Formally Verified ZTA Requirements for OT/ICS Environments with Isabelle/HOL

Preprint

Yakoub Nemouchi, Sriharsha Etigowni, Alexander Zolan, and Richard Macwan

Presented at 15th International Conference on Verified Software: Theories, Tools, and Experiments
Ames, Iowa
October 23, 2023

NREL is a national laboratory of the U.S. Department of Energy
Office of Energy Efficiency & Renewable Energy
Operated by the Alliance for Sustainable Energy, LLC

This report is available at no cost from the National Renewable Energy Laboratory (NREL) at www.nrel.gov/publications.

Contract No. DE-AC36-08GO28308

Conference Paper
NREL/CP-5R00-86154
November 2023

Formally Verified ZTA Requirements for OT/ICS Environments with Isabelle/HOL

Preprint

Yakoub Nemouchi, Sriharsha Etigowni, Alexander Zolan, and Richard Macwan

Suggested Citation
Nemouchi, Yakoub, Sriharsha Etigowni, Alexander Zolan, and Richard Macwan. 2023. *Formally Verified ZTA Requirements for OT/ICS Environments with Isabelle/HOL: Preprint*. Golden, CO: National Renewable Energy Laboratory. NREL/CP-5R00-86154. https://www.nrel.gov/docs/fy24osti/86154.pdf.

Conference Paper
NREL/CP-5R00-86154
November 2023

Formally verified ZTA requirements for OT/ICS environments with Isabelle/HOL

Yakoub Nemouchi[0000−0001−7498−6691], Sriharsha Etigowni, Alexander Zolan[0000−0003−2601−7604], and Richard Macwan

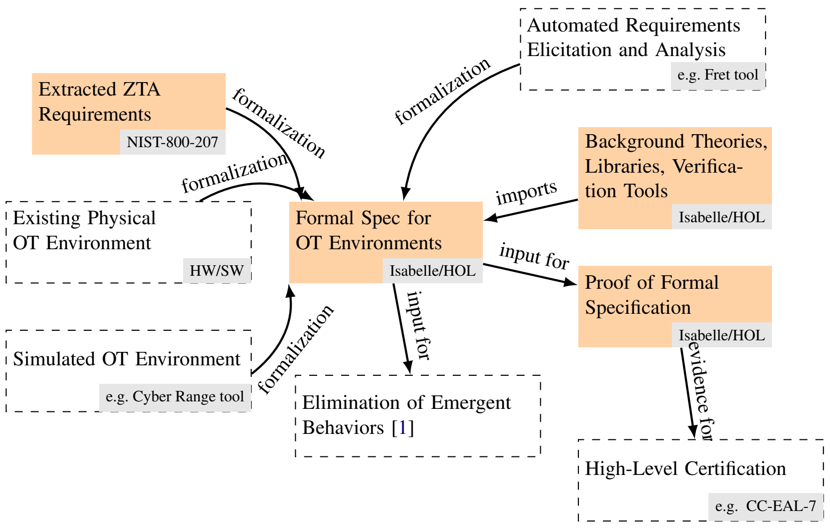
National Renewable Energy Laboratory
firstname.lastname@nrel.gov

Abstract. The clean energy transformation led to the integration of distributed energy resources on a top of the grid, and so a substantial increase in the complexity of power grids infrastructure and the underlying operational technology environment. Operational technology environments are becoming a system of systems, integrating heterogeneous devices which are software/hardware intensive, have ever increasing demands to exploit advances in commodity of software/hardware infrastructures, and this for good reasons – improving energy systems requirements such as cybersecurity and resilience. In such a setting, system requirements at different levels mix, thus undesirable outcomes will surely happen. The use of formal methods will remove ambiguity, increase automation and provide high levels of assurance and reliability. In this paper, we contribute a methodology and a framework for the system level verification of zero trust architecture requirements in operational technology environments. We define a formal specification for the core functionalities of operational technology environments, the corresponding invariants, and security proofs. Of particular note is our modular approach for the formal verification of asynchronous interactions in operational technology environments. The formal specification and the proofs have been mechanized using the interactive theorem proving environment Isabelle/HOL.

Keywords: formal methods, Isabelle/HOL, OT security, microgrids.

1 Introduction

Operational technology (OT) environments are cyber-physical systems (CPSs) used to integrate, monitor, and enforce control actions in industrial control systems (ICSs) [43]. OT environments include devices such as supervisory control and data acquisition (SCADA) systems, programmable logic controllers (PLCs), intelligent electronic devices (IEDs), and remote terminal units (RTUs) [56], each of which can be connected to a distributed control network infrastructure featuring Lightweight Directory Access Protocol (LDAP) servers, routers, and firewalls [20]. The ongoing transition to renewable energy systems provides a new dimension of security risks to the OT landscape [48]. The next generation of OT environments will integrate distributed energy resources (DERs), which are broadly defined to include (i) microturbines and other combustion technologies [45], (ii) wind plants [35], (iii) solar energy plants [57], (iv) smart buildings [49], (v) electrical vehicles (EVs) [23], (vi) fuel cells [46], and (vii) other kinds of hybrid systems [41]. A microgrid configuration can comprise one or more of these items (i)–(vi) [28]. These highly critical and software-intensive CPSs [42, 55] contain OT/ICS devices that are interconnected with each other, with the internet, with the



Automated Requirements Elicitation and Analysis
e.g. Fret tool
Extracted ZTA Requirements
NIST-800-207
formalization
formalization
formalization
Background Theories, Libraries, Verification Tools
Isabelle/HOL
imports
Existing Physical OT Environment
HW/SW
Formal Spec for OT Environments
Isabelle/HOL
input for
Proof of Formal Specification
Isabelle/HOL
Simulated OT Environment
e.g. Cyber Range tool
formalization
input for
evidence for
Elimination of Emergent Behaviors [1]
High-Level Certification
e.g. CC-EAL-7

Fig. 1: Formal verification framework for OT security.

environment, with the energy infrastructure, and with other critical infrastructure and key resources [31], which can lead to a large collection of entry points for cyberattacks [22]. CPSs can create domains of *mixed criticalities* [5], wherein system requirements of different levels mix. In such a setting, undesirable outcomes are more likely to happen, highlighting a need to increase the level of assurance and reliability for OT environments. Our proposed solution combines the use of known security architectures, such as the zero trust architecture (ZTA) [27], with the use of formal methods (FMs) [29], the latter of which provides assurance evidence with an absolute guarantee that an OT environment meets ZTA requirements.

The goal of this work is to demonstrate that developing dependably secure OT environments to the level of trustworthiness required by ZTA is possible. Our solution consist of defining and proving correctness (Section 5) for OT environments under requirements imposed by ZTA. This requires a formal specification of the core functionalities of OT environments, a formal specification of ZTA tenets, and a semantics framework enforcing rigorous and modular reasoning, i.e., allowing the verification of ZTA tenets for individual devices and then the composition of the verification results for the overall OT environment. The seven tenets of ZTA that we adopt are described informally using natural language in the National Institute of Standards and Technology (NIST) Special Publication 800-207 [44], in which each tenet addresses a specific security requirement. For example, the second tenet requires the implementation of access control policies to preserve data integrity and confidentiality, regardless of the physical location of the client accessing the network of the OT environment.

Vision. Our vision, illustrated in Figure 1, is a back end for system-level formal verification of ZTA requirements in OT environments. The colored boxes are contributions of this paper, and the white dashed boxes are use cases for future work. We use the interactive theorem proving (ITP) environment Isabelle/HOL [37] to define an array of formal verification tools, background theories, and libraries, that we leverage to specify and verify behaviors and security properties of OT environments at the system level. The

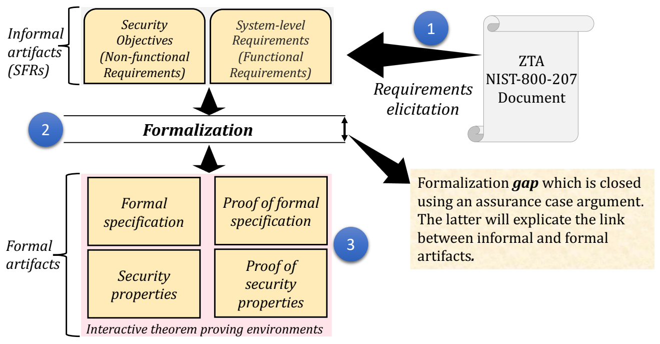

Informal artifacts (SFRs)
Security Objectives (Non-functional Requirements)
System-level Requirements (Functional Requirements)
1
Requirements elicitation
ZTA NIST-800-207 Document
2
Formalization
Formalization gap which is closed using an assurance case argument. The latter will explicate the link between informal and formal artifacts.
Formal artifacts
Formal specification
Proof of formal specification
Security properties
Proof of security properties
3
Interactive theorem proving environments

Fig. 2: Approaching OT security with verification based on deductive proofs.

workflow starts by manually extracting ZTA requirements (in natural language) from NIST 800-207. We then determine the core functionalities of OT environments to be formalized and verified together with the extracted requirements.

Contributions. The main contribution of this paper is a methodology for system level formal verification of ZTA requirements in OT environments (Section 2). The application of our methodology to OT environments (see Section 3 for details on the microgrid configuration we are using as a case study) led to a formal framework as an additional contribution with the following results: (1) the extraction of a set of security functional requirements (SFRs) (e.g., see **SFR 1** in Section 3, §7) for microgrids based on ZTA tenets listed in NIST 800-207; (2) the use of Isabelle/HOL to define small step semantics (see Table 1), which formally describes control actions, i.e., critical control actions for both device-level and system-level interactions in microgrid configurations; (3) the use of Isabelle/HOL to define big step semantics (see Table 2) to formally describe system-level and device-level behaviors, e.g., the behavior of the OT environment as a whole; (4) the definition of data models to describe properties of the state space of OT environments (see subsection 4.2); (5) the definition of correctness for OT environments (see subsection 5.1); (6) the formalization of ZTA tenets as security properties on top of Isabelle/HOL (see subsection 4.6); (7) machine-checked proofs, including the proofs of the well-formedness conditions and the proofs of the state invariant for each individual device; and (8) machine-checked proofs of the security properties that provide assurance evidence that the OT environment meets the security objectives of ZTA.

2 Approaching OT security with ZTA and deductive proofs

Approach. Figure 2 describes our approach. Step (1) is to manually extract SFRs (e.g., **SFR 1**) from ZTA tenets defined in NIST-800-207, the report in which ZTA standards are specified in natural language. Because the content of NIST-800-207 is broadly defined and intended for information technology (IT) environments, this step also includes the identification of the core functionalities of the OT environment (system-level requirements for verifiable ZTA in microgrids). The extraction of SFRs was done manually and involved collaborations with subject matter experts in power systems engineering, systems security engineering, and formal methods engineering[1]. The same requirement elicitation process led to the choice of a system architecture inspired by the *Enclave*

[1] All these roles are fulfilled by the authors of this paper.

Gateway Model found in the same NIST-800-207 standard (see Figure 3b) as an architecture model for verifiable ZTA in microgrid configurations. The main output from this step is a collection of SFRs for the security of microgrids that follow ZTA. An example of SFRs would be **SFR 1** defined in Section 3. Once the SFRs are defined, we then extract two artifacts. The first artifact consists of system-level requirements (functional requirements), i.e., core functionalities of OT environments that allow us to implement a secure microgrid following the architecture of the Enclave Gateway Model (see Section 3, §2–6). The second artifact consists of a list of ZTA security objectives for microgrids.

Step (2) is *formalization* (see Figure 2), i.e., translation of informal artifacts (e.g., the extracted SFRs which are written in natural language) to formal artifacts (e.g., SFRs written as a statement in logic) to allow verification based on deductive proofs. This formalization step is important because it removes ambiguity, inconsistency, incompleteness, and errant reasoning from SFRs. In this paper, the formalization of SFRs is done manually. A manual formalization process could lead to a non-correspondence between the informal and formal artifacts, i.e., the traceability link between informal artifacts and formal artifacts might not be accurate nor complete. To resolve this issue, we develop an assurance case argument similar to those in [13, 18, 36] to justify the gap (see Figure 2, step 2) created by a manual formalization step.

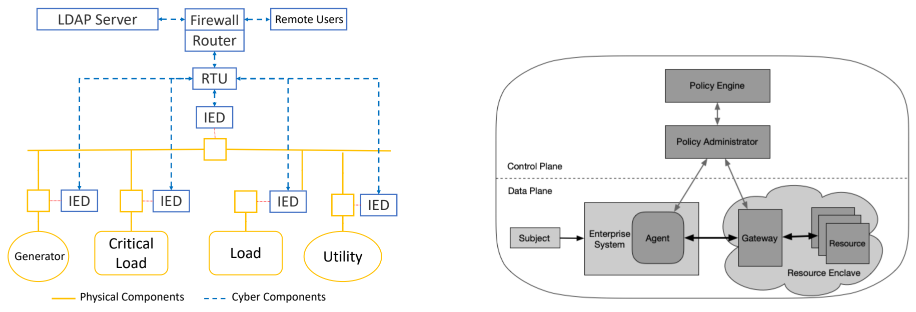
The output artifacts from the formalization step are: (a) the behavioral specification, namely, a formal specification of the core functionalities of microgrids (e.g., the closed loop controller **`WholeMG`** formalized in subsection 4.7), well-formedness conditions, and state invariants (e.g., the state invariant **`ECurrentCBrkr_inv`** formalized in subsection 4.6), i.e., an embedding of the functional requirements in the logic of Isabelle/HOL; and, (b) security properties for microgrid configurations (e.g., the SFR **`RTU_SFR`** formalized in subsection 4.6), i.e., an embedding of the security objectives in the logic of Isabelle/HOL.

Finally, step (3) is to use Isabelle/HOL to generate machine checked deductive proofs for the formal specification; these deductive proofs are our evidential artifacts supporting the security claim stating that ”The OT environment is secure following ZTA”. This includes proving that the behavioral specification of a given microgrid configuration preserves the state invariant, the well-formedness conditions, and the security properties.

Main challenges. Of particular note are the challenges related to the formal verification of OT environments, such as system complexity (e.g., the system of systems nature of OT environments) and maintenance (e.g., continuous deployment). Given the ongoing transition to renewable energy, OT environments are under continuous deployment (frequently updated and maintained), where heterogeneous devices that have individual control actions and are implemented separately, are integrated to operate together using asynchronous communication protocols. Continuous deployment means composing small new configurations (subsystems) with the existing configuration of the OT environment. The design of OT environments also creates a cyber-physical environment mixing continuous behaviors (the physical side, i.e., the controller infrastructure and its corresponding control actions) and discrete behaviors (the cyber side, i.e., the communication infrastructure and its corresponding control actions). From the point of view of verification based on deductive proofs, this requires a heterogeneous state-space representation to capture the cyber-physical properties and behaviors of the different subsystems. It also requires a modular semantics framework that allows us to describe and to reason about the behavior of individual subsystems (e.g., ICS devices) and then to compose the results for the whole system. Thus, one can expose the verification results obtained at the subsystem level (or device level) to system-level interactions. These challenges are severe barriers for the formal verification of OT environments, and we provide novel solutions in what follows.

Solutions. Modular formal verification of CPSs such as OT environments that are under continuous deployment is a significant challenge for state-of-the-art formal methods. Our solution is to use Isabelle/UTP [15] as a semantics framework to carry out the overall verification. This choice is motivated by multiple factors. First, Isabelle/UTP builds on the seminal work of Hoare & He, which uses unifying theories of programming (UTP) [21]. The latter offers an extensible framework and uses alphabetized relational calculus as a semantic foundation for the unification of features of CPSs. For example, one can incrementally extend UTP with semantics for hybrid programs [12], probabilistic programs [58], imperative programs with exceptions, heaps, and stacks, and also methods of handling the subtleties of concurrent [6, 14], parallel [54], and real-time executions [9]. UTP is thus suitable for the formal verification of OT environments (or any system of systems) that are under continuous deployment. This is because UTP is an extensible verification framework that allows us to semantically describe and incrementally integrate verified features for CPSs. Additionally, Isabelle/UTP has a generic state-space representation using lenses [16], allowing us to describe heterogeneous state spaces, and thus mixed-discrete and continuous (hybrid) behaviors [18] of CPSs can be modeled and verified. Finally, Isabelle/UTP is based on Isabelle/HOL, and thus we will benefit from sophisticated proof engineering tools, such as parallel proof-checking [3]; proof tactic customization via Eisbach [34], a large collection of libraries for the implementation of domain-specific formal languages [36,47,51,52]; and sophisticated provers and constraint solvers [10].

This paper develops multiple extensions to Isabelle/UTP. First, we present a small-step semantics to describe device-level control actions (mostly cyber and a few physical) in microgrid configurations. Next, we develop a big-step semantics to compose device-level and system-level control actions, so one can describe the overall behavior of microgrids. A distinguishing feature of our big-step semantics is the **`MODIFY`** operator (see Table 2), which is used to express *dynamic frames* [24,25]. This allows us to reason about independent regions of the state space in a modular way, i.e., a modular way to carry verified properties through independent regions of the state space (subsystems of the OT environment under verification) and compose them with properties of other independent regions to form a larger state space inheriting the verified properties for free. Then, we develop an extended rule set for Hoare logic to reason about the small-step and big-step semantics in a syntax-directed way. Our *dynamic frame rule* (introduced in subsection 5.2) for the **`MODIFY`** operator is a novel contribution that allows for modular reasoning with Hoare logic. Finally, we developed a modular and dynamic verification condition generator (VCG), which uses our extended rule set for Hoare logic to automatically generate verification conditions. The novelty for our VCG is the automated usage



(a) A microgrid configuration (our case study) (b) Enclave Gateway Model [44, Figure 4]

Fig. 3: Figure 3a is inspired by Figure 3b, where components of Figure 3b are instantiated with ICS devices.

of the dynamic frame rule to discharge an already proven statement about properties of smaller regions of the state space (subsystems) when composed with larger regions.

3 Case study: A microgrid configuration for the control of DERs

To demonstrate the applicability of our approach presented in Section 2, we use the OT environment depicted in Figure 3a, a microgrid configuration for the control of a set of DERs. The microgrid consists of a generator, a critical load, a noncritical load, utility, and breakers (denoted by orange squares) as physical components. The configuration also integrates devices such as IEDs, RTUs, router, firewall, and an LDAP server as the cyber components. The orange solid line (physical) represents the power line, i.e., where the electric current is transported. The blue dashed line (cyber) is the distributed control network infrastructure (the communication network). This paper focuses on the formal verification of the devices (in blue) and the system-level interactions between them (blue dashed line). The verification of the physical components is out of scope of this work.

Figure 3a is a security architecture inspired by Figure 3b, where components of Figure 3b are instantiated by ICS devices in Figure 3a. For example, IEDs and RTU are the main critical resources we would like to protect (our *Resource Enclave*). The firewall and router are our *Gateway* securing access to the critical resources. The LDAP server is the *Policy Administrator* and *Engine* implementing the access control policy. The *Agent* is not considered; instead, we consider *System* (i.e., web portal) through which *subjects* (i.e., remote users) can have a remote access to the Resource Enclave.

In this microgrid configuration, the role of IEDs is to read data from sensors and process these data following the implemented control logic at the device level. In Figure 3a, the main functionality of IEDs is to issue control commands to actuate (i.e., open or close) its assigned circuit breaker. IEDs are critical components because their control commands can badly affect the underlying DERs. For example, if the IED randomly open and close the breakers, then the generator can receive damage [53]. The damage can range from an electrical outage for a short period of time to a more safety critical event, such as a long-term outage (e.g., Florida 2008 outage shutdown[2]). Given

[2] https://www.icscybersecurityconference.com/demo-hacking-protective-relay-taking-control-grid-risk/

their critical role in microgrid configurations, and because IEDs can directly connect to the network and have system-level interactions with other devices (e.g., an RTU), it is essential to formally verify that these devices are secure following ZTA requirements, and thus avoiding scenarios with bad outcomes.

Similarly, the RTU is another critical resource we would like to prove to be secure because of its critical role in microgrid configurations. The role of the RTU is to act as a SCADA that receives the data from IEDs. Based on the collected data, the RTU can send requests to IEDs to either open or close the circuit breakers. The data collected by the RTU are usually sent to other devices for analysis and to determine the current condition of the microgid. Because RTUs are devices that can be accessed remotely, and because they can control other critical devices (i.e., IEDs) and store critical data about the microgrid, we formally verify that access to these devices is secure, and that their system-level interactions do not lead to an inconsistent state of the microgrid.

The LDAP server has the role of policy administrator and engine in the microgrid configuration depicted in Figure 3a. The LDAP server is an application protocol for accessing and maintaining distributed directory information services over an Internet Protocol (IP) network. Directory services play an important role in developing intranet and Internet applications by allowing the sharing of information about users, systems, networks, services, and applications throughout the network. For examples, directory services can provide any organized set of records, often with a hierarchical structure, such as a corporate email directory. All the usernames, passwords, and their corresponding authorization attributes are stored in the LDAP server. LDAP servers are responsible for the authentication and authorization of access of user clients to each device.

Finally, firewall and router in Figure 3a represent the security gateway. The router forwards the network packets to the desired devices based on their IP addresses. The firewall monitors the incoming and outgoing network packets and determines whether the packet need to be transmitted or dropped based on the previously configured set of rules. The network packets can be filtered according to the source and destination IP addresses, protocol, source and destination ports, and encryption state. The filtering in our system is based on IP addresses, ports, and protocol.

To formally verify that the microgrid configuration depicted in Figure 3a is secure, and that its behavior is functionally correct as required by NIST 800-207, we follow the 3 steps captured in Figure 2 and explained in Section 2. The first step is to translate ZTA tenets to SFRs (i.e., implementations of ZTA tenets with components of microgrids). For example, the following SFR is manually extracted from ZTA tenet number 2:

SFR 1 *If the microgrid configuration depicted in Figure 3a is in a consistent state, then a user can have remote access to any of the devices in the microgrid through web portal login by the following steps. i) Server authentication: If a user wants to log into any of the devices, first, the user should verify the server certificate to authenticate the server. In this microgrid configuration, this is internally achieved by the use of Transport Layer Security (TLS) protocol [40]. ii) User Authentication: Once the encrypted channel is established, the user must be authenticated to perform any actions related to the device, such as changing settings/configurations and issuing commands. The user authentication is performed after the user enters a username and password. This information is sent to the LDAP server, and the LDAP server checks the database and issues an authentication* *for the user if the credentials are valid. iii) User Authorization: User authorization is based on user credentials. The LDAP server checks its database and issues the authorization attributes, such as privilege level, to the user, and this information is propagated to devices. The steps (i)–(iii) should not lead to inconsistent states of the microgrid configuration.*

The formal verification presented in Section 5 is centered around this SFR. This SFR directly responds to the ZTA Tenet 2 in NIST 800-207 because it prevents cyber threats, such as man-in-the-middle attacks and server spoofing. Other SFRs were defined to address this and other ZTA tenets (the relation between ZTA tenets and SFRs can be one-to-many), but it will not be presented here due to the space limits.

4 Formal modeling of microgrids

This section presents a formal specification for the microgrid configuration described in Figure 3a. Our goal is to verify **SFR 1** introduced in Section 3. The formal specification is composed of: (1) models to describe the state space of the microgrid (see subsection 4.2); (2) state invariants and enabling conditions to describe the consistency of the microgrid configuration (see subsection 4.6); (3) a small-step semantics to describe single stage operations (see subsection 4.5); (4) a big-step semantics to compose these operations and model the behavior of devices and the behavior of the system as whole (see subsection 4.7); and (5) automated and modular proofs of the desired properties (see subsection 5.1 and subsection 5.2). In subsection 4.1 we will introduce preliminaries on Isabelle/HOL which is the formal specification language we are using.

4.1 Notations

The notation used in this paper assumes familiarity with higher order logic (HOL) and its implementation in the ITP Isabelle/HOL [37]. Isabelle/HOL is a proof assistant that has features of functional programming languages and includes a support for HOL specifications and structured interactive proofs. The $\wedge$, $\vee$, $\forall$, and $\exists$ are the usual logical connectors for Boolean expressions. Notations for set theory—such as $\cap$, $\cup$, and `{}`—are also supported. The term language of Isabelle/HOL has a lambda calculus abstract syntax. For example, a non-recursive function has the following notation:

```
definition add1 x = x + 1
```

. . . where **add1** is the name of the function, `x` is the input argument, and `x + 1` is the ”return” value (the image of `x`). The same function can have an equivalent notation using a lambda-like abstract syntax, as follows: `λx . x + 1`. Functional programming features such as **let** expressions, well-founded recursive functions via **fun**, syntax trees via **datatype** (which can also be used as an enumeration type), primitive-type definitions via **typedef**, and records-type definitions via **record** (tuples with an advanced infrastructure improving proof automation) are also supported by the term language of Isabelle/HOL. Boolean terms that require interactive proofs are specified using:

```
lemma inter_is_idempotent:
  "A ∩ A = A"
by auto
```

. . . where `inter_is_idempotent` is an optional label name for the theorem to be proven, `"A ∩ A = A"` is the theorem specifying the algebraic property, and `by auto` is the proof of the theorem; in this case, the proof was automatically generated using the proof tactic `auto`. Structured and human readable proofs are also supported using the proof language Isabelle/Isar [50]. We believe that this brief introduction to Isabelle's term language reviews the frequently used terms and symbols, so the rest of this paper will make free use of Isabelle notations.

4.2 Data models

The data models used for the specification of the state space of the microgrid configuration described in Figure 3a is bifurcated into: the state space for each individual device (e.g., state space model of IED devices described in subsection 4.3) and the state space for the whole system (introduced in subsection 4.4). The data model describing the state space for individual devices is an abstraction of CONFIG files used by real world ICS devices. In our model each individual device is seen as a CONFIG file which has a logic module and communications module. For example, the data model used to specify components of CONFIG files used by IEDs is defined in Isabelle/HOL as follows:

```
type_synonym ECURRENT = int
type_synonym TIME = nat -- {*Time unit*}
type_synonym ID=nat--{*Identification #*}
type_synonym CBRKRID = ID
type_synonym IEDID = ID
datatype CBRKRSTATUS = cOpen  | cClosed
datatype ALARM = silent | alarming
datatype IEDSTATUS =
  iedOperational | gotCmdRTU |iedShutdown|
  ...
```

. . . where types fully written with upper cases are either type synonyms of Isabelle's primitive types (specified using **`type_synonym`**) or enumeration types (specified using **`datatype`**). For example, the **`type_synonym`** `ECURRENT` denotes electrical current, and it is specified using Isabelle's primitive type for signed integers `int`. We use signed integers to model the direction of the electrical current flow. For example, in `+5`, the sign + means that the electrical current flow away from the device, and the value `5` is the magnitude of current (in Amperes).

4.3 State space of ICS devices in Isabelle/UTP

The state space of ICS devices that are components of the microgrid under verification is specified using **`alphabet`**, which is used by Isabelle/UTP to mimic Isabelle/HOL record types. The main difference between Isabelle/HOL record types specified using **`record`** and Isabelle/UTP types that are specified using **`alphabet`** is that the fields of **`alphabet`** are lenses [16] (i.e., an algebraic structure used to describe variables in an axiomatic way). We have contributed a new version for the **`alphabet`** package allowing for type overloading. For example, the Isabelle/UTP representation of the state space of IEDs is specified using a record of lenses as follows:

```
alphabet IEDCyberInter =              alphabet IED =
  status        :: IEDSTATUS            ied_id        :: IEDID
  rtuCmdTimeout:: TIME                  iedCyberInter:: IEDCyberInter
                                        eCurrentCBrkr:: CBrkrECurrent
```

```
alphabet CBrkrECurrent =
  cBrkrID        :: CBRKRID
  cBrkrStatus    :: CBRKRSTATUS
  currentECurrent:: ECURRENT
  ...
```

. . . where each field of record types specified using **alphabet** models an independent region of the state space. For example, the root type `IED` is a record with three fields. The first field of `IED` is the lens `ied_id`, which has the view type `IEDID`, and models the identification number of the device. As a result, the lens `ied_id` will use a value from the type `IEDID` to store the identification number of a given IED. The second field of the record type `IED` is `iedCyberInter`, a lens that characterizes a region of the state space independent from the one characterized by the lens `ied_id`. The lens `iedCyberInter` stores information related to cyber interactions between an IED and other devices, such as RTU. The lens `iedCyberInter` has the view type `IEDCyberInter`, specifying a record of lenses, and it includes the field `status` which uses the type `IEDSTATUS` to store the status of the IED when interacting with its assigned RTU (see Figure 3a). The field `rtuCmdTimeout` of the record type `IEDCyberInter` stores the time-out threshold for responding to requests sent by the RTU. The third field of the record type `IED` is the lens `eCurrentCBrkr`, which characterizes another independent region of the state space, and it is used to store information about the physical components interacting with IED. In this case, the lens `eCurrentCBrkr` will store information related to the circuit breaker that is controlled by the IED. The lens `eCurrentCBrkr` has the view type `CBrkrECurrent`, specifying a record of lenses, in which each lens stores information related to the circuit breaker controlled by the IED. For example, the lens `cBrkrStatus` of the record type `CBrkrECurrent` is used to model the status of the circuit breaker. The lens `cBrkrStatus` has the view type `CBRKRSTATUS`, which allows the status of the breaker to take one of two values: `cOpen`, when it is open; or `cClosed`, when it is closed. Similarly, we model the state spaces of the other devices, such as the RTU, the LDAP server, the firewall, and the router.

4.4 State space of the whole microgrid

The data model describing the state space of the whole microgrid (described in Figure 3a) is used to additionally capture the effects of system-level interactions (i.e., signals sent between devices). The Isabelle/UTP model for the state space of the system as a whole is defined as follows:

```
alphabet ControlledVars =
  cBrkrCon   :: CBRKRSTATUS
  ...
alphabet MonitoredVars =
  eCurrentMon    :: ECURRENT
  maxECurrentMon:: ECURRENT
alphabet Environment =
  controlled:: ControlledVars
  monitored :: MonitoredVars
```

```
alphabet Devices =
  ied :: IEDID => IED option
  rtu :: RTU
  ...
alphabet MGConfig =
  devices:: Devices
  env    :: Environment
```

. . . where the root record has the type `MGConfig`, and it specifies the whole microgrid configuration. The field `devices` of the lens record `MGConfig` has the type `Devices`,

Table 1: Syntax and description of the small-step semantics (selected list).

Constructs	Description
UserLogOnToRTU	This is a multistage operation. This operation can be instantiated for any device allowing remote access through the web portal (in this case, the RTU). It represents a state-transition system where each transition is a single-stage operation that represents a step that is required to log on to the device. The transitions are mainly related to authentication and authorization operations which model the steps of the TLS protocol.
IEDOpenBreaker	This is a single-stage operation. This operation can be executed by IEDs. The main effect of this operation is to open the breaker.
IEDCloseBreaker	The effect is to close the breaker. Otherwise, same as **IEDOpenBreaker**.
RTUSendLDAPAuthCheckRequest	The RTU sends a signal to the LDAP server to check if the user has the access rights to log on to this RTU.
RTURecvLDAPAuthCheckRequestOK	RTU receives a signal from the LDAP that the user access request is accepted.
RTURecvLDAPAuthCheckRequestFail	RTU receives a signal from the LDAP that the user access request is denied.

which is itself a record of lenses. It is used to model the state space of devices that are components of the microgrid configuration. The field `env` of the lens record `MGConfig` has the type `Environment`, and it is used to model the state space of the physical environment.

The modeling pattern used to describe the state space of the physical environment is similar to the one used in [2, 18, 36]. The pattern splits the physical environment into: monitored variables (sensors) and controlled variables (actuators). Monitored variables store the sensed data from the outside environment (e.g., sensing the value of the electrical current, which is modeled by the lens `eCurrentMon`), and the controlled variables store the status of the physical components that can be changed by devices, e.g., the IED can actuate (i.e., open and close) the circuit breaker, which is modeled by the lens `cBrkrCon`.

4.5 Small-step semantics

The system-level view of the microgrid configuration depicted in Figure 3a consists of a composition of single-stage operations performed by devices to update their own state space and also single-stage operations to interact with other devices and update the state space of the whole microgrid. We call these single-stage operations the small-step semantics. A general representation for these single-stage operations in Isabelle/HOL is:

```
op = λ s s'. E(s) ⟶ A(s,s')
```

. . . where **op** can be substituted by any operation from Table 1, and **E**`(s)` is a Boolean expression, i.e., it is a unary relation (a predicate) on the initial state `s`. **E**`(s)` specifies the enabling condition for the single-stage operation **op**. **A**`(s,s')` is a binary relation between the initial state `s` and the final state `s'`. **A**`(s,s')` specifies the effect of the operation on the state space, i.e., substitutions of the values of variables (lenses) at state `s` with new values that yield a new state `s'`.

In fact, enabling conditions, **E**`(s)`, are guards for single-stage operations. That is if **E**`(s)` is evaluated to be true, then the behavior of **op** is characterized by the set of reachable states denoted by the relation **A**`(s,s')`. If **E**`(s)` is false, however, then the behavior of **op** is completely nondeterminisitc, i.e, **op** will have a divergent set of reachable states and will behave exactly as **DIVERGE** from Table 2. At the implementation level, enabling conditions are used to describe constraints on a particular region of the state space, such as time constraints that a given variable should satisfy or constraints about the range of possible values for a given variable. For example, the single-stage operation **IEDOpenBreaker** from Table 1, which is performed by IED, has the following semantics in Isabelle/HOL:

```
definition IEDOpenBrkr eCurrentValue =
λ s s'.
 abs (lens_lookup maxECurrent (s)) ≤ abs (eCurrentValue) ⟶
 s' = lens_upd cBrkrStatus s (cOpen)
```

. . . where `E(s)` is substituted by the Boolean expression:

```
abs (lens_lookup maxECurrent (s)) ≤ abs (eCurrentValue)
```

. . . and `A(s,s')` is substituted by `s' = lens_upd cBrkrStatus s (cOpen)`. The input argument `eCurrentValue` represents the electrical current that the IED senses from the power line. `maxECurrent` is a lens characterizing a region of the state space where the maximum value of the electrical current is stored. `abs` is a function that returns the absolute value of a signed integer. `lens_lookup` is a function that retrieves the value of a given lens at a specified state; in this case, it was used to retrieve the value of the maximum current stored in `maxECurrent`. For simplicity, we will use the notation `X!!s` instead of `lens_lookup X (s)`. `cBrkrStatus` is a lens storing the status of the circuit breaker (described in subsection 4.2). `lens_upd` is a function that updates a lens with a given value at a given state; in this case, it was used to update `cBrkrStatus` with the value `cOpen`. The full definition of **IEDOpenBreaker** has additional input arguments and performes more substitutions on the state `s`. Details are omitted here for simplicity and brevity.

4.6 Invariants and security properties

We use state invariants to specify the consistency of the microgrid. A state invariant is a predicate on the state, `s`, that specifies a consistent state for a given device and for the microgrid as a whole. To prove that the state invariant is preserved by the microgrid configuration, we assume that the invariant holds on the initial state, and we prove that it still holds after performing any microgrid's operation. For the case of circuit breakers controlled by IEDs, the state invariant has the following notation in Isabelle/HOL:

```
definition ECurrentCBrkr_inv s =
(abs (maxECurrent!!s) >
 abs (currentECurrent!!s) ⟷ ((cBrkrStatus!!s) = cClosed ∧ ...)
     ∧
((eCurrentAlarm !!s) = alarming ⟷
 ((cBrkrStatus !!s) = cClosed ∧ (currentTime!!s) ≥ (
    alarmTimeout!!s)∧ ...))
```

. . . Here, the invariant is a conjunction of cases specifying a consistent state of the circuit breaker. This specifies the consistent state of the variables `cBrkrStatus` and `eCurrentAlarm`. The invariant **ECurrentCBrkr_inv** has the following specification pattern:

```
case1 ∧ case2 ∧ ...
```

. . . where each case has the following pattern:

```
 Z(s) ⟷ P(s) ∧ Q(s) ∧ ...
```

For example, the case:

```
(eCurrentAlarm!!s)=alarming ⟷ (cBrkrStatus !!s)=cClosed ∧...
```

is the case where the invariant **ECurrentCBrkr_inv** specifies when the alarm should be `alarming`. Other state invariants that specify the consistency of the other devices are formalized but omitted here for brevity.

Similarly, we formalize **SFR 1** from Section 3 as a security property for the RTU in the form of a predicate on the state. The reason we do not instantiate **SFR 1** for IEDs is because we assume that user access to IEDs is possible only through a web portal log-in to RTU. Specifically, we assume that no direct access to IEDs is possible for remote users without admin privileges. To prove that the security property **RTU_SFR** defined below is satisfied, we assume it to be true on the initial state of the microgrid and then prove it to be true after the execution of the microgrid's operations.

```
definition RTU_SFR cportID usrID t s =
((rtuSrvrStatus!!s) = rtuOperational ∧
 RTU_inv_1_10 cportID usrID t s ∧
 ValidLDAPServer (rtuCurrentLDAP!!s) ∧
 ValidDigiCertificate t (rtuDigiCert!!s) ∧
 (∃ c. c = (rtuDigiCert!!s) ∧
  ValidKeyStore t c (rtuKeyStore!!s)) ∧
 (∀ s'. s ∈ {s.
  (rtuSrvrStatus cportID usrID t s) = rtu_UserLogOn ∧
   UserLogOnToRTU cportID usrID t s s'} ⟶
 UCWithRTUCADC cportID usrID t s ∧
 UCAuthCheckSuccess cportID usrID t s ∧
 UCLDAPCredSuccess cportID usrID t s))
```

RTU_SFR, described above, is a formalization of **SFR 1** from Section 3 instantiated for the RTU device. `rtuOperational` means that the RTU is not administrated by an admin, and it is in a state of waiting for log-in requests from remote user, `usrID`, through the web portal, `cportID`. The input arguments `t` and `s` specifies the current time and the current state of the microgrid, respectively. **RTU_inv_1_10** is an Isabelle **definition** that specifies the state invariant of the RTU, i.e., the consistent state of the RTU. As explained in Section 3, because **SFR 1** is enforcing authentication and authorization via TLS protocol and the LDAP server, we use **ValidLDAPServer** to specify the consistency of the LDAP server. For example, **ValidLDAPServer** allows us to check the consistency of the LDAP server after its enrollment by an administrator. For the same reason, we use **ValidDigiCertificate** to specify the validity of the digital certificate. Both **ValidLDAPServer** and **ValidDigiCertificate** are Isabelle **definition**s, and the details of their specifications are omitted for brevity. The rest of the specification of **RTU_SFR**, starting from `∀s'. s ∈ {s. ...} ⟶ ...` until the end, describes the set of states that lead to `rtu_UserLogOn` by executing the multistage operation **UserLogOnToRTU** while satisfying the following:

– **UCWithRTUCADC**, which specifies that the server authentication was successful, i.e., the user client authenticated the digital certificate of the RTU.
– **UCLDAPCredSuccess**, which specifies that the user authentication was successful, i.e., the RTU authenticated the credentials of the user client. This check is done by the LDAP server.
– **UCAuthCheckSuccess**, which specifies that the user client is authorized to log into the RTU with its assigned privileges.

Table 2: Syntax and description of the big-step semantics.

Constructs	Description
`SKIP`	Used to semantically capture stutter states; (e.g., to model a non-terminating loop, one can use `SKIP` as a single statement for the body of the while-loop and keep the loop-condition true).
`MAGIC`	The program that has an empty set of reachable states. Known as the perfect program. It is perfect because it refines any specification. Namely, the Hoare triple $\{\!|$`P`$|\!\}$`MAGIC`$\{\!|$`Q`$|\!\}$ is proven to be true for any assumption `P` and any guarantee `Q`! We are using this program to make properties of the other constructs semantically visible.
`DIVERGE`	The program that has a divergent set of reachable states. The worst program!
`X :== e`	Basic assignment of the value of an expression `e` to a region of the state space characterized by the lens `X`.
`P ;; Q`	Sequentially execute the program `P` then `Q`; this is used to sequentially compose statements.
μ `R• P`	Least fixed point (LFP). Used to model recursion of program `P` with `R` occurring in `P` and representing the point where the recursion is unfolded.
Conditional	Notation: `IF b THEN P ELSE Q FI`; It means execute `P` if `b`, else execute `Q`.
Nondeterminism	Notation: `P` $\sqcap$ `Q`. It means the union between the set of states that are reachable by the program `P` and the set of states that are reachable by the program `Q`.
Iterations	Notation: `FROM P UNTIL b DO Q OD`. It means execute `P` one time, then repeatedly execute `Q` until `b` becomes true. Can be modelled with a combination of sequential composition construct, conditional statement, and the recursion construct as follows: `P ;; (`μ` R• IF¬ b THEN Q ;; R ELSE SKIP FI).`
Framing	Notation: `MODIFY X DO P OD`. It means execute the program `P` and discard all changes made by `P` outside the region of the state space (the frame) characterized by the lens `X`.

4.7 Big-step semantics

The behavior of the system as a whole is modeled using the closed-loop controller defined by **`WholeMG`** below. Input arguments (parameters) of **`WholeMG`** are omitted here for simplicity. **`WholeMG`** uses big-step semantics constructs from Table 2 to compose single-stage operations from Table 1 and define the behavior of the microgrid as a whole.

```
definition WholeMG =
  FROM init s s' until False
    DO SensingEnv  s s';; ChangingEnv s s';;
       DiscreteControl s s';; ContinuousDynamics s s' OD
```

`WholeMG` performs a sequence of controls, each of which changes a region of the state space of the whole microgrid (the state space of the microgrid as a whole is specified using the record of lenses `MGConfig` in subsection 4.4). In **`WholeMG`**, the system-level behavior **`SensingEnv`** will update the state space of each device by the value of the corresponding monitored variable (i.e., the behavior **`SensingEnv`** reads values from regions of the record of lenses `MonitoredVars` specified in subsection 4.4). For example, an IED will periodically sense the magnitude and the direction of the current from the power line. The sensed value is stored in the monitored variable (lens) `eCurrentMon`.

The system-level behavior **`ChangingEnv`** specifies periodic updates on controlled variables from the physical environment (the state space of the physical environment is specified using the record of lenses `Environment`, which is in subsection 4.4). For example, an IED can actuate the physical circuit breaker to change its status. The physical status of the circuit breaker is modeled by the controlled variable (lens) `cBrkrCon`. The system-level behavior **`DiscreteControl`** specifies all possible cyber controls that can be performed by devices belonging to the microgrid configuration. For example, a possible behavior for **DiscreteControl** is the multistage operation **UserLogOnToRTU** from Table 1. Finally, **ContinuousDynamics** specifies the periodic updates that are done by the environment on the monitored variables. For example, we allow the environment to assign random values for the monitored variable eCurrentMon.

5 Modular formal verification of microgrids

Modular verification is a significant challenge formal methods. Because a microgrid configuration is a system of systems that is under continuous deployment, it requires a modular verification approach. To enable modular verification for the behavior of microgrids, we use the framing operator **MODIFY** from Table 2. The framing operator allows us to lift the verification results obtained at the level of single-stage operations to the device level and then to the system level. Specifically, the framing operator will allow us to carry through the verified properties on a given region of the state space (the frame) using the lens characterizing that region. For example, we prove that the single-stage operation **IEDOpenBrkr**, which we formally defined in subsection 4.5, maintains the invariant **ECurrentCBrkr_inv** (which is defined in subsection 4.6), then we use the framing operator to lift the operation **IEDOpenBrkr** to the device level using **IEDOpenBrkr_dvc** defined below.

```
definition IEDOpenBrkr_dvc =
MODIFY eCurrentCBrkr DO IEDOpenBrkr OD
```

IEDOpenBrkr_dvc is a lifting that enables the use of the frame rule (see subsection 5.2), which carries the verified properties about the operation **IEDOpenBreaker** through to the device level using the lens eCurrentCBrkr. Similarly, we lift the device level behavior **IEDOpenBrkr_dvc** to the system level, which allows us to use the frame rule again to carry the verified properties about **IEDOpenBrkr_dvc** through to the system level using the lens devices:

```
definition IEDOps_sys =
MODIFY devices DO IEDOpenBrkr_dvc ⊓ IEDCloseBrkr_dvc OD
```

where **IEDOps_sys** describes the system-level behavior for IEDs, which is one possible behavior for **DiscreteControl**. The latter is the discrete (cyber) part of the cyber physical behavior of the closed-loop controller **WholeMG** described in subsection 4.7.

5.1 Correctness for microgrids

The goal of this work is to expose single-stage operations to system-level interactions and ensure that the security properties and invariants (e.g., see subsection 4.6) still hold. To prove correctness for the microgrid configuration depicted in Figure 3a, we first prove that the closed-loop controller **WholeMG** that we introduced in subsection 4.7 preserves the state invariants (e.g., system-level interactions don't break the state invariant **ECurrentCBrkr_inv** from subsection 4.6). Then, we prove that **WholeMG** preserves **SFR 1** that we described in natural languages in Section 3 and then formalized in Isabelle/HOL in subsection 4.6 using the definition **RTU_SFR**. To do such proofs, we employ Hoare logic and VCG-based reasoning, such as in [4, 30, 59]. Our Hoare triple is defined in Isabelle/HOL as follows:

```
definition ⦃P⦄B⦃Q⦄ =
{(s,s'). P (s)} ∩ {(s,s'). B (s, s')} ⊆ {(s,s'). Q (s')}
```

. . . where `P` is a predicate that characterize the set of initial states (what we assume), `B` is a behavior expressed using Table 2 or using any operation from Table 1 (e.g., `B` can be substituted by the closed-loop controller **`WholeMG`** from subsection 4.7), and `Q` is a predicate that characterizes the set of final states (what we guarantee), e.g., `Q` can be substituted by the state invariant **`ECurrentCBrkr_inv`** and the security property **`RTU_SFR`** we introduced in subsection 4.6. Based on this notation for the Hoare triple, we define the Hoare logic for the big-step and small-step semantics.

5.2 Frame rule

A distinguishing feature of our Hoare logic is the generic scheme for the frame rule, which allows us to enforce modular reasoning. Our frame rule is expressed in Isabelle/HOL as follows:

```
lemma frame_rule:
 assume⦃P⦄B⦃Q⦄
 shows⦃P∧R⦄MODIFY X DO B OD⦃(∃X•Q)∧(∃Y•R)⦄
proof ...
```

This Isabelle/HOL theorem specifies our `frame_rule`. Intuitively, the theorem means: If a behavior, `B`, guarantees the property, `Q`, starting from the assumptions, `P`, then the frame operator, **`MODIFY`**, will use the lens, `X`, to carry the guarantee, `Q`. The guarantees of regions of the state space which are independent from `X`, are characterized by `R`, and are carried through using the lens, `Y`. For example, `Y` can characterize the already-deployed regions of the state space (i.e., the existing system of systems), and `X` characterizes the state space of the newly deployed and integrated subsystem. This yields the ability to verify a system of systems, such as microgirds, in a modular way. Other assumptions required to prove the theorem `frame_rule` are omitted here for brevity.

Our implemented VCG uses this frame rule (its weakest precondition version) to automatically discharge proofs about security properties and state invariants such as those presented in subsection 4.6. These properties are proved to be true on single-stage operations, and the proofs are lifted to the system level using the frame rule. In contrast to the state of the art, where the frame rules are rather specific to spatial reasoning (e.g., reasoning on heaps, such as in separation logic) or temporal reasoning (see, e.g., in [59]), our frame rule is generic and can be instantiated for both spatial and temporal reasoning.

6 Related work

While the use of integrated formal methods with assurance cases for the full specification and verification of an OT system's safety and security properties is described as a relatively new opportunity in [19], here are a few cases in the literature in which formal methods are integrated with system behavioral models and verified using theorem proving. Khan et al. develop a three-part approach to prove system reliability and security for an OT system, in which they use Coq [39] to prove that the system is secure by design, dReal to perform vulnerability analysis, and their own product, ARMET [26], to assess vulnerability to false data injection attacks in real-time, using an example of a gravity-draining water tank. In [7, 33], the authors present VeriDrone, a framework to specify and verify OT systems within the Coq proof assistant; however, the authors focus their application on safety properties of the system while we focus more on security at the cyber and cyber-physical layers.

The closest work to that which we propose is the work by Foster et al. [13], in which the authors formally verify the CPS Tokeneer using Isabelle/SACM [36]. In a similar approach, Cofer et al. [8] develop a formal specification of two separate unmanned aerial vehicles, using the JKind model checker [17] to formally verify model correctness, then using the Isabelle/HOL theorem prover [37] to prove the system's security properties and that the software implementation matches the specification. The work of Cofer et al. includes the automatic generation of software using the formal specification, but unlike the work in Foster et al. , it does not include a model-based assurance case to connect less formalized factors (e.g., regulatory text) with artifacts that can be modeled using formal methods. Finally, in an industrial case study that focuses on protocols, Dreier et al. [11] formally define a metric related to secure message passing, which they term *flow integrity*, and then apply this framework to two known protocols used in ICS, OPC-UA [32] and Modbus [38].

Acknowledgment

This work was authored by the National Renewable Energy Laboratory (NREL), operated by Alliance for Sustainable Energy, LLC, for the U.S. Department of Energy (DOE) under Contract No. DE-AC36-08GO28308. This work was supported by the Laboratory Directed Research and Development (LDRD) Program at NREL. The views expressed in the article do not necessarily represent the views of the DOE or the U.S. Government. The U.S. Government retains and the publisher, by accepting the article for publication, acknowledges that the U.S. Government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this work, or allow others to do so, for U.S. Government purposes.

References

1. Prashant Anantharaman, Vijay H. Kothari, J. Peter Brady, Ira Ray Jenkins, Sameed Ali, Michael C. Millian, Ross Koppel, Jim Blythe, Sergey Bratus, and Sean W. Smith. Mismorphism: The heart of the weird machine. In Jonathan Anderson, Frank Stajano, Bruce Christianson, and Vashek Matyás, editors, *Security Protocols XXVII - 27th International Workshop, Cambridge, UK, April 10-12, 2019, Revised Selected Papers*, volume 12287 of *Lecture Notes in Computer Science*, pages 113–124. Springer, 2019.
2. Janet Barnes, Randy Johnson, and James C. Widmaier. Engineering the tokeneer enclave protection software. In *1st IEEE International Symposium on Secure Software Engineering, ISSSE 2006, Proceedings*, 2006.
3. Bruno Barras, Lourdes Del Carmen González-Huesca, Hugo Herbelin, Yann Régis-Gianas, Enrico Tassi, Makarius Wenzel, and Burkhart Wolff. Pervasive parallelism in highly-trustable interactive theorem proving systems. In Jacques Carette, David Aspinall, Christoph Lange, Petr Sojka, and Wolfgang Windsteiger, editors, *Intelligent Computer Mathematics - MKM, Calculemus, DML, and Systems and Projects 2013, Held as Part of CICM 2013, Bath, UK, July 8-12, 2013. Proceedings*, volume 7961 of *Lecture Notes in Computer Science*, pages 359–363. Springer, 2013.
4. Josh Bockenek, Peter Lammich, Yakoub Nemouchi, and Burkhart Wolff. Using isabelle/utp for the verification of sorting algorithms. In *Proceedings of the Isabelle Workshop, FLoC 2018, Proceedings*, 2018.

5. Alan Burns and Robert Davis. Mixed criticality systems-a review. *Department of Computer Science, University of York, Tech. Rep*, pages 1–69, 2013.
6. Andrew Butterfield, Anila Mjeda, and John Noll. Utp semantics for shared-state, concurrent, context-sensitive process models. In *2016 10th International Symposium on Theoretical Aspects of Software Engineering (TASE)*, pages 93–100, 2016.
7. Matthew Chan, Daniel Ricketts, Sorin Lerner, and Gregory Malecha. Formal verification of stability properties of cyber-physical systems. *Proc. CoqPL*, 2016.
8. Darren Cofer, Andrew Gacek, John Backes, Michael W Whalen, Lee Pike, Adam Foltzer, Michal Podhradsky, Gerwin Klein, Ihor Kuz, June Andronick, et al. A formal approach to constructing secure air vehicle software. *Computer*, 51(11):14–23, 2018.
9. Hung Dang Van and Hoang Truong. *Modeling and Specification of Real-Time Interfaces with UTP*, pages 136–150. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.
10. Martin Desharnais, Petar Vukmirovic, Jasmin Blanchette, and Makarius Wenzel. Seventeen provers under the hammer. In June Andronick and Leonardo de Moura, editors, *13th International Conference on Interactive Theorem Proving, ITP 2022, August 7-10, 2022, Haifa, Israel*, volume 237 of *LIPIcs*, pages 8:1–8:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.
11. Jannik Dreier, Maxime Puys, Marie-Laure Potet, Pascal Lafourcade, and Jean-Louis Roch. Formally verifying flow properties in industrial systems. In *SECRYPT 2017-14th International Conference on Security and Cryptography*, pages 55–66, 2017.
12. Simon Foster. Hybrid relations in isabelle/utp. In Pedro Ribeiro and Augusto Sampaio, editors, *Unifying Theories of Programming*, pages 130–153, Cham, 2019. Springer International Publishing.
13. Simon Foster, Yakoub Nemouchi, Mario Gleirscher, Ran Wei, and Tim Kelly. Integration of formal proof into unified assurance cases with Isabelle/SACM. *Formal Aspects of Computing*, 33(6):855–884, 2021.
14. Simon Foster, Kangfeng Ye, Ana Cavalcanti, and Jim Woodcock. Automated verification of reactive and concurrent programs by calculation. *J. Log. Algebraic Methods Program.*, 121:100681, 2021.
15. Simon Foster, Frank Zeyda, Yakoub Nemouchi, Pedro Ribeiro, and Burkhart Wolff. Isabelle/utp: Mechanised theory engineering for unifying theories of programming. *Arch. Formal Proofs*, 2019, 2019.
16. Simon Foster, Frank Zeyda, and Jim Woodcock. Unifying heterogeneous state-spaces with lenses. In *13th International Colloquium on Theoretical Aspects of Computing*, ICTAC 2016, pages 295–314. Springer International Publishing, October 2016.
17. Andrew Gacek, John Backes, Mike Whalen, Lucas Wagner, and Elaheh Ghassabani. The JKind model checker. In *Computer Aided Verification: 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part II 30*, pages 20–27. Springer, 2018.
18. Mario Gleirscher, Simon Foster, and Yakoub Nemouchi. Evolution of formal model-based assurance cases for autonomous robots. In Peter Csaba Ölveczky and Gwen Salaün, editors, *Software Engineering and Formal Methods - 17th International Conference, SEFM 2019, Oslo, Norway, September 18-20, 2019, Proceedings*, volume 11724 of *Lecture Notes in Computer Science*, pages 87–104. Springer, 2019.
19. Mario Gleirscher, Simon Foster, and Jim Woodcock. New opportunities for integrated formal methods. *ACM Computing Surveys (CSUR)*, 52(6):1–36, 2019.
20. Benjamin Green, Ric Derbyshire, William Knowles, James Boorman, Pierre Ciholas, Daniel Prince, and David Hutchison. ICS testbed tetris: Practical building blocks towards a cyber security resource. In *The 13th USENIX Workshop on Cyber Security Experimentation and Test (CSET'20)*, 2020.

21. Charles Antony Richard Hoare and He Jifeng. *Unifying Theories of Programming*. Prentice Hall Englewood Cliffs, 1998.
22. Abdulmalik Humayed, Jingqiang Lin, Fengjun Li, and Bo Luo. Cyber-physical systems security — A survey. *IEEE Internet of Things Journal*, 4(6):1802–1831, 2017.
23. Jay Johnson, Timothy Berg, Benjamin Anderson, and Brian Wright. Review of electric vehicle charger cybersecurity vulnerabilities, potential impacts, and defenses. *Energies*, 15(11):3931, 2022.
24. Ioannis T. Kassios. Dynamic frames: Support for framing, dependencies and sharing without restrictions. In Jayadev Misra, Tobias Nipkow, and Emil Sekerinski, editors, *FM 2006: Formal Methods, 14th International Symposium on Formal Methods, Hamilton, Canada, August 21-27, 2006, Proceedings*, volume 4085 of *Lecture Notes in Computer Science*, pages 268–283. Springer, 2006.
25. Ioannis T. Kassios. The dynamic frames theory. *Formal Aspects Comput.*, 23(3):267–288, 2011.
26. Muhammad Taimoor Khan, Dimitrios Serpanos, and Howard Shrobe. ARMET: Behavior-based secure and resilient industrial control systems. *Proceedings of the IEEE*, 106(1):129–143, 2017.
27. John Kindervag. Build security into your network's DNA: The zero trust network architecture. *Forrester Research*, pages 1–26, 2010.
28. Velin Kounev, David Tipper, Attila Altay Yavuz, Brandon M Grainger, and Gregory F Reed. A secure communication architecture for distributed microgrid control. *IEEE Transactions on Smart Grid*, 6(5):2484–2492, 2015.
29. Tomas Kulik, Brijesh Dongol, Peter Gorm Larsen, Hugo Daniel Macedo, Steve Schneider, Peter WV Tran-Jørgensen, and James Woodcock. A survey of practical formal methods for security. *Formal Aspects of Computing*, 34(1):1–39, 2022.
30. Peter Lammich and Simon Wimmer. IMP2 - simple program verification in isabelle/hol. *Arch. Formal Proofs*, 2019, 2019.
31. Ted G Lewis. *Critical infrastructure protection in homeland security: defending a networked nation*. John Wiley & Sons, 2019.
32. Wolfgang Mahnke, Stefan-Helmut Leitner, and Matthias Damm. *OPC unified architecture*. Springer Science & Business Media, 2009.
33. Gregory Malecha, Daniel Ricketts, Mario M Alvarez, and Sorin Lerner. Towards foundational verification of cyber-physical systems. In *2016 Science of Security for Cyber-Physical Systems Workshop (SOSCYPS)*, pages 1–5. IEEE, 2016.
34. Daniel Matichuk, Toby Murray, and Makarius Wenzel. Eisbach: A proof method language for Isabelle. *Journal of Automated Reasoning*, 56(3):261–282, March 2016.
35. Mohammed Moness and Ahmed Mahmoud Moustafa. A survey of cyber-physical advances and challenges of wind energy conversion systems: Prospects for internet of energy. *IEEE Internet of Things Journal*, 3(2):134–145, 2015.
36. Yakoub Nemouchi, Simon Foster, Mario Gleirscher, and Tim Kelly. Isabelle/sacm: Computer-assisted assurance cases with integrated formal methods. In Wolfgang Ahrendt and Silvia Lizeth Tapia Tarifa, editors, *Integrated Formal Methods - 15th International Conference, IFM 2019, Bergen, Norway, December 2-6, 2019, Proceedings*, volume 11918 of *Lecture Notes in Computer Science*, pages 379–398. Springer, 2019.
37. Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002.
38. MODBUS Organization. *MODBUS Messaging on TCP/IP Implementation Guide: v1.0b*. MODBUS Organization, 2006.
39. Christine Paulin-Mohring. Introduction to the Coq proof-assistant for practical software verification. *Tools for Practical Software Verification: LASER, International Summer School 2011, Elba Island, Italy, Revised Tutorial Lectures*, pages 45–95, 2012.

40. Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *J. Comput. Secur.*, 6(1-2):85–128, 1998.
41. André Platzer. *Logical analysis of hybrid systems: proving theorems for complex dynamics*. Springer Science & Business Media, 2010.
42. André Platzer. *Logical foundations of cyber-physical systems*. Springer, 2018.
43. Marco Rocchetto and Nils Ole Tippenhauer. Towards formal security analysis of industrial control systems. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 114–126, 2017.
44. Scott W Rose, Oliver Borchert, Stuart Mitchell, and Sean Connelly. Zero trust architecture. Technical report, NIST, 2020.
45. AK Saha, Sunetra Chowdhury, SP Chowdhury, and PA Crossley. Modeling and performance analysis of a microturbine as a distributed energy resource. *IEEE Transactions on Energy Conversion*, 24(2):529–538, 2009.
46. A Boudghene Stambouli and Enrico Traversa. Solid oxide fuel cells (sofcs): a review of an environmentally clean and efficient source of energy. *Renewable and sustainable energy reviews*, 6(5):433–455, 2002.
47. Frédéric Tuong and Burkhart Wolff. Deeply integrating C11 code support into isabelle/pide. In Rosemary Monahan, Virgile Prevosto, and José Proença, editors, *Proceedings Fifth Workshop on Formal Integrated Development Environment, F-IDE@FM 2019, Porto, Portugal, 7th October 2019*, volume 310 of *EPTCS*, pages 13–28, 2019.
48. Roman Vakulchuk, Indra Overland, and Daniel Scholten. Renewable energy and geopolitics: A review. *Renewable and sustainable energy reviews*, 122:109547, 2020.
49. Steffen Wendzel, Jernej Tonejc, Jaspreet Kaur, and Alexandra Kobekova. Cyber security of smart buildings. *Security and Privacy in Cyber-Physical Systems: Foundations, Principles and Applications*, pages 327–351, 2017.
50. Makarius Wenzel. Structured induction proofs in isabelle/isar. In Jonathan M. Borwein and William M. Farmer, editors, *Mathematical Knowledge Management, 5th International Conference, MKM 2006, Wokingham, UK, August 11-12, 2006, Proceedings*, volume 4108 of *Lecture Notes in Computer Science*, pages 17–30. Springer, 2006.
51. Makarius Wenzel. Isabelle/jedit as IDE for domain-specific formal languages and informal text documents. In Paolo Masci, Rosemary Monahan, and Virgile Prevosto, editors, *Proceedings 4th Workshop on Formal Integrated Development Environment, F-IDE@FLoC 2018, Oxford, England, 14 July 2018*, volume 284 of *EPTCS*, pages 71–84, 2018.
52. Makarius Wenzel. Interaction with formal mathematical documents in isabelle/pide. In Cezary Kaliszyk, Edwin C. Brady, Andrea Kohlhase, and Claudio Sacerdoti Coen, editors, *Intelligent Computer Mathematics - 12th International Conference, CICM 2019, Prague, Czech Republic, July 8-12, 2019, Proceedings*, volume 11617 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2019.
53. Wikipedia. Aurora Generator Test. https://en.wikipedia.org/wiki/Aurora_Generator_Test.
54. Jim Woodcock and Arthur Hughes. Unifying theories of parallel programming. In Chris George and Huaikou Miao, editors, *Formal Methods and Software Engineering*, pages 24–37, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.
55. Jean-Paul A Yaacoub, Ola Salman, Hassan N Noura, Nesrine Kaaniche, Ali Chehab, and Mohamad Malli. Cyber-physical systems security: Limitations, issues and future trends. *Microprocessors and microsystems*, 77:103201, 2020.
56. Geeta Yadav and Kolin Paul. Architecture and security of SCADA systems: A review. *International Journal of Critical Infrastructure Protection*, 34:100433, 2021.
57. Jin Ye, Annarita Giani, Ahmed Elasser, Sudip K Mazumder, Chris Farnell, Homer Alan Mantooth, Taesic Kim, Jianzhe Liu, Bo Chen, Gab-Su Seo, et al. A review of cyber–physical

security for photovoltaic systems. *IEEE Journal of Emerging and Selected Topics in Power Electronics*, 10(4):4879–4901, 2021.

58. Kangfeng Ye, Simon Foster, and Jim Woodcock. Automated reasoning for probabilistic sequential programs with theorem proving. In Uli Fahrenberg, Mai Gehrke, Luigi Santocanale, and Michael Winter, editors, *Relational and Algebraic Methods in Computer Science*, pages 465–482, Cham, 2021. Springer International Publishing.
59. Bohua Zhan, Yi Lv, Shuling Wang, Gehang Zhao, Jifeng Hao, Hong Ye, and Bican Xia. Compositional verification of interacting systems using event monads. In June Andronick and Leonardo de Moura, editors, *13th International Conference on Interactive Theorem Proving, ITP 2022, August 7-10, 2022, Haifa, Israel*, volume 237 of *LIPIcs*, pages 33:1–33:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.