applied sciences

MDPI

Article

Tradeoffs in Key Rotation Strategies for Industrial Internet of Things Devices and Firmware

Sunil Joshi [1,*], Kenneth Crowther [2] and Jarvis Robinson [3]

[1] Enterprise Product Architecture, Cumming, GA 30040, USA
[2] Product Security, Richmond, VA 23233, USA; kenneth.crowther@xylem.com
[3] Cyber Security, Sandy Springs, GA 30328, USA; jarvis.robinson@xylem.com
* Correspondence: sunil.joshi@xylem.com

Abstract: This paper provides an overview of several secure boot architectures with a focus on key rotation. It expands on a practitioner note that the authors submitted to the 2023 IEEE Secure Development Conference. Key rotation is important due to the frequency of lost signing keys and the difficulty of managing secret keys for the long lifetimes of Industrial Internet of Things (IIOT) devices. Key rotation is not simple for IIOT due to limited resources during a secure boot process and the constraints of the firmware utilities that come from the chip vendors. This paper reviews and compares five common architectures for a secure boot that are seen across the IIOT community. For each architecture, it provides some key strengths and weaknesses associated with that architecture. The paper then provides a detailed comparison and analysis of the architectures to convince the IIOT community to move towards a strong use of certificates (instead of the traditional use of raw public keys). The intent of this paper is to provide a practitioner's perspective on these challenges and the tradeoffs in hopes of inviting comments from chip vendors and the broader community.

Keywords: firmware; signing; integrity; key; certificate; rotation

check for updates

Citation: Joshi, S.; Crowther, K.; Robinson, J. Tradeoffs in Key Rotation Strategies for Industrial Internet of Things Devices and Firmware. *Appl. Sci.* **2024**, *14*, 9942. https://doi.org/10.3390/app14219942

Academic Editors: Vladimir Simov Jotsov and Iqbal H. Sarker

Received: 5 March 2024
Revised: 1 October 2024
Accepted: 5 October 2024
Published: 30 October 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

IIOT is becoming increasingly valuable to operations because they cost less and can be updated with features tailored to precise user needs. Risks from IIOT connections are generally known [1]; however, much of the discussion is focused on secure communication and cloud configurations [2]. As practitioners, we have experienced difficulties in finding specific guidance on device security topics, such as secure boot, provisioning sequences, key rotation, and others, even though the device is a highly exposed component of the IIOT system. This is an extension to a practitioner's paper presented at IEEE Sec Dev Conference [3]. While the original paper [3] was limited to two pages and provided only a very brief overview of several secure boot architectures and their impact on key rotation, this paper provides detailed descriptions of the architectures, criteria for architecture evaluation, and an analysis of the architectural alternatives. It also provides a more detailed discussion of theoretical foundations and recommendations for the community.

It is important to note the key differences of Industrial Internet of Things (IIOT) from Internet of Things (IOT). IIOT application and requirements [4,5] are more frequently directly associated with the control of physical entities, such as pumps, motors, drives, etc. IIOT are more likely to involve real-time data processing [6], stringent security requirements due to potential safety outcomes [6], and operations in dynamic environments [7]. In addition to these, the heterogeneity of devices in IIOT systems further complicates the implementation of uniform security measures such as key rotation [8].

There are several decision points in the design of IIOT that impact the security of the system [9]. One is determining how to establish root of trust and chain of trust for securing device firmware [10]. These concepts span hardware and firmware and constitute "secure

boot" [11,12]. Consider the following procedures associated with secure boot. When the system resets, the root of trust, which may reside in a hardware-based secure enclave or may be based on the immutable structure of the reset function, initiates the boot process, and verifies the integrity of the first bootloader. If the bootloader's digital signature matches the expected signature, it is considered authentic and trusted. The bootloader then proceeds to authenticate and to execute the next component in the chain, repeating the process until the system is fully booted. This chain of trust is typically enabled using cryptographic algorithms (e.g., ECDSA or RSA). For example, a private key is used by the firmware developer to sign the firmware. The chain of trust in the secure boot process then uses a raw public key, a preshared key, or a certificate that is embedded with the bootloader to verify that the firmware is authentic and untampered. Many IIOT devices are cheap, low-energy, low-resource, and, as such, rely on inventive techniques to perform secure boot with minimal memory and processing requirements (this paper is mainly referring to OS-less devices, or those running RTOS for resource-constrained systems, such as Zephyr, BreezeOS, or FreeRTOS).

1.1. Threats to Firmware Signing Keys

The cryptographic material used to sign a firmware segment constitutes the weakest link because if that material is lost, then the secure boot process is compromised. Unfortunately, there are many ways that the private keys used in firmware signing are compromised [13]. Methods to compromise private keys include activities such as the following:

- Attacker gains physical access to the key storage medium;
- Attacker deceives an authorized individual with access to the signing key;
- Insider sells or loses the key;
- Attacker reverse engineers the key due to weak encryption;
- Attacker uses malware to access key storage;
- Attacker intercepts the key during a signing operation;
- Other means such as flaws in the algorithm.

As an example, Tincher estimates that about one in five private Bitcoin keys have been lost [14]. This is confounded by the fact that most key management systems have weaknesses [15], IIOT devices have long lifetimes [16], and many IIOT devices are designed and produced by small, start-up developers. This motivates the need for key rotation capability, despite the low energy, small resources, and small capacity of many IIOT. Most standards provide layers of controls to protect the devices and their systems. For example, Giaconia and Bignalet [17] provided a summary of the International Electrotechnical Commission (IEC) 62443 family of standards for securing industrial automation and control systems. While IEC 62443 standards [17] cover hundreds of security controls across 14 publications summing to over 800 pages, the Giaconia and Bignalet [17] paper includes specifically one security concept that has been difficult to implement—key rotation (specially rotation of firmware signing keys for IIOT). IEC 62443 Part 4-2 is the standard that lists security controls especially for industrial automation and controls devices, but is not explicit about requirements for key rotation—instead, it focuses on the need for verifying the integrity of the firmware, allowing for firmware updates, methods for dealing with incidents such as lost or stolen keys, etc.

1.2. Definition and Theory of Key Rotation

The key words used in this paper (e.g., key rotation, private key, public key, public key certificate, asymmetric-key cryptography) are consistent with definitions in the NIST Glossary. Key rotation is the process of regularly and systematically replacing new keys for the existing cryptographic keys used for various purposes such as signing and secure boot, authentication, or encryption. The purpose of the key rotation is to enhance security by minimizing the impact of compromised keys and reducing the window of opportunity for attackers. As such, it makes key compromise more difficult and reduces the impact of a potential/successful compromise. This paper focuses on the rotation of keys used for

signing firmware and for secure boot in devices that are resource constrained on memory and processor. Throughout this paper, we will refer to the private key as the asymmetric key used to sign the firmware segment (which might be the bootloader, the entire firmware image, or a specific set of functionality that constitutes a piece of firmware function); this private key is intended to be kept secret by the firmware maker, and its compromise results in the potential ability to sign the firmware (as long as the forger knows the algorithm, method of appending the signature, encoding, etc.). Each private key has a paired public key that can go onto the device in two forms: as a raw public key or as a certificate. We will use the term raw public key when the key itself is stored in device memory and is used to validate the firmware segment. We will use the term certificate when the public key is stored as part of a package (i.e., certificate) that contains the public key, some metadata about the key, and a signature from a key-issuing authority.

Figure 1 shows a generic concept of how code signing works for secure boot. A private–public key pair is generated in a key vault. The public key is exported and loaded into the bootloader. The firmware is hashed, and the generated hash is signed by the private key. The signed hash is appended to the firmware as the firmware signature. These artifacts are loaded onto the IIOT device. The bootloader is then able to use the public key to validate the authenticity and integrity of the firmware.

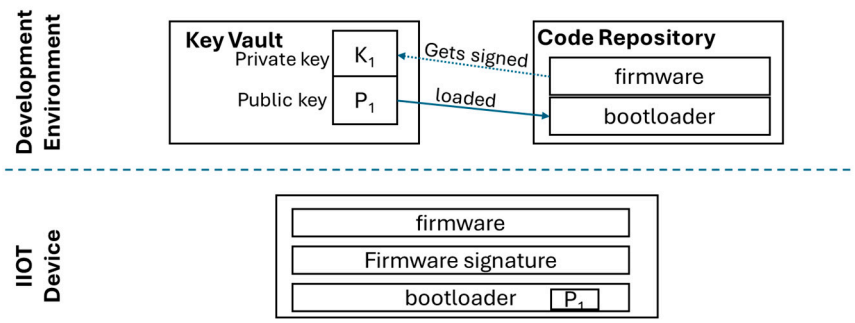


Figure 1. Generic concept of public key on device to validate firmware signature.

For key rotation to work in this context, one must generate a new private-public key pair, export the new public key, and obtain the new firmware signed by the new private key (for the new secure boot process) and signed by the old private key (to be recognized by the chain of trust by the device in operations). For key rotation to be successful, it must replace both the public key on the device and the firmware signature without violating security principles. One example of such a violation is allowing full simultaneous replacement of both public key and firmware signature without any linked chain of trust, i.e., old private key, which would allow arbitrary replacement of the firmware. Another example of such a violation is allowing the use of past keys, which would allow the reuse of compromised keys.

We make a clear distinction between key rotation and certificate rotation. Key rotation discussed in this document replaces key materials with newly generated ones as a preventive measure against cryptoanalysis of keys. Certificate rotation (not addressed in this paper) creates a new certificate for the existing key material and is used in conjunction with short-living certificates to ensure fine temporal granularity of the permitted use of keys. Certificate rotation is out of the scope of this paper. Key management, which is an organization function of how crypto keys are generated and governed according to NIST SP 800-57, is out of the scope of this paper.

1.3. Desired Attributes of Key Rotation

Different architectures for secure boot in embedded systems that support key rotation vary in their tradeoffs concerning device resilience, memory use, security of key rotation, and boot time. The most effective architecture for enabling key rotation in secure boot for embedded systems is contingent on specific application requirements and constraints.

Gebremichael et al. [6] described hardware roots of trust like hardware security modules (HSMs) and trusted platform modules (TPMs) to provide robust security features including encryption and key management, but they also increase memory use, development time, and device cost. They also described homomorphic and searchable encryption mechanisms for secure data handling in embedded systems, though they described that additional research is necessary to improve scalability and performance [6]. Jun [18] described redesign of microcontrollers to enhance boot time, reduce power consumption, and enhance device resilience. Table 1 summarizes the desired attributes of a key rotation system (labeled for ease of reference in final analysis).

Table 1. Attributes and impacts of compare key rotation capabilities of boot architectures.

Attribute	Explanation	Impacts
Product development time	Implementing key rotation should not make the creation and production of IIOT more complex.	Dev Time
Time Required to boot up	While adding layers for key rotation we should not make the product unusable by increasing boot time by an order of magnitude.	Boot Time
Ability to rotate all keys	Ideally, we want to be able to rotate any key materials that are susceptible to potential compromise.	Rotate
Regular interval of rotation	More frequent key rotation means a smaller window for compromise, but more complexity, bandwidth, and operational resources. Some of the resource-constrained devices considered in this document are battery-powered and increased utilization decreases lifetime.	Frequency
Secure key generation and distribution	The mechanism for key generation and distribution also matters, but this paper is primarily focused on the device architecture for rotation	Delivery
Proper storage	Protected limited-access storage of private key by the developer, immutable or verifiable storage of public key on device.	Storage
Independent of specific microprocessor platform	We need key rotation architectural principles that can apply across a large variety of chips and toolchains.	Independence
Memory restrictions and key deletion of revoked keys	The method must ensure that previous valid keys are revoked and unusable in the device but do not have memory or processor capability for revocation lists.	Memory
Allows multiple versions of firmware segments	IIOT devices are not continuously connected and frequently have different wake-up and bandwidth usage schemes. As such, the methods must ensure that device operations are resilient to expired keys and that the system enables multiple versions during rotation timeframes.	Versioning
Device resilience	Like above, we must ensure that the device works under a variety of scenarios that might degrade, but not eliminate security.	Resilience

This paper describes various experiences focused on the implementation of key rotation for IIOT devices. The objective is to both educate on some lessons learned and also to initiate a discussion across the community on the use of certificates for key rotation on low-energy and low-resource IIOT devices. The ability to rotate keys provides a level of adaptability for IIOT devices that will enable the preservation of customer trust and the continued and expanded use of IIOT.

2. Representative Architectures for Secure Boot

This section provides a high-level overview of example architectures to describe tradeoffs and experiences.

2.1. Minimal Immutable Bootloader (A)

To baseline the conversation, consider the idea of a small primary bootloader that cannot be modified by any firmware operation [19]. Some chips come from the vendor with an immutable bootloader and that looks to a specific memory slot for a CRC checksum or a raw public key. The raw public key is loaded at the time of manufacturing and its corresponding private key must be protected for the lifetime of the manufactured devices. Some product makers break the secure boot process by only using the CRC checksum to boot to a bootloader that contains a raw public key used to verify the firmware. This preserves the chain of trust but does not protect against private key compromise. Considering the attributes in Table 1, the most obvious challenge with an immutable bootloader is that if it contains the public key and is write protected, then there is not a default mechanism for key rotation (therefore, in tradeoff analysis in Figure 5, it scores in the lowest tier in the "Rotate" impact category.) The tradeoff is that we can optimize boot time ("Boot Time" in Table 1) and reduce time to market ("Dev Time" in Table 1). The pros and cons of this approach are summarized in Table 2 below.

Table 2. Major tradeoffs of minimal immutable bootloader as a baseline.

Pros	Cons
Quick development time to market. Optimized boot for the architecture.	No key rotation.

2.2. Immutable Bootloader with Secondary Bootloader (B)

This scenario builds on the last. In this case, we develop the primary bootloader as intended (it cannot be written to during operation). However, its only function is to validate the signature of the secondary bootloader (e.g., MCUboot or wolfBoot) [20]. The secondary bootloader is compiled with a separate public key (or certificate) that is used to validate the firmware. In this scenario, we can rotate the firmware signing key through a firmware update. However, we are still unable to rotate the key used to sign the secondary bootloader as it is immutable after operation. Considering the attributes in Table 1, the most obvious benefit is that we can now change the public key in the secondary bootloader, which means that we can now rotate the key used to validate the firmware. This improves the attribute labeled "Rotate" in Table 1 (as shown in Figure 5). We are still likely to keep a short Boot Time and Dev Time, and we also have opportunities to improve the functionality of the device independent of the defaults associated with microprocessor platform ("Independence" in Table 1). Refer to Table 3 for the tradeoffs of this approach.

Table 3. Major tradeoffs of immutable bootloader with secondary bootloader.

Pros	Cons
Cannot rotate firmware signing key. Still has rapid boot sequence. Can customize the bootloader.	Cannot rotate bootloader signing key.

2.3. Immutable Bootloader with Preslotted Multiple Public Keys (C)

Some chip vendors provide a fixed number (slots) of public keys that are configured within the immutable bootloader [21]. These preslotted keys are injected at the time of manufacturing of the device into the immutable storage (or at least the hashes of the public keys are immutable) and are unchanged throughout the life of the hardware product. Also, when hardware constraints prevent storage of public keys (e.g., ECC or RSA keys which

are relatively longer), the hash values of the public keys are configured. Every such public key, when generated, will have a corresponding private key. These are potential signing keys that will be used to sign the firmware components. These private keys are maintained by the device maker. In most cases, a secondary bootloader exists on the device, and a separate set of signing keys set is maintained by the device maker. Figure 2 depicts this configuration arrangement.

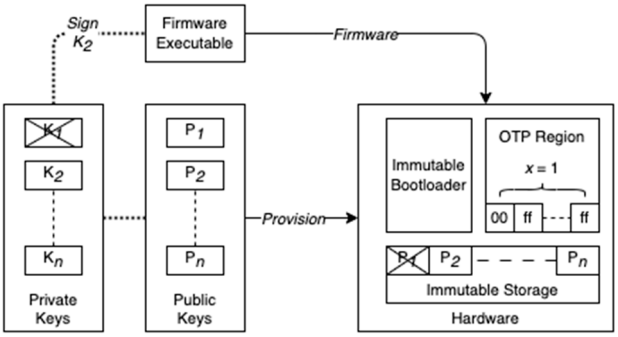


Figure 2. Preslotted keys approach.

As there are a fixed number of key pairs created, generally for an entire product line, it becomes critically important to protect these private keys throughout the life of the product. The approach, although significantly better than (A) or (B), is still limited to the preslotted number of keys. Any eventual compromise of the key management system significantly weakens the firmware's authenticity and integrity.

When a firmware package update is invoked, the public key slot must be identified for use for the firmware update operation (e.g., P_1 to P_n). One of the strategies to achieve this is an incrementing counter maintained in a secured storage or execution area accessible to the bootloader. An incoming firmware package can have a reference to the sequence number of the public key corresponding to the private key that was used to sign the firmware or bootloader. Alternatively, such a public key itself can be incorporated along with the firmware. Such a key can be compared against the set of keys that are already present on the hardware. If none match, it will be treated as an integrity violation, and the process aborts. For space-constrained devices where hash values of the public keys are stored, firmware delivery must provide the public key associated and the bootloader can then derive a hash of the public key to match against the present hash values in the immutable storage. If none match, that is an integrity violation.

At the time of the power cycle, the secure immutable bootloader verifies that it has a valid public key to use. Any mismatch there should fail the firmware upgrade process to prevent any further damage. Subsequently, it also verifies the secondary bootloader (if applicable), and that the firmware package is valid. This is performed by way of typical hash-matching based on the chosen public key. Upon successful loading of a bootloader or firmware, if a newer public key (never used before) has been used, the device must be able to determine invalid keys and have a process of revocation of keys. As most of the devices will be space-constrained, it is not possible to maintain a certificate revocation list; therefore, the revocation must be performed in a logical way. One of the ways manufacturers have achieved this is by treating on-device public keys as a sequential array. When a public key P_x is referred (by way of a direct sequence number, a matching public key via firmware package, or a matching hash value), all prior keys (i.e., $P_1 \ldots P_{x-1}$) are treated as invalid and are permanently revoked for that device. As such a revocation must be permanent, one way it is achieved is by introducing a one-time programmable (OTP) region with a monotonic counter [22] (e.g., Nordic nRF9160 and nRF5340). OTP monotonic counters work by allowing one-time half-word length write operation only where the original value of the half-word is 0xff. At manufacturing, several half-word slots are preset to a value of 0xff. The number of such slots is equal to the number of public keys provisioned. A slot

corresponding to the index of the key is written with a non-0xff value when such a public key is deemed retired. Thus, at any point in time, the number of such sequential OTP slots which have a non-0xff value indicates the number of sequential keys retired.

If a mechanism like an OTP memory region is not available, such a monotonic counter becomes a mutable element, and a motivated adversary can potentially alter its contents to effectively bring prior keys into effect.

Considering the attributes in Table 1, this method represents a sweet spot of sorts. As outlined in Table 4, Boot Time is still relatively fast because the process can be implemented very efficiently. Dev Time is still short because the architecture is predefined and there is typically a lot of documentation that one could follow to have secure boot running quickly. It allows key rotation at every level (although limited to preslotted memory for key rotation at the lowest level). Memory protections, key access restrictions, revocation processes, etc., are built into the microprocessor design and, thus, by default, this aids the developer in setting up low-energy devices with secure methods for authenticating and loading firmware. The downside comes with the number of predefined slots, limiting the frequency of key rotation.

Table 4. Major tradeoffs of immutable bootloader with multiple preslotted keys.

Pros	Cons
Relatively fast boot time. Allows key rotation. Can customize the bootloader.	May require manual intervention. Predefined keys to maintain

2.4. UPTANE Platform for Firmware Updates and Key Rotation (D)

UPTANE (Update Protocol for Transportation Network and Electronics) is a framework designed for securing over-the-air updates in automotive vehicles but is general enough to be applied to IIOT examples that use inexpensive chips and limited bandwidth [16]. UPTANE combines several security philosophies to provide this security while still enabling the system to be agile. The major examples include the following [23]:

Multiple keys/signatures—multiple independent entities sign the firmware increments.

Metadata system—a cloud system that tracks the state of firmware, keys, signatures, etc., across repositories and devices.

Separation of trust—the requirement for multiple keys, firmware increments, different signing of metadata, and firmware increments enables distributing trust as an inexpensive way to protect complex and scaled systems.

Figure 3 shows an illustration of this process in which different images on different boards are signed with different keys, images of firmware increments are provided by both the image repo, and metadata are provided by the director repo. Rotation and revocation of keys are managed through this same process. In this case, the Director repo can act as a certificate authority of sorts by lending authority to image increments, even though in this case it does not control images or the individual keys for signing images.

UPTANE works across a variety of platforms. One must have access to a secure time source. The IIOT device must have some form of secure boot implemented with access to the keys or certificates for exchange. The devices will need to be provisioned with the current set of metadata with appropriate timestamps. There will need to be signing keys unique to individual devices, and the Director repository inventory database will need public keys in addition to the provisioned devices. Finally, it will need some sort of repository mapping file to know what aspects of infrastructure will provide which information.

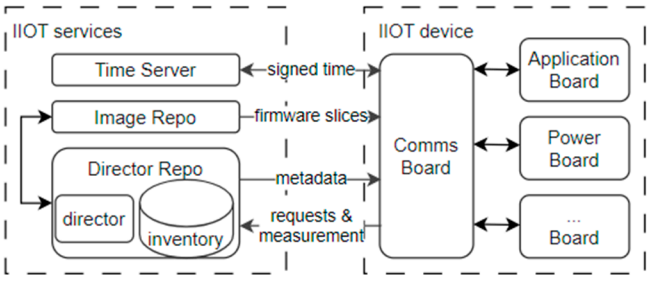


Figure 3. UPTANE design.

Considering the attributes in Table 1, this method benefits from an overall infrastructure that supports secure updates. The greatest benefit is that there is separation of trust, which means that we do not rely on a single root of trust—the IIOT services now monitor firmware measurements, which provide the ability to have remote devices with a weaker root of trust. It allows key rotation at every level—no longer limited to preslotted keys. It promotes the concept of firmware segmentation to more easily sign and update chunks of functionality without having to update the entire package. Memory protections, key access restrictions, revocation processes, etc., are built into the overall architecture through layers of controls that are not only on the device. The downside comes from the added development time required to not just develop, test, and release a device, but also connections to IIOT services and the development of the services. Table 5 gives a summary of pros and cons of this architecture.

Table 5. Major tradeoffs of UPTANE.

Pros	Cons
Relatively fast boot time. Allows key rotation. Addresses revocation via separate metadata and firmware measurement.	Requires trustworthy communication to separated repositories. Currently tailored for automotive devices only. Supply chain weakness. Can be complex to set up.

2.5. Signed Certificate in the Immutable Bootloader (E)

This approach is an extension to these prior approaches intended to improve ability to rotate keys. It still works with asymmetric key pairs with the private (signing) key in possession of the device maker. Instead of preconfiguring a fixed number of keys at manufacturing, it treats the public key as a dynamic artifact of the bootloader system. To enable verification of authenticity of the public key, it utilizes X.509 in having a Certificate Authority (Issuing CA and typically a third party) sign the public key. It requires the Issuing CA certificate and the Root CA certificate to be loaded into the primary bootloader at the time of manufacturing as immutable. The Issuing CA adds an additional root of trust. It essentially federates the trust so that secure boot is no longer solely dependent on the predetermined keys because a new key can now be validated based on a secondary, external root of trust—the Issuing CA.

A firmware update package over the air (typically) contains the following elements: binary, firmware signature, firmware public certificate, secondary bootloader signature (when applicable), secondary bootloader public certificate (when applicable)

At the time of the power cycle, the secure immutable bootloader verifies that the secondary bootloader (if applicable) and the firmware package are present. Upon success, crypto functions verify the authenticity of the bootloader public certificate using the immutable Issuing CA certificate. If the operation is successful, the secondary bootloader takes control, as outlined previously. The secondary bootloader continues in the same manner to verify the authenticity of the firmware public certificate and firmware package.

If the verification is successful, the firmware update process commences. At any point in time, if any of these checks fail, it is treated as an integrity violation and process aborts.

Figure 4 below illustrates how X.509 system is utilized, and the relationship between various elements of the system in a broader context. In cases where the firmware provider deems it necessary to rotate the bootloader or firmware signing keys, it can be achieved at the provider's key management system by rotating the private–public key pair (e.g., generating P_2 and K_2 in this context) and signing it using the Issuing CA to generate a new public certificate (C_2 in this context). Separate key pairs can be generated to sign the secondary bootloader and firmware packages depending on the security. Further bootloader or firmware signing operations immediately utilize the new key K_2. For the subsequent firmware update operation, the firmware package will contain a new version of the firmware binary, signature signed by K_2, the P_2, and optionally a new version of the secondary bootloader binary (optional), signature signed by K_2, and P_2. The firmware maker may choose to sign the firmware and bootloader with separate keys as well.

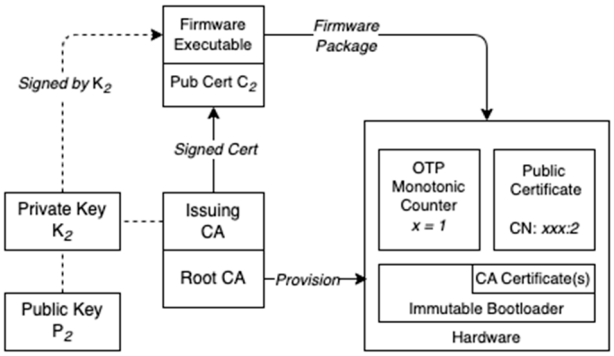


Figure 4. X.509 for firmware signing.

As an extension and improvement to the prior approach, signing key rotation is physically independent of the device and can happen without having the limitation to stay within the preslotted number of keys at the time of manufacturing. This is possible because the secondary root of trust is the CA certificate which acts as an anchor to verify any public certificate that is signed by the certificate authority the CA certificate represents. It is an important consideration, however, that a revoked or retired key pair and certificate must be treated as revoked by the device as well. For this purpose, an approach like prior architecture (C) can be used where, preferably, a monotonic counter using an OTP memory region is provisioned with several half-word OTP slots, all hydrated with the value of 0xff, which indicates that the slot is writable (or empty) [22]. The device maker can configure additional metadata within the certificate at the time of generating a certificate signing request. As an example, a common name field, abbreviated as "CN", can be leveraged for this purpose. The device maker can define a naming convention scheme to provide all the information necessary for this field to be meaningful (e.g., CN: package-name:2, where the number 2 is an incrementing identification of the certificate and means any previous certificates with the number less than 2 are deemed retired). The device maker must ensure that this number always increments from its prior value. It can start with 0, which means that no OTP memory slots are consumed (i.e., non-0xff value). Referring to Figure 4, the incoming signing certificate's CN value is xxx:2 (where xxx is the package name and the number 2 is the identifier of the certificate), whereas the monotonic counter value is 1, which means that the current certificate used is of version 1 and one slot from the monotonic counter would be burned. The CN has version number as 2, which is less than that of the identifier from the certificate. The bootloader, after all the integrity verification, increments the monotonic counter to the next number by burning the next slot within the OTP memory, thus making the number of slots used equal to the identification number provided by the certificate metadata.

Considering the attributes in Table 1, the limitations in using the above architecture come from the device limitations of processing capacity, available memory, and utilization requirements of how much delay incurred in cryptographic checks is acceptable. While the architecture supports an unlimited number of key rotations throughout the life of the product, it still places a limitation on the number of rotations (and revocations thereof) by the number of available slots in the OTP memory, also known as the ceiling of the monotonic counter. However, it will certainly be a much larger number, as storing half-words requires much less memory as compared to storing full-length cryptographic keys or their 128-bit hash values. Another important design time consideration of this system is to have the life (longer expiry) of the root certificate and the issuing certificate be long enough to outlive the device lifetime. Refer to Table 6 for a summary of tradeoffs when using this architecture.

Table 6. Major tradeoffs of signed certificate in immutable bootloader.

Pros	Cons
Unlimited key rotations. Relatively more secure.	Relatively slower boot times. Requires extra effort to address certificate revocation. Relative longer development times.

3. Discussion and Analysis of Tradeoffs

In this section, we compare the various representative architectures across the criteria listed in Table 1. In the water sector and other sectors, the low cost of IIOT devices is critical. This frequently means constraints on power, memory, and resources. While boot time is important, it is often not as critical. Establishing strong key management can be very difficult for new developers and large organizations alike. There is a tradeoff between the ability to perform quick boot and the ability to rotate signing keys throughout the long lifetimes of IIOT products. Chip vendors are frequently settling on architecture (C) and, as such, are not providing supporting utilities that enable rapid development of a certificate-based secure boot that would enable regular key rotations in a secure way. Figure 5 provides a high-level summary of the tradeoffs among the various architectures. These architectures are labeled by the letters in parentheses of each subsection in Section 1. The scale for each feature in Table 1 is a subjective scale from 1 to 5 to provide a visual of the tradeoffs, where 5 is the best possible outcome that we have seen as practitioners in the field.

Note that there is a major tradeoff that must be made between boot time, development time, and device resilience that favors the simple architectures like those described in (A), (B), and (C), and the chip-vendor-independent, and that allows for multiple simultaneous versions, and frequency of rotation associated with the more complex architectures of (D) and (E). It is worth noting that architectures (A) and (B) are quite common now in the firmware development utilities that come from chip vendors and system-on-chip (SOC) utilities. An increasing number of vendors are providing architecture (C), which is decreasing development time. Architecture (C) hits a sweet spot in that with little additional development effort, one can significantly improve the ability to rotate keys at all levels while still maximizing device resilience, boot time, memory use, etc. The implementation of architectures (D) and (E) still require the IIOT device developer to place significant time, innovation, and effort into building out the infrastructure, including crypto function utilities. However, as more chip and SOC vendors provide these capabilities in their default bootloaders and utilities, it could decrease the development time, further shifting the trade space and making key rotation on resource-constrained IIOT devices more feasible.

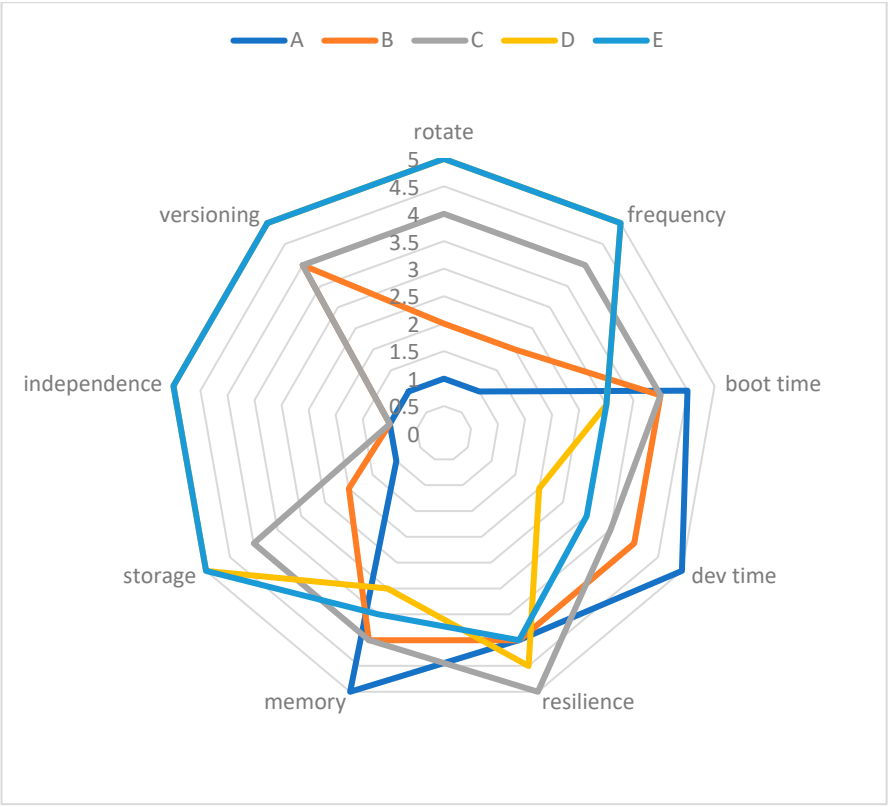


Figure 5. Tradeoffs between IIOT key rotation architectures (A) through (E) described in the previous subsections according to the measures defined in Table 1.

The cons listed for architecture (E) could be overcome through a community effort to develop the basic concepts and reusable utilities for developing bootloaders, firmware, and builds. Moreover, a simple solution to certificate revocation could be a single-byte increment that disables the use of past keys (which will need to be the topic of a future whitepaper). We recommend that the community migrate to the more prevalent use of certificate-based secure boot for even low-energy, low-resource, and inexpensive devices for IIOT.

With this general model, it is worth exploring some concepts, for example, crypto agility. Sharma [22] says that post-quantum risks are already here for long-lifetime equipment like IIOT because nation-states are implementing "harvest now, decrypt later" strategies by gathering products and keys that can be exploited after the computational capabilities are available. NIST [24] discussed this in depth with various approved post-quantum algorithms, but those are not yet available from chip vendors as a part of their utilities. Sharma [22] recommends the concept of "crypto agility" in which one not only rotates keys but also rotates key algorithms. Architecture (E) allows the rotation of the algorithm and/or the key as long as the new certificates are cross-signed by the same chain of trust as the original. Therefore, although the Issuing CA cannot be changed on the device, the signing keys for any bootloader or firmware segments can be rotated, even by using new algorithms. Unfortunately, we are not aware of any secure way to rotate the immutable certificate of the Issuing CA in architecture (E). Architecture (D) provides the same capability but requires regular connectivity to IIOT services. For architecture (D) the IIOT services promote the separation of trust and cross-checking that yield additional flexibility in what can securely be performed on the device. The bottom line is that there are ultimate limits to what can be rotated. Making the issuing CA certificate on the device immutable will protect its authenticity and integrity and allow for rotation of firmware signing keys but only within the confines of issuing CA signing specifications (e.g., signing algorithm and existence of the CA).

Thinking about post-quantum brings up the concept of IIOT constraints, such as memory, processing capacity, and network bandwidth. Architecture (C) optimizes many of these features where there are heavy restrictions on resources; however, it limits the number of low-level rotations. Architecture (D) is designed around inexpensive chips that would be found in the automotive community with constrained memory and processor capabilities; while it requires connectivity, it has been designed for limited and potentially inconsistent connections. UPTANE does require additional public keys and metadata associated with how the device communicates with the director repository. Architecture (E) does not depend as heavily on connectivity but requires additional memory for storage of a certificate.

4. Recommendations and Conclusions

Based on the discussion points and analysis mentioned above, consider the following recommendations. First, identify and evaluate overarching factors that may pose an impact or influence over a particular rotation approach. For example, organizational constraints, such as industry regulatory requirements, organizational policies, standards, and accepted norms such as risk appetite, might influence the decision to use one rotation approach over the other. As a specific case, IEC 62443 is an internationally recognized standard that covers the cybersecurity requirements of industrial automation and control systems. Part 62433-4-2 specifies capabilities that ICS components must include, depending on the associated security level. Among the IEC 62443 parts are requirements for the protection of secrets, which necessitates strong key management capabilities [17]. As a result, the longevity of the keys combined with key management requirements should be considered when deciding on a given rotation approach. Do consult qualified experts as needed, such as cybersecurity staff and legal counsel. As a guiding principle and based on perceived risks such as the impact of compromised keys, such secrets must be adequately protected, and the product must be resilient to attack. Key management is a separate topic for discussion and is out of the scope of analysis in this paper.

Second, evaluate the technical constraints presented in this paper in conjunction with risk factors such as the inherent or current risk [25] associated with the target system followed by residual risk. Inherent risk provides asset owners with a way to help rationalize cybersecurity requirements for a given system and ensures that the requirements are proportionally aligned to the inherent risk level. Specifically, inherent risk should consider factors such as the intended behavior and technical capabilities afforded a given system along with the intended deployment scenario and use cases. For example, locally accessible devices that provide read-only water consumption insights hold much less inherent risk than IIOT devices capable of remote water shut-off for an entire city. Residual risk is subsequently factored after additional controls such as rotation of certificates and keys have been applied. A key objective is to ensure residual risk is reduced to a level that falls within the risk appetite of related stakeholders such as the risk owner for the target system. As a result, consider inherent risk followed by the extent of residual risk afforded the target system when weighing options related to rotation of certificates and keys.

Based on the tradeoffs covered in the previous section, consider the following recommendations:

- Take advantage of certificate attributes such as organizational unit to set additional scope and filtering on the firmware validation process. Like access control, consider the pros and cons of keeping track of the mapping between attributes, firmware updates, and intended scopes, however.
- Understand that there is no one-size-fits-all solution. Systems with fewer requirements for fast boot can engage in more adaptive systems, while other systems with rapid boot requirements and few resources might need to accept limited rotation capabilities at the lowest level.
- Predetermine rotation practices based on the tradeoffs and implications involved with rotating certificates versus associated keys. For example, under a planned rotation

scenario, certificates should be time-bound, with planned rotation based on a predetermined percentage of the certificate expiry in years as a threshold (e.g., rotate no later than x% of expiry).

- Closely evaluate hardware platform constraints, such as processor architecture. Also, look for nuances such as related build utilities. On one hand, open systems might pave the way for increased levels of flexibility and modular use of cryptography. On the other hand, some might pose rigidity due to limited cryptographic support options.
- Look for, and utilize, suppliers that include key management solutions as part of their offering to accelerate secure IIOT implementation. Specifically, by leveraging shared services, additional value might be unlocked.
- Take advantage of IIOT services when the assurance of network latency or tolerance for retry is at least moderate. For example, reliance on shared connectivity channels could result in unplanned interruptions and intermittent signals. Where possible, couple connectivity with redundant or alternate connectivity channels such as wired and wireless should be used.

Last, but not least, it is important to always consider the long-term impact and the bigger picture. Selecting a viable rotation approach is not just about building a cybersecure system based on technical considerations. It is about building systems, that intended stakeholders consider trustworthy, which is the ultimate form of validation that the right rotation approach has been chosen.

In conclusion, selecting a certificate and key rotation approach depends on various factors tied to each organization's overarching goals and objectives. Technical considerations are one, but not the only, element involved in arriving at the right approach. Specifically, only when multiple factors are considered can a truly adaptive security approach be realized. Also, the emerging trends in using artificial intelligence show two facets. One side of AI is being widely explored by adversaries to weaken the traditional compensating controls [26]. From the defensive perspective, AI is emerging in the threat detection and mitigation [27]; however, there are many limitations that need a significant amount of work before it can be relied upon. From both of these perspectives, i.e., counteracting offensive artificial intelligence and the limitations in utilizing defensive artificial intelligence, it is important to have more evolutionary systems, and making key rotation the integral part of the system elevates the cybersecurity of the system to a significantly better posture.

Future work will be required to steer chip vendors to provide first-class utilities readily provided that support advanced cryptography methods. Signed Certificate with Immutable Bootloader (E) is based on the assumption of immutable Root Certificate, which will need further research and experimentation to address root certificate refresh in case of CA compromise, or post-quantum crypto challenges. There are several outcomes that are impacted by key rotation architecture and methodology (see Table 1 and Figure 5). Experimentation will need to consider many of these measures as the application of key rotation impacts a wide range of outcomes from both security and development efficiency. More research and experimentation is required on architectures to address key revocation. Further, this work should be connected to the overall software supply chain to derive more novel approaches to the overall problem, which is a subject worthy of a separate paper.

Author Contributions: Conceptualization: S.J., K.C. and J.R.; methodology: K.C. and J.R.; software: S.J.; validation: K.C.; formal analysis: K.C.; investigation: S.J. and K.C.; resources: S.J. and J.R.; data curation: S.J. and K.C.; writing—original draft preparation: S.J., K.C. and J.R.; writing—review and editing: S.J., K.C. and J.R.; visualization: S.J. and K.C.; supervision: J.R.; project administration: S.J.; funding acquisition: S.J. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: No new data were created or analyzed in this study. Data sharing is not applicable to this article.

Acknowledgments: We would like to acknowledge Piotr Cofta and the key rotation best practices document written for internal consumption for several product teams across Xylem—that inspired this evaluation.

Conflicts of Interest: All authors are full-time employees of Xylem, Inc. and this work was done in the course of product development and product security reviews. The authors declare that there is no commercial or financial relationships that impact this paper outside that employment and the typical relationships with component suppliers as part of Xylem business. None of these relationship represent a conflict of interest for this paper.

References

1. Sen, S.; Song, L. An IIOT-based networked industrial control system architecture to secure industrial application. In Proceedings of the 2021 IEEE Industrial Electronics and Applications Conference (IEACon), Penang, Malaysia, 22–23 November 2021; pp. 280–285. [CrossRef]
2. Abosata, N.; Al-Rubaye, S.; Inalhan, G.; Emmanouilidis, C. Internet of things for system integrity: A comprehensive survey on security, attacks and countermeasures for industrial applications. *Sensors* **2021**, *21*, 3654. [CrossRef] [PubMed]
3. Joshi, S.; Crowther, K.; Robinson, J. Adaptive security: Certificate and key rotation for firmware integrity. In Proceedings of the 2023 IEEE Secure Development Conference (SecDev), Atlanta, GA, USA, 18–20 October 2023; pp. 214–215. [CrossRef]
4. Sedrati, A.; Mezrioui, A. A Survey of Security Challenges in Internet of Things. *Adv. Sci. Technol. Eng. Syst. J.* **2018**, *3*, 274–280. [CrossRef]
5. Lynn, T.; Endo, P.T.; Ribeiro, A.M.N.; Barbosa, G.B.; Rosati, P. The Internet of Things: Definitions, Key Concepts, and Reference Architectures. In *The Cloud-to-Thing Continuum: Opportunities and Challenges in Cloud, Fog, and Edge Computing*; Lynn, T., Mooney, J., Lee, B., Endo, P., Eds.; Springer: Berlin/Heidelberg, Germany, 2020; Available online: https://link.springer.com/chapter/10.1007/978-3-030-41110-7_1 (accessed on 1 October 2024).
6. Gebremichael, T.; Ledwaba, L.P.I.; Eldefrawy, M.H.; Hancke, G.P.; Pereira, N.; Gidlund, M.; Akerberg, J. Security and Privacy in the Industrial Internet of Things: Current Standards and Future Challenges. *IEEE Access* **2020**, *8*, 152351–152366. [CrossRef]
7. Turcu, C.O.; Elena, C. Industrial Internet of Things as a Challenge for Higher Education. *Int. J. Adv. Comput. Sci. Appl.* **2018**, *9*, 55–60. [CrossRef]
8. Elkhodr, M.; Shahrestani, S. The Internet of Things: New Interoperability, Management and Security Challenges. *arXiv* **2016**, arXiv:1604.04824. [CrossRef]
9. Fluchs, S.; Taştan, E.; Trumpf, T.; Horch, A.; Drath, R.; Fay, A. Traceable security-by-design decisions for cyber-physical systems (CPSs) by means of function-based diagrams and security libraries. *Sensors* **2023**, *23*, 5547. [CrossRef] [PubMed]
10. *NIST SP 800-193*; Platform Firmware Resiliency Guidelines. NIST: Gaithersburg, MD, USA, 2018. [CrossRef]
11. Wang, R.; Yan, Y. A survey of secure boot schemes for embedded devices. In Proceedings of the 2022 24th International Conference on Advanced Communication Technology (ICACT), PyeongChang, Republic of Korea, 13–16 February 2022. [CrossRef]
12. Dent, A.W. Secure Boot and Image Authentication. Qualcomm. 2019. Available online: https://www.qualcomm.com/content/dam/qcomm-martech/dm-assets/documents/secure-boot-and-image-authentication-version_final.pdf (accessed on 1 October 2024).
13. Reffett, A.; Scanlon, R. Implications and Mitigation Strategies for the Loss of End-Entity Private Keys. Available online: https://insights.sei.cmu.edu/blog/implications-and-mitigation-strategies-for-the-loss-of-end-entity-private-keys/ (accessed on 1 October 2024).
14. Tincher, L. What Is a Private Key? How Cryptocurrency Keys Work. 2021. Available online: https://www.sofi.com/learn/content/private-key/ (accessed on 1 October 2024).
15. Kuzminykh, I.; Yevdokymenko, M.; Ageyev, D. Analysis of encryption key management systems: Strengths, Weaknesses, Opportunities, Threats. In Proceedings of the 2020 IEEE International Conference on Problems of Infocommunications, Science and Technology (PIC S&T), Kyiv, Ukraine, 6–9 October 2020. [CrossRef]
16. Serror, M.; Hack, S.; Henze, M.; Schuba, M.; Wehrle, K. Challenges and opportunities in securing the industrial internet of things. *IEEE Trans. Ind. Inform.* **2020**, *17*, 2985–2996. [CrossRef]
17. Giaconia, M.; Bignalet, X. Demystifying the ISA/IEC 62443 and Secure Elements. Microchip AN3983. 2021. Available online: https://ww1.microchip.com/downloads/en/Appnotes/Demystifying-ISA-IEC-62443-and-Secure-Elements-DS00003983.pdf (accessed on 1 October 2024).
18. Jun, K. *SRAM-Based Microcontroller Optimizes Security*; Semantic Scholar: Seattle, WA, USA, 2008.
19. Yao, J.; Zimmer, V. *Building Secure Firmware: Armoring the Foundation of the Platform*; Apress: New York, NY, USA, 2020.
20. PiEmbSys Tech. Secondary Bootloader (SBL) in Two-Stage Bootloader. 2024. Available online: https://piembsystech.com/secondary-bootloader-sbl-in-two-stage-bootloader/ (accessed on 1 October 2024).
21. Nordic. Signature Keys. 2024. Available online: https://docs.nordicsemi.com/bundle/ncs-latest/page/nrf/app_dev/bootloaders_dfu/mcuboot_nsib/bootloader_signature_keys.html (accessed on 1 October 2024).

22. Sharma, V. Cryptographic Agility: A Critical Enabler for Post-Quantum Resilience. Forbes. 2022. Available online: https://www.forbes.com/sites/forbestechcouncil/2022/10/10/cryptographic-agility-a-critical-enabler-for-post-quantum-resilience (accessed on 1 October 2024).
23. UPTANE. Uptane Standard for Design and Implementation 2.1.0. Available online: https://uptane.org/docs/latest/standard/uptane-standard (accessed on 1 October 2024).
24. NIST. Post-Quantum Cryptography (PQC) Projects Page Online. 2023. Available online: https://csrc.nist.gov/projects/post-quantum-cryptography (accessed on 1 October 2024).
25. Slabotsky, R. Inherent Risk vs. Residual Risk Explained in 90 Seconds. 2023. Available online: https://www.fairinstitute.org/blog/inherent-risk-vs.-residual-risk-explained-in-90-seconds (accessed on 1 October 2024).
26. Guembe, B.; Azeta, A.; Misra, S.; Osamor, V.C.; Fernandez-Sanz, L.; Pospelova, V. The Emerging Threat of Ai-driven Cyber Attacks: A Review. *Appl. Artif. Intell.* **2022**, *36*, 2037254. [CrossRef]
27. Olafuyi, B.A. Artificial Intelligence in Cybersecurity: Enhancing Threat Detection and Mitigation. *Int. J. Sci. Res. Publ.* **2023**, *13*, 194–200. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.