A Digital Twin Runtime Verification Framework for Protecting Satellites Systems from Cyber Attacks

Zhé Hóu*, Qinyi Li*, Ernest Foo*, Jin Song Dong*† and Paulo de Souza*
*Griffith University, Australia, {*z.hou, qinyi.li, e.foo, j.dong, paulo.desouza*}*@grffith.edu.au*
†National University of Singapore, *dcsdjs@nus.edu.sg*

***Abstract*—This paper presents the conceptualisation of a framework that combines digital twins with runtime verification and applies the techniques in the context of security monitoring and verification for satellites. We focus on special considerations needed for space missions and satellites, and we discuss how digital twins in such applications can be developed and how the states of the twins should be synchronised. In particular, we present state synchronisation methods to ensure secure and efficient long-distance communication between the satellite and its digital twin on the ground. Building on top of this, we develop a runtime verification engine for the digital twin that can verify properties in multiple temporal logic languages. We end the paper with our proposal to develop a fully verified satellite digital twin system as future work.**

***Index Terms*—model checking, runtime verification, satellite, spacecraft, digital twins, cybersecurity**

I. Introduction

Satellite communications are critical for data and space asset control. Jamming of satellite communications is a known capability of several nations. However, jamming signals is a blunt tool. Adversaries that can access satellite communications can obtain data collected by the satellite and log information stored on the satellite, including previous settings and locations of the satellite and the data that was collected. For example, satellite communication is now assumed to transmit high-speed financial information, as well as strategic military information. Breach of such data may lead to catastrophic events. Satellites also control space assets such as the Mars rover, which relays communications and control commands via the Mars Reconnaissance Orbiter, a satellite in orbit above Mars. Adversaries that control satellite communications can inject or modify command messages to change the satellite's location and trajectory or any other linked space assets. Therefore, it is essential that satellite functionality and deep space network communications are protected and secured.

The current trend in satellite design is to have smart satellites with more features and modules to ensure that more efficient communications and secure onboard analysis can occur. However, those more complex systems are more likely to be vulnerable to cyber-attacks.

The two major challenges facing the protection of deep space satellites include (1) addressing the isolated nature of the deployed satellite and (2) the high latency and high error environment that communications must travel in. The other major challenge is the limited processing capacity on board of the satellite, specifically as power is mainly provided from a solar battery. Although modern satellites employ faster CPUs, many models are still significantly slower than consumer desktops [1], [2]. Running formal verification directly on satellites may take up too much computational resource that could have been allocated to communication and other tasks.

One of the efficient ways to monitor and verify the security of satellites and other space assets is runtime verification. In fact, NASA has developed a runtime verification technique that can be applied to check autonomous agents running on the PLASMA planning system [3]. However, new solutions must be developed if we want to further offload the computational tasks onboard the satellite.

The US Air Force is using a digital twin of the Lockheed Martin GPS IIR satellite to detect cybersecurity issues by performing penetration testing on the digital replica of the satellite. Their approach demonstrates the feasibility and potential of performing security analysis for satellite systems using digital twins. In this paper, we aim to improve the above approaches by combining runtime verification with digital twins and extending the application to both low-orbit satellites and deep-space satellites.

The combination of digital twins and runtime verification is still a very young idea that has only been briefly explored in the analysis of cyber-physical systems recently [4], [5]. However, there is a significant lack of technical details in the literature. Moreover, the aforementioned challenges require an innovative design of the communication protocols to factor in the delays while maintaining the security of communication and synchronisation of digital twins. Thus, this paper focuses on the application in satellites and space missions and the foundations of the involved techniques.

The main contributions of this paper are as follows: First, we present a conceptual framework of a digital twin system specialised for satellites and space communications. We then present an efficient and secure communication protocol for maintaining state between the digital and physical twin. Our future plans include formally verifying the design and implementation of the digital twin system and the protocols using Process Analysis Toolkit (PAT) [6]. Towards building an integrated system that combines required functionalities and verification tools, we propose implementing runtime monitoring and verification of the digital twin system using PAT as well. This way, we can minimise the overall system design and

software development load. Therefore, this paper also develops a versatile runtime verification tool driven by PAT that can deal with multiple temporal logic languages.

II. Preliminaries

A. Cybersecurity of Satellites

The main purpose of satellites is to enable communications over the horizon or to a location in deep space such as the Mars rover. Satellite communications consist of a ground station transmitter, the satellite and the receiving entity. Satellites can broadcast messages to multiple receiving entities.

In addition to the communications payload, satellites have a Tracking Telemetry and Control (TT&C) system. The TT&C system is a two-way communications link between the ground station and the satellite. The TT&C system allows the satellite to be controlled from the earth, maintain its orbit and temperatures, and operate other systems. Telemetry that can be collected from the satellite includes temperature, electrical voltages and other information on satellite system status.

Of all the components of the satellite communication system, the ground station is the main target of attackers. From the ground station, attackers can use the TT&C system to control the satellite in orbit. Compromise of the ground station also allows attackers to control the data payload transmitted to the satellite. Another option for attackers is to compromise the authentication system and conduct man in the middle attacks between the ground station and the satellite. It is also possible for attackers to embed malicious code in the satellite during maintenance or even before it is launched.

In this paper we propose an additional system to provide defence in depth for the satellite system. The use of digital twin systems will also allow the monitoring of satellite systems for indicators of compromise and the ability to simulate and verify the best response actions. In particular, the digital twin system will continuously monitor 1) the integrity of satellite systems and 2) authenticity of satellite communications.

B. Digital Twins

The concept of a digital twin is the creation of a digital representation of a physical object, process or system. System and sensor data from the physical twin are sent to the digital twin to maintain the latter. The digital twin simulates the physical twin, analyses the outcomes of the simulations, determines the best course of action and then sends the commands necessary to undertake those actions back to the physical twin. The cycle of update, analyse and respond continues.

There are several challenges for digital twin systems. To be effective, the digital twin must be able to synchronise states with the physical twin. The digital twin simulation and analysis need to be such that there is enough bandwidth to transmit state data from the physical twin and enough processing power to ensure efficient simulations to ensure effective decisions can be made in a timely manner. Our proposed synchronisation protocol will help with this.

The advantage of the digital twin is that it allows the more powerful simulation and analysis tools to be deployed at the digital twin that may not have been able to be run on the physical twin alone. Digital twins can be used in single object systems and complex multiple node systems. This technology has mainly been used to optimise and maintain systems, so exploring the possibilities in runtime monitoring and verification for space assets is a new angle.

C. Runtime Verification with Temporal Logic

Runtime verification [7] monitors the execution of a software or hardware system and checks user-specified properties against the currently observed execution trace of system states. It is computationally cheaper than other formal verification methods such as model checking and theorem proving because it only concerns a single execution trace. In contrast, model checking attempts to check every possible execution trace of the system. This work is focused on runtime verifying properties that are specified using temporal logics.

Linear Temporal Logic (LTL) [8] is a widely used language for specifying properties such as safety, fairness, liveness, etc.

Definition II.1 (Syntax of LTL)**.** *The syntax of LTL is given in the Backus–Naur form (BNF) below.*

$$F ::= \top \mid p \mid \neg F \mid F \wedge F \mid \mathbb{X}\, F \mid F\ \mathbb{U}\ F$$

There are two temporal modalities in the above syntax: $\mathbb{X}$ is written prefix and means *next*, and $\mathbb{U}$ is written infix and means *until*.

The formal semantics of LTL is often defined using a Kripke model M, which is a tuple $M = (S, I, R, L)$ where

- S is a finite set of states,
- $I \subseteq S$ is a set of initial states,
- $R \subseteq S \times S$ is the transition relation between states,
- $L : S \to 2^{AP}$ is a labelling function that maps a state to a subset of atomic propositions. Here, AP is the set of all atomic propositions.

A (possibly infinite) sequence of states is called a *path*, which is denoted by w. Assuming that the indexing starts with 0, we write $w[i]$ for the $(i+1)$th state in w, and we write w^i for the sub-path starting from the $(i+1)$th state.

Definition II.2 (Semantics of LTL)**.** *The Kripke-style semantics of LTL is defined via a* forcing *relation $\models$ as below.*

$M, w \models \top$	*iff*	*always*
$M, w \models p$	*iff*	$p \in L(w[0])$
$M, w \models \neg A$	*iff*	$M, w \not\models A$
$M, w \models A \wedge B$	*iff*	$M, w \models A$ *and* $M, w \models B$
$M, w \models \mathbb{X}\ A$	*iff*	$M, w^1 \models A$
$M, w \models A\ \mathbb{U}\ B$	*iff*	*there exists* $i \geq 0$ *s.t.* $M, w^i \models B$, *and for all* $0 \leq j \leq i$, $M, w^j \models A$

We say the pair (M, w) of a model M and a path w force an atomic proposition p, written as $M, w \models p$, when p belongs to the subset of atomic propositions in the *first state* in w. The classical logic connectives are interpreted in the usual way. M, w force $\mathbb{X}\ A$ if and only if M and the path w^1 starting from the *next* state force A. In other words, A is true in the

next state. M, w force $A \mathbb{U} B$ if B is true in the future, and before that, A must be true.

Traditionally, the semantics of LTL is defined on infinite paths. However, this paper is focused on runtime verification, which usually concerns a finite sequence of states extracted from an execution instance. Consequently, runtime verification often adopts some variants of LTL, such as LTL on finite traces (FLTL) and three-valued LTL (LTL_3) [9]. This paper adopts FLTL with *strong* next. That is, $\mathbb{X} A$ is true when the next state exists and makes A true; otherwise, $\mathbb{X} A$ is false. In this semantics, $\mathbb{F} A$ is only true when there is a future state that makes A true; otherwise, it is false.

Past-time LTL (PTLTL) [10] is another useful language for specifying security-related properties [11]. PTLTL has two distinct temporal operators called *previously* ($\mathbb{P}$) and *since* ($\mathbb{S}$). Their semantics are defined on past state traces, which are symmetric to FLTL, which looks into the future. In PTLTL, $\mathbb{P} A$ is true when the previous state exists and makes A true; this is symmetric to $\mathbb{X} A$ in FLTL. $A \mathbb{S} B$ is true if 1) the current state makes B true, or if 2) B was true sometime in the past, and since then, A has been true. The semantics of $A \mathbb{S} B$ in PTLTL is symmetric to $A \mathbb{U} B$ in FLTL.

III. Our Approach

A. Overview of the Digital Twin system

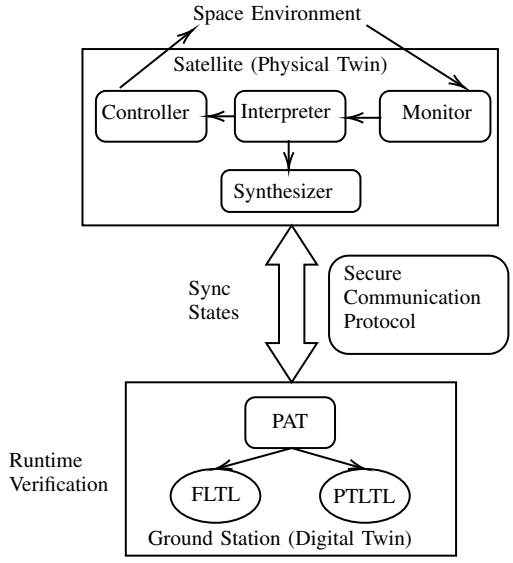

Fig. 1: An overview of the proposed digital twin run-time verification framework.

We present a simplified view of the overall system in Figure 1. The top element in Figure 1 is the physical twin, which runs on the actual space asset. The physical twin collects, monitors and interprets *engineering data* such as control and sensor raw data. This component will periodically synthesise engineering data into *scientific data* such as processed datasets, which in our framework contains the states to be synchronised and checked, and send them to the ground station.

The scientific data are transmitted to the ground station using our delay-tolerant communication protocol. Our protocol guarantees the correctness of the synchronised states and the security of transmitted information via cryptographic algorithms.

Finally, the ground station, which runs the digital twin, simulates the state of the satellite using processed data and performs computationally heavy tasks. More specifically, the digital twin models two essential aspects of the satellite: the physical behaviour, captured by sensor data, and the communications, captured by transmitted messages. We assume that even if the satellite is under attack, the sensors will still provide the correct data of the physical behaviours, and the system can still provide the correct log of event executions. In this work, we perform runtime monitoring and verification on the digital twin. In future, we will perform more complicated tasks such as static verification, AI-based planning and goal reasoning, model and plan update, etc. on the digital twin.

B. Secure Time Synchronisation for Digital Twins

Runtime verification for digital twins relies on correct (i.e., non-corrupted) state data. It is also crucial for the verifier to know the freshness of the state data (to decide if they are still available for current-time verification or be discarded). While the correctness of data can be achieved via standard data integrity/authentication techniques (e.g., message authentication codes and digital signatures), the latter is particularly challenging considering that the satellite communication suffers from unpredictable delays. Thus, we focus on establishing a proper level of time synchronisation among digital twins. The time synchronization (usually established by a protocol) between the satellites and the ground station relies on the communication network which is exposed to potential attacks. These attacks may sabotage the time synchronisation by preventing the packets being correctly transmitted, rendering inaccurate synchronization and further making runtime verifier receive and check an outdated or manipulated state. Thus, time synchronisation protocols must be sufficiently robust against active adversaries. Once an accurate time synchronization is established, the subsequent network packets containing state information can be time-stamped and then authenticated to allow the verifier to decide if the received state information is fresh for runtime verification. We note that we are not aiming for reducing network delays which is another technical problem and beyond the scope of this work. Our design is to ensure such delay can be detected and measured by the receiver so outdated packets can be identified, even in presence of active attacks.

Traditionally, time synchronization is implemented by the NTP (Network Time Protocol). However, it is not a direct option for us as the NTP does not offer a reasonable level of security against active attacks. An existing secure time sychronisation protocol called Authenticated Network Time Protocol (ANTP), designed by Dowling et al. [12] achieves properties close to our needs. ANTP consists of two phases. The client and the time server negotiate cryptographic algorithms and

Ground Station (server/digital twin) k for MAC		**Satellite** (client/physical twin) k for MAC
	Time synch phase $p = 1, \ldots, n$	
$state \leftarrow$ `in-progress` $r \leftarrow \{0,1\}^{256}$ $t_1 \leftarrow \mathsf{Now}()$ $m_1 \leftarrow t_1 \| r$	$\xrightarrow{m_1}$	$t_2 \leftarrow \mathsf{Now}()$ $\vdots$ (processing time or delay) $t_3 \leftarrow \mathsf{Now}()$ $\tau \leftarrow \mathsf{MAC}(k, m_1 \| t_1 \| t_2 \| t_3)$
	$\xleftarrow{m_2}$	$m_2 \leftarrow \tau \| t_1 \| t_2 \| t_3$
$t_4 \leftarrow \mathsf{Now}()$ $RTT \leftarrow (t_4 - t_1) - (t_3 - t_2)$ if $RTT > E$ then $state \leftarrow$ `reject` verify $\tau = \mathsf{MAC}(k, m_1 \| t_1 \| t_2 \| t_3)$ if verify fails then $state \leftarrow$ `reject` & abort $offset = (t_3 + t_2 - t_1 - t_4)/2$ $time_p \leftarrow \mathsf{Now}() + offset$ $state \leftarrow$ `accept` if $p = n$ then terminate		

TABLE I. Authenticated Time Synchronization Protocol for satellite digital twins.

keys in the first phase. Then a time synchronization process is run in which the message flows are authenticated (using message authentication codes). No encryption is performed. ANTP comes with a security proof in the well-established Bellare-Rogaway model [13]. However, the cryptographic negotiation of ANTP needs separate implementations and as it is different from the maturely deployed TLS handshake protocol.

In our scenario, authenticity is the major concern when executing time synchronization. That is, at the completion of the time synchronization protocol, the client and the time server can confirm that they have been synchronized in terms of time with each other (rather than someone else). We see confidentiality as a less important requirement for the time synchronization protocol - the attackers are assumed to know that the time is being synchronized and the time difference between the client and server[1].

With this consideration, we adopt the ANTP protocol which uses message authentication codes (MACs) as the only cryptographic tool to provide authenticity. Furthermore, we greatly simplify the cryptographic algorithm and key establishment of the ANTP protocol by simply specifying the MAC algorithm storing the keys the MAC in the satellite and the ground station. In Table I, we give the description of the time synchronization protocol, which is based on [12]. We write $||$ for the concatenation of messages. We note that, the first phase of ANTP is to establish a MAC algorithm and a shared secret key. Here we assume the MAC algorithm and the key is securely pre-established. So, the security and correctness of the protocol follows directly from ANTP.

[1]The actual time difference between the client and server can be caused by the time used to transfer the time synchronization protocol packets. It is mostly determined by the physical distance between the satellite and ground station which is generally known by the attacker.

The time synchronization protocol starts with a pre-specified message authentication code MAC and pre-shared key k. The variable $state$ represent the state of the client in terms of the time synchronization protocol. The procedure $\mathsf{Now}()$ returns the participant's current time. RTT denotes the total round-trip delay that the satellite observes. This can be bounded by the desired accuracy E. $time_p$ stands for the time of the satellite after the synchronization. Here we can use any secure message authentication code for MAC which given a secret key and message deterministically produce a tag, e.g., CBC-MAC.

We emphasise that the value $offset$ approximately represents the network delays and can always be checked by the protocol initiator (e.g., ground station), allowing it to decide the freshness of current data. The delta of our time synchronisation process, (i.e., the actual time difference between the ground station and satellite), is defined as Δ. Δ is bounded by the propagation time from the satellite to the ground station θ_1, and the propagation time from the ground station to the satellite θ_2. According to [12], $\Delta \leq \frac{1}{2} \cdot |\theta_1 - \theta_2| \leq RTT$ where the round-trip delay $RTT = (t_4 - t_1) - (t_3 - t_2)$. Δ therefore defines the freshness of the state of the digital twins being verified, i.e., the state that is being verified is at earliest $t_c - \Delta$. Depending on the communication distance, Δ can vary from around a few microseconds to dozens of minutes when considering the communication from earth to low-earth orbits (as low as 200km) to deep space (up to 30,000km). According to the protocol, we can manually set the RTT bound E to fail the synchronization when RTT grows to too large.

Finally, we note that malicious activities that 1) modify packets containing t_1, t_2, t_3 will results abortion of the protocol thanks to the MAC, and 2) delay the packets will not make Δ exceed RTT (and thus E) without causing the abortion of the protocol.

C. Runtime Verification

The secure time synchronisation protocol protects against malicious entities that are not part of the digital twin system. However a compromised satellite can send incorrect state information. Runtime verification can be used to confirm the behaviour of the satellite by verifying state information.

We propose a runtime verification framework that supports both FLTL and PTLTL in one package and is driven by the model checker Process Analysis Toolkit (PAT) [6], [14]. Although model checking is considered a harder problem than runtime verification, it is not unprecedented to use model checking to solve runtime verification problems. For example, Kejstová et al. [15] proposed a novel idea of adapting a model checker to perform precise runtime verification for a full-stack consisting of an operating system, libraries, and programs. In the same vein, we propose to take advantage of the advanced algorithms built in PAT and develop an extension that can check different languages in runtime verification.

a) Modelling: In our satellite application, we are mainly interested in verifying properties over the state variables of the system. Let us name the state variables $var_1, var_2, \cdots$. A state S is simply a snapshot of the values of state variables, i.e., $S ::= \{var_1 = val_1, var_2 = val_2, \cdots\}$. In PAT, we model a state via a process in Communicating Sequential Processes [16] with C# (CSP#) [6]. The process performs variable assignments as below.

```
S() = s{var1 = val1; var2 = val2; ...} -> Skip;
```

A final trace T is a sequence of states, modelled as below.

```
T() = S1(); S2();...
```

The user can define properties over state variables. For example, the below code defines a proposition that states "$var1$ is not 0."

```
#define v1Safe (var1 != 0);
```

We can then use PAT to check a safety property that "var_1 should never be 0" using the temporal modality $\mathbb{G}$, which is written as `[]` in PAT.

```
#assert Trace() |= [] v1Safe;
```

b) Verification: The foundation of our runtime verification framework is based on the observation that verifying LTL with finite traces in PAT corresponds to verifying FLTL with strong next/future.

Lemma III.1. *Let T be a finite state trace, M_T be a model of T in CSP# defined above, and P an LTL property. $T \models P$ in FLTL with strong next/future if and only if $M_T \models P$ returns positive in PAT.*

Moreover, based on the observation of symmetry between FLTL and PTLTL in Section II-C, we can simply reverse the trace and check the symmetric properties in PTLTL. We denote $\overline{T}$ the reversed trace of T. That is, if $T = S_1; S_2; S_3$, then $\overline{T} = S_3; S_2; S_1$.

Lemma III.2. *Let T be a finite state trace, $M_{\overline{T}}$ be a model of $\overline{T}$ in CSP# defined above, and A, B be propositions. $T \models \mathbb{P}\ A$ in PTLTL if and only if $M_{\overline{T}} \models \mathbb{X}\ A$ returns positive in PAT. $T \models A\ \mathbb{S}\ B$ in PTLTL if and only if $M_{\overline{T}} \models A\ \mathbb{U}\ B$ returns positive in PAT.*

Therefore, we can integrate runtime verification of both FLTL with strong next/future and PTLTL into one verification framework driven by PAT. A similar form of bidirectional temporal logic verification has been realised in tools such as Formula Builder [17]. The above result further allows integration with more complex modelling languages, such as (Probabilistic) CSP#, that are supported by the PAT verification engine.

c) Security Properties: The security properties that this paper considers are integrity and authentication. The integrity property ensures that the operation of the satellite remains correct during a specified period of time. Runtime verification will verify that current states are within specified tolerances for sensor readings. Formally, the cases of integrity we consider are generally in the following form: "since the *pre-condition* holds, the state variable should hold a value within a certain range until the *post-condition* holds". This can be expressed as two formulae, one in PTLTL and the other in FLTL, as below, where pre stands for the pre-condition, $post$ the post-condition, l the lower-bound of the variable, and u the upper-bound of the variable.

$$(l \leq v \leq u)\ \mathbb{S}\ pre$$
$$(l \leq v \leq u)\ \mathbb{U}\ post$$

The authentication property ensures that the communications from the ground station are valid. For the authentication property, we will model signal strength, terrestrial location, and cryptographic verification. The combination of these features is used to determine whether the satellite is communicating with the correct target. The instances of such a property take the following form, where sig is a variable for signal strength, x, y, z are location coordinates of the target, l_i, u_i, $1 \leq i \leq 4$ are upper bounds and lower bounds of variables, $verified$ is a Boolean variable that indicates whether the target signature has passed the cryptographic verification in our synchronisation protocol, and $connect$ is a Boolean variable that indicates that the communication connection is established:

$$connect \rightarrow$$
$$\mathbb{P}\ ((l_1 \leq x \leq u_1) \wedge (l_2 \leq z \leq u_2) \wedge (l_3 \leq y \leq u_3) \wedge$$
$$(l_4 \leq sig \leq u_4) \wedge verified)$$

d) Monitoring: We have developed an extension of PAT that provides the following interfaces to the satellite digital twins system:

- `init`: initialise the monitor and set up state variables.
- `addState`: adds a new state to the trace.
- `check`: automatically constructs a CSP# model of a finite state trace and calls PAT to verify the model. This interface has two modes: one checks FLTL properties and the other checks PTLTL properties.

Since the CSP# model is a linear sequence of states, there are no branches in the state-space, and therefore we do not have the state-explosion problem of model checking. PAT can usually return the verification result in a second, which is fast enough for our purpose. The above interface is then integrated into our digital twin system for continuous security monitoring and verification.

IV. Conclusion and Future Work

In this paper, we have discussed an approach that combines digital twins and runtime verification to provide trustworthy and secure communication for satellites. Building on top of this protocol, we design a digital twins system for satellites with a focus on security monitoring and verification. Finally, we develop a runtime verification framework based on PAT that supports multiple temporal logics. The above form a self-contained system that provides enhanced security for satellites and other space assets.

Equipped with the digital twin system of this paper, we plan to perform penetration testing and attack simulation for satellites in the future. We will generate a large amount of data while performing penetrating testing, including the status of the system and the parameters of the attacker and the defender (called agents in what follows). From the data, we can use clustering to obtain the states of the agents and learn how their states evolve over time. With the states and their transitions, we will model the behaviour of agents in Markov decision processes and use reinforcement learning to train the agents towards optimal policies: most efficient hacks for the attacker and best counteractions for the defender. With the AI-based simulations, we expect to check more corner cases that might have been missed by human attackers.

Part of the reason why we chose to build the runtime verifier on PAT instead of using an existing one or developing one from scratch is that our bigger plan is to formally verify the entire satellite communication system, which includes satellite behaviour, synchronisation protocol, digital twins system, among others, and PAT will be used to model and verify many components of the system. In particular, we plan to describe the correct behaviour of space assets in Petri-nets [18] and then model the Petri-nets in PAT. We will then express desired properties of the system as reachability, deadlock-freeness, liveness, or temporal logic formulae and then verify those properties in PAT. With PAT as a central verification engine, we can also extend this work with other temporal logic variants and even customised languages that suit our specific application requirements. We will adopt theorem provers such as Isabelle/HOL when an exhaustive check is not suited, and inductive reasoning is required. To ensure that the new key establishment protocol and end-to-end secure communication protocol achieve the security properties for the deep-space DTN environment, we will formally prove the protocol using a provable security framework [19], [20] commonly used to evaluate cryptographic protocols. Our goal is to provide a fully verified execution stack for satellites and other space assets such as rovers.

The final test of our approach includes running the proposed framework in a simulation environment based on Gilmore Space Technologies' Electrical Ground Support Equipment satellite simulator. The new simulation environment will reimplement both the satellite and the digital twin, as well as their communication methods. Once the final tests are passed, we plan to launch a test satellite with Gilmore Space Technologies in mid-2022.

References

[1] Gaisler, "Leon5 processor," https://www.gaisler.com/index.php/products/processors/leon5, 2021.

[2] Z. Hóu, D. Sanán, A. Tiu, Y. Liu, K. C. Hoa, and J. S. Dong, "An isabelle/hol formalisation of the SPARC instruction set architecture and the TSO memory model," *JAR*, vol. 65, no. 4, pp. 569–598, 2021.

[3] A. Goldberg, K. Havelund, and C. McGann, "Runtime verification for autonomous spacecraft software," in *2005 IEEE Aerospace Conference*, 2005, pp. 507–516.

[4] F. Flammini, "Digital twins as run-time predictive models for the resilience of cyber-physical systems: a conceptual framework," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 379, no. 2207, p. 20200369, 2021.

[5] J. Leng, D. Wang, W. Shen, X. Li, Q. Liu, and X. Chen, "Digital twins-based smart manufacturing system design in industry 4.0: A review," *Journal of Manufacturing Systems*, vol. 60, pp. 119–137, 2021.

[6] J. Sun, Y. Liu, J. S. Dong, and J. Pang, "Pat: Towards flexible verification under fairness," ser. Lecture Notes in Computer Science, vol. 5643. Springer, 2009, pp. 709–714.

[7] E. Bartocci and Y. Falcone, *Lectures on Runtime Verification: Introductory and Advanced Topics*. Springer, 2018, vol. 10457.

[8] A. Pnueli, "The temporal logic of programs," in *18th Annual Symposium on Foundations of Computer Science*. IEEE, 1977, pp. 46–57.

[9] A. Bauer, M. Leucker, and C. Schallhart, "Comparing LTL semantics for runtime verification," *J. Log. and Comput.*, vol. 20, no. 3, p. 651–674, Jun. 2010.

[10] A. Bauer, R. Goré, and A. Tiu, "A first-order policy language for history-based transaction monitoring," in *Theoretical Aspects of Computing - ICTAC 2009*, M. Leucker and C. Morgan, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 96–111.

[11] X. Du, A. Tiu, K. Cheng, and Y. Liu, "Trace-length independent runtime monitoring of quantitative policies," *IEEE Trans. Dependable Secur. Comput.*, vol. 18, no. 3, pp. 1489–1510, 2021.

[12] B. Dowling, D. Stebila, and G. Zaverucha, "Authenticated network time synchronization," in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, 2016, pp. 823–840.

[13] M. Bellare and P. Rogaway, "Entity authentication and key distribution," in *Annual international cryptology conference*. Springer, 1993, pp. 232–249.

[14] H. Bride, C. Cai, J. S. Dong, R. Goré, Z. Hóu, B. P. Mahony, and J. McCarthy, "N-PAT: A nested model-checker - (system description)," in *International Joint Conference on Automated Reasoning, IJCAR 2020, Paris, France, July 1-4, 2020, Proceedings, Part II*, 2020, pp. 369–377.

[15] K. Kejstová, P. Ročkai, and J. Barnat, "From model checking to runtime verification and back," in *Runtime Verification*, S. Lahiri and G. Reger, Eds. Cham: Springer International Publishing, 2017, pp. 225–240.

[16] C. A. R. Hoare, "Communicating sequential processes," *Commun. ACM*, vol. 21, no. 8, p. 666–677, 1978.

[17] S. Jörges, T. Margaria, and B. Steffen, "Formulabuilder: A tool for graph-based modelling and generation of formulae," in *Proceedings of the 28th International Conference on Software Engineering*, ser. ICSE '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 815–818.

[18] C. A. Petri and W. Reisig, "Petri net," *Scholarpedia*, vol. 3, no. 4, p. 6477, 2008, revision #91647.

[19] O. Goldreich, *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*, ser. Algorithms and Combinatorics. Springer, 1998, vol. 17.

[20] J. Katz and Y. Lindell, *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014. [Online]. Available: https://www.crcpress.com/Introduction-to-Modern-Cryptography-Second-Edition/Katz-Lindell/p/book/9781466570269

APPENDIX

Lemma A.1. *Let T be a finite state trace, M_T be a model of T in CSP# defined above, and P an LTL property. $T \models P$ in FLTL with strong next/future if and only if $M_T \models P$ returns positive in PAT.*

Proof. By inspection of the semantics. In particular, if $T \models \mathbb{X}\ A$, then the next state must exist and make A true, in which case $M_T \models \mathbb{X}\ A$ should return positive in PAT. If the next state does not exist or if it exists but does not make A true, then $T \not\models \mathbb{X}\ A$ and $M_T \models \mathbb{X}\ A$ should return negative. Similarly, if there is not a future state in the current trace that makes A true, then $T \not\models \mathbb{F}\ A$, and $M_T \models \mathbb{F}\ A$ should return negative in PAT. Otherwise, $T \models \mathbb{F}\ A$, and $M_T \models \mathbb{F}\ A$ should return positive in PAT. The semantics of other language components can be checked in a similar way. □

Lemma A.2. *Let T be a finite state trace, $M_{\overline{T}}$ be a model of $\overline{T}$ in CSP# defined above, and A, B be propositions. $T \models \mathbb{P}\ A$ in PTLTL if and only if $M_{\overline{T}} \models \mathbb{X}\ A$ returns positive in PAT. $T \models A\ \mathbb{S}\ B$ in PTLTL if and only if $M_{\overline{T}} \models A\ \mathbb{U}\ B$ returns positive in PAT.*

Proof. Based on the symmetry of the semantics of FLTL with strong next/future and PTLTL. For simplicity, we only consider next and until as first-class temporal operators in FLTL and previous and since as first-class temporal operators in PTLTL. $T \models \mathbb{P}\ A$ in PTLTL is equivalent to $\overline{T} \models \mathbb{X}\ A$ in FLTL with strong next — they both require that the previous/next state exists in the trace and makes A true.

$T \models A\ \mathbb{S}\ B$ in PTLTL is equivalent to $\overline{T} \models A\ \mathbb{U}\ B$ in FLTL with strong next. There are two cases: 1) B holds in the current state, then both $T \models A\ \mathbb{S}\ B$ and $\overline{T} \models A\ \mathbb{U}\ B$. 2) B holds in the past in T, which means that B holds in the future in $\overline{T}$. In this case, $T \models A\ \mathbb{S}\ B$ requires that A holds in every state from the state when B holds to the current state; and $\overline{T} \models A\ \mathbb{U}\ B$ requires that A holds from the current state until the point where B holds. Clearly, $T \models A\ \mathbb{S}\ B$ in PTLTL iff $\overline{T} \models A\ \mathbb{U}\ B$ in FLTL with strong next/future. In all other cases, both formulae do not hold in the respective logic.

The other formulae can be checked similarly. The cases for classical logic formulae are straightforward as the semantics is the same in both logics.

Finally, let F be a formula in PTLTL, and $\overline{F}$ be the corresponding symmetric formula in FLTL. By Lemma A.1, $\overline{T} \models \overline{F}$ in FLTL with strong next/future iff $M_{\overline{T}} \models \overline{F}$ returns positive in PAT, and by the above arguments, these are equivalent to $T \models F$ in PTLTL. □