Information systems. Computer sciences. Issues of information security
Информационные системы. Информатика. Проблемы информационной безопасности

UDC 004.415.53
https://doi.org/10.32362/2500-316X-2026-14-2-17-28
EDN XDBAIA

RESEARCH ARTICLE

Architecture of a distributed system for testing Internet of Things devices at the development stage

Danila S. Belyakov @

Tomsk State University of Control Systems and Radioelectronics, Tomsk, 634050 Russia
@ *Corresponding author, e-mail: cauze4concern@yandex.ru*

• **Submitted:** 11.07.2025 • **Revised:** 15.10.2025 • **Accepted:** 06.02.2026

Abstract

Objectives. The paper sets out to develop an architecture for a distributed testing system for Internet of Things (IoT) devices to ensure secure transmission and the isolated execution of test scenarios on dedicated execution modules. The study takes account of the rapid growth in the number of IoT devices operating in untrusted computing environments, in which the testing process can pose a risk of confidential data leakage or unauthorized interference with software components.

Methods. A comparative analysis of existing solutions such as *NI TestStand*, *MagicDAQ*, *PHiLIP*, and *KEOLABS ContactLAB* was conducted. Architectural components and test scenario life-cycle processes were examined and compared.

Results. The analysis identified the main stages of the test scenario life cycle, including preparation and storage of scripts, transmission and interpretation, interaction with the device under test, as well as registration and analysis of results. In addition, existing and proposed architectural solutions were compared according to the following key characteristics: application domain; type of architecture (distributed or centralized); test scenario execution environment; system scalability; level of execution isolation; availability of protection mechanisms; capability for remote management. The results of the study are presented in the form of a proposed architecture that includes a control module and autonomous execution modules with an isolated virtual MicroPython environment. To ensure security, test scenarios are transmitted over an encrypted communication channel using constrained application protocol and datagram transport layer security (protocol, while the execution of test code takes place in a restricted environment isolated from the main operating system.

Conclusions. The comparative analysis confirmed that the proposed solution eliminates the key limitations of existing solutions, namely the lack of encryption mechanisms and isolation of execution. The developed architecture enhances the security and reliability of the IoT device testing process, offering protection for intellectual property and test scenario logic in untrusted computing environments.

Keywords: Internet of Things, IoT, functional testing, testing architecture, test scenarios

For citation: Belyakov D.S. Architecture of a distributed system for testing Internet of Things devices at the development stage. *Russian Technological Journal.* 2026;14(2):17–28. https://doi.org/10.32362/2500-316X-2026-14-2-17-28, https://www.elibrary.ru/XDBAIA

Financial disclosure: The author has no financial or proprietary interest in any material or method mentioned.

The author declares no conflicts of interest.

НАУЧНАЯ СТАТЬЯ

Архитектура распределенной системы тестирования устройств интернета вещей на этапе их разработки

Д.С. Беляков @

Томский государственный университет систем управления и радиоэлектроники, Томск, 634050 Россия
@ *Автор для переписки, e-mail: cauze4concern@yandex.ru*

• **Поступила:** 11.07.2025 • **Доработана:** 15.10.2025 • **Принята к опубликованию:** 06.02.2026

Резюме

Цели. Цель работы заключается в разработке архитектуры распределенной системы тестирования устройств интернета вещей (Internet of Things, IoT), обеспечивающей защищенную передачу тестовых сценариев и их изолированное исполнение на исполнительных модулях. Актуальность исследования обусловлена стремительным ростом числа IoT-устройств, функционирующих в недоверенных вычислительных средах, где процесс тестирования может создавать риски утечки конфиденциальных данных или несанкционированного вмешательства в программное обеспечение.

Методы. Проведен сравнительный анализ существующих решений, таких как *NI TestStand*, *MagicDAQ*, *PHiLIP* и *KEOLABS ContactLAB*. Выполнено сопоставление их архитектурных компонентов и процессов жизненного цикла тестовых сценариев.

Результаты. На основании анализа выделены основные этапы жизненного цикла, на которых применяются рассмотренные инструменты: подготовка и хранение, передача и интерпретация, взаимодействие с тестируемым устройством, регистрация и анализ результатов. Кроме того, проведено сравнение существующих и предложенного архитектурных решений по ключевым характеристикам: предметная область применения, тип архитектуры (распределенная или централизованная), среда исполнения тестовых сценариев, масштабируемость системы, уровень изоляции среды исполнения, наличие механизмов защиты и возможность удаленного управления. Результаты работы представлены в виде предложенной архитектуры, включающей управляющий модуль и автономные исполнительные модули с изолированной виртуальной средой исполнения MicroPython. Для обеспечения безопасности предусмотрена передача тестовых сценариев по зашифрованному каналу связи с использованием протоколов CoAP[1] и DTLS[2] , а также выполнение кода тестовых сценариев в ограниченной среде, изолированной от основной операционной системы.

[1] Constrained application protocol – облегченный протокол интернета вещей.
[2] Datagram transport layer security – протокол передачи данных, обеспечивающий защищенность соединений для протоколов, использующих датаграммы.

Выводы. Проведенный сравнительный анализ продемонстрировал, что предлагаемое решение устраняет ключевые ограничения аналогов, связанные с отсутствием механизмов шифрования и изоляции исполнения. Разработанная архитектура повышает безопасность и надежность процесса тестирования IoT-устройств и может использоваться в недоверенных вычислительных средах для защиты интеллектуальной собственности и логики тестовых сценариев.

Ключевые слова: интернет вещей, IoT, функциональное тестирование, архитектура тестирования, тестовые сценарии

Для цитирования: Беляков Д.С. Архитектура распределенной системы тестирования устройств интернета вещей на этапе их разработки. *Russian Technological Journal*. 2026;14(2):17–28. https://doi.org/10.32362/2500-316X-2026-14-2-17-28, https://www.elibrary.ru/XDBAIA

Прозрачность финансовой деятельности: Автор не имеет финансовой заинтересованности в представленных материалах или методах.

Автор заявляет об отсутствии конфликта интересов.

INTRODUCTION

The widespread use of Internet of Things (IoT) devices requires compliance with high quality, safety, and reliability standards. In these circumstances, testing is crucial for the identification and elimination of potential malfunctions to ensure that the devices meet the specified technical requirements and can function stably in real operating conditions [1, 2].

Unlike traditional software, testing IoT devices presents a number of specific challenges. Since encompassing both hardware and software components, such testing involves the consideration of various communication protocols, limited computing resources, and the characteristics of low-level interfaces. An additional complication arises due to testing often being outsourced to third-party organizations that may not be trustworthy, which creates risks of unauthorized interference and compromised results [3].

In this regard, the paper aims to analyze existing solutions for testing IoT devices and to develop a new distributed testing system architecture that addresses the identified shortcomings.

1. THEORETICAL BASIS OF TESTING

Based on the specific characteristics of IoT devices, testing can be divided into three main levels [4–6]: device, network, and system testing.

Device testing involves checking the device itself as a separate unit. This covers both the hardware components (e.g., the microcontroller, sensors and actuators) and the embedded software. The first stage involves verifying the correct operation of the peripheral interfaces (serial peripheral interface (SPI), universal asynchronous receiver-transmitter (UART), etc.) that facilitate communication between the microcontroller and external modules. Next, the device's firmware logic, data processing algorithms, performance, and reliability under load or interference are evaluated.

Network testing focuses on the communication infrastructure. The measured parameters include the bandwidth, latency, reliability, and scalability of the communication channels used (e.g., Wi-Fi, cellular networks, low-power wide-area network, etc.). This ensures the required quality of service for IoT devices [7].

On the other hand, system testing covers the entire range of devices and infrastructure, carrying out checks into how several devices interact with each other, as well as with gateways and the cloud platform. Here it is also necessary to ascertain the correctness of end-to-end data processing and user scenarios. This comprehensive integration testing involves working through scenarios from an event on a device to the receipt of data in the cloud and the return response.

Therefore, to comprehensively assess the functioning of IoT systems, thorough testing must be conducted at each level to cover various aspects of their operation, ranging from hardware components to device interaction within a distributed environment.

2. ARCHITECTURAL ANALYSIS OF EXISTING SOLUTIONS

Testing tools are software or hardware solutions designed to automate and streamline the testing of IoT devices. They support specialists in evaluating various characteristics, including functionality, performance, stability and security. These tools are used to to simulate a wide range of operating scenarios and environmental conditions, thereby ensuring more reliable testing of device behavior in various conditions.

The majority of contemporary research and development in the realm of IoT testing revolves around system- or network-level testing. The most common solutions are those that verify the correct implementation of protocols (constrained application

protocol, MQTT[3], 6LoWPAN[4], etc.), node compatibility, performance, and network connection reliability. Platforms such as PatrIoT [8], Hector [9], F-Interop [10], MATTER [11], and Eclipse IoT-Testware [12] typically use virtualization mechanisms to facilitate scalable testing of architectural scenarios without the involvement of real physical equipment.

Checks implemented at the physical device level are usually limited to highly specialized solutions and are less formalized. Therefore, it seems appropriate to consider system architectures specifically oriented towards the device level and compare them. The basic architecture of the testing system shown in Fig. 1 includes the following components:

- The control module coordinates the launch of tests and distributes tasks among the execution modules.
- Executive modules are one or more devices that interact directly with the device under test through physical interfaces.
- The device under test is the object being tested.

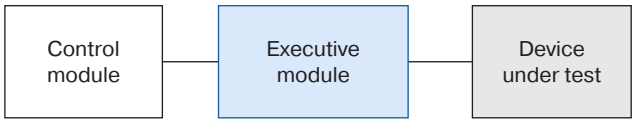

Fig. 1. Typical architecture of a testing system

Subject analysis is carried out to identify the tools used at the device testing level.

NI TestStand[5] is a commercial test management system that integrates with the *NI LabVIEW* measurement programming environment. The corresponding system architecture is depicted in Fig. 2.

The control module, which is installed on a PC, is designed for centralized control of automated execution modules. It also records test results to ensure their collection and visualization.

Executive modules are data collection systems connected to the control module that interact directly with the device under test via physical interfaces to execute test commands.

The script execution environment is located directly within the operating system (OS) on the control module. Built-in OS functions provide system security by controlling resources and test results.

The *NI TestStand* functional capabilities include creating test scenarios that can be executed for a single device or multiple devices simultaneously. Testing is coordinated through parallel interaction between several control modules via a local network. However, this interaction is currently limited to internal enterprise or laboratory networks, as the system does not initially support full remote control via an internet connection. Nevertheless, *NI TestStand* provides a command interface for initiating testing from external continuous integration (CI) systems.

Thus, *NI TestStand* is a local, centralized testing system offering a sophisticated user interface and support for continuous integration processes. However, it lacks built-in mechanisms for distributed processing or the isolated operation of test modules.

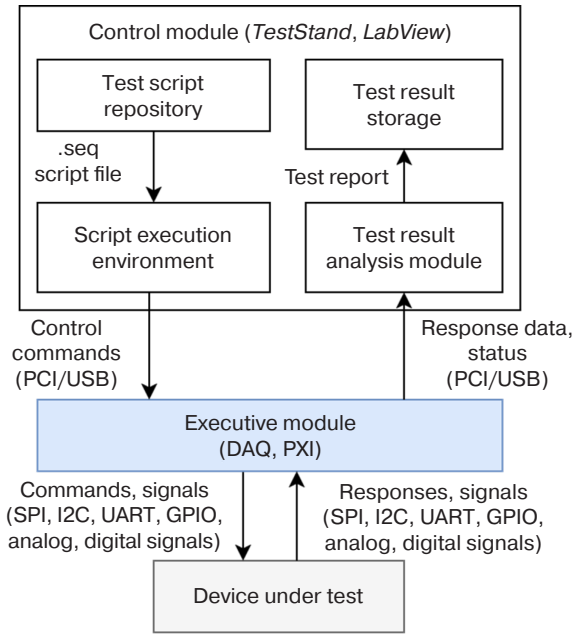

Fig. 2. *NI TestStand* testing architecture for a single device. PCI (Peripheral Component Interconnect) is an input/output bus for connecting peripheral devices to a computer motherboard; USB (Universal Serial Bus) is a universal serial bus; I2C (Inter-Integrated Circuit) is a serial asymmetric bus; GPIO (General Purpose Input/Output) is a general purpose input/output interface; DAQ (Data Acquisition) is software and hardware for collecting data; and PXI (PCI eXtensions for Instrumentation) is a PCI extension for measurement systems

MagicDAQ[6] is an executive module that connects to the control module (PC) via a USB interface (Fig. 3). All testing logic is implemented on the control module by executing test scripts in Python, which then send commands to the executive module via the USB connection. It should be noted that *MagicDAQ* is not a completely distributed system or cloud service;

[3] Message queuing telemetry transport is a lightweight, publish–subscribe, machine-to-machine network protocol for message queue/message queuing service.

[4] IPv6 over Low-Power Wireless Personal Area Networks, based on the IEEE 802.15.4 standard.

[5] TestStand Release Notes. https://www.ni.com/en/support/documentation/release-notes/product.teststand.html. Accessed July 07, 2025.

[6] *MagicDAQ* Docs. https://magicdaq.github.io/magicdaq_docs/. Accessed July 07, 2025.

rather, it is a peripheral module for a local PC, whose scalability is determined by the number of available USB ports. As test scripts are written in Python and transmitted via a local connection, security is implemented using built-in operating system tools.

Thus, *MagicDAQ* is characterized as a specialized solution for local use in measurement systems with limited scalability and no distributed data processing mechanisms.

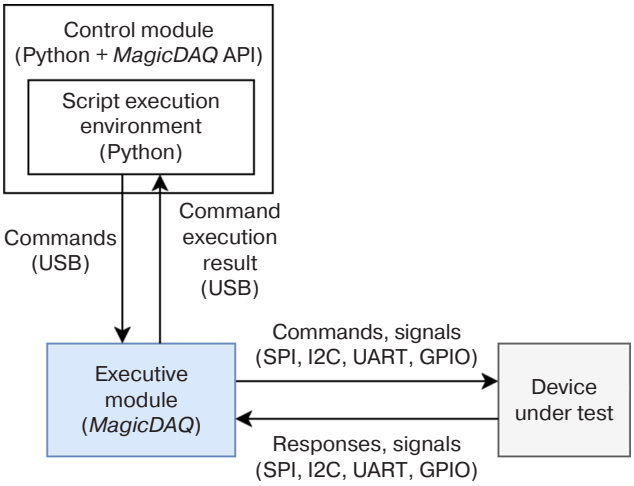

Fig. 3. *MagicDAQ* testing architecture.
API is Application Programming Interface

A software and hardware platform known as *PHiLIP* [8] has been developed for the automated testing of microcontroller peripheral interfaces in embedded systems, including IoT devices.

The *PHiLIP* testing system architecture is based on a distributed model, which provides a central control module in the form of a continuous integration server (such as Jenkins) integrated with the Git version control system for storing and updating test scripts. As illustrated in Fig. 4, the control module coordinates test runs, stores, and distributes test cases, as well as transferring firmware to the devices under test and collecting test result data for further analysis.

Each executive module, which is based on a Raspberry Pi single-board computer (manufactured by Raspberry Pi Holdings plc, United Kingdom), functions as an intermediary between the CI server and the *PHiLIP* hardware interface module. The module runs test scenarios in the Robot Framework environment[7] and uses serial wire debug or joint test action group interfaces to flash the tested devices and download new software versions. The *PHiLIP* interface module is connected to via the UART interface.

In turn, the *PHiLIP* interface module is responsible for generating and registering signals on the device under test's physical interfaces including SPI, I2C, UART, and GPIO. This module emulates the behavior of the device's external environment and records its responses, which are then returned to the execution module and subsequently to the control module for test result analysis.

As shown in Fig. 5, the system architecture allows for scaling: several executive modules can be connected to one control module, each executive module controlling its own instance of the interface module and the device under test. This structure enables parallel testing of different devices while providing centralized control and collection of results. However, [8] does not describe the mechanisms for ensuring information security, nor does it specify how test case execution is isolated, how protection against unauthorized access is ensured, or how integrity control is maintained during the transmission of test scenarios.

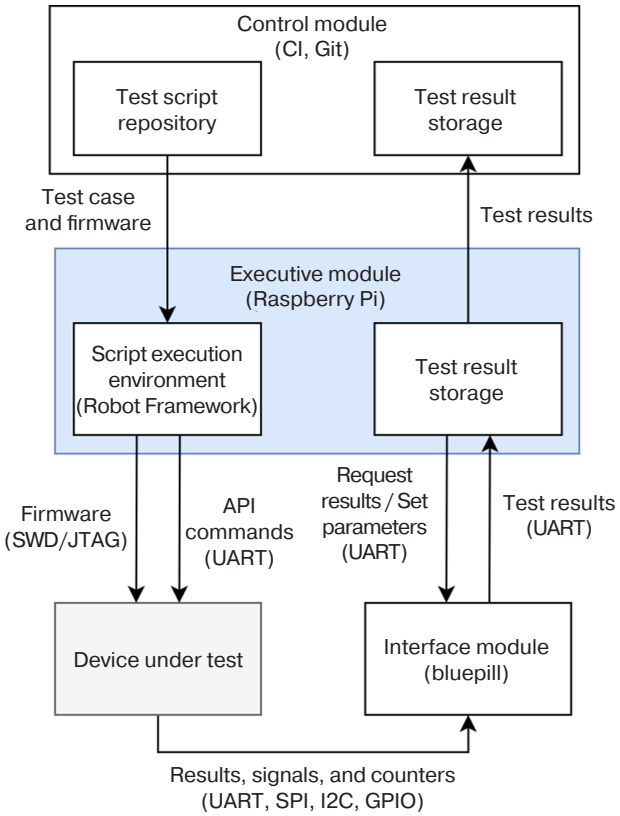

Fig. 4. *PHiLIP* testing architecture
for a single device under test

KEOLABS ContactLAB[8] is a commercial software and hardware platform designed for functional testing of smart cards, microcontrollers

[7] Robot Framework. https://robotframework.org/. Accessed July 07, 2025.

[8] Contact Tester. https://www.keolabs.com/products/platforms/contact-tester. Accessed July 07, 2025.

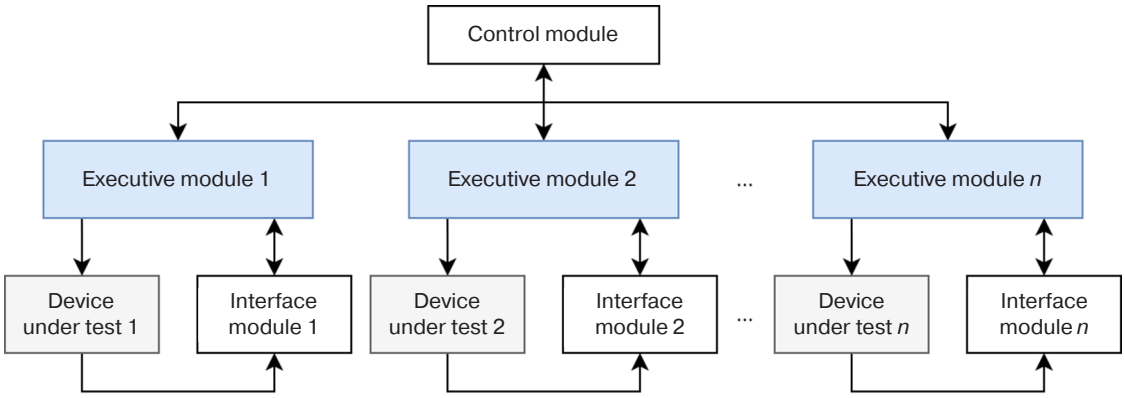

Fig. 5. *PHiLIP* test architecture for multiple devices under test

with security modules (Secure Element), as well as devices supporting NFC[9] and ISO 7816[10] contactless interfaces. It is widely used for certification testing and the development of authentication and identification tools.

The architecture of the solution (Fig. 6) is based on the use of two modules: the *SCRIPTIS*[11] software control module, which is installed on a PC, and the *ContactLAB HW* hardware execution module. The latter generates commands, measures time characteristics, and records signals from the device under test. The control and execution modules interact via a USB or Ethernet interface. Testing is performed entirely at a single workstation with no option for distributed testing or remote access. With this solution, the only way to achieve scalability is by increasing the number of stands and manually coordinating their operation. The architecture does not allow for a centralized control module or the parallel execution of scenarios on multiple devices. Security of the testing logic is ensured within the closed *SCRIPTIS* environment and its built-in access mechanisms. Test scenarios are executed in a standard user OS environment.

Therefore, *KEOLABS* is a platform for the low-level testing of secure devices in a laboratory environment. Its architecture is designed for centralized operation and lacks mechanisms for remote access or secure distributed test execution.

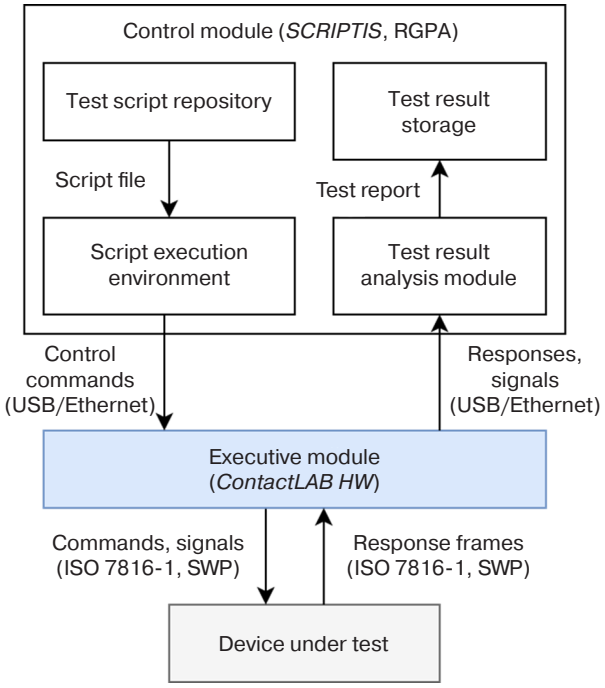

Fig. 6. *KEOLABS ContactLAB* testing architecture. RGPA (Real-Time General-Purpose Analyzer); SWP (Single Wire Protocol)

3. ANALYSIS OF TESTING PROCESSES

A number of typical processes can be identified based on the review of existing testing system architectures. As shown in Fig. 7, these processes provide a general overview of how such systems function, regardless of the specific implementation. These processes can be defined as a sequence of stages in the test scenario lifecycle:

1) preparation and storage of test scenarios;
2) planning and launching of test scenarios;
3) transfer of test scenarios to the execution module;
4) interpretation of test scenarios;
5) interaction with the object under test;
6) recording of test results;
7) analysis and generation of reports.

[9] Near Field Communication (NFC) is a technology for transmitting data wirelessly over short distances.

[10] ISO/IEC 7816-1 Identification cards—Integrated circuit cards. https://www.iso.org/standard/54089.html. Accessed July 07, 2025.

[11] *SCRIPTIS*: an intuitive testing environment. https://www.keolabs.com/products/solutions/emvco-l1-payment-testing#. Accessed July 07, 2025.

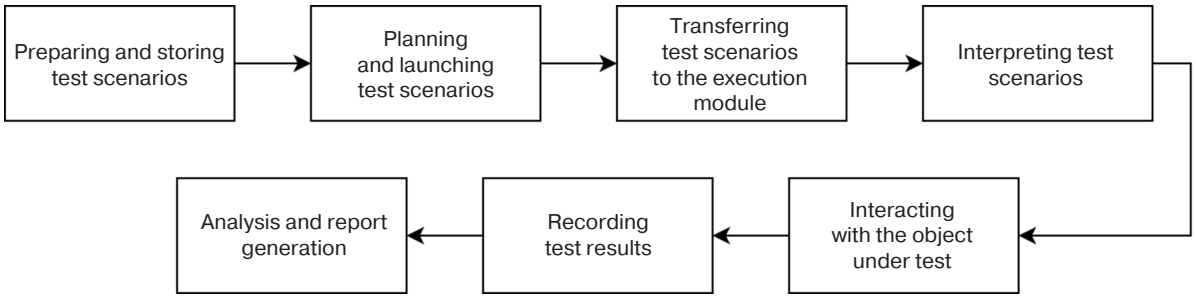

Fig. 7. Test scenario life cycle

During the preparation and storage of test scenarios, the appropriate sequence of actions, execution conditions and criteria for successful test completion are developed and described. These are created by specialists using scripting languages such as Python (e.g., in *MagicDAQ*) and Robot Framework (in *PHiLIP*), as well as other tools. Once developed, the scenarios are stored in a database or repository for subsequent use by the control module or continuous integration systems (e.g., Jenkins).

Test scenario planning and execution involves selecting the necessary tests and determining their execution sequence, as well as synchronizing and initiating the testing process. This can be performed either by an operator through the control module interface or automatically from an external continuous integration system, as with *PHiLIP* and Jenkins.

Test scripts are transferred to the execution module of the systems under review via a local network (e.g., *PHiLIP*) or physical interfaces, such as USB (e.g. *NI TestStand*, *MagicDAQ*, and *KEOLABS ContactLAB*) and Ethernet (*KEOLABS ContactLAB*). However, the analyzed solutions do not specify whether any encryption mechanisms or other measures are employed to safeguard the transmitted data.

During the test scenario interpretation stage, the scenario is processed in the execution environment, where the test logic is converted into specific commands for controlling the physical interfaces. Depending on the system, this may be the TestStand engine, a Python interpreter (*MagicDAQ*), a Robot Framework environment (*PHiLIP*), or a built-in *SCRIPTIS* environment (*KEOLABS ContactLAB*). This approach enables the test logic to be abstracted from the hardware implementation.

The test object is interacted with by an executive module that controls physical interfaces (SPI, I2C, UART, ISO 7816, etc.), ensuring the transmission of signals and commands directly to the device under test. For example, *PHiLIP* uses a separate interface module for this purpose which interacts with and depends on the Raspberry Pi as a control module.

Test result recording involves capturing data obtained during the test, such as device responses, signals, timestamps, statuses, and measured values. This data can be stored locally on the execution module or transferred immediately to the control module for further analysis.

In the final stage, the recorded data is processed and the criteria for successfully completing the test case are calculated. Reports are then generated for operators, or the results are automatically transferred to external analysis and quality management systems.

The analysis of existing IoT device testing platforms and selected typical processes revealed several limitations. All of the considered solutions have one key limitation in common: a lack of a mechanism for the secure execution of test scenarios. Test scenarios are either executed locally, delivered manually, or via a continuous integration infrastructure, with no explicit use of cryptographic protection. Additionally, many tools are local and do not support remote execution modules. In today's environment, where testing is often performed in an untrusted environment that requires the autonomous operation of execution modules, it is essential to ensure that tests can be carried out without risking interference or conflict.

Therefore, it is necessary to develop a new testing architecture for eliminating the identified shortcomings and ensuring the necessary level of security and scalability when testing IoT devices.

4. PROPOSED TESTING SYSTEM ARCHITECTURE

The proposed architecture comprises a distributed network for testing Internet of Things devices. In this network, several executive modules interact with the control module to exchange encrypted test scenarios. This architecture (Fig. 8) comprises the following key components:

- Control module. This is responsible for coordinating distributed execution modules and storing test scenarios.

- Execution modules. These are autonomous nodes that are physically connected to the devices under test. The modules execute test scenarios received from the control module, record the results, and interact with the object under test via connected interfaces. Each execution station contains a built-in, isolated execution environment in which the received test scenarios are interpreted.
- Tested devices. These are real IoT devices connected to the execution modules and tested via physical interfaces. Each test station connects to the device under test via one of the interfaces (SPI, I2C, UART, etc.). In the proposed architecture, these connections are referred to as channels, each of which comprises a logically independent unit for controlling input and output signals. Each channel corresponds to a specific interface type or interaction protocol.

The testing process begins when the operator uses the control module to initiate execution of the script. The script is then transmitted to the execution module via a secure channel and executed in an isolated environment, generating commands to interact with the device under test's interfaces. The test results are then sent back to the control module in an encrypted format for analysis.

When designing the architecture, special attention should be paid to protecting the testing logic and mitigating risks arising from operation in a potentially untrusted environment. Studies [13, 14] have shown that the various stages of the device life cycle are accompanied by threats of unauthorized access to software and leakage of test scenarios. Furthermore, the microcontrollers themselves can act as both a source and a target of attacks. In light of these threats, the system architecture incorporates a set of protective measures to be applied at various stages of the test scenario life cycle.

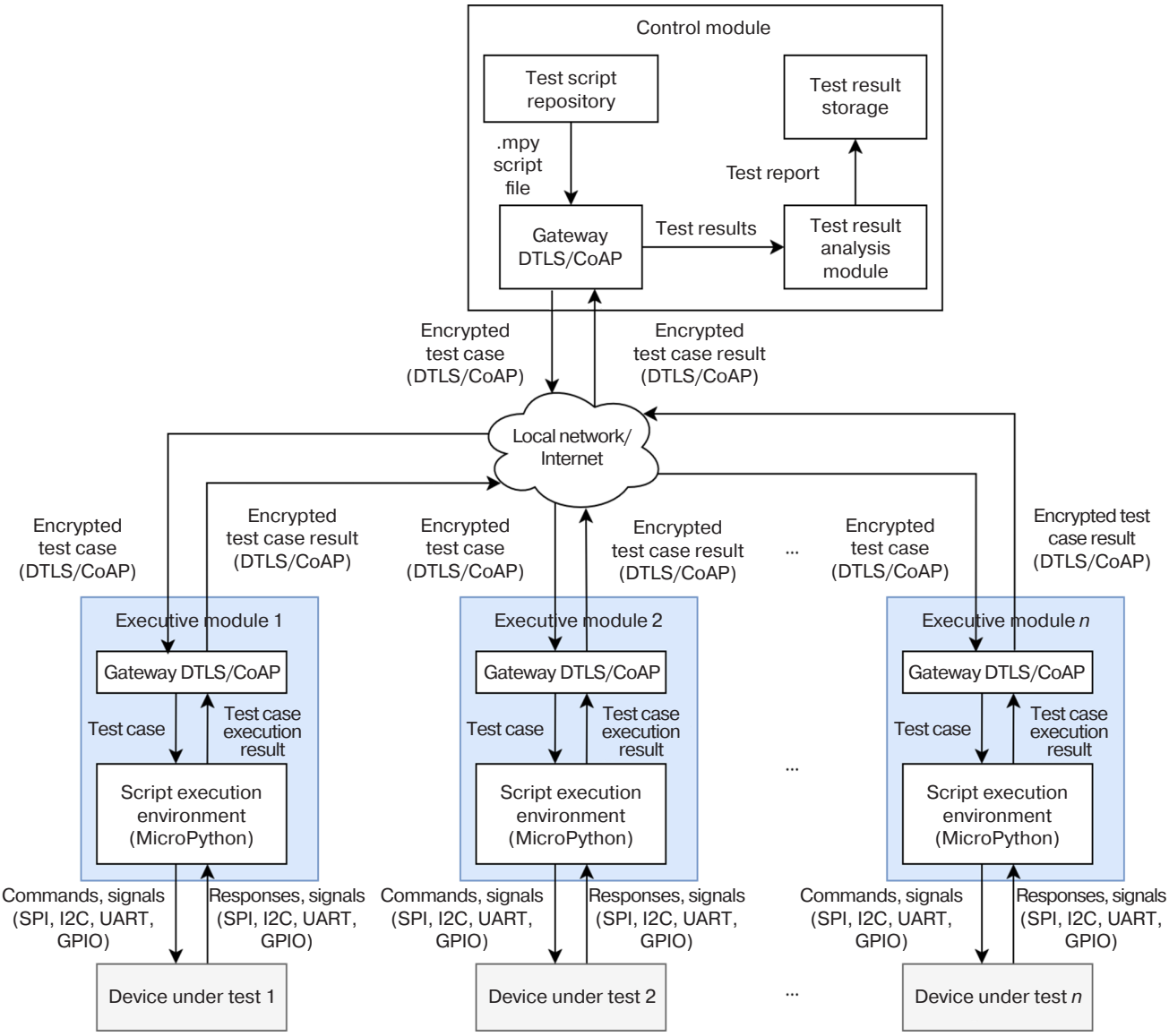

Fig. 8. Test system architecture

Table. Comparison of characteristics of existing and proposed solutions

Characteristic	*NI TestStand*	*MagicDAQ*	*PHiLIP*	*KEOLABS*	Proposed solution
Architecture type	Centralized	Centralized	Distributed	Centralized	Distributed
Execution environment	PC	PC	Executive module	PC	Executive module
Scalability	Local network (remote launch via CI)	Limited by the number of USB ports	High (adding Raspberry Pi nodes)	Limited by manual deployment	High (adding execution modules)
Execution isolation	OS	OS (Python)	Robot Framework	*SCRIPTIS*	MicroPython
Protection mechanisms	–	–	Not specified	–	DTLS
Remote control	Yes (CI)	No	Yes (CI)	No	Yes (control module)
Scope of application	General tests of electronic devices	General tests of electronic devices	Built-in OS APIs	Smart-cards, NFC	IoT devices, microcontroller interfaces

The DTLS protocol ensures the confidentiality and integrity of test scripts and results during transmission and reception. The protection of data when working in untrusted networks is particularly important for devices with limited computing resources [15].

Test scripts are executed in an isolated MicroPython[12] environment. This isolation ensures the secure execution of test scripts through compartmentalization and restricted access to resources [16].

The scalable system supports the connection of new execution modules, each of which operates autonomously. This enables parallel test execution and eliminates a single point of failure.

The table below shows the characteristics of existing and proposed architectural solutions. As can be seen, most existing systems are characterized by a centralized architecture, limited scalability, and a lack of protection mechanisms. In contrast, the proposed solution features a distributed network architecture of execution modules, an isolated execution environment, as well as encryption and authentication mechanisms. This approach ensures effective system scalability and significantly increases the protection of the test logic.

The proposed solution enhances and streamlines the processes involved in the test scenario lifecycle. Let us consider its key features and how it differs from other solutions:

- Transfer of test scenarios to the execution module. Unlike similar solutions, which transfer data via USB/Ethernet (e.g., *NI TestStand*, *MagicDAQ*, and *KEOLABS ContactLAB*) or a local network (e.g., *PHiLIP*), without using data transfer protection mechanisms, the proposed solution encrypts scenarios before transferring them from the control module to the execution module using the DTLS protocol. This ensures the confidentiality and integrity of test scenarios, even when operating in a potentially untrusted network.
- Interpretation of test scenarios. Scenarios are described as Python scripts and executed on a MicroPython virtual machine on an execution module that is isolated from the main OS. This isolation is not provided in previously discussed solutions (e.g., *NI TestStand* and *MagicDAQ*), where execution takes place in a shared user environment, creating the risk of interference with the testing logic.
- Test result recording. In many current solutions, test results either remain local (e.g., *KEOLABS ContactLAB*) or are processed manually (e.g., *MagicDAQ*), with no centralized analysis. The proposed architecture involves sending encrypted reports from modules to the control server.

The efficiency and performance of the proposed architecture are confirmed during the integration testing of the developed device. The tests are carried out on an experimental bench made up of a control module, several executive modules, and the device

[12] Micropython. https://micropython.org/. Accessed July 07, 2025.

under test. The control module is implemented on a PC, while the executive modules are implemented on SE-Discovery-GD32F427 boards (GigaDevice Semiconductor Inc., China), which have built-in software and the MicroPython environment. Interaction between the components takes place via secure network channels within a local area network. The architecture allows for remote interaction between the modules when they are connected to the internet. During testing, the following key system functions are verified: secure transmission of test scenarios and execution results and correct operation of the executive module's virtual machine. All test scenarios have been completed successfully, which confirms the operability and effectiveness of the proposed architecture.

The developed solution is used to test smart cards and hardware tokens, including verifying the correct execution of applets. During testing, power failures and unstable communication interfaces were simulated. One executive module controls the power supply to the device under test, while another ensures communication with it to reproduce real operating conditions and confirm the resistance of the system to abnormal situations.

CONCLUSIONS

The paper proposes a new architecture for a distributed testing system for IoT devices. This system was developed with consideration for the threats posed by operating in an untrusted environment. The analysis confirms that existing device testing systems are limited in terms of scalability and fail to ensure the secure transmission or execution of test scenarios. The proposed testing architecture solves these problems by implementing security measures. The use of isolated virtual execution environments and encrypted transmission channels protects test scenarios from unauthorized access or substitution.

The key advantages of the proposed architecture are therefore security (encryption and logic isolation) and scalability (the ability to add nodes without changing the architecture), making it a suitable solution for testing IoT devices in a potentially untrusted environment.

REFERENCES

1. Dyakov O.N., Belyakov D.S., Kalinin E.O. Using ePKI technology to securely update of embedded software of trusted hardware and software system. *Bezopasnost' informatsionnykh tekhnologii = IT Security* (*Russia*). 2025;32(2):152–177 (in Russ.). https://doi.org/10.26583/bit.2025.2.12
2. Venugopal M., Nanda M., Anand G., Chandana Voora H. An integrated Hardware/Software Verification and Validation methodology for Signal Processing Systems. *ITM Web Conf.* 2022;50:02001. https://doi.org/10.1051/itmconf/20225002001
3. Bures M., Cerny T., Ahmed B.S. Internet of Things: Current Challenges in the Quality Assurance and Testing Methods. *arXiv*. arXiv:1805.01241. 2018. https://doi.org/10.48550/arXiv.1805.01241
4. Mazhar T., Talpur D.B., Shloul T.A., Ghadi Y.Y., Haq I., Ullah I. Analysis of IoT Security Challenges and its Solutions Using Artificial Intelligence. *Brain Sciences*. 2023;13(4):683. https://doi.org/10.3390/brainsci13040683
5. Minani J.B., Sabir F., Moha N., Guéhéneuc Y.G. A Multimethod Study of Internet of Things Systems Testing in Industry. *IEEE Internet Things J*. 2024;11(1):1662–1684. https://doi.org/10.1109/JIOT.2023.3291233
6. Papulovskaya N.V., Izotov I.N., Blinichkin D.Y., Kataev A.Y. Core Platform Development for IoT-devices Automated Testing. *Int. J. Open Inf. Technol*. 2021;9(6):38–45 (in Russ.). https://elibrary.ru/ybxvtg
7. Castelo Branco K.D.S., Dantas V.L.L., Carvalho L.M. Interoperability Testing Guide for the Internet of Things. In: *Proceedings of the 30th Brazilian Symposium on Multimedia and the Web* (*WebMedia 2024*). (*Sociedade Brasileira de Computação*). 2024. P. 188–196. https://doi.org/10.5753/webmedia.2024.242058
8. Weiss K., Rottleuthner M., Schmidt T.C., Wählisch M. PHiLIP on the HiL: Automated Multi-Platform OS Testing with External Reference Devices. *ACM Trans. Embed. Comput. Syst*. (*TECS*). 2021;20(5s):1–26. https://doi.org/10.1145/3477040
9. Behnke I., Thamsen L., Kao O. Héctor: A Framework for Testing IoT Applications Across Heterogeneous Edge and Cloud Testbeds. In: *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing Companion.* ACM; 2019. P. 15–20. https://doi.org/10.1145/3368235.3368832
10. Ziegler S., Fdida S., Viho C., Watteyne T. F-Interop – Online Platform of Interoperability and Performance Tests for the Internet of Things. In: Mitton N., Chaouchi H., Noel T., Watteyne T., Gabillon A., Capolsini P. (Eds.). *Interoperability, Safety and Security in IoT. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*. Springer; 2017. V. 190. P. 49–55. Available from URL: https://link.springer.com/chapter/10.1007/978-3-319-52727-7_7. Accessed July 07, 2025.
11. Olianas D., Leotta M., Ricca F. MATTER: A tool for generating end-to-end IoT test scripts. *Software Qual. J*. 2021;30(2): 389–423. Available from URL: https://link.springer.com/article/10.1007/s11219-021-09565-y. Accessed July 07, 2025.
12. Schieferdecker I., Kretzschmann S., Rennoch A., Wagner M. IoT-Testware – An Eclipse Project. In: *2017 IEEE International Conference on Software Quality, Reliability and Security* (*QRS*). IEEE; 2017. https://doi.org/10.1109/QRS.2017.59
13. Belyakov D.S., Kalinin E.O., Konev A.A., Shelupanov A.A., Mitsel A.A. Life-cycle models and security threats to the microchip during its development and exploitation. *Doklady Tomskogo gosudarstvennogo universiteta sistem upravleniya i radioehlektroniki* (*Doklady TUSUR*) = *Proceedings of TUSUR University.* 2023;26(1):76–81 (in Russ.). https://doi.org/10.21293/1818-0442-2023-26-1-76-81

14. Konev A.A. Security threat model for protected microcontroller and the information it processes. *Doklady Tomskogo gosudarstvennogo universiteta sistem upravleniya i radioehlektroniki* (*Doklady TUSUR*) = *Proceedings of TUSUR University.* 2022;25(4):80–87 (in Russ.). https://doi.org/10.21293/1818-0442-2022-25-4-80-87
15. Restuccia G., Tschofenig H., Baccelli E. Low-Power IoT Communication Security: On the Performance of DTLS and TLS 1.3. In: *Proceedings 2020 9th IFIP International Conference on Performance Evaluation and Modeling in Wireless Networks* (*PEMWN*). https://doi.org/10.23919/PEMWN50727.2020.9293085
16. Lowther D., Jacob D., Trevor J., Singer J. Secure Scripting with CHERIoT MicroPython. In: *Proceedings of the 34th ACM SIGPLAN International Conference on Compiler Construction*. ACM; 2025. P. 180–191. https://doi.org/10.1145/3708493.3712694

СПИСОК ЛИТЕРАТУРЫ

1. Дьяков О.Н., Беляков Д.С., Калинин Е.О. Использование технологии ePKI для безопасного обновления встроенного программного обеспечения доверенных программно-аппаратных комплексов. *Безопасность информационных технологий.* 2025;32(2):152–177. https://doi.org/10.26583/bit.2025.2.12
2. Venugopal M., Nanda M., Anand G., Chandana Voora H. An integrated Hardware/Software Verification and Validation methodology for Signal Processing Systems. *ITM Web Conf.* 2022;50:02001. https://doi.org/10.1051/itmconf/20225002001
3. Bures M., Cerny T., Ahmed B.S. Internet of Things: Current Challenges in the Quality Assurance and Testing Methods. *arXiv*. arXiv:1805.01241. 2018. https://doi.org/10.48550/arXiv.1805.01241
4. Mazhar T., Talpur D.B., Shloul T.A., Ghadi Y.Y., Haq I., Ullah I. Analysis of IoT Security Challenges and its Solutions Using Artificial Intelligence. *Brain Sciences*. 2023;13(4):683. https://doi.org/10.3390/brainsci13040683
5. Minani J.B., Sabir F., Moha N., Guéhéneuc Y.G. A Multimethod Study of Internet of Things Systems Testing in Industry. *IEEE Internet Things J*. 2024;11(1):1662–1684. https://doi.org/10.1109/JIOT.2023.3291233
6. Папуловская Н.В., Изотов И.Н., Блиничкин Д.Ю., Катаев А.Ю. Разработка ядра платформы автоматизированного тестирования устройств интернета вещей. *Int. J. Open Inf. Technol.* 2021;9(6):38–45. https://elibrary.ru/ybxvtg
7. Castelo Branco K.D.S., Dantas V.L.L., Carvalho L.M. Interoperability Testing Guide for the Internet of Things. In: *Proceedings of the 30th Brazilian Symposium on Multimedia and the Web* (*WebMedia 2024*). (*Sociedade Brasileira de Computação*). 2024. P. 188–196. https://doi.org/10.5753/webmedia.2024.242058
8. Weiss K., Rottleuthner M., Schmidt T.C., Wählisch M. PHiLIP on the HiL: Automated Multi-Platform OS Testing with External Reference Devices. *ACM Trans. Embed. Comput. Syst*. (*TECS*). 2021;20(5s):1–26. https://doi.org/10.1145/3477040
9. Behnke I., Thamsen L., Kao O. Héctor: A Framework for Testing IoT Applications Across Heterogeneous Edge and Cloud Testbeds. In: *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing Companion.* ACM; 2019. P. 15–20. https://doi.org/10.1145/3368235.3368832
10. Ziegler S., Fdida S., Viho C., Watteyne T. F-Interop – Online Platform of Interoperability and Performance Tests for the Internet of Things. In: Mitton N., Chaouchi H., Noel T., Watteyne T., Gabillon A., Capolsini P. (Eds.). *Interoperability, Safety and Security in IoT. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*. Springer; 2017. V. 190. P. 49–55. URL: https://link.springer.com/chapter/10.1007/978-3-319-52727-7_7. Accessed July 07, 2025.
11. Olianas D., Leotta M., Ricca F. MATTER: A tool for generating end-to-end IoT test scripts. *Software Qual. J.* 2021;30(2): 389–423. URL: https://link.springer.com/article/10.1007/s11219-021-09565-y. Accessed July 07, 2025.
12. Schieferdecker I., Kretzschmann S., Rennoch A., Wagner M. IoT-Testware – An Eclipse Project. In: *2017 IEEE International Conference on Software Quality, Reliability and Security* (*QRS*). IEEE; 2017. https://doi.org/10.1109/QRS.2017.59
13. Беляков Д.С., Калинин Е.О., Конев А.А., Шелупанов А.А., Мицель А.А. Модели жизненного цикла и угрозы безопасности микросхемы во время ее разработки и эксплуатации. *Доклады Томского государственного университета систем управления и радиоэлектроники* (*Доклады ТУСУР*). 2023;26(1):76–81. https://doi.org/10.21293/1818-0442-2023-26-1-76-81
14. Конев А.А. Модель угроз безопасности защищенного микроконтроллера и обрабатываемой им информации. *Доклады Томского государственного университета систем управления и радиоэлектроники* (*Доклады ТУСУР*). 2022;25(4):80–87. https://doi.org/10.21293/1818-0442-2022-25-4-80-87
15. Restuccia G., Tschofenig H., Baccelli E. Low-Power IoT Communication Security: On the Performance of DTLS and TLS 1.3. In: *Proceedings 2020 9th IFIP International Conference on Performance Evaluation and Modeling in Wireless Networks* (*PEMWN*). https://doi.org/10.23919/PEMWN50727.2020.9293085
16. Lowther D., Jacob D., Trevor J., Singer J. Secure Scripting with CHERIoT MicroPython. In: *Proceedings of the 34th ACM SIGPLAN International Conference on Compiler Construction*. ACM; 2025. P. 180–191. https://doi.org/10.1145/3708493.3712694

About the Author

Danila S. Belyakov, Senior Lecturer, Department of Complex Information Security of Computer Systems, Faculty of Security, Tomsk State University of Control Systems and Radioelectronics (40, Lenina pr., Tomsk, 634050 Russia). E-mail: cauze4concern@yandex.ru. Scopus Author ID 57359476400, ResearcherID AAQ-4613-2021, RSCI SPIN-code 3368-8751, https://orcid.org/0000-0002-6111-455X

Об авторе

Беляков Данила Сергеевич, старший преподаватель, кафедра комплексной информационной безопасности электронно-вычислительных систем, факультет безопасности, ФГАОУ ВО «Томский государственный университет систем управления и радиоэлектроники» (634050, Россия, Томск, пр-т Ленина, д. 40). E-mail: cauze4concern@yandex.ru. Scopus Author ID 57359476400, ResearcherID AAQ-4613-2021, SPIN-код РИНЦ 3368-8751, https://orcid.org/0000-0002-6111-455X

Translated from Russian into English by K. Nazarov
Edited for English language and spelling by Thomas A. Beavitt