Scientific Research Publishing

Journal of Information Security, 2018, 9, 100-132
http://www.scirp.org/journal/jis
ISSN Online: 2153-1242
ISSN Print: 2153-1234

A Security Architecture for SCADA Systems

Arun Velagapalli, Mahalingam Ramkumar

Mississippi State University, Starkville, MS, USA
Email: arunvelagapalli@gmail.com

How to cite this paper: Velagapalli, A. and Ramkumar, M. (2018) A Security Architecture for SCADA Systems. *Journal of Information Security*, **9**, 100-132.
https://doi.org/10.4236/jis.2018.91009

Received: December 14, 2017
Accepted: January 20, 2018
Published: January 23, 2018

Abstract

Supervisory Control and Data Acquisition (SCADA) systems are attractive targets for attackers, as they offer an avenue to attack critical infrastructure (CI) systems controlled by SCADA systems. Ultimately, an attack on any system exploits some undesired (malicious or accidental) functionality in the components of the system. Unfortunately, it is far from practical to eliminate undesired functionality in *every* component of a system. The contribution of this paper is a novel architecture for securing SCADA systems that guarantee that "any malicious modification of the deployment configuration or the design configuration of the SCADA system will be detected"—even if undesired functionality may exist in SCADA system components.

Keywords

SCADA, Security: Trusted Computing: Authenticated Data Structure, Critical Infrastructure, DCS, PCS, Cyber Security, CIP

1. Introduction

Several important systems like power grids, water supply systems, oil and gas production/distribution systems, mass transportation systems, etc., are considered as *critical infrastructure* (CI) systems due to the catastrophic nature of damages that can result from their failure. The task of monitoring and controlling such systems is often entrusted to Supervisory Control and Data Acquisition (SCADA) systems.

SCADA systems are an attractive target for attackers, as they offer an avenue for launching attacks against high valued CI systems. A typical SCADA system may include several remote terminal units (RTU), one or more master terminal units (MTU), a variety of communication equipment and links, computers running human machine interface (HMI) software to enable more intuitive operator driven control when necessary. *Hidden malicious/accidental functionality* in any

SCADA system component could be exploited by an attacker to launch attacks such as the above. Such hidden functionality could exist in (the logic programmed into) programmable logic controllers (PLC) in RTUs and MTUs, in any computer used for programming PLCs, or in any peripheral of the computer running the HMI software or the SCADA data logger, in the operating system of such computers, in the HMI software, or even, ironically, in a computer that runs the intrusion detection system (IDS) intended for protecting the SCADA system.

In 2010, a virus known as Stuxnet[1] that had evaded detection for over a year [1] was identified. This virus targeted nuclear plants, and shut down centrifuges inside the plant by overwriting some set-points. In November 2011, the Illinois Statewide Terrorism and Intelligence Center reported[2] a cyber-attack on a small, rural water utility outside Springfield, where attackers had gained remote access to pumps. In May 2003 [2], a Slammer worm exploiting[3] an un-patched version of Microsoft SQL erased crucial SCADA system logs. "In March 2016, the U.S. Justice Department claimed that Iran had attacked U.S. infrastructure by infiltrating the industrial controls of a dam in Rye Brook, New York. The attackers compromised the dams command-and-control system in 2013 using a cellular modem" [3]. "In December 2015, a power company located in western Ukraine suffered a power outage that impacted a large area that included the regional capital of Ivano-Frankivsk. The cybercriminals had facilitated the outage by using BlackEnergy malware to exploit the macros in Microsoft Excel documents" [4].

It is indeed for very good reasons that such threats have been recognized as "Advanced Persistent Threats" [5] [6] [7] [8] [9]. Due to the high value of targets, the possibility of sophisticated state sponsored attacks has to be considered. Sophisticated malicious functionality may be introduced even during the manufacturing process of various components that could ultimately end up in SCADA systems. In addition, we cannot afford to ignore the possibility that an attacker may have actually participated in the deployment of the SCADA system, or testing of the deployed system, and taken advantage of such an opportunity to inject hidden functionality in some component.

While it is important to take all possible practical steps to reduce the threat of hidden malicious functionality, we may never be able to *eliminate* such functionality in *every* component. Such functionality may be exploited to launch attacks while simultaneously reporting "all clear" messages to the stake-holders. It is of vital importance that we are at the minimum able to *reliably detect* such attacks, *even* if hidden malicious functionality is inevitable.

[1]Stuxnet was able to use about twenty zero-day vulnerabilities [11] in a popular commercial SCADA-system design software to gain control over the plant. Stuxnet has the potential to turn off pumps, control actuators, and still report that everything is normal. Due to its popularity, this worm is freely available, and could lead to more drastic attacks upon re-engineering.

[2]"A hacker calling himself 'Prof' posted screen shots from his computer showing him logged onto the control system of a water utility in the Texas town of South Houston" [12].

[3]The SCADA systems data acquisition server was infected through the corporate network. The entry point for the Slammer worm [13] was discovered to be a laptop.

1.1. Active vs Passive Security Measures

The process of securing any system can be seen as consisting of three broad steps: 1) enumeration of desired assurances; 2) identification of reasonable assumptions; and 3) development of a process, viz., a security protocol, to translate the assumptions into the desired assurances. In other words, **if** we begin with good assumptions, and *if* the security protocol is correct, and **if** the agency responsible for executing the protocol is trustworthy, **then** the desired assurances are guaranteed.

Approaches to secure systems can be broadly classified [10] into:

1) *active* approaches based on attack models; and

2) *passive* approaches based on system-state models.

Underlying active approaches is the notion that violations of desired assurances result from attacks. As attacks exploit pitfalls in the implementation of systems (resulting in undesired functionality) some of the practical tools used in active approaches for identification, removal and/or isolation of attacks, include machine learning for modeling, detecting, and classifying intrusions; isolation mechanisms like hypervisors, containers, and various cryptographic mechanisms; and good programming practices to minimize bugs and potentially malicious functionality. The complexity of most tools used in active approaches render it infeasible to make meaningful assumptions regarding the integrity of both: 1) the tools, and 2) the process that utilizes the tools, to realize the desired assurances. Furthermore, due to the unrestricted freedom of attacks, active approaches will forever be engaged in an evolutionary arms race with attacks.

Passive approaches view digital assets of a system as a dynamic set of states. The desired assurances dictate the nature of protection to be extended to each state, and are expressed in the form of an unambiguous system-state transition model. Executing the model, (or model-driven verification) is a process of 1) actually verifying that the state-transition rules specified by the model are *not* violated when the system is operational, and 2) reporting such findings to stake-holders. To ensure that the model is correct, the model should be made open to scrutiny. In such an event, the extent of trust in the assurances offered by such an approach (that no state violation will go undetected) is only limited by the trust in the environment for model-execution. The novel STCB (SCADA Trusted Computing Base) approach proposed in this paper is a passive approach, which leverages a rigorous standard for a trustworthy model-execution environment.

Current approaches to secure systems, and more specifically, CI systems monitored by SCADA systems, are overwhelmingly active approaches. Ultimately, both active and passive approaches are necessary. Active approaches attempt to deflect/repel as many attacks as possible. Passive approaches diminish the pay-off for attackers, as even state violations resulting from attacks that slip-past active approaches *will* be detected by passive approaches. In addition, active approaches are also essential for the narrow purpose of thwarting attacks that seek

to compromise the integrity of the model-execution environment.

1.2. Trusted Computing Base

For any system with a desired set of security requirements $\mathcal{A}$, the trusted computing base (TCB) is "a small amount of software and hardware we rely on" (to realize the requirements $\mathcal{A}$) and "that we distinguish from a much larger amount that can misbehave without affecting security" [14]. In other words, *as long as the TCB is worthy of trust* the TCB can be leveraged to realize the desired assurances $\mathcal{A}$ regarding the operation of the entire system.

In the proposed *passive* approach to secure SCADA systems, a resource limited trustworthy module—which we shall refer to as an STCB (SCADA Trusted Computing Base) module serves as the TCB for model-execution. The main contributions of this paper are: 1) a strategy for expressing of state-transition models for SCADA systems; and 2) a functional specification for STCB modules, for executing the model.

To improve the confidence in the integrity of STCB modules, they should ideally be manufactured under a well-controlled environment, and consummately tested for the designed functionality. To facilitate consummate testing, it is necessary to *deliberately* constrain STCB modules to possess simple functionality. For low-cost mass-realization of reliable STCB modules to be practical, the simple functions executed inside STCB modules should nevertheless permit them to serve as the TCB for *any* SCADA system—irrespective of the nature and scale of the CI system. While the "instruction set" for specifying the state transition model should be rich enough to be suitable for *any* SCADA system, it should simultaneously be simple enough to be executed even by severely resource limited STCB modules.

The main components of the proposed STCB based security architecture include

1) a systematic strategy for designing SCADA state-transition models for *any* SCADA system, consisting of

a) an instruction set for expressing the model,

b) role of the *designer* of the SCADA system, and

c) role of the *deployer* of the system;

2) a functional specification for STCB modules, suitable for executing the instruction set for *any* SCADA system; and

3) an STCB protocol, for interacting with STCB modules, and obtaining SCADA state reports.

1.3. Organization

The rest of this paper is organized as follows. Section 2 is an overview of STCB approach. Section 3 outlines the STCB design process. Section 4 outlines processes for STCB deployment and operation. Section 5 provides a detailed description of the STCB functionality. Section 6 describes the STCB protocol. Finally, conclusions are offered in Section 7.

2. Overview of STCB Approach

While a state-based security approach can be extended to any system, such an approach is indeed natural for critical infrastructure SCADA systems. Note that the ultimate purpose of a SCADA system, viz., to monitor and report CI system states to stake-holders, is indeed identical to that of state-model based security architecture, consisting of model-driven verification and reporting.

The state reports from a SCADA system can be seen as a function of the current states of all sensors associated with the system. For a SCADA system characterized by n sensors, let $v_1 \cdots v_n$ represent the states of the n sensors, and let

$$\left[o_1 \cdots o_s\right] = \mathcal{F}\left(v_1 \cdots v_n\right) \tag{1}$$

represent a function that captures the "physics" of the controlled system, and reports values $\left[o_1 \cdots o_s\right]$ to the stake-holder as the "state of the system". More specifically, as inputs $v_1 \cdots v_n$ to the SCADA system (sensor measurements) may be received asynchronously, the function $\mathcal{F}(\,)$ is often realized as

$$\mathcal{F} \equiv \mathcal{U}_1(\,) \circ \mathcal{U}_2(\,) \circ \cdots \circ \mathcal{U}_n(\,) \tag{2}$$

where $\mathcal{U}_i, 1 \le i \le n$ is evaluated whenever an fresh measurement v_i is made available. Furthermore, in practical SCADA systems, evaluation of $\mathcal{F}(\,)$ is performed jointly by numerous system components that may include PLCs in multiple RTUs and MTUs, the HMI, and even actions by human operators. Consequently, notwithstanding current active measures, the integrity of the state reports is far from assured. Specifically, current active approaches include features like a) cryptographic protection of links between RTUs and MTUs [15]-[20] to prevent message injection attacks by attackers and b) intrusion detection systems to facilitate early detection of attacks [21] [22] [23] to detect and evade attacks.

The goal of the STCB security model is to *guarantee the integrity of state reports provided by the agent*. To achieve its goals, the STCB security model relies only on a) the integrity of STCB modules, and b) the integrity of clearly defined processes to be adopted by entities identified as the *designer* and the *deployer* of the SCADA system. The designer is an entity with good domain knowledge (regarding the CI system); the deployer is a security professional who is not required to possess any knowledge of the CI system. To the extent the stake-holder trusts the integrity of the STCB modules, and the verifiable processes adopted by the designer and the deployer, the stake-holder is assured of the integrity of the state report—even if malicious functionality may exist in SCADA system components.

2.1. STCB System Components

The additional components introduced into a STCB-secured SCADA system include

1) an *untrusted* "STCB system manager" **U**,

2) STCB modules $\{M_1 \cdots M_k\}$, and M_0.

All STCB modules are identical, and are capable of executing a set of simple TCB functions. Modules $M_1 \cdots M_k$ are "closely bound" to SCADA system sensors. In the rest of this paper we shall use the term sensor module (SM) for STCB modules $M_1 \cdots M_k$, and the term central module (CM), for STCB module M_0.

The untrusted STCB manager **U** periodically receives *sensor reports* from SMs $M_1 \cdots M_k$ and makes them available to CM M_0. CM M_0 evaluates $\mathcal{F}()$, and outputs *state reports*, see Figure 1. From a broad perspective, the authenticity of the *inputs* to $\mathcal{F}()$ are assured by SMs $M_1 \cdots M_k$; the integrity of the function $\mathcal{F}()$ is assured by the CM.

The exact make up of the manager **U** is irrelevant for our purposes of guaranteeing the integrity of $\mathcal{F}()$, as **U** is not trusted. Unless **U** performs it tasks faithfully, valid state reports cann*ot* be sent to the stake holders.

The state reports are relayed by the STCB manager **U** to an STCB module M_r associated with a stake-holder. Any number of stake holder modules like M_r may exist. More generally, a stake-holder module may be the CM for another STCB deployment.

For example, the state reports from different SCADA systems may be provided as "sensor reports" to a system at a higher level of hierarchy. In such a scenario, the stake-holder module M_r can be seen as the CM of an STCB deployment at a higher level of hierarchy. Module M_r considers the state reports from the CMs of systems at the lower level as "sensor reports" from *foreign* STCB deployments.

Any number of hierarchical levels may exist. For example, state reports from multiple SCADA systems in a town may be inputs to a single SCADA system that monitors the health of all such systems in the town. The reports from such

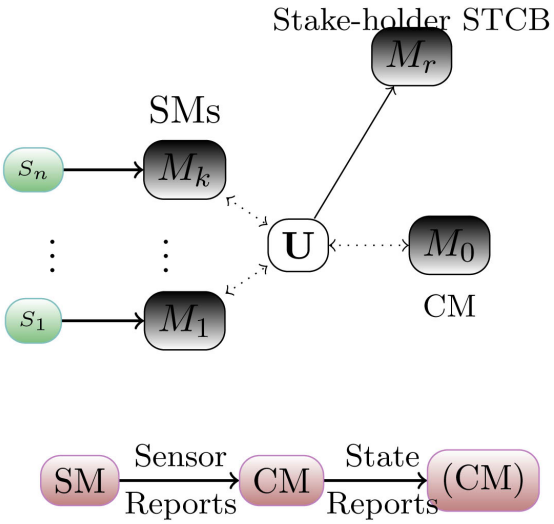

Figure 1. Top: STCB components. Bottom: Information flow in the STCB model. The STCB module M_r associated with the stake-holder can be the CM for a deployment at a higher hierarchical level.

SCADA systems in different towns may be inputs to another SCADA system at an even higher level of hierarchy, that monitors the health of all systems in a state, and so on.

2.2. Evaluating $\mathcal{F}(\,)$

The main challenge lies in the choice of a strategy for evaluating *any* $\mathcal{F}(\,) \equiv \mathcal{U}_1(\,) \circ \mathcal{U}_2(\,) \circ \cdots \circ \mathcal{U}_n(\,)$ inside the trusted confines of resource challenged STCB modules. Recall that we desire to *deliberately* constrain STCB modules to possess only modest memory and computational abilities. Consequently, we constrain STCB modules to perform only logical and cryptographic hash operations. By performing simple logical operations the STCB modules support a simple instruction set $\mathcal{A}$ for representing different $\mathcal{U}_i(\,)$s. As no restrictions are placed on the nature and scale of the SCADA system, some of the specific challenges are that

1) the number of sensors n can be unlimited;

2) evaluation of $\mathcal{U}_i(\,)$'s may require evaluation of complex functions, and thus challenging to represent using merely the instruction set $\mathcal{A}$.

Both challenges are addressed through the use of Merkle hash trees [24].

2.2.1. Merkle Trees

A Merkle tree is a *binary hash tree* which permits a resource limited entity to assure the integrity of a dynamic database of practically *any* size, even while the database is stored in an untrusted location. Specifically, the resource limited entity only needs to store a single cryptographic hash—the root of the tree.

A Merkle hash tree with $n = 2^L$ leaves (for simplicity we shall assume that is a power of (2) has L levels. For storing a database with n records, each record is interpreted as a leaf of the tree. Corresponding to each leaf (record) R is a *leaf-hash* obtained as $h(R)$, by hashing the leaf using a secure cryptographic hash function $h(\,)$.

The n leaf-hashes (say, $v_0^0 \cdots v_{n-1}^0$) corresponding to n records $R_0 \cdots R_{n-1}$ are at level 0 of the binary tree. At level 1 of the tree are $n/2$ leaf hashes $v_0^1 \cdots v_{n/2-1}^1$, where $v_i^1 = h\left(v_{2i}^0, v_{2i+1}^0\right)$. Similarly, the $n/2$ leaves in level 1 result in $n/4$ nodes in level 2, and so on. Construction of the tree stops at level L with a single node $v_0^L = r$—the *root* of the tree. For any leaf R_i (with corresponding leaf node $v_i = h(R_i)$), there exists a set of L *complementary* nodes, say, $\mathbf{v}_i$, and a sequence of L hash operations represented as $f_v(\,)$ such that $f_v(v_i, \mathbf{v}_i) = r$, see **Figure 2**.

Protocols that employ Merkle hash trees can be seen as an interaction between two parties—a prover and a verifier. The *prover* stores all n leaves and all $2n-1$ nodes (distributed over levels 0 to L). The verifier stores only the root r (the lone node at level L). To demonstrate that a record R is part of the tree the prover provides L complementary nodes $\mathbf{v}$ as proof. The verifier accepts record R as authentic only if $r = f_v\left(h(R), \mathbf{v}\right)$. To update record R to R' the verifier simply sets it's root to $\xi' = f_v\left(h(R'), \mathbf{v}\right)$.

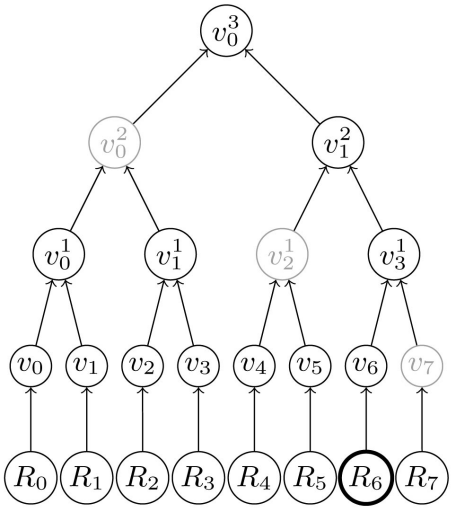

Figure 2. A binary hash tree with 8 leaves. The set of complementary nodes for R_6 are "the siblings of all ancestors of R_6"—v_7 (sibling of v_6), v_2^1 (sibling of ancestor v_3^1) and v_0^2 (sibling of ancestor v_1^2).

2.2.2. Merkle Trees in the STCB Approach

In the STCB approach resource challenged STCB modules store only the root of the tree, and have the ability to perform $f_v()$ operations. This capability is leveraged to assure the integrity of

1) a dynamic database of n sensor measurements;

2) any number of simple static "algorithms" to evaluate different $\mathcal{U}_i()$s, where each algorithm is a small number of instructions (belonging to the instruction set $\mathcal{A}$) supported by STCB modules; and

3) static *look-up tables* (of any size) for evaluating complex functions that may be necessary to execute some (instruction in) $\mathcal{U}_i()$.

Specifically, the STCB module M_0 for an STCB deployment stores a (static) root of a static Merkle tree, and the (dynamic) root of a dynamic Merkle tree. The leaves of the static tree are the specifications for a specific STCB deployment—provided by the designer and the deployer of the system. The leaves of the dynamic tree are the current states of the n sensors of the system. The leaves and all intermediate nodes of both trees, are stored by the untrusted STCB manager **U**.

2.3. STCB Designer and Deployer

One of the main motivations for clearly demarcating between the roles of a designer and a deployer is that entities with good domain knowledge (for example, an entity with in-depth knowledge about the domain of specific CI system, like a nuclear plant) are often unlikely to be security experts. Likewise, security experts are unlikely to be experts in the domain of the specific CI system.

The designer is a domain expert with good knowledge of the CI system. The designer is required to be aware of the purpose of each sensor in the system, and the interpretation of their states. For example, (say) in a water-tank control sys-

tem, "if $S_5 > 100$ (water level greater than 100) S_6 should be zero (the pump should be off)." The responsibility of the designer is to come up with a specification for the function $\mathcal{F}(v_1 \cdots v_n)$ that captures the physics of the system. Specifically, the designer specifies two types of records. For a system with n sensors, the designer specifies

1) n static records $\mathbf{G}_1 \cdots \mathbf{G}_n$ which convey cryptographic commitments to values (instructions, inputs, constants) necessary for executing each $\mathcal{U}_i$;

2) n records $\mathbf{S}_1 \cdots \mathbf{S}_n$ which convey the *initial* state of n sensors. During operation, a sensor report from $\mathbf{S}_i$ will trigger execution of instructions defined in a record $\mathbf{G}_i$ to modify record $\mathbf{S}_i$.

The deployer is a security professional who may not possess any CI system domain knowledge. The responsibility of the deployer is to procure and install STCB modules, and be aware of steps to be taken, for example, to

1) facilitate establishment of shared secrets between modules;

2) securely connect (for example, using tamper-evident connectors) physical sensor outputs to SMs, and record such bindings (for example, (S_5, M_8) indicating that sensor S_5 is connected to module M_8);

3) deploy the STCB manager **U**—which includes installation of all hardware/software necessary to relay SM outputs to the STCB manager **U**, setting up a channel to the CM M_0, and a channel to be used for conveying state reports to stake holders. However, **U**, and such channels, are *not* trusted.

Ultimately, the components of the system-model specified by the deployer take the form of two types of records. Records of type **B** (binding records) specify binding between a sensor identity S_i and the module identity M_j responsible for authenticating reports from S_i. Records of type **R** (or reporting records) specify the identity of the STCB module to which a report regarding a specific system-state is to be made.

3. STCB Design

The designer is entrusted with the responsibility of describing function

$$\mathcal{F}(\,) \equiv \mathcal{U}_1(v_1) \circ \mathcal{U}_2(v_2) \circ \cdots \circ \mathcal{U}_n(v_n)$$

where $\mathcal{U}_i(v_i), 1 \le i \le n$ is evaluated whenever an fresh measurement from sensor S_i is available.

3.1. STCB Design Tree

The designer provides a specification of $\mathcal{U}_i, 1 \le i \le n$ by constructing a static Merkle tree—the design tree—with root ξ_s. The tree includes $n+1$ leaves

$$\xi_0, \mathbf{G}_1 \cdots \mathbf{G}_n \tag{3}$$

where ξ_0 is itself a root of a Merkle tree with n leaves. Each leaf specifies the *initial* state of n sensors as records $\mathbf{S}_i, 1 \le i \le n$. The other n leaves correspond to the n design records $\mathbf{G}_i, 1 \le i \le n$. The sensor records and design records are of the form

$$\begin{aligned}\mathbf{S}_i &= \left[S_i, v_i, t_i, o_{i_1} \cdots o_{i_w}, \tau_i\right] \\ \mathbf{G}_i &= \left[S_i, \alpha_i, \lambda_i, \left\{S_{i,1} \cdots S_{i,q}\right\}, S_i'\right]\end{aligned} \tag{4}$$

Each sensor is associated with a set of $w+3$ *dynamic* values, where w is a constant. Specifically, 1) v_i is the latest measurement of sensor S_i; 2) t_i is the time of the measurement; 3) $o_{i_1} \cdots o_{i_w}$ are the w outputs of function $\mathcal{U}_i$, evaluated when the last report (v_i, t_i) was received from sensor S_i. The value τ_i is a measure of time associated with the outputs $o_{i_1} \cdots o_{i_w}$ (and is not necessarily the same as t_i).

In the design record $\mathbf{G}_i$ for sensor S_i, value $\alpha_i = h(\mathbf{A}_i)$ is the hash of a small number (say, m) of instructions chosen from the set $\mathcal{A}$. Specifically, the instructions $\mathbf{A}_i$ define the function $\mathcal{U}_i$ to be evaluated on receipt of a report from sensor S_i. The value λ_i is a one way function of a set of (say, l) constants $\mathbf{C}$. Such constants may specify various values like set-points, permitted ranges of measurements, minimum expected frequency of reports from sensors, etc. In addition, such constant values may also be used as look up tables. The values $S_{i,1} \cdots S_{i,q}$ specify (up to) q *related* sensors. Such sensors are "related" to S_i as the states of such sensors can influence $\mathcal{U}_i$. Some or all of the q values $S_{i,1} \cdots S_{i,q}$ can be set to zero if less than q related sensors suffice. Finally, the value S_i' is optional, and is the identity of a "synthetic" sensor (explained later).

3.1.1. Example System

To describe different steps involved in the construction of a design tree we will use a simplified version of a thermal power plant with six sensors $S_1 \cdots S_6$ as a running example, see **Figure 3**.

S_1 temperature sensor inside boiler.
S_2 coal weight sensor (coal fed into the boiler).
S_3 position of fire regulator.
S_4 temperature inside turbine cell.
S_5 pressure inside turbine cell.
S_6 speed of turbine.

The sensor records $S_1 \cdots S_6$ are of the form

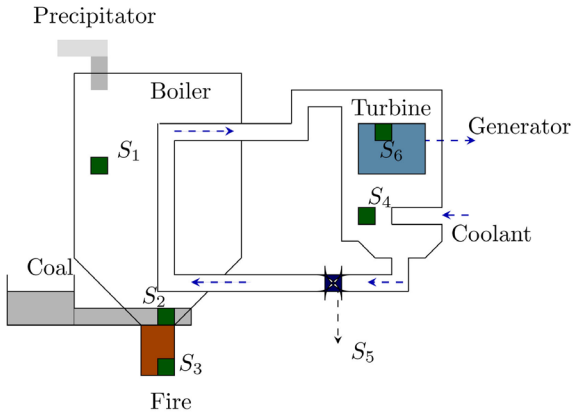

Figure 3. Example: simplified version of thermal power plant.

$$\mathbf{S}_1 = \left[S_1, v_1, t_1, o_{1_1} \cdots o_{1_w}, \tau_1 \right]$$
$$\vdots$$
$$\mathbf{S}_6 = \left[S_6, v_6, t_6, o_{6_1} \cdots o_{6_w}, \tau_6 \right]$$

The design records $G_1 \cdots G_6$ are of the form

$$\mathbf{G}_1 = \left[S_1, \alpha_1, \lambda_1, \left\{ S_{1,1} \cdots S_{1,q} \right\}, S_1' \right]$$
$$\vdots$$
$$\mathbf{G}_6 = \left[S_6, \alpha_6, \lambda_6, \left\{ S_{6,1} \cdots S_{6,q} \right\}, S_6' \right]$$

3.2. Inputs and Outputs of $\mathcal{U}_i$

Due to limited memory inside STCB modules, there is a need for a strict upper bound on the number of inputs to, and outputs of, each $\mathcal{U}_i$. In other words, irrespective of the total number of sensors *n*, note that $\mathcal{U}_i$s are restricted to specifying only 1) up to *q* related sensors, 2) *l* constants, 3) *m* instructions, and 4) one synthetic sensor as inputs. Each $\mathcal{U}_i$ produces *w* outputs.

As $\mathcal{U}_i$ is re-evaluated whenever a fresh report $\tilde{v}_i, \tilde{t}_i$ is available from sensor S_i, the inputs necessary to evaluate $\mathcal{U}_i$ are stored in reserved volatile registers inside STCB modules, and include

1) values in the record $\mathbf{S}_i$ associated with sensor S_i (stored in a register S_0 inside the module);

2) values $\tilde{v}_i, \tilde{t}_i$ in a fresh report from sensor S_i (register $\mathbf{r}$);

3) values in records $\mathbf{S}_{i,1} \cdots \mathbf{S}_{i,q}$ for related sensors $S_{i,1} \cdots S_{i,q}$ (registers $s_1 \cdots s_q$);

4) l constants in $\mathbf{C}_i$ (register $\mathbf{c}$); and

5) m instructions $\mathbf{A}_i$ (some of which may be set to 0 to represent "no operation" if *m* instructions are not required to evaluate $\mathcal{U}_i$);

The m logical operations in $\mathbf{A}_i$ provide the instructions to recompute the outputs $\tilde{o}_{i_1} \cdots \tilde{o}_{i_w}$ of $\mathcal{U}_i$ following a fresh report from S_i. On evaluation of $\mathcal{U}_i$ the record $\mathbf{S}_i$ is modified. Specifically,

1) $\tilde{v}_i, \tilde{t}_i$ replace the previous values $\left(v_i, t_i\right)$,

2) outputs $\tilde{o}_{i_1} \cdots \tilde{o}_{i_w}$ replace outputs $o_{i_1} \cdots o_{i_w}$ of the previous execution of $\mathcal{U}_i$,

3) and τ_i is replaced with

$$\tilde{\tau}_i = \min\left(\tau_{i_1} \cdots \tau_{i_q}, \tilde{t}_i \right) \tag{5}$$

to reflect the *staleness* of the *w* outputs.

Note that dynamic values associated with any S_i may be affected not just by values corresponding to sensors directly related to S_i, but also sensors *indirectly* related to S_i—for example sensors related to a related sensor S_j (once removed) or sensors related to a sensor related to S_j (twice removed) and so on. Computing the value τ as in Equation (5) ensures that the value τ_i will be the least of the sensor-report time *t* corresponding to every sensor that is directly or

indirectly related sensor S_i.

On a continuous basis, as and when new sensor reports are available, the states of the reporting sensors are modified. A subset of dynamic values corresponding to a subset of sensors may be reported to the stake-holder as values $o_1 \cdots o_m$ describing the state of the system. For example, if value o_{j_2} (second output of $\mathcal{U}_i$) is one of the values reported as the state of the system, the time associated with the state o_{j_2} is reported as τ_j.

3.2.1. Example $\mathcal{U}_i$ for Power Plant

For the example system 0.0.3, let the maximum number of related sensors be $q=3$; the number of outputs of each $\mathcal{U}_i$ be $w=2$; and the number of constants $l=8$. In this example, say there exists a rule that the values v_1, v_2 of sensors S_1, S_2 have to be within threshold ranges $\left(x_l^1, x_h^1\right), \left(x_l^2, x_h^2\right)$ respectively. The design of function $\mathcal{U}_1$ checks if v_1, v_2 are within thresholds $\left(x_l^1, x_h^1\right), \left(x_l^2, x_h^2\right)$ respectively. The inputs are v_1, v_2 and constants. The output is written in o_{0_1}. For evaluation of $\mathcal{U}_1$, S_2 is specified as a related sensor. As no other related sensors are used, S_{1_2} and S_{1_3} are set to 0.

As $\mathcal{U}_1$ is re-evaluated whenever a fresh report $\tilde{v}_1, \tilde{t}_1$ is available from sensor S_1, the inputs necessary to evaluate $\mathcal{U}_1$ are stored in reserved volatile registers inside STCB modules, and include

1) values in the record $\mathbf{S}_1$ associated with sensor S_1 (stored in a register s_0 inside the module);

2) values $\tilde{v}_1, \tilde{t}_1$ in a fresh report from sensor S_1 (register **r**);

3) values of record $\mathbf{S}_2$ is specified for $S_{1,1}$ for related sensors, and $S_{1,2}$ and $S_{1,3}$ are set to 0 (registers $s_1 \cdots s_3$);

4) l constants in $\mathbf{C}_1$ (register **c**); and

5) m instructions $\mathbf{A}_1$

The m logical operations in $\mathbf{A}_i$ provide the instructions to recompute the outputs $\tilde{o}_{0_1}$ of $\mathcal{U}_1$ following a fresh report from S_1. On evaluation of $\mathcal{U}_1$ the record $\mathbf{S}_1$ is modified. Specifically,

1) $\tilde{v}_1, \tilde{t}_1$ replace the previous values $\left(v_1, t_1\right)$,

2) output $\tilde{o}_{0_1}$ replaces output o_{0_1} of the previous execution of $\mathcal{U}_1$,

3) and τ_1 is replaced with

$$\tilde{\tau}_1 = \min\left(\tau_{1_1} \cdots \tau_{1_3}, \tilde{t}_1\right) \tag{6}$$

to reflect the *staleness* of the w outputs.

3.3. Synthetic Sensors

The sensors $S_1 \cdots S_n$ can be of three types—*real* sensors, *state-report* sensors, and *synthetic* sensors.

Real sensors are physical sensors in the SCADA deployment. Specifically, during the STCB deployment phase, real sensors are bound to SMs.

State reports from a foreign STCB system are seen by the receiving CM as a

"sensor" report; as such reports are authenticated by the CM of the foreign deployment, state-report sensors are bound to foreign CMs.

Synthetic sensors are *not* bound to CMs or SMs. In a design record $\mathbf{G}_i$, if $S_i' \neq 0$, implies that evaluation of $\mathcal{U}_i$ results in the "synthesis of a fresh report from a (synthetic) sensor $S_i' = S_j$". Just as a fresh report from a sensor S_i should be followed by evaluation of $\mathcal{U}_i$, a fresh report from synthetic sensor $S_j = S_i'$ should be followed by evaluation of $\mathcal{U}_j$.

The primary motivation for using such synthetic sensors is to cater for complex $\mathcal{U}_i$ where the fixed number of (m) instructions in $\mathbf{A}_i$ may be insufficient. By specifying a synthetic sensor $S_j = S_i'$, evaluation of $\mathcal{U}_i$ is *continued* as evaluation of $\mathcal{U}_j$. Similarly, evaluation of $\mathcal{U}_j$, specified by the designer as $\mathbf{G}_j = \left[S_j, S_{j,1} \cdots S_{j,q}, \alpha_j, \lambda_j, S_j'\right]$ may be continued again, if necessary, by specifying $S_j' \neq 0$.

3.4. Constants and Look-Up Tables

In general, the value λ_i —which is a one way function of constants required to evaluate $\mathcal{U}_i$—may be a function of *multiple* sets of *l* constants (*l* constants in each set). More specifically, λ_i is itself the root of a Merkle tree, where each leaf specifies a set of *l* constants. Any number of such leaves may exist, with a minimum of one.

Permitting an unlimited number of constants facilitates the use of look-up tables (LUT) for evaluating $\mathcal{U}_i$. An LUT for evaluating a complex function $y = f(x)$ will have many sets of *l* constants—say $C_{j,1} \cdots C_{j,l}$ where there are no practical limits on *j*. In each set $c_1 = C_{j,1} \cdots c_l = C_{j,l}$ two of the *l* constants will specify the range of the independent variable *x*, and one will specify the corresponding dependent variable *y*. For 2 dimensional LUTs of the form $y = f(x_1, x_2)$, four of the *l* constants will specify the ranges for the two independent variables, and a fifth constant will specify the corresponding value of *y*.

Special instructions (say LUT1 and LUT2) in the instruction set $\mathcal{A}$ will specify the operands—the dependent and independent variables. As one possible design of the two instructions, instruction LUT1 interprets constants c_1 and c_2 as the range of the independent variable *x* and constant c_3 as the corresponding dependent variable *y*. Before the module executes the instruction LUT1, it expects the value of the input operand to be within the range of constants c_1 and c_2—else the execution will not proceed. If the input operand satisfies the requirement, then the value of the output operand is set to c_3. Similarly, for LUT2, constants c_1 and c_2 specify the range of the first input operand x_1; c_3, c_4 specify the range of the second operand x_2; c_5 is the corresponding output *y*.

3.5. Instruction Set $\mathcal{A}$

Each instruction in $\mathcal{A}$ specifies a logical operation (opcode), input operands (1, 2 or 3) depending on the type of opcode, and an output operand. The operands are restricted to be values in STCB registers $\mathbf{s}_0 \cdots \mathbf{s}_q$, **c**, **r**, etc. Specifically, as the

instructions in each $\mathcal{U}_i$ can modify only values in register $\mathbf{s}_0$ (the current state of sensor S_i when $\mathcal{U}_i$ is computed), only such values, and a temporary register T can be specified as output operands.

Examples of simple logical operations include traditional operations like addition/subtraction, logical operations, bit-wise operations, COPY, MOV, etc., and some special instructions like LUT1 and LUT2. Other potentially useful special instructions for SCADA systems is a bounds checking operation CHKB which checks a specific value is within set-points specified as constants and tolerance checking TOL where two values are verified to be close enough—within a tolerance specified by a third value.

Ultimately, a comprehensive specification for STCB modules will fix values like the number of related sensors q, number of outputs w, and the number of constants I (and hence the number of addressable values in the STCB registers). Such a specification will also include a detailed listing of all permitted opcodes and their interpretation. This paper, however, is restricted to describing some of the salient features of STCB modules.

3.5.1. Instructions for Thermal Power Plant Example

For the example system 3.1.1, let $v_1 \cdots v_6$ be the values reported by sensors $S_1 \cdots S_6$ required to determine the state of the system.

Let us assume that the state report expected by the stake-holder is a single bit value— $o_1 = 1$ if the system is in an acceptable state, and $o_1 = 0$ otherwise. According to the designer, the system is in an acceptable state if the following conditions are satisfied:

1) $v_1 \cdots v_6$ are all within thresholds $\left(x_l^1, x_h^1\right) \cdots \left(x_l^6, x_h^6\right)$ respectively, where $\left(x_l^i, x_h^i\right)$ represents lower and higher thresholds for sensor S_i.

2) The speed of turbine should be between upper and lower limits depending on the temperature and pressure inside the turbine cell. $v_6 = f_1\left(v_4, v_5\right) \pm \delta_1$ where δ_1 is another threshold (the speed of the turbine should be a specific function of the pressure and temperature inside the turbine cell).

3) The position of the fire regulator should be between upper and lower limits depending on the current speed of turbine v_6 and the current temperature and pressure values inside the boiler cell. $v_3 = f_2\left(y_1, v_6\right) \pm \delta_2$ where $y_1 = f_3\left(v_1, v_2\right)$ is a function of the temperature and pressure of the boiler.

Let the maximum number of related sensors be $q = 3$; the number of outputs of each $\mathcal{U}$ be $w = 2$; and the number of constants $I = 8$. A possible design of functions $\mathcal{U}_i, 1 \le i \le 6$ is as follows:

1) $\mathcal{U}_1$ checks if v_1, v_2 are within thresholds $\left(x_l^1, x_h^1\right), \left(x_l^2, x_h^2\right)$ respectively. The inputs are v_1, v_2 and constants. The output is written in o_{0_1}. For evaluation of $\mathcal{U}_1$, S_2 is specified as a related sensor. As no other related sensors are used, S_{1_2} and S_{1_3} are set to 0.

2) $\mathcal{U}_2$ performs LUT2 operation for function $f_3(\,)$. The inputs are v_1, v_2 and an LUT leaf. The output is written o_{0_2}. For chaining the output of $\mathcal{U}_1$ (now stored in register s_1) to entire system state, the current value at output

register o_{1_1} of S_1 is copied to S_2's output register o_{0_1}.

3) $\mathcal{U}_3$ regards S_2 and S_6 as related sensors, and perform LUT2 operation $f_2()$ on o_{1_2} (an output of related sensor S_2) and value $u_2 = v_6$ of other related sensor S_6 (now stored in register s_2. The contents of output register o_{1_1} of S_2 are copied to o_{0_1} of S_3.

4) $\mathcal{U}_4$ checks if v_4, v_5 are within thresholds $\left(x_l^4, x_h^4\right), \left(x_l^5, x_h^5\right)$ respectively. The inputs are v_4, v_5 and constants. The output is provided in register o_{0_1}. S_5 is specified as a related sensor of S_4.

5) $\mathcal{U}_5$ regards S_4 as related sensor and perform a LUT2 operation $f_1()$ on value v_4 of related sensor S_4 (value u in record s_1) and v_5 (u in register s_0), the output is stored in o_{0_2}. The content register o_{1_1} (of S_4) are copied to o_{0_1} (of S_5).

6) $\mathcal{U}_6$ regards S_3 and S_5 as related sensors. This function checks if v_3, v_6 are within thresholds $\left(x_l^3, x_h^3\right), \left(x_l^6, x_h^6\right)$ respectively and the output is stored at register o_{0_1} of S_6. In addition, $\mathcal{U}_6$ also performs the following steps:

a) if v_3 available at u_1 satisfies $f_2 \pm \delta_2$ —the result of f_2 is now available at o_{1_2} of related sensor S_3; the result of the check is stored at o_{0_2} of S_6. The result of an AND operation performed on outputs in o_{0_1} and o_{0_2} is stored back in o_{0_1};

b) if v_5 available at u_2 satisfies $f_1 \pm \delta_1$ —the output of f_1 is available at o_{2_2} of related sensor S_5; the 1/0 result is stored in register o_{0_2}.

A result of AND operation of output registers of $S_6 — 0_{0_1}$, 0_{0_2} placed in 0_{0_1} of S_6 represents the entire state of system (acceptable-1, unacceptable-0).

The constants required to evaluate $\mathcal{U}_i$ are provided as a leaf that can be proved against corresponding root λ. The designer specifies the following constant trees:

1) A tree with one leaf with 8 constants $\left[x_l^1, x_h^1, x_l^2, x_h^2, x_l^4, x_h^4, x_l^5, x_h^5\right]$ with root λ_a.

2) A tree with one leaf with 8 constants $\left[\delta_1, \delta_2, x_l^6, x_h^6, x_l^3, x_h^3, 0, 0\right]$ with root λ_b.

3) Three trees—one for a 2D LUT for function $f_1()$ with root λ_c; one for a 2D LUT for function $f_2()$ with root λ_d; and the third for 2D LUT for function $f_3()$ with root λ_e.

With available information from $U_1 \cdots U_6$ the designer specifies the following design records:

$$\mathbf{G}_1 = \left[S_1, \{S_2, 0, 0\}, \alpha_1, \lambda_a, 0\right]$$

$$\mathbf{G}_2 = \left[S_2, \{S_1, 0, 0\}, \alpha_2, \lambda_e, 0\right]$$

$$\mathbf{G}_3 = \left[S_3, \{S_2, S_6, 0\}, \alpha_3, \lambda_d, 0\right]$$

$$\mathbf{G}_4 = \left[S_4, \{S_5, 0, 0\}, \alpha_4, \lambda_a, 0\right]$$

$$\mathbf{G}_5 = \left[S_5, \{S_4, 0, 0\}, \alpha_5, \lambda_c, 0\right]$$

$$\mathbf{G}_6 = \left[S_6, \{S_3, S_5, 0\}, \alpha_6, \lambda_b, 0 \right] \tag{7}$$

where $\alpha_1 \cdots \alpha_2$ are hashes of instructions outlined in Table 1.

4. STCB Deployment

The deployer of the SCADA system is trusted to verify the integrity of the physical bindings between various sensors and SMs. Specifically, the deployer is required to permanently connect the outputs of every sensor to an SM, and apply tamper-evident seals to such connections. The deployer specifies *binding* records of the form

Table 1. Instructions for thermal power plant.

Algorithm	OPCODE	Input registers	Output registers
	CHKB	$\mathbf{c}:C_1, \mathbf{s}_0:u_0$	T
α_1	CHKB	$\mathbf{c}:C_3, \mathbf{s}_1:u_1$	$\mathbf{s}_0:o_{0_2}$
	AND	$T, \mathbf{s}_0:o_{0_2}$	$\mathbf{s}_0:o_{0_1}$
	LUT2	$\mathbf{s}_1:u_1, \mathbf{s}_0:u_0$	$\mathbf{s}_0:o_{0_2}$
α_2	COPY	$\mathbf{s}_1:o_{1_1}$	$\mathbf{s}_0:o_{0_1}$
	COPY	$\mathbf{s}_1:o_{1_1}$	$\mathbf{s}_0:o_{0_1}$
α_3	LUT2	$\mathbf{s}_1:o_{1_2}, \mathbf{s}_2:u_2$	$\mathbf{s}_0:o_{0_2}$
	CHKB	$\mathbf{c}:C_5, \mathbf{s}_0:u_0$	T
α_4	CHKB	$\mathbf{c}:C_7, \mathbf{s}_1:u_1$	$\mathbf{s}_0:o_{0_2}$
	AND	$T, \mathbf{s}_0:o_{0_2}$	$\mathbf{s}_0:o_{0_1}$
	LUT2	$\mathbf{s}_1:u_1, \mathbf{s}_0:u_0$	$\mathbf{s}_0:o_{0_2}$
α_5	COPY	$\mathbf{s}_1:o_{1_1}$	$\mathbf{s}_0:o_{0_1}$
	CHKB	$\mathbf{c}:C_3, \mathbf{s}_0:u_0$	T
	CHKB	$\mathbf{c}:C_5, \mathbf{s}_1:u_1$	$\mathbf{s}_0:o_{0_2}$
	AND	$T, \mathbf{s}_0:o_{0_2}$	$\mathbf{s}_0:o_{0_1}$
	MOV	$\mathbf{c}:C_2$	T
α_6	TOL	$\mathbf{s}_1:o_{1_2}, \mathbf{s}_1:u_1$	$\mathbf{s}_0:o_{0_2}$
	AND	$\mathbf{s}_0:o_{0_1}, \mathbf{s}_0:o_{0_2}$	$\mathbf{s}_0:o_{0_1}$
	MOV	$\mathbf{c}:C_1$	T
	TOL	$\mathbf{s}_2:o_{2_2}, \mathbf{s}_2:u_2$	$\mathbf{s}_0:o_{0_2}$
	AND	$\mathbf{s}_0:o_{0_1}, \mathbf{s}_0:o_{0_2}$	$\mathbf{s}_0:o_{0_1}$

$$\mathbf{B}_i = \left(S_i, M_j, \epsilon_i, \theta\right) \tag{8}$$

to convey that

1) measurements corresponding to sensor S_i will be reported by a module M_j; if S_i is a real sensor, then M_j is an SM; if S_i is a state-report sensor, then M_j is the identity of the CM of a foreign system. Recall that synthetic sensors are not bound to CMs or SMs and thus have no binding records. The deployer may be totally oblivious of the existence of such records;

2) ϵ_i is an *achievable minimum round-trip duration* between module M_j, and the CM for the deployment; and

3) $\theta = 0$ implies a record corresponding to a real sensor; $\theta \neq 0$ implies report from a foreign STCB system with STCB descriptor $\xi_{sp} = \theta$.

To indicate that a module M_0 was deployed as the CM for the STCB system, the binding records includes a record for the CM M_0 as

$$\mathbf{B}_0 = \left(S = 0, M_0, \epsilon = 0, \theta = 0\right)$$

Depending on the requirements specified by the stake holder, the deployer also specifies *reporting* records of the form

$$\mathbf{R}_j = \left(S_r, M_r, S_j, l\right), 1 \le l \le w \tag{9}$$

to indicate that the value o_{j_l} (corresponding to sensor S_j) should be reported to the stake-holder module M_r, and that the report should indicate o_{j_l} as the latest (time τ_j) "measurement" from state-report sensor S_r.

All such $\mathbf{B}_i$ and $\mathbf{R}_j$ are included as leaves of a static Merkle hash tree constructed by the deployer—the STCB *deployment tree*. Let ξ_p be the root of the deployment tree with leaves

$$\left\{\cdots \mathbf{B}_i \cdots\right\}, \left\{\cdots \mathbf{R}_j \cdots\right\} \tag{10}$$

The end-result of the design and deployment processes are two hash trees with roots ξ_s and ξ_p. The design root ξ_s can be seen as concise representation of the physics $\mathcal{F}\left(v_1 \cdots v_n\right)$ of the system. The deployment root ξ_p is a concise representation of the bindings between real-sensors & SMs, and state-report-sensors & CMs of foreign STCB deployments. The value

$$\xi_{sp} = h\left(\xi_s, \xi_p\right) \tag{11}$$

can now be seen as the root of a Merkle tree with two sub-trees—the design tree to the left, with root ξ_s, and the deployment tree to the right, with root ξ_p. The static value ξ_{sp} is the unique descriptor for a specific STCB deployment (deployed to secure a specific SCADA system), see **Figure 4**.

Note that two different deployments of identical SCADA systems may have the same design root ξ_s, but will have different deployment root ξ_p as different STCB modules will be used in the two deployments. If $h(\,)$ is collision resistant, no two STCB deployments will have the same descriptor ξ_{sp}.

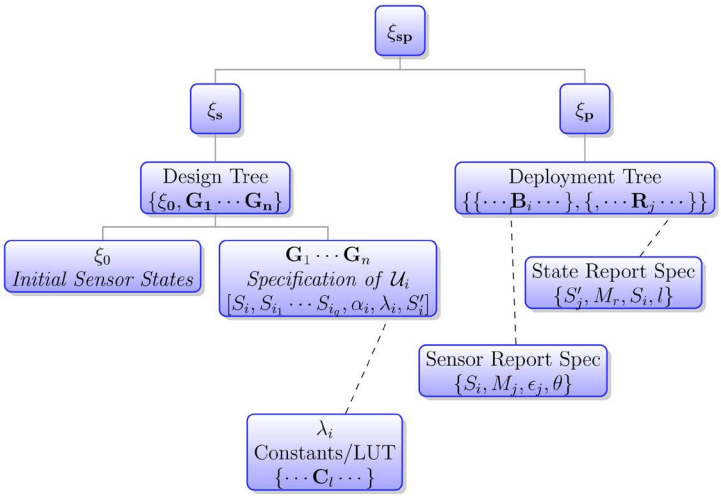

Figure 4. Static descriptor ξ_{sp} is a concise summary of the specification for an STCB system.

4.1. STCB Operation

To commence operation in a SCADA deployment, the STCB module M_0 associated with the system is initialized with the STCB descriptor ξ_{sp}, and the value ξ_0 corresponding to the initial state of sensors. All leaves of the tree with root ξ_{sp} (which includes two sub-trees—the design tree and the deployment tree) are stored by **U**; all n leaves of the form $\mathbf{S}_i, 1 \le i \le n$ corresponding to the initial states of all sensors are also stored by **U**.

During regular operation of the system the STCB manager **U** receives authenticated sensor reports from SMs (and possibly CMs from foreign deployments), and submits them one at a time, to the CM M_0. Such reports take the form of a *message authentication code* (MAC) computed as

$$\mu = h\left(S, v, t, c, \xi_{sp}, K'\right) \tag{12}$$

where

1) values (S, v, t) indicates a report for sensor S to convey a fresh measurement v and measurement time t;

2) c is the clock-tick value of the module that created the report;

3) ξ_{sp} is STCB descriptor of the creator of the report (which was the value used to initialize the module);

4) K' is a shared secret between the creator (SM or foreign CM) and receiver (CM M_0).

Along with the report from sensor S_i (authenticated by a module M_j), **U** also submits a binding record $\mathbf{B}_i$ constructed by the deployer, consistent with the static root ξ_{sp}.

The STCB manager **U** is then required to submit other values required for the CM to evaluate $\mathcal{U}_i$. Such values include: 1) sensor state record $\mathbf{S}_i$; 2) the state records of the (up to) q related sensors; 3) a design record $\mathbf{G}_i$ consistent with descriptor ξ_{sp}; 4) l constant values $\mathbf{C}_i$; and 5) m instructions $\mathbf{A}_i$.

As dynamic sensor states associated with the n sensors are maintained by **U** as leaves of dynamic a Merkle tree with root ξ, sensor state records provided by **U** will be accepted as valid only if they can be verified to be consistent with a copy of the dynamic root ξ stored inside the CM M_0. The instructions $\mathbf{A}_i$ will be accepted as valid only if $\alpha_i = h(\mathbf{A}_i)$ where α_i is specified in the design record $\mathbf{G}_i$. The set of constants $\mathbf{C}_i$ will be accepted as valid only if $h(\mathbf{C}_i)$ can be demonstrated to be a node in the binary tree with root λ_i —where λ_i is specified in the design record $\mathbf{G}_i$.

The values used for evaluating $\mathcal{U}_i$ have fixed reserved locations in the internal memory of the STCB modules, and are specified as the operands for the m instructions in **A**. An internal module function $f_{eval}(\mathbf{A}_i)$ executes every instruction sequentially. The end result is the modification of the values in the sensor state record $\mathbf{S}_i$ of S_i. To "remember" such changes to $\mathbf{S}_i$ the module modifies the dynamic root ξ.

If untrusted **U** does *not* modify $\mathbf{S}_i$ in the same manner, then $\mathbf{S}_i$ will no longer be consistent with the root ξ stored inside the CM. Thus, *if* **U** provides a fresh sensor report to the CM to invoke $\mathcal{U}_i$, it is forced to modify the state of S_i exactly in the manner specified by the designer.

At any time, a reporting record $\mathbf{R} = (S_r, M_r, S_j, l \in \{1 \cdots w\})$ consistent with ξ_{sp} can be provided as input, along with the state record $\mathbf{S}_j$ consistent with ξ, to request the STCB module M_0 to report (to module M_r) values o_{j_l} and τ_l in a state-report "sensor" S_r.

If **U** does not invoke functions to evaluate any $\mathcal{U}_i$, the time t_i associated with S_i cannot be updated. Thus, in any state-report that directly or indirectly depends on sensor S_i the time τ will be stuck at t_i, and will thus be recognized as stale by the stake-holder (or CM of a foreign deployment).

4.1.1. STCB Interfaces

To interact with the STCB modules (SMs and CM), **U** employs various interfaces exposed by the modules. An interface $F_{peer}()$ is used to set up shared keys between modules that send and receive sensor/state reports. A function $F_{hs}()$ can be invoked to engage two modules in a hand-shake sequence within a duration ϵ, to enable the modules to estimate their respective clock offsets with an error less than ϵ. Interface $F_{init}()$ is used to initialize a module as a CM for a deployment with identifier ξ_{sp}, or as a SM for a specific *real* sensor S_{in} (in an STCB deployment with identifier ξ_{sp}).

A function $F_{snd}()$ is invoked to request a module to send a report to another module, in which $F_{rcv}()$ is invoked to accept the report. Function $F_{ld}()$ is used to load various values necessary to evaluate some U_i. Function $F_{upd}()$ is then invoked to evaluate $\mathcal{U}_i$ and update the root ξ stored inside the module. Functions $F_{rcv}(), F_{ld}$ and $F_{upd}()$ are utilized only in modules used as CMs.

To the extent the SMs and the deployer (who is trusted to verify and specify bindings between sensors and SMs) are trusted, we can trust the authenticity of sensor reports provided to the CM module. To the extent the CM module M_0,

and the designer (who is responsible for specifying the functions $\mathcal{U}_i(\,)$) are trusted, the stake-holder trusts the integrity of the state reports.

In practice, the SMs will need to be located as close as possible to the sensors to improve the security of the binding between sensors and SMs. The CM could be housed in any location—for example, a secure location far removed even from the SCADA control center. Components of **U** will need to be housed close to SMs, *and* close to the CM.

5. STCB Architecture

STCB modules have a unique identity, and a unique secret issued by a trusted key distribution center (KDC). Two modules (say) M_i and M_j can use their respective secrets (say) Q_i and Q_j issued by the KDC to compute a common pairwise secret K_{ij} [25]. Specifically, associated with a pair of modules M_i, M_j is a non-secret value P_{ij} which is also made available by the KDC (for example, in a public repository), where P_{ij} is computed as

$$P_{ij} = h\left(Q_i, M_j\right) \oplus h\left(Q_j, M_i\right) \tag{13}$$

Module M_i and M_j can compute a common secret $K_{ij} = h\left(Q_i, M_j\right)$,

$$K_{ij} = \begin{cases} h\left(Q_i, M_j\right) \oplus 0 & \text{Computed by Module } M_i \\ h\left(Q_j, M_i\right) \oplus P_{ij} & \text{Computed by Module } M_j \end{cases} \tag{14}$$

Every module possesses three values that are affected whenever a module is powered on:

1) a clock tick counter c, which is set to 0;

Self-secret	γ
Static root	ξ_{sp}
Dynamic root	ξ
Peer module params	$\mathbf{p} = \{M', \sigma', K_{in}, K_{out}\}$
Constants	$\mathbf{c} = [C_0 \cdots C_l]$
Sensor report register	$\mathbf{r} = \{\tilde{S}, \tilde{v}, \tilde{t}\}$
Sensor state register	$\mathbf{s}_0 = \left[S_0, u_0, t_0, o_{0_1} \cdots o_{0_w}, \tau_0\right]$
Related sensor states	$\mathbf{s}_1 \cdots \mathbf{s}_q$
Where	$\mathbf{s}_j = \left[\hat{S}_j, \hat{u}_j, \hat{t}_j, \hat{o}_{j,1} \cdots \hat{o}_{j_w}, \hat{\tau}_j\right]$
Temporary register	T
SM Registers	S_{in}, v_{in}

2) a non-volatile session counter σ, which is incremented; and

3) a random secret γ, which is spontaneously generated inside the module.

5.1. Module Registers

Non-volatile storage inside the module is used to store three values—secret Q issued by the KDC, session counter σ, and module identity M.

Every module has the following reserved volatile registers:

The self-secret γ spontaneously generated inside the module remains privy

only to the module. This secret is used for computing self-certificates. A self-certificate is a "memorandum to self",—memoranda issued by the module for verification by itself at a later time, during the same session σ.

The register ξ_{sp} is the (160-bit) descriptor of the STCB system. The register ξ is the dynamic root of a Merkle tree. Register $\mathbf{p}$ contains various parameters regarding a peer module from which 1) sensor/state report has to be received, or 2) a report has to be sent.

The register $\mathbf{c}$ is reserved for storing a set of l constants to be used to evaluate some $\mathcal{U}_i$ corresponding to a sensor S_i. Values that reflect the current state $\mathbf{S}_i$ of S_i should be stored in register s_0. The states of q related sensors are stored in registers $s_1 \cdots s_q$. Register $\mathbf{r}$ is reserved for a freshly received sensor report from S_i.

For example, if at some instant of time,

1) the contents of the dynamic record $\mathbf{S}_i$ corresponding to sensor S_i is stored in location s_0, and

2) records corresponding to sensor $S_{i,1} \cdots S_{i,q}$ are stored on location $\mathbf{s}_1 \cdots \mathbf{s}_q$ respectively, and

3) values from a fresh report from sensor $\mathbf{S}_i$ are stored in register $\mathbf{r}$, then

$$S_i = \tilde{S}$$

$$\hat{S}_j = S_{i,j}$$

$$u_0 = v_i \ (\text{measurement of } S_i)$$

$$\hat{u}_j = v_{i,j} \ (\text{measurements of related sensors of } S_i)$$

$$\hat{o}_{j_x} = o_{(i,j)_x}, 1 \le x \le w \tag{15}$$

The SM register S_{in} indicates the sensor to which the module is bound (if the module is used as a SM) or is set to zero (if the module is used as a CM). If the module is used as an SM, the register v_{in} always contains the (dynamic) sensor measurement.

STCB modules have a built in hash function $h(\,)$ which is reused extensively for binary tree $\left(f_v(\,)\right)$ computations, computing shared secrets, computing message authentication codes, and self-certificates.

An in-built function $f_{exec}(\mathbf{A})$ in every module can execute a set of m instructions $\mathbf{A}$, where each instruction (chosen from the set $\mathcal{A}$) identifies a) an opcode (type of logical operation), b) one or more input operands (from the values stored in registers $\mathbf{s}_0$, $\mathbf{s}_1 \cdots \mathbf{s}_q$, $\mathbf{c}$, T), depending on the type of opcode; and c) the output operand ($o_{0_1} \cdots o_{0_w}$, or temporary register T).

5.2. Initializing Peer Parameters

The pairwise secret K that a module M shares with a peer module M' is used for computing message authentication codes (MAC) for outgoing messages to peer M', and for verifying incoming MACs from peer M'. Specifically, the secret used by M for computing outgoing MACs is $K_{out} = h(K, \sigma)$ where σ is

the session counter of M; consequently, the secret used for *verifying* MACs received from M' is $K_{in} = h(K, \sigma')$, where σ' is the session counter of M'.

STCB modules possess reserved registers to store the identity M' of a peer module (to which it needs to send a message, or from which it needs to receive a message), the session counter σ' of the peer, and MAC secrets K_{in} and K_{out}. Function $F_{peer}()$ exposed by a module can be invoked to populate values $M', \sigma', K_{in}, K_{out}$ related to a peer module M'.

$$
\begin{aligned}
&F_{peer}(I, P, s)\{ \\
&M' := I; \sigma' := s; K := h(Q, M') \oplus P; \\
&K_{in} := h(K, \sigma'); K_{out} := h(K, \sigma); \\
&\}
\end{aligned}
$$

To facilitate secure communications between two modules M_i and M_j, $F_{peer}(M_j, P_{ij}, \sigma_j)$ should be invoked on M_i, and $F_{peer}(M_i, 0, \sigma_i)$ should be invoked on M_j. M_i computes the pairwise secret using the public value P_{ij}; M_j computes the same value without using P_{ij} (or $P_{ij} = 0$ as XORing with 0 results in no change).

5.3. Self Certificates

Two types of certificates are computed by STCB modules—binary hash tree certificates, and offset certificates.

5.3.1. Binary Tree Certificates

A binary hash tree certificate is computed as

$$\rho_{bt} = h(x, x', y, y', \gamma) \tag{16}$$

Such a self-memoranda states that "x is a node in a binary hash tree with root y", and "if $x \to x'$ then $y \to y'$".

STCB modules expose a function $F_{mt}()$ which evaluates a sequence of hash operations $f_v()$, and output a binary tree certificate.

$$
\begin{aligned}
&F_{mt}(x, x', \mathbf{v}_x)\{ \\
&y := f_v(x, \mathbf{v}_x); y' := f_v(x', \mathbf{v}_x); \\
&\text{RETURN } \rho_{bt} := h(x, x', y, y', \gamma); \\
&\} \\
&F_{mtc}(x, x', y, y', z, z', \rho_1, \rho_2)\{ \\
&\text{IF } (\rho_1 \neq h(x, x', y, y', \gamma)) \text{ RETURN ERROR}; \\
&\text{IF } (\rho_2 \neq h(y, y', z, z', \gamma)) \text{ RETURN ERROR}; \\
&\text{RETURN } \rho_{bt} := h(x, x', z, z', \gamma); \\
&\}
\end{aligned}
$$

A function $F_{mtc}()$ concatenates two such certificates to create another certificate. Specifically, a certificate binding node x (and x') to an ancestor y (and y') and a certificate binding a node y (and y') to an ancestor z (and z') can be combined to a certificate binding node x (and x') to an ancestor z (and z').

The primary need for the function $F_{mtc}()$ is due to restrictions on the size of

inputs to module functions. Specifically, we can now place a hard limit on the size of the input $\mathbf{v}_x$—to (say) 8 hashes. For computing relationships between a node and the root of a tree with a million leaves (20 levels) three calls to $F_{mt}(\,)$ (to produce certificates binding 1) a level zero node to a level 8 node, 2) level 8 node to a level 16 node, and 3) a level 16 node to a level 20 node) and two calls $F_{mtc}(\,)$ (to combine the first two certificates, and combine the resulting certificate with the third certificate) can be used.

5.3.2. Offset Certificates

An *offset certificate* is computed as

$$\rho_{os} = h\left(M', \sigma, \sigma', os, \epsilon, \gamma\right) \tag{17}$$

and states that the module M (that issued the certificate) had performed a handshake within a duration ϵ with a module M', and had estimated the offset between their clocks to be os. The certificate also states that the handshake was performed when it's session counter was σ and the session counter of M' was σ'. The offset certificate is issued by a function $F_{hs}(\,)$ exposed by modules.

The function $F_{hs}(\,)$ can be invoked on pair of modules to perform a handshake, after which the initiator of the handshake obtains an estimate of the clock offset of the responder. Before $F_{hs}(\,)$ is invoked, $F_{peer}(\,)$ should be invoked on both nodes to set up respective peer identities, session counters, and secrets K_{in} and K_{out} to be used for incoming and outgoing MACs.

The function $F_{hs}(\,)$ has three inputs—a received MAC μ' (from peer M') with time stamp c', and a time-stamp $\tilde{c}$ that was previously sent to peer M' (which had triggered the response μ' from M'). The output of $F_{hs}(\,)$ is either a MAC intended for the peer or a self-MAC intended for itself, indicating the estimated offset for peer M'.

$F_{hs}(\,)$ is first invoked on the initiator with all inputs $\left(\mu', c', \tilde{c}\right)$ set to zero; the output of $F_{hs}(\,)$ is $\mu_1 = h\left(c_i^1, 0, \sigma_r, h\left(K, \sigma_i\right)\right)$ where c_i^1 and σ_i are the current clock-counter and the session counter of the initiator and σ_r is the session counter of the responder.

$F_{hs}(\,)$ is then invoked in the responder module with inputs $\left(\mu_1, c_i^1, 0\right)$. If the clock tick count of the responder is c_r, the output is $\mu_2 = h\left(c_r, c_i^1, \sigma_i, h\left(K, \sigma_r\right)\right)$.

$F_{hs}(\,)$ is then invoked on the initiator for the second time, at time c_i^2, with inputs $\left(\mu_2, c_r, c_i^1\right)$. The offset between the clock of the initiator and the responder can be estimated by the initiator to within the round-trip duration $\epsilon = c_i^2 - c_i^1$. The best estimate of the initiator is that, when the clock tick count of responder was c_r, the clock tick count of the initiator was $\left(c_i^1 + c_i^2\right)/2$, and thus, the best estimate of the offset is $os = \left(c_i^1 + c_i^2\right)/2 - c_r$.

The output of $F_{hs}(\,)$ in this case is the offset certificate. This certificate can be provide to the module at any time to convince the module that "the offset to M' was estimated as os with a tolerance of ϵ", and that "the offset to M' was estimated when the session counters of the modules were σ_i and σ_r". If

any of the two session counters had changed since the certificate was issued, the certificate becomes invalid.

$F_{hs}(\mu', \tilde{c}, c')\{$
IF $(\mu' = 0)$ // Send challenge
RETURN $h(c, 0, \sigma', K_{out})$; / /Sent as challenge
IF $(\mu' \neq h(c', \tilde{c}, \sigma, K_{in})$ RETURN ERROR;
IF $(\tilde{c} = 0)$ / /Respond to challenge
RETURN $h(c, c', \sigma', K_{out})$;
ELSE // Process response to estimate offset
$\epsilon := c - \tilde{c}; os := (c + \tilde{c})/2 - c'$;
RETURN $\rho_{os} := h(M', \sigma, \sigma', os, \epsilon, \gamma)$;
}

5.4. Initializing STCB Modules

Initializing an STCB module M implies initializing three internal registers reserved for values ξ_{sp}, ξ and S_{in}. Specifically, a module M can be initialized to participate in a deployment ξ_{sp} only if a binding record for module M can be demonstrated to be consistent with ξ_{sp}.

As ξ_0 is a node in a tree with root ξ_{sp}, **U** can use $F_{mt}(\,)$ to obtain a certificate

$$\rho = h(\xi_0, \xi_0, \xi_{sp}, \xi_{sp}, \gamma) \tag{18}$$

Similarly, as binding record $\mathbf{B}_i = [S_i, M, \epsilon, \theta]$ that exists in deployment tree is used to initialize the register S_{in} Now **U** can use $F_{mt}(\,)$ to obtain a certificate

$$\rho = h(v, v, \xi_{sp}, \xi_{sp}, \gamma) \tag{19}$$

Function $F_{init}(\,)$ can be used to initialize a module M as

1) a CM for a deployment ξ_{sp} or

2) as an SM for a sensor S in deployment ξ_{sp}.

To initialize a module as a CM for the deployment, the inputs (ξ_1, ξ_2, ρ) to $F_{init}(\,)$ are such that ρ is a binary tree certificate relating a node $\xi_0 = \xi_1$ and root $\xi_2 = \xi_{sp}$ (inputs S' and ϵ' are set to 0).

To initialize the module as a SM the certificate, inputs S' and ϵ are non-zero. The binary tree certificate should relate $\xi_1 = h(S', M, \epsilon', 0)$ and $\xi_2 = \xi_{sp}$ to prove to the module that "in an STCB system with descriptor ξ_{sp}, the module M (which is being initialized) is authorized to report measurements corresponding to sensor S'". Accordingly, the register S_{in} in the module M is set to S'.

For a module M_j initialized as a SM for a sensor S_k, the output of the sensor S_k is physically connected to module M_j using a tamper-evident seal by the deployer. The physical connection ensures that the sensor measurement v_k is always available in the register v_{in} of the SM. measurement Later (during regular operation) module M_j cannot be initialized to act as a SM for any

other sensor $S' \neq S_k$, as no record binding S' to M_j can be demonstrated to be a part of the tree with root ξ_{sp}.

$$
\begin{aligned}
&F_{init}\left(\xi_0, r, S, \epsilon, \rho_1, \rho_2\right)\{\\
&\text{IF } \left(\rho_1 \neq h\left(\xi_0, \xi_0, r, r, \gamma\right)\right) \text{ RETURN ERROR;}\\
&x := h\left(S, M, \epsilon, 0\right);\\
&\text{IF } \left(\rho_2 \neq h\left(x, x, r, r, \gamma\right)\right) \text{ RETURN ERROR;}\\
&\xi_{sp} = r; \xi := \xi_0; S_{in} = S'; \text{ RETURN;}\\
\}&
\end{aligned}
$$

During regular operation, any dynamic sensor record can be loaded on to any register $\mathbf{s}_0$ or $\mathbf{s}_1 \cdots \mathbf{s}_q$ using function $F_{ld}(\,)$. A record $\mathbf{s}$ provided as input is simply loaded onto register $\mathbf{s}_j$ where j is the index specified. Specifically, the record is loaded only if the inputs ρ and $h(\mathbf{s})$ are consistent with dynamic root ξ.

5.5. Sensor and State Reports

In an STCB deployment with SMs $M_1 \cdots M_k$, and STCB module M_0, the state reports are made available to a module M_r associated with the stake holder. In general, M_r can be seen as an STCB module associated with a different STCB system at a higher level of hierarchy.

The handshake sequence (which involves two calls to $F_{hs}(\,)$ in the initiator module and one $F_{hs}(\,)$ call in the responder module) the handshake sequence is orchestrated by $\mathbf{U}$ between

1) k responders $M_1 \cdots M_k$, with M_0 (as initiator)

2) M_r as initiator and M_0 as responder.

After the $k+1$ hand-shake sequences have been completed, k self-certificates of type OS ρ_{os} are created by M_0—one corresponding to each SM, and one self-certificate is created by M_r. Such certificates indicate both the estimated offset *os*, and the maximum error ϵ in the estimate *os*.

Now modules are ready to exchange authenticated messages. More specifically, SMs send authenticated and time-stamped sensor reports to M_0, and STCB module M_0 can send state reports to M_r. Such messages exchanged between modules are computed as

$$\mu = h\left(S, v, t, c, \xi_{sp}, K_{out}\right) \tag{20}$$

where for sensor reports (from SMs to M_0)

1) $S = S_{in}$ is the identity of a sensor that is bound to the module that created the report, and $v = v_{in}$;

2) $t = c$ is the current clock tick count of the creator of the report;

3) ξ_{sp} is the value used to initialize the module.

A report from M_0 to stake holders is made in accordance with a record $\left[S_r, M_r, S_i, 1 \leq l \leq w\right]$. Such a report can be created for the benefit of M_r only if the state of sensor S_i is loaded in register s_0 (or $S_0 = S_i$), to report the value

e_0^l and time τ_0. In such a report $S = S_r$ is a label assigned to the report, $v = e_0^l$, and $t = \tau_t$ is the time associated with $v = e_0^l$ (or $t \neq c$).

When $F_{snd}()$ is invoked in a SM (with register $S_{in} \neq 0$) the output is $\mu = h\left(S_{in}, v_{in}, c, c, \xi_{sp}, K_{out}\right)$. When invoked on an STCB module, this function should be able to verify the existence of an appropriate reporting record **R** that authorizes the module to report one of the w values $e_o^1 \cdots e_o^w$ stored in register $\mathbf{s}_0$.

$$
\begin{aligned}
&F_{snd}\left(S', j, \rho\right)\{\\
&\text{IF } \left(S \neq 0\right)\\
&\quad \text{RETURN } c, \mu = h\left(S_{in}, v_{in}, c, c, \xi_{sp}, K_{out}\right);\\
&y = h\left(S', M', S_0, j\right);\\
&\text{IF } \left(\rho = h\left(y, y, \xi_{sp}, \xi_{sp}, \gamma\right)\right)\\
&\quad \text{RETURN } c, \mu = h\left(S', o_{0_j}, \tau_0, c, \xi_{sp}, K_{out}\right);\\
&\quad \text{ELSE RETURN ERROR;}\\
&\}
\end{aligned}
$$

Corresponding to a state report for a "sensor" S_r, while the STCB module M_0 that generates the report is initialized with the ξ_{sp}, the stake-holder module M_r—which in general can be seen as the STCB module associated with a foreign STCB system may be initialized with a different descriptor ξ'_{sp}. For the module M_r to accept the report from a foreign system, the deployment tree in the foreign system with root ξ'_{sp} should include a binding record

$$\mathbf{B} = \left[S_r, M_0, \epsilon, \theta = \xi_{sp} \neq 0\right] \tag{21}$$

Function $F_{rcv}()$ can be invoked in the STCB module to provide a fresh sensor report to the module. Specifically, before a sensor report from a module M_j can be provided to a module, the function $F_{peer}()$ should be invoked on the receiving module to set $M' = M_j$. Now,

1) inputs S_i, ϵ', θ to $F_{rcv}()$ are used to compute the leaf hash $x = h\left(S_i, M', \epsilon', \theta\right)$ of a binding record $\mathbf{B}_i$.

2) input ρ is used to confirm that $x = h\left(S_i, M', \epsilon', \theta\right)$ (the hash of the binding record) is a node in a tree with root ξ_{sp}.

3) inputs ϵ, os and ρ_{os} are used to verify that $\rho_{os} = h\left(M', \sigma, \sigma', os, \epsilon, \gamma\right)$.

4) and inputs v_i', t', c' and μ' are used to verify that $\mu' = h\left(S_i, v_i', t', c', y, K_{in}\right)$ where $y = \xi_{sp}$ if $\theta = 0$, or $y = \theta$ if $\theta \neq 0$.

The function returns error if $\epsilon > \epsilon'$, or on failure of verification of inputs ρ_{nv} or ρ_{os} or μ'.

Ultimately, the purpose of function $F_{rcv}()$ is to receive two values v_i' and time t_i' corresponding to a sensor S_i where $t_i' = t + os$ is the offset corrected time associated with v_i'. Values S_i, v_i' and t_i' are then stored in a reserved register **R** for further processing.

$F_{rcv}(S_i, \theta, \mathrm{n}', \rho_{bt}, os, \mathrm{n}, \rho_{os}, v_{i'}, \rho, t', c', \mu')\{$
IF $(\mathrm{n} > \mathrm{n}')$ RETURN ERROR;
IF $\left(\rho_{os} \neq h(M', \sigma, \sigma', os, \mathrm{n}, \gamma)\right)$ RETURN ERROR;
$y = (\theta = 0)?\xi_{sp} : \theta;$
$x = h(S_i, M', \mathrm{n}', y);$
IF $\left(\rho_{bt} \neq h(x, x, \xi_{sp}, \xi_{sp}, \gamma)\right)$ RETURN ERROR;
IF $\left(\mu' \neq h(S_i, v_{i'}, t', c', y, K_{in})\right)$ RETURN ERROR;
$\mathbf{r} := (S = S_i, v = v_{i'}, t = t' + os);$
}

5.6. Sensor Updates and Incremental State Evaluations

Values **r** in a fresh sensor report from S_i are part of the inputs used to evaluate $\mathcal{U}_i$. Evaluation of $\mathcal{U}_i$ results in modifications to the state $\mathbf{S}_i$ of sensor S_i. Before $\mathcal{U}_i$ can be evaluated it should be ensured that appropriate values are loaded on to registers **r**, $\mathbf{s}_0$, $\mathbf{s}_0 \cdots \mathbf{s}_q$, and **c**.

Recall that register **r** is populated by function $F_{rcv}()$. Function $F_{ld}()$ can be used to load the dynamic values **S** associated with any sensor on to any of the $q+1$ registers $\mathbf{s}_0$, $\mathbf{s}_0 \cdots \mathbf{s}_q$.

$F_{ld}(\mathbf{s}', j, \rho)\{$
$x := h(\mathbf{s}');$
IF $\left(\rho \neq h(x, x, \xi, \xi, \gamma)\right)$ RETURN ERROR;
// Record is a leaf in the dynamic tree
$\mathbf{s}_j := \mathbf{S};$
}
$F_{upd}(\mathbf{C}, \lambda, \rho_c, \mathbf{A}, \rho, \xi', \rho_{upd}, S_i')\{$
IF $(S_0 \neq S)$ RETURN ERROR;
$tmp = h\left(\left[S_0, R_1 \mathrm{n} R_q, h(\mathbf{A}), \lambda, S_i'\right]\right);$// hash of a design record
// tmp should be a node in the static tree with root ξ_{sp}
IF $\left(\rho \neq h(tmp, tmp, \xi_{sp}, \xi_{sp}, \gamma)\right)$ RETURN ERROR;
$tmp = h(\mathbf{C});$// hash of constant record
// should be a node in the constant tree with root λ
IF $\left(\rho_c \neq h(tmp, tmp, \lambda, \lambda, \gamma)\right)$ RETURN ERROR;
$tmp := h(\mathbf{s}_0);$// Record before evaluation of U
$\tau_0 := \min(t, \tau_1' \mathrm{n} \tau_q');$
$T := v; v_0 := v; t_0 := t;$
IF $\left((f_{\mathbf{A}}() = \mathrm{ERROR}) \vee \left(\rho_{upd} \neq h(tmp, h(\mathbf{s}_0), \xi, \xi', \gamma)\right)\right)$
CLEAR-ALL AND RETURN;
$\xi := \xi';$
IF $(S_i' \neq 0)$ $S := S_{i'};$
}

The main purpose of $F_{upd}(\,)$ is to evaluate $\mathbf{U}_i$ corresponding to a sensor S_i. For this purpose, $F_{upd}(\,)$ verifies that all inputs required to evaluate $\mathbf{U}_i$ are available. If a design record corresponding to sensor S_i is

$$\mathbf{G}_i = \left[S_i, S_{i_1} \cdots S_{i_q}, \alpha_i, \lambda_i, S_i'\right] \tag{22}$$

it should be ensured that the current state of sensor S_i is loaded onto register $\mathbf{s}_0$, and the states of related sensors $S_{i_1} \cdots S_{i_q}$ are loaded on to registers $\mathbf{s}_1 \cdots \mathbf{s}_q$ respectively. Other values required to evaluate $\mathcal{U}_i$ are provided as inputs to $F_{upd}(\,)$.

Specifically,

1) the constants $\mathbf{C}$ should be such that $x = h(\mathbf{C})$ is node in a tree with root λ_i. This can be demonstrated by providing a certificate $\rho_c = h(x, x, \lambda_i, \lambda_i, \gamma)$.

2) the instructions $\mathbf{A}$ should be such that $h(\mathbf{A}) = \alpha_i$.

3) λ_i and α_i should exist in the design record $\left[S_0, S_{i_1} \cdots S_{i_q}, \alpha_i, \lambda_i\right]$. More specifically, $y = h\left(S_0, S_{i_1} \cdots S_{i_q}, \alpha_i, \lambda_i, N_i\right)$ should be a node in the tree with root ξ_{sp}. This can be demonstrated by providing a certificate $\rho = h\left(y, y, \xi_{sp}, \xi_{sp}, \gamma\right)$.

4) the values $S_0 \in \mathbf{s}$ (the identity of the sensor to be updated) and $\tilde{S} \in \mathbf{r}$ (the sensor from which a fresh report has been received) should be the same.

During execution of the algorithm $\mathbf{A}$, in situations where many options exist for choosing the set of constants $\mathbf{C}$ consistent with λ it is the responsibility of $\mathbf{U}$ to choose the correct set of constants that satisfy the range of the independent variable(s). On successful evaluation of the algorithm $\mathbf{A}$ the status of sensor S_0 in register $\mathbf{s}_0$ will be updated. If x is the hash of register $\mathbf{s}_0$ *before* the update, and if x' is the hash of register $\mathbf{s}_0$ *after* the update, then a new root ξ' and a certificate ρ should be provided as input satisfying

$$\rho = h(x, x', \xi, \xi', \gamma) \tag{23}$$

6. STCB Protocol

The STCB protocol can be seen as the actions to be performed by the untrusted STCB manager $\mathbf{U}$ to submit sensor reports from SMs and CMs of foreign deployments to the CM M_0 of the STCB deployment, obtain state-reports from M_0, and submit such reports to stake holders (or CMs of foreign deployments).

6.1. Generation of Offset Certificates

The first step in the operation of an STCB deployment is that of performing handshakes between various modules to obtain offset certificates. In general one offset certificate will be generated for every module specified in the binding records of the deployment.

For a STCB system with n sensors (real, state-report and synthetic) the total number of binding records is $n'+1$, where $n-n'$ is the number of synthetic records: no binding record exists for synthetic records, and one binding record is for the CM M_0. The total number of distinct modules in general will be $n'' \le n'$. Specifically, while there will exist one module corresponding to every real sensor,

as a single CM may report multiple states, the number of state-report sensors may be greater that the number of foreign CMs that provide state reports.

A total of n'' hand shake sequences will be invoked to obtain n'' offset certificates. Recall that each such sequence begins with a challenge from the CM M_0 generated using $F_{hs}(\,)$ to which a response is generated by invoking $F_{hs}(\,)$ in the responder module, and finally, the response is submitted to the CM to generate the certificate. If any SM is rebooted, the offset certificate corresponding to the SM has to be regenerated. If the CM is rebooted, all offset certificates will need to be regenerated.

Before $F_{hs}(\,)$ is invoked, $F_{peer}(\,)$ should be invoked on both modules to set up the MAC secrets K_{in} and K_{out}.

6.2. Generating Static Binary Tree Certificates

The second step is for **U** to obtain binary tree certificates corresponding to all leaves of the static tree with root ξ_{sp}. Specifically, as **U** maintains the tree with root ξ_{sp}, **U** can readily provide the complementary nodes for any leaf in the static tree to function $F_{bt}(\,)$, and obtain certificates of the form

$$\rho_s = h\left(x_s, x_s, \xi_{sp}, \xi_{sp}, \gamma\right) \tag{24}$$

where x_s is the cryptographic hash of the s^{th} leaf of the static tree. The total number of static leaves is $(n+1)+(n'+1)+m$ where

1) n is the number of design records: one for each sensor (real, state-report, or synthetic);

2) one leaf corresponds to the value ξ_0 in the design tree;

3) $n'+1$ is the number of binding records (including one for the CM M_0); and

4) m is the number of reporting records.

6.3. Initialization and Regular Operation

The third step is the initialization of STCB modules to operate in deployment ξ_{sp}—by invoking $F_{init}(\,)$ on each STCB module. For initializing the modules the two binary tree certificates are required: one linking ξ_0 to ξ_{sp}, and one linking a binding record for the module with the static root ξ_{sp}.

On completion of the three steps, the STCB manager maintains a dynamic Merkle tree with leaves as sensor records. As the initial values of such records are specified by the designer, the root of the tree should be the same as the initial value $\xi = \xi_0$ stored by the STCB module.

Once all STCB modules have been initialized, sensor reports from SMs (or CMs of foreign deployments) are submitted to the CM as and when they are received. In general, not all sensors may report at the same frequency.

As all SMs send messages only to the CM, $F_{peer}(\,)$ needs to be invoked only once on each SM (which was already performed before invoking $F_{hs}(\,)$ to generate offset certificates).

To create a sensor report from an SM, **U** will invoke $F_{snd}(\,)$ on an SM. Some

sensor reports corresponding to state reports from foreign CMs may also be received by $\mathbf{U}$.

Once a sensor report for some sensor S_i has been received, $\mathbf{U}$ is expected to make appropriate modifications to the sensor state $\mathbf{S}_i$, and accordingly, modify the Merkle tree maintained by $\mathbf{U}$. Let $\xi \to \xi'$ be the change in the root of the dynamic tree, corresponding to the modification $\mathbf{S}_i \to \mathbf{S}_i'$ triggered by the received sensor report. If $h(\mathbf{S}_i) = x$ and $h(\mathbf{S}_i') = x'$, $\mathbf{U}$ can readily determine complementary nodes required to obtain the certificate

$$\rho_{bt} = h(x, x', \xi, \xi', \gamma) \tag{25}$$

The STCB manager invokes $F_{peer}()$ followed by $F_{rcv}()$ to submit the report to the CM. Recall that inputs to $F_{rcv}()$ include a MAC received from an SM/CM, a binding record along with a certificate linking the record to ξ_{sp}, and an offset certificate corresponding to the reporting module.

Corresponding to the sensor state record for sensor S_i and q related sensors $\mathbf{U}$ invokes $F_{bt}()$ to obtain $q+1$ certificates binding the sensor state records to dynamic root ξ. Following this, the STCB manager uses $F_{ld}()$ up to $q+1$ times to load the 1) previous sensor state $\mathbf{S}_i$ on to register $\mathbf{s}_0$ and 2) the states of related sensors on to registers $\mathbf{s}_1 \cdots \mathbf{s}_q$.

Finally, the STCB manager invokes $F_{upd}()$. Recall that the inputs to $F_{upd}()$ include ρ_{bt} obtained as per Equation (25), a certificate binding design record $\mathbf{G}_i$ to STCB descriptor ξ_{sp}, the set of m instructions $\mathbf{A}_i$, a set of l constants, and a certificate binding the constants to a value λ in the design record. Only if the modification $\mathbf{S}_i \to \mathbf{S}_i'$ computed by the CM is exactly the same as that performed by the STCB manager $\mathbf{U}$ will the update be successful in modifying the dynamic root ξ stored inside the CM to ξ'.

At any time the STCB manager can invoke a $F_{upd}()$ to load a state record consistent with ξ on to register $\mathbf{s}_0$. Now $F_{snd}()$ can be invoked to create a state-report. Note that when $F_{snd}()$ is invoked on a CM a certificate binding a reporting record to the static root should be provided as input.

7. Conclusions

The ever growing complexity of systems poses a severe threat—the possibility of hard-to-detect hidden functionality that can be exploited to take control of the system. Current strategies for securing SCADA systems are predominantly focused on development of suitable intrusion detection systems. Such security measures ignore the very real possibility of hidden functionality in the intrusion detection systems themselves.

In the proposed approach to secure SCADA systems only STCB modules are trusted to provide the assurance that "no attack will go undetected". The proposed approach involves three stages—a design process carried out by a designer with good domain knowledge, a deployment process carried out by a security professional, and regular operation of the STCB system. The designer and deployer together specify a concise description ξ_{sp} of the system. During regular

operation, an STCB module reports the state of a system identified as ξ_{sp} to stake-holders.

Some of the important features of the STCB approach that make it well suited for any SCADA system of any size include

1) the ability to support hierarchical deployments;

2) the ability to support any type of function $\mathcal{U}$ —if necessary through the use of 1-D, or 2-D look up tables (which are also specified as leaves of the design tree); and

3) the ability to specify synthetic sensors.

Such features are intended to enable the use of STCB modules for securing any SCADA system.

The first pre-requisite for deployment of STCB based security solutions is the actual availability of STCB modules/chips. Towards this end, the work that has been performed is a small first step—identification of a functional specification of such chips. In arriving at an appropriate functional specification, some our main goals have been

1) reducing computational and memory requirement inside STCB chips;

2) reducing interface complexity (size of inputs and outputs to/from the STCB chips); and

3) simplifying the STCB protocol—which is a specification of a sequence of interactions with the STCB modules—to realize the desired assurances.

The proposed functional specification (for STCB modules) is merely *a* specification, and not *the* specification. Just as there are numerous ways to realize a block-cipher or a hash function, there are numerous ways to arrive at a "set of STCB functions" (which can be leveraged to realize the same assurances). The functional specification in this paper is however the first of its kind.

Acknowledgements

This research was funded by the Department of Homeland Security (DHS)-sponsored Southeast Region Research Initiative (SERRI) at the Department of Energy's Oak Ridge National Laboratory.

References

[1] Matrosov, A., Rodionov, E., Harley, D. and Malcho, J. (2010) Stuxnet under the Microscope. https://www.esetnod32.ru/company/viruslab/analytics/doc/Stuxnet_Under_the_Microscope.pdf

[2] Turk and Robert, J. (2005) Cyber Incidents Involving Control Systems. Idaho National Engineering and Environmental Laboratory, Idaho Falls.

[3] Lynch, L.E., Comey, J.B. and Carlin, J.P. (2016) Manhattan U.S. Attorney Announces Charges Against Seven Iranians for Conducting Coordinated Campaign of Cyber Attacks against U.S. Financial Sector on Behalf of Islamic Revolutionary Guard Corps-Sponsored Entities. News, Southern District of New York.

[4] Eric, A. and Jim, F. (2015) Ukraine Utility Cyber Attack Wider than Reported: Ex-

perts.
http://www.reuters.com/article/us-ukraine-crisis-malware-idUSKBN0UI23S20160104

[5] McWhorter, D. (2013) Mandiant Exposing APT1—One of China's Cyber Espionage Units & Releases 3,000 Indicators.

[6] Reid, W. (2013) Spear Phishing Attempt. Digital Bond.
http://www.digitalbond.com/blog/2012/06/07/spear-phishing-attempt/

[7] Gorman, S. (2013) Chinese Hackers Suspected in Long-Term Nortel Breach. *The Wall Street Journal*.

[8] Fidler and David, P. (2011) Was Stuxnet an Act of War? Decoding a Cyberattack. *IEEE Security & Privacy*, **9**, 56-59. https://doi.org/10.1109/MSP.2011.96

[9] Sood, A. and Enbody, R. (2013) Targeted Cyber Attacks—A Superset of Advanced Persistent Threats. *IEEE Security & Privacy*, **11**, 54-61.

[10] Ramkumar, M. (2016) Cybersecurity: It's All about the Assumptions. *National Cyber Summit* (*NCS*), Huntsville, 8-9 June 2016.

[11] Weaver, N., Paxson, V., Staniford, S. and Cunningham, R. (2003) A Taxonomy of Computer Worms. *Proceedings of the* 2003 *ACM Workshop on Rapid Malcode*, Washington DC, 27 October 2003, 11-18. https://doi.org/10.1145/948187.948190

[12] Reid, W. (2011) Cyber Attacks on Texas Utility.
http://www.washingtontimes.com/news/2011/nov/18/hackers-apparently-based-in-russia-attacked-a-publ

[13] Moore, D., Paxson, V., Savage, S., Shannon, C., Staniford, S. and Weaver, N. (2003) Inside the Slammer Worm. *IEEE Security and Privacy*, **99**, 33-39.
https://doi.org/10.1109/MSECP.2003.1219056

[14] Lampson, B., Abadi, M., Burrows, M. and Wobber, E. (1992) Authentication in Distributed Systems: Theory and Practice. *ACM Transactions on Computer Systems* (*TOCS*), **8**, 18-36. https://doi.org/10.1145/138873.138874

[15] Wright, A., Kinast, J. and McCarty, J. (2004) Low-Latency Cryptographic Protection for SCADA Communications. Applied Cryptography and Network Security. *Lecture Notes in Computer Science*, Vol. 3089, Springer, Berlin, 263-277.
https://doi.org/10.1007/978-3-540-24852-1_19

[16] Tsang, P. and Smith, S.W. (2007) YASIR: A Low-Latency, High-Integrity Security Retrofit for Legacy SCADA Systems. *Proceedings of the Ifip Tc* 11 23rd *International Information Security Conference*, Vol. 278, Springer, Boston, 445-459.
https://doi.org/10.1007/978-0-387-09699-5_29

[17] Wang, Y. and Chu, B.-T. (2012) sSCADA: Securing SCADA Infrastructure Communications. *International Journal of Communication Networks and Distributed Systems*, **6**, 59-78. https://doi.org/10.1504/IJCNDS.2011.037328

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

[19] Hieb, J., Graham, J. and Patel, S. (2007) Security Enhancements for Distributed Control Systems, Mathematics and Its Applications. 2nd Edition, Critical Infrastructure Protection, Springer, Berlin, 133-146.
https://doi.org/10.1007/978-0-387-75462-8_10

[20] Shahzad, A. and Musa, S. (2012) Cryptography and Authentication Placement to

Provide Secure Channel for SCADA Communication. *International Journal of Security*, **6**, 28.

[21] Patel, A., Joaquim Jr., C. and Pedersen, J. (2013) An Intelligent Collaborative Intrusion Detection and Prevention System for Smart Grid Environments. *Computer Standards & Interfaces*. https://doi.org/10.1016/j.csi.2013.01.003

[22] Berthier, R. and Sanders, W.H. (2010) Intrusion Detection for Advanced Metering Infrastructures: Requirements and Architectural Directions. 2010 Fi*rst IEEE International Conference on Smart Grid Communications* (*SmartGridComm*), Gaithersburg, 4-6 October 2010, 350-355. https://doi.org/10.1109/SMARTGRID.2010.5622068

[23] Zhu, B., Sastry, S. and Fefferman, C. (2010) SCADA-Specific Intrusion Detection/Prevention Systems: A Survey and Taxonomy. *Proceedings of the* 1*st Workshop on Secure Control Systems* (*SCS*), Stockholm, 12 April 2010, 34.

[24] Merkle, R.C. (1980) Protocols for Public Key Cryptosystems. *IEEE Symposium on Security and Privacy*, Oakland, 14-16 April 1980, 122. https://doi.org/10.1109/SP.1980.10006

[25] Ramkumar, M. (2008) On the Scalability of an Efficient Nonscalable Key Distribution Scheme. 2008 *International Symposium on a World of Wireless*, *Mobile and Multimedia Networks*, Newport Beach, 23-26 June 2008, 1-6.