World Journal of Management Science
Print ISSN: 2959-9628
Online ISSN: 2959-9636
DOI: https://doi.org/10.61784/wms3077

IMPLEMENTATION OF TRUELICENSE, AN OPEN-SOURCE CERTIFICATE MANAGEMENT ENGINE

ZheKai Huang
Portland Institute, Nanjing University of Posts and Telecommunications, Nanjing 210023, Jiangsu, China.
Corresponding Email: hzk040317@outlook.com

Abstract: This article addresses the shortcomings of the open-source certificate management engine TrueLicense in terms of security, flexibility, and usability. A new license management system based on RSA asymmetric encryption and hardware binding is proposed, with a focus on strengthening three aspects: preventing time callback attacks through encrypted time anchors, transferring core verification logic to C++to resist JAVA decompilation risks, and implementing code obfuscation to protect software intellectual property. The system significantly improves authorization security through multiple security verification mechanisms, while adopting a non-invasive integrated design to lower the threshold for developers to use. This system can effectively defend against time callback attacks and license forgery, support distributed deployment in high concurrency scenarios, and provide an automated integration solution based on Maven. This article elaborates on the implementation principles, architecture design, and operational processes of the system, providing a reliable solution for enterprise level software authorization management.
Keywords: TrueLicense framework; Key pair; Client-side validation; Three-tier architecture

1 INTRODUCTION

Computer programs have become an important foundation for socio-economic development, and countries generally adopt copyright protection laws to establish developers' ownership of source code to safeguard fundamental interests. Licenses are divided into two categories: proprietary licenses and open source licenses. Exclusive licenses allow developers to commercially release paid versions of closed source software, making revenue generation easier. Open source licenses preserve fundamental freedoms to make software more accessible. Developers choose their software licensing strategy based on factors such as usage type, time, region, etc. Although developers have gradually realized the importance of effective licensing software, many issues limit the popularity of license usage. Many developers, especially those of dependent languages such as JavaScript, adopt a multi license model in their software to alleviate conflicting dependencies and help address the need for profitable open source development, but still face license incompatibility issues. With the application of new technologies such as containerization and microservices, traditional licensing methods are no longer sufficient to meet demand, affecting the economic interests of software vendors and posing a threat to the secure and stable operation of critical infrastructure. These challenges urgently need to be addressed.

As the current mainstream open source certificate management framework, TrueLicense provides basic support for license management of Java applications, but it exposes various limitations in practical applications. Firstly, the time verification mechanism of this framework has serious flaws. License objects usually contain fields for effective time and expiration time, and the current system time is compared with these two fields during verification. To launch a timestamp forgery attack, the opponent only needs to forge the timestamp of a new block. In the event of an attack, the attacker's block timestamp becomes a timestamp, which prevents the opponent's block from being immediately verified upon receipt and marks it as "temporarily invalid" [1]. Secondly, the TrueLicense native version does not include a hardware acquisition module, so developers need to manually input a large amount of verification code in critical business processes, which not only increases system complexity but also increases performance overhead. Tests have shown that in a distributed environment, the centralized verification mode of TrueLicense can cause significant network latency, making it difficult to meet the requirements of high concurrency scenarios. In addition, the framework lacks flexible policy configuration capabilities and cannot support fine-grained authorization based on dimensions such as usage and functional modules. These defects severely limit the application value of TrueLicense in commercial scenarios with high security requirements.

2 IMPLEMENTATION PRINCIPLE OF KEY PAIR GENERRATION

2.1 Key Pair Generation and Digital Signature

This system adopts RSA asymmetric encryption mechanism to achieve license authentication and secure signature. The Java Keytool generates RSA 1024 bit public-private key pairs, with the private key stored in the privateKeys.keystore key repository and double password protection set for storepass and keypass. The authorization information (including expiration date, hardware serial number, and other key data) is organized in JSON format, and a digital signature is generated using the private key of the key data, which is then verified through the corresponding public key to ensure the integrity and authenticity of the data [2]. This mechanism ensures both encryption security and identity authentication, making it impossible to forge valid licenses even if the public key is made public. At the same time, the system combines hardware serial number binding and time verification (anti callback mechanism) to provide multiple security protections, fully meeting the requirements of enterprise level software authorization management.

2.2 Client authentication mechanism

Client verification includes multiple security verification mechanisms for time verification. The system checks whether the current time is within the validity period specified in the license file to ensure that the license is used within the validity period. To prevent time tampering attacks, the system will encrypt and store the last verification time in the logs/busi/time.exe file. If a system time callback is detected, the license will be deemed invalid. In terms of hardware verification, the system obtains hardware information by reading the motherboard serial number and system serial number within the system, and strictly compares it with the hardware serial number recorded in the license to achieve hardware binding function.

2.3 Non Invasive Integration

The system adopts a non-invasive integrated design, which is a software extension method that does not require structural modifications to the original system, achieves functional enhancement and maintains the integrity of existing code through external mechanisms such as dependency injection or modular encapsulation [3], and automatically embeds license verification logic by providing pre compiled JAR packages (cs pcc license). Developers only need to add specified dependencies in the project pom.xml to complete the integration of the validation module without modifying business code. This JAR package is distributed through the Maven central repository, and only the<version>tag value needs to be adjusted during version upgrades. The specific version number is maintained by the technical team.

2.4 Code Confusion

To address the inherent security risks of Java bytecode being easily decompiled and to protect the core logic of license verification from reverse analysis and tampering, this system has implemented deep code obfuscation and reinforcement for the Java verification engine (csc_ppc_license module). Code obfuscation is a detection avoidance technique that can be used to resist reverse engineering and reengineering projects, and protect software intellectual property rights. The confusion process of this system goes far beyond simple identifier renaming. It is a system protection system that automatically performs multiple transformations on bytecode during the compilation and construction phase [4]. The core measures include replacing all meaningful class names, method names, and field names with short and meaningless characters, eliminating the self interpretability of the code; Flatten the control flow, breaking the original code block structure, inserting opaque predicates and redundant jumps, greatly increasing the difficulty of manually analyzing the execution path; At the same time, all sensitive strings and configuration constants in the code are statically encrypted and only dynamically decrypted during runtime, effectively preventing rapid identification of key code through string searches. In addition, all debugging information, line number tables, and local variable tables have been removed, making it almost impossible for the decompiled code to correspond to the original source code. After this combination processing, even if the resulting JAR package is successfully converted by a decompilation tool, its output is difficult to read and understand, greatly increasing the cost and time for attackers to analyze system vulnerabilities, understand business logic, and ultimately find ways to bypass them. This constitutes a crucial first software protection barrier for the security of the entire license management system. The comprehensive application of these obfuscation techniques has been proven to significantly increase the difficulty of reverse engineering, serving as a proven and effective method in software protection [5].

3 SYSTEM IMPLEMENTATION

3.1 Infrastructure

This system adopts a three-tier architecture design, consisting of a license issuance service, a core verification engine, and a runtime protection module, forming a complete license management loop.
At the issuance service layer, the system has built a complete digital certificate management system. Create RSA key pairs using key generation tools, securely store private keys in a key repository for license signing, and export public keys for verification purposes. The issuance service is encapsulated as an independent application, providing a standardized license generation interface and supporting the configuration of key authorization parameters such as expiration date and hardware fingerprints. The core validation engine adopts a lightweight design and is integrated into the business system through dependency injection. The engine consists of three core verification units: the digital signature verification unit is responsible for checking the integrity of the license, the time verification unit ensures the timeliness of use, and the hardware fingerprint unit compares the actual hardware information of the system. Each verification unit adopts a plug-in design, supporting on-demand activation and policy adjustment. The runtime protection layer provides active safety protection capabilities. The time protection mechanism prevents tampering

through encrypted timestamps and regular verification, while hardware protection directly reads the underlying information of the system to ensure authenticity. Real time hardware verification data is uploaded to the network platform through technologies such as RFID and sensor networks, constructing a new centralized license management framework [6]. The system also includes automated deployment support, including version management, configuration updates, and anomaly monitoring.
This architecture design achieves functional decoupling and flexible expansion, while maintaining core verification capabilities, significantly improving system security and reliability through layered protection. The interaction between each layer through clearly defined interfaces ensures both functional integrity and avoids invasive impacts on the business system.

3.2 Migration and Security Enhancement of Core Verification Methods to C++

To thoroughly address the security risks of Java implementation being prone to decompilation, this system has undergone a critical refactoring of the license verification process: the complete logic of the core validation validation in the original Java validation class, com. unicom. microservice. csp. vcl. license. Customs License Manager, has been migrated from the Java language to the C++language implementation. The refactored C++code is compiled into a platform specific dynamic link library. At runtime, the Java layer uses JNI technology to call functions in this Native library to perform the final verification calculation, retaining only interface encapsulation and result processing functionality. By transforming the core algorithm from interpreted Java bytecode to compiled native machine code, this migration prevents attackers from directly obtaining the complete verification logic through decompiling the Java layer, cutting off the most direct reverse analysis path. This hybrid architecture of "Java interface scheduling and C++core computing" provides compile level security protection for the core verification function while maintaining the original integration method of the system, significantly improving the overall anti reverse capability of the system. It is an important architectural evolution in this security optimization.

3.3 Engineering Composition

The system consists of two core projects:
(1) Csc_scense_gen is a license generator:
Provide executable JAR packages and REST interfaces (POST/license/generateLiceLicense) to receive JSON requests containing parameters such as subject (project name), privateAlias (private key alias), licensePath (generation path), etc;
(2) Csc_ppc_scene is the verification module:
The generated license.lic and public key library publicCerts.keyhole need to be placed in the resources directory, and injected into the business system through Maven dependencies to automatically execute multi-layer verification logic including signature verification and hardware information comparison [7].

4 TESTING AND VERIFICATION

4.1 License Generation Test

When testing the license generation function, the first step is to execute the csc_scense_gen project, call the/license/generateLiceLicense interface, and pass in a JSON request containing complete parameters (such as subject, privateAlias, expiryTime, etc.). Verify that the generated license.lic file contains the correct authorization information, especially checking if the mainBoardSerial and systemSerial match the input parameters. Simultaneously test abnormal scenarios, such as whether the system can correctly return error prompts when entering invalid privateKeysStorePath or incorrect keyPass. Perform validation on each input structure, check the compatibility of the software package license, and consider the license used in each file of the software package (i.e. extracted license).

4.2 Time Callback Detection

When testing the time tamper proof mechanism, first ensure that the system is running normally and record the current time to the encrypted file logs/busi/time.exe. Then manually reverse the system time by more than 1 hour and observe the system behavior: the verification module should be able to decrypt the time anchor file through EncryptionUtil, detect time anomalies, and trigger authorization invalidation. The theoretical basis of this mechanism originates from the Byzantine fault tolerance problem in distributed systems: time callbacks provide incorrect information at certain time nodes, and when malicious nodes exist, the system must achieve consistency through assumptions such as time synchronization. As Lamport pointed out, "The necessity of clock synchronization may not seem intuitive, but it can be proven that solving the Byzantine Generals problem relies on this assumption or equivalent mechanism. [8]" At the same time, check whether the 1-hour timed task can correctly update the encrypted timestamp, and whether the authorization state automatically recovers after the time returns to normal.

4.3 POM Dependency Loading Test

When verifying Maven dependency integration, add the cs pvc license dependency (specifying the correct version number) in the pom.xml of the test project. Post deployment check:
(1) Is the license.lic and publicCerts.keystore in the resources directory loaded correctly;
(2) Whether the verification logic is automatically executed during system startup; 3) Can the system correctly throw an exception when intentionally providing invalid license files. When testing dependent version updates, the business system only needs to modify the version number to take effect.

5 FRONT END INTEGRATION

5.1 License Management

The front-end provides a license management console. After logging in, users can enter the enterprise name, product name, system serial number, motherboard serial number, expiration time, creation time, creator filter and query license records through the composite area at the top of the page. The query content will appear in the table below, and the download link for the license will also be provided. Users can manually input relevant information to create records and generate licenses through the New button, or perform deletion operations on this interface.

6 CONCLUSION

This article proposes a new license management system to address the time tampering vulnerability (local timing bypass verification), hardware binding deficiency, and high integration complexity of the TrueLicense framework. The system uses hardware serial number binding to solve security issues; Adopting non-invasive Maven dependency integration to reduce usage costs; Introduce obfuscation techniques such as control flow flattening and string encryption to enhance code level protection; Design a three-tier architecture (issuance service → verification engine → protection module) to achieve module decoupling; Transforming the core code for license verification from Java to C++implementation, significantly improving anti reverse engineering capabilities through native compilation and JNI calls; The reliability of the solution in time tampering defense and distributed deployment has been verified through REST interface and automated testing; A call back detection mechanism based on encrypted time anchor is proposed to effectively defend against system time tampering attacks.

COMPETING INTERESTS

The authors have no relevant financial or non-financial interests to disclose.

REFERENCES

[1] Zhang X, Li R, Wang Q, et al. Time-manipulation Attack: Breaking Fairness against Proof of Authority Aura. Proceedings of the ACM Web Conference, 2022: 2076–2086.
[2] Rúa E A, Salomón F J G, Pérez-Freire L. DEMO: Gradiant Asymmetric Encryption and verification systems based on handwritten signature. CCS'13: 2013 ACM SIGSAC Conference on Computer and Communications Security, 2013.
[3] ISO/IEC/IEEE 26515: 2018(E): ISO/IEC/IEEE International Standard - Systems and software engineering? Developing information for users in an agile environment, 2018.
[4] Zheng J, Tong Z, Wu G, et al. Code Confusion confrontation method based on feature comparison. 2021 International Conference on Intelligent Computing, Automation and Systems (ICICAS), 2019: 61–65.
[5] Collberg C, Thomborson C, Low D. A Taxonomy of Obfuscating Transformations. Technical Report, 1997. https://researchspace.auckland.ac.nz/handle/2292/3491.
[6] Yu Y, Huang W, Zhang C. A New Centralized License Management Framework based on Internet of Things (IoT-CLMF). SPML 2024: 2024 7th International Conference on Signal Processing and Machine Learning, 2024: 328–334.
[7] Permit Implementation Principle and Operation Manual. Tencent Docs. 2023. https://docs.qq.com/doc/DT1R5ZUpteFhLa1FD.
[8] Lamport L, Shostak R, Pease M. The Byzantine Generals problem. ACM Transactions on Programming Languages and Systems, 1982, 4(3): 382–401.