Dartmouth College

Dartmouth Digital Commons

Computer Science Technical Reports

Computer Science

4-18-2008

YASIR: A Low-Latency, High-Integrity Security Retrofit for Legacy SCADA Systems (Extended Version)

Patrick P. Tsang
Dartmouth College

Sean W. Smith
Dartmouth College

YASIR: A Low-Latency, High-Integrity Security Retrofit for Legacy SCADA Systems (Extended Version)[*][†]

Patrick P. Tsang[‡] and Sean W. Smith[§]

Department of Computer Science
Dartmouth College
Hanover, NH 03755
USA

Dartmouth Computer Science Technical Report
TR2008-617

April 18, 2008

Abstract

We construct a bump-in-the-wire (BITW) solution that retrofits security into time-critical communications over bandwidth-limited serial links between devices in legacy Supervisory Control And Data Acquisition (SCADA) systems, on which the proper operations of critical infrastructures such as the electric power grid rely. Previous BITW solutions do not provide the necessary security within timing constraints; the previous solution that does is not BITW. At a hardware cost comparable to existing solutions, our BITW solution provides sufficient security, and yet incurs minimal end-to-end communication latency.

Keywords: SCADA network security, bump-in-the-wire security retrofit, data privacy, authenticity and refreshness, hardware implementation

[*] This technical report is the extended version of the paper to appear in IFIP SEC 2008 under the same title. An earlier version of the paper appeared as Dartmouth Computer Science Technical Report TR2007-603 [TS07].

[†] This work was supported in part by the National Science Foundation, under grant CNS-0524695, the U.S. Department of Homeland Security under Grant Award Number 2006-CS-001-000001, and the Institute for Security Technology Studies, under Grant number 2005-DD-BX-1091 awarded by the Bureau of Justice Assistance. The views and conclusions do not necessarily represent those of the sponsors.

[‡] patrick@cs.dartmouth.edu

[§] sws@cs.dartmouth.edu

Contents

1 Introduction

1.1 SCADA Systems

Supervisory Control And Data Acquisition (SCADA) systems are real-time process control systems that monitor and control local or geographically remote devices. They are widely used in industrial facilities and critical infrastructures such as electric power generation and distribution systems, oil and gas refineries and transportation systems, allowing operators to ensure their proper functioning.

Electric power utilities, for instance, were among the first to widely adopt remote monitoring and control systems. Their earliest SCADA systems provided simple monitoring through periodic sampling of analog data, but have evolved into more complex communication networks. In this paper, we focus on SCADA systems for electric power generation and distribution. However, our proposed solution and discussion are applicable to many other SCADA systems.

Devices and Communications A SCADA system consists of *physical devices*, as well as *communication links* (we simply call them *links* from now on) that connect them together. Typical communications in a SCADA system include exchanging control and status information between master and slave devices. Master devices, most of which are PCs or *programmable logic controllers (PLCs)*, control the operation of slave devices; a slave device, e.g., a *remote terminal unit (RTU)*, can be a sensor, an actuator, or both. Sensors read status or measurement data of field equipments such as voltage and current, whereas actuators send out commands or analog set-points such as opening or closing a switch or a valve.

Most SCADA systems have traditionally used low-bandwidth links, e.g., radio, direct serial and leased lines, with typical baud rates from 9600 to 115200. They are known as *serial-based SCADA systems*. Communication protocols used in these systems are very compact—messages are short, and slave devices send information only when polled. Popular protocols include Modbus (`http://www.modbus.org/`) and DNP3 (`http://www.dnp.org`).

Security Trouble Many serial-based SCADA systems in operation today were deployed decades ago with availability and personnel safety as the primary concerns, rather than security. As with any complex systems not designed to withstand adversarial action, these systems are vulnerable to a variety of malicious attacks such as sniffing and tampering. The risks due to such a lack of security in these systems are ever increasing, as an initial protection of "security through obscurity" breaks down.

First, after initial dependence on proprietary elements, it is now common to build SCADA systems using commercial off-the-shelf (COTS) hardware and software that speak open communication protocols, the technical internals of which are often easily accessible. Second, many utilities have replaced, to various extent, their private networks by public ones such as the Internet. Their SCADA networks and corporate networks have also become highly inter-connected to achieve efficient information exchange—leading to increased risks of intentional or inadvertent exposure to the Internet. Finally, teams of sophisticated hackers are now employed by criminal organizations or terrorists to break into these systems.

Retrofitting Security Failures of critical infrastructures could lead to devastating consequences. As an example of cyber-attacks on critical infrastructures, in 2001, an Australian man hacked into a computerized sewage management system and dumped millions of liters of untreated waste into

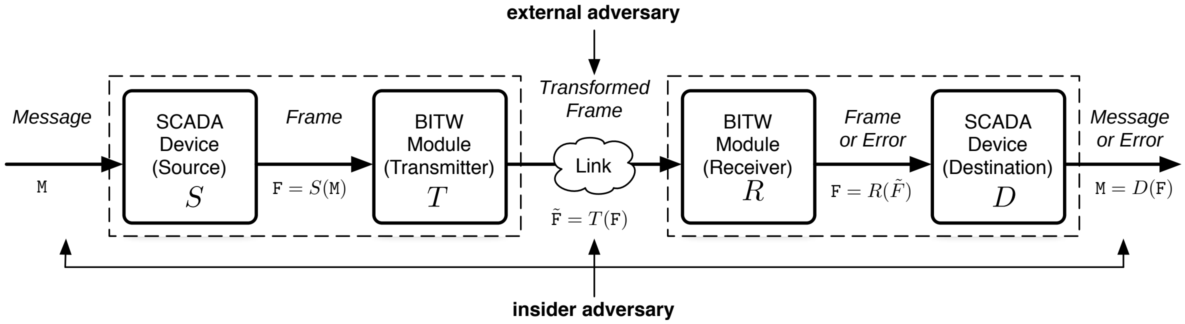
external adversary
Message
M
SCADA Device (Source)
S
Frame
$\mathtt{F} = S(\mathtt{M})$
BITW Module (Transmitter)
T
Transformed Frame
Link
$\tilde{\mathtt{F}} = T(\mathtt{F})$
BITW Module (Receiver)
R
Frame or Error
$\mathtt{F} = R(\tilde{F})$
SCADA Device (Destination)
D
Message or Error
$\mathtt{M} = D(\mathtt{F})$
insider adversary

Figure 1: System and attack model for "bump-in-the-wire" approach.

local parks and rivers [Smi01]. It is therefore paramount to secure SCADA systems against malicious attacks. In the long run, existing insecure SCADA systems may eventually be replaced by newer ones built with better technologies and with security as a primary goal—we will gradually see devices that are computationally more powerful, links with higher bandwidths, as well as devices and protocols with built-in security, e.g., DNPSec [MPPW06] and IEC 61850 (`http://www.61850.com/`).

Nonetheless, for the next several decades (the expected lifetime of many SCADA equipments spans from 20 to 50 years), achieving security requires *non-intrusively retrofitting it* to existing insecure and legacy SCADA systems, as it is economically infeasible, if not technically impossible, to simply throw away the existing infrastructures overnight. In such an effort, several *"bump-in-the-wire" (BITW)* solutions have been developed. In a BITW solution, two hardware modules are inserted into the link connecting two communicating SCADA devices, one next to each of the device, as depicted in Figure 1. These modules transparently augment the necessary security through mechanisms such as encryption and authentication.

1.2 The Challenge

BITW solutions secure SCADA communications at the expense of incurring end-to-end communication delay, due to the processing and buffering in the BITW modules. Buffering can be prohibitively expensive in low-bandwidth links. For instance, a serial link at 9600 baud per second has a *byte time* (i.e., the time to send one byte of data) of roughly 1ms. If each of the two BITW modules buffers up a message of 20 bytes before processing it, then a timing overhead of 40ms is incurred, due to buffering alone. If the message has 250 bytes, the overhead becomes 0.5s.[1]

Such an overhead could be intolerable for serial-based SCADA systems that have timing constraints on communication latencies. For example, the exchange of event notification information for bus and transformer protection function between *Intelligent Electronic Devices (IEDs)* within a power substation must be accomplished within 10ms, and the maximum delivery time for information such as response to data poll and phasor measurements is up to 0.2s [IEE05].

As we will see in Section 2 when we review some of the existing solutions, retrofitting *data privacy* to the communications in serial-based SCADA systems, even the time-critical ones, is a relatively trivial task; the real challenge lies in retrofitting *data authenticity and freshness* in a

[1] A typical SCADA message has a length of roughly 20 bytes. However, some SCADA protocols allow a maximum message-length of more than 250 bytes.

timely manner, as the straightforward application of conventional data authentication techniques does not provide the required timing guarantee: the BITW module at the receiving end of the communication must *"hold back"* the message, i.e., it must wait until the receipt of the entire message and its authentication information, before relaying the message to the destination SCADA device, if the message is indeed authentic and fresh. This incurs a latency dependent linearly on the length of the message being secured.

1.3 Our Contributions

We present *Yet Another SecurIty Retrofit*, or *YASIR*, which is a novel BITW solution for retrofitting security to time-critical communications in serial-based SCADA systems. To the authors' best knowledge, our solution is the first that achieves all of the following goals simultaneously:

- **High Security.** *YASIR* provides data authenticity and freshness, and optionally data privacy, against not only eavesdroppers but stronger adversaries such as insiders, at a security level of 80 bits.[2]
- **Low Latency.** *YASIR* incurs an overhead of at most 18 byte times, irrespective of the length of the message being authenticated, and can hence secure time-critical SCADA communications.
- **Comparable Cost.** *YASIR*'s BITW modules have hardware costs comparable to many existing solutions. Deploying *YASIR* is thus economically practical.
- **Standard and Patent-free Tools.** All cryptographic tools and techniques used in *YASIR*, such as AES-CTR and HMAC, are NIST-standardized and patent-free.

The rest of this paper is organized as follows. In Section 2, we review several existing BITW solutions. Section 3 covers SCADA preliminaries. Section 4 studies the threat model and security goals of BITW solutions. We give an overview to our solution in Section 5 and provide the details of its actual construction in Section 6. Section 8 concludes the paper. In the extended version of this paper [TS08], we evaluate *YASIR*'s security, performance and costs in depth. We also report on a micro-controller prototype of *YASIR*.

2 Existing Solutions

We do not consider encryption-only solutions as retrofitting only data privacy does not provide sufficient security. Also, since we are interested in non-intrusively retrofitting security into legacy SCADA communications, we do not consider non-BITW solutions, i.e. solutions that require replacing the link with one of a higher bandwidth, e.g., from RS-232 to Ethernet, and/or upgrading the (software or hardware of) the SCADA devices to allow for newer technologies such as IPSec [KA98].

Below, we review several existing BITW solutions, all of which fall short in some critical property: they don't provide data authenticity against realistic attacks, or they delay messages too long. Table 1 summarizes this picture.

[2] A security solution attains a security level of ℓ bits if brute-forcing a space of 2^ℓ possibilities is the most effective strategy for an adversary to break the solution's security.

Approach	Bump-In-The-Wire?	Confidentiality?	Integrity?	Strong Threat Model?	High Security Level?	Low Latency? (byte times)
SEL 3021-1	Yes	Yes	**No**	**No**	Yes	Yes (5)
SEL 3021-2	Yes	Yes (option)	Yes	Yes	Yes	**No**
AGA12/Cisco, PE mode	Yes	Yes (option)	Yes	**No**	**No**	Yes (~32)
AGA12/Cisco, other modes	Yes	Yes (option)	Yes	Yes	Yes	**No**
PNNL SSCP BITW	Yes	Yes (option)	Yes	Yes	Yes	**No**
PNNL SSCP embedded	**No**	Yes (option)	Yes	Yes	Yes	Yes (<10)
YASIR (our approach)	**Yes**	**Yes (option)**	**Yes**	**Yes**	**Yes**	**Yes (≤18)**

Table 1: Previous BITW solutions for securing legacy SCADA communications all fall short in some critical property; the one previous approach that provides the critical property is not BITW. Our approach meets all the criteria.

2.1 SEL's Serial Encrypting Transceiver

The SEL-3021 Serial Encrypting Transceiver from Schweitzer Engineering Laboratories, Inc (SEL, `http://www.selinc.com`) is a BITW module for securing RS-232 serial links between SCADA devices against malicious attacks. Both available models, SEL-3021-1 and SEL-3032-2, support all standard SCADA protocols, including DNP3-Serial and Modbus/RTU, at data rates up to 115200 bps.

The *SEL-3021-2* provides data authenticity through HMAC-SHA-1/-256. It also optionally provides data privacy through AES-CTR-128. Unfortunately, SEL-3021-2 does not provide an upper-bound on the delay it may incur [Sch]. In fact, SEL suggests that SEL-3021-2 *"may not be suitable to secure links that require time-critical communications with low latency (i.e., links for electrical tele-protection)"* [Sch]. Another model in the family, the *SEL-3021-1*, is an encryption-only solution.

2.2 AGA's SCADA Cryptographic Module

The American Gas Association (AGA) (`http://www.aga.org/`) Task Group 12 designed the *SCADA Cryptographic Module (SCM)* [Ame06] as a BITW solution that retrofits data authenticity to SCADA communications while maintaining the performance requirements. AGA's SCM provides several cipher-suites to choose from. The most secure ones use AES-CTR for data privacy and HMAC-SHA-1/-256 for data authenticity. Unfortunately, messages must be held back by the receiving SCM using these cipher-suites.

PE Mode of Operation In one of the cipher-suites provided by AGA's SCM, data authentication is achieved by operating AES in the *Position-Embedded (PE) mode* [WKM04]. Using this cipher-suite, SCMs provide data authenticity with an overhead of only 32 byte times, regardless of the message-length. To the best knowledge of the authors, AGA's SCM and our *YASIR* are the only BITW solution for legacy SCADA systems that provide data authenticity without message hold-back.

Unfortunately, AES operating in the PE mode attains a security level of only 16 bits at maximum, which is far below the generally accepted minimum of 80-bit level of security: with a probability of at least 2^{-16}, SCADA devices protected by SCMs will accept maliciously crafted messages as authentic. As a remedy, SCMs rely in addition on traditional HMAC for more secure data authentication. However, as pointed out by Majdalawieh et al [MPPW06], although unauthentic messages can eventually be detected by the SCM, the late detection can't stop the SCM from forwarding them to the SCADA devices. Moreover, AES in PE mode is proven secure only under known-plaintext attacks [WKM04]. Hence, this solution is not guaranteed to be secure against stronger and yet realistic attacks, such as chosen plaintext and/or ciphertext attacks launched by, e.g., a compromised employee working in the control center.

2.3 PNNL's Secure SCADA Communications Protocol

A SCADA communications authenticator technology is under development by a group led by Mark Hadley at the Pacific Northwest National Laboratory (PNNL, `http://www.pnl.gov/`). In PNNL's solution, SCADA messages are "wrapped" by an authenticator and potentially some other information such as a unique identifier. Their solution is effectively a protocol wrapper that converts an insecure SCADA protocol into their Secure SCADA Communications Protocol (SSCP).

PNNL's technology is being implemented both as a BITW solution and an embedded solution [Had07]. The BITW solution requires message hold-back. The embedded solution is fast but is not a BITW solution: it requires upgrading the hardware and/or software of the SCADA devices.

3 Preliminaries

3.1 SCADA Protocols

The data link layer of a SCADA protocol specifies how control and data messages are encoded into bit-sequences known as *frames* for transmission over the communication link. Let $||$ denote the concatenation of (bit- or octet-) strings. A frame $\mathtt{F}$ has the form $\mathtt{s}||\mathtt{H}||\mathtt{P}||\mathtt{e}$.

The header $\mathtt{H}$, if present,[3] may contain control information about the frame such as its length. The payload $\mathtt{P}$ contains a message in its encoded form and usually has variable length. The starting symbol $\mathtt{s}$ and the ending symbol $\mathtt{e}$ are bit-sequences distinct from any code symbols used in the rest of the frame so that a SCADA device can detect frame boundaries unambiguously. In many asynchronous protocols including Modbus/ASCII and DNP3-Serial, frame boundaries can be recognized within two byte times. In Modbus/RTU, which is a synchronous protocol, a silence of 3.5 byte times indicates the end of a frame.

3.2 A Classification of Legacy SCADA Protocols

There are more than a hundred SCADA protocols in use today, many of which are closed and proprietary. A practical BITW solution should make few assumptions about the SCADA protocol it is protecting, so that it can be used to, upon simple configuration, protect a majority of protocols.

[3] In some SCADA protocols such as Modbus, frames do not have a header.

Our solution to be presented in Section 6 does require certain assumptions to be made about the underlying protocols, but is otherwise designed so that those assumptions hold for the majority of SCADA protocols. Specifically, we introduce a classification of SCADA protocols into Type-I and Type-II in the following; our solution assumes that a SCADA protocol is of Type-I or Type-II, or both.

- *Type-I Protocols.* The last few octets in the frame is a checksum of (a part of) the rest of the frame produced according to certain known CRC algorithm. A receiving SCADA device flags an error and drops the frame if the checksum is incorrect. For example, in Modbus/ASCII (resp. DNP3), the last two octets is a CRC-16 on the rest of the payload (resp. the previous 16 bytes).
- *Type-II Protocols.* A frame contains in its fixed-sized header information from which the length of the frame (and thus that of the payload) can be calculated. If the actual length of the frame is *smaller than*[4] the length as calculated using the header information, a receiving SCADA device flags an error and drops the frame.[5] For example, DNP3 frames contain in the header the size (in terms of the number of 16 octets) of the payload excluding the CRCs.

Most existing SCADA protocols are of Type-I or Type-II: it has long been a commonly adopted practice to append CRC checksums to frames at the data-link layer of a communication protocol for detecting transmission errors. Similarly, length verification is employed in many communication protocols as a mechanism for detecting errors. Moreover, it is fairly easy to check if a protocol is of one of the types and determine the CRC algorithm used. Even if the protocol is closed and proprietary, one can do so by examining several actual frames coming out of a real SCADA device speaking that protocol.

3.3 Formalizing BITW Solutions

As Figure 1 illustrates, a *source* SCADA device S converts messages such as data or control information into frames for transmission. We overload S to denote the function that models the device, which takes a message $\mathtt{M}$ as input and outputs the corresponding frame $\mathtt{F}$. Similarly, the *destination* SCADA device D is modeled as a function D, which takes a frame $\mathtt{F}'$ as input and output an *error*, if $\mathtt{F}'$ fails to pass certain conformance checks such as the random-error detection, or else the corresponding original message $\mathtt{M}'$.

If no error was introduced (randomly or maliciously) into $\mathtt{F}$ during its transmission (i.e., if $\mathtt{F}' = \mathtt{F}$), then a correct pair of S and D must always give $D(\mathtt{F}') = D(\mathtt{F}) = D(S(\mathtt{M})) = \mathtt{M}$. If $\mathtt{F}' \neq \mathtt{F}$, then D may or may not return an error, depending on whether F' passes the conformance checks in D. Virtually all SCADA devices have random-error detection mechanisms such as CRCs, and are thus capable of catching most random errors.

Now, any BITW solution injects two hardware modules into the link in the model, one next to S and the other next to D, which we call the *Transmitter* T and the *Receiver* R respectively. Refer to Figure 1 again for a diagrammatic illustration. Again T is overloaded to denote the function that models the Transmitter, which takes each frame $\mathtt{F}$ output by S as input and returns

[4] Replacing "smaller than" with "different from" results in a more restrictive assumption as there may exist protocols that reacts to frames longer than what is specified in the header by, rather than dropping them as error, truncating them to the specified length and operating on the truncation.

[5] This implicitly implies that the device will do the same if the frame is shorter than a header.

the corresponding transformed frame $\tilde{\mathtt{F}}$ to be transmitted over the insecure channel. Similarly, the Receiver R is modeled as a function R that takes in a transformed frame $\tilde{\mathtt{F'}}$ and outputs either an *error*, or the corresponding original frame $\mathtt{F'}$ to be given to D. If no error was introduced (randomly or maliciously) into $\tilde{\mathtt{F}}$, i.e., $\tilde{\mathtt{F'}} = \tilde{\mathtt{F}}$, a correct pair of T and R must always give $R(\tilde{\mathtt{F'}}) = R(\tilde{\mathtt{F}}) = R(T(\mathtt{F})) = \mathtt{F}$. In most existing BITW solutions that provides data authenticity and freshness, if for whatever reason $\tilde{\mathtt{F'}} \neq \tilde{\mathtt{F}}$, then R should output an error with overwhelming probability. Effectively, R acts as a guard in these solutions and discards all malformed frames so that D won't even see them.

We note that while S and D do not output the corresponding output until they receive the input in its entirety, this is not necessarily the case for T and R: they could start outputting part of the output after having received only part of the input. For example, T and R output data of size equal to an AES-block for every receipt of data of the same size in AGA's SCMs; in the solution we are going to propose, T and R output a byte upon receiving a byte.

Finally, a SCADA device can be the source at one time and the destination at another (but never at the same time). A BITW module in operation will thus switch between the role of a Transmitter and that of a Receiver accordingly.

4 Security Requirements

A BITW solution retrofits security to legacy SCADA communications to thwart adversarial attacks. Here we study the adversary's goals and capabilities when attempting to launch those attacks and the security properties a BITW solution must possess to defend against them. A more formal treatment of the discussion in this section can be found in Appendix B.

4.1 Threat Model

When attempting to break the security provided by a BITW solution, the adversary may interact with the various components in the SCADA system through all interfaces exposed to him in any malicious and arbitrarily intelligent way, in order to increase his advantage in launching a successful attack. Formalizing a threat model by correctly identifying the adversary's capabilities is thus critical in the evaluation of the security of any BITW solution.

Communication Links It is impossible to keep the adversary away from the entirety of links as they travel through a long distance to connect end SCADA devices together. This is the case for private leased lines, and even more so for public networks such as the Internet. As Figure 1 shows, in our threat model, *links are insecure:* an adversary may arbitrarily sniff, tamper, inject and replay communications.

SCADA Devices and *YASIR* Modules We assume that the adversary knows how S, D, T and R operate, i.e., the complete specification of how they convert an input into the corresponding output. For SCADA systems that speak open protocols, such information is readily available to the public. Even for systems that use closed and proprietary standards, one should that the same information can be obtained by the adversary through reverse-engineering or insider leaks.

However, we assume that the adversary can't physically tamper with any of them, e.g., manipulate their internal operations, or extract or overwrite their internal states, including the secret keys in the case of T and R. Assumptions on physical tamper-resistance as such are inevitable for most cryptographically secure hardware. One usually achieves physical tamper-resistance by carefully controlling who can have physical accesses to the hardware, and/or by introducing tamper-resistant/-responsive mechanisms to the hardware itself.

Insider Attacks If there existed security boundaries around the substations and the control centers, then attacking the communication links would be all the adversary could possibly do. Unfortunately, such security boundaries do not exist. For example, an adversary may physically break into an under-guarded substation, compromise an employee working in the control center, or remotely hack into the computers auditing the SCADA devices.

In our threat model, *SCADA devices and the attached YASIR modules are insecurely located:* as depicted in Figure 1, an adversary may feed D with maliciously crafted inputs and learn the corresponding outputs at T; he may also feed R with maliciously crafted inputs and learn the corresponding outputs at D.

As we have discussed, the security of AGA's SCMs using AES operating in PE mode assumes the absence of any insider. We think that this is a rather unrealistic assumption. The actual security of their solution is unclear in practice when the assumption ceases to hold.

4.2 Security Goals

A BITW solution must provide data authenticity and freshness to SCADA communications. If desired, it must also provide data privacy.

Data Authenticity and Freshness A destination SCADA device D equipped with a *YASIR* Receiver R only accepts a transformed frame $\tilde{\mathtt{F}}$ as valid, i.e., it outputs the corresponding original message $\mathtt{M}$ instead of flagging an *error*, if:

- *(Authenticity.)* $\mathtt{M}$ was an input to a source SCADA device S equipped with the *YASIR* Transmitter T that shares its secret keys with R.
- *(Freshness.)* $\tilde{\mathtt{F}}$ is fresh, i.e., not a replayed/re-ordered frame. More precisely, if T output any other transformed frame $\tilde{\mathtt{F}'}$ after outputting $\tilde{\mathtt{F}}$, R has not been given $\tilde{\mathtt{F}'}$ as an input.

Data Privacy No information about the corresponding original frame can be inferred from a transformed frame in transit, except perhaps its size. More precisely, an adversary is allowed to choose two messages $\mathtt{M}_0$ and $\mathtt{M}_1$ such that their corresponding frames, $\mathtt{F}_0$ and $\mathtt{F}_1$ respectively, as output by S, are distinct but of the same length. The goal of data privacy is so that when given the transformation $\tilde{\mathtt{F}}$ by T of either $\mathtt{F}_0$ or $\mathtt{F}_1$, the adversary does not know which is the case.

We remark that there are scenarios when data privacy is *not* a concern. For example, it is fine for an IED to report the current temperature reading to another IED within the same substation over an unencrypted channel because an adversary who has broken into the substation might as well go to read off the temperature directly from the sensing IED instead of tapping into the serial link. There are even scenarios when data privacy is *undesirable*, such as when a message has multiple recipients. One example is when the control center wants to broadcast the same control message to all RTUs. Also, one might want to install a logging device that audits all the messages leaving or entering a SCADA device.

As will become clear, our proposed solution provides both data privacy and data authenticity and freshness by default, and yet can easily be modified to provide only data authenticity and freshness and send transformed frames in cleartext.

5 Solution Overview

5.1 An Observation

Recall that the BITW receiver module R acts as a guard for the destination SCADA device D in most existing solutions. R can't decide if a frame is authentic and fresh and hence can't start relaying it until the receipt of the entire frame and its authentication tag. The latency thus grows linearly with the frame length. AES in PE mode used in AGA's SCM as previously discussed is, however, a novel exception. R starts relaying the frame to D before the authenticity of the frame is known. However, R operates on the frame in such a way that, with probability close to 1, D will flag a CRC error and drop the frame if it has been tampered.

In a sense, AGA's solution converts random-error detection, already built in to the legacy SCADA devices, into a mechanism for verifying data authenticity against malicious attacks. In their solution, the conversion relies on the "real-or-random indistinguishability" property [BDJR97a] of AES when used as a block cipher. However, this solution has three drawbacks: (1) one 16-byte block of data must be buffered at each of both BITW modules. (2) There is a non-negligible probability (as high as 2^{-8} or 2^{-16}, depending on the underlying protocol) that a maliciously tampered frame can get through R and be operated on by D. (3) This approach is proven secure only against known-plaintext attacks, but not against stronger and yet still very realistic attacks such as chosen-plaintext and/or chosen-ciphertext attacks.

5.2 Our Approach

Our solution shares the same idea of converting random-error detection to data authenticity and freshness checking, but is different in how that conversion is done, which enables our solution to offer three advantages: (1) our BITW modules operate on a frame as a stream of bytes instead of 16-byte blocks so that latency to due buffering is minimal. (2) Our solution uses HMAC (but in a way so that no message hold-back is required) so that R knows, at a 80-bit security level, when a frame has been tampered with, in which case R is always capable of forcing D to drop the frame. (3) The use of HMAC also allows our solution to be secure against stronger and yet realistic attacks, namely chosen-plaintext-and-ciphertext attacks.

To provide data privacy and freshness, our solution makes appropriate use of encryption and sequence numbers respectively, as we will describe in details in the next section. However, if we ignore data privacy and freshness for now, the following explains at a high level how our solution provides data authenticity.

For each frame `F` the BITW Transmitter T receives from the source SCADA device S, T appends an HMAC-SHA-1-96 on `F` to the back of `F` and sends it off to the insecure channel. This can done without holding back the frame. At the other end, the BITW Receiver R relays every byte it gets from the insecure channel to the destination SCADA device D, but with a delay of 14 byte times. Since a HMAC-SHA-1-96 MAC has 12 bytes, by the time R is about to relay last byte, it will have already received the whole HMAC and will thus be able to verify the authenticity of the received frame. Now if the HMAC verifies, all R has to do is to finish up relaying the frame by sending the last byte. However, if the HMAC does not verify, R manipulates the last byte to cause the conformance checks at D to fail.

6 Solution Details

We now present the construction for our *YASIR* Transmitter and *YASIR* Receiver. Our single *YASIR* Transmitter construction works for both Type-I and Type-II SCADA protocols; we have two *YASIR* Receiver constructions, one for each type of SCADA protocols. If a SCADA protocol to be secured is of both Type-I and Type-II, then either *YASIR* Receiver construction may be used.

Let Hash denote the cryptographic hash function SHA-1, the output of which has an octet-length of $\ell_H = 20$. Let Hmac denote the HMAC function HMAC-SHA-1-96, the output of which has an octet-length of $\ell_M = 12$. Further let Encrypt denote the encrypting (resp. decrypting) function AES-CTR-128, which takes a nonce of octet-length $\ell_N = 4$, and a plaintext (resp. ciphertext) of any length, and outputs the corresponding ciphertext (resp. plaintext) of the same length. Finally, let Crc denote the CRC algorithm used by the Type-I SCADA protocol, which takes a frame and outputs boolean answer of the validity of a frame, as described in Section 3.

The BITW Transmitter T and Receiver R share a 128-bit AES key ek and a 160-bit HMAC-SHA-1 key hk. These keys are re-negotiated on a regular basis, such as once every day.[6] T and R keep counters ctr_T and ctr_R of octet-length $\ell_S = 4$ respectively, both of which are reset to zero every time keys are re-negotiated.[7]

6.1 *YASIR* Transmitter

On input an incoming frame $\mathtt{F} = \mathtt{s}||\mathtt{H}||\mathtt{P}||\mathtt{e}$, the *YASIR* Transmitter T does the following:

1. Output the corresponding transformed frame $\tilde{\mathtt{F}} = \mathtt{s}||\mathtt{CTXT}||\mathtt{x}||\mathtt{MAC}||\mathtt{SEQ}||\mathtt{e}$, where

$$\mathtt{CTXT} = \textsc{Encrypt}_{\mathsf{ek}}(\mathsf{ctr}_T, \mathtt{H}||\mathtt{P}),\ \mathtt{MAC} = \textsc{Hmac}_{\mathsf{hk}}(\mathsf{ctr}_T||\mathtt{CTXT}),\ \mathtt{SEQ} = \mathsf{ctr}_T,$$

   and $\mathtt{x}$ is, like $\mathtt{s}$ and $\mathtt{e}$, a special symbol distinct from any code symbol used in the rest of the frame. It indicates the end of $\mathtt{CTXT}$ and hence the start of $\mathtt{MAC}$.[8]

2. Increments ctr_T by 1.

In a nutshell, T transforms $\mathtt{F}$ to $\tilde{\mathtt{F}}$ by first encrypting $\mathtt{F}$'s content (i.e., header and payload) for data privacy, then appending a "time-stamp" on the ciphertext with a unique sequence number for data authenticity and freshness, and finally appending the sequence number itself.

The above describes how T operates on an input to produce the corresponding output, without detailing the timeliness of the operation, i.e. which part of the output is available when. We specify this in the following.

[6] Key management is outside the scope of this paper. One can borrow key distribution and re-negotiation techniques from other existing BITW solutions.

[7] There is no practical chance of exhausting a 4-byte counter in any SCADA deployment.

[8] Alternatively, one can use a character escaping mechanism to allow for proper frame parsing.

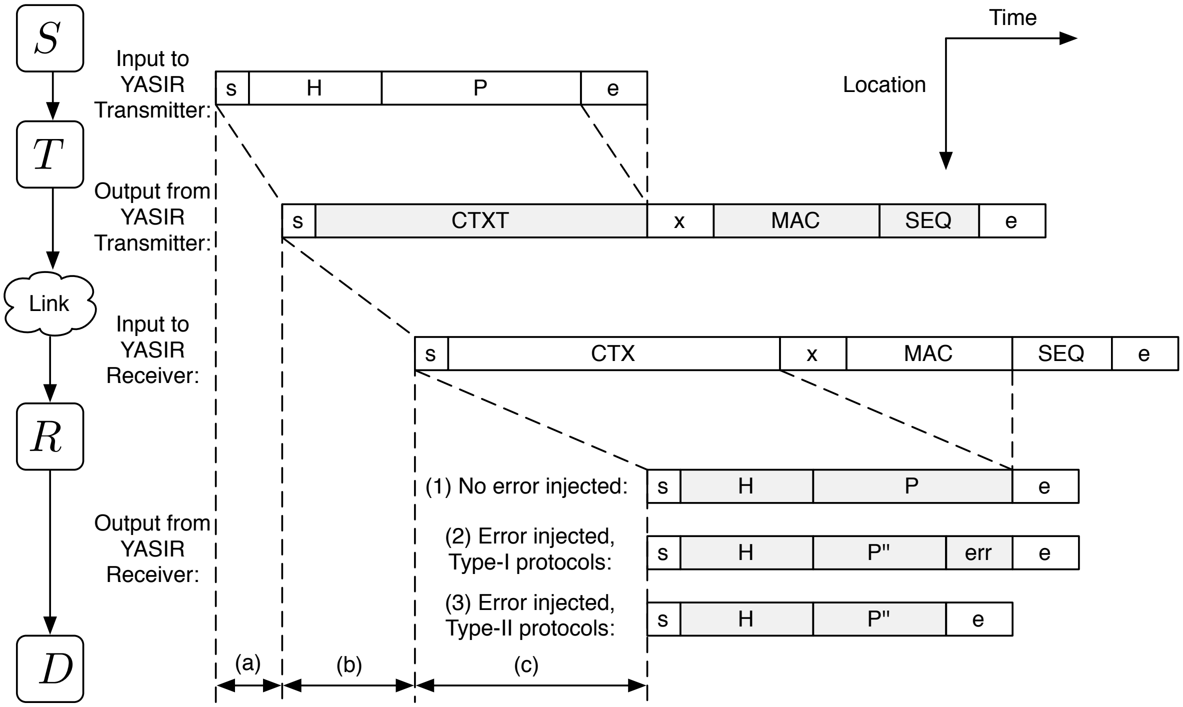
S
T
Link
R
D
Input to YASIR Transmitter:
Output from YASIR Transmitter:
Input to YASIR Receiver:
Output from YASIR Receiver:
Time
Location
s
H
P
e
CTXT
x
MAC
SEQ
CTX
(1) No error injected:
(2) Error injected, Type-I protocols:
(3) Error injected, Type-II protocols:
P"
err
(a)
(b)
(c)

Figure 2: Latency incurred by (a) *YASIR* Transmitter, (b) the communication link itself, and (c) *YASIR* Receiver. Shaded boxes indicates values computed by the *YASIR* components.

Operation Timeliness T leverages the "stream"-nature of AES-CTR, which, upon receiving one byte in the plaintext, can compute the corresponding byte in the ciphertext. Consequently, T processes each of the bytes in the incoming frame $\mathtt{F}$ as they come in, and immediately outputs a byte in the corresponding transformed frame $\tilde{\mathtt{F}}$. The processing of each byte involves only a byte-wise XOR operation in the critical path, which incurs negligible latency.

When T has received $\mathtt{F}$ in its entirely, it immediately computes the HMAC on the internal counter and the ciphertext and starts outputting the result as well. We adopt an iterative computation of HMAC so that both the latency and storage requirement of this HMAC computation is a small constant independent of the length of the ciphertext, and thus that of $\mathtt{F}$.

Consequently, the transformation done by T incurs no delay, except the time needed to decode a code symbol or detect frame boundaries in the input frame, which takes at most 4 byte times in almost all protocols, as discussed in Section 3.

Figure 2 gives a pictorial illustration of this.

6.2 *YASIR* Receiver for Type-I Protocols

On input a transformed frame $\tilde{\mathtt{F}}' = \mathtt{s}||\mathtt{CTXT}'||\mathtt{x}||\mathtt{MAC}'||\mathtt{SEQ}'||\mathtt{e}$, denote

$$\mathtt{H}'||\mathtt{P}' = \textsc{Encrypt}_{\mathsf{ek}}(\mathsf{ctr}_R, \mathtt{CTXT}'),\ \mathtt{MAC}'' = \textsc{Hmac}_{\mathsf{hk}}(\mathsf{ctr}_R||\mathtt{CTXT}'),$$

and $l = |\mathtt{P}'|$. The *YASIR* Receiver R does the following:

- If $\mathtt{MAC}' = \mathtt{MAC}''$ then output the frame $\mathtt{F}' = \mathtt{s}||\mathtt{H}'||\mathtt{P}'||\mathtt{e}$. and increment ctr_R by 1.

- Otherwise, output the frame $\texttt{F}'' = \texttt{s}||\texttt{H}'||\texttt{P}''||\texttt{e}$, where $\texttt{P}'' = \texttt{P}'[1\ldots(l-1)]\,||\texttt{err}$ and $\texttt{err}$ is any single octet such that $\textsc{Crc}(\texttt{F}'')$ is invalid. Furthermore, if $\texttt{SEQ}' > \texttt{ctr}_R$ and $\texttt{MAC}' = \textsc{Hmac}_{\textsf{hk}}(\texttt{SEQ}'||\texttt{CTXT}')$, set $\texttt{ctr}_R = \texttt{SEQ}' + 1$.

In other words, R reconstructs $\texttt{F}'$ from $\tilde{\texttt{F}}'$ simply by decrypting $\texttt{CTXT}'$ if $\texttt{F}'$ contains a valid HMAC. Otherwise, R replaces the last byte of $\texttt{F}'$ with a byte $\texttt{err}$ during its reconstruction in such as way that the error-injected frame $\texttt{F}''$ will fail the conformance check in D. R calculates $\texttt{err}$ by first computing the correct CRC for $\texttt{F}''$ and then choosing $\texttt{err}$ to be any byte different from the last byte of the correct CRC.

Sequence Numbers Contrary to many other protocols in which sequence numbers are contained in frame headers, T in *YASIR* puts the sequence number at the end of a frame to reduce the amount of data R must receive before it can reconstruct a frame and decides on the authenticity and freshness of the frame. Since *YASIR* uses a 4-octet sequence number, the latency at R is reduced by 4 byte times.

Note that R does not know the actual sequence number of a frame by the time it has finished relayed the frame to D. To properly decrypt and verify the integrity the incoming transformed frame, R predicts the sequence number of the frame using its internal counter value. The prediction will be correct if there was no random or malicious corruption in one or more frames recently sent. The sequence number at the back of the frame is used for re-synchronizing the internal counters between T and R in case they have gone out of synchronization, but only when the integrity of the frame can be verified using that sequence number, to prevent malicious manipulation of the value of R's counter.

Operation Timeliness Similar to T, R is designed to minimize the latency it incurs by attempting to start outputting bytes of the detransformed frame once they become available. The use of AES-CTR once again allows R to reconstruct the original frame at a per-byte basis by decrypting the input bytes as they arrive.

The output of R depends on the validity of the HMAC inside the transformed frame it receives. R behaves indifferently until when it has finished outputting the second to last byte in the payload and has to decide whether it should inject an error or not, depending on the validity of the HMAC. This implies that R must have received the entire 12-byte-long HMAC in the input at that moment. To ensure this, R must delay its operation by at least 12 byte times.

As argued in Section 6.1, decrypting a byte and verifying a HMAC both take negligible time. Also, the CRC checksum for F'' and thus the value of $\texttt{err}$ can be computed in negligible time and even pre-computed. Therefore, if we assume that the symbol $\texttt{x}$ can be decoded in 2 byte times, the total latency incurred by R is thus $12 + 2 = 14$ byte times. Finally, while R may operate on the sequence number in the input, the operation does not incur additional latency as the detransformation does not depend on it.

Figure 2 illustrates this.

6.3 *YASIR* Receiver for Type-II protocols

On input a transformed frame $\tilde{\texttt{F}}' = \texttt{s}||\texttt{CTXT}'||\texttt{x}||\texttt{MAC}'||\texttt{SEQ}'||\texttt{e}$, denote

$$\texttt{H}'||\texttt{P}' = \textsc{Encrypt}_{\textsf{ek}}(\texttt{ctr}_R, \texttt{CTXT}'),\ \texttt{MAC}'' = \textsc{Hmac}_{\textsf{hk}}(\texttt{ctr}_R||\texttt{CTXT}'),$$

and $l = |\mathtt{P}'|$. Let also l_h be the length of $\mathtt{P}'$ as indicated in $\mathtt{H}'$. The *YASIR* Receiver R does the following:

1. If (i) $l < l_h$, or (ii) $l = l_h$ and $\mathtt{MAC}' = \mathtt{MAC}''$, then output the frame $\mathtt{F}' = \mathtt{s}||\mathtt{H}'||\mathtt{P}'||\mathtt{e}$. Otherwise, output the frame $\mathtt{F}'' = \mathtt{s}||\mathtt{H}'||\mathtt{P}''||\mathtt{e}$, where $\mathtt{P}'' = \mathtt{P}'[1 \ldots (l_h - 1)]$.

2. If $\mathtt{MAC}' = \mathtt{MAC}''$, then increment ctr_R by 1. Otherwise, if $\mathtt{MAC}' = \textsc{Hmac}_{\mathsf{hk}}(\mathtt{SEQ}'||\mathtt{CTXT}')$ and $\mathtt{SEQ}' > \mathsf{ctr}_R$, set $\mathsf{ctr}_R = \mathtt{SEQ}' + 1$.

R for Type-II protocols mostly operates in the same way as that for Type-I protocols. Here we only highlight the differences between the two.

The output behavior of R for Type-II protocols depends not only on the validity of the HMAC, but also the frame's actual length, l, compared to the length as indicated in the header, l_h. There are three cases to consider. (1) If $l < l_h$, then there is probably some random or malicious error in the frame. However, the frame is malformed in itself already and D will drop it when it checks the length. Therefore, R does not inject any error in this case. (2) If $l = l_h$ and the HMAC verifies, R decides that the frame is authentic and fresh and hence does not inject any error. (3) If $l > l_h$, or $l = l_h$ but the HMAC verification fails, R must inject an error. To do so, R outputs a frame with payload one byte shorter than what is indicated in the header.

Operation Timeliness R for Type-I protocols and R for Type-II protocols both require a 12-byte HMAC to decide on whether an error should be injected; they also need to make that decision at the same time: after the receipt of the second to last byte of the payload. Therefore, even though they differ in how an error is injected, a delay of 14 byte times is sufficient for both of them. Again, we have an illustration of this in Figure 2.

7 Evaluation

7.1 Security

We state the theorem regarding the security of our *YASIR* construction in the following. We have given a discussion of what security means for a BITW solution in Section 4. We have yet to define it formally, which we will do in Appendix B. We therefore defer the proof of the theorem to Appendix C.

Theorem 1 (*YASIR*'s Security) *Our YASIR construction has data privacy, and data authenticity and freshness under our security model defined in Appendix B if the cryptographic primitives underlying YASIR, namely AES-CTR, HMAC-SHA-1, are secure.*

7.2 Hardware Costs

A hardware implementation of the *YASIR* BITW module, $\mathcal{T}$ or $\mathcal{R}$, possesses an AES core, a SHA-1 core (which enables HMAC-SHA-1) and some registers for storing various internal states. Also, a circuit for calculating the CRC checksum must be present in $\mathcal{R}$ for Type-I protocols, while a circuit must exist to extract the frame length from the frame header in $\mathcal{R}$ for Type-II protocols. We stress that the hardware size (in terms of gate count) is independent of the length of the frames being secured.

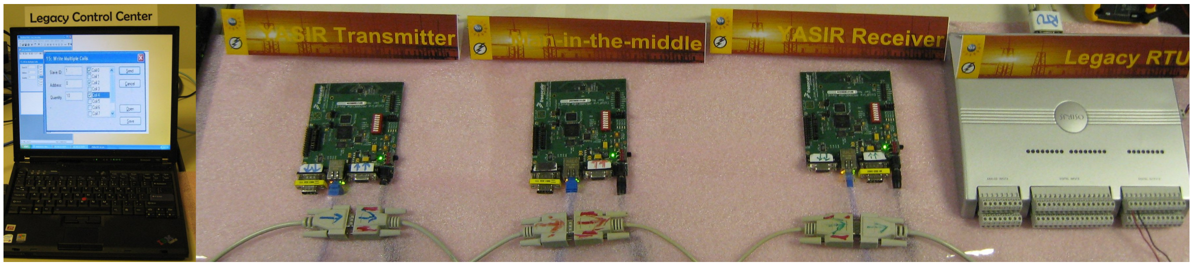

Figure 3: Our experimental testbed. The PC on the left represents a source SCADA device located in a control center. The RTU on the right acts as a destination SCADA device found in a remote substation. The left (resp. right) micro-controller board is our *YASIR* Transmitter (resp. Receiver). The middle board plays the role of the adversary who has tapped into the communication link.

In practice, a *YASIR* BITW module functions both as a Transmitter and a Receiver, but it never needs to play both role at the same time. Therefore, the module can reuse most of the hardware for both functionalities.

7.3 End-to-end Latency

As have analyzed in Section 6, our *YASIR* Transmitter and Receiver incurs 4 and 14 byte times respectively. Therefore a total of 18 byte times is incurred by *YASIR*. This confirms with Table 1.

One could reduce the end-to-end latency from 18 to only 10 byte times by replacing HMAC-SHA-1-96 with HMAC-SHA-1-32. The relatively low level of security provided by HMAC-SHA-1-32 may still be enough for SCADA communications, as the communication links have bandwidth low enough to preclude too many attack attempts [NIS02, Appendix B].

To achieve low latency, we move the AES block-cipher operation off the critical path. This requires the throughput of the AES core to be at least the data-rate of the SCADA links. This is not an issue at all: AES cores with throughput high enough even for Gigabit Ethernet are commercially widely available and also exist as academic FPGA/ASIC prototypes. Similarly, the latency due to the HMAC operation using an FPGA is only a negligible fraction of 1 byte time, due to low baud rates of the SCADA protocols.

A comprehensive survey of AES and SHA hardware performance can be found in [GLOK05].

7.4 Prototyping and Experimentation

As a proof-of-concept prototype, we have implemented *YASIR* on two Freescale MCF5235EVB micro-controller boards, one running as the *YASIR* Transmitter and the other as the Receiver. In our experimental testbed, each board ran the μClinux operating system, and communicated via its two RS-232 serial ports configured at the baud rate of 9600. For the destination SCADA device, we used OSIRIS, an RTU manufactured by the company OSI. For the source SCADA device, we used a PC running a software that emulates the *Human Machine Interface (HMI)* at control centers. We connected everything using RS-232 serial cables as in Figure 1, except that we injected an extra micro-controller board in the middle of the communication link, which allowed us to mimic the presence of an adversary who sniffs and tampers with traffic. Figure 3 shows our experimental testbed.

Experimenting with the testbed setup, we were able to demonstrate several properties of *YASIR*. First, bumping the *YASIR* Transmitter and Receiver pair into the wire does not affect the normal operations. For example, the PC was still able to set the digital switches in the RTU and also poll analog data from the RTU. Second, after adding the *YASIR* modules, the traffic sniffed by the adversary was no longer in clear. Finally, the RTU stopped responding to commands crafted and injected by the adversary once *YASIR* was in place.

8 Conclusions

In this paper, we have proposed *YASIR*, which is a BITW solution for retrofitting security to serial-based SCADA systems where communications are time-critical, such as those for electric power generation and distribution. As Table 1 has shown, our solution is the first to provide data integrity in a timely manner, at a high security level even against strong and yet realistic adversaries. Hence, *YASIR* is a pragmatic solution to a high-threat security problem we are facing right now.

We have implemented our solution as a proof-of-concept prototype. As our next step towards a real industrial deployment of *YASIR*, we are going to implement it on FPGA for better cost-effectiveness. Furthermore, we have been in contact with Working Group C6 in the Substation Committees of IEEE. The group is drafting a standard for a cryptographic protocol for cyber security of substation serial links [IEE07]. We are working on the potential incorporation of *YASIR* into that standard.

References

[Ame06] American Gas Association. Cryptographic Protection of SCADA Communications Part 1: Background, Policies and Test Plan. Technical Report AGA Report No. 12, American Gas Association, March 2006. http://www.gtiservices.org/security/AGA%2012%20Part%201%20Final%20Version.pdf.

[BDJR97a] Mihir Bellare, Anand Desai, E. Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *FOCS*, pages 394–403, 1997.

[BDJR97b] Mihir Bellare, Anand Desai, E. Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *FOCS*, pages 394–403, 1997. Full version: http://www-cse.ucsd.edu/users/mihir/papers/sym-enc.html.

[BN00] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *ASIACRYPT*, volume 1976 of *LNCS*, pages 531–545. Springer, 2000.

[BT04] Alexandra Boldyreva and Nut Taesombut. Online encryption schemes: New security notions and constructions. In *CT-RSA*, volume 2964 of *LNCS*, pages 1–14. Springer, 2004. Full version: http://www.cse.ucsd.edu/users/aboldyre/papers/bt.html.

[Dwo01] Morris Dworkin. Recommendation for block cipher modes of operations–methods and techniques. Technical report, National Institute of Standards and Technology (NIST), Dec 2001.

[GLOK05] Benjamin Gittins, Howard Landman, Sean O'Neil, and Ron Kelson. A presentation on vest hardware performance, chip area measurements, power consumption estimates and benchmarking in relation to aes, sha-256 and sha-512. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/078, 2005. http://www.ecrypt.eu.org/stream.

[Had07] Mark Hadley. Personal communication, Aug 2007.

[IEE05] IEEE standard communication delivery time performance requirements for electric power substation automation. IEEE Std 1646-2004, 2005.

[IEE07] IEEE Substation Committees, Working Group C6. IEEE Trial Use Standard for a Cryptographic Protocol for Cyber Security of Substation Serial Links. IEEE P1711 Draft, Feb 2007.

[KA98] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol. Internet Engineering Task Force: RFC 2401, November 1998. http://www.ietf.org/rfc/rfc2401.txt.

[MPPW06] Munir Majdalawieh, Francesco Parisi-Presicce, and Duminda Wijesekera. DNPSec: Distributed Network Protocol Version 3 (DNP3) Security Framework. In *Advances in Computer, Information, and Systems Sciences, and Engineering: Proceedings of IETA 2005, TeNe 2005, EIAE 2005*, pages 227–234. Springer, 2006.

[NIS01a] NIST. FIPS 180-2: Secure hash standard (SHS). Technical report, National Institute of Standards and Technology (NIST), 2001.

[NIS01b] NIST. FIPS 197: Announcing the advanced encryption standard (AES). Technical report, National Institute of Standards and Technology (NIST), 2001.

[NIS02] NIST. FIPS 198: The keyed-hash message authentication code (HMAC). Technical report, National Institute of Standards and Technology (NIST), 2002.

[Sch] Schweitzer Engineering Laboratories, Inc. SEL-3021-2 datasheet. `http://www.selinc.com/datasheets/3021-2_DS_20070109.pdf`.

[Smi01] Tony Smith. Hacker jailed for revenge sewage attacks. *The Register*, Oct 31 2001. `http://www.theregister.co.uk/2001/10/31/hacker_jailed_for_revenge_sewage/`.

[TS07] Patrick P. Tsang and Sean W. Smith. A low-latency, high-integrity security retrofit for legacy scada systems. Technical Report TR2007-603, Department of Computer Science, Dartmouth College, September 2007.

[TS08] Patrick P. Tsang and Sean W. Smith. YASIR: A Low-Latency, High-Integrity Security Retrofit for Legacy SCADA Systems (Extended Version). Technical Report TR2008-617, Dartmouth College, Computer Science, Hanover, NH, April 2008.

[WKM04] Andrew K. Wright, John A. Kinast, and Joe McCarty. Low-latency cryptographic protection for scada communications. In *ACNS*, volume 3089 of *LNCS*, pages 263–277. Springer, 2004.

A Cryptographic Tools

AES-CTR Advanced Encryption Standard (AES) [NIS01b] is a block cipher with block size of 16 octets. A block cipher can operate in different *modes*, which specify ways of turning an operation on block-length values into an operation on longer-length messages [Dwo01]. In the *counter mode (CTR)*, a counter (incremented for each new block) is combined with a nonce and then encrypted under the block operation; the output is then XOR'ed against the corresponding block of plaintext. The security of CTR mode is proven in [BDJR97a]. We denote AES operating in counter mode as AES-CTR, and denote AES-CTR with 128-bit keys as AES-CTR-128. AES-CTR is NIST-standardized and patent-free.

SHA and HMAC SHA-1 (resp. SHA-256) [NIS01a] is a cryptographic hash that operate on messages of any length and produce 160-bit (resp. 256-bit) outputs. A secure cryptographic hash must possess properties such as preimage resistance and collision resistance. HMAC-SHA-1 and HMAC-SHA-256 [NIS02] are keyed hash message authentication codes (HMACs) built from SHA-1 and SHA-256 respectively. HMAC-SHA-1-96 (resp. HMAC-SHA-1-32) [NIS02] is the same as HMAC-SHA-1, except that the 160-bit output is truncated to the first 96 (resp. 32) bits. The above hash functions and HMACs have the same block size of 512 bits. SHA and HMAC are NIST-standardized and patent-free.

B Security Model

To formally model the adversary's capabilities as described in Section 4.1, we allow the adversary to make black-box queries to the following oracles:

- The *transformation oracle*, $\mathcal{O}_{\mathcal{ST}}$, takes a SCADA message $\mathtt{M}$ as input and returns the transformed frame $\tilde{\mathtt{F}} = T(S(\mathtt{M}))$. The presence of this oracle models the adversary's capability similar to launching chosen-plaintext attacks in authenticated encryption [BN00].
- The *detransformation oracle*, $\mathcal{O}_{\mathcal{RD}}$, takes a transformed frame $\tilde{\mathtt{F}}$ as input and returns $D(R(\tilde{\mathtt{F}}))$, i.e., either an *error* or the corresponding original message. The presence of this oracle models the adversary's capability similar to launching chosen-ciphertext attacks in authenticated encryption.

We say that a BITW solution is secure if it has data authenticity and freshness, and data privacy. These notions are defined in the following.

Data Authenticity and Freshness A BITW solution has data authenticity and freshness if no probabilistic poly-time (PPT) adversary can win, with non-negligible probability, in the game defined as follows:

- *(Setup Phase.)* The challenger generates the necessary keys and initializes T and R.
- *(Learning Phase.)* The adversary can arbitrarily and adaptively query $\mathcal{O}_{\mathcal{ST}}$ and $\mathcal{O}_{\mathcal{RD}}$.
- *(Challenge Phase.)* The adversary returns a transformed frame $\tilde{\mathtt{F}}^*$. The challenger computes $\mathcal{O}_{\mathcal{RD}}(\tilde{\mathtt{F}}^*)$. If the output is an *error*, the adversary loses immediately. Otherwise, let the output be $\mathtt{M}^*$.

The adversary wins in the game if:

- *(Case 1.)* $\nexists \mathcal{O}_{\mathcal{ST}}(\mathtt{M}^*)$, or,
- *(Case 2.)* $\tilde{\mathtt{F}}^* \leftarrow \mathcal{O}_{\mathcal{ST}}(\cdot)$, $\exists \tilde{\mathtt{F}}' \leftarrow \mathcal{O}_{\mathcal{ST}}(\cdot)$, $\exists \mathcal{O}_{\mathcal{RD}}(\tilde{\mathtt{F}}')$, and $\tilde{\mathtt{F}}^* \leftarrow \mathcal{O}_{\mathcal{ST}}(\cdot)$ happened before $\tilde{\mathtt{F}}' \leftarrow \mathcal{O}_{\mathcal{ST}}(\cdot)$.

Data Privacy A BITW solution has data privacy if no probabilistic poly-time (PPT) adversary can win, with probability non-negligibly greater than 1/2, in the game defined as follows:

- *(Setup Phase.)* The challenger generates the necessary keys and initializes T and R.
- *(Learning Phase 1.)* The adversary can arbitrarily and adaptively $\mathcal{O}_{\mathcal{ST}}$ and $\mathcal{O}_{\mathcal{RD}}$.
- *(Challenge Phase.)* The adversary queries the challenge oracle $\mathcal{O}_{\mathcal{C}}$ with two messages $\mathtt{M}_0$ and $\mathtt{M}_1$ such that their respective frames $\mathtt{F}_0$ and $\mathtt{F}_1$ of the same length. $\mathcal{O}_{\mathcal{C}}$ behaves as $\mathcal{O}_{\mathcal{ST}}(\mathtt{M}_b)$, where $b = 0$ or 1 with equal probability, and returns the oracle output $\tilde{\mathtt{F}}^*$.
- *(Learning Phase 2.)* The adversary can arbitrarily and adaptively query $\mathcal{O}_{\mathcal{ST}}$ and $\mathcal{O}_{\mathcal{RD}}$, except $\mathcal{O}_{\mathcal{RD}}(\tilde{\mathtt{F}}^*)$.
- *(End Game Phase.)* The adversary outputs $b' \in \{0, 1\}$.

The adversary wins in the game if $b' = b$.

C Proof Sketch for Theorem 1

Data Authenticity and Freshness We assume the contrary that there exists a PPT adversary who can win the corresponding game with non-negligible probability, and arrive at a contradiction.

The challenger generates the encryption key for the T and R, but does not know the HMAC key. The challenger answers the adversary's queries to the oracles according to specification throughout the game. To do so, he queries the HMAC oracle to produce valid HMACs in the transformed frames. With non-negligible probability, the adversary will return $\tilde{\mathtt{F}}^*$ and win the game, after some polynomial time and having made a polynomial number of queries.

Now, if is it the case that $\nexists \mathcal{O}_{\mathcal{ST}}(\mathtt{M}^*)$, i.e. it is *case 1* in the game, then $\tilde{\mathtt{F}}^*$ contains a valid HMAC on some input such that the challenger has never queried the HMAC oracle. This violates the security guarantee of HMAC. Otherwise, i.e. it is *case 2* in the game, then the sequence number s^* in $\tilde{\mathtt{F}}^*$ must be no less than the internal counter c^* in R during the challenge phase, i.e. $s^* \geq c^*$. Moreover, s^* is strictly less than the sequence number s' in $\tilde{\mathtt{F}}'$ as $\tilde{\mathtt{F}}^*$ was output earlier than $\tilde{\mathtt{F}}'$ was, i.e. $s^* < s'$. Finally, the existence of $\mathcal{O}_{\mathcal{RD}}(\tilde{\mathtt{F}}')$ implies that the internal counter of R was set to s' before the challenge phase and because the counter value can only increases, we have $c^* \geq s'$. We have a contradiction: $s^* \geq c^* \geq s' > s^*$.

Data Privacy We assume the contrary that there exists a PPT adversary who can win the corresponding game with probability non-negligibly greater than 1/2, and arrive at a contradiction.

The challenger generates the HMAC key for the T and R, but does not know the encryption key. The challenger answers the adversary's queries to the oracles according to specification throughout the game. To be able to properly produce the ciphertext in the transformed frames when answering $\mathcal{O}_{\mathcal{ST}}$ queries, he queries the AES-CTR oracle using T's internal counter as the nonce. Also, the challenger is able to, with overwhelming probability, answer $\mathcal{O}_{\mathcal{RD}}$ queries thanks to the security of HMAC: the challenger needs only decrypt ciphertexts the plaintext of which he already knows about, as the only way to produce a transformed frame that will be be not detransformed to an *error* is through $\mathcal{O}_{\mathcal{ST}}$.

The challenger embeds the AES-CTR problem instance when answering the adversary's query to $\mathcal{O}_{\mathcal{C}}$ and relays the adversary's answer to whether $b = b'$ as his answer to the AES-CTR challenge. It is trivial to see that the challenger's answer is correct if and only if the adversary's answer is correct. Therefore, given the adversary, the challenger can break the security of AES-CTR, contradicting to the assumption that AES-CTR is secure.

Remarks It has been shown in [BT04] that it is impossible for on-line encryption to be secure against chosen-ciphertext attacks in the conventional sense defined for symmetric encryption. As the *YASIR* modules also operate input "on-the-fly" like on-line encryption, one might wonder why *YASIR* could possibly be secure against chosen-ciphertext attacks, i.e. IND-CCA-secure [BDJR97b]. The reason can be summarized as follows. The on-line nature of the *YASIR* modules is not exposed to the adversary: to probe T, the adversary must provide a message in its entirety to S; he is unable to get a partial transformed frame from a partial message and then adaptively devise the rest of the message. Similarly, even though R outputs bytes to D before checking their authenticity, the adversary can only see them at the output of D when those bytes are eventually checked to be authentic. In other words, S with T attached (resp. D with R attached) exposes an interface to the adversary similar to the encryption (resp. decryption) device in conventional (authenticated) symmetric encryption. Therefore, the impossibility result in [BT04] is not applicable in *YASIR*'s case.