unsecured, slow legacy network that 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 represen-

T. Moore and S. Shenoi (Eds.): Critical Infrastructure Protection IV, IFIP AICT 342, pp. 63–79, 2010.

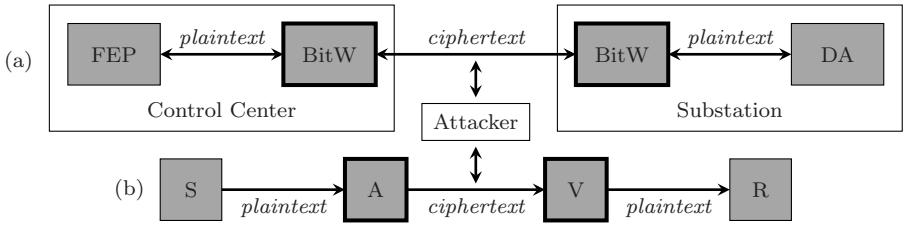


*Figure 1.* Typical BitW device setup.

tation of the data received from a front end processor (FEP), which exchanges messages with data aggregators (DAs) in substations. A FEP and DA connect via a slow legacy point-to-point network, which is often unsecured [4].

When communications are unsecured, an adversary can insert messages in network traffic and impersonate any device on the network. For example, an adversary could impersonate a FEP and command a DA to perform tasks that are not appropriate under normal operations. The adversary may 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.

An adversary who impersonates a DA can replay old DA status messages to the FEP, which would forward these messages to the 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 likely not detect the attack. Even if the operator discovers abnormalities through an alternative channel, understanding the scope of the problem is difficult in the absence of correct data; at the very least, this would slow the operator's response to the attack, giving the adversary time to subvert other substations.

Such attacks violate the message authenticity assumption made by the FEP and DA. Although FEPs and DAs can be upgraded to authenticate messages, the upgrades are prohibitively expensive and the upgraded devices would still have to communicate over the slow legacy network. Also, the required network upgrade is expensive. The cheaper and faster option is a bump-in-the-wire (BitW) device [10, 14, 15, 17] that secures all messages in a legacy network.

Two BitWs work in concert to authenticate a message (Figure 1(a)). Before a message from the FEP leaves the control center, the authenticator BitW reformats the original plaintext message into a ciphertext message with an added counter and a redundancy check (or message digest). The counter prevents an attacker from replaying old messages. The digest depends on the message, counter and a digest key shared by the pair of BitWs. Due to the cryptographic properties of the mechanism used 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 at 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

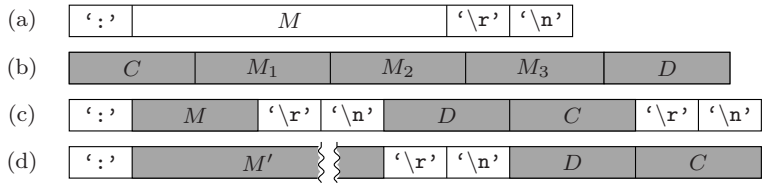


*Figure 2.* Message formats.

message and forwards it to the DA. Note that a BitW is still susceptible to attacks if the adversary gains access to an FEP or DA before the BitW.

When a DA sends a message to a FEP, the authenticator ($A$) and the verifier ($V$) switch roles. Due to this symmetry, we prefer to refer to the entities as sender ($S$) and receiver ($R$) (Figure 1(b)).

Figure 2 presents the Modbus/ASCII protocol message formats, which are used in this paper. Figure 2(a) shows the plaintext representation of a message while Figure 2(b) shows the position embedding (PE) ciphertext in blocks (described later). Figures 2(c) and 2(d) present the YASIR and Predictive YASIR ciphertext representations, respectively (described later). Note that the $C$ and $D$ denote the counter and digest, respectively. Also, the shaded areas in a message correspond to portions that are modified or generated by a BitW.

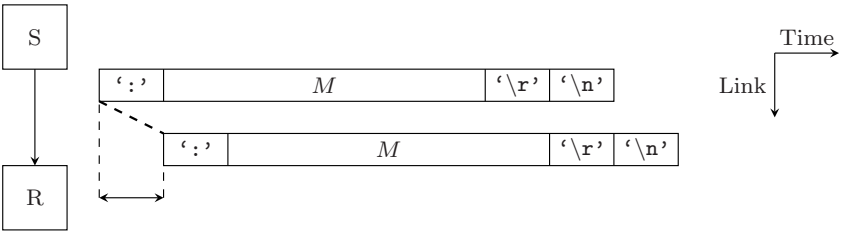


*Figure 3.* Transmission latency without authentication.

In a slow legacy power grid network, the end-to-end latency of a message typically should not exceed a certain value. Suppose this bound is 300 ms (Figure 3, where we use the diagram style from the YASIR paper [14]). 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 9,600 baud if a byte has 10 bits. Because of the low bandwidth, a BitW should not wait to receive the entire message before processing it, a practice known as "hold-back." If both BitWs in a pair hold-back a message that is longer than 144 bytes, the delay would exceed 300 ms. Instead, a BitW should forward each byte quickly or process the message online. The online processing must thwart an attacker who attempts to replay or modify ciphertext.

Figure 4 illustrates the notion of latency with hold-back, where both BitWs delay the entire message before forwarding it. As before, the shaded areas are modified or generated by a BitW. In the figure, the hold-back delays a message

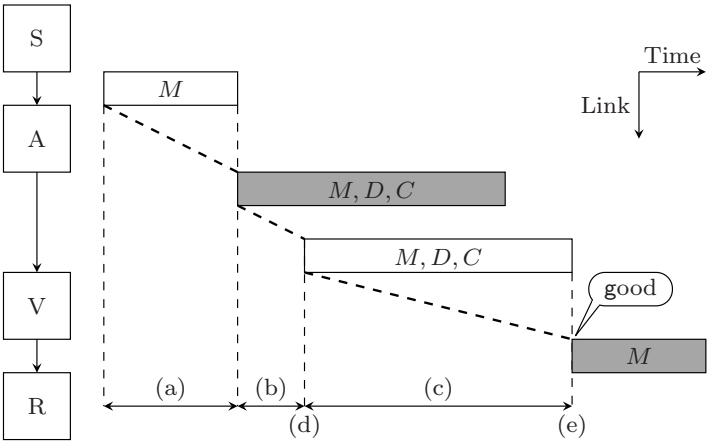


*Figure 4.* Latency with hold-back.

by time intervals (a) and (c), and the transmission delay is (b). The verifier starts receiving the message at time (d), but begins 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.

Next, we consider how online processing addresses each type of attack.

Tsang and Smith [14] have demonstrated that an online BitW can stop an attacker who attempts to replay an old message (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 entire 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 is received by the verifier, the received counter is ignored. If the calculated value is smaller than what is received, the verifier sets its own counter to the received value to synchronize with the authenticator. Before the counter overflows, the pair of BitWs reset their values to zero and change the digest key.

To prevent an attacker from modifying a message, the BitW pair appends a digest after the message but before the counter. This digest depends on the message, counter and digest key. The verifier compares the received digest with the value it calculates. 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 entire digest before it can forward the message to the receiver, thus incurring a byte-time of latency for each byte in the message. Although this is done in some earlier approaches, Wright, *et al.* [17] have suggested that the message be forwarded as soon as the verifier receives it, but to introduce a CRC error if an attacker modifies the message. This exploits the receiver's ability to detect random errors. We use a similar technique (described later) to enable the verifier to process the message online.

When an authenticator appends the digest to a message, it must ensure that the verifier can distinguish between them. Two options are available. The first option is to prepend the message length to the ciphertext. However, a common protocol like Modbus/ASCII (Figure 2(a)) has variable length messages and does not specify the length in a message. Consequently, to identify the length of a message in this protocol, an authenticator must hold-back the entire message. The second option is do it online by delimiting the ciphertext portions using 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 enhance its authenticity. Indeed, they constitute encoding inefficiencies required for online authentication. One of our goals is to eliminate these inefficiencies in online processing.

Note that we do not attempt 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 entire plaintext message; this would weaken the strength of the digest. 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.

## 3. Related Work

In most SCADA environments, message integrity is more important than confidentiality. An attacker can always learn the state of the system from the physical world. For example, an attacker may see the open flood gates of a dam and deduce that the control station has sent an "open flood gates" message and the substation has sent a "flood gates open" message. On the other hand, if no measures are in place to preserve message integrity, an attacker can cause actions such as opening the flood gates or shutting down a generator with significant negative repercussions. If a utility also needs to hide message content, a BitW device can provide confidentiality with zero latency using a stream cipher (e.g., AES in counter mode), which encrypts a plaintext stream by XORing it with a pseudo-random stream. Therefore, we consider related BitW solutions only if they authenticate messages, namely, SEL-3021-2, AGA SCM and YASIR. Also, we ignore solutions such as SEL-3021-1 because it does not protect message integrity and PNNL's SSCP embedded device because it does not represent a bump-in-the-wire solution.

### 3.1 SEL-3021-2

SEL-3021-2 [10] is a commercial off-the-shelf BitW device from Schweitzer Engineering Laboratories. The device uses the Message Authentication Protocol (MAP) [11] to provide integrity with HMAC-SHA-1 or HMAC-SHA-256

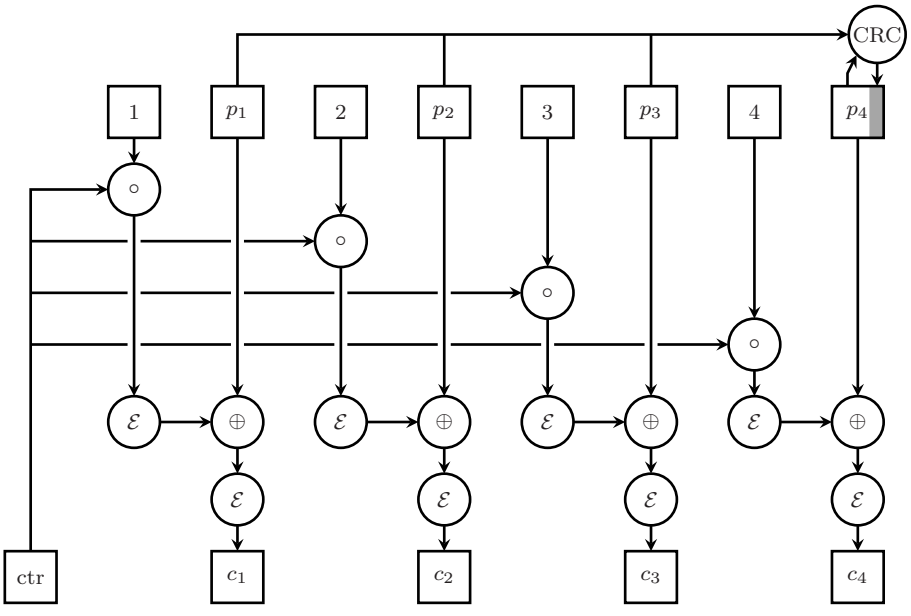


*Figure 5.* Position embedding (PE) mode.

digests. The specifications do not provide latency information for this device; however, Schweitzer does not recommend the use of SEL-3021-2 if low latency is desired.

## 3.2 AGA SCM

The American Gas Association SCADA Cryptographic Module (AGA SCM) [15] is a BitW device proposed by the AGA 12 Task Group. A reference implementation is available that provides several hold-back modes and one online mode [16]. The hold-back modes buffer the entire message before checking the digest and sending the message, slowing the message by a time interval in linear proportion to its size (see Figure 3). The online position embedding (PE) mode [17] 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 5).

As shown in Figure 5, the PE mode is AES-CTR followed by AES-ECB with the same key. The mode relies on CRC to authenticate messages. The symbols $\circ$ and $\oplus$ denote concatenation and XOR, respectively. 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 $\mathrm{ctr} \circ c_1 \circ c_2 \circ c_3 \circ c_4$. Blocks $p_i$ and $c_i$ are 16 bytes long; the message counter *ctr* is 14 bytes long and the block counters (1–4 in Figure 5) are 2 bytes long. Depending on the protocol, the CRC is 2 to 4 bytes long.

Figure 6 demonstrates the latency in the PE mode. The PE mode delays a message by 32 byte-times, because both BitWs in a pair buffer 16-byte blocks

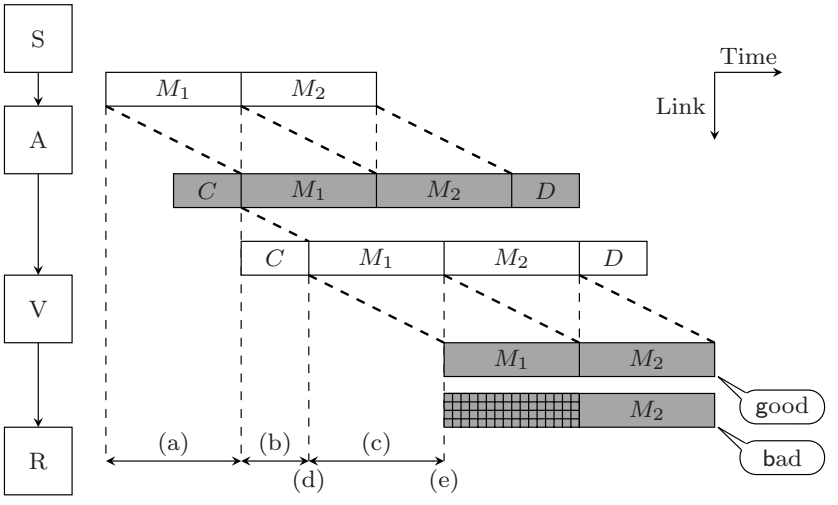


*Figure 6.* Latency in the PE mode.

of data to apply AES-ECB. $M_1$ 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 by (a) and (c). The verifier starts receiving the message at time (d) and begins to forward 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 12 modifies the manner in which AES-CTR generates counters. First, the authenticator increments a 14-byte session clock by one every $r$ microseconds, where $r$ is the "counter resolution." Next, 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 XORs the encrypted counter with a plaintext block like the standard AES-CTR. To avoid using the same counter for two messages, AGA 12 requires the counter resolution to be set so that an authenticator can send at most one message in a single session clock tick.

The PE mode relies on CRC for message integrity. Note that using CRC for plaintext protects against random errors, but not from malicious attacks on message integrity. Similarly, a BitW device cannot protect message integrity using AES-CTR or AES-ECB alone. The counter mode is malleable [3, 7], i.e., an adversary can modify the ciphertext with predictable changes to the CRC even without the encryption key.

The ECB mode is vulnerable to a known-plaintext attack, where an adversary who knows the plaintext of two messages can splice message portions to create a third message such that the CRC of the new message equals the CRC of one of the original messages. The PE mode prevents splicing and predictable

changes to the ciphertext by combining the CTR and ECB modes. However, using such a combination with CRC to ensure message integrity is not recommended because of potential security problems. One example is the cipher block chaining (CBC) mode, which may not provide message integrity protection with CRC. This result was demonstrated by Stubblebine and Gligor [13], who exploited the predictability of CRC to create undetectable bogus messages for Kerberos and remote procedure calls (RPCs).

Thus, the 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 adversary modifies a message in the PE mode, the verifier BitW scrambles the plaintext in the ECB step. Because of such scrambling, the receiver will detect a CRC error. The probability of this error is $2^{-h}$ where $h$ is the length of the CRC. CRC variants range in length from 8 bits to 32 bits, but AGA 12 specifies that this mode be used when the CRC is at least 16 bits.

## 3.3 YASIR

YASIR [14] is a BitW device that authenticates messages with at most 18 byte-times of overhead (Figure 2(c)). The actual overhead depends on the underlying protocol. With Modbus/ASCII, YASIR delays a message by about 16 byte-times. 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 [15], YASIR turns malicious errors into random errors by sending an incorrect CRC to the receiver if the digest is invalid. Compared with the PE mode, YASIR delays a message by fewer byte-times and authenticates a message with a fully standard and accepted cryptographic technique [9].

Figure 7 illustrates the latency obtained with YASIR. As mentioned above, the authenticator buffers two bytes of the message to detect its end, and the verifier buffers 14 bytes of the message to verify its digest. The total latency is 16 byte-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 if the digest is correct. If an attacker attempts to modify a message, the verifier sends the wrong CRC (crossed out) to the receiver.

# 4. Predictive YASIR Approach

Previous work [10, 14, 15, 17] focusing on the authentication of individual messages with a digest has tended to delay messages because of encoding inefficiencies (i.e., searching for special symbols in the plaintext and escaping special symbols in the ciphertext). Predictive YASIR attempts to eliminate these inefficiencies by examining broader message patterns.

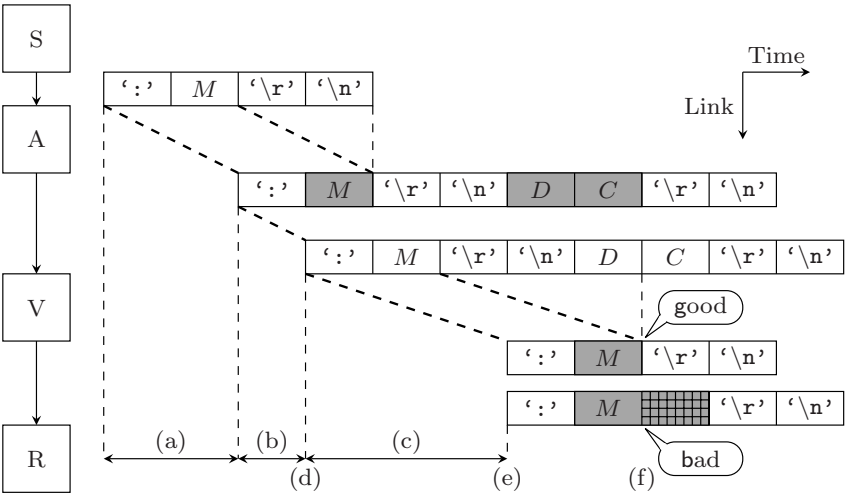


*Figure 7.* Latency with YASIR.

The approach employs a Bayesian network to predict the incoming plaintext and pre-send the prediction. As each byte reaches 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. Prediction is used to take advantage of the higher bandwidth for ciphertext that is provided by this optimistic *a priori* compression of plaintext. Intuitively, the solution augments YASIR by predicting plaintext messages to eliminate encoding inefficiencies.

Figure 8 illustrates latency with Predictive YASIR. The authenticator does not buffer the message, but must delay it by one byte-time (denoted by (a)). The verifier also does not buffer the message and must delay it by one byte-time (denoted by (c)). In addition, the verifier delays the message by $|D| - 1$ byte-times (denoted by (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 if the digest is correct. If an attacker attempts to modify a message, the verifier resets the receiver with `':'` instead of forwarding the entire message.

A BitW device can use compression to avoid overloading a channel that is close to its capacity. This is done by overlapping compressed messages with digests and counters. As shown in Figure 9, messages $M_2$ and $M_3$ are compressed and 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 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 digest for comparison. If the two digests match, the verifier forwards the last

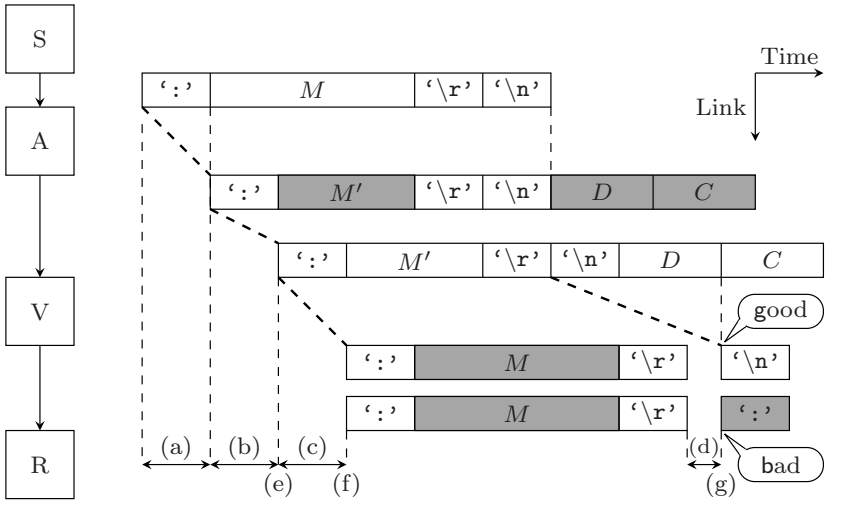


*Figure 8.* Latency with Predictive YASIR.

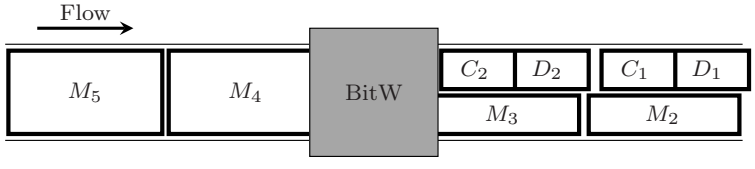


*Figure 9.* Overlapping compressed messages.

byte of the digest to the receiver, who now has the entire valid message. On the other hand, if the two digests differ, the verifier forwards the reset symbol to the receiver, who has to drop the incomplete message to adhere to the Modbus/ASCII protocol specifications.

Figure 10 illustrates the operation of Predictive YASIR. The sender begins message transmission (Figure 10(a)). The authenticator receives prefix $a$, predicts that the message is *abcd*, compresses and encrypts the prediction into ciphertext $xy$ and transmits $xy$ (Figure 10(b)). The authenticator receives prefix *abe*, changes its prediction to *abel*, compresses and encrypts the prediction to $xz$ and transmits the back-away to replace $y$ with $z$ (Figure 10(c)). The authenticator receives the entire message from the sender and transmits the digest $\tau$ to the verifier (Figure 10(d)). The authenticator transmits the counter $\upsilon$ to the verifier (Figure 10(e)). The verifier compares the received digest $\tau$ 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.

Note that 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 increment for the correct ciphertext (Figure 10(c)). When the authenticator receives the entire message, it sends the digest and counter for

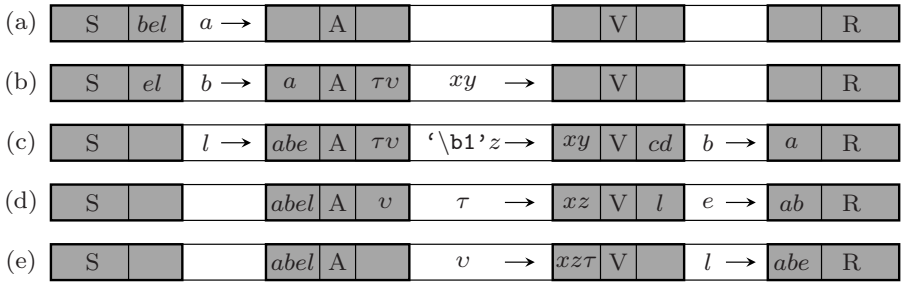


*Figure 10.* Example of Predictive YASIR operation.

the message (Figures 10(d) and 10(e)). The device then updates the weights in the Bayesian network (described in the next section).

Our solution is a non-intrusive way to "steal" bandwidth for security needs via data coding techniques and to utilize this bandwidth with the help of message prediction. The coding is effective because the data that is sent has sufficiently low entropy and can, therefore, be compressed and predicted to some extent. Note that if a BitW device compresses without predicting, it would have to wait for a larger portion of the message, incurring more latency.

# 5. Experimental Evaluation

This section describes the experimental methods used to evaluate the performance of Predictive YASIR and presents the experimental results. Interested readers are referred to [12] for additional details.

## 5.1 Modbus Protocol

Control centers often communicate with substations using Modbus/ASCII [8]. The sender transmits a message starting with the reset symbol `':'` (colon); when a device receives this reset symbol, it drops any incompletely received message (i.e., it resets itself). Every byte of a Modbus/ASCII message is encoded in ASCII. The sender appends a CRC and the terminating symbols `'\r\n'` (carriage return and a newline) at the end of the message.

Figure 11 shows an example Modbus/ASCII message. If the CRC of `0xABCD` is `0xEF`, then the sender encodes the hex message `0xABCD` in a Modbus/ASCII message `':ABCDEF\r\n'` which is `0x3A4142434445460D0A` in hex.

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; we leverage it to compress messages.

## 5.2 Scalable Simulation Framework

We use the Scalable Simulation Framework (SSF) [1] to conduct experiments with Predictive YASIR and measure the overhead of the approach. SSF sim-

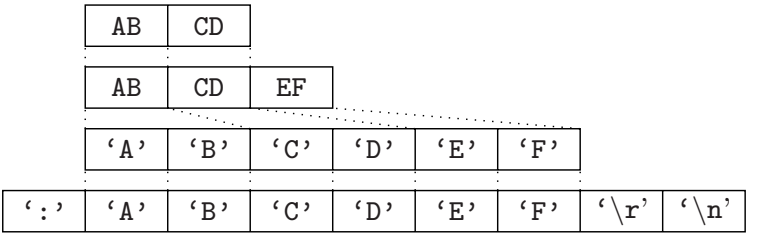


*Figure 11.* Modbus/ASCII message.

ulates networked entities that exchange events. The framework automates the collection of various statistics related to a simulation. If the simulation is large and runs slowly on a single computer, it can be scaled up with minimal effort by distributing the workload over a set of machines. The device entities exchange single byte events to ensure that they can process one byte at a time. To synchronize the timing, a BitW device outputs at most one byte for each byte that it receives, except after it has received the entire message.

## 5.3 BitW Devices

A BitW device has two ports: one for plaintext and the other 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.

## 5.4 Bayesian Network

In order to predict the incoming plaintext, the authenticator models the network traffic using a Bayesian network. A Bayesian network can be represented as a labeled directed acyclic graph. A vertex label is either a message prefix or an entire message and its frequency. All edges are directed from the prefixes to the full-length messages. A prefix vertex may have multiple outgoing edges. For example, the prefix `':'` has an edge to all observed messages because all Modbus/ASCII messages begin with this symbol. Note that a message vertex has inbound edges from all of its prefixes.

We implement the Bayesian network with a hash table of prefixes and a table of tuples $(m, f)$ corresponding to messages and their frequencies.

Figure 12 uses the symbol $\mathcal{H}$ 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, from which the authenticator predicts the most frequent message.

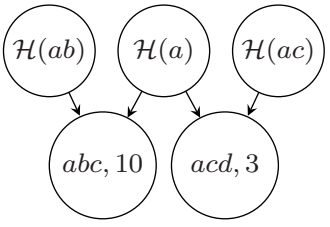


*Figure 12.* Bayesian network as a bipartite graph.

To prevent incorrect predictions, the authenticator uses Bayes' theorem to calculate the probability $\Pr(H|D)$ of the correctness of a hypothesis for the current data observation. A hypothesis expresses the message prediction while a data observation expresses 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. According to Bayes' theorem, $\Pr(H|D) = \Pr(D|H) \cdot \Pr(H) \,/\, \Pr(D)$.

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

- $\Pr(H)$ is the prior probability of a hypothesis. It is the ratio of the number $h$ of occurrences of the hypothesis to the total number $t$ of messages of the same or greater length that passed through the device. Therefore, $\Pr(H) = h/t$.

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

Substituting these terms into the Bayes' equation yields $\Pr(H|D) = (h \cdot o)/(t \cdot d)$.

## 5.5 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 10(c)). The discarded bytes are always the last bytes that the device sends because Predictive YASIR uses stream compression and encryption algorithms. For example, if the authenticator has to discard the last byte and replace it with $z$, then it sends the back-away signal `'\b1'`$z$. The verifier computes the digest for the final version of the message after it discards all the incorrect predictions.

## 5.6 Cipher Format

Modbus/ASCII uses only half the available bandwidth and our compression reclaims this space. The authenticator converts each ASCII character (‘0’ to ‘9’ and ‘A’ to ‘F’) into its equivalent four-bit representation: 0x0 to 0xF. The authenticator appends the digest and the counter after the terminating symbol ‘\r\n’. Thus the entire encrypted and authenticated message comprises the ‘:’ symbol, followed by message data, followed by ‘\r\n’, followed by the digest and the counter (Figure 2(d)).

## 5.7 Simulation Experiments

The simulation contains four components: one FEP, two BitWs and one DA (Figure 1(a)). The FEP connects to the plaintext port of the first BitW. The two BitWs connect to each other 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 entire 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 the ciphertext being longer than the plaintext.

The data used in the experiments was a trace from a GE XA/21 SCADA/Energy Management System that communicated with a GE D400 Substation Data Manager in a laboratory setting. The devices used the DNP3 protocol to communicate and record the trace. Before the simulation, the trace was converted to the Modbus/ASCII format for input into the SSF. The conversion was done because it is difficult to obtain real-world Modbus/ASCII traces.

YASIR and Predictive YASIR were run 30 times each. During the $i^{th}$ run of the simulation, the FEP had $10i$ unique messages to send to the DA. The number of unique messages was varied because the prediction ability can deteriorate with many unique messages. Each run lasted for 200,000 SSF ticks, enough to send every message more than once. The Bayesian network was reset after every run.

The simulation assumes that the BitWs have sufficient computational power so that prediction, compression, encryption and authentication operations do not affect latency. The average byte-time latency was measured in each test and the improvement percentage from YASIR to Predictive YASIR was computed. No comparisons were made with other approaches because they exhibit higher latency than YASIR.

The results of the simulation are presented in Figure 13. The byte-time latency for perfect prediction with a 12-byte HMAC digest is presented as a reference.

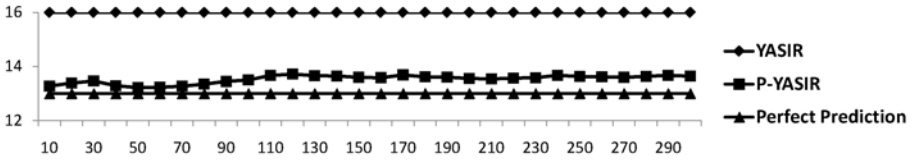


*Figure 13.* Average end-to-end latency of YASIR and Predictive YASIR.

These results demonstrate that Predictive YASIR has 15.48% less average latency than YASIR with a 95% confidence interval of 0.35 percentage points. Recall that Predictive YASIR is not compared with other bump-in-the-wire devices that provide message authenticity, because they have higher latency than the original YASIR.

Note that prediction does not degrade when the number of unique messages increases. The latency for Predictive YASIR is 13.52 byte-times with a 95% confidence interval of 0.06. In contrast, original YASIR always has a latency of 16 byte-times. When the authenticator makes a prediction mistake, successful recovery occurs with a back-away. The verifier determines all the messages to be valid because errors are not introduced in the ciphertext stream.

## 6. Future Work

Our future research will focus on several enhancements to Predictive YASIR. Also, it will attempt to validate its performance in real-world settings.

- **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 for predicting a natural phenomenon such as temperature. For example, consider a sensor that measures the temperature of water in a river. The temperature is usually 10°C, but assume that it recently increased to 11°C and will remain at this level. An authenticator that only uses statistics will continue to mistakenly predict the temperature as 10°C. On the other hand, a historical system would adjust its predictions even though the long-term majority of the temperature reports is 10°C.

- **SCADA Protocols:** This work has focused on the Modbus/ASCII protocol, but we believe that the technique should scale to other industrial control protocols such as DNP3 [2]. While it is easy to compress a Modbus/ASCII message, all sensors should have a finite and small number of states. For instance, the outdoor water temperature has only 100 integer states in Celsius and varies little.

- **Space:** 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 messages in the stream. Of course, more efficient stream statistics algorithms (e.g., [5, 6]) may be used.

- **Key Management:** We do not address key distribution, but concentrate on the BitW algorithm. Other researchers have addressed key distribution. For example, the AGA SCM design [15] specifies how devices negotiate the keys and ScadaSafe [16] implements these specifications. Key management implementations are easy to misconfigure or ignore, creating a false sense of security. Therefore, it is important to consider security policies that can be configured easily and correctly.

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

## 7. Conclusions

Message prediction and coding can be used to decrease latency arising from encoding inefficiencies, supporting the implementation of message authentication in slow legacy power grid networks. This method, which we call Predictive YASIR, is effective because message data in these networks has low enough entropy to enable BitW devices to predict and compress the data. Simulation results demonstrate a $15.48 \pm 0.35\%$ improvement in byte-time latency without compromising security. These savings can be significant in congested networks that require fast response as well as in other applications that suffer from encoding inefficiencies.

## Acknowledgements

This research is based on work supported by the Department of Energy under Award No. DE-OE0000097. We are also grateful to Paul Myrda of Electric Power Research Institute for providing data used in this research.

## References

[1] J. Banks, J. Carson, B. Nelson and D. Nicol, *Discrete-Event System Simulation*, Prentice Hall, Upper Saddle River, New Jersey, 2005.

[2] DNP Users Group, Overview of the DNP3 Protocol, Pasadena, California (www.dnp.org/About), 2010.

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

[4] T. Fleury, H. Khurana and V. Welch, Towards a taxonomy of attacks against energy control systems, in *Critical Infrastructure Protection II*, M. Papa and S. Shenoi (Eds.), Springer, Boston, Massachusetts, pp. 71–85, 2008.

[5] S. Ganguly, A. Singh and S. Shankar, Finding frequent items over general update streams, *Proceedings of the Twentieth International Conference on Scientific and Statistical Database Management*, pp. 204–221, 2008.

[6] P. Indyk and D. Woodruff, Optimal approximations of the frequency moments of data streams, *Proceedings of the Thirty-Seventh ACM Symposium on the Theory of Computing*, pp. 202–208, 2009.

[7] A. Menezes, P. van Oorschot and S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, Boca Raton, Florida, 2001.

[8] Modbus IDA, MODBUS Application Protocol Specification v1.1b, North Grafton, Massachusetts (www.modbus.org/specs.php), 2006.

[9] National Institute of Standards and Technology, Secure Hash Standard, FIPS Publication 180-3, Gaithersburg, Maryland (csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf), 2008.

[10] Schweitzer Engineering Laboratories, SEL-3021-2 Serial Encrypting Transceiver, Pullman, Washington (www.selinc.com/SEL-3021-2), 2007.

[11] Schweitzer Engineering Laboratories, SEL-3021-2 Serial Encrypting Transceiver Data Sheet, Pullman, Washington (www.selinc.com/WorkArea/DownloadAsset.aspx?id=2855), 2007.

[12] R. Solomakhin, Predictive YASIR: High Security with Lower Latency in Legacy SCADA, Technical Report TR2010-665, Department of Computer Science, Dartmouth College, Hanover, New Hampshire, 2010.

[13] S. Stubblebine and V. Gligor, On message integrity in cryptographic protocols, *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pp. 85–104, 1992.

[14] P. Tsang and S. Smith, YASIR: A low-latency, high-integrity security retrofit for legacy SCADA systems, *Proceedings of the Twenty-Third IFIP TC 11 International Information Security Conference*, pp. 445–459, 2008.

[15] A. Wright, AGA 12 Part 2-AKW Proposed SCADA Encryption Protocol (scadasafe.sourceforge.net/Protocol), 2006.

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

[17] A. Wright, J. Kinast and J. McCarty, Low-latency cryptographic protection for SCADA communications, *Proceedings of the Second International Conference on Applied Cryptography and Network Security*, pp. 263–277, 2004.