The Hidden Lifecycle Cost of Device Credentials

CoreBond Research Series — CBRS-003
August 2026
Todd Kirton
DOI: 10.5281/zenodo.22028909

What must remain synchronized, replaceable, recoverable, and trustworthy after authentication succeeds

Executive Summary

Authentication looks like an event. A device presents evidence, a verifier checks it, and a decision is made. The event may take milliseconds. The lifecycle that makes that moment possible can last years. Here, cost does not mean only dollars. It includes engineering work, operational dependency, failure exposure, recovery effort, and the institutional commitments required to keep trust working over time.

Behind a successful authentication sit provisioning, issuance, protected storage, monitoring, renewal, rotation, revocation, trust-store maintenance, ownership changes, repair, recovery, cryptographic migration, and eventual retirement. Some of that work is fundamental to maintaining trust over time. Some of it comes from the particular way an architecture represents trust through renewable credentials, certificate chains, registries, and institutional PKI dependencies.

This paper does not argue that credential lifecycle is waste or that PKI is obsolete. Many recurring lifecycle operations buy real security: revocation, compromise containment, privacy, ownership control, interoperability, and offline autonomy. The narrower question is more useful: which lifecycle obligations are unavoidable, and which are created or amplified by the credential architecture itself?

The better comparison may therefore be less about how many credentials a system carries and more about the dependency surface required to keep trust coherent. By dependency surface, this paper means the devices, services, authorities, roots, registries, policies, and records that must remain coherent for trust to function. A simple authentication event can depend on all of them.

1. The Credential Is Not Free

A device authenticates successfully. The certificate is valid. The key proves possession. The connection opens. From the outside, the system looks almost static.

Pull the camera back and the picture changes. Someone generated or injected the original credential. Something records which device it belongs to. A trust anchor has to remain valid. The device needs to know when the operational credential expires. A renewal path has to exist. Revocation must be possible after compromise. Ownership may change. Hardware may be repaired. Algorithms may age out. The manufacturer, cloud service, registry, or certificate authority may change before the device leaves service.

The lifecycle is distributed across organizations as well as software. A factory may own provisioning. A product team may own certificate code. A cloud platform may own registry state. A security team may own the CA. Field service may own recovery when hardware changes. Operations may notice the whole chain only when a device fails to reconnect. The burden is easy to overlook because no single team sees all of it at once.

NIST's TLS server certificate-management guidance describes ongoing certificate maintenance as labor-intensive and emphasizes inventory, monitoring, renewal, replacement, and disaster recovery.[1] Although that guidance is server-certificate scoped, commercial IoT platforms separately expect devices to support certificate rotation, replacement, and lifecycle monitoring.[2][3] The credential is not merely an object stored at manufacture. It participates in a system that has to keep functioning.

> ***Authentication is a moment. Trust is a system that has to survive time.***

2. Some Lifecycle Work Is Unavoidable

It would be easy to look at that machinery and conclude that persistent credentials created the problem. That conclusion does not survive a serious counterargument.

Any architecture that maintains trust over time still needs a way to establish first contact, invalidate old trust, recover after compromise, transfer authority, retire a device, and migrate cryptography. Those obligations do not disappear if X.509 disappears.

Even a verifier-held model would need durable enrollment state and a way to decide when that state is no longer valid. A hardware-derived model would still need recovery rules after legitimate repair. A behavioral model would still need baseline management, drift handling, and a way to revoke trust when the behavior model itself is compromised. Changing the representation of identity changes the mechanics, but not the fact that trust has to be maintained.

A reset device still needs some controlled basis for enrollment. A compromised relationship still needs to be ended. A new owner still needs to replace the old owner's authority. A verifier database can be lost. A secure element can be replaced. A cryptographic algorithm can become unacceptable. A long-lived controller can outlive the organization that first provisioned it.

BRSKI makes this especially clear. Its manufacturer-installed identity and Manufacturer Authorized Signing Authority (MASA) support automated onboarding, while the specification explicitly considers trust-anchor maintenance, resale, offline modes, and even the death of a manufacturer.[4] The lesson is not that BRSKI is overly complicated. The lesson is that durable trust across time is complicated.

That distinction matters. Some lifecycle work is the cost of secure trust itself. Counting all of it as credential overhead would be dishonest.

3. What Certificates Add

Once the unavoidable trust work is separated out, certificate-based systems still introduce a recognizable set of lifecycle mechanics.

Without that distinction, every certificate-management task gets blamed on the wrong thing. Some tasks exist because trust changes over time. Others exist because X.509 certificates and PKI are the chosen mechanism for expressing that trust. The second category is where the architectural comparison begins.

Consider a fielded controller whose operational certificate expires next month. The device has to know when renewal is due. It may need to generate a request, receive and install a replacement, preserve an overlap window, reconnect successfully, and recover if any step fails. The verifier still has to build and validate a certificate chain through intermediates and roots that may have their own lifetimes. Certificate-status information has to remain usable, and the trust store itself may eventually need to change. The controller can be healthy while the surrounding credential machinery is already preparing for its next transition.

RFC 9810 formalizes certificate management as an interaction among end entities, Registration Authorities, and Certification Authorities rather than a one-time issuance step.[5] AWS recommends unique per-device certificates and expects devices to support rotation and replacement.[2] Azure's current certificate-management architecture coordinates issuing authorities, device provisioning, registry state, renewal, revocation, and policy.[3][6]

None of those mechanisms is evidence of bad design by itself. They are the machinery required by this particular representation of trust.

The design tradeoff becomes visible when the credential lifetime is intentionally shorter than the device lifetime. That may be desirable for security, but it guarantees recurring lifecycle work. The architecture must therefore support continuity of the device relationship while allowing the credential that represents that relationship to expire, rotate, or be replaced.

4. When Lifecycle Becomes Availability

Lifecycle mechanics are easiest to notice when they fail.

Azure's IoT root-certificate migration is a clean example. Fielded devices needed to trust a new DigiCert root before the old trust path was retired. A device could be physically unchanged and still lose connectivity because the surrounding server-trust environment changed. Devices that did not receive the necessary trust-store update could no longer authenticate the Azure IoT endpoint and establish the connection.[7]

The migration required operators to update trust stores, avoid brittle certificate pinning, and monitor reconnect behavior. The failure path existed outside the device's own client-authentication credential: the device could still possess its credential while the TLS trust environment around it had changed.[7]

That distinction matters because it shows why lifecycle cannot be reduced to 'protect the private key.' A device may protect its key perfectly and still lose access because a root expires, a CA hierarchy changes, an intermediate is pinned incorrectly, or a platform's trust policy evolves. The credential is only one node in a larger validation path.

Let's Encrypt's DST Root CA X3 expiration exposed the same general class of problem for older clients that could not easily receive new trust roots or software updates.[8] The endpoint can remain physically healthy while its surrounding trust assumptions age out.

That does not make certificate systems uniquely fragile. It means availability can depend on keeping the surrounding trust graph current.

5. When More Lifecycle Buys More Security

A serious lifecycle analysis has to resist counting every recurring operation as waste.

AWS Greengrass provides a concrete example. Its local MQTT server certificate expires every seven days by default and can be configured from two to ten days; AWS says the limited lifetime helps mitigate impersonation risk if the certificate and private key are stolen.[9] The recurring rotation work is part of the security control.

Automotive security makes the tradeoff even clearer. ETSI's ITS trust architecture defines recurring enrollment and authorization processes, certificate-trust-list updates, CA certificate renewal, and revocation distribution.[10] Its authorization mechanisms also support privacy-preserving credential use. More lifecycle activity can therefore be intentional rather than accidental.

Revocation has the same character. Maintaining invalidation state creates operational work, but revocation is how an operator can deliberately end trust before natural expiration. A cheaper architecture that removes the work by removing the control is not necessarily a better architecture.

The same is true of credential scoping. A system may issue different credentials to reduce cross-context tracking or to enforce least privilege. That increases issuance, renewal, and revocation events, but it may also reduce the damage caused by one compromised credential. Lifecycle volume by itself therefore says very little about architectural quality.

The right question is not how much lifecycle activity exists. It is what security property each activity buys, how reliably it works, and whether the same property could be preserved with a smaller or more survivable dependency surface. Recurring complexity earns its keep when the control it provides is explicit enough to defend: shorter compromise windows, stronger revocation, privacy, scoped authority, or another concrete benefit.

6. When Authority Changes Hands

Credential lifecycle is also authority lifecycle.

FIDO Device Onboard provides a useful example because it separates manufacturing, ownership transfer, onboarding, and normal application operation into distinct trust stages. Its Ownership Voucher is passed and extended through the supply chain, and onboarding allows the new owner to establish new cryptographic control of the device.[11] FDO's security requirements also treat device security parameters, their storage, protection, movement, and destruction as explicit lifecycle concerns.[12]

That is more than changing a row in a database. Old authority has to stop being authoritative, new authority has to be established, and device security parameters have to be protected through the transition.[11][12] The exact mechanism varies by architecture, but the authority transition itself does not disappear.

An architecture that claims to reduce credential lifecycle still owes these functions. It needs a safe way to end old authority and create new authority. The representation may change, but the lifecycle event does not disappear.

This is where ownership transfer becomes a useful test for any identity architecture. If old credentials are retained, the previous owner may remain able to act. If everything is erased, the new owner may have no trustworthy path to bootstrap the device. If manufacturer roots are retained, the system must decide what power the manufacturer continues to hold. The lifecycle is not just a sequence of certificates. It is a sequence of authority decisions.

7. The Institutions Behind the Credential

The effective lifecycle of a device credential extends beyond the device. Its dependency surface includes the institutions and records that sustain the trust relationship: issuers, roots, registries, provisioning services, policy, status information, and long-term support.

A certificate can depend on a manufacturer, an issuing CA, a Registration Authority, a root program, a device registry, a provisioning service, a policy engine, revocation or status information, and long-term support for the trust store installed on the endpoint. Those institutions and records become part of the trust architecture.

A small fleet may tolerate a complex dependency surface because every dependency is well managed. A million-device fleet may amplify tiny lifecycle assumptions into operational risk. A long-lived industrial fleet may face a different problem: even if the machinery works today, every institutional dependency becomes a promise about the future. Someone must still possess the right keys, maintain the right records, publish the right status, and understand the recovery process years later.

BRSKI's manufacturer authorization service is one example.[4] Azure's root migration is another.[7] NIST's post-quantum migration work adds another dimension: organizations must understand where vulnerable public-key cryptography is used across hardware, software, and services and plan migration to new algorithms.[13]

A long-lived device may remain in service through several organizational, software, and cryptographic transitions. Its identity architecture has to survive institutional and cryptographic aging as well as hardware aging.

8. Automation Does Not Mean Absence

Modern credential lifecycle management is increasingly automated. Any fair argument has to give that fact real weight.

Cloud platforms can track certificate validity, issue replacements, maintain registries, enforce policy, and monitor fleets at scales that would be impossible with manual spreadsheets and field technicians. In a well-run fleet, automation can make certificate lifecycle cheap, quiet, and nearly invisible during normal operation. That is a genuine strength of mature PKI systems.

But automated lifecycle is still lifecycle. The issuing service has to exist. The policy has to be correct. The device has to support renewal. Monitoring has to detect failures. Recovery has to exist when the automated path breaks.

AWS IoT Device Defender includes audit checks for expiring device certificates and revoked device certificates that remain active.[14]

The important distinction is between reducing manual burden and removing architectural dependency. Automation can do the first without necessarily doing the second.

That matters during abnormal conditions. The automated path may be excellent until the issuing service is unavailable, a policy is wrong, a device clock is broken, a trust root is missing, or a recovery credential has expired. Healthy automation can make lifecycle work quiet; abnormal recovery exposes the underlying dependencies again. The point is not that automation fails often. It is that the architecture still needs a survivable answer for the day it does.

9. A Better Way to Compare Identity Lifecycles

This leaves an awkward measurement problem. Credential count is not enough. Renewal frequency is not enough. A list of administrative steps is not enough.

A better comparison asks what each architecture depends on, how critical each dependency is, how failures propagate, and how trust is recovered. The useful measures are concrete: blast radius, recovery path, offline survivability, migration burden, and the consequences of losing irreplaceable state.

It also asks where the lifecycle state lives. Some architectures keep more state on the device. Others move state into verifiers, registries, enrollment services, or relationship models. The amount of state may matter less than the durability, replaceability, and recoverability of that state. A small but irreplaceable verifier database can be a worse lifecycle dependency than a larger but easily reissued certificate population.

Which institutions must remain available? Which records must survive? What happens if a root changes? What happens if a verifier database is lost? Can the device work offline? Can compromised trust be invalidated quickly? Can ownership change without exposing prior identity? Can cryptography migrate without replacing the physical asset? Can recovery be performed without giving an attacker the same recovery path?

Those questions charge every architecture for equivalent controls. A certificate system has to answer them. A verifier-held, relational, hardware-derived, or behavioral system would have to answer them too.

The simpler architecture is not necessarily the one with fewer credentials. It is the one that preserves the controls the system actually needs with a dependency surface that remains understandable, recoverable, and survivable.

10. What Deserves to Be Discovered

The lifecycle burden of device credentials is real, but it is easy to describe it badly.

Some costs are financial. Others are engineering hours, failure modes, support procedures, institutional commitments, downtime risk, or simply the number of things that must remain correct at the same time. Without good comparative data, those should not be collapsed into one dollar figure.

It is not enough to say that certificates expire, roots change, or fleets need renewal. Those operations often exist because we want control over compromise, privacy, ownership, and recovery. It is equally incomplete to treat the successful authentication event as though the surrounding lifecycle were free. The fair question is what each architecture must keep alive, what happens when one dependency fails, and how trust is restored afterward.

Which parts of the lifecycle are unavoidable costs of maintaining trust? Which parts are specific to certificate chains, validity windows, trust-store migration, and institutional PKI? Which dependencies are reduced by automation, and which merely become less visible? Which failure modes have the largest blast radius? Which recovery paths survive when a manufacturer, CA, registry, or cloud service does not?

Any alternative architecture would have to answer those questions before claiming a lower lifecycle burden.

> *The question is not whether a credential can authenticate a device today. It is what must remain synchronized, replaceable, recoverable, and trustworthy for that authentication model to keep working years from now.*

That is not an argument against credentials. It is an argument for accounting for the full trust system that keeps them meaningful.

Research Agenda

- How should device-identity lifecycle cost be measured across labor, infrastructure, availability, recovery, and institutional dependency?
- What lifecycle machinery would a non-certificate architecture need to reproduce to preserve equivalent revocation, privacy, recovery, and offline behavior?
- How should architectures compare the survivability of CAs, registries, verifier records, manufacturer services, and device-local roots?
- Can dependency surface become a practical design metric rather than just an analytical description?

References

1. **National Institute of Standards and Technology (NIST).** NIST SP 1800-16C: Securing Web Transactions - TLS Server Certificate Management. 2020. https://doi.org/10.6028/NIST.SP.1800-16
2. **Amazon Web Services.** X.509 client certificates - AWS IoT Core. https://docs.aws.amazon.com/iot/latest/developerguide/x509-client-certs.html
3. **Microsoft.** What is Microsoft-backed X.509 Certificate Management? - Azure IoT. https://learn.microsoft.com/en-us/azure/iot/iot-certificate-management-overview
4. **Internet Engineering Task Force.** RFC 8995: Bootstrapping Remote Secure Key Infrastructure (BRSKI). 2021. https://www.rfc-editor.org/rfc/rfc8995.html
5. **Internet Engineering Task Force.** RFC 9810: Internet X.509 Public Key Infrastructure - Certificate Management Protocol (CMP). 2025. https://www.rfc-editor.org/rfc/rfc9810.html
6. **Microsoft.** Certificate Revocation and Policy Management - Azure IoT. https://learn.microsoft.com/en-us/azure/iot/concepts-certificate-policy-management
7. **Microsoft.** How to migrate IoT Hub resources to a new TLS certificate root. https://learn.microsoft.com/en-us/azure/iot-hub/migrate-tls-certificate
8. **Internet Security Research Group / Let's Encrypt.** DST Root CA X3 Expiration (September 2021). Updated 2024. https://letsencrypt.org/ca/docs/dst-root-ca-x3-expiration-september-2021/
9. **Amazon Web Services.** Device authentication and authorization for AWS IoT Greengrass. https://docs.aws.amazon.com/greengrass/v2/developerguide/device-auth.html
10. **European Telecommunications Standards Institute (ETSI).** ETSI TS 102 941 V2.1.1: Intelligent Transport Systems (ITS); Security; Trust and Privacy Management. 2021. https://www.etsi.org/deliver/etsi_ts/102900_102999/102941/02.01.01_60/ts_102941v020101p.pdf
11. **FIDO Alliance.** FIDO Device Onboard Specification 1.1. 2022. https://fidoalliance.org/specs/FDO/FIDO-Device-Onboard-PS-v1.1-20220419/FIDO-Device-Onboard-PS-v1.1-20220419.html
12. **FIDO Alliance.** FDO Security Requirements. 2022. https://fidoalliance.org/specs/fdo-security-requirements/FDO-security-requirements-v1.0-fd-20220913.html
13. **National Cybersecurity Center of Excellence (NCCoE).** Migration to Post-Quantum Cryptography. https://www.nccoe.nist.gov/applied-cryptography/migration-to-pqc

14. **Amazon Web Services.** Audit checks - AWS IoT Device Defender. https://docs.aws.amazon.com/iot-device-defender/latest/devguide/device-defender-audit-checks.html