# *Design and Implementation of Lightweight Public Key Infrastructure for Internal Organizational Document Signing*

Maria Vransiska Pingkhan - 18223119
Program Studi Sistem dan Teknologi Informasi
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: mariavransiska@gmail.com , 18223119@std.stei.itb.ac.id

***Abstract*—Organizations increasingly rely on digital documents for internal communication and decision-making. However, ensuring the authenticity, integrity, and non-repudiation of these documents remains a significant challenge. This paper presents the design, implementation, and evaluation of lightweight Public Key Infrastructure (PKI) intended for internal organizational document signing and verification. The proposed system utilizes X.509 digital certificates, Elliptic Curve Digital Signature Algorithm (ECSDA), and a simplified Certificate Authority (CA) architecture to provide secure document authentication while minimizing deployment complexity. A prototype was implemented using Python and OpenSSL-based cryptographic libraries. An experimental evaluation was conducted to measure signing time, verification time, certificate generation overhead, and tamper detection capability under different document sizes. The results demonstrate that the proposed lightweight PKI successfully ensures document integrity and authenticity with minimal performance overhead, making it suitable for small and medium-sized organizations.**

***Keywords—Public Key Infrastructure, PKI, Digital Signature, ECDSA, X.509 Certificate, Document Security***

## I. Introduction

The adoption of digital documents has significantly increased within organizations. Internal documents such as approval letters, financial reports, meeting minutes, and policy documents are frequently exchanged electronically. Although digital workflows improve efficiency, they also introduce security challenges related to document authenticity and integrity.

Traditional handwritten signatures are difficult to verify in digital environments and do not provide adequate protection against document tampering. Digital signatures based on public-key cryptography offer stronger security guarantees, including authentication, integrity, and non-repudiation.

Public Key Infrastructure (PKI) provides a framework for managing cryptographic keys and digital certificates required for digital signatures. However, enterprise-grade PKI systems are often costly and complex to deploy. Therefore, a lightweight PKI architecture may provide a practical alternative for internal organizational use.

## II. Theoretical Background

### *A. Public Key Cryptography*

Public Key Cryptography, also known as asymmetric cryptography, is a cryptographic approach that uses a pair of mathematically related keys (a public key and a private key). Asymmetric cryptography separates these operations to enhance security and simplify key distribution.

Public key cryptography enables digital signatures, where a sender (user) encrypts data using the recipient's public key, and only the corresponding private key can decrypt the ciphertext. This cryptography method also enables digital signatures, which is when a sender (user) signs a message using a private key and other parties verify the signature using the corresponding public key.

### *B. Digital Signatures*

A digital signature is a cryptographic mechanism used to verify the authenticity and integrity of digital data. Digital signatures ensure that a document originates from the claimed sender and has not been modified after signing. Unlike handwritten signatures, digital signatures provide cryptographic proof of authenticity and can be automatically verified by software systems.

The signing process begins by generating a cryptographic hash of the document using a hash function such as SHA-256. The resulting hash value is then encrypted using the signer's private key to create the digital signature. During verification, the receiver computes the hash of the received document and compares it with the hash recovered from the signature using the signer's public key. If both values match, the signature is considered valid. In this current era, digital signatures are widely used in electronic contracts, software distribution, secure email systems, and document management platforms.

### C. X.509 Digital Certificates

X.509 digital certificate is a standardized electronic credential used to associate a public key with the identity of an individual, organization, or device. The standard of this digital certificate is defined by the International Telecommunication Union (ITU-T) and is often adopted in modern PKI implementations.

A digital certificate contains several important fields, such as subject identity, issuer identity, public key information, serial number, validity period, and digital signature of the issuing Certificate Authority (CA). Verifying the CA's signature is important because it helps users establish trust in the certificate's authenticity and the public key it contains.

### D. Public Key Infrastructure (PKI)

Public Key Infrastructure (PKI) is a framework consisting of hardware, software, policies, procedures, and cryptographic mechanisms for managing digital certificates and public keys. PKI provides mechanisms for certificate issuance, validation, renewal, and revocation.

In PKI, there are five important things. First is the certificate authority (CA), which is responsible for issuing and signing digital certificates; second is the registration authority (RA), which verifies the identity of the certificate applicant before issuance; third is the digital certificates, which is to bind public keys to user identities; fourth is the certificate repository to store issued certificates; lastly is revocation mechanism which is to identifies certificates that are no longer trusted.

In a document signing system, PKI enables users to trust the authenticity of digital signatures by verifying certificates issued by a trusted CA. Organizations can deploy an internal PKI system to manage employee certificates and secure internal document workflows without relying on external certification providers.

### E. Elliptic Curve Digital Signature Algorithm (ECDSA)

The Elliptic Curve Digital Signature Algorithm is a digital signature scheme based on Elliptic Curve Cryptography (ECC). ECDSA provides equivalent security to traditional algorithms such as RSA while requiring significantly smaller key sizes. ECDSA operates using mathematical operations defined over elliptic curves. A user first generates an elliptic curve key pair consisting of a private key and a corresponding public key. During the signing process, the document hash is combined with the private key to generate a signature pair. Verification is performed using the public key and the document hash.

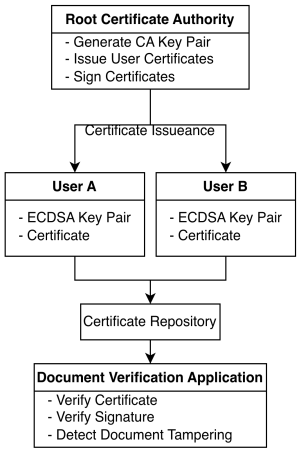
## III. System Design

### A. System Overview

The proposed system implements a lightweight Public Key Infrastructure (PKI) for internal organizational document signing and verification. The system consists of a Root Certificate Authority (CA), organizational users, a certificate repository, and a document verification module.

The Root CA is responsible for issuing and signing digital certificates for organizational members. Each user possesses a unique ECDSA key pair consisting of a private key and a public key. The private key is used to sign documents, while the public key is distributed through X.509 certificates issued by the CA.

When a document is created, the signer generates a digital signature using ECDSA and attaches the corresponding certificate. Recipients can verify both the certificate and the digital signature before accepting the document as authentic.




### B. Certificate Issuance Process

The certificate issuance process establishes trust between users and the Certificate Authority. Before signing documents, each user must obtain a valid X.509 certificate issued by the CA.

The process consists of the following steps:

1) *The user generates an ECDSA public-private key pair.*
2) *The user creates a Certificate Signing Request (CSR).*
3) *The CSR is submitted to the CA.*
4) *The CA verifies the request.*
5) *The CA issues and signs an X.509 certificate.*
6) *The certificate is stored in the certificate repository.*

### C. Document Signing Process

When a user wishes to sign a document, the document is first hashed using SHA-256. The resulting hash value is then signed using the user's ECDSA private key. The generated signature and the user's certificate are attached to the document. The final signed document package contains the original document, digital signature, and user certificate.

### D. Document Verification Process

The verification process ensures that the document originates from a legitimate signer and has not been modified after signing. The verification procedure consists of two stages, which are certificate validation and signature verification.

### E. Threat Model

The proposed system is designed to mitigate several common threats associated with digital document management.

a. Document Tampering
   An attacker modifies document content after it has been signed. SHA-256 hashing and ECDSA signature verification are used for mitigation.
b. Signature Forgery
   An attacker attempts to create a fake signature. Private key secrecy and ECDSA cryptographic security are used for mitigation.
c. Fake Certificates
   An attacker creates a forged certificate. Certificate validation using the CA public key is used for mitigation.
d. Unauthorized Signers
   An unauthorized user attempts to sign organizational documents. Only users with CA-issued certificates are trusted.

## IV. IMPLEMENTATION

### A. Development Environment

The proposed lightweight PKI system was implemented using Python and the Cryptography library. The implementation utilizes Elliptic Curve Digital Signature Algorithm (ECDSA) with the NIST P-256 curve (SECP256R1) and SHA-256 as the cryptographic hash function.

The system was designed as a modular application consisting of four primary modules:

1. Certificate Authority Management Module
2. Cryptographic Utility Module
3. Document Signing and Verification Module
4. Benchmark and Evaluation Module

### B. Cryptographic Utility Module

The cryptographic utility module provides reusable functions for cryptographic operations throughout the system. This component handles key generation, hashing, certificate serialization, Base64 encoding, and JSON processing.

#### 1) ECDSA Key Generation

The system generates ECDSA key pairs using the NIST P-256 elliptic curve.

```
def generate_ec_private_key() -> ec.EllipticCurvePrivateKey:
    return ec.generate_private_key(ec.SECP256R1())
```

Compared to RSA, ECDSA provides equivalent security with substantially smaller key sizes, making it suitable for lightweight PKI deployments.

#### 2) SHA-256 Hashing

Document integrity is protected using SHA-256.

```
def sha256_file(path: Path) -> str:
    digest = hashlib.sha256()
    with path.open("rb") as file:
        for chunk in iter(lambda: file.read(1024 * 1024), b""):
            digest.update(chunk)
    return digest.hexdigest()
```

Files are processed incrementally in chunks of 1 MB to improve scalability when handling large documents.

#### 3) Certificate Fingerprinting

Each certificate is uniquely identified using a SHA-256 fingerprint.

```
def certificate_fingerprint(certificate: x509.Certificate) ->
str:
    return certificate.fingerprint(hashes.SHA256()).hex()
```

The fingerprint is later used for certificate validation and revocation checking.

### C. Certificate Authority Implementation

The Certificate Authority (CA) acts as the trust anchor of the PKI. The CA is responsible for issuing, signing, and validating user certificates.

#### 1) Root CA Initialization

During initialization, the CA generates its own ECDSA key pair and creates a self-signed X.509 certificate. The root CA certificate is distributed to all organizational users and serves as the basis for certificate trust.

#### 2) Certificate Storage

The system stores cryptographic assets using the following structure:

```
workspace/
├── ca/
│   ├── ca_private_key.pem
│   └── ca_certificate.pem
├── users/
│   └── user
│       ├── certificate.pem
│       └── private_key.pem
└── signatures/
```

This structure separates CA credentials from user credentials and signed document packages.

### D. User Certificate Issuance

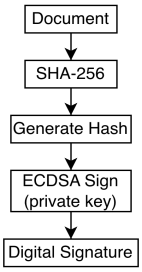
Each user receives an X.509 certificate issued by the internal CA. From the user, it generates a key pair and creates a CSR, then in the certificate authority, it verifies the request and signs the certificate, and an issued certificate as the output.

The issued certificate contains:

```
certificate = (
        x509.CertificateBuilder()
        .subject_name(subject)
        .issuer_name(issuer)
        .public_key(private_key.public_key())
        .serial_number(x509.random_serial_number())
        .not_valid_before(now - timedelta(minutes=1))
        .not_valid_after(now + timedelta(days=validity_days))
        .sign(private_key, hashes.SHA256())
    )
```

### E. *Document Signing Implementation*

Document signing if performed using this workflow:




The document hash is first computed using SHA-256.
A signing payload is then generated containing the document hash, certificate fingerprint, timestamp, and signature algorithm.

```
def _signature_payload(document_hash: str,
signer_cert_fingerprint: str, timestamp_utc: str) -> dict[str,
str]:
    return {
        "algorithm": "ECDSA-P256-SHA256",
        "document_hash_sha256": document_hash,
        "signer_certificate_fingerprint_sha256":
signer_cert_fingerprint,
        "timestamp_utc": timestamp_utc,
    }
```

The payload is digitally signed using the signer's private key.

```
signature = private_key.sign(canonical_json(payload),
ec.ECDSA(hashes.SHA256()))
```

The resulting signature is encoded using Base64 and stored inside a JSON signature package.

### F. *Document Verification Implementation*

Verification is performed in multiple stages to ensure document authenticity and integrity.

1. Stage 1: Hash Validation
   The SHA-256 hash of the received document is recalculated. The generated hash is compared with the signed hash stored in the signature package. If the values differ, the document is immediately rejected.
2. Stage 2: Certificate Validation
   The signer's certificate is validated against the Root CA certificate.
3. Stage 3: Certificate Validity Check
   The certificate validity period is examined. The expired certificates are rejected.
4. Stage 4: Revocation Check
   The certificate fingerprint is checked against the revocation database. Certificates that have been revoked cannot be used for signing.
5. Stage 5: Signature Verification
   Finally, the ECDSA signature is verified using the public key contained within the certificate.
   If verification succeeds, the document is considered authentic and unmodified.

### G. *Certificate Revocation Implementation*

The proposed PKI includes a lightweight certificate revocation mechanism. Revoked certificates are maintained in a JSON-based revocation list.

## V. Experimental Evaluation

The evaluation focuses on whether the system can correctly support certificate issuance, document signing, signature verification, multi-user signing, document modification detection, certificate revocation, and lightweight certificate usage. The main cryptographic algorithms used in the implementation are ECDSA P-256 for digital signatures and SHA-256 for document hashing. The experiments were conducted using internal test documents with different file sizes, 100 KB, 1 MB, and 10 MB.

### A. *Experimental Setup*

The experimental setup consists of an internal root Certificate Authority, multiple internal users, and digital documents used as signing objects. The CA is responsible for issuing X.509 certificates for internal users. Each user owns a private key and a certificate signed by the internal CA.
The following test users were created:

| User ID | Department | Purpose |
|---|---|---|
| finance_admin | Finance | Main document signer |
| legal_manager | Legal | Additional signer |
| hr_staff | Human Resources | Additional signer |

### B. *Certificate Issuance and Valid Document Verification*

The first experiment verifies whether the system can initialize and internal CA, issue a user certificate, sign a document, and verify the generated signature. The CA was initialized using a self-signed X.509 certificate. After that, a certificate was issued for finance_admin.

The document was signed by finance_admin, and the resulting signature package was verified using the original document and the internal CA certificate. The verification result is as shown:

```
================================================================================
TEST 1 — CA INITIALIZATION AND CERTIFICATE ISSUANCE
================================================================================
Step                          | Result
------------------------------+-------------------------------------
Internal root CA created      | Success
Signer certificate issued     | Success
CA fingerprint                | 08d27962242f0f1ceb12368f15f13506...
Signer fingerprint            | 274fb56b2093e0a6089643e54323eab3...

================================================================================
TEST 2 — DOCUMENT SIGNING AND VALID VERIFICATION
================================================================================
Condition          | Expected | Actual | Detail
-------------------+----------+--------+-----------------------
Original document | Valid    | Valid  | Verification succeeded
```

The result shows that the system successfully verifies a document when the document content digital signature, signer certificate, and CA trust chain are valid.


```
{
  "document": "data/documents/manual_approval.txt",
  "document_hash_sha256": "149829f97d1f04a3a9e3c6a0fcb4e1558688c8e7300a81fb0f61e0e46af12ebf",
  "signed_at_utc": "2026-06-19T12:56:22.556732+00:00",
  "signer_common_name": "Finance Admin",
  "signer_user_id": "finance_admin",
  "status": "VALID",
  "valid": true
}
```

The verification output also identifies the signer as finance_admin, proving that the system supports signer authentication.

### C. Multi-User Signing and Verification

The second experiment evaluates whether the system supports multiple internal users. Three users were issued separate certificates and private keys. Each user signed the same document independently. The verification process was then performed for each generated signature package.

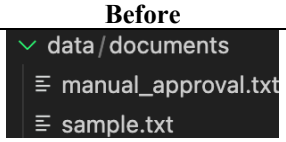
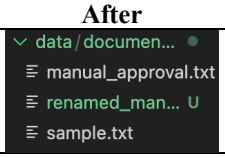
```
TEST 3 — MULTI-USER SIGNING AND VERIFICATION
User ID       | Department      | Expected | Verified Signer | Detail
finance_admin | Finance         | Valid    | finance_admin   | Signer identity matched
legal_manager | Legal           | Valid    | legal_manager   | Signer identity matched
hr_staff      | Human Resources | Valid    | hr_staff        | Signer identity matched
```

The results show that the system can distinguish different singers even when they sign the same document. This is possible because each signature package contains the signer certificate, and the verification process check the signer identity from the certificate subject.

### D. Document Modification Detection

The third experiment evaluates whether the system can detect changes to the signed document. Four conditions were tested: the original document, a document with modified content, a document with a changed file name but identical content, and a document with additional content appended.

Test 1: Change Document Name

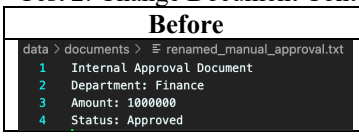
| Before | After |
|---|---|
|  |  |



The document result will be VALID because the content is still the same.


```
{
  "document": "data/documents/renamed_manual_approval.txt",
  "document_hash_sha256": "149829f97d1f04a3a9e3c6a0fcb4e1558688c8e7300a81fb0f61e0e46af12ebf",
  "signed_at_utc": "2026-06-19T12:56:22.556732+00:00",
  "signer_common_name": "Finance Admin",
  "signer_user_id": "finance_admin",
  "status": "VALID",
  "valid": true
}
```


Test 2: Change Document Content

| Before | After |
|---|---|
|  |  |



Because the content has changed, the result of the verification is going to be ERROR: Document hash mismatch. The document was changed or the wrong file was selected.

The picture next shows the whole modification testing result:

```
TEST 4 — DOCUMENT MODIFICATION DETECTION
Condition        | Expected    | Actual      | Detail
Asli             | Valid       | Valid       | Verification succeeded
Isi diubah       | Tidak valid | Tidak valid | Document hash mismatch. The document was changed or the wrong file was selected.
Nama file diubah | Valid       | Valid       | Verification succeeded
Konten ditambah  | Tidak valid | Tidak valid | Document hash mismatch. The document was changed or the wrong file was selected.
```

The results show that the system successfully detects document content modification. When the content was changed or additional content was appended, the verification failed because the SHA-256 hash of the document no longer matched the hash stored in the signature package.

The file name change case remained valid because the proposed system signs the document content hash, not the file name. This design decision ensures that the cryptographic identity of the document depends on its content rather than its storage name.

### E. Certificate Revocation Testing

The fourth functional experiment evaluates whether the system rejects signatures from revoked certificates. After finance_admin signed a document, the certificate was revoked and added to the revocation list. The same signature was then verified again.

```
TEST 5 — CERTIFICATE REVOCATION
Condition           | Expected    | Actual      | Detail
Certificate revoked | Tidak valid | Tidak valid | Signer certificate has been revoked
```

The verification failed with the message indicating that the signer certificate had been revoked. This confirms that the verification process does not only check the signature mathematically, but also validates the current trust status of the signer certificate.

### F. Performance Evaluation

The performance evaluation measures signing time and verification time for documents with different file sizes. Each file size was tested for 30 rounds, and the average time was recorded.

```
TEST 6 — PERFORMANCE BENCHMARK
CSV output: workspace/paper_tests_20260619_194420/benchmark/performance_results.csv
File Size | Avg Signing Time | Avg Verification Time | Status
100 KB    | 0.6205 ms        | 0.6530 ms             | VALID
1 MB      | 0.9990 ms        | 1.0196 ms             | VALID
10 MB     | 5.4258 ms        | 5.6493 ms             | VALID
```

The results show that the proposed system can perform document signing and verification with low processing overhead. Larger documents require more processing time mainly because the SHA-256 hashing process must read the entire file content. However, even for a 10 MB document, the average signing and verification times remain practical for internal organizational use.

The results also show that verification time increases with file size. This is expected because the verification process includes document hashing, certificate validation, revocation checking, and ECDSA signature verification.

### G. Certificate Size Comparison

To evaluate the lightweight characteristic of the proposed system, the certificate and public key sizes of RSA 2048 and ECDSA P-256 were compared.

```
TEST 7 — CERTIFICATE SIZE COMPARISON
CSV output: workspace/paper_tests_20260619_194420/benchmark/certificate_size_comparison.csv
Algorithm   | Certificate Size | Public Key Size
RSA 2048    | 1168 bytes       | 451 bytes
ECDSA P-256 | 639 bytes        | 178 bytes
```

The result shows that ECDSA P-256 produces a smaller certificate and public key compared to RSA 2048. The

ECDSA P-256 certificate is approximately 45.3% smaller than the RSA 2048 certificate. This supports the design goal of building a lightweight PKI system for internal document signing.

### *H. Implementation Result*

The experimental results demonstrate that the proposed implementation satisfies the main security requirements of internal document signing. Integrity is achieved because any modification to the document content causes verification failure. Authentication is achieved because the signer identity is bound to an X.509 certificate issued by the internal CA. Non-repudiation is supported because each signature is generated using the signer's private key.

The system also supports certificate revocation, which is important when a private key is compromised or when a user is no longer authorized to sign documents. Although the revocation mechanism in this implementation is based on a simple JSON revocation list, it is sufficient for demonstrating the concept of lightweight internal PKI.

However, the implementation still has several limitations. The private keys are stored as PEM files without password encryption. The revocation list is file-based and does not yet support distributed synchronization. The system also does not include an external timestamp authority, hardware security module, or web-based management interface. These limitations can be addressed in future work.

## VI. Conclusion

The design and implementation of a lightweight Public Key Infrastructure for internal organizational document signing prototype provides essential PKI functions, including internal CA initialization, certificate issuance, document signing, signature verification, certificate revocation, and performance benchmarking.

The implementation uses ECDSA P-256 and SHA-256 to provide digital signatures with relatively small key and certificate sizes. The system stores signatures in a JSON-based signature package that contains the document hash, signing metadata, signer certificate, and digital signature. During verification, the system checks the document hash, signer certificate, certificate chain, certificate validity period, revocation status, and ECDSA signature.

Experimental results show that the system successfully verifies valid documents, detects modified document content, supports multiple internal signers, and rejects signatures from revoked certificates. The performance evaluation also shows that signing and verification can be performed efficiently for documents up to 10 MB. In addition, the certificate size comparison shows that ECDSA P-256 produces smaller certificates and public keys than RSA 2048, making it suitable for a lightweight internal PKI environment.

Overall, the proposed system demonstrates that a lightweight PKI can be implemented effectively for internal organizational document signing. Future work may include encrypted private key storage, audit logging, integration with organizational identity systems such as LDAP or SSO, a web-based interface, and support for trusted timestamping.

## GitHub Link

https://github.com/MariaVransiska/LightweightPKI


## Acknowledgment

The author would like to express sincere gratitude to Dr. Ir. Rinaldi Munir, M.T., for providing valuable guidance and insights throughout the completion of this study. The author also acknowledges the support of academic resources that contributed to the successful design, implementation, and evaluation of the proposed lightweight Public Key Infrastructure system.


## References


[1] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*, 3rd ed. Boca Raton, FL, USA: CRC Press, 2020.

[2] NIST, "Digital Signature Standard (DSS)," NIST FIPS PUB 186-5, National Institute of Standards and Technology, Gaithersburg, MD, USA, Feb. 2023.

[3] NIST, "Secure Hash Standard (SHS)," NIST FIPS PUB 180-4, National Institute of Standards and Technology, Gaithersburg, MD, USA, Aug. 2015.

[4] S. Turner, D. Brown, K. Yiu, R. Housley, and T. Polk, "RFC 5280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," Internet Engineering Task Force (IETF), May 2008.

[5] R. Housley, T. Polk, W. Ford, and D. Solo, "Internet Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," IETF RFC 5280, 2008.

[6] NIST, "Recommendation for Key Management: Part 1 – General," NIST Special Publication 800-57 Part 1 Rev. 5, National Institute of Standards and Technology, 2020.

[7] The Python Cryptographic Authority, "Cryptography Documentation," Available: https://cryptography.io/. Accessed: Jun. 2026.

[8] M. Toorani and A. A. Beheshti Shirazi, "LPKI - A Lightweight Public Key Infrastructure for the Mobile Environments," arXiv:1002.3299 [cs.CR], 2010.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 18 Juni 2026

Maria Vransiska Pingkhan 18223119