# Device Identity Bootstrapping in Constrained Environments: A BLE-based BRSKI Extension

1st Julian Krieger
*Munich University of Applied Sciences*
Munich, Germany
0009-0005-3000-0727

2nd Tobias Hilbig
*Munich University of Applied Sciences*
Munich, Germany
0000-0002-2904-4758

3rd Thomas Schreck
*Munich University of Applied Sciences*
Munich, Germany
0000-0002-8960-6986

***Abstract*—In contrast to user authentication in the digital domain, device authentication entails distinct requirements and challenges. In the enterprise context, Internet-of-Things (IoT) devices must support strong identities to ensure secure and trusted operations. A critical first step in developing such systems is the secure bootstrapping of these devices. However, achieving trusted bootstrapping is challenging when Internet access is prohibited. In such environments, an identity must be provided to the device beforehand. The solution should also work with minimal human intervention in our scenario. To achieve this, we developed cBRSKI-PRM, a hybrid combination of BRSKI-PRM and cBRSKI. BRSKI-based communication between components is facilitated via Bluetooth Low Energy. We implemented the proposed solution on an ESP32 platform, ensuring reliable performance in both hardware and software while optimizing for low power consumption. Additionally, we identified and addressed several issues in the specifications of the employed protocols. We published all source code used in this project under the permissive MIT license. By combining and extending existing open protocols, we developed a highly assured bootstrapping process for IoT devices under minimal trust assumptions in constrained environments.**

***Index Terms*—IoT, Bluetooth, bootstrapping, BRSKI, ESP32**

## I. Introduction

In enterprise environments, identity bootstrapping for constrained devices faces significant challenges due to the resource demands of existing solutions. Many current approaches rely on protocols and cryptographic operations that exceed the capabilities of low-power devices. They also often require initial network connectivity, which is not always desired in enterprise settings. The rapid rise of Internet-of-Things (IoT) in these environments highlights the importance of secure identity bootstrapping to protect systems against unauthorized access. However, constrained devices with limited processing power, memory, and battery capacity cannot effectively utilize existing bootstrapping methods.

This paper proposes a novel solution that addresses these limitations by leveraging Bluetooth Low Energy (BLE) as the sole communication protocol. By combining two bootstrapping protocols, Bootstrapping Remote Secure Key Infrastructure with Pledge in Responder Mode (BRSKI-PRM) and Constrained Bootstrapping Remote Secure Key Infrastructure (cBRSKI), we develop a lightweight, scalable, and secure identity bootstrapping system tailored to the specific needs of enterprise IoT devices. The design, implementation, and evaluation of this approach demonstrate its effectiveness in advancing secure identity management for constrained devices in enterprise settings.

The contributions of this work include: (1) a proposal for a hybrid, BRSKI-PRM and cBRSKI based bootstrapping protocol over BLE, (2) the first official open-source implementation of cBRSKI-PRM, including our proposed modifications, (3) virtual and physical, ESP-32C3 and Android based, unit and integration tests to evaluate the functional correctness of our implementation.

This work is organized as follows: We begin by providing background information in Section II and discussing requirements and related work in Section III. In Section IV, we present our modifications to the protocol in detail. Information about our proof-of-concept together with the evaluation is given in Section V. We discuss our findings and provide directions for future work in Section VI. Section VII concludes this work.

## II. Background

IoT devices are characterized by limited power and processing capabilities and require innovative approaches for secure identity bootstrapping. The process involves the verification of a device's identity, followed by the issuance of key material and the distribution of configuration data. Effective bootstrapping schemes are essential for integrating these devices into existing systems while ensuring robustness and scalability. The security of this process plays an important role in enterprise contexts. The bootstrapping process can also include the tracking of devices by a registration service hosted in the customer domain.

Ideally, vendors configure their devices with identification information during the manufacturing process, for example, in the form of an attestation key held in a secure enclave. Instead of providing a single shared key, it is imperative to equip each device with a unique identifier. Upon delivery, the customer may read this information from the device, and match it to a

source of known devices provided by the device vendor. While such information can be delivered to the customer alongside the device shipment, a vendor may also host an externally available service for automated verification processes. With such a system, vendors can vouch for their devices during the bootstrapping process. This allows customers to securely integrate manufacturer-affirmed devices in their environments.

Some bootstrapping schemes choose X.509 end-entity certificates as the attestation mechanisms. Vendors can readily provide the CA certificate that issued the device certificate, enabling verification of the trust chain at the customer side. Depending on the architecture, customers can supply devices with a certificate and key material valid for their domain. This information may be used to set up secure connections after bootstrapping has succeeded. In our use-case, IoT devices are intentionally prevented from accessing the local network before they can successfully prove their identity. Therefore, we want to make use of a proxy component, an additional entity that acts as a service proxy between a device and the customer's registration authority. This service proxy facilitates communication between devices and the customer domain's registration authority.

BLE [1] is a wireless communication protocol designed for efficient, short-range data transmission. It is optimized for quick and lightweight interactions without requiring pairing. This makes it ideal for applications like real-time control systems and smart infrastructure. Its low power consumption and flexible communication make it suitable for a wide range of devices, offering a cost-effective solution for scalable, high-performance networks.

BLE uses the Generic Attribute Profile (GATT) [1, G] protocol to organize information into a hierarchy of fundamental data units called *attributes*. Attributes include *services* and *characteristics* and are distinguished by UUIDs. GATT services are collections of related characteristics that define specific functionalities and enable organized data exchange between devices [1, G-2.6.2]. GATT characteristics are individual data points within a GATT service that represent specific values or attributes [1, G-2.6.4]. Operations encompass the specific actions performed on a service's characteristics [1, G-3.3.1.1]. This includes reading values, writing new values, and subscribing to notifications or indications of changes. While similar protocols exist, GATT's structured architecture comes with the advantage of allowing a clear mapping to REST operations.

## III. Requirements and Related Work

We begin by describing our requirements for the desired solution to ensure a secure bootstrapping process for devices in constrained environments. We then review existing protocols to assess their capabilities.

### A. Requirements

A bootstrapping protocol for IoT devices in constrained enterprise environments must meet several critical requirements: Given that our primary application are low-powered, constrained devices, the scheme must not exceed their limitations. To accommodate varying numbers of devices, we need to ensure scalability without compromising performance. The scheme should require minimal outside intervention, simplifying the user experience for service technicians and enhancing efficiency. Most of the complexity should be hidden, with the only necessary interaction being the verification of the correct installation location. Devices should not require access to the local network before completing bootstrapping, ensuring customers do not need to trust unauthenticated devices on first use. To reduce network load, passive devices should remain dormant until they receive an impulse to initiate the bootstrapping process.

Utilizing open-source hardware and software is preferable to foster innovation and community engagement. Adhering to established enterprise standards can facilitate integration with existing infrastructure, allowing for a smoother transition for enterprises. Strong security measures must be embedded to protect sensitive data and prevent unauthorized access. Therefore, we require the usage of attestation keys unique to each device, rather than identical pre-shared keys. Finally, vendors should cooperate in the identity-proofing process, for example by providing openly accessible interfaces for device identity verification.

In accordance with these requirements, we reviewed the literature for standardized bootstrapping schemes suitable for IoT devices: We first considered Automatic Certificate Management Environment (ACME) [2] to handle the provisioning of device certificates. Through the ACME protocol, clients can request and retrieve certificates from a central service in an automated fashion. Initially, clients must establish a DHCP connection and employ a discovery scheme like DNS Service Discovery (DNS-SD) [3]. Both steps require an initial network connection that does not exist in our scenario. For these reasons, we did not consider ACME any further.

Another protocol, called Enrollment over Secure Transport (EST) [4], seemed to better fit our requirements. However, EST is designed for devices which can establish a secure transport channel via HTTP over TLS. It does not specify how this connection is intended to be initially established or how key material is to be distributed to enrollment candidates. This makes the protocol unsuitable for scenarios in which vendor-installed pre-shared key material or the manual installation of secret keys is not desired or possible.

Finally, Bootstrapping Remote Secure Key Infrastructure (BRSKI) [5] offers automated trust establishment for devices and scalability for large-scale device deployments. BRSKI builds upon and extends the functionalities provided by EST and provides further functionality to establish an initial connection to the local registration authority. In addition to the target device, called *Pledge*, BRSKI defines a *Manufacturer Authorized Signing Authority (MASA)* component operated by the device vendor. It further specifies a customer-operated *Domain Registrar*, which handles communication between the onboarding candidate and its vendor. In enterprise environments, minimizing the local network connectivity of foreign,

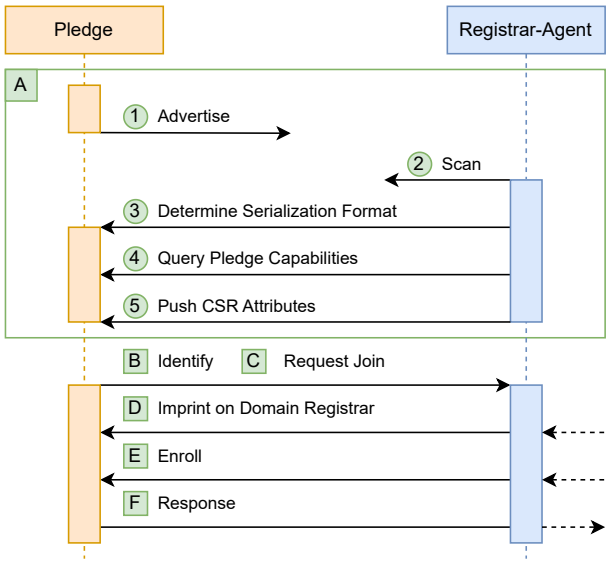


Figure 1. Protocol sequence diagram. Steps (1) – (5) are our modifications, while steps [A] – [F] show the usual BRSKI-PRM procedure. The interactions with the MASA and Domain Registrar in steps [D] – [F] are not depicted [7].

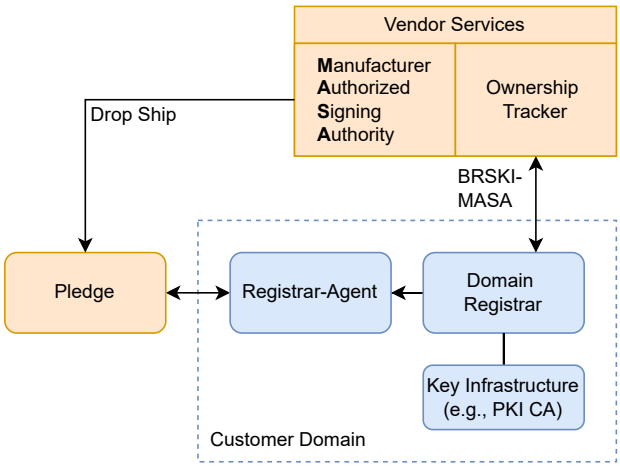


Figure 2. Architecture of BRSKI-PRM with abstract components [7].

untrusted devices is often desirable. Therefore, BRSKI only requires that newfound Pledges establish a direct connection to a so-called *Join Proxy*, which acts as an intermediary between the Pledge and the Domain Registrar [5, 2.5.1-2].

Communication in BRSKI is based on public key infrastructure and is implemented as the exchange of signed payloads carrying device- or service-owned certificates. The device-owned certificate is installed by the vendor during manufacturing. Initially, mutual authentication between new devices and a customer's network is achieved through a vendor-issued, signed artifact called *voucher* [6]. This voucher is used to securely deliver a root CA certificate onto the Pledge for verification of the registrar's identity [5, 2.5.5]. After the trust anchor is installed onto the device, BRSKI continues with the enrollment as defined in EST. By providing an automated preliminary phase for the authorization of customer domain CA certificates, BRSKI eliminates the need for the manual interaction defined by EST [5, 1.1]. Therefore, BRSKI meets our requirements, as it offers a mutual verification process that provides the security guarantees we require.

## IV. Proposed Protocol Extension

This section outlines the practical aspects of the proposed scheme. We first discuss our modifications of BRSKI-PRM and cBRSKI that enable transport over BLE. In Figure 1, all steps of the final scheme are depicted, with detailed explanations given throughout this section. We then highlight the shortcomings of the current library ecosystem and present our contributions to addressing them.

### A. BRSKI with Pledge in Responder Mode

BRSKI-PRM [7] enhances BRSKI with a pledge-passive mode, wherein devices wait for an initial trigger to start the onboarding process. It introduces a *Registrar-Agent* component, which may replace, or make use of, the Join-Proxy defined in BRSKI. Its architecture with all individual components is depicted in Figure 2. By using the Registrar-Agent in the form of a manually operated, out-of-band commissioning tool, Pledges may be enrolled into a network without the need for a co-located intermediary device [7, 5.2]. Once an impulse is sent, both the Pledge and the Registrar-Agent generate, sign, and swap initial artifacts. To prevent replay attacks, these include the Pledge's serial number and a nonce or a timestamp. While the standard describes a HTTP(S) connection between Registrar-Agent and Pledge, it is not limited to IP-based protocols [7]. By utilizing a BLE or Near Field Communication (NFC) connection, one could entirely eliminate the need for untrusted bootstrapping candidates to enter a network before and during the bootstrapping process. However, the specification does not provide details on non-IP-based implementations. To fulfill our requirement of enrolling devices without initial network connectivity, we enable both the Pledge and the Registrar-Agent to exchange data via the BLE protocol. Furthermore, we adapt BRSKI-PRM with techniques better suited for embedded hardware and its unique constraints.

### B. Payload Compression

Constrained devices with limited computation resources benefit from minimal transfer sizes and serialization formats with a focus on a small memory footprint. They often do not exchange data in the format of HTTP messages, but instead rely on transport protocols that use bespoke binary message formats. The size of traditional BRSKI data packets is often inflated by multiple, possibly large certificate chains. Accodring to the BRSKI specification [5, 5], these and other binary data must be encoded in Base64 [8]. Base64 is a binary-to-text encoding scheme, representing every three bytes of binary data as four bytes of ASCII text, resulting in an overhead of approximately 33%.

Choosing a serialization format better suited for constrained IoT devices could minimize the payload size. Therefore, we

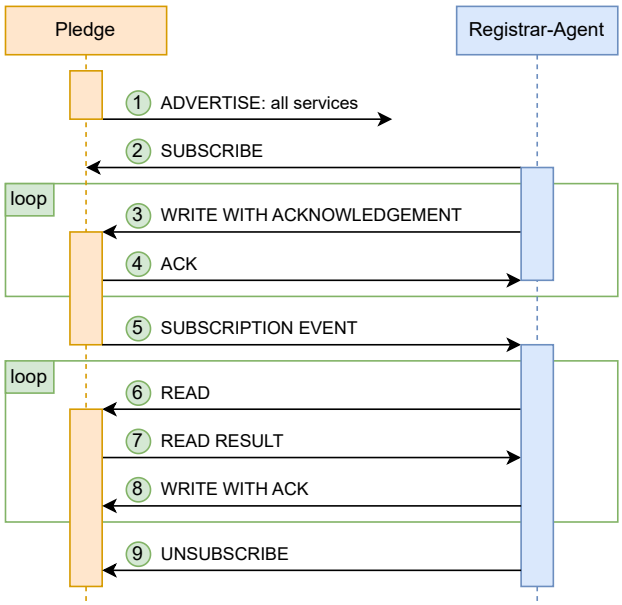


Figure 3. Sequence diagram for GATT service access. BLE-based interactions between the Pledge and the Registrar-Agent component operates as depicted.

chose to look into cBRSKI [9], which utilizes Concise Binary Object Representation (CBOR) [10] as a message encoding format. In CBOR, binary data does not need to be converted to and from Base64 encoding, as they are directly transmitted in their original representation. BRSKI and BRSKI-PRM make use of JSON Web Signature (JWS) tokens as a signature framework for their JSON payloads. Alternatively, cBRSKI makes use of CBOR Object Signing and Encryption (COSE) [11], a protocol for the exchange of signed CBOR payloads. While cBRSKI is a BRSKI extension with a direct focus on hardware constrained devices, it comes with a decisive disadvantage. Similar to BRSKI, Pledges in cBRSKI need to be directly connected to the local network in order to communicate with the registrar. cBRSKI also does not specify bootstrapping for Pledges for which the onboarding proxy handles transport via a non-IP-based transport protocol. Our proposed solution is thus a fusion of a BLE-modified BRSKI-PRM protocol and features from cBRSKI to limit data transfer sizes. This hybrid protocol works as a pledge-passive one-touch bootstrapping scheme for constrained devices.

### C. Multi Signature Tokens

The use of JWS and COSE as signature and encoding protocols poses some problems. BRSKI and its extensions make use of multi-signed tokens. When transmitting data between the device and the manufacturer, the customer's Domain Registrar may add its signature in-flight. At the time of writing, we were unable to find third-party libraries that correctly implemented the handling of such multi-signature tokens. To circumvent this issue, we considered the signed payloads to be nested instead. After consultation with the committee in charge of developing BRSKI-PRM, this idea was rejected, as the challenges we faced in our ecosystem do not warrant a deviation from the standard. Thus, in order to be compliant with BRSKI-PRM's specification, we developed our own implementation for multi-signature COSE tokens by extending *coset* [12], a COSE library by Google.

### D. Bluetooth Extension for BRSKI-PRM

BRSKI and its extensions rely on IP-based transport protocols, making use of the REST paradigm for data exchange. In our implementation, the Registrar-Agent is a BLE client device, with the Pledge acting as a GATT central server, see steps (1) and (2) in Figure 1. Similar to an HTTP route, each communication endpoint in the BRSKI-PRM specification is mapped to a single BLE service identified by a UUID. Figure 3 depicts how both entities communicate. The process itself is the same for all operations. In the following, we explain the steps in detail: After the Pledge is powered on, (1) it advertises all services by their UUIDs. Each mapped service consists of a pair of characteristics for writing and reading data. Instead of sending a request, the Registrar-Agent subscribes to a writing-enabled characteristic (2) and writes chunked data (3), receiving confirmation for every write (4). A subscription event (5) is sent after all data was received, deserialized, and processed by the Pledge. After finishing its computation, it serializes the response data and writes them onto the characteristic designed for reading data. The Pledge reads the chunked response (6), receives the result (7) and confirms the successful retrieval (8). Finally, it unsubscribes from the characteristic (9).

### E. Device Capability Advertisement

Depending on the context, payloads in BRSKI-PRM are of different types. In order to verify that a request is answered by data of the correct format, clients and servers must be able to communicate the intended content type. This is solved in BRSKI and its extensions via request-integrated HTTP content type negotiation strategies [13]. However, when using BLE as the transport protocol, no native functionality is available for this method. Therefore, we have extended our protocol with a content negotiation strategy, see step (3) in Figure 1. To save on transfer size, it is executed before the actual bootstrapping process takes place. To communicate intent, the Pledge advertises its capabilities as a set of BLE characteristics. First, the Pledge communicates the serialization format it wishes to use for communication with the Registrar-Agent. The format is transmitted via plain UTF-8-encoded text and is a choice between JSON and CBOR for our use-case. With the interchange format set, the Registrar-Agent can now request additional capabilities from the device, see step (4) in Figure 1. This information bundle specifies supported types of both unsigned and signed tokens. It also includes the requested serialization format of the BRSKI voucher object and a list of supported encryption and signing algorithms. Using this protocol, the Registrar-Agent can seamlessly support devices with differing capabilities. Devices in turn can exactly specify their supported feature-set that best fits their hardware poten-

tial. Finally, the Registrar-Agent pushes the certificate signing request attributes to the Pledge, see step (5) in Figure 1.

## V. Proof of Concept and Evaluation

We describe how we implemented and evaluated the proposed scheme in the following sections.

### A. SoC and Software

We chose Espressif's ESP32-C3 as a development board for our project. Espressif provides a fully-featured software development kit and comprehensive documentation for their boards. One of the use-cases we envision involves access control features that require maximum dependability. Therefore we chose to make use of the Rust programming language for the implementation of the embedded firmware and other software components. Leveraging the programming language's inherent memory safety, performance, and focus on correctness, we aim to increase the solution's dependability by preventing critical memory handling-related vulnerabilities at the language level. Using official Rust abstractions for the ESP32, we can utilize Espressif's software while also harnessing the power of a safe-by-default ecosystem. However, using Rust is not entirely without disadvantages. Compared to C, Rust's third-party software ecosystem is still relatively immature, leading to challenges of varying severity, which we discuss in the following sections. We contributed all developed solutions to the open source community.

We intend to use the device's Bluetooth antenna as a beacon for indoor navigation in future use-cases. Therefore, we need to use both the ESP32's WiFi and Bluetooth chipset in tandem for IoT-devices that connect wirelessly. In the majority of ESP boards, Bluetooth and WiFi functionalities use a single 2.4 GHz module. Writing application code correctly for using both in tandem is complex and possibly error-prone. Not handling the simultaneous use correctly may lead to starvation of either functionality or to system instability [14]. RF coexistence is further complicated when relying on Bluetooth Mesh functionality, which we intend to use for indoor navigation. To solve this issue, we use Rust's asynchronous capabilities for embedded hardware [15]. This allows us to implement our firmware as a non-blocking, event-based state machine. Using this technique, we receive the benefit of near-instant switching between both components. We can further guarantee that each component is correctly setup or torn down before and after use. The microchip can efficiently shift between computing intensive workloads by switching contexts without blocking the computation of a parallel thread. Furthermore, we can query the ESP32's Real Time Operating System for a status report of available computing resources. In case of resource exhaustion, we can effortlessly block computation on the current thread and periodically check for resources to become available.

Another drawback of Rust is a lack of mature TLS libraries for embedded architectures. While Espressif provided OpenSSL bindings in past versions of their development kit [16], the ecosystem now relies on WolfSSL or MbedTLS for SSL connectivity [17]. While Rust bindings were in development for both solutions at the time of writing, we have found them to be not yet ready for production use. To integrate TLS and signing functionality more tightly into our needs for safety in critical systems, we relied instead on the *Ring* [18] library. Ring is a Rust-based implementation of Google's BoringSSL [19], which itself is a fork of OpenSSL. It provides us with the security primitives needed to verify X.509 certificates and to sign COSE payloads in accordance to BRSKI-PRM's specification. Finally, we added upstream support to compile Ring on ESP32 architectures [20].

### B. Wireless Communication

The ESP32-C3 supports two means of communication with clients. IP-based protocols can be employed using its wireless antenna and network stack. The SoC is also equipped with a Bluetooth chipset for serial communication via Bluetooth Classic or BLE. By choosing BLE, we gain the advantage of minimizing pollution in the 2.4 GHz spectrum with numerous pairable devices, as we only require short-range communication. With future battery-powered devices in mind, we can also benefit from drastically lower power consumption. We chose the NimBLE Bluetooth implementation due to its minimal memory footprint and low power usage.

Even with minimal CBOR-based payloads, BRSKI based payloads are slightly larger than intended for BLE communication. This is mostly due to the inclusion of trust chains in the form of DER-encoded certificates [21], [22]. The required size for our own extensions to control messages, e.g., content type negotiation, is not pivotal here. To handle transactions larger than the minimum MTU of 23 bytes [1, A-4.22], the BLE specification includes long read and long write operations. However, the size of transmitted data within a single operation is still limited by the largest possible attribute size of 512 bytes [1, F-3.2.9]. Although we exceed this limit, we do not consider the resulting challenge sufficiently critical to justify using another protocol.

We therefore implemented support for long reads and writes directly at the application level. Our solution is heavily inspired by Silicon Labs [23] and employs a fragmentation algorithm for the exchange of payload fragments. First, we instruct the BLE peripheral to negotiate the maximum possible MTU size. We split the data into fragments that fit with the maximum possible MTU length, while also including padding for header information overhead. By using the *Write with Acknowledgment* operation, we are able to verify a successful transfer of each chunk.

### C. Evaluation

To evaluate the proposed solution, we created two implementations. First, we set up all required components in a virtual environment, including both Registrar-Agent and Pledge simulated as web servers. This allowed us to trace each individual step of the protocol to confirm its correctness. Unit tests allowed us to ensure the functional correctness of the BRSKI-PRM related parts of our implementation. In

addition, we conducted integration tests to verify that the Domain Registrar and MASA correctly handle malformed payloads and errors. Then, we converted most of the simulated Pledge logic to code running on the ESP32-C3 platform and began testing the physical components for functionality and correctness.

Instead of building a complex public key infrastructure with the sole purpose of testing our prototype, we generated test certificates for all entities with the *openSSL* utility. Their structure matched the examples in BRSKI [5, 7.1]. In addition, we prepared a wireless network with a pre-generated, matching X.509 certificate for testing. Connecting a serial flash tool [24] to the ESP32-C3 enables debugging capabilities and allows streaming device logs to the host computer. After powering up the SoC, we scanned for BLE devices using the Android-based Registrar-Agent. We were able to confirm the detection of a device advertising the serial number included in the flashed firmware. Instructing the Registrar-Agent to establish a connection allowed us to verify that the BLE device successfully promoted all required services.

To gain insight into the Registrar-Agent's function, we instructed it to send the logging output to a central logging collection service. Additionally, we have deployed both the MASA and the Domain Registrar services to be reachable in the local network to which the Registrar-Agent is also connected. Upon starting the bootstrapping process, we matched the embedded device logs to similar output we gathered when testing the simulated device. After swapping the required artifacts, the Pledge successfully generates both a voucher request and an enrollment request containing a Certificate Signing Request (CSR) and transmits both to the Registrar-Agent. This voucher request is forwarded to the Domain Registrar, which then wraps the voucher request with identifying information and forwards it to the MASA.

The MASA verifies the payload's correctness and then issues and responds with a voucher artifact. Next, the Registrar-Agent forwards the CSR to the Domain Registrar, which in turn issues and responds with the pre-generated network certificate. The voucher, the CA certificate bundle as well as the issued credentials are sent to the Pledge by the Registrar-Agent. Next, the Pledge returns a status report, confirming the successful verification of the domain trust anchor and voucher artifact. By observing the logging output, we confirmed that the device certificate and the domain trust anchor were successfully installed on the device in its native trust store. Furthermore, after completing the bootstrapping process, we observed that the device successfully connects to the testing network using the supplied device certificate.

## VI. Discussion and Future Work

In this paper, we have proposed a hybrid bootstrapping scheme tailored to constrained devices in enterprise environments. We first identified requirements and challenges for bootstrapping these devices in our target environment. Our literature review resulted in a number of candidate protocols, none of them viable in our use-case. The modularity of BRSKI and its extensions allowed us to compose a suitable solution, making it a good fit for bootstrapping approaches tailored to unique needs that arise in enterprise environments. In this work, we chose BLE as a replacement for the default IP-based networking stack. Additionally, we solved the challenges posed by our choice of transport protocol, demonstrating the use of BRSKI on resource-constrained devices. Our work demonstrates that BRSKI is a promising candidate for securing the onboarding process in enterprise environments. By implementing our hybrid approach on real hardware, we demonstrate that the proposed scheme is compatible with low-cost integrated chips. The BLE-based message exchange protocol was successfully tested with an Android phone acting as the Registrar-Agent and a ESP32-C3 device as the Pledge. The Rust-based implementation allowed us to successfully realize the protocol on hardware-constrained IoT devices, ensuring correctness, dependability, and security of the solution.

Future research can focus on further optimizing BLE data transfer and exploring new extensions to enhance the scalability and security of the proposed scheme in real-world deployments. Additionally, this work does neither include a re-enrollment nor certificate revocation scheme, which could be explored in forthcoming work. Despite promising results such as a quantifiable reduction of payload sizes, effective data transfer protocols for BLE with payloads exceeding the maximum GATT characteristic size require further investigation. Exploring existing, but currently unsupported BLE protocols within the ESP32 ecosystem for large data transfer could offer further improvements to our solution.

## VII. Conclusion

We presented a hybrid bootstrapping scheme designed specifically for IoT devices in constrained enterprise environments. Through the combination of BRSKI-PRM and cBRSKI, and by leveraging BLE, we developed a secure and efficient solution that addresses the challenges of intermittent connectivity, scalability, and resource limitations in enterprise IoT systems. Furthermore, we were able to provide insights and improvements to the BRSKI-PRM committee during this project. We also contributed to the broader ecosystem by providing the first official open-source example implementation for BRSKI-PRM [25]. Our work on secure firmware for the ESP32 includes improvements to the official Espressif Rust SDK's and their Rust-based BLE stack. Finally, we supported maintainers with multiple bug fixes related to the flash algorithm verification and debugging capabilities. This work is a step towards a future in which strong identity bootstrapping processes become the norm in enterprise IoT environments.

## Competing interests

All authors declare that they have no conflicts of interest.


## Acknowledgment

We would like to thank Steffen Fries and Thomas Werner at Siemens for their support and valuable insight.

## REFERENCES


[1] Bluetooth SIG, Inc, "Bluetooth Core Specification Version 6.0 Vol. 3," Bluetooth Special Interest Group, Tech. Rep. [Online]. Available: https://www.bluetooth.com/specifications/bluetooth-core-specification/

[2] R. Barnes, J. Hoffman-Andrews, D. McCarney, and J. Kasten, "Automatic certificate management environment (ACME)." [Online]. Available: https://www.rfc-editor.org/info/rfc8555

[3] S. Cheshire and M. Krochmal, "DNS-Based Service Discovery," RFC 6763. [Online]. Available: https://www.rfc-editor.org/info/rfc6763

[4] M. Pritikin, P. E. Yee, and D. Harkins, "Enrollment over secure transport." [Online]. Available: https://www.rfc-editor.org/info/rfc7030

[5] M. Pritikin, M. Richardson, T. Eckert, M. H. Behringer, and K. Watsen, "Bootstrapping remote secure key infrastructure (BRSKI)." [Online]. Available: https://www.rfc-editor.org/info/rfc8995

[6] K. Watsen, M. Richardson, M. Pritikin, and T. Eckert, "A Voucher Artifact for Bootstrapping Protocols," RFC 8366. [Online]. Available: https://www.rfc-editor.org/info/rfc8366

[7] S. Fries, T. Werner, E. Lear, and M. Richardson, "BRSKI with pledge in responder mode (BRSKI-PRM)." [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-anima-brski-prm/16/

[8] S. Josefsson, "The Base16, Base32, and Base64 Data Encodings," RFC 4648. [Online]. Available: https://www.rfc-editor.org/info/rfc4648

[9] M. Richardson, P. V. der Stok, P. Kampanakis, and E. Dijk, "Constrained bootstrapping remote secure key infrastructure (cBRSKI)." [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-anima-constrained-voucher/26/

[10] C. Bormann and P. E. Hoffman, "Concise binary object representation (CBOR)." [Online]. Available: https://www.rfc-editor.org/info/rfc8949

[11] J. Schaad, "CBOR Object Signing and Encryption (COSE)," RFC 8152, Jul. 2017. [Online]. Available: https://www.rfc-editor.org/info/rfc8152

[12] D. Drysdale and P. Crowley, "google/coset: A set of rust types for supporting COSE." [Online]. Available: https://github.com/google/coset

[13] R. T. Fielding, M. Nottingham, and J. Reschke, "HTTP Semantics," RFC 9110. [Online]. Available: https://www.rfc-editor.org/info/rfc9110

[14] Espressif Systems, "RF coexistence - ESP32-c3 – ESP-IDF programming guide v5.3.1 documentation." [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/stable/esp32c3/api-guides/coexist.html

[15] C. Lerche, T. d. Zeeuw, and tokio-rs. mio. [Online]. Available: https://lib.rs/crates/mio

[16] Espressif Systems, "ESP-IDF release v5.0 is a major update | espressif systems." [Online]. Available: https://www.espressif.com/en/news/ESP-IDFv5

[17] ——, "ESP-TLS - ESP32 – ESP-IDF programming guide v5.3.1 documentation." [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/v5.3.1/esp32/api-reference/protocols/esp_tls.html

[18] B. Smith. briansmith/ring: Safe, fast, small crypto using rust. [Online]. Available: https://github.com/briansmith/ring

[19] Google LLC. boringssl - git at google. [Online]. Available: https://boringssl.googlesource.com/boringssl

[20] J. Krieger, B. Smith, and L. Pirchio. Adding XTENSA architecture as a ring target · issue #2088 · briansmith/ring. [Online]. Available: https://github.com/briansmith/ring/issues/2088

[21] S. Boeyen, S. Santesson, T. Polk, R. Housley, S. Farrell, and D. Cooper, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," RFC 5280. [Online]. Available: https://www.rfc-editor.org/info/rfc5280

[22] C. Gardiner and C. Wallace, "ASN.1 Translation," RFC 6025. [Online]. Available: https://www.rfc-editor.org/info/rfc6025

[23] S. Labs, "Working with long characteristic values." [Online]. Available: https://github.com/SiliconLabs/bluetooth_stack_features/blob/master/gatt_protocol/working_with_long_characteristic_values/readme.md

[24] probe-rs Contributors, "probe-rs/probe-rs: A debugging toolset and library for debugging embedded ARM and RISC-V targets on a separate host." [Online]. Available: https://github.com/probe-rs/probe-rs

[25] J. Krieger, "hm-seclab/open-brski: cBRSKI-PRM Example Implementation." [Online]. Available: https://github.com/hm-seclab/open-brski