White Paper

Shorter Lifecycle for TLS Certificates: Recommendations for the 47-Day World



Contents

Executive summary 3

Introduction 4

Certificate management challenges 6

Unique challenges with mobile applications 8

Akamai's recommended best practices for certificate management 12

Automation and DevOps in certificate lifecycles 16

Akamai's vision for next-generation certificate management 18

Executive checklist: Preparing for 47-day validity 20

Impact comparison table: 398-day vs. 47-day lifecycles 21

Executive summary

Certificates are used to secure online communication. Many organizations manage hundreds or thousands of certificates. Managing certificates and monitoring their expiration with current lifespans of 90–398 days is already complex. Changes in the industry will see certificate lifespan decrease to 47 days over the coming years. This will require organizations to rethink how to stay on top of their certificate management and certificate workflows.

In this white paper, we explain the underlying principles of certificates, why certificate lifespans are shortening, and how organizations should prepare for the changes that are coming in the industry.

For Akamai customers wanting a more in-depth analysis and discussion on their future certificate management and certificate workflows, our Global Services organization is happy to engage on these topics. Please reach out to your Akamai representative.

Introduction

The way that trust is established in the online world is through digital certificates. Your passport allows a border agent to trust your identity since the passport is issued by a trusted authority, i.e., your country.

Similarly, digital certificates are issued by a trusted authority to allow an app or a web browser to trust that websites you access are really owned by the organization they claim to be. Digital certificates are issued by a certificate authority (CA). Popular certificate authorities include Let's Encrypt, GlobalSign, DigiCert, and others.

Verifying the identity is only part of the challenge. The communication must also be encrypted to prevent eavesdropping. For certificates, cryptography and the TLS protocol are used to achieve this. A website's certificate is, therefore, essentially a digital document proving which entity in the real world owns a particular digital public key.

However, no technical standard defines whether a web browser should actually trust a particular certificate. So how does a browser decide whether to trust the certificate? As mentioned previously, that is dependent on whether the certificate is issued by a trusted CA, which prompts the question: Which certificate authorities should the browser trust?

The browser — or in some cases, the operating system (OS) — maintains a list of trusted CAs in what is called a certificate trust store. The trust store is a list of trusted CAs that is shipped as part of the browser or OS. This means inclusion or exclusion from a trust store can determine whether a certificate authority has a business at all. For website owners, it would be very cumbersome if all apps and browsers did not trust roughly the same set of CAs.

The CA/Browser Forum

To overcome the challenges described above, the CA/Browser Forum was established in 2012 for browser manufacturers (Google, Mozilla, Microsoft, Apple, and more) and certificate authorities to meet. The CA/Browser Forum is "a voluntary gathering of certificate issuers and suppliers of internet browser software and other applications that use certificates."

The CA/Browser Forum forms an integral part of the web public key infrastructure (PKI) that underpins secure encrypted communication on the modern internet. The forum discusses and decides on a set of requirements that certificate authorities should meet when issuing certificates.

In practice, the decisions reached in the CA/Browser Forum heavily affect how we secure our life online — including bank transactions, online shopping, communication with authorities, or staying in touch with family and friends.

The challenge with mis-issued certificates

A key weakness in the TLS certificate–certificate authority–trust store chain was always when a certificate authority was compromised, or cooperated to mis-issue certificates. A mis-issued certificate is one in which a CA attests that a certificate belongs to a different entity than actually acquired it.

When such an issue is discovered, the browser can of course distrust the certificate authority completely. However, that can mean not only distrust of the mis-issued certificate, but also all legitimate certificates that the CA has issued, causing massive disruption. Hence the need for certificate revocation, in which individual certificates can be distrusted.

Plenty of mechanisms have been proposed and deployed to efficiently manage certificate revocation over the years, but they have all been plagued by resource constraints, performance impact, or privacy concerns.

What is changing with how certificates are issued?

In 2025, the CA/Browser forum concluded that a key way to limit the negative impact that mis-issued certificates can have is to shorten the certificate lifespan. In ballot SC-081v3, it was agreed that by 2029, maximum certificate validity will be reduced to 47 days, with a gradual decrease of days in the years between now and then. Since many certificates were previously valid for somewhere between 90 and 398 days, this will unquestionably necessitate a dramatic change in how organizations acquire and manage certificates.

The change, while justified to improve security, introduces significant operational challenges across the digital delivery chain. It increases operational overhead, and also increases operational complexity, automation dependency, and risk of outages if renewal processes are not seamless across all entities involved.

Certificate management challenges

Most organizations manage their TLS certificates using their preferred certificate authority (e.g., DigiCert, Sectigo, Let's Encrypt, or internal PKI). For larger organizations, this can present a number of challenges. While the cost of an individual certificate usually is very reasonable, the challenges presented below tend to drive indirect costs and slow down value realization unless carefully managed.

Operational complexity

Multiple CAs and certificate hierarchies may exist, each with their own renewal processes, formats, and policies. Lack of standardized renewal workflows across different environments and vendors (cloud, on-prem, hybrid) introduce further complexities. This creates a lot of special cases that require both know-how and potentially a multitude of technical integrations. Manual coordination across customer security, IT, and application teams increase the risk of delays or misconfigurations.

Discovery and inventory

It's difficult to maintain an up-to-date inventory of all TLS certificates, especially in large organizations and with shadow IT. Certificates tied to legacy systems or undocumented endpoints may be overlooked, causing unexpected outages.

Renewal timing and automation

Missing renewal deadlines results in expired certificates, leading to service outages, broken integrations, or loss of customer trust. CA approval flows (organizational validation, DNS/HTTP challenges) may not complete in time. Automating renewals is complex when customer CAs do not provide APIs or automation-friendly tooling. Staggered expiration dates across many certificates complicate scheduling and tracking.

Dependencies and integration challenges

Certificates may be tightly coupled with load balancers, web servers, or application gateways; renewal requires coordinated redeployment. Some systems require service restarts or downtime to apply renewed certificates. Integration with external SaaS providers or third-party services may require re-upload of new certificates.

Security and compliance

Mismanagement during renewal may lead to weak keys, outdated algorithms, or use of noncompliant certificate chains. Risk of introducing insecure practices (e.g., reusing old keys, failing to rotate keys). Meeting audit and compliance requirements (PCI DSS, ISO 27001, etc.) adds additional overhead.

Customer-specific CA limitations

Some customer-managed CAs have outdated software, poor logging, or limited visibility into certificate lifecycle events. Self-signed or private CA chains require special handling in apps and devices. Internal policies may prevent delegation or automation, requiring manual approval workflows.

Governance and ownership risk

Dependence on individual administrators' knowledge rather than institutionalized processes can be an issue. There may be errors in CSR (Certificate Signing Request) generation, wrong SAN (Subject Alternative Name) entries, or mismatched key lengths during renewals. Team silos (network vs. app vs. security teams) slow down the process. There may also be no defined authority to enforce certificate policies, renewal SLAs, or standard tooling. Each team or application owner handles certificates ad hoc, resulting in inconsistent practices, missed renewals, and finger-pointing during outages.

Unique challenges with mobile applications

Unlike web platforms, where trust decisions are largely decoupled from client release cycles, mobile applications embed trust logic directly into distributed binaries. As certificate rotation frequency increases, this architectural mismatch becomes a primary source of application instability.

Certificate pinning failures and “bricked” applications

Traditional **leaf certificate pinning**, where a mobile app pins a specific end-entity certificate, becomes a high-risk practice under short-lived certificate regimes. In this model, if an application pins a certificate (Cert_A) and the server rotates to a new certificate (Cert_B) at the end of the validity window, the application detects a hash mismatch and terminates the TLS connection.

This results in a **“bricked” application state**, where users encounter generic network errors and are unable to authenticate or perform API operations. Recovery is dependent on releasing a new application version and waiting for user adoption, creating unavoidable downtime.

Long-tail user impact

Mobile update adoption follows a long-tail distribution. Auto-updates are not universally enabled, and many users remain on older app versions for weeks or months. Under a 398-day certificate model, this delay was manageable. Under a 47-day model, the margin for delayed adoption disappears.

As soon as a server certificate rotates, users running older application versions experience **near-immediate service denial**, despite the server operating correctly. This disproportionately affects regions with slower update adoption and users on older devices.

Third-party API and SDK dependency risks

Modern mobile applications integrate multiple third-party services such as payment gateways, analytics platforms, fraud detection engines, and customer support SDKs. If these third parties adopt short-lived certificates and the application pins their certificates, the app can break due to **external certificate changes outside the organization's control**.

This introduces systemic fragility where availability depends on third-party rotation schedules and communication practices.

Clock skew and partial rollout sensitivity

Short certificate lifetimes amplify otherwise minor issues:

- **Clock skew:** Devices with incorrect system time may interpret certificates as expired or not yet valid, leading to TLS failures even when the server configuration is correct.
- **Partial rollouts:** In multi-region or multi-CDN deployments, inconsistent certificate propagation can result in region-specific or intermittent failures, complicating detection and incident response.

The mobile asymmetry gap: Why mobile apps are more affected than browsers

Mobile platforms suffer from a fundamental asymmetry between **server-side cryptographic agility** and **client-side update realities**.

- **Release and adoption delays:** Server certificates can be rotated in seconds using automation, while mobile updates require development, QA, App Store submission, review, and gradual user rollout.
- **Embedded trust logic:** Mobile applications frequently embed static trust assumptions — such as pinned certificates or keys directly into the binary.
- **Inability to force updates:** Organizations cannot mandate immediate updates across the entire install base, leaving a segment of users perpetually vulnerable to certificate-related outages.

Browsers, by contrast, typically rely on platform trust stores and avoid hard-coded trust material, allowing them to adapt to certificate rotation without user intervention.

Best practices for achieving crypto-agility

To maintain high availability in a rapid-rotation environment, mobile architectures must **decouple trust logic from the application release cycle**.

Safer pinning strategies

- **SPKI (Subject Public Key Info) pinning:**
  Applications should pin the public key rather than the certificate itself. Certificates can be reissued with new expiry dates while reusing the same key pair, allowing multiple rotations without breaking trust. However, in regulated industries this may not be a valid approach.
- **Planned key rotation:**
  Key reuse must be time-bounded. Organizations should define a rotation window (e.g., annually) and ship overlapping pin sets to avoid outages during key transitions.
- **Backup pins:**
  Applications should include multiple pins: a primary production key, a secondary standby key, and an emergency offline key stored securely (e.g., HSM or cold storage).
- **Avoid leaf certificate pinning:**
  Pinning an Intermediate CA or higher trust anchor offers greater stability, though with broader trust scope. This approach is often acceptable when combined with strict hostname validation.

Dynamic and remote pinning models

Dynamic pinning removes static trust material from the application binary. In this model, the application retrieves a **cryptographically signed configuration** (e.g., JSON or JWT) from a secure endpoint at runtime, containing the current set of trusted pins.

Key requirements:

- The configuration must be signed and verified client-side.
- Unsigned or improperly validated updates must be rejected.
- Versioning, TTLs, and rollback protection must be enforced.

This approach enables real-time trust updates across the install base without App Store submissions.

DevOps automation and key management

- **ACME-based automation:**
  Manual certificate issuance and deployment is infeasible at 47-day intervals. Automatic Certificate Management Environment (ACME) clients must be used for issuance, renewal, and deployment.
- **Key reuse policies:**
  For SPKI pinning to function correctly, certificate automation must reuse private keys across multiple certificate renewals within a defined window.
- **Rollout validation:**
  Multi-region synthetic checks and canary endpoints should validate certificate health prior to and during rollout.

Monitoring, observability, and incident response

Robust observability is essential to detect and mitigate TLS-related failures.

- **Granular metrics:**
  TLS failures should be categorized by cause, including pin mismatch, certificate expiry, hostname mismatch, and clock skew.
- **Segmentation:**
  Metrics should be segmented by app version, OS version, region, and ASN to identify affected populations.
- **Emergency kill switch:**
  A server-driven mechanism should exist to disable or relax pinning for specific app versions in the event of a catastrophic failure.



Akamai's recommended best practices for certificate management

In the era of shrinking certificate lifespans, merely issuing certificates isn't enough. Organizations must treat certificate lifecycle management as a continuously operating service.

Recommendations:

1. Maintain a centralized certificate inventory and metadata

A centralized certificate inventory is fundamental to effective certificate management. Organizations must always know where certificates are deployed, who owns them, and when they expire. All edge, origin, and internal certificates should be tracked in a single authoritative inventory, with metadata such as ownership, environment, domains and SANs, issuing CA, and deployment locations. Issued certificates should be continuously reconciled with deployed endpoints to detect drift or orphaned certificates. Without accurate inventory and discovery, certificate automation and reliable renewal are not possible.

2. Centralize certificate lifecycle management

Organizations should centralize certificate lifecycle management across all environments, including edge, origin, and internal services. End-to-end automation must span certificate enrollment, renewal, deployment, rotation, and decommissioning, while preserving consistent metadata for auditability, ownership, and operational visibility.

Certificate lifecycle operations should be driven by API-based workflows integrated into CI/CD and infrastructure automation pipelines rather than manual processes. Human-in-the-loop renewals and ad hoc certificate handling introduce operational risk, obscure accountability, and do not scale as certificate validity periods shrink and rotation frequency increases. This applies equally to edge-facing certificates and origin certificates, which are often more numerous, more distributed, and more operationally fragile.

A centralized, automation-first approach ensures consistent policy enforcement, reliable rotation, and uniform governance across all certificate types, regardless of where they are deployed or which platform or vendor is used to provision them.

3. Automate renewal triggers and validation

Certificate renewal should be driven by automated workflows that monitor expiration thresholds and certificate change status rather than calendar reminders or manual intervention. By integrating API checks into operational pipelines, organizations can detect renewal failures early, automatically handle pre-verification warnings, and ensure certificates are always deployable before they reach critical expiration windows, significantly reducing outage risk. Early detection of failures is essential to preventing service disruption.

4. Control deployment through change management and scheduling

Certificate deployment should follow the same release discipline as application changes. Using deployment scheduling and change management capabilities allows organizations to stage certificate rollouts, align them with maintenance windows, apply phased deployments, ensure a rollback path in case of unexpected issues, and track deployment status across all endpoints.

5. Design for failure and operational resilience

In environments with frequent certificate rotation, failures are inevitable and must be explicitly anticipated. Certificate lifecycle workflows should be designed to fail safely, recover automatically, and surface issues early.

Renewal and deployment pipelines should be idempotent, include automated retries, and generate alerts well before certificates approach expiration. When automation fails, it should be treated as a production incident with clear ownership and escalation paths.

Operational resilience is achieved not by preventing all failures, but by ensuring failures are detectable and recoverable, and do not result in service disruption.

6. Treat certificate metadata as an observability signal

Certificate state should be continuously monitored just like application health. By extracting expiration data, cryptographic parameters, and trust chain details through the API, organizations can build proactive alerting, compliance reporting, and incident diagnostics around TLS, transforming certificates from invisible infrastructure into measurable operational assets.

7. Embed certificate policy checks into CI/CD pipelines

Integrating CLIs and Terraform modules into CI/CD pipelines enables teams to automatically enforce standards, prevent deployments that rely on expiring or noncompliant certificates, and ensure that trust posture evolves at the same speed as application releases.

8. Automate third-party CA and CSR workflows

For environments that rely on external certificate authorities, automation must extend beyond Akamai-native issuance. By programmatically generating CSRs, ingesting signed certificates, and validating deployments through CPS APIs, organizations can ensure that even OV and EV certificate workflows fit seamlessly into automated pipelines, eliminating bottlenecks that would otherwise become critical failure points under short certificate lifetimes.

An edge-protected app has two certificates

In the context of an edge platform like Akamai's, there is actually not one, but two certificates in play. One certificate is the certificate presented to an end user from the edge platform, and that is the one we mostly discuss in this paper. However, the origin where the edge platform fetches the content is also presenting a second certificate toward the edge platform.

Just as a browser verifies that it is talking to a server with a valid certificate, the edge platform also verifies the certificate sent by the origin to prevent a rogue origin from replacing the actual origin. This is how a chain of trust is maintained through the entire communication flow.

Unlike a browser, the edge platform is much more configurable in terms of how it should establish trust. Plus, unlike a browser — which may require a whole new software version to change its trust setting — the edge platform can be reconfigured quickly.



Prior to the changes discussed in this paper, Akamai's recommendation would in most cases have been to maintain a certificate issued by a public CA on the origin and then use the Akamai trust store to verify the origin identity.

In the 47-day world, this becomes a less obvious choice, and going forward our recommendations are as follows:

- Provided the origin allows for **smooth automated origin certificate updates**, then continue using a public CA on the origin and the Akamai trust store to verify the origin certificate.
- If rotating the origin certificate **requires manual intervention**, we recommend that customers pin a specific origin certificate in the edge configuration. An annual or biannual process then needs to be established to rotate the origin certificate and pin the new certificate in the edge configuration. This approach should strike a balance between manual work required and periodically refreshing the cryptographic material to maintain a good security posture.

Note: Taking the second approach should be considered a risk and could only be seen as a stopgap measure while automation is implemented, which is the topic of our next section.



Automation and DevOps in certificate lifecycles

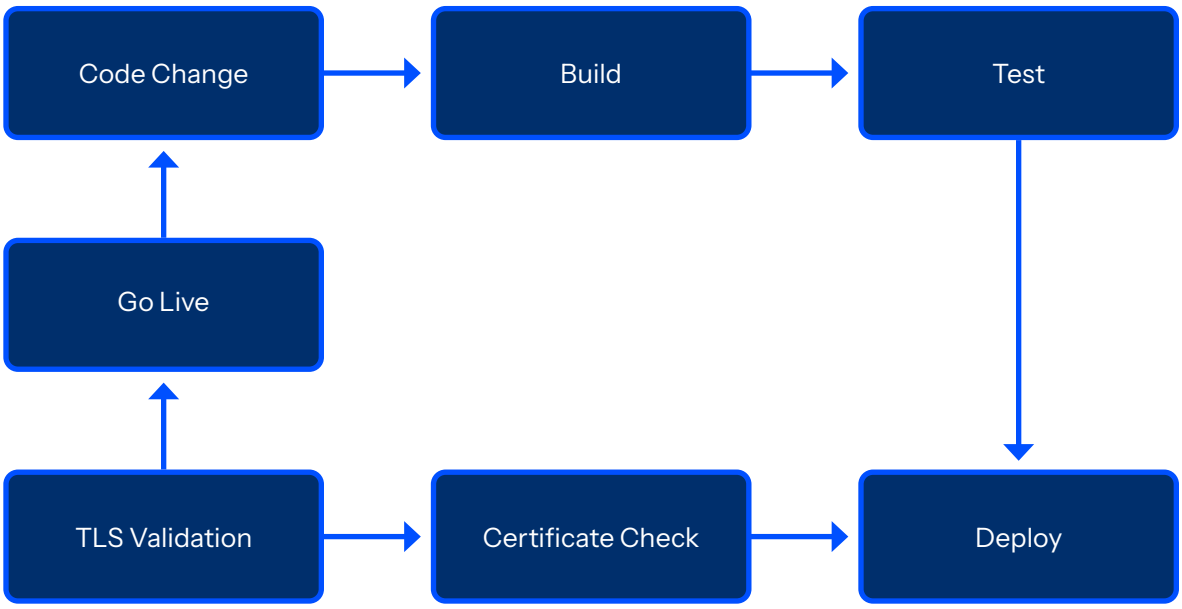
In a 47-day world, trust infrastructure must operate at the same speed as software delivery. This requires a fundamental shift: Certificates must become first-class components of DevOps and CI/CD workflows, powered by automation rather than manual intervention.

In modern environments, certificate health — edge, origin, and internal — directly affects application availability, deployment velocity, and incident response. An expired or misconfigured certificate can disrupt production just as effectively as a failed code release. For this reason, leading organizations are embedding certificate health into the software supply chain, ensuring that trust posture is validated continuously rather than assumed implicitly. Build pipelines increasingly include checks for certificate expiry, cryptographic compliance, and trust chain integrity, while release workflows enforce policy gates that prevent deployments from proceeding when certificate risks are detected.

Limit or remove human intervention

This approach reflects a broader principle: In high-frequency rotation environments, humans must be removed from the critical path of certificate operations. Manual renewals, ticket-based workflows, and calendar-driven processes introduce latency and inconsistency that simply do not scale. Instead, successful teams adopt event-driven automation models, where certificate issuance, renewal, and deployment are triggered programmatically and executed through repeatable pipelines. In this model, reliability comes not from individual expertise, but from system design. Pipelines should handle failures gracefully, with automated retries, backoff, alerting, and clear escalation paths, ensuring that outages are avoided even when issues occur.

Certificate management tools



Some enterprise certificate management tools provide centralized discovery, policy enforcement, and secure key management, ensuring that cryptographic standards are applied consistently across the organization. At the same time, they integrate with CI/CD pipelines and delivery platforms to ensure that governance does not come at the expense of speed. Security teams retain control over keys and policies, while platform and application teams maintain the agility required to support continuous delivery. Observability metrics — such as coverage of automated renewals, deployment success rates, and time-to-expiry thresholds — enable teams to track system health and maintain confidence in rapid rotation workflows.

In this architecture, CI/CD pipelines provide automation, validation, and release control. Delivery platforms such as Akamai execute deployment and enforce trust at scale. Together, these layers form a system in which certificates are no longer fragile dependencies managed in isolation, but fully automated, auditable components of modern application delivery.

Customers looking to get started with automation for Akamai or enhance existing CI/CD approaches are welcome to contact Akamai's Global Services for expert guidance.

Akamai's vision for next-generation certificate management

As certificate lifecycles become shorter, automation is more critical. Akamai's vision for certificate management is rooted in the belief that certificates should be secure, automated, observable, and decoupled from infrastructure complexity, while still aligning with enterprise trust models and governance requirements.

Akamai identifies the common set of challenges that enterprises face:

- Increasing reliance on short-lived certificates and frequent rotation
- Manual processes that do not scale with modern CI/CD pipelines
- Fragmented certificate management across multiple CAs and tooling ecosystems
- Limited observability into certificate health, expiration risk, and failures

Akamai's approach is focused on simplifying this complexity, not by abstracting away enterprise controls, but by enabling automation that respects existing CA relationships, validation models, and security policies.

Akamai believes certificate management should have the following platform capability:

- **Standards-based:** Leveraging widely adopted protocols such as ACME to ensure interoperability and future-proofing
- **API-first:** Designed for programmatic access and automation, rather than portal-driven workflows
- **CA-agnostic:** Supporting customer choice of certificate authorities while maintaining a consistent control plane

ACME and external account binding as an industry direction

Within this broader vision, ACME — particularly with External Account Binding (EAB) — represents an important industry trend rather than a single feature commitment.

ACME EAB addresses a critical gap in enterprise certificate automation by allowing authenticated, policy-driven interactions with commercial certificate authorities. This enables automation while preserving:

- Enterprise identity and authorization controls
- Existing CA validation and approval workflows
- Clear separation of responsibilities between platform provider, CA, and customer

From Akamai's perspective, ACME EAB aligns well with the long-term goal of enabling **secure, zero-touch certificate lifecycle automation** without weakening trust models or compliance requirements.

Looking forward

Akamai's long-term vision for certificate management is to provide a unified, extensible foundation that evolves alongside industry standards, customer needs, and security best practices. **Akamai Cloud Certificate Manager** represents an important step in this journey and serves as a proving ground for next-generation certificate management concepts — rather than a static or final solution.

From Akamai's perspective, ACME EAB is a key enabler in our long-term vision to deliver secure, zero-touch certificate lifecycle automation through Cloud Certificate Manager, ensuring compliance and trust while evolving with industry standards and customer needs.

By focusing on standards, automation, and enterprise trust alignment, Akamai aims to help customers move from reactive certificate operations to intentional, automated certificate governance across the platform.



Executive checklist: Preparing for 47-day validity

To maintain operational resilience and security in this high-velocity environment, leadership should ensure the following actions are prioritized.

For Akamai customers looking for a more tailored approach, our Global Services team is here to help. Please contact your Akamai representative.

Immediate tactical actions

- **Audit certificate inventory:** Establish a single source of truth for all certificates, including location, usage, and expiration.
- **Identify manual dependencies:** Map all renewal workflows that currently require human intervention, such as manual CSR generation or email-based approvals.
- **Review mobile pinning:** Audit all mobile applications for "leaf certificate pinning" and transition to public key (SPKI) pinning to avoid application failure during rapid rotations.

Strategic automation goals

- **Standardize on ACME:** Deploy ACME-based automation to handle high-frequency issuance and renewals across all environments.
- **Integrate with CI/CD:** Embed certificate policy checks and deployment triggers directly into existing software delivery pipelines.
- **Leverage centralized governance:** Consolidate management onto a platform like Cloud Certificate Manager to maintain a unified control plane across multiple certificate authorities.

Operational resilience

- **Establish backup pins:** Ensure mobile and embedded systems include secondary standby pins for emergency failover.
- **Monitor mission-critical metrics:** Treat certificate metadata, such as expiration and issuance sources, as mission-critical observability signals.
- **Synchronize edge and origin:** For CDN-fronted applications, ensure automated workflows are in place to keep origin certificates and edge configurations in sync.

Impact comparison table: 398-day vs. 47-day lifecycles

Feature	Traditional model (398 days)	New mandate (47 days)	Impact assessment
Renewal frequency	~1 renewal per year	~8-9 renewals per year	9x increase in operational workload per certificate.
Domain control validation (DCV) reuse	Reusable for up to 398 days	Reusable for only 10 days (as of 2029)	Proof of domain ownership must be established almost weekly.
Management method	Manual (Spreadsheets/ Calendars)	Full Automation (ACME/API-first)	Manual processes "mathematically impossible" at this scale.
Risk exposure	12 months for a compromised key to be valid	< 7 weeks for a compromised key to expire	Dramatically reduces the "blast radius" of security breaches.
Outage potential	Annual risk; often manageable	Continuous risk; razor-thin margin for error	A single missed renewal can lead to near-immediate service denial.
Organizational impact	"Set and forget" administrative task	Continuous DevOps/ CI/CD imperative	Requires deep integration into technical delivery pipelines.

Akamai is the cybersecurity and cloud computing company that powers and protects business online. Our market-leading security solutions, superior threat intelligence, and global operations team provide defense in depth to safeguard enterprise data and applications everywhere. Akamai's full-stack cloud computing solutions deliver performance and affordability on the world's most distributed platform. Global enterprises trust Akamai to provide the industry-leading reliability, scale, and expertise they need to grow their business with confidence. Learn more at akamai.com and akamai.com/blog, or follow Akamai Technologies on X and LinkedIn. Published 03/26.