KAPITEL 5 / *CHAPTER 5* [5]
SEARCHING FOR SOLUTIONS FOR SERVER IMPLEMENTATION OF DIGITAL SIGNATURE IN AN AUTOMATED DOCUMENT MANAGEMENT SYSTEM

DOI: 10.30890/2709-2313.2024-33-00-010

Introduction

In an increasingly digitalized world, secure and efficient document management has become essential for modern organizations. Implementing digital signatures within electronic document management systems facilitates workflow and enhances security by ensuring document authenticity, integrity, and non-repudiation. The effectiveness of such systems depends on robust cryptographic methods and structured management of both private and public keys. This research addresses the current challenges and proposes an optimized digital signature implementation approach, focusing on symmetric and asymmetric encryption methods. Through this work, we aim to improve document processing efficiency and security in electronic document management systems [1].

Relevance of the Research Topic: Achieving the objectives of any organization reflects the efficiency of its information management system, particularly in aspects of human resources management. Automated document processing is key for companies' and enterprises' effective operation and continuous development. A significant element of this process is the digital signature module, which enhances convenience, speeds up, and reduces the costs of document signing. Therefore, there is a need to analyze and improve methods and tools for implementing digital signatures in a specialized document management system. The primary focus is improving the document processing workflow by developing and implementing a new digital signature module for the electronic document management system.

Purpose of the Research: This research aims to increase the efficiency of the electronic document management system by implementing a digital signature subsystem.

[5]***Authors:*** *Korobeinikova Tetiana Ivanivna*

5.1. Understanding Digital Signatures and Their Implementation

Concept of Digital Signatures and Their Realization

A digital signature (DS) is an electronic document attribute designed to protect it from forgery. It is generated by cryptographic transformation using a private key, enabling the identification of the certificate owner and confirming that no information distortion has occurred in the document, thereby ensuring the signatory's non-repudiation [2].

Digital signatures are often used with hash functions, verifying that the message remains unchanged. Hash functions produce a fixed-size string (message digest) based on the content, where even minor changes in the text will alter the digest. The process of applying a hash function includes several steps:

1) Before sending, the message is processed through the hash function, producing a compressed digest without altering the original text, which remains encrypted for secure transmission.
2) The resulting digest is encrypted with the sender's private key (signed with the DS) and sent to the recipient.
3) The recipient decrypts the digest using the sender's public key.
4) The recipient processes the message with the same hash function to obtain the digest. If both digests match, the message has not been altered.

There are several widely used hash functions: MD5, SHA-1, etc. A digital signature scheme generally includes the following components:

- Key pair generation algorithm for users,
- Signature computation function,
- Signature verification function.

There are two primary digital signature schemes:

1) Symmetric Encryption-Based Schemes: This scheme involves a trusted third party, an arbitrator, who authorizes the document by encrypting it with a secret key.
2) Asymmetric Encryption-Based Schemes: These schemes are more widely

used and applicable. Besides, other types of digital signatures (group signatures, undeniable signatures, trusted signatures) also exist and have been modified to meet various needs.

Symmetric Scheme for Digital Signatures [3].

The cryptographic strength of symmetric encryption schemes depends on the selected algorithm. For a symmetric (single-key) algorithm to be reliable, it should meet specific properties:

- Algorithm Strength: It must prevent decryption without a key, relying on statistical regularities or other analysis methods.
- Confidentiality Dependence: Security should depend on the key's confidentiality, not the algorithm's. Specialists should verify the algorithm to rule out weak spots.
- Key Confidentiality: The algorithm must not compromise the key, even if numerous plaintext-ciphertext pairs are available.

Symmetric schemes are known for fast encryption and decryption, making them suitable for quickly encrypting large data volumes while ensuring confidentiality, authenticity, and integrity.

5.2. Types of Symmetric Algorithms

Symmetric encryption algorithms are categorized into:

- Block Ciphers: These encrypt data in blocks (64 or 128 bits in DES or AES, for example), processing them with a set sequence, often including several hashing and substitution rounds.
- Stream Ciphers: Here, each bit or byte of the plaintext is encrypted using a sequence of pseudorandom bits.

Standard symmetric encryption algorithms include AES, DES, IDEA, Blowfish, RC4, RC5, and RC6.

Asymmetric Digital Signature Scheme. Asymmetric digital signature schemes are

part of public-key cryptography. Unlike asymmetric encryption, which uses a public key for encryption and a private key for decryption, digital signature schemes sign documents with a private key and verify them with a public key.

A typical digital signature scheme involves three processes:

1) Key Pair Generation: A private key is randomly selected, and the corresponding public key is calculated.
2) Signature Generation: The private key generates a signature for a specific electronic document.
3) Signature Verification: The document and signature are validated using the public key to confirm the signature's legitimacy.

To ensure the usefulness of digital signatures, two conditions must be met:

- The signature must be verifiable with the public key corresponding to the private key used for signing.
- It should be computationally challenging to forge a legitimate digital signature without the private key.

Types of Asymmetric Algorithms. To maintain security, asymmetric digital signature algorithms rely on complex computational problems, such as:

- Discrete Logarithm Problem (EGSA)
- Factorization Problem (RSA)

Digital signature algorithms are generally divided into regular digital signatures and document-recoverable signatures. Document-recoverable digital signatures automatically restore the document during verification, unlike regular signatures, which require attaching the document to the signature. For example, RSA can be used for document-recoverable signatures [4].

RSA Algorithm Overview. The RSA algorithm is an asymmetric cryptographic algorithm that operates using two keys: public and private keys. The public key is accessible to anyone, while the private key remains confidential.

Here's a typical use case:

- The client sends a public key to the server and requests some data.
- The server encrypts the data with the client's public key and sends it back.

- The client decrypts the data using their private key.

RSA Key Generation Steps:

1) Select two large prime numbers, p and q.
2) Compute $n = p * q$
3) Choose a public key e such that e is not a factor of $(p-1) * (q-1)$.
4) Compute a private key d such that $e \times d \equiv 1 mod\ (p-1)(q-1) = 1$ or d is the modular inverse of E modulo $(p-1)(q-1)$.

In RSA, d is private; e and n are public:

1) Alice creates her digital signature by calculating $S = M^d\ mod\ n$, where M is the message.
2) Alice sends the message M and the signature S to Bob.
3) Bob computes $M_1 = S^e\ mod\ n$.
4) If $M_1 = M$, then Bob accepts the data sent by Alice.

5.3. Management and Storage of Public and Private Keys

Public Key Management: A crucial issue in public-key cryptography, including digital signatures, is managing public keys. Since public keys are accessible to all, a mechanism is needed to verify their authenticity. This mechanism must allow any user to access the genuine public key of another user, protect these keys from malicious substitution, and facilitate key revocation if compromised.

This issue is addressed with certificates. Certificates authenticate data about the key owner and their public key through the signature of a trusted party. Certificate systems can be centralized or decentralized. In decentralized systems, a trusted network is built by cross-signing certificates between trusted individuals. Centralized systems rely on certification authorities (CAs), which generate private keys and certificates for end users, verify their authenticity with a digital signature, and manage certificate revocation records.

Private Key Storage: The private key is the most vulnerable part of a digital

signature system. If an attacker obtains a user's private key, they can sign any document in their name. Thus, careful attention must be given to private key storage methods. Users can store their private key on their personal computer, protecting it with a password. However, this approach depends entirely on the computer's security and restricts the user to signing documents only on that device .

Currently, private keys are often stored on devices like:

- Floppy disks
- Smart cards
- USB drives
- Touch Memory tokens

The loss or theft of such devices is usually quickly noticeable, allowing the user to revoke the relevant certificate immediately. The **most secure method** is storing the private key on a smart card, which requires both possession of the card and a PIN code, providing **two-factor authentication**. The document or its hash is sent to the card's processor for signing, and the private key is never copied or exposed outside the card.

5.4. Analysis of Signature Forgery Attempts

Attempts to forge a digital signature or signed document are called "attacks" in cryptanalysis. Standard attack models and their possible outcomes include:

- Public Key Attack: The cryptanalyst only has access to the public key.
- Known Message Attack: The attacker possesses valid signatures for known documents.
- Adaptive Chosen Message Attack: The attacker can obtain signatures for documents they choose.
- Complete Signature Break: Obtaining the private key equates to a complete algorithm compromise.
- Universal Signature Forgery: Discovering an algorithm that can forge signatures for any document.

- Selective Signature Forgery: Forging signatures for specific documents chosen by the cryptanalyst.
- Existential Signature Forgery: Producing a valid signature for an arbitrary document without pre-selection.

Among these, the *adaptive chosen message attack* is considered the most critical, as modern digital signature algorithms are designed to resist it through computationally secure processes.

5.5. Digital Signature Methods

A digital signature encompasses a broad range of methods applicable to electronic documents to capture signing intent and approval and, to various extents, ensure document authenticity, integrity, and non-repudiation. When implemented according to relevant laws and regulations, a digital signature can carry the same legal weight as a handwritten signature on a paper document.

Most documents are now generated on digital platforms and managed within *Electronic Document Management Systems (EDMS)*. These systems support adequate document storage, organization, and transmission on electronic platforms. Digital signatures enhance EDMS by allowing documents to be signed electronically, eliminating the need for paper-based processes. The core feature of digital signatures within EDMS is non-repudiation, ensuring that the signature cannot be denied once affixed.

Digital Signature in Browser Applications. Electronic document management systems typically operate within browser-based environments containing server and client applications that communicate through network protocols. Users interact with the application via web browsers and an interface that houses the client application. Digital signature applications need to access data from a digital signature USB key.

Each individual has a unique USB key. The client application must be able to access hardware resources, but this access depends on the browser's permissions. Most

browsers do not natively support access to computer hardware, which is often addressed using third-party programs that work within the browser, such as Java applets.

A Java applet is a small program written in Java or another programming language, compiled into Java bytecode, and delivered to users in this form. Users run the applet from a web page, which executes within the Java Virtual Machine (JVM) separately from the web browser. By using Java applets, users can sign documents directly in the browser. The applet detects hardware changes on USB ports and allows users to enter a password to access the digital signature key. Once entered, the applet signs the document digitally and submits it to the server for storage.

5.6. Digital Signature in Desktop Applications

In some cases, running Java applets in a browser is either restricted or not best practice. While Java applets provide a convenient way to sign documents through the browser, popular browsers are phasing out support for the Netscape Plugin Application Programming Interface (NPAPI) required for Java applets. Additionally, granting browser access to hardware and local files raises security concerns. Desktop applications offer a solution in these cases, though each computer must install the software, and each update requires reinstallation.

The user logs in using a desktop application with a username/password or digital signature credentials. Once authorized, the application retrieves data from the server and displays documents awaiting signatures. The user then connects their USB key, and the application reads data from the key, allowing them to enter a password for signature authorization. The document is signed with a digital signature and sent back to the server. Desktop applications can also verify digitally signed documents, showing the signer's identity and timestamp if available, adding a layer of validity.

Modern API Development Technologies. Today, the popularity of web services and platforms largely depends on the internal structure and hierarchy of the software.

Developing an efficient, high-performance API enables easy integration into other services and applications. A well-organized backend is critical for seamless functionality expansion and scalability. Providing an API allows developers to popularize the product, identifying new use cases and applications [5].

Microservice Architecture. Microservices represent an architectural approach to building applications from small to large scales. With this approach, the application is divided into small, independent components. Unlike monolithic architectures, where all functionalities are embedded into a single project/application, microservices enable modular development, allowing all modules to work together to fulfill specific target tasks.

This software architecture offers greater modularity, flexibility, and the ability to reuse similar functions across multiple applications.

Benefits of microservices include:

- Faster Development: By dividing the application into smaller modules, development time is reduced, enabling quicker changes to existing software functionalities and more straightforward issue resolution in production.
- High Scalability: As user demand for certain services grows, those services can be deployed on multiple servers and infrastructures. Load balancing can efficiently handle the increased service demand from end users.
- Resilience and Independence: Independently developed services ensure that failure in one does not disrupt the entire application, unlike in monolithic models.
- Easier Deployment: Microservices-based applications are more modular, reducing deployment issues.
- Ease of Development: As larger applications are divided into smaller modules, developers can more easily understand, update, and improve functionality.
- Openness: Microservices do not depend on a specific language or technology, allowing developers to choose the best options.

REST Architectural Style. REST API (Representational State Transfer API) refers

to APIs that follow REST design principles. REST was defined in 2000 by computer scientist Dr. Roy Fielding in his dissertation and offered high flexibility and freedom for developers, making RESTful APIs a common choice for connecting components and applications in a microservices architecture.

At its simplest, an API enables a program or service (client) to access resources in another program or service (server). REST APIs are known for their adaptability, support various data formats, and allow communication over HTTP.

The six main REST architectural constraints are:

1) Uniform Interface: Ensures consistent API request formats regardless of the source.
2) Client-Server Independence: The client needs only the URI of the resource; the server handles the data processing.
3) Statelessness: Each request must contain all necessary information without relying on server sessions.
4) Cacheability: Resources should be cacheable on the client or server side to boost performance.
5) Layered System Architecture: Requests pass through different layers, preventing the client or server from directly connecting.
6) Code on Demand (Optional): REST APIs may sometimes send executable code (like Java applets) that runs on demand.

REST APIs use HTTP requests to perform standard CRUD operations–Create, Read, Update, Delete–on resources. A well-designed REST API operates like a website, functioning seamlessly within a web browser with inherent HTTP capabilities.

5.7. Solutions for Server Implementation of Digital Signature in an Automated Document Management System

Tasks formulation for digital signature use in Electronic Document Management Systems are shown in this section. The education sector, particularly universities, has

increasingly adopted digital solutions, including digital signatures, to facilitate remote administrative tasks and expedite signing official documents. Universities handle numerous documents requiring signatures, such as departmental protocol extracts. Digital signatures provide a reliable solution to protect these confidential documents [7].

Using digital signatures simplifies, automates, and digitizes standard administrative processes, improving efficiency and resource optimization. When an electronic document is signed, its original content remains unchanged; a data block known as a digital signature is appended, which must meet these requirements:

1) The signature should be a unique bit pattern dependent on the document.
2) The signature should incorporate unique sender information to prevent forgery or denial.
3) Creating the digital signature should be a relatively simple process.
4) Forging a digital signature must be computationally infeasible, whether by generating a new message for an existing signature or creating a fake signature for a given document.
5) The digital signature should be compact, requiring minimal storage space.

A strong hash function, encrypted with the sender's private key, fulfills these requirements.

Considering this research, developing a RESTful Web API service for implementing digital signature capabilities within a microservice architecture for electronic document management systems is recommended. The proposed service should be implemented using ASP.NET Core RESTful Web API technology, enhancing the efficiency of electronic document management systems through a secure digital signature subsystem.

Summary and conclusions

This research established that integrating digital signatures into electronic document management systems significantly enhances the security and efficiency of document management in organizations. Digital signatures ensure the authenticity, integrity, and non-repudiation of documents, which are critical aspects for information protection. The proposed approach to digital signature implementation, utilizing both symmetric and asymmetric cryptographic methods, allows optimization of document workflow and increases reliability.

Workflow Optimization – the proposed system structure, which employs a RESTful Web API within a microservice architecture, allows for efficient integration of digital signatures into existing document management systems, reducing maintenance and modernization costs.

Confidentiality and Authenticity Protection – the use of asymmetric cryptography secures documents against forgery and misuse, as the signature verification is performed using a public key, enabling user authentication directly within the system.

Resistance to Attacks – the selected cryptographic methods protect the system from critical attacks, including the adaptive chosen message attack, which is one of the most serious threats to digital signature security.

Scalability Potential – due to the microservice architecture of the proposed system, flexibility is increased, allowing easy adaptation to meet the specific needs of organizations of various sizes and industries.

The proposed solutions create a reliable platform for storing and managing digitally signed documents, which is especially important for the education sector and other fields where document confidentiality and legitimacy are of particular importance.