Certificate Pinning Behind Enterprise Proxies

Giving Enterprise Admins Control

Glenn L. Austin, AustinSoft.com

Glenn Austin is an independent mobile security architect operating through AustinSoft in Mount Juliet, Tennessee. He has approximately fifty years of computing experience, including mobile security architecture at Atlassian and earlier work at Apple, Adobe, and Symantec. He holds US Patent 10,972,253 for cryptographic systems. The research presented in this paper was conducted independently as part of AustinSoft's consulting practice; the author has no affiliation with or financial interest in any certificate authority.

tl;dr;

Certificate pinning works great — until it encounters a closed network environment where its communications are blocked by a proxy or TLS introspection system. This paper will build upon the Certificate Pinning Done Right[1] white paper to show how certificate pinning can be done *even in an Enterprise managed environment,* where the enterprise is in charge of controlling Internet access. Doing so means making sure that any additions to the pinning chains are controlled and distributed at the end of the authentication process, meaning that enterprises only have to provide unrestricted internet access for the application authentication process, and provide the *additional* certificate pins necessary for their environment.

Designing for the Enterprise

Getting network security to work in an enterprise environment requires a different approach than getting it to work on the internet. Applications written that use network connections — especially in a zero-trust environment — have to be written to address the fact that connections may fail normal network hardening processes. The server being queried might be behind a TLS introspection system, or behind a proxy where the certificate the application sees for the connection is not the same certificate that was generated for that server. These configurations break normal certificate pinning, and require specialized configurations — or even disabling certificate pinning. With the OS-provided certificate pinning, none of this is possible, so another option has been to provide an enterprise-specific version of applications, where certificate pinning is disabled for these environments.

There is another way, and this paper will outline the process to implement administrator-managed pinning that will work both on the Internet and in enterprise networking environments. Critically, it doesn't compromise on security, maintaining the same level of security regardless of the network configuration.

Application-Provided Certificate Pinning for Enterprise Environments

OS-provided certificate pinning — such as Apple's App Transport Security — is designed for a single, fixed configuration. Once an application ships with OS-level pinning configured, that configuration cannot be changed without shipping a new version of the application. For consumer applications targeting the general internet, this rigidity is acceptable. For SaaS providers targeting enterprise customers, it creates an untenable situation.

Enterprise network environments vary significantly between organizations. A financial institution may deploy TLS introspection across all outbound connections. A healthcare provider may route all application traffic through an authenticated proxy. A defense contractor may operate in a fully air-gapped environment with its own certificate authority. No single OS-level pinning configuration can accommodate this range of network topologies — and the OS provides no mechanism to adapt at runtime.

The result is a difficult choice for SaaS providers: ship a consumer-grade application that breaks in enterprise network environments, ship an enterprise-specific build with certificate pinning disabled, or ship two discrete versions, one for normal customers and one for enterprises. None of those options is truly acceptable. The first creates support burden and failed deployments. The second removes a meaningful security control precisely in the environments where security requirements are most stringent. The third opens issues around development and support of two similar, but discrete versions that can be confusing for the users.

Application-provided certificate pinning addresses this directly. By managing the pinning configuration within the application rather than delegating it to the OS, the application gains the flexibility to adapt its pinning behavior to the enterprise

environment it's deployed in — without compromising the security guarantees that pinning provides.

CN Pinning in Enterprise Environments

The first paper in this series, *Certificate Pinning Done Right: Why Current Practice Pins the Wrong Thing*, established that effective certificate pinning should target the identity expressed through the certificate chain's naming conventions rather than the cryptographic key material. This approach — CN pinning — provides meaningful security across certificate rotations, where SPKI-based pinning fails, because organizational identity doesn't change when keys rotate.

CN pinning also provides the flexibility that enterprise deployment requires. Because pinning expressions are evaluated against certificate chain naming conventions rather than fixed cryptographic values, they can be extended at runtime to accommodate enterprise-specific certificate authorities, proxy certificates, or TLS inspection certificates — without abandoning the security guarantees that pinning provides.

This flexibility enables a three-mode operational model that maps naturally to the application authentication lifecycle, described below

Logging in

The application initiates its authentication process using its base CN pinning configuration. This configuration covers the application's own servers and represents the minimum pinning policy. The authentication process delivers the enterprise-specific pinning expressions, signed by the authentication server, before any enterprise network resources are accessed. In addition, it's an extremely limited scope opening within the enterprise network rules, and a SaaS provider would be well-served by placing their authentication endpoint on a different host to facilitate enterprises by limiting any network openings to *only* that host.

Finally, once the user has authenticated against the SaaS provider, they have the information necessary to provide additional information to the application, such

as enterprise-defined certificate pins, so that further communications can be secured using an expanded pinning configuration.

Logged in

The application can now operate with an expanded pinning policy — the base configuration plus the enterprise-specific expressions delivered during authentication. The updated CNPinning library (version 1.2.0) manages this as a composed, immutable configuration. Neither the base policy nor the enterprise additions can be modified at runtime; they can only be replaced by a new composed configuration.

Logging out

The enterprise-specific expressions are discarded. The application returns to its base CN pinning configuration, ready for the next authentication cycle.

This three-mode model ensures that enterprise network requirements are accommodated without permanently altering the application's security posture, and without requiring a separate enterprise build.

Services Provided from Authentication

The authentication process is the natural delivery point for enterprise-specific pinning configuration. The application must authenticate before accessing any enterprise resources, and the authentication endpoint is already the one host that enterprises must permit through their network controls. This makes it the logical — and secure — point at which to deliver additional configuration to the application.

Signed Pinning Policy Delivery

Enterprise-specific pinning expressions must be delivered as a signed artifact over a known-secure connection. These steps serve two purposes: they verify that the expressions originated from the SaaS provider's authentication server, and they prevent a man-in-the-middle from substituting their own pinning expressions during the bootstrap window.

The signing key is any private key part of a keypair, whether it is the authentication server's own private key or the private part of the keypair generated for this activity. The application validates the signature against the public part of that keypair, which is baked into the application and provided to CNPinning as an initialization parameter. This creates a trust chain rooted in the application itself — no external trust authority is required.

Note: The proxy specification does not require TLS on the internal leg between the proxy and the client. This is a known characteristic of proxy architecture that predates modern TLS deployment. TLS introspection systems depend upon this same characteristic — the proxy terminates the external TLS connection, inspects the content, and forwards it internally. Enterprise administrators who have deployed TLS introspection have already accepted this architectural reality within their controlled environment. The JWS signing of the pinning policy protects its integrity regardless of internal leg encryption — an attacker with access to the internal leg can observe the policy content but cannot forge a valid signature without the SaaS provider's private signing key, closing the insider man-in-the-middle attack vector.

Enterprise Administrator Responsibilities

The enterprise administrator provides their environment's certificate information to the SaaS provider — specifically, the CN expressions that represent their proxy, TLS inspection system, or internal certificate authority certificate pinning policy. The SaaS provider incorporates these expressions into the signed pinning policy delivered at authentication.

This is a deliberate operational boundary. The SaaS provider controls the signing and delivery mechanism. The enterprise administrator controls the content of the enterprise-specific expressions. Neither party can unilaterally alter the other's contribution.

Policy Format and Administration

The enterprise-specific pinning policy uses a JSON schema defined by CNPinning, including two JWT-required date claims (`iat` and `exp`)[2] and one or more

hostname or wildcard entries mapping to arrays of certificate chain expressions. The SaaS provider should validate the policy before signing and distributing it.

A policy document takes the following form:

```
{
  "iat": 1750000000,
  "exp": 1750086400,
  "example1.com": [[
    {"type": "suffix", "value": ".enterprise.example.com"},
    {"type": "exact", "value": "proxy.enterprise.example.com"}
  ]],
 "example2.com": [[
    {"type": "suffix", "value": ".enterprise.example.com"},
    {"type": "exact", "value": "proxy.enterprise.example.com"}
  ]]
}
```

Or, because CNPinning now takes the authentication hostname, which should never be pinned to anything other than what the SaaS provider specifies, the policy document could be simplified to:

```
{
 "iat": 1750000000,
 "exp": 1750086400,
 "*": [[
    {"type": "suffix", "value": ".enterprise.example.com"},
    {"type": "exact", "value": "proxy.enterprise.example.com"}
  ]]
}
```

The enterprise administrator provides this JSON through the SaaS provider's administrative interface. The SaaS provider validates the content, then wraps it as a JWS[3] artifact for delivery to the application. JWS is the appropriate delivery format because it explicitly requires signing — the signature is not optional or separable from the payload.

Note: The enterprise wildcard expression `"*"` augments the SaaS provider's existing certificate pinning — it does not replace or expand it to additional hosts. When a wildcard enterprise entry exists but *no base entry is defined for a given host*, CNPinning treats the connection as unpinned. This ensures that normally-pinned connections will continue to be pinned and enterprise administrators cannot unilaterally expand pinning to additional hosts.

The application receives the JWS then passes the payload to CNPinning for validation and composition with the base policy.

CNPinning 1.2.0 Implementation

The updated CNPinning library implementations[4,5] provide the client-side implementation of this model. Upon receiving a signed pinning policy from the authentication server, the application passes it to CNPinning for validation and composition, as demonstrated by these Swift examples, the Android version follows the same pattern:

```
let basePolicy = CNPinning(
    authenticationHost: hostname,
    policySigningKey: embeddedPublicKey
)
```

or

```
let basePolicy = CNPinning(
    authenticationHost: hostname,
    policySigningKey: embeddedPublicKey,
    configuration: [...]
)
```

and immediately after authentication, if a policy was provided by the authentication server:

```
try basePolicy.applyEnterprisePolicy(with: signedPolicy)
```

When the enterprise policy requires renewal before expiry:

```
try basePolicy.refreshEnterprisePolicy(with: signedPolicy)
```

On logout:

```
basePolicy.signOut()
```

The `enterprisePolicyExpiry` property returns the expiration date of the current enterprise policy, or `nil` if no enterprise policy is active. Applications should monitor this value and call `refreshEnterprisePolicy` before expiry to maintain uninterrupted enterprise connectivity. Applications can implement this as follows:

```
// Check expiry and refresh before it lapses
// refreshThreshold can be set to one hour before
// expiration or whatever you find acceptable
if let expiry = pinning.enterprisePolicyExpiry,
   expiry.timeIntervalSinceNow < refreshThreshold {
    // Fetch the latest pinning policy from your
    // authentication endpoint
    try pinning.refreshEnterprisePolicy(with: fetchedPolicy)
}
```

Applications should monitor `enterprisePolicyExpiry` and make regular calls to the authentication endpoint to refresh the enterprise pinning policy before it expires. The policy lifetime — derived from `iat` and `exp` — determines the appropriate refresh interval, typically one third of the policy lifetime. Calling `refreshEnterprisePolicy(with:)` with a newly signed policy keeps the enterprise configuration current without requiring the user to sign out and back in.

Applications should establish a maximum polling interval — recommended 24 hours — regardless of the policy's `exp` claim. Some enterprise configurations may set long-lived or non-expiring policies, but regular polling ensures the application receives certificate authority changes in a timely manner. The `enterprisePolicyIssuedAt` property can be used to detect when a new policy has been issued, even before the existing policy expires.

The enterprise policy is immutable once applied — neither the base policy nor the enterprise additions can be modified at runtime. `applyEnterprisePolicy` requires no existing enterprise policy; `refreshEnterprisePolicy` requires an existing one. Both validate the signed policy against the embedded public key before applying it. `signOut` clears the enterprise policy, returning the instance to its base configuration.

MAM and MDM Deployment Considerations

Mobile Application Management (MAM) and Mobile Device Management (MDM) provide fundamentally different mechanisms for enterprise deployment, and certificate pinning has unique considerations for each environment.

MAM Environments

In a MAM environment, the enterprise manages applications without managing the device. This creates a fundamental constraint that extends beyond certificate pinning — nothing works through an enterprise proxy unless the proxy's certificate or its issuing CA is already trusted by the device OS. MAM has no mechanism to install certificates into the device trust store.

This means enterprise proxy certificates in MAM environments must chain to a publicly trusted CA. Certificates that permit proxy interception from public CAs exist but are not generally available — they are tightly controlled precisely because they enable legitimate man-in-the-middle interception. Enterprises operating MAM environments should understand this constraint clearly before selecting a proxy strategy.

MDM Environments

MDM provides device-level management, including the ability to install certificates into the device trust store. This resolves the proxy certificate constraint that MAM cannot address. However, MDM also provides the ability to set UserDefaults values for any application — including values that could have potentially dangerous influence in providing pinning configuration.

Because pinning configuration is fundamental to the security of the application, it is this author's studied opinion that pinning configuration should not be delivered via UserDefaults, even in MDM environments. The authentication server delivery mechanism — signed JWS, validated against the embedded public key — maintains the integrity of the trust chain regardless of deployment model. UserDefaults bypasses that signing mechanism entirely, creating a tampering surface that undermines the security the pinning policy is meant to provide.

MDM's appropriate role is certificate delivery to the device trust store. Pinning policy delivery remains the authentication server's responsibility in both MAM and MDM environments.

Common Considerations

A significant benefit of delivering pinning configuration through the authentication server in both environments is that the application code path is identical regardless of deployment model. The application authenticates, receives the signed JWS policy if one exists, validates it, and composes the expanded CNPinning configuration. Whether the device is MAM or MDM enrolled is irrelevant to the application — the deployment complexity is handled at the infrastructure level, not the application level.

How We Move Ahead

Certificate pinning in enterprise environments is not a solved problem — it is a deployment problem. The cryptographic foundation is sound. The challenge is delivering and maintaining pinning configuration across the variety of network environments that enterprise customers operate.

The approach outlined in this paper addresses that challenge without compromising security. By delivering enterprise-specific pinning expressions through the authentication process — signed, validated, and composed with the base configuration — SaaS providers can support enterprise environments without shipping separate builds, disabling security controls, or requiring broad firewall exceptions.

CNPinning 1.2.0 provides the client-side implementation of this model for both iOS and Android. The three-mode operational lifecycle — logging in, logged in, logging out — maps directly to the authentication lifecycle that enterprise applications already implement. The additional code required is minimal. The security improvement is significant.

The CNPinning GitHub repositories include a bidirectional JWT reference implementation covering both policy signing (for the authentication server) and policy validation (for the mobile client). Server-side implementors can use this as a starting point, translating the signing logic to their backend language of choice. The operations involved — base64url encoding, JWS signature creation, and JSON serialization of the CNPinning schema — are supported by standard cryptographic libraries in all major server-side languages.

Enterprise deployment of certificate pinning is achievable. It requires deliberate architecture, clear operational boundaries between the SaaS provider and enterprise administrator, and implementation tooling that accommodates the constraints of both MAM and MDM environments. This paper has attempted to address all three.

---

[1] Austin, Glenn L. "Certificate Pinning Done Right: Why Current Practice Pins the Wrong Thing." AustinSoft, June 2026. https://www.austinsoft.com/white-papers/download/CNPinning.pdf

[2] Jones, M., J. Bradley., and N. Sakimura. "JSON Web Token (JWT)." RFC 7519, IETF, May 2015. https://www.rfc-editor.org/rfc/rfc7519

[3] Jones, M., J. Bradley., and N. Sakimura. "JSON Web Signature (JWS)." RFC 7515, IETF, May 2015. https://www.rfc-editor.org/rfc/rfc7515

[4] CNPinning-Apple. AustinSoftCom, GitHub. https://github.com/AustinSoftCom/CNPinning-Apple

[5] CNPinning-Android. AustinSoftCom, GitHub. https://github.com/AustinSoftCom/CNPinning-Android