A Deployable SCADA Authentication Technique for Modern Power Grids

Curtis R. Taylor [1], Craig A. Shue [2], Nathanael R. Paul [3]

[1,2] *Computer Science Department, Worcester Polytechnic Institute*
100 Institute Rd., Worcester, MA 01609, USA
[1] crtaylor@cs.wpi.edu
[2] cshue@cs.wpi.edu

[3] *University of South Florida*
4202 E. Fowler Avenue, Tampa, FL 33620, USA
[3] pauln@cse.usf.edu

***Abstract*—The modern power grid makes extensive use of automated data collection and control. These supervisory control and data acquisition (SCADA) systems often use communication protocols that were developed for isolated networks. However, the underlying SCADA systems often use the Internet for data transit, exposing these SCADA devices to remote, malicious adversaries. Unfortunately, these protocols are often vulnerable to impersonation attacks, and the devices can be susceptible to cryptographic key compromise. This allows adversaries to pollute the protocols with misinformation. In this paper, we propose an approach to authenticate the underlying SCADA protocols that combines a different approach to data authenticity and hardware-protected key distribution approach. Unlike prior work, our approach does not require modification to the SCADA end-points themselves, allowing the technique to be combined with legacy devices.**

***Keywords*-Security, SCADA, Authentication**

I. Introduction

The modern power grid requires automation. Automated systems can rapidly respond to changing events, yielding increased reliability and efficiency. The adoption of the smart grid, will only accelerate this need for automation [1]. The smart grid will have dynamic loads due to renewable power sources, which have volatile generation patterns, and the increased usage of electric vehicles, which can result in volatile load on the system.

Supervisory control and data acquisition (SCADA) systems provide the automation in today's power systems. These systems control infrastructures such as the electric grid, water supplies, and pipelines, and provide monitoring throughout the grid at substations through a series of measurement units. The importance of SCADA makes it an attractive target for attackers. By disrupting the proper operation of SCADA systems, attackers could cause events such as over-dosing or under-dosing of chemicals in water supplies [2] or altogether stopping production of electricity [3].

SCADA systems now make extensive use of the Internet to transit communication between operators and measurement and actuation devices. This exposes SCADA systems to remote attackers seeking to provide misinformation to operators. Unfortunately, SCADA systems are unable to apply many of the best practices associated with modern computer systems. In particular, SCADA devices are often installed with an expected operating lifetime measured at the decade granularity [4]. These systems are expected to operate without hardware or software upgrades. Accordingly, the modern patch and upgrade approaches used in commodity computers are often not viable for SCADA systems [5].

The balance between practical SCADA operations and security is no more apparent than in the DNP3 (Distributed Network Protocol 3.0) protocol [6]. DNP3 does not provide any authentication between the communicating parties, allowing adversaries to trivially alter messages between the parties or impersonate one of the parties. This authenticity problem has been recognized for years, with updates to DNP3 in both DNP3 Secure Authentication Version 5 (SAv5) [7] and DNP3Sec [8]. However, neither of these protocols have gained much traction for the following two reasons: 1) many vendors do not have support for these protocols yet and 2) they would require software or hardware modification in existing systems to be deployed. These limitations threaten to slow adoption.

In this paper, we propose a proxy approach for SCADA communication. This approach has several important properties that are required for practical application: 1) it does not introduce discernible latency, 2) it does not require modifications to the SCADA devices themselves, 3) it is protocol-agnostic, making it compatible with multiple SCADA protocols, and 4) it can be partially deployed between communicating pairs, providing incremental security improvements as it is rolled out. In designing this approach, we use keyed cryptographic operations that ensure a message is authentic. We also present a novel approach for sharing these keys while providing both forward and backward key secrecy.

To evaluate our proposed approach, we used the DNP3 protocol as an example SCADA case. We evaluate the latency and performance characteristics associated with our protocol and the computational loads that would be required. We find that we could provide authenticity and integrity to SCADA traffic without expensive computer hardware and without modifications or performance degradation at SCADA devices.

II. Related Work

The security of SCADA systems has been previously explored by researchers. Hadbah *et al.* [9] explored security vulnerabilities introduced as a result of common-place protocols used in SCADA for the power grid. Kang *et al.* [10] also considered cyber security threats SCADA systems faced as a result of becoming a part of the Internet rather than traditional closed-loop substations. The community has also examined the security issues associated with the Smart Grid. Work by Khurana *et al.* [11] broadly introduces security and privacy concerns about the Smart Grid including existing authentication and encryption solutions for SCADA networks while presenting concerns on resource constrained devices. Simmhan *et al.* [12] reiterated similar concerns on security and privacy of the Smart Grid but considered issues from a cloud setting due to the cloud's elastic resource capabilities and shared resources that SCADA systems could leverage.

While power security is a broad area, we focus on SCADA security and the DNP3 protocol in particular. DNP3 [6] is a popular SCADA protocol with significant adoption in power systems. Accordingly, securing this protocol is essential for power grid security. Two prior approaches have attempted to address the security vulnerabilities associated with DNP3: 1) DNP3 Secure Authentication Version 5 (SAv5) [7] and 2) DNP3Sec [8]. DNP3 SAv5 is the more popular security approach, with the ability to provide authentication, integrity, and replay protection but does not provide confidentiality. The approach in DNP3Sec makes significant revisions to the original DNP3 specification, adding confidentiality, integrity and authentication, but requires major modifications to the protocol making adoption more challenging. While both approaches are viable from a security standpoint, they require modifications to the DNP3 devices, slowing adoption.

Although our research focuses on DNP3, our approach can be applied to various other SCADA protocols. In this way, our work resembles "bump-in-the-wire" (BITW) approaches to securing SCADA protocols such as [13] and [4]. These approaches attempt to retrofit security by inserting hardware devices in between the source and destination. In particular [4] focused on allowing data authenticity, freshness, low cost, and low latency, with the option of adding encryption for confidentiality. Their approach involves an overhead upwards of 18 bytes per frame. To achieve lower latency than other BITW solutions, the authors take advantage of AES stream ciphering to avoid having to buffer data before computing the cipher text to send to the destination. While our work is similar in the overall goal of authenticity, freshness to avoid replay attacks, and low latency, we believe our research differs in several ways.

The distinguishing features of our approach are that we are more flexible than BITW solutions because we can support partial deployment and unmodified legacy software. Our approach is lower cost because we do not require additional hardware. Using "after-the-fact authentication", we introduce no added latency to the existing infrastructure, and depending on the rate at which the authentication is occurring, we have the potential to have lower bandwidth usage than existing solutions. Unlike BITW approaches, we do not provide encryption for confidentiality. However, we note that existing security solutions have considered encryption to be an optional feature.

III. Background: The DNP3 Protocol

The DNP3 protocol allows a controlling device, often simply called the "master" system, to query SCADA devices at remote locations. These queried devices, called simply "outstations" or "Remote Terminal Units," provide their readings to the master on demand. Accordingly, a simple data acquisition workflow is initiated when the master system sends a "read" request to an outstation with a particular "class" to indicate the type of data requested. The outstation then accesses its own local database, acquires the relevant data, and sends it to the master in a response packet. This simple query design has led to a highly reliable communication protocol.

To minimize costs, grid operators often use commodity communication providers to provide connectivity between the master and outstation devices, often over a standard Internet connection. However, by using the Internet for transport, the DNP3 devices are now potentially exposed to attacks from the broader Internet [8]. These attacks can try to corrupt information in communication, such as altering requests or responses, or try to forge messages that impersonate one communicating party when speaking to another. These impersonations can be designed to cause malfunctions on the devices or simply preoccupy the devices to deny services to legitimate users (such as the authorized master system). Simply put, a DNP3 device will process any query it receives without trying to validate the request's authenticity.

The major security goals for DNP3 are authenticity, integrity, and availability. However, confidentiality is not itself a security concern: the data transported over the protocol does not need to be kept secret to ensure the protocol's mission objectives. We will only focus on authenticity, integrity, and availability in this work.

While these security goals are important, SCADA communication can often be sensitive to latency [14] [15]. Any approach to meet these security goals must not dramatically increase latency in the communication channel.

IV. Out-of-Band Authenticity for DNP3

Rather than modify the existing DNP3 protocol or the devices using DNP3, we propose an out-of-band technique to authenticate the messages transmitted by the DNP3 devices. This authentication protocol runs in parallel to the existing protocol, providing flexibility in deployment. Simply put, the devices running DNP3 can be modified to run our protocol, if desired, or a proxy device may be installed between the DNP3 device and its upstream network connection. This proxy does not delay packet processing by using in-line cryptographic operations. Instead, it allows packets to pass without alteration, much like a network router. However, it then provides a separate message afterward attesting to the authenticity of the packet. This approach allows the parties to verify a packet was valid, after the fact, while avoiding interference with latency-sensitive operations.

In designing our approach, we remain protocol agnostic. While our discussion centers around DNP3 due to its ubiquity, the approach is applicable to any bi-directional protocol. Even in power systems that use other protocols, we can add authenticity and integrity feedback to grid operators.

We describe two different modes of operation for the approach. In the first, we provide one-way protection, allowing the master to authenticate the responses it receives from the outstation. In our second mode, we add the ability for the master to authenticate not only the response from the outstation but also the request the outstation is responding to. This second mode incidentally provides cryptographic assurance that the outstation is responding to the specific request the master made (and not simply a legitimate answer from the outstation responding to a query from a machine impersonating the master). Our one-way protection approach is simpler and we introduce it first and later discuss the bi-directional protection scheme.

A. Mode 1: Unidirectional Authenticity

We begin by describing an approach allowing the master to authenticate responses from the outstation. In doing so, our approach provides data integrity as an ancillary benefit: if a bit has been corrupted, due to malice or network errors, the message authentication will fail. We depict this process in Figure 1.

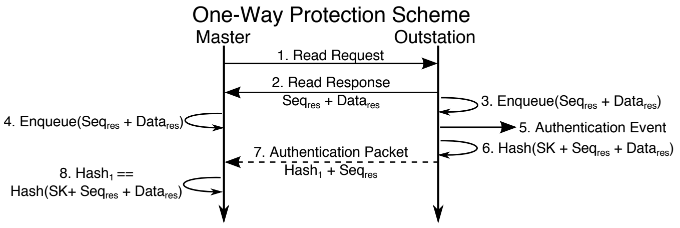

One-Way Protection Scheme
Master
Outstation
1. Read Request
2. Read Response
Seq_{res} + Data_{res}
3. Enqueue(Seq_{res} + Data_{res})
4. Enqueue(Seq_{res} + Data_{res})
5. Authentication Event
6. Hash(SK + Seq_{res} + Data_{res})
7. Authentication Packet
Hash_1 + Seq_{res}
8. Hash_1 ==
Hash(SK+ Seq_{res} + Data_{res})

Fig. 1. Protocol messages for DNP3 and unidirectional authenticity approach.

The approach does not modify the original DNP3 protocol or the devices communicating over DNP3. Accordingly, the DNP3 operation continues as it does today. However, an application at the outstation (or on a proxy device in front of the outstation) monitors all the communication that is sent by the outstation to the master. Each time a DNP3 response is seen, the monitoring application makes a copy of the packet. After a designated number of packets have been copied, or after a designated amount of time has elapsed, the monitoring application will send an "authentication packet" to provide cryptographic proof of authenticity and integrity to the master.

To create the authentication packet, the outstation obtains each packet in its database and places them in a buffer in order. The outstation then prepends this buffer with a secret key known only to the master and the outstation. The outstation then performs a hash-based message authentication code (HMAC) of the buffer. The outstation takes the HMAC output, plus the TCP sequence number from the packets[1] and places them in an authentication packet. It then transmits this packet to the master. Full details of the implementation of this approach are described in Section V.

The master system behaves similarly: it also monitors incoming packets (responses from the outstation) to create a database of responses. When the master receives an authentication packet, it uses the sequence numbers in the authentication packet to order them into a buffer. It then prepends the buffer with the secret key and performs an equivalent HMAC operation. If the HMAC the master constructs matches the HMAC in the authentication packet, the master knows the packets it received were authentic and were not modified or forged by an adversary.

B. Mode 2: Bidirectional Authenticity

The bidirectional protection is identical to the unidirectional approach with one exception: it allows the master to verify the request the outstation is answering. Without this modification, the master has no way of knowing if the response received, even if authenticated, is the answer to the question the master asked. For example, the master could perform a "Class 0 Read" operation, but an adversary could modify it to a "Class 1 Read" before it reaches the outstation. The outstation would then dutifully provide the results of a Class 1 Read and they would be successfully authenticated. However, they would not be the results the master requested and this fact may not be evident to the master.

To address this limitation, the outstation monitor records both incoming and outgoing DNP3 packets in separate databases. When the outstation is required to craft an authentication packet to send to a given master, it again creates a buffer of packets. However, it also includes the incoming packets received from the indicated master system in the HMAC. It then sends the packet to the master for consideration.

The master station, which also maintains a database of outgoing queries and incoming responses, again consults the authentication packet to create the input buffer and performs an HMAC operation similar to the outstation. If the HMACs match, the master knows the responses are authentic, but also knows the results were in response to the query the master issued.

V. Implementing Out-of-Band Authenticity

To evaluate our proposed approach, we created a prototype implementation of the protocol that allows participating parties to achieve three different security goals: 1) authentication, 2) integrity and 3) replay attack protection. Our approach does

[1] The sequence numbers are provided as a convenience to the master. If the master and outstation use a transport layer other than TCP, the sequence numbers are omitted. Instead, the master must try several permutations of the packets it receives and verify if any match the HMAC provided by the outstation. For non-TCP communication, the number of packets, n, that are included in a hash should be bounded to a small number, since otherwise the computation could become intractable (since the number of possible permutations is $n!$).

not require changes to existing systems. To achieve these security goals, we use pre-shared keys for HMAC computations and the use of TCP sequence numbers and a nonce to prevent replay attacks.

Our implementation was written in the C programming language using libpcap [16], OpenSSL [17], and the OpenDNP3 (Automatak fork) library [18]. While we implemented both schemes, we only mention the bidirectional authentication scheme for brevity, since the unidirectional case is a subset of the functionality.

In using OpenDNP3, we implemented a master and outstation. At the time of writing, the OpenDNP3 implementation does not support the SA authentication extensions so we did not test these cases. We set a probe interval of one probe per second on the master. Each probe used a Class 0 Read containing a single integer value. In this scenario, the master station has no authentication or integrity guarantees.

We then use separate executables on the master and outstation systems to implement our authentication protocol. We use the `libpcap` library and set a filter to exclude any traffic other than DNP3 communication. Each time the executable detects a DNP3 packet, it examines the packet headers to determine the TCP source and destination ports. It uses these ports to distinguish incoming packets from outgoing packets[2]. The executable then places each packet, including the full DNP3 payload and the TCP sequence numbers, into a database. We keep a separate list for requests and responses.

Periodically, our executables must create an authentication packet. The authentication packets can be triggered at a set temporal interval or when a certain number of DNP3 packets have been received. If the database contains pending packets, an authentication packet is created. If a trigger is received, but the database is empty (such as when a timer elapses, but no packets have been received), the authentication packet is not sent.

A. Authentication Packet Generation

To create an authentication packets, we use a structure of the following format.

```
typedef struct {
    char hash[SHA256_LENGTH];
    unsigned int seqInNums[MAX_SEQ];
    unsigned int seqOutNums[MAX_SEQ];
    unsigned int nonce;
} authStruct;
```

Each item in the authentication packet is as follows:

- **hash:** a SHA256 hash including the secret key, sequence numbers, all packet data (incoming + outgoing), and the nonce
- **seqInNums:** list of all incoming sequence numbers included in the hash
- **seqOutNums:** list of all outgoing sequence numbers included in the hash
- **nonce:** a randomly generated unsigned integer

In Figure 2, we depict the bidirectional authenticity protocol. In this protocol, we communicate the TCP sequence numbers to allow the master to quickly reference the packets being authenticated when independently verifying the communication. Once the authentication packet stage is triggered, either by a timer or a set number of packets, the outstation constructs the HMAC and sends an authentication packet carrying this information in a UDP packet.

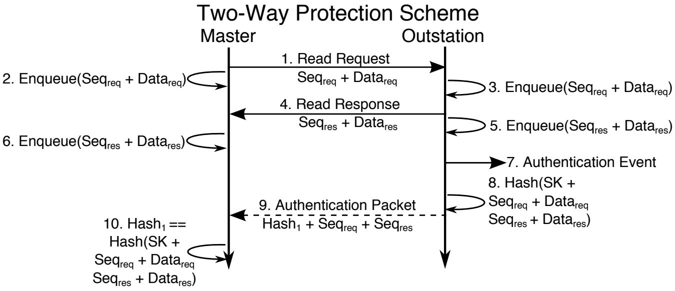

Two-Way Protection Scheme
Master
Outstation
1. Read Request
Seq_{req} + $Data_{req}$
2. Enqueue(Seq_{req} + $Data_{req}$)
3. Enqueue(Seq_{req} + $Data_{req}$)
4. Read Response
Seq_{res} + $Data_{res}$
5. Enqueue(Seq_{res} + $Data_{res}$)
6. Enqueue(Seq_{res} + $Data_{res}$)
7. Authentication Event
8. Hash(SK + Seq_{req} + $Data_{req}$ Seq_{res} + $Data_{res}$)
9. Authentication Packet
$Hash_1$ + Seq_{req} + Seq_{res}
10. $Hash_1$ == Hash(SK + Seq_{req} + $Data_{req}$ Seq_{res} + $Data_{res}$)

Fig. 2. Protocol messages for DNP3 and bidirectional authenticity approach.

Like the outstation, the master maintains a list of requests sent and responses received. Upon receiving an authentication packet, the master independently generates a HMAC of the relevant messages covered in the authentication packet. It then compares the HMAC with the value received in the authentication packet. Rather than trying to use all packets in the list, the master uses the sequence numbers in the authentication packet to determine which packets to use in authenticating. After receiving all necessary data, the hash is computed. In the case that both hashes are identical, the master can continue to operate with the assurance that the requests received by the outstation and the responses received from the outstation have not been modified or retransmitted using a replay attack. The replay protection is inherent in the fact that a nonce is included in the HMAC as well as TCP sequence numbers, which are incremental.

VI. Verifying Security Functionality

To test the approach's ability to accurately authenticate and detect packet modification, we inserted a malicious router into the communication path between the master and outstation. This router was configured to falsify DNP3 data in transit to determine if the master or outstation would detect malicious activity. During this testing, no SA techniques were used. The conditions that the master and outstation operated were identical to those discussed in section V; the master continually sent a Class 0 read, which contained a single integer value.

In the experiment, the master's request was allowed to be transmitted unmodified. However, when the router noticed an outstation's response, the value of the integer, *X*, being returned was altered, to *Y*, so that it no longer matched the original value. Such modifications were trivial at the router: the router altered the DNP3 packet contents and recalculated the CRC checksum value using an existing library [19]. Naturally,

[2] If non-standard DNP3 ports are used, the monitors would need to examine the DNP3 function codes to distinguish a request and response.

the master received the modified value and accepted it as valid since it lacks any other mechanism for determining the data authenticity.

After an authentication packet is sent by the outstation, the master learns that the results were suspicious. When the outstation constructed the HMAC in the authentication packet, it used the DNP3 data containing *X*, but when the master computes the same HMAC, it uses the data it received, which contained *Y*. Since the HMACs did not match, the master knows that the DNP3 results are untrustworthy and may be fabricated.

VII. Performance Overheads

To quantify the performance overheads of the approach, we used the same scenario as that in verifying our protocol's functionality. We created a master station and an outstation in separate virtual machines (VMs). We also created a router VM that connects the two machines. The machine hosting the VMs has a 3GHz quad core processor and 16GB of RAM, but both the master and outstation only each had 1 core and 512MB of RAM.

The master's communication in the outstation resulted in $1\frac{request}{second}$; naturally, the outstation gave $1\frac{response}{second}$. We then enabled two-way protection between the machines in order to find an upper bound on the required resources needed for our approach. While the authentication protocol ran, the outstation was configured to authenticate the master every 5 seconds. The experiment was then ran for a duration of 2 minutes. The summary of the experiment is shown in Table I.

TABLE I
Performance Result for Two-Way Protection

	Master	Outstation
Base Memory (MB)	4.0	1.9
Temporary Memory (MB)	0.47	0.47
Total Memory (MB)	4.47	2.37
CPU Time (seconds)	0.30	0.38
CPU Cycles	$9*10^8$	$1.14*10^9$
Network Traffic Sent (MB)	0.234	0.238

The base memory is the memory required to load the protocol application into the operating system's memory. The temporary memory represents the storage requirements for all the incoming and outgoing packets that will be authenticated. Because this variable is dependent on the data being transmitted, we estimated this to be $2,048$ bytes, matching the IEEE suggested maximum for DNP3 packets [6]. This approximation likely overestimates the network traffic transmitted and memory used in practical application.

We note that our authentication protocol implementation was not specially optimized and has many opportunities for performance improvements. However, our performance analysis shows the protocol is extremely lightweight and can be handled on commodity microcomputers or as an application on existing infrastructure, as needed.

VIII. Keying Infrastructure

Prior approaches to authenticate SCADA messages assume a reliable mechanism to share keys. However, these approaches often do not discuss how such a keying system could be implemented. In particular, the challenges of addressing a compromise of a master or outstation system are simply not discussed. Since we also require a use of a symmetric key in our approach, we propose a keying system that provides both forward and backward key secrecy. In our approach, if a system is compromised from time t_1 to t_2, the adversary learns nothing about keys in use from before $t_1 - \epsilon$ or after $t_2+\epsilon$, where ϵ represents a negligible amount of time (e.g., less than 10 seconds). Accordingly, if a compromise is detected and remediated, the same systems can be used without any additional changes to the keying infrastructure. This yields a system that is practical for deployers.

We now describe background in key infrastructure and then discuss how our system could be implemented.

A. Background on Public and Shared Key Approaches

In many network security protocols, key distribution is often tied to developing a Public Key Infrastructure (PKI) that will allow the parties to learn the keys of other participants. Unfortunately, a PKI requires greater computational load on devices, which may not be practical on embedded systems. Further, if a key is compromised, the re-keying process may require visiting each device to update its keys. Other approaches, such as using a TPM to isolate cryptographic keys may not be available on systems that have been deployed.

A shared key infrastructure can avoid the computational overheads of public key operations. Unfortunately, if the shared key is compromised, the system may remain insecure until the shared keys have been updated across a large number of devices. As an example, in 2011, RSA revealed that it had been compromised [20]. Soon after, RSA replaced many of their authentication tokens, and it is believed that attackers were able to compromise the shared secrets between the authentication tokens and the RSA servers. Accordingly, RSA was forced to reissue tokens since the secrets were exposed.

While both PKI and existing shared-key approaches have drawbacks, we propose a shared-key infrastructure that uses hardware to rate-limit secret exposure, allowing regular operation, but prevents an adversary from gaining all the keying data if a system is compromised. We now describe the system, key deployment, and the security properties that it has.

B. System Overview

To deploy the approach, the security device manufacturer will create a central key server. Then, for each outstation device, the manufacturer will create a random stream of bits and encode both a copy on the device and on the key server. These bits will then be used by both devices to obtain random bits for deriving shared keys.

To derive a shared key, both the central key server and the outstation device will consume the same random bit string from the data they share. Given a shared data set of size M

bits, the data set can be divided into N groups of size $\frac{M}{N}$. These groups, denoted (b_1, b_2, ... b_N), each provide bits of random data that can be used as input to a suitable function, F, to generate a master key. To construct shared secrets for communication, the master key can be used with an incrementing counter to generate a one-time authentication key. Periodically, or upon discovery of a suspected key compromise, either party may abandon the chain and generate a new master key using the next random bits. Each party can verify the new hash chain is valid and transition to the new hash chain [21].

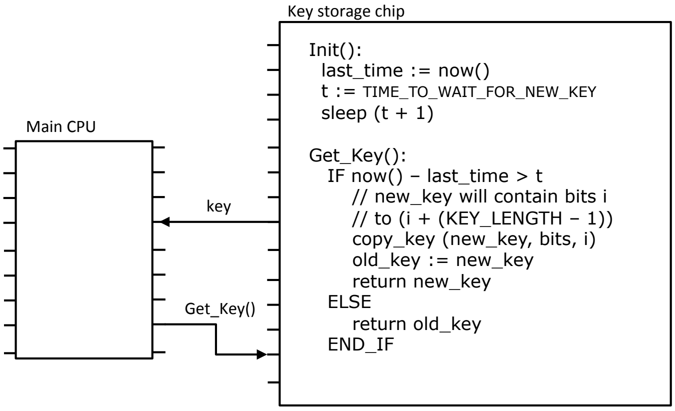
We note that this approach does not rely upon the availability of a TPM. Instead, we use a simple hardware device, such as a USB device, that parcels out random bits on a specific schedule. This device may emulate a USB disk, providing support to a large variety of legacy systems. This device may hold all M bits of shared data and expose only a small set of bits at a time. This approach allows a key store to be used for decades without onerous system requirements. As an example, if we consider a device that exposes 256-bit key at most once every 10 seconds, a device could provide keys for 50 years with about 4.7GB of storage. USB devices with far more memory can be obtained inexpensively today. Since SCADA control systems send three to five packets a second, a 10 second seed refresh frequency would result in hash chains of up to 50 iterations.

Others have used a similar approach. However, in these previous approaches, there is nothing to constrain the generation of new key values; typically, without the use of complicated trusted hardware (e.g., a TPM), an adversarial compromise of key material will compromise all of the keys. In the event of a compromise, a device will need to be given new secret keys. To mitigate key compromise and support key recovery, we simply use a hardware-enforced secret exposure interface to prevent a remote adversary from compromising future target keys.

C. Limiting Key Exposures

In Figure 3, we show a storage device and the control logic it uses to manage key release. This algorithm releases keys no faster than once per `t` seconds. If multiple requests are issued in the same `t` second window, the same key value is returned. If an attacker were to remotely compromise a device that receives a key from a storage device implementing this approach, the attacker can read a key every `t` time units. However, the approach bounds the number of seed values an attacker can learn. If the device is ever reset to a non-compromised state, there is no need to reset the shared secret data on the device. The device would continue generating keys just as if a compromise never occurred, because a remote adversary will not have had access to any future keys. Therefore, the approach ensures an attacker can only learn seed values while the adversary has the system compromised: the attacker does not learn previous or future seed values.

If the deployer can determine when the system was first compromised, limited key exposure may provide the deployer with the ability to determine how long a system may have been providing inaccurate information. Further, once the system is recovered, it immediately regains authenticated communication without updates to the keying infrastructure. This dramatically reduces the maintenance and administrative overheads associated with the system.



```
Init():
  last_time := now()
  t := TIME_TO_WAIT_FOR_NEW_KEY
  sleep (t + 1)

Get_Key():
   IF now() – last_time > t
       // new_key will contain bits i
       // to (i + (KEY_LENGTH – 1))
       copy_key (new_key, bits, i)
       old_key := new_key
       return new_key
   ELSE
       return old_key
   END_IF
```

Fig. 3. Key storage device servicing main CPU. The manufacturer distributes a copy of the key bits to each key storage chip. Only the manufacturer and the key storage chip has a copy of the shared keys.

This approach is robust against a remote adversary if the secret data is distributed securely (e.g., by the manufacturer), each device pair has a different secret and the hash function is computationally difficult to reverse (such as using cryptographically secure one-way hash functions). However, the approach provides no protection against an adversary that can physically alter the data storage device to read the secret values. Accordingly, the device will require physical security protections, as with traditional SCADA devices.

To ensure both devices use the same seed value, they must have synchronized clocks and agree on a starting value. This can be configured at device loading time. Importantly, the outstation device must be able to maintain a clock even if it is being transported to the outstation, necessitating a battery to power the clock. To accommodate timer drift, the approach may allow access to a small number of previous and future keys, allowing the clocks to vary slightly while ensuring continued operation.

D. Implementation Options

Since the key storage device we propose is simple to implement, it can be integrated with SCADA devices in a variety of ways. The device may be integrated as an embedded device, either at the time of manufacture or as an after-market modification. In other cases, the device may be external to the system, attached via a serial interface (e.g., through a serial communication port or a USB interface). This flexibility allows the approach to be integrated with older computer hardware, while providing future-proofing assurances as computer systems evolve.

The operating system may need a driver to interact with the device. In the case of a device attached through a serial communication port, a driver would need to be added to request key values. However, the driver would be extremely simple to implement since the interface is minimal. A custom driver could also be created for a USB-attached device; however, such a device may also be able to reuse existing drivers and present itself as a hard disk[3] This disk would simply present a single file that contained a key value that could be read. Each time the device updated the value, the contents of the file would change, allowing a program to access the values simply by monitoring the file.

The simplicity of the device and its interface with the system provides significant flexibility in deployment.

IX. Discussion

Our implementation is particularly appealing for partial deployment scenarios. A power provider may choose to implement our authentication protocol on outstations before deploying it on the master (or vice versa). If only one communicating end-point deploys the approach, the other communicating system will simply discard the authentication messages. As new components begin deploying the approach, the other endpoint can learn to expect the authentication packets. If the authentication packets stop arriving or a string of packets are not authenticated properly, an alert can be generated for the provider's operators.

Our approach does not attempt to proactively verify incoming messages. Instead, our approach reacts to packets that cannot be validated and generates alerts afterward. This design decision allows us to authenticate messages without introducing latency between the endpoints. Even with our reactionary approach, operators will quickly learn that the information being communicated is not authentic. This allows operators to respond appropriately based on the communicated information. This technique is particularly appealing for SCADA systems in which physical changes occur slowly since an alert can be issued an examined before the affected physical systems are dramatically affected.

In our approach, we proposed a technique to allow key sharing between the parties using inexpensive, backwards-compatible hardware while providing key recovery guarantees. This approach combines the benefits of shared key cryptography with perfect forward and backward security.

Acknowledgements

The authors would like to thank Adam Crain of Automatak for insight into OpenDNP3 and Lammert Bies for his freely available implementation of DNP3's CRC calculation. The authors would also like to thank Evan Frenn for his prior work on surveying related work. The authors would additionally like to thank Stacy Prowell for his insights in early discussions related to the work.

[3] The device would need to signal to the operating system that it would not be allowed to cache file values to ensure programs would see new values.

References

[1] F. Boroomand, A. Fereidunian, M.-A. Zamani, M. Amozegar, H. R. Jamalabadi, H. Nasrollahi, M. Moghimi, H. Lesani, and C. Lucas, "Cyber security for smart grid: A human-automation interaction framework," in *Innovative Smart Grid Technologies Conference Europe (ISGT Europe), 2010 IEEE PES*, 2010, pp. 1–6.

[2] D. Dickinson, "Protecting Water Industry Control and SCADA Systems from Cyber Attacks," http://www.graybar.com/documents/phoenix-contact-protecting-water-industry-control.pdf, accessed: 2013-09-27.

[3] E. D. Knapp and R. Samani, *Applied Cyber Security and the Smart Grid: Implementing Security Controls into the Modern Power Infrastructure*. 225 Wyman Street, Waltham, MA 02451, USA: Elsevier Inc., 2013, ch. Smart Grid Network Architecture.

[4] P. P. Tsang and S. W. Smith, "YASIR: A Low-Latency, High-Integrity Security Retrofit for Legacy SCADA Systems," 2007.

[5] E. D. Knapp and R. Samani, *Applied Cyber Security and the Smart Grid: Implementing Security Controls into the Modern Power Infrastructure*. 225 Wyman Street, Waltham, MA 02451, USA: Elsevier Inc., 2013, ch. Hacking the Smart Grid.

[6] "IEEE Standard for Electric Power Systems Communications-Distributed Network Protocol (DNP3)," *IEEE Std 1815-2012 (Revision of IEEE Std 1815-2010)*, pp. 1–821, 2012.

[7] G. Gilchrist, "Secure authentication for DNP3," in *Power and Energy Society General Meeting - Conversion and Delivery of Electrical Energy in the 21st Century, 2008 IEEE*, 2008, pp. 1–3.

[8] M. Majdalawieh, F. Parisi-Presicce, and D. Wijesekera, "DNPSec: Distributed Network Protocol Version 3 (DNP3) Security Framework," in *Advances in Computer, Information, and Systems Sciences, and Engineering*, K. Elleithy, T. Sobh, A. Mahmood, M. Iskander, and M. Karim, Eds. Springer Netherlands, 2006, pp. 227–234. [Online]. Available: http://dx.doi.org/10.1007/1-4020-5261-8_36

[9] A. Hadbah, A. Kalam, and H. Al-Khalidi, "The subsequent security problems attributable to increasing interconnectivity of SCADA systems," in *Power Engineering Conference, 2008. AUPEC '08. Australasian Universities*, 2008, pp. 1–4.

[10] D.-J. Kang, J.-J. Lee, S.-J. Kim, and J.-H. Park, "Analysis on cyber threats to SCADA systems," in *Transmission Distribution Conference Exposition: Asia and Pacific, 2009*, 2009, pp. 1–4.

[11] H. Khurana, M. Hadley, N. Lu, and D. Frincke, "Smart-grid security issues," *Security Privacy, IEEE*, vol. 8, no. 1, pp. 81–85, 2010.

[12] Y. Simmhan, A. Kumbhare, B. Cao, and V. Prasanna, "An Analysis of Security and Privacy Issues in Smart Grid Software Architectures on Clouds," in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, 2011, pp. 582–589.

[13] "SEL-3021 Serial Encrypting Transceiver Security Policy," https://www.selinc.com/WorkArea/DownloadAsset.aspx?id=2855, accessed: 2013-09-27.

[14] "IEEE Standard Communication Delivery Time Performance Requirements for Electric Power Substation Automation," *IEEE Std 1646-2004*, pp. 1–24, 2005.

[15] A. Shah, A. Perrig, and B. Sinopoli, "Mechanisms to Provide Integrity in SCADA and PCS devices ," in *Proceedings of the International Workshop on Cyber-Physical Systems - Challenges and Applications*, 2008.

[16] "TCPDUMP/LIBPCAP," http://www.tcpdump.org/, accessed: 2013-07-01.

[17] "OpenSSL," http://www.openssl.org/, accessed: 2013-07-01.

[18] "Automatak OpenDNP3," https://github.com/automatak/dnp3, accessed: 2013-07-01.

[19] "Software for RS232 communications," http://www.lammertbies.nl/comm/software/index.html, accessed: 2013-07-01.

[20] R. Richmond, "The RSA hack: How they did it," http://bits.blogs.nytimes.com/2011/04/02/the-rsa-hack-how-they-did-it/, April 2011.

[21] L. Lamport, "Password authentication with insecure communication," *Communications of the ACM*, vol. 24, no. 11, pp. 770–772, 1981.

[22] J. Langill, Ed., *Applied Cyber Security and the Smart Grid: Implementing Security Controls into the Modern Power Infrastructure*. 225 Wyman Street, Waltham, MA 02451, USA: Elsevier Inc., 2013.