BROADCOM
MAINFRAME SOFTWARE

Digital Certificates on the Mainframe

Securing the Mainframe with Digital Certificates

White Paper

Table of Contents

Chapter 1: Introduction 6
1.1 Document Details 6
Chapter 2: Understanding the Certificate Challenge 7
2.1 Real-World Scenario: Critical Alert 7
2.2 The Three Core Challenges 8
2.3 Modern Access Tool Integration 8
2.4 API Security and AT-TLS 8
2.5 Certificate Lifecycle Management 8
2.5.1 Reducing the Validity Time of Digital Certificates 9
2.5.2 The Broader Impact 9
2.5.3 Compliance Lens: DORA, NIS2, and PCI DSS 9
2.6 Key Takeaways for Certificate Challenges 10
Chapter 3: Digital Certificates in the Mainframe World 11
3.1 The Modernization Imperative 11
3.2 Why REST APIs Changed Everything 11
3.3 Real-World Scenario: The Certificate Explosion 11
3.4 Beyond REST APIs: The Broader Certificate Landscape 12
3.5 Digital Certificate Creation and Management 12
3.5.1 Step 1: Generate the Digital Certificate owned by the User 12
3.5.2 Step 2: Create and Manage Key Rings 13
3.5.3 Reference Documentation 13
3.6 Managing Your Certificate Environment 13
3.7 The Path to Certificate Maturity 13
3.8 Looking Forward 14
3.8.1 Building Your Foundation 14
Chapter 4: Demystifying Digital Certificates for Mainframe Security 15
4.1 Core Concepts of Digital Certificates 15
4.2 Certificate Fundamentals Explained: CA and Trust Hierarchies 15
4.3 Public and Private Key Infrastructure 16
4.4 Validity Period and Renewal 16
4.5 Real-World Applications in Mainframe Environments 17
4.6 Common Pain Points with Middleware Integration 18
4.7 Real-World Scenario: Cross-System Communication Failure 19
4.8 Mutual Authentication of Client Certificates 19
4.9 ESM Configuration Examples 19
4.10 Validating Security End-to-End 19
4.11 Zowe: Effectively Deal with Certificates in the Client Platform 20

4.12 Zowe: Bridging Mainframe and Modern Clients 20
4.13 Real-World Scenario: Zowe in a Regulated Financial Environment 21
4.14 Key Takeaways for Zowe Environments 22
Chapter 5: The Building Blocks of Trust **23**
5.1 Trust in the Mainframe Context 23
5.2 Anatomy of a Digital Certificate 23
5.2.1 The Trust Chain in Action 25
5.2.2 Components of a Trust Chain 25
5.3 Viewing and Verifying Certificates by ESM 26
5.4 Understanding Key Rings 26
5.5 Certificate Deployment: Real-World Flow 26
5.6 Troubleshooting Common Issues 27
5.7 Best Practices 27
5.8 Wildcard Certificates: Operational Convenience at a Security Cost 27
5.8.1 What are Wildcard Certificates 27
5.8.2 Why We Do Not Use Wildcard Certificates 28
5.9 Regulatory Guidance and Security Standards 28
5.10 Trust and Compliance 29
5.11 Wildcard Certificates 29
5.12 Key Takeaways for Certificate Security 29
Chapter 6: The Definitive Guide to Certificate Management **30**
6.1 Planning Phase 30
6.1.1 Requirement Gathering 31
6.2 Security Policy Alignment 31
6.3 Resource Allocation 31
6.4 Certificate Creation and Issuance 32
6.4.1 Certificate Generation 32
6.4.2 ESM Certificate Generation Examples 32
6.5 Certification Signing Request Generation 32
6.6 Validation Process 32
6.7 Installation and Configuration 33
6.7.1 Installation and Configuration: Verification Process 34
6.7.2 Testing Protocols 34
6.8 Certificate Monitoring, Expiration, and Renewal 34
6.9 Renewal Process 35
6.9.1 Advance Planning 35
6.9.2 Renewal Procedures 35
6.10 Emergency Procedures 35
6.10.1 Revocation Process 35

6.10.2 Incident Response 36
6.10.3 Recovery Protocols 36
6.11 Best Practices for Mainframe Environments 36
6.12 Troubleshooting and Problem-Solving 36
6.12.1 Private Key Not Found 36
6.12.2 Key Ring Not Found 37
6.12.3 Certificate Insertion Issues 37
6.13 First-Line Documentation for Troubleshooting 37
6.14 Key Takeaways 37
Chapter 7: Common Real-World Scenario: Paradigm Shift 38
7.1 Actionable Steps toward PQC Readiness 39
7.1.1 Comprehensive Cryptographic Inventory 39
7.1.2 Risk Prioritization Framework 39
7.1.3 Test PQC Algorithms in Controlled Environments 39
7.1.4 Design Hybrid Cryptographic Architectures 39
7.1.5 Zero Trust: Identity as the New Perimeter 39
7.2 Key Zero Trust Certificate Practices 40
7.2.1 The IoT Revolution Meets the Mainframe 40
7.2.2 Artificial Intelligence and Machine Learning: The Game Changers 41
7.2.3 Assessing Your PKI Maturity Level 42
7.2.4 Understanding the PKIMM Framework 42
7.3 PKIMM Maturity Levels 42
7.4 Conducting Your Assessment 42
7.5 The Path Forward 43
7.6 Conclusion 43

Chapter 1: Introduction

This sentiment, expressed by a seasoned mainframe security specialist, reflects a common challenge in our evolving technical landscape. While mainframe professionals excel at maintaining secure systems, the growing complexity of certificate management can seem daunting even to experienced professionals.

This document addresses the specific needs of mainframe security specialists and systems programmers who face increasing pressure to implement and maintain certificate-based security solutions. We focus on practical, common, real-world scenarios that you likely encounter in your daily operations.

"I've managed mainframe security for over a decade, but digital certificates still present new challenges every day."
– A Security Leader

1.1 Document Details

You will find the following information in this document:

- Practical guidance on implementing modern tools with secure and properly configured digital certificates
- Strategies for establishing strong certificate management processes that align with audit and compliance requirements
- Step-by-step instructions for integrating certificates into your mainframe environment using your external security manager (ESM)
- Industry best practices for managing the full certificate lifecycle, from issuance to renewal and revocation
- Troubleshooting techniques to identify and resolve common certificate-related issues effectively

By the end of this document, you will have the following information:

- A clear understanding of certificate concepts relevant to mainframe operations
- Practical, tested procedures for certificate management
- Ready-to-use commands and configurations for your specific ESM
- Systematic approaches to certificate-related problem-solving

Chapter 2: Understanding the Certificate Challenge

Why can digital certificates ruin your day, and how can you prevent this from happening?

2.1 Real-World Scenario: Critical Alert

It is Monday morning, coffee in hand, the mainframe security team receives a critical alert:

Critical alert: Multiple Certificate Issues Detected
SSL HANDSHAKE FAILURE IN CICS REGION PROD1
Certificate: CN=APIGATE.ENTERPRISE.COM expired at
2024-02-11 23:59:59
- Impact: API Authentication Services Affected
- Status: Production Impact

Meanwhile, a systems programmer is scrambling. A new application rollout failed.

Problem: API Service Deployment blocked
Error: Certificate validation error in AT-TLS configuration
Impact: New application rollout halted
Priority: High

By mid-morning, help desk tickets are piling up. Teams are tracing errors, rebooting services, and fielding questions from management.

Root Cause: The certificate expired and nobody noticed.

Certificate management which was once seen as a low-priority, back-end responsibility, has now become central to enterprise security and operational resilience, especially on the mainframe. Application programming interfaces (APIs), encrypted channels, and cross-platform authentication all depend on digital certificates that must be trusted, valid, and properly configured.

When certificate management fails, services crash, deployments stall, and audit teams start asking tough questions.

This chapter introduces the daily challenges faced by mainframe teams, explains what is at stake, and outlines how small issues can lead to major disruptions unless organizations take a proactive, collaborative approach.

2.2 The Three Core Challenges

Challenge	Description	Impact	Who Is Involved
Modern access tool integration	Secure connections require both ends to trust the same certificate authority chain.	API call from cloud gateway fails due to incomplete trust chain.	Security team, ESM admins
API security through Application Transparent Transport Layer Security (AT-TLS)	AT-TLS policies secure data in transit without touching application code.	Expired cert in TCP/IP stack blocks service deployment.	Network team, middleware, systems programmers
Certificate lifecycle management	Certificates must be inventoried, tracked, renewed, and rotated (automatically when possible).	Expired certificate halts production due to missed renewal window.	Everyone, especially ops and security

Let us take a closer look at these challenges.

2.3 Modern Access Tool Integration

Modern workloads, whether on mainframe, cloud, or mobile often use mutual TLS (mTLS) to verify both the client and the server. If the certificate trust is not properly configured on either side, authentication fails.

Risk: Application-to-mainframe connections are rejected because the ESM does not recognize the certificate authority (CA), or the chain is incomplete.

2.4 API Security and AT-TLS

Many organizations rely on AT-TLS to enforce TLS encryption without changing application logic. While powerful, AT-TLS introduces new complexity:

- AT-TLS policies are stack-level, not application-level.
- Certificate management happens in the ESM.
- Coordination with middleware and networking teams is essential.

Failure modes:

- Misconfigured AT-TLS policies
- Expired or missing certificates
- Mismatch in cipher settings or trust anchors

These failures often result in halted deployments and service unavailability.

2.5 Certificate Lifecycle Management

Certificates have expiration dates, just like passports. If they are not tracked and renewed on time, the consequences can be dramatic. The following common problems exist:

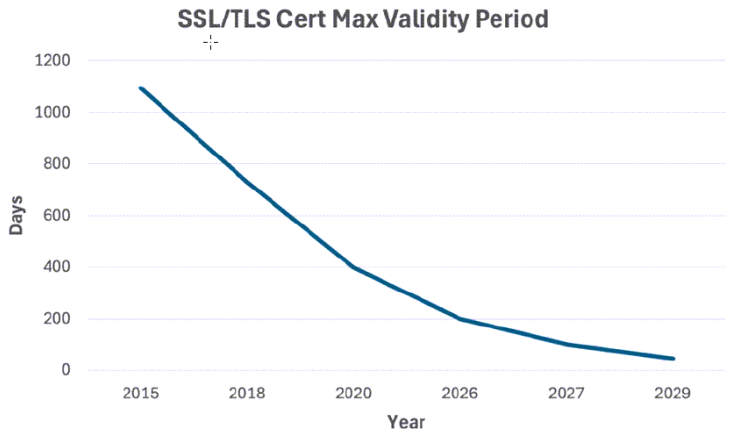
- No centralized inventory
- No automated renewal process
- No emergency playbook for compromised certificates

Outcome: Downtime, failed integrations, audit findings, and angry stakeholders.

2.5.1 Reducing the Validity Time of Digital Certificates

The validity period of digital certificates, especially Secure Sockets Layer (SSL)/Transport Layer Security (TLS) certificates, has sharply decreased over the past decade to improve security and reduce exposure to compromised certificates. While certificates could be valid for up to three years as recently as 2015, industry policies will mandate a maximum of 47 days by March 2029. This dramatic reduction is driven by security concerns and the need for more frequent revalidation, ultimately making automation of certificate management essential.

Figure 1: SSL/TLS Cert Max Validity Period



"A single expired cert can bring down six services. And unless you've rehearsed this before, it takes hours just to figure out what went wrong."
– A Security Lead

2.5.2 The Broader Impact

Certificate failures do not just affect one component, they ripple outward:

- APIs fail to authenticate
- New features are blocked
- Users lose access
- Developers are delayed
- Compliance teams escalate

2.5.3 Compliance Lens: DORA, NIS2, and PCI DSS

Regulators now explicitly expect robust cryptographic practices. Poor certificate hygiene is no longer just a technical debt; it is a compliance failure.

- **Digital Operational Resilience Act (DORA)** – Article 9 requires robust ICT risk management, including cryptographic controls and regular testing. Expired certificates are a direct resilience risk.
- **Network and Information Security Directive (NIS2)** – Emphasizes secure authentication and encryption within critical infrastructure. Certificate validation and revocation are mandatory across supply chains.
- **Payment Card Industry Data Security Standard (PCI DSS) v4.0** – Requirements 4 and 6.4.3 mandate the use of secure encryption and documented cryptographic protocols for all sensitive data transmissions, especially public-facing systems.

By structuring the certificate lifecycle, organizations reduce both operational risk and audit friction.

Mainframe certificate management is not one team's job. It requires coordination between the following groups:

- Security teams (policies, compliance)
- Systems programmers (stack config, ESM rules)
- Application owners (dependencies, rollout timing)
- Middleware and network teams (trust chains, endpoints)

When these groups collaborate, visibility improves, and emergencies are rare. When they do not, chaos ensues.

2.6 Key Takeaways for Certificate Challenges

- Certificates are critical to secure communications on the mainframe.
- Tools like AT-TLS and mTLS make trust chains more complex.
- Without proper lifecycle management, small issues become big risks.
- Compliance frameworks now demand strong cryptographic controls.
- Cross-team collaboration is the only sustainable path forward.

Chapter 3: Digital Certificates in the Mainframe World

Certificate management has become a critical part of today's mainframe landscape.

3.1 The Modernization Imperative

Banks are launching mobile apps that let customers deposit checks with their cameras. Airlines are rolling out applications that provide real-time baggage tracking and gate changes. Insurance companies are developing platforms where customers can file claims and receive instant quotes. Behind each of these modern digital experiences lies a fundamental truth; the core business logic still runs on mainframes and must be exposed through Representational State Transfer (REST) APIs for digital applications.

When a customer checks their account balance on a mobile banking app, that request travels through multiple layers of security before reaching the mainframe system that stores the account data. At each step, digital certificates verify that communication is secure, and the parties involved are who they claim to be.

3.2 Why REST APIs Changed Everything

Mainframe applications were designed for direct terminal access or batch processing. Users connected through 3270 terminals, and data flowed through well-defined, closed networks. Security was largely about controlling who could access the system and what they could do once inside.

REST APIs turned this model upside down. Organizations needed to expose their core mainframe applications (Customer Information Control System [CICS]-based claims processing systems, payment processing engines, airline reservation systems, and customer account management platforms) to modern digital channels. Now, mainframe data needs to be accessible to the following connections:

- Mobile applications running on thousands of different devices
- Web portals accessed from anywhere in the world
- Partner systems that need real-time data access
- Cloud-based analytics platforms
- Third-party applications through API marketplaces

Each of these connections represents a potential security risk. Unlike the controlled environment of terminal access, API connections come from unpredictable sources with varying security capabilities. Digital certificates became the primary mechanism for ensuring these connections remain secure.

3.3 Real-World Scenario: The Certificate Explosion

A regional bank that had managed perhaps a dozen certificates for their mainframe operations suddenly needed hundreds of certificates when they launched their digital banking platform. Each customer-facing application required its own certificate. Every API endpoint needed proper authentication. Partner integrations demanded mutual certificate verification.

This explosion in certificate requirements creates several challenges:

- **Scale management** – Organizations that previously renewed a handful of certificates annually now face managing hundreds of certificates with varying renewal cycles. While most certificates have annual validity periods, the sheer volume means that organizations typically have multiple certificates expiring each month, requiring constant attention to renewal processes and coordination across teams.
- **Lifecycle coordination** – When a mobile app certificate expires, it is not just an IT problem; it is a business problem. Customers cannot access their accounts, transactions fail, and the help desk gets flooded with calls. Coordinating certificate renewals across multiple applications requires careful planning and communication.
- **Compliance requirements** – Financial institutions must ensure their API certificates meet standards such as PCI DSS for payment processing. Healthcare organizations need certificates that comply with HIPAA requirements for protected health information and FDA guidelines for medical device connectivity. These compliance frameworks often specify minimum key lengths, approved certificate authorities, certificate lifecycle management procedures, and audit documentation requirements.

3.4 Beyond REST APIs: The Broader Certificate Landscape

While REST APIs drive much of the current certificate demand, modern mainframe environments increasingly use certificates for other critical purposes. These applications represent the broader trend toward certificate-based authentication as organizations move away from traditional username and password security models in favor of more robust cryptographic authentication methods.

The key difference is volume and management complexity. While these other use cases are important, they typically involve fewer certificates with longer lifecycles. REST APIs, by contrast, can generate certificate requirements that scale with business growth.

3.5 Digital Certificate Creation and Management

Let us create a digital certificate for authentication through public-key cryptography using an internal trusted certificate authority and add it to a user's key ring. In this case, the internal authority is provided by the Top Secret™ ESM. We can use the GENCERT command of Top Secret to create a digital certificate from an accessible public key. The public key encrypts data for safekeeping. The data can then be decrypted only through the private key. Only one party possesses the private key, which is carefully guarded. The public key is shared with the world.

NOTE: This document uses Top Secret to demonstrate certificate creation. For alternative ESMs, kindly consult their respective documentation; the general procedure remains consistent.

3.5.1 Step 1: Generate the Digital Certificate owned by the User

Use the GENCERT command to create a new digital certificate:

```
TSS GENCERT(USER01) DIGICERT(TOMSRV) LABLCERT('TOMCAT CERT') -
SUBJECTN('cn=TOMCAT-SERVER-TSS-CERT') -
ALTNAME('DOMAIN=EXAMPLE.DOMAIN.NET') NADATE(05/31/27)
```

Here are the details:

- USER01 – Accessor ID (ACID) to be associated with the generated certificate
- TOMSRV – Internal name for TSS to distinguish between multiple certificate records for the same ACID
- TOMCAT CERT – Certificate label that is needed when configuring application or AT-TLS
- cn=TOMCAT-SERVER-TSS-CERT – Subject name of the certificate that uniquely identifies the user, task or server
- DOMAIN=EXAMPLE.DOMAIN.NET – Alternative name that allows a single certificate to specify multiple types of identifiers
- NADATE – The expiration date for the certificate

3.5.2 Step 2: Create and Manage Key Rings

A key ring is a collection of digital certificates that are associated with an individual user. A user can have more than one key ring.

First, check the user's ACID for existing key rings:

```
TSS LIST(USER01) KEYRING(ALL)
```

Create a new key ring and assign it to ACID USER01:

```
TSS ADD(USER01) KEYRING(MCI1RING)
```

Add the certificate to the key ring for USER01:

```
TSS ADD(USER01) KEYRING(MCI1RING) RINGDATA(USER01,TOMSRV)
```

3.5.3 Reference Documentation

For detailed information about various flags and options available with these commands, refer to the official Broadcom® Top Secret documentation:

https://techdocs.broadcom.com/us/en/ca-mainframe-software/security/ca-top-secret-for-z-os/16-0/administrating/digital-certificates/implement-digital-certificates-to-certify-identity-for-secure-transactions.html

3.6 Managing Your Certificate Environment

These commands provide the foundation for certificate creation and key ring management in Top Secret. For comprehensive certificate administration, you can perform additional operations to list all certificates and their detailed properties, view certificate chains and trust relationships, examine which users are associated with specific certificates, and analyze key ring configurations across your environment.

3.7 The Path to Certificate Maturity

To effectively assess and improve certificate management capabilities, organizations should consider conducting a comprehensive maturity assessment. Chapter 7 of this document provides a detailed PKI Maturity Model (PKIMM) that helps organizations evaluate their current state across multiple dimensions and plan their improvement journey.

3.8 Looking Forward

The trend toward API-driven modernization shows no signs of slowing. If anything, it is accelerating as organizations discover new ways to leverage their mainframe investments through modern digital channels. The emergence of generative AI and large language models is creating additional opportunities for data integration, whether through agentic AI systems that need real-time access to business data, retrieval-augmented generation (RAG) implementations that enhance AI responses with enterprise information, or Model Context Protocol (MCP) servers that provide AI models with structured access to organizational data. Each of these innovations requires secure REST API access to mainframe systems, further increasing the importance of robust certificate management.

This means certificate management will only become more critical to business success. Organizations that treat certificate management as a strategic capability, investing in proper processes, tools, and expertise, find themselves better positioned to capitalize on modernization opportunities. Those that continue to manage certificates as an afterthought will likely find themselves constrained by operational overhead and risk exposure.

3.8.1 Building Your Foundation

Understanding the importance of certificates in modernization initiatives is just the first step. To implement effective certificate management, you need a solid grasp of the underlying concepts and technologies. In the next chapter, we explore the fundamental building blocks of digital certificates and public-key infrastructure (PKI), providing the technical foundation needed to transform certificate management from a reactive necessity into a strategic advantage.

Chapter 4: Demystifying Digital Certificates for Mainframe Security

Mainframe professionals must develop an understanding of the certificate lifecycle on the mainframe platform.

Many mainframe professionals excel at system-level security, yet digital certificates often remain a mystery, especially when used across platforms, APIs, or encrypted services. This chapter clarifies key certificate concepts by linking them to enterprise security models and offering practical, mainframe-specific guidance.

Understanding digital certificates is not just about cryptography; it is about enabling trust across your systems, improving interoperability, and meeting the expectations of modern compliance frameworks like DORA, PCI-DSS, and NIST.

4.1 Core Concepts of Digital Certificates

Mainframe environments use digital certificates to perform the following tasks:

- Authenticate users and systems
- Negotiate a key to encrypt data exchanged between services
- Protect integrity of software/firmware
- Secure API and web traffic
- Comply with security regulations
- Enable supply chain security (code signing certificates)

At the heart of this system lies the PKI, combining identity, encryption, and lifecycle management.

These principles apply not only to mainframes, but also to modern distributed environments using tools such as OpenSSL, keytool, and Public-Key Cryptography Standards PKCS#12 containers to manage and transfer certificates securely across systems.

4.2 Certificate Fundamentals Explained: CA and Trust Hierarchies

Digital certificates are issued by trusted CAs that digitally sign certificates to vouch for identity.

On z/OS, the hierarchy includes the following certificates:

- Root CAs (ultimate trust anchor)
- Intermediate CAs (delegated trust)
- End-entity certificates, known as PERSONAL certificates
- CERTAUTH certificates represent trusted root and intermediate CAs in the mainframe ESM

Trust Chain: Mainframes validate a full chain of trust from the end certificate to the root. Any missing or untrusted element leads to trust failure and service disruption.

4.3 Public and Private Key Infrastructure

Each certificate comes with a public/private key pair:

- Public key: Shared with everyone
- Private key: Kept secret and securely stored

This infrastructure allows two fundamental cryptographic actions:

- Encryption of data that only the private key holder can decrypt
- Authentication through digital signatures created with the private key, which anyone can verify using the public key

Let us break it down using a client-server model:

- The server holds the private key
- All clients have access to the server's public key
- Clients can use the model for key negotiation or distribution:
  - Encrypt messages using the public key that only the server can decrypt
  - Verify that data was sent by the server, if it is signed with the private key

However, clients cannot decrypt traffic encrypted with the public key, and clients cannot impersonate the server, because they do not have the private key.

This asymmetry is what enables secure distributed communication at Internet scale.

Private key security is paramount, if compromised, all encrypted data and authentication is at risk.

4.4 Validity Period and Renewal

Certificates expire! A typical validity period ranges from one to three years, although this validity period is rapidly shrinking. By March 2029, a typical validity period will be only 47 days.

Best practices:

- Monitor expirations regularly
- Start renewal many days in advance
- Test new certificates in non-production environments
- Validate the trust chain post-renewal

Table 1: Mini Checklist

	Is the new certificate loaded in TEST?
	Is the private key secure?
	Is the full chain trusted in PROD?

4.5 Real-World Applications in Mainframe Environments

AT-TLS encrypts traffic without modifying application code. This encryption is ideal for securing API traffic on the mainframe.

Steps to view AT-TLS configuration:

1. Locate the policy in the STC PAGENT located in your PROCLIB.
2. Look for STDENV DD, the member associated is the policy member.

A typical AT-TLS rule might look like the following example:

```
TTLSRule APISSL1~
{
   LocalAddr ALL
   RemoteAddr ALL
   LocalPortRange 8000
   Direction Inbound
   TTLSGroupActionRef APIGroup
   TTLSEnvironmentActionRef APIEnv
}
TTLSGroupAction APIGroup
{
   TTLSEnabled On
}
TTLSEnvironmentAction APIEnv
{
   TTLSKeyRingParms
   {
      KEYRING ringowner/keyringname
   }
   HandshakeRole Client
   Trace 7
}
```

The certificate and key ring information defined in the ESM is referenced in the TTLSKeyRingParms section using the KEYRING parameter.

The `ringowner` value specifies the user ID under which the key ring was created.

The `keyringname` value specifies the *case-sensitive* name of the key ring.

Using the key ring created in Step 1: Generate the Digital Certificate owned by the User as an example, the corresponding AT-TLS policy definition for KEYRING would be as follows:

```
TTLSKeyRingParms
   {
   KEYRING USER01/MCI1RING
   }
```

The key ring was created under ACID USER01, and the absence of a LABLRING specification on the `TSS ADD` command causes it to default to the KEYRING value.

A typical AT-TLS rule might look like the following example:

```
TTLSRule APISSL1~
{
   LocalAddr ALL
   RemoteAddr ALL
   LocalPortRange 8000
   Direction Inbound
   TTLSGroupActionRef APIGroup
   TTLSEnvironmentActionRef APIEnv
}
TTLSGroupAction APIGroup
{
   TTLSEnabled On
}
TTLSEnvironmentAction APIEnv
{
   TTLSKeyRingParms
   {
   KEYRING ringowner/keyringname
   }
   HandshakeRole Client
   Trace 7
}
```

What this means:

- LocalPortRange 8000: API is listening here
- Direction Inbound: Protecting incoming traffic
- TTLSEnabled ON: Encryption is active
- Trace 7: Enables debugging

TIP: Use Trace to troubleshoot broken chains or certificate mismatches.

4.6 Common Pain Points with Middleware Integration

The following security failures often arise with certificates for middleware:

- Missing from the ESM
- Not associated with the correct key ring
- Incomplete (trust chain not fully loaded)
- Expired or revoked without timely replacement

Key principle: Every component must have the correct certificate, key ring association, and ESM configuration to enable secure communication.

4.7 Real-World Scenario: Cross-System Communication Failure

Problem: RACF603I VERIFICATION FAILED, CERTIFICATE NOT TRUSTED **Impact:** Cross-system communication blocked **Resolution:** Certificate chain verification required **Cause:** Certificate chain incomplete or untrusted **Fix:** Install the full chain (root and intermediate) and associate properly in the key ring

Cross-system APIs *would not work* unless all parties trust each other's certificates.

4.8 Mutual Authentication of Client Certificates

In many integrations, especially API-based or business-to-business, the server is not the only entity presenting a certificate. The client must also present one.

This connection is called mutual TLS (mTLS):

- Both client and server validate each other's certificates
- Both sides must trust the other's issuing CA
- Ensures that both parties are known, not just one

Mainframe teams must plan client-side certificate deployment, including the following components:

- Key ring entries for client certificates
- Trust anchors for validating remote servers
- Secure private key storage (for example, hardware cryptographic modules)

4.9 ESM Configuration Examples

The ACF2™ ESM, Top Secret ESM, and IBM RACF ESM each handles certificates differently. Broadcom offers a wide range of videos and examples for ACF2, Top Secret, and IBM RACF available in our official TechDocs site. These resources provide practical guidance to help teams implement and manage digital certificates effectively within each security environment.

These videos are also available in full-screen view format. You can view the Digital Certificates playlist on the Educate YouTube channel.

NOTE: Always refer to your current ESM documentation for the most up-to-date command syntax and options. These examples are based on commonly used versions but may need adjustment for your specific environment.

4.10 Validating Security End-to-End

When deploying or renewing certificates, validate the following components:

- Certificate chain is intact and trusted
- Permissions in the ESM allow certificate usage
- AT-TLS policies are correct
- Key rings are properly linked
- Client-side certificates (if mTLS is required) are installed
- Secure traffic is tested and confirmed

4.11 Zowe: Effectively Deal with Certificates in the Client Platform

In any secure client-server communication, both ends must establish trust using digital certificates. While much attention is placed on configuring server-side certificates, the client platform must also be properly configured to trust the server's identity and, in some cases, present its own credentials. This configuration becomes especially relevant in hybrid infrastructures where mainframe services are exposed to distributed applications and web-based front ends, precisely the kind of environment Zowe enables.

4.12 Zowe: Bridging Mainframe and Modern Clients

Zowe is an open-source framework that modernizes access to IBM Z systems by exposing APIs, command-line tools, and web-based interfaces. It allows developers and DevOps teams to interact with the mainframe in a way that is more familiar to them. For these interactions to be secure, the certificates involved in TLS communication, both for the Zowe server and the clients connecting to it, must be managed properly.

Let us explore how to effectively deal with the certificates that must be present on the **client side,** using Zowe as a practical example.

1. Clarify the trust model and communication flow.

   When a user accesses the Zowe API Mediation Layer through a browser or script, the client (browser, curl, Zowe CLI, and so on) must validate the server certificate chain:
   - The Zowe server must present a valid certificate.
   - The client platform must trust the CA that issued that certificate.

   If mutual TLS is required (for example, internal integrations or regulatory compliance), the client must also present its own certificate and private key. Therefore, both trust anchors (CA roots and intermediates) and client credentials (certificate and private key) must be correctly installed and managed.

2. Client-side certificate management strategies.

   Depending on the client type, the approach to certificate management differs:
   - Browser clients (accessing Zowe through web UI) – Users must trust the CA that signed the Zowe certificate. If using self-signed or internal CAs, the root/intermediate certificates must be imported into the browser or system trust store (such as, Windows Trusted Root Certification Authorities).
   - Command-line clients (such as, Zowe CLI, curl, custom scripts) – These clients often use the system's trust store or can be directed to use a specific certificate bundle (through environment variables or CLI flags like --cacert). For mutual TLS, they may also require a --certand --key parameter.
   - Automated scripts or CI/CD pipelines – These environments should use securely stored certificates and keys, leveraging vaults, encrypted files, or secure environment variables. Avoid hard-coding or exposing secrets.

3. Secure storage of certificates and keys.

   Storing certificates and especially private keys requires caution. Best practices include:
   - Using system key stores (such as, macOS Keychain, Windows Certificate Store, Java KeyStore, or z/OS key ring).
   - Setting correct file permissions to prevent unauthorized access.
   - Store certificate private keys in Unix System Services (USS).
   - Encrypting private keys or using secure credential managers (such as, HashiCorp Vault, CyberArk, AWS Secrets Manager) when certificates are accessed programmatically.

4. Automate certificate provisioning and rotation.

   Manual certificate installation is time-consuming, error-prone, and incompatible with agile and DevSecOps practices. Automation is key to maintaining uptime and compliance:

- Automated Certificate Management Environment (ACME) Protocol (such as, Let's Encrypt) or SCEP to automatically provision and renew certificates in distributed environments.
- On the client side, scripts or management tools can be built to:
  - Download updated CA bundles.
  - Install them into the proper location.
  - Restart affected services if needed.
- Zowe CLI and API consumers can include logic to periodically validate and reload certificates.
- If your client accesses Zowe using mutual TLS, ensure that client certificates are rotated regularly and that there is a mechanism to alert users or systems before expiration.

5. Test and monitor trust from the client perspective.

   Validation does not end at deployment. Frequent testing of the client-server certificate chain is necessary to avoid surprises. Some tools to support testing and monitoring:

   - `openssl s_client -connect zowe-host:port`

     To see the full chain and identify trust issues.
   - `curl -v` or `zowe auth` login

   To simulate real usage and confirm trust.

   - Monitoring tools or scripts that test connectivity and certificate validity (including expiration warnings).

6. Revocation checks and compliance.

   In regulated environments or for high-security clients, it is important to validate that server certificates have not been revoked:

   - CRL (Certificate Revocation Lists) and OCSP (Online Certificate Status Protocol) are standard mechanisms.
   - Ensure that your client environment supports and enforces revocation checks or implements additional controls.

4.13 Real-World Scenario: Zowe in a Regulated Financial Environment

Problem: Modernizing mainframe access through REST APIs while meeting strict DORA and internal financial security policies.

Impact: Risk of unauthorized access, compliance failures, or critical CI/CD pipeline disruptions.

Resolution: Implementation of a centralized internal trust model and automated lifecycle management.

Cause: Requirement for secure, auditable communication between Zowe servers and a diverse fleet of client systems and Jenkins runners.

Fix:

- Sign Zowe server certificates with a trusted internal CA.
- Distribute the internal CA root to all client local trust stores.
- Automate client certificate rotation every 60 days using ACME and GitOps.
- Deploy a central monitoring dashboard fed by nightly curl-based connectivity checks.

Zero certificate-related outages were achieved in over a year, providing full traceability for regulatory audits.

Effectively managing the certificates needed on the client platform, especially in ecosystems like Zowe where client diversity is the norm, requires a thoughtful balance of automation, secure storage, and proactive testing. When done well, it strengthens the security posture, supports compliance, and ensures a seamless experience across hybrid architectures.

4.14 Key Takeaways for Zowe Environments

By understanding the anatomy of digital certificates and applying best practices, such as trust validation, lifecycle management, client certificate handling, and middleware integration, mainframe teams can accomplish the following goals:

- Avoid reactive firefighting
- Build resilient, compliant systems
- Enable secure communication across platforms

In the next chapter, we explore how to design a scalable certificate management framework tailored to your environment and aligned with compliance expectations.

Chapter 5: The Building Blocks of Trust

An overview of digital certificate structure, trust with certificate chains, and managing certificate validation in mainframe environments.

Understanding digital certificates is one thing. Grasping how they enable trust across critical workloads in the mainframe is something else entirely and essential for building secure, compliant, and modern enterprise systems.

This chapter explores the anatomy of a digital certificate, how trust is established through certificate chains, and the practical steps required to manage and validate certificate chains in mainframe environments. We walk through real-world deployment workflows, troubleshooting scenarios, and offer actionable best practices.

Whether you are a mainframe security administrator, infrastructure engineer, or application developer working in highly regulated environments, this chapter will help bridge the gap between cryptographic theory and operational trust.

5.1 Trust in the Mainframe Context

Every day, mainframes process vast amounts of sensitive transactions, from financial payments and identity authentications to secure API calls and back-end system integrations. At the core of each secure interaction lies an invisible but essential foundation; a chain of trust based on digital certificates.

In these trust chains, each certificate is validated based on the following criteria:

- Who issued the certificate (the CA)
- The certificate's validity period
- Whether the chain leading back to a trusted root authority is complete and intact

If any link in the chain is missing, expired, or untrusted, trust is broken and can lead to the following issues:

- Application failures
- Security incidents and alerts
- Regulatory non-compliance

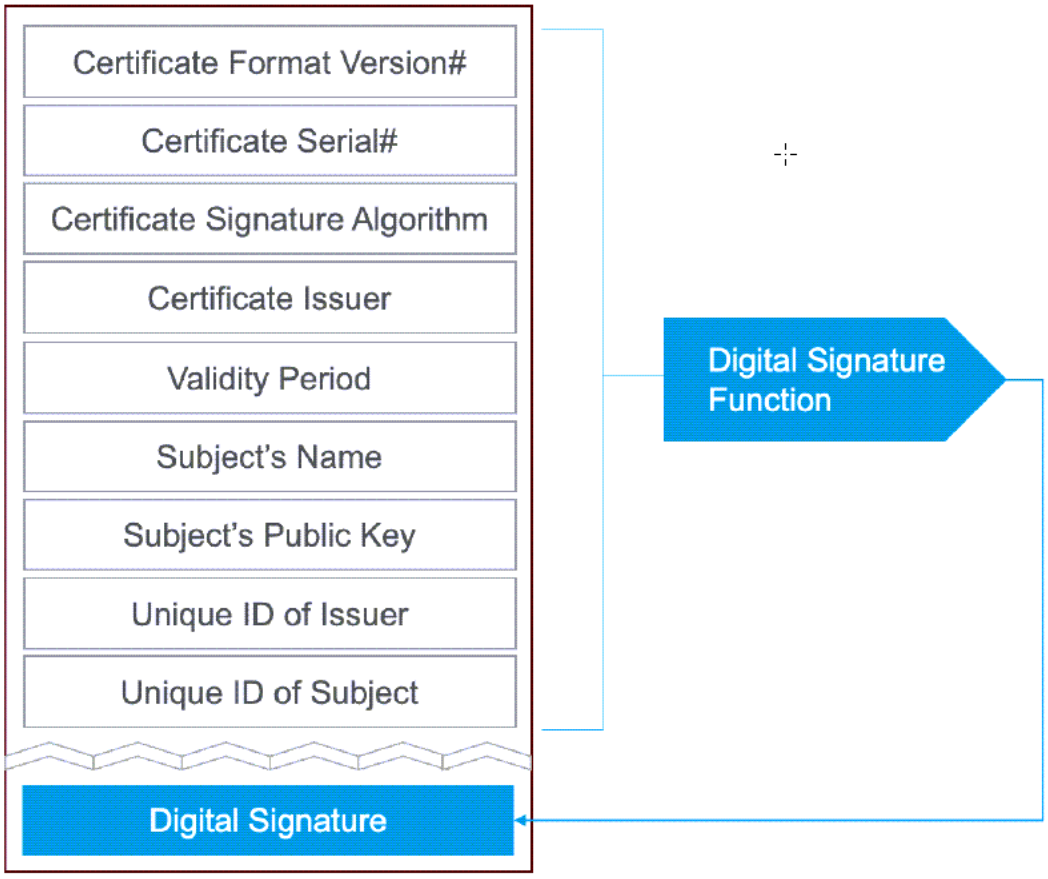
In a highly regulated world where frameworks like DORA, NIST, and PCI DSS demand provable trust and encryption hygiene, understanding how these certificate relationships work is foundational and no longer optional.

5.2 Anatomy of a Digital Certificate

Let us examine a real production certificate:

```
/* View detailed certificate information */
For Top Secret: TSS LIST(acid) DIGICERT(digicert name)
For ACF2: CHKCERT logonid Label(label)
For RACF: RACDCERT ID(Certowner)LIST(LABEL('your-cert-label'))
```

Figure 2: Digital Certificate Elements



Each digital certificate follows the X.509 standard, and carries key elements that define its identity, cryptographic function, and trust relationships.

Let us break down a real-world certificate and explore its components:

- Version number

  Most digital certificates in use today conform to X.509 version 3. This version supports important extensions like subject alternative names, CRL distribution points, and key usage.

  ```
  X.509 Version: v3
  ```

- Serial number

  A globally unique identifier for the certificate. Think of it like a fingerprint or an employee ID. Example:

  ```
  Serial Number: 2c:80:0e:cf:...
  ```

- Signature algorithm

  Indicates the hashing and encryption algorithms used to sign the certificate. This identification is crucial for interoperability with middleware and cryptographic toolkits.

  ```
  Signature Algorithm: sha256WithRSAEncryption
  ```

- Issuer and subject information
  - Issuer – The CA that issues the certificate.
  - Subject – The entity (for example, a CICS region) that owns the certificate.

```
Issuer: CN=Enterprise-Root-CA, O=YourCompany, C=US
Subject: CN=PROD-CICS-Region1, OU=Mainframe, O=YourCompany
```

- Validity period

```
Not Before: 2024-01-01 00:00:00
Not After: 2025-01-01 23:59:59
```

5.2.1 The Trust Chain in Action

A certificate chain is like a path of trust. It starts with the application's certificate and leads up to a widely trusted root authority.

5.2.2 Components of a Trust Chain

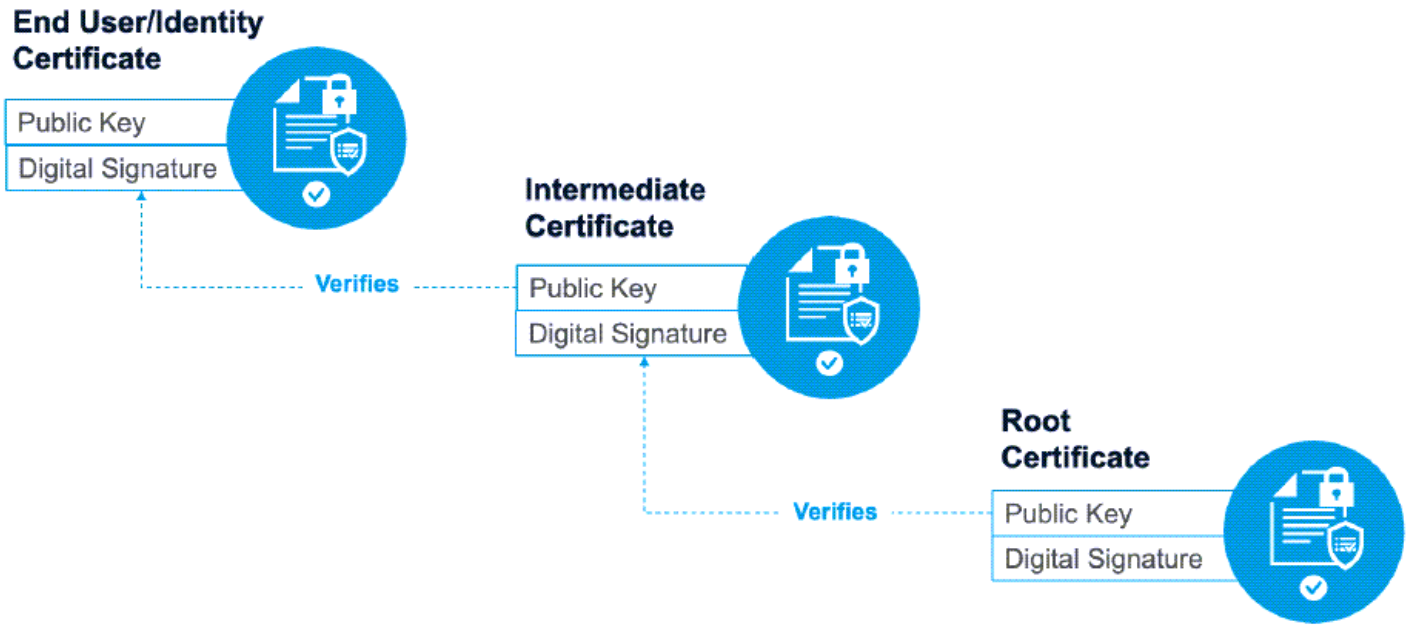
- End-entity certificate – Used by your application (such as, DB2, CICS, z/OSMF)
- Intermediate certificates – Signed by the root and used to sign the end-entity certificate
- Root certificate – The ultimate trust anchor that is inherently trusted by your system

For trust to be established, the system has the following requirements:

- View the entire chain
- Validate that each certificate is current and not revoked
- Confirm that each signature aligns with the issuing certificate's public key

The following figure represents a typical certificate chain.

Figure 3: Certificate Chain



5.3 Viewing and Verifying Certificates by ESM

Refer to the following table for various commands related to certificate management within mainframe.

Task	RACF	TSS	ACF2
View certificate details	RACDCERT userid LIST(LABEL('cert label'))	TSS LIST(acid) DIGICERT(label)	CHKCERT logonid.suffix
View certificate chain	RACDCERT userid LISTCHAIN(LABEL ('cert label'))	TSS LIST(acid) DIGICERT(label) CHAIN	CHKCERT logonid.suffix CHAIN
View key rings	RACDCERT ID(userid) LISTRING(ringname)	TSS LIST(acid) KEYRING(ringname)	SET P(USER) DIV(KEYRING) LIST LIKE(logonid-)
List all certificates with expiration	IBM Health Checker or SYSVIEW® Essentials	SAFCRRPT utility or SYSVIEW Essentials	SAFCRRPT utility or SYSVIEW Essentials

5.4 Understanding Key Rings

Before diving into how to view or manage key rings, it is important to understand what they represent in the mainframe security model. A key ring is a security-managed container used to group together one or more digital certificates, typically including a server certificate, trusted root certificates, and any intermediate certificates.

From an operational perspective, a key ring allows an application to perform a single access check through the ESM (ACF2, Top Secret, RACF) and gain access to all the certificates it needs in one go. This design greatly simplifies security administration and enables flexible reuse of trust configurations across different workloads.

Key rings are associated with user IDs or applications, and they are referenced by name, meaning naming conventions and ownership become critical to avoid runtime errors. Ensuring that the correct key ring is available, correctly named (case-sensitive), and accessible to the right application is a foundational part of secure certificate deployment on the mainframe.

5.5 Certificate Deployment: Real-World Flow

Imagine that you are securing a new application or API, such as a CICS web service or an encrypted FTP connection. The certificate deployment process could follow these steps:

- Generate or import the end-entity certificate: Use OpenSSL, RACDCERT, or vendor-issued certificates.
- Install intermediate and root certificates in the CERTAUTH section of your ESM.
- Assign certificates to the appropriate key ring: Ensure proper pairing and unique, consistent labels.
- Configure the application to reference the key ring, such as in AT-TLS policies or CICS definitions.
- Grant required access to key rings and certificates using ESM access control rules.

TIP: Be meticulous with names. Certificate label, key ring name, and owner must match exactly, including case sensitivity, or your application might silently fail.

5.6 Troubleshooting Common Issues

Issue	Likely Cause	Resolution
Certificate not trusted	Missing or misconfigured CA/Intermediate cert	Verify chain completeness and trust anchors in CERTAUTH
Private key not found	Certificate created without storing private key in ESM	Regenerate or ensure private key is available and associated properly
Key ring not found or incorrect	Case-sensitive name mismatch or incorrect owner	Confirm key ring owner and names match application configuration
Certificate expired	Renewal overlooked or update not propagated	Implement proactive expiration monitoring and coordinated renewals

5.7 Best Practices

- Enforce consistent naming conventions for certificates and key rings (such as, CN=APP-TLS-2025).
- Audit trust chains regularly to avoid expired or missing links.
- Automate expiration alerts using IBM Health Checker, SYSVIEW, or custom scripts.
- Include certificates in change management so renewals and reconfigurations are tracked and peer reviewed.
- Test everything in staging first to ensure real-world compatibility before production rollouts.
- Maintain a certificate inventory with metadata like expiration dates, owners, key ring names, and use cases.
- Apply the principle of least privilege, only allow key ring access to required applications or services.
- Document client certificate usage and authentication mechanisms, especially when implementing mutual TLS.

5.8 Wildcard Certificates: Operational Convenience at a Security Cost

We have seen instances of Wildcard certificates usage recently, while it is convenient to configure, maintain and use them, they can lead to some significant Security issues.

5.8.1 What are Wildcard Certificates

Wildcard certificates are a type of TLS/SSL certificate that use an asterisk (*) in the common name (CN) or subject alternative name (SAN) to cover all subdomains of a given domain. For example, use a certificate issued to *.example.com to secure the following domains:

- mail.example.com
- login.example.com
- api.example.com
- Any other first-level subdomain under example.com

This mechanism offers significant operational convenience, especially for organizations managing numerous subdomains or dynamically provisioned services.

5.8.2 Why We Do Not Use Wildcard Certificates

While wildcard certificates can simplify certificate management, they introduce critical security risks that often outweigh their benefits in regulated or security-sensitive environments:

1. Single point of compromise

   A single leaked or compromised private key can be used to impersonate all subdomains covered by the wildcard, potentially leading to large-scale man-in-the-middle (MITM) attacks or unauthorized access.

2. Inconsistent key management

   These certificates are often deployed across multiple systems or teams, increasing the risk of poor key hygiene, including insecure storage, redundant copies, and lax access controls.

3. Reduced visibility and accountability

   Monitoring and auditing wildcard certificate usage becomes more complex, which impairs incident response and forensic investigations.

4. Conflict with least privilege principles

   Wildcard certificates often violate the principle of segmentation and least privilege, as multiple unrelated systems may share the same certificate and key material.

5.9 Regulatory Guidance and Security Standards

- **DORA (Digital Operational Resilience Act – EU)** – DORA mandates that financial entities maintain operational resilience through tight control of digital assets, including cryptographic material. Though it does not mention wildcard certificates explicitly, its requirements imply that any practice increasing systemic exposure (like sharing a wildcard certificate across services) should be avoided.

  DORA-aligned practice: Use individual certificates per system or service. Apply strict access controls, regular rotation, and lifecycle monitoring.

- **PCI DSS v4.0 (Payment Card Industry Data Security Standard)** – PCI DSS includes specific guidance for TLS certificate use and explicitly cautions against wildcard certificates:

  “Use of wildcard certificates should be carefully evaluated and controlled. Their compromise may allow attackers to impersonate multiple hosts.”
  –PCI DSS v4.0, FAQs

  PCI also requires:

  - Strong encryption in transit (Req. 4.2.1)
  - Secure key storage and management (Req. 3.6)
  - Minimization of shared resources (Req. 2.2.5)

  PCI-aligned practice: Avoid wildcard certificates unless strictly limited to a single administrative domain, with tight controls on private key access.

- **NIS2 Directive (EU Network and Information Systems Directive)** – NIS2 expands the cybersecurity obligations for essential and important entities across sectors. It emphasizes:
  - Segmentation of networks and systems
  - Containment of incidents
  - Reduction of attack surfaces

  A wildcard certificate, by nature, increases the attack surface and the impact radius of a single breach, making it a poor fit for NIS2-compliant architectures.

  NIS2-aligned practice: Issue service-specific certificates and isolate key usage to contain potential compromise.

5.10 Trust and Compliance

Maintaining an auditable trust infrastructure supports key requirements of modern regulations:

- DORA Article 9 – ICT risk management, including cryptographic trust and secure authentication
- NIS2 – Ensures secure communications in supply chains
- PCI DSS – Demands the use of valid digital certificates and secure key management for cardholder environments

5.11 Wildcard Certificates

Wildcard certificates may offer convenience, but in high-assurance environments, this ease of use comes at the cost of increased risk and regulatory misalignment. Instead, use the following processes:

- Use individual certificates per service or subdomain
- Store private keys in Hardware Security Modules (HSMs) or secure vaults
- Apply strict access controls and auditing
- Automate certificate issuance and renewal with short validity periods (such as, ACME)

5.12 Key Takeaways for Certificate Security

- Building proper trust chains and validating them regularly is not just good practice it is a regulatory imperative.
- Private Keys should be stored in HSMs or secure vaults, and strict access controls and auditing should be put in place.
- Trust on the mainframe goes beyond simple encryption. It is built on verifiable certificate chains, accurate key ring configurations, and a disciplined management process that touches security, operations, and compliance.
- By understanding certificate anatomy, mastering the tools of your ESM, and applying structured deployment and auditing practices, mainframe teams can ensure that their systems remain not only secure, but trusted, stable, and ready for the challenges of modern digital business
- Wildcard certificates should not be used in environments where segmentation, accountability, and minimal impact are critical, especially when operating under frameworks like DORA, PCI DSS, and NIS2. Strong certificate hygiene is not just good practice, it is an operational resilience requirement.

In the next chapter, we will see how these trust foundations are applied in real-world scenarios involving business continuity, regulatory audits, and modernization projects.

Chapter 6: The Definitive Guide to Certificate Management

This chapter provides a practical guide to understanding, implementing, and managing certificates on mainframe.

Managing digital certificates on the mainframe is a multifaceted discipline that combines planning, coordination, execution, monitoring, and readiness to respond to unexpected events. A well-structured certificate lifecycle strategy strengthens security, supports regulatory compliance, and maintains the integrity of digital communications across enterprise systems.

In mainframe environments, where uptime and trust are paramount, certificate lifecycle hygiene must be rigorous and embedded in operational routines. This chapter walks through all phases of the lifecycle and provides technical guidance, compliance alignment, troubleshooting tips, and best practices.

A visual summary of the certificate lifecycle is shown in the following figure.

Figure 4: Certificate Lifecycle

6.1 Planning Phase

In mainframe environments, careful planning is essential before implementing certificates. Security specialists with years of experience know that having a written plan is a critical first step for ensuring that certificate deployment aligns with both technical and business requirements.

Cross-team communication is vital during this stage. Security teams must align with systems programming teams to establish clear responsibilities and repeatable processes. The following planning activities address critical questions related to scope and responsibility.

6.1.1 Requirement Gathering

Start by identifying your organization's specific needs:

- Inventory systems and applications requiring secure communication (such as, CICS, FTP, HTTP servers).
- Determine certificate types:
  - SSL/TLS for encrypted sessions
  - Code-signing for trusted binaries
  - Client authentication for identity validation
- Decide whether to use an external CA or an internal PKI service.
- Decide where certificates will be generated (on or off mainframe).
- Align all efforts with compliance standards such as PCI DSS, NIST, HIPAA, or DORA.

Table 2: Sample Requirement Checklist

	Application inventory complete
	Certificate types identified
	CA strategy determined
	Generation location decided
	Compliance requirements documented

6.2 Security Policy Alignment

Ensure certificates comply with existing security frameworks:

- Validate encryption strength and cryptographic algorithms.
- Define renewal and revocation strategies aligned with enterprise policies.
- Implement naming and labeling conventions across the organization.

6.3 Resource Allocation

Clear accountability prevents chaos. Mapping roles and responsibilities is important to ensure smoother execution.

Use the following guidelines to map responsibilities across teams:

Task	Primary Team	Secondary Team
Policy development	Security	Compliance
Certificate generation	Security	System programming
ESM configuration	Security	System programming
Application configuration	System programming	Security
Monitoring	Operations	Security
Renewal processing	Security	Operations
Emergency response	Security	All teams

Also, plan for technical resources (Integrated Cryptographic Service Facility [ICSF], backups, audit logs, and so on) and engage early with stakeholders (PKI teams, application owners, ESM admins).

6.4 Certificate Creation and Issuance

Once planning is complete, the certificate generation phase begins.

6.4.1 Certificate Generation

Certificates can be created in multiple ways:

- On the mainframe: using OpenSSL, gskkyman, or directly through the ESM.
- Externally: then imported through secure means.

NOTE: ESMs like RACF, ACF2, and Top Secret use X.509 standard, which requires the keypair to remain intact. *Do not delete original keys after CSR generation*.

6.4.2 ESM Certificate Generation Examples

- **Top Secret**

```
TSS GENCERT(USERA) DIGICERT(APICERT) LABLCERT('APIService') -
SUBJECTN('CN="API Service" OU="MyCo" C="US"')
```

- **ACF2**

```
GENCERT USERA.APICERT SUBJ(CN='API Service' OU='MyCo' C=US) -
LABEL(APIService)
```

- **RACF**

```
RACDCERT GENCERT ID(USERA) WITHLABEL('APIService') +
SUBJECTSDN(CN('API Service') OU('MyCo') C('US'))
```

6.5 Certification Signing Request Generation

For certificates requiring external signing, a certification signing request (CSR) must be generated; after sending the CSR to the third-party CA, you will receive a signed certificate with the signing chain.

Consider the following key aspects about CSRs:

- A CSR contains the public key and associated metadata to request a signed certificate from a CA.
- CSRs do not contain the private key. Never delete your original certificate after generating a CSR
- Use the same tool that generated the certificate to create the CSR.

Use the following commands to generate a CSR within mainframe:

- **Top Secret**

```
TSS GENREQ(USERA) DIGICERT(APICERT) DCDSN('USERA.CERT.UNSIGNED')
```

- **ACF2**

```
GENREQ USERA.APICERT DSN('USERA.CERT.UNSIGNED')
```

- **RACF**

```
RACDCERT ID(USERA) GENREQ(LABEL('APIService'))
DSN('USERA.CERT.UNSIGNED')
```

6.6 Validation Process

After receiving the signed certificate, verify that it contains the expected information (expiration dates, signing certificates). Once validated, the next phase starts which is installation and configuration.

6.7 Installation and Configuration

Certificate installation and configuration on mainframes primarily use key rings and your ESM, rather than Unix or Linux keystores. The following procedure is a complete step-by-step guide for the full installation and configuration process:

1. Add the certificate and signing chain to the ESM database.
   – Add the signed certificate and any certificates in the signing chain.
   – If generated outside the ESM, repackage the signed certificate with the private key.
   – If generated in the ESM, add the newly signed certificate over the original.
2. Create a key ring.
   – Ensure that the key ring owner matches the user ID of the application or server.
   – The key ring serves as a collection point for certificates needed by the application.
3. Connect required certificates to the key ring.
   – Connect all certificates in the signing chain.
   – Specify CERTAUTH for signing certificates.
   – Specify PERSONAL for end certificates.
4. Permit access to key rings and certificate private keys. When a z/OS task accesses a key ring through the Unix system services (USS) R_datalib call, several resource checks occur:
   a. Resource check 1: Ring-specific profile check (RDATALIB class).
      - Allows a user ID access to a specific key ring.
      - Resource name format: ringowner.ringname.LST.
      - READ access required.
   b. Resource check 2: Global profile check (FACILITY class).
      - This check will only occur if the resource for check 1 does not exist.
      - Resource name: IRR.DIGTCERT.LISTRING.
      - READ access for the key ring owned by task's user ID.
      - UPDATE access for the key ring owned by another user ID.
   c. Resource check 3: Private key access only, one of these conditions must be met:
      - Task user ID owns the certificate.
      - Task user ID has UPDATE access to ringowner.ringname.LST in RDATALIB.
      - The certificate is owned by SITECERT/CERTSITE/SITE and task has DELETE/CONTROL access to IRR.DIGTCERT.GENCERT in FACILITY.
5. Configure the server or application to use the ESM key ring.
   – Application-specific configuration.
   – Requires a minimum the case-sensitive key ring name.
   – May require key ring owner and certificate label.

6.7.1 Installation and Configuration: Verification Process

Use the following checklist to verify if the installation and configuration has been successfully completed.

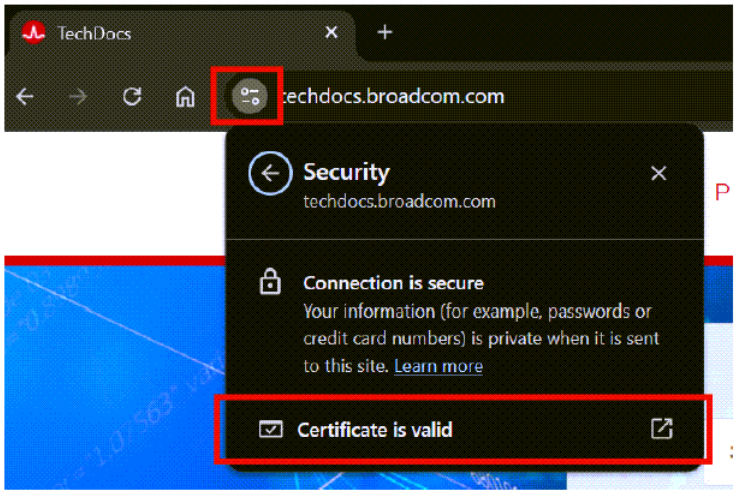
Table 3: Install/Config Verification Checklist

	Signing chain is complete
	Certificates are trusted
	Certificates are not expired
	Key ring and certificate ownership is correct
	Proper permissions granted in FACILITY/RDATALIB
	Entire signing chain exists on key ring
	Key ring DEFAULT specifies correct personal certificate
	Personal certificate label is correct
	Server domain name/IP matches certificate's SAN

6.7.2 Testing Protocols

Test the functionality and security of your certificate deployment. This test might be as simple as accessing a web server from a browser and inspecting certificate information.

Figure 5: Example of a Deployment Test



Once the installation and configuration are complete, the next phase will be monitoring and renewal.

6.8 Certificate Monitoring, Expiration, and Renewal

All ESMs provide methods to track certificate expiration:

- ACF2 and Top Secret: SAFCRRPT report (JCL in product library)
- RACF: IBM Health Checker RACF_CERTIFICATE_EXPIRATION check
- **Third-party tools like** SYSVIEW Essentials (ESM agnostic, real-time reporting)

Monitoring certificates consistently prevents unexpected expirations that can cause service disruptions.

6.9 Renewal Process

Certificate renewal is inevitable and requires proper management to prevent security gaps.

6.9.1 Advance Planning

Experienced security specialists emphasize planning certificate renewals well in advance. Waiting until the last minute increases the risk of service disruption due to the following events:

- Internal coordination requirements
- Third-party CA processing time
- Application testing needs

6.9.2 Renewal Procedures

The renewal process varies by certificate generation method. For ESM-managed certificates with third-party signing, perform the following steps:

1. Create a CSR for the expiring certificate.
2. Send the CSR to the external CA.
3. Receive the signed certificate.
4. Verify certificate information.
5. Insert the signed certificate into the ESM database.
6. Add new signing certificates to the ESM and key rings if the chain has changed.
7. Verify the expiration date has updated.

This procedure maintains the same public/private keypair. For new keypairs, use REKEY at the beginning and ROLLOVER at the end of the process.

Application and ESM refresh commands may be needed to activate the new certificate.

6.10 Emergency Procedures

Even with careful management, certificate emergencies can occur. Having established protocols for these situations is essential.

6.10.1 Revocation Process

When certificates are compromised or invalid, the following events occur:

- CAs add the certificates to CRLs or OCSP.
- Use the System SSL Certificate Management Services API for validation.
- Inform stakeholders and re-secure communications with new certificates.

6.10.2 Incident Response

Establish a clear incident response plan that includes the following items:

- Initial investigation procedures
- Damage mitigation steps
- Certificate replacement process
- Communication protocols

6.10.3 Recovery Protocols

Effective recovery requires the following tasks:

- Secure backups of ESM databases, ICSF datasets, and certificate data.
- Clear procedures for quick certificate re-issuance.
- Perform post-recovery monitoring and analysis.

6.11 Best Practices for Mainframe Environments

Automated monitoring:

- Real-time certificate status tracking
- Automated alerting system
- Proactive renewal notifications

Inventory management:

- Centralized certificate repository
- Detailed tracking system
- Relationship mapping

Compliance tracking:

- Audit trail maintenance
- Policy enforcement
- Regular compliance checks

6.12 Troubleshooting and Problem-Solving

Even with best practices, certificate issues can occur. Here are solutions for the top **three** problems encountered in mainframe environments:

6.12.1 Private Key Not Found

This error typically occurs for three reasons:

- The private key does not exist in ESM.
  - Happens when certificates are deleted while awaiting CA signature.
  - CSRs and returned signed certificates do not include private keys.

  Solution: Check the ESM database for private key information.
- There are missing permissions.
  - Review the Installation and Configuration requirements for step 3.
  - Verify that the task has proper access to certificate resources

- There is an incorrect key ring connection.
  - Certificate connected with CERTAUTH instead of PERSONAL usage.

  Solution: Check the key ring listing in ESM.

6.12.2 Key Ring Not Found

There are three common causes for this error:

- Incorrect case-sensitive key ring name in configuration
- Incorrect key ring owner specified
- Application restrictions on key ring ownership

Solution: Compare ESM key ring listing with application configuration. Create a new key ring with proper ownership if needed.

6.12.3 Certificate Insertion Issues

ESMs have specific certificate format requirements:

- Must use X.509 standard
- Can be individual certificates, PKCS#7, or PKCS#12 packages
- Can be in binary or B64 format

Troubleshooting steps:

1. Verify the certificate file was not truncated during transfer
2. Test certificate import on another platform
3. Ensure password case sensitivity is maintained

6.13 First-Line Documentation for Troubleshooting

When facing certificate issues, gather these five critical pieces of information:

1. Server and client logs showing certificate errors
2. ESM listing of the key ring being used
3. Certificate chain verification (CHKCERT CHAIN or LISTCHAIN)
4. Access verification for key ring and private key
5. R_datalib call trace to verify proper returns

If the ESM configuration is correct, proceed to deeper troubleshooting with system SSL traces.

6.14 Key Takeaways

- Effective certificate management on the mainframe is not just technical, it is strategic.
- By embedding planning, monitoring, and emergency response into your operations, you ensure trust, uptime, and compliance.
- As threats evolve and regulatory requirements grow (like DORA, NIS2, PCI DSS), robust lifecycle management of digital certificates becomes not just recommended, but mandatory.

Chapter 7: Common Real-World Scenario: Paradigm Shift

Yet another quarterly mainframe security planning session draws to a close, the conversation takes an unexpectedly strategic turn. Moments earlier, the team had received a high-level briefing on emerging threats from quantum computing. Meanwhile, system programmers were deep into architectural planning to bring Zero Trust principles to life. For once, everyone at the table shared a common concern, “Will our certificate infrastructure be ready for what comes next?”

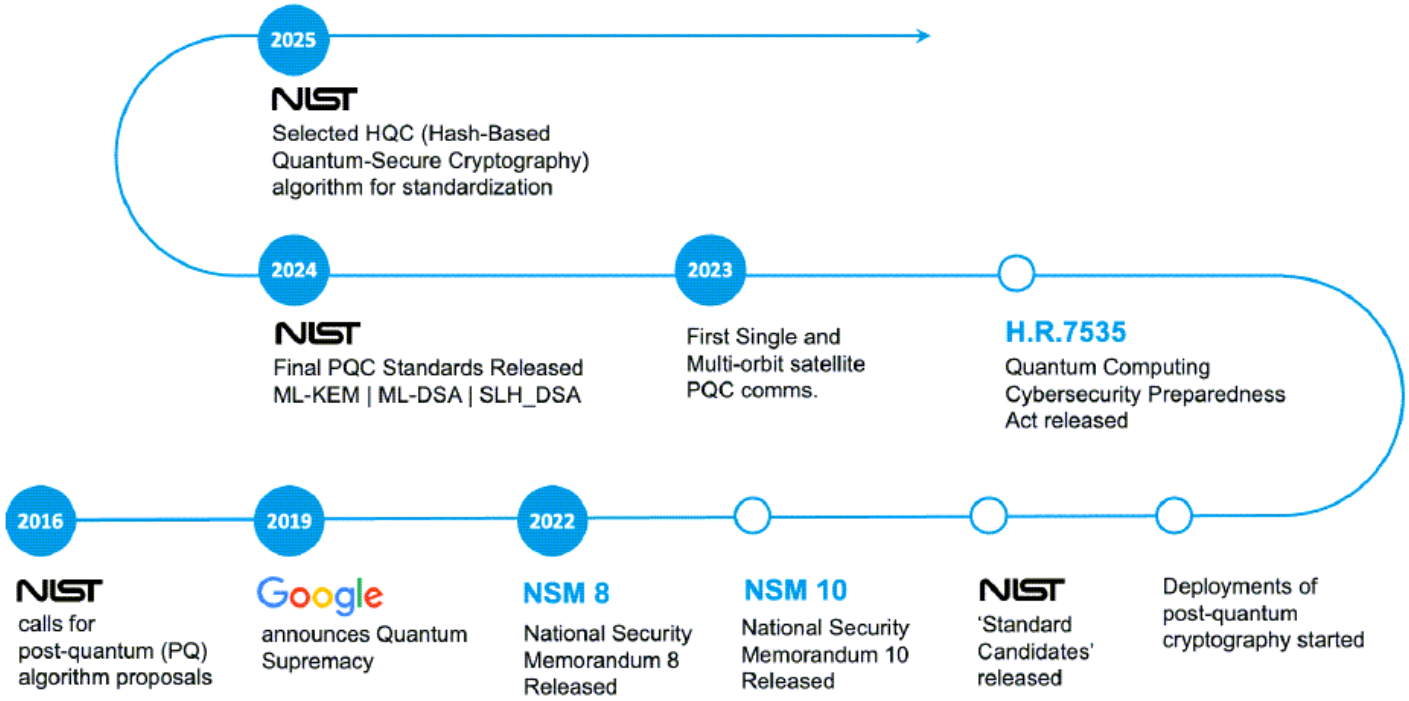
A veteran security engineer quips, “Remember when our biggest worry was just keeping certificates from expiring?” Nervous chuckles follow; everyone knows that era is long gone. Today, certificate management is not just operational. It is foundational to long-term resilience, security, and compliance.

Quantum computing presents one of the most disruptive threats to cryptography in decades. According to multiple national and industry assessments, quantum computers may render today’s asymmetric algorithms (like RSA and ECDSA) obsolete. This obsolescence might potentially occur within the lifecycle of systems being designed today.

This poses an acute risk to mainframe environments, where cryptographic protections underpin millions of transactions daily. Unlike minor updates or patches, migrating to post-quantum cryptography (PQC) is a necessary advancement.

NIST’s standardized PQC algorithms, such as ML-KEM (CRYSTALS-Kyber algorithm) and ML-DSA (CRYSTALS-Dilithium algorithm) have already been standardized by NIST and are leading the way in quantum-resilient cryptographic standards. However, organizations cannot just change overnight. Transitioning requires careful planning and multi-year roadmaps.

Figure 6: Transition of PQC Algorithms



7.1 Actionable Steps toward PQC Readiness

Digital certificates a no longer just encrypt; they authenticate, segment, and trigger policy decisions. There are steps you can take to transform your mainframe environment.

7.1.1 Comprehensive Cryptographic Inventory

Work with your ESM administrators to identify all deployed certificates, their key lengths, and algorithms in use. Track which systems are most vulnerable to quantum threats (PCI DSS 4.0 requirement).

7.1.2 Risk Prioritization Framework

Use a tiered matrix to determine which systems require early PQC transition:

- Financial transaction systems
- Personally Identifiable Information (PII) and regulated data stores
- Long-term archives with retention >10 years
- External APIs and partner gateways

7.1.3 Test PQC Algorithms in Controlled Environments

Begin experimenting with NIST-standardized PQC algorithms. Monitor computational performance and compatibility impacts.

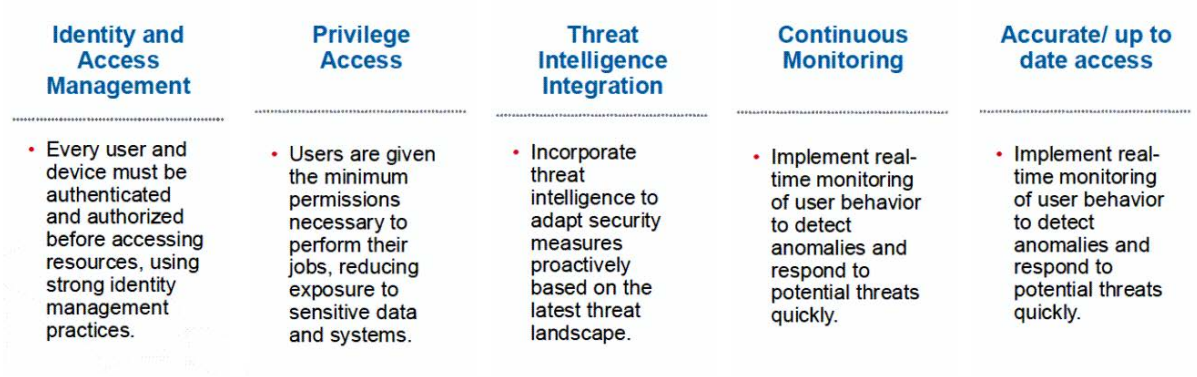
7.1.4 Design Hybrid Cryptographic Architectures

Where possible, introduce systems that support both traditional and post-quantum algorithms during the transition phase. Ensure that legacy interoperability is addressed without compromising forward security.

7.1.5 Zero Trust: Identity as the New Perimeter

The conventional mainframe model of a secured perimeter with trusted internal zones is fading. In its place, Zero Trust Architecture (ZTA) enforces the principle of “never trust, always verify”. Every identity, API call, session, and transaction must be validated in real time.

Figure 7: Zero Trust Principles



One systems programmer says, “The biggest challenge isn't the math, it’s the messy web of legacy integrations we’ve built over the last 30 years”.

Digital certificates are central to this transformation. They no longer just encrypt; they authenticate, segment, and trigger policy decisions.

7.2 Key Zero Trust Certificate Practices

- Real-time validation

  Periodic revocation checks are not enough. Mainframes should leverage real-time OCSP, CRL updates, and ESM policy enforcement for every certificate-based interaction.
- Micro-segmentation

  Divide large monolithic logical partitions (LPARs) and workloads into discrete zones, each with its own certificate trust boundaries and validation requirements.
- Dynamic access policies

  Go beyond certificate presence. Incorporate contextual parameters: device type, user behavior, access time, geolocation, and authentication method (for example, MFA-backed certificates).
- Continuous certificate monitoring

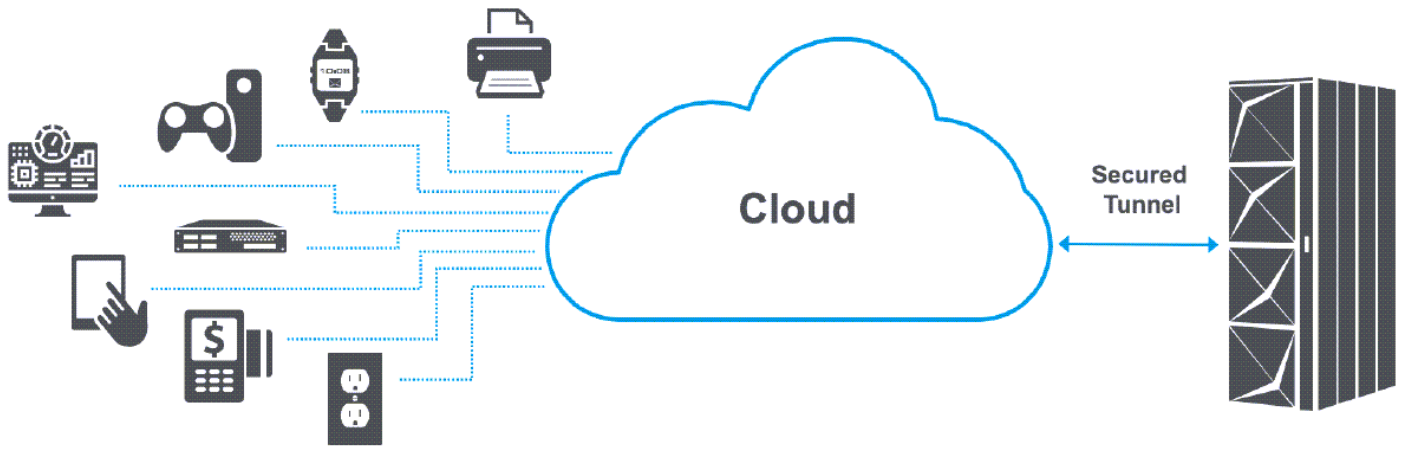
  Validate certificate integrity not only at session initiation but throughout the session. Compromised or revoked certificates must trigger session termination and alerts.

A Security Architect emphasizes, "The mainframe is no longer an isolated island. It's a highly connected node in a zero-trust mesh, and we must treat it as such".

7.2.1 The IoT Revolution Meets the Mainframe

Perhaps the most dramatic change on the horizon is the explosive growth of connected devices sending data to mainframe systems. From medical devices to industrial sensors, organizations are connecting thousands of IoT endpoints to their mainframe environments.

Figure 8: Connected Devices are Sending Data to Mainframe Systems



This growth creates unprecedented certificate management challenges:

- **Scale:** Managing certificates for thousands or millions of devices
- **Diversity:** Supporting various device types with different capabilities
- **Lifecycle:** Automating certificate provisioning and renewal for devices with long operational lives
- **Compliance:** Maintaining regulatory requirements across distributed environments

Leading organizations are addressing these challenges through:

- **Automated certificate provisioning:** Develop automated systems that can provision certificates for IoT devices at scale. Work with your ESM teams to create standardized certificate templates and automated enrollment processes that can handle thousands of device registrations without manual intervention. Consider implementing certificate management platforms that integrate with your ESM to provide automated lifecycle management.
- **Certificate hierarchy design:** Design specialized certificate hierarchies specifically for IoT deployments. Create dedicated Certificate Authorities for different device types (industrial sensors, building systems, mobile endpoints) to enable granular control and simplified management. This hierarchical approach also facilitates easier revocation and renewal processes for specific device categories.

Figure 9: Example Certificate Hierarchy

7.2.2 Artificial Intelligence and Machine Learning: The Game Changers

Artificial intelligence and machine learning are transforming certificate management from reactive to predictive.

Organizations at the forefront of this trend are deploying systems with the following capabilities:

- Predict potential certificate failures before they happen
- Identify unusual access patterns that might indicate compromise
- Automate routine certificate operations
- Learn from past incidents to prevent future issues

AI also enhances certificate capabilities:

- Anomaly detection in certificate usage patterns
- Predictive analytics for certificate lifecycle events
- Automated incident response recommendations
- Certificate inventory optimization

Security teams are using these capabilities to shift from firefighting to strategic management:

- **Optimized Renewal Timing:** AI systems that determine the ideal time to renew certificates based on risk profiles and operational patterns.
- **Proactive Security Anomaly Detection:** Machine learning algorithms that identify unusual certificate usage that might indicate security incidents.
- **Automated Compliance Reporting:** Systems that continuously monitor certificate compliance and generate required documentation.
- **Capacity Prediction:** Forecasting tools that anticipate certificate infrastructure needs based on usage trends and planned initiatives.

7.2.3 Assessing Your PKI Maturity Level

Organizations preparing their certificate infrastructure for future challenges can benefit from understanding their current maturity level by leveraging established frameworks such as the PKI Maturity Model (PKIMM) offered by the PKI Consortium (pkic.org). The PKIMM framework builds upon the Capability Maturity Model Integration (CMMI) developed by Carnegie Mellon University, providing a structured approach to evaluating PKI capabilities.

7.2.4 Understanding the PKIMM Framework

The PKIMM model encompasses four modules and 15 categories that comprehensively address all aspects and activities related to PKI implementation, covering people, processes, and technology dimensions.

The four modules are designed to focus on specific components of PKI infrastructure:

Module	Description
Governance	Consist of the leadership, overall structures, and processes to enable the organization to use the PKI in a sustainable way. It also consists of having strategy and objectives and proper decision making
Management	Translates the governance into actions that support the PKI, management of the resources to maintain the required level of trust
Operations	Includes day-to-day business as usual activities that lead to a secure and future-proof PKI in accordance with the organization goals
Resources	Ensures that the activities related to the PKI are performed with proper knowledge and experience, with enough capacities, and that it provides complete and accurate information to relying parties

7.3 PKIMM Maturity Levels

The overall organizational maturity level is determined by evaluating performance across the 15 categories within the previous four modules. The PKI Maturity Model defines five progressive maturity levels:

Number	Maturity Level	Description
1	Initial	Unpredictable process with poor control and always reactive
2	Basic	Process is characterized by each particular case or project, and controls are often reactive
3	Advanced	Process is characterized by organizational standards and controls are proactive
4	Managed	Processes are measured and controlled, proactive approach
5	Optimized	Continuous improvement of the processes and procedures, proactive approach for future technology improvement

7.4 Conducting Your Assessment

Organizations can evaluate their current PKI maturity using the PKI Maturity Model Self-Assessment Tool, which provides structured evaluation across all framework dimensions.

To access the assessment tool and detailed framework documentation:

- PKI maturity model: https://pkic.org/pkimm/model/
- Self-assessment tool: https://pkic.org/pkimm/tools/self-assessment/

The assessment results help organizations identify their current capabilities, recognize improvement opportunities, and develop strategic roadmaps for advancing their PKI maturity in alignment with business objectives and emerging technology requirements.

7.5 The Path Forward

Understanding your PKI maturity level through PKIMM assessment provides a foundation for addressing emerging certificate management challenges. Organizations at lower maturity levels should prioritize establishing basic governance and operational processes before tackling advanced initiatives like post-quantum cryptography or zero-trust implementations. Those at higher maturity levels can leverage their existing capabilities to evaluate quantum-safe algorithms.

Organizations that align their certificate management capabilities with their assessed maturity level will be better positioned to capitalize on modernization opportunities while avoiding the operational overhead that constrains less prepared organizations.

7.6 Conclusion

Digital certificates form the security foundation for mainframe modernization initiatives, from mobile banking applications to AI-powered data integration. This document has shown that effective certificate management requires understanding fundamental concepts, implementing structured processes, and maintaining strategic alignment with organizational capabilities.

The PKIMM assessment framework provides an industry-standard approach for evaluating current capabilities and identifying improvement priorities. By understanding your maturity level, organizations can systematically build competencies that support both immediate modernization needs and future technological challenges.

As mainframe environments continue evolving to support modern digital experiences, certificate management will remain a critical enabler of secure, compliant operations. The principles and practices outlined in this document, combined with honest maturity assessment, provide the foundation for transforming certificate management from operational burden to strategic advantage.


BROADCOM®