Design and Implementation of a Cryptographic Hardware-Bound Licensing Framework Using Machine Fingerprinting and Digital Signatures

Darryl Rizqi Ramadhan - 18223084
Program Studi Sistem dan Teknologi Informasi
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: rizqidarryl@gmail.com , 18223084@std.stei.itb.ac.id

Abstract — Software piracy and license abuse remain critical challenges in commercial software distribution. Traditional licensing mechanisms relying solely on product keys are highly vulnerable to License Copy Attacks, where a valid license is transferred and reused across unauthorized machines. This paper proposes the design and implementation of a Hardware-Bound Licensing Framework that cryptographically ties a software license to the physical identity of the user's machine. The framework employs RSA-2048 for digital signature generation and verification, and SHA-256 for machine fingerprint derivation. Unlike conventional rigid binding systems, this framework introduces a Weighted Similarity Evaluation mechanism that provides configurable fault tolerance for minor hardware changes,such as OS reinstallation or RAM upgrades,while effectively preventing cross-machine piracy. The license structure is stored in a JSON format containing cryptographic fields and hardware weight parameters. Experimental results demonstrate that the system successfully grants access to legitimate hardware configurations while consistently rejecting license copy attacks and tampered license files. The framework represents a practical, cryptographically sound approach to software IP protection that balances security with user experience.

Keywords — *Hardware-Bound Licensing; Machine Fingerprinting; Digital Signatures; RSA-2048; SHA-256; Fault Tolerance; Weighted Similarity; Software Protection*

I. Introduction

A.Background

In the modern software industry, Intellectual Property (IP) protection is a fundamental concern. Commercial software developers invest considerable resources in building and maintaining their products, and unauthorized redistribution directly undermines this investment. One of the most pervasive threats is the License Sharing or License Copying attack, where a legitimate license is duplicated and used simultaneously on multiple machines beyond the authorized scope.

Many existing licensing frameworks mitigate this by binding a license to a specific machine's hardware fingerprint. However, strict hardware-matching systems create significant usability problems: any hardware component change,even a minor upgrade such as replacing a storage drive or reinstalling the operating system may render the license invalid, forcing users through cumbersome reactivation processes. This creates friction and erodes user trust.

This paper addresses this tension by proposing a Hardware-Bound Licensing Framework that provides strong cryptographic guarantees while accommodating common, legitimate hardware modifications through a Weighted Similarity Evaluation model. The framework leverages industry-standard RSA-2048 asymmetric cryptography for license signing and verification, and SHA-256 for one-way fingerprint derivation, ensuring that license integrity can be verified without exposing raw hardware identifiers.

B. Problem Formulation

This paper addresses the following research problems:

- How to design a license scheme that is cryptographically secure against illegal manipulation, tampering, and unauthorized reuse?
- How to implement a hardware detection system that provides fault tolerance for minor legitimate component changes while maintaining robust protection against license copying across different machines?
- How to determine an optimal weight distribution for hardware components that accurately reflects their uniqueness and replaceability in the threat model?

II. Literature Review And System Design

A. Asymmetric Cryptography and Hashing

Asymmetric cryptography utilizes a mathematically related key pair: a private key for signing or decryption operations, and a corresponding public key for verification or encryption. This paradigm eliminates the need for a shared secret between communicating parties, a fundamental limitation of symmetric systems [1].

RSA (Rivest-Shamir-Adleman) is one of the most widely deployed asymmetric cryptographic algorithms. RSA-2048, employing a 2048-bit key length, is currently the industry-recommended minimum for secure applications and

is expected to remain computationally infeasible to break with classical computers for the foreseeable future [2]. In the context of digital signatures, the RSA-PSS scheme is used: the signer computes a hash of the message, encrypts it with the private key to produce the signature, and the verifier decrypts it with the public key to confirm authenticity and integrity.

SHA-256, a member of the SHA-2 family standardized by NIST, produces a deterministic 256-bit digest from any input. Its collision resistance and one-way properties make it ideal for fingerprinting: the same hardware configuration will always produce the same fingerprint hash, but the hash cannot be reverse-engineered to recover the original hardware identifiers [3].

B. Machine Fingerprinting

Machine fingerprinting refers to the collection and aggregation of unique system attributes to produce a stable identifier for a specific hardware configuration. Common attributes include motherboard serial numbers, CPU identifiers, persistent disk serials, and operating-system-level GUIDs. When these identifiers are combined and hashed, the resulting fingerprint is specific enough to differentiate between machines, while the use of a one-way hash function protects user privacy by preventing the recovery of raw hardware data from the fingerprint alone [4].

A well-designed fingerprinting scheme must balance two competing requirements: stability (the fingerprint should remain consistent across minor changes) and uniqueness (the fingerprint should differ significantly between distinct machines). Weighted component models address this by assigning differential importance to hardware components based on how unique and how rarely they change [5].

C. Threat Model

The system is designed and evaluated against the following threat model. Threats within scope include:

- License Copy Attack: Copying a valid license file from Machine A to Machine B with different hardware.
- License Tampering: Manually modifying the contents of the license file to alter permissions, expiry, or hardware parameters.
- Fake License Creation: Generating a syntactically valid license file without possessing the signing private key.
- OS Reinstallation Attack: Attempting to exploit the fault-tolerance window by performing a targeted OS reinstall to shift the fingerprint.

Threats explicitly out of scope for this framework include Kernel-Level Hardware Spoofing, Binary Patching or code injection to bypass the license verification routine, and full Reverse Engineering of the application binary. These attack classes require operating system-level privilege escalation or sophisticated binary analysis and represent a separate layer of protection (e.g., code obfuscation, anti-tamper tooling) not addressed here.

D. System Architecture

The framework is composed of two distinct operational components: a License Issuer (server-side) and a License Verifier (client-side).

The License Issuer is operated exclusively by the software vendor. It receives a hardware fingerprint from the client during the activation process, constructs the license data structure, signs it with the vendor's RSA-2048 private key, and delivers the signed license file to the client.

The License Verifier is embedded within the client application. On each launch, it performs the following sequential verification pipeline: (1) Hardware Information Collection, (2) Real-Time Fingerprint Generation via SHA-256, (3) License File Loading and Parsing, (4) RSA Digital Signature Verification using the embedded public key, and (5) Weighted Similarity Evaluation comparing the current

fingerprint components against those stored in the license. Access is granted only if both the signature verification succeeds and the similarity score meets or exceeds the configured threshold.

The private key never leaves the vendor's infrastructure. The public key is embedded in the compiled client binary. This asymmetric distribution ensures that even if the license file and public key are both accessible to an attacker, it is computationally infeasible to forge a valid license without the private key.

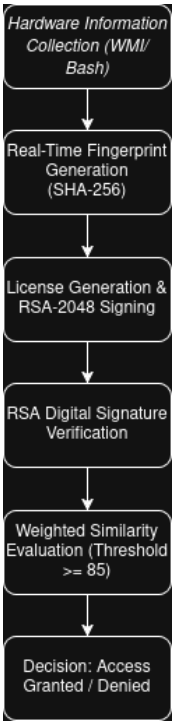

FIG 1.System Architecture and Sequential Verification Pipeline.

III. IMPLEMENTATIONS

A. Machine Fingerprint Extraction

The fingerprint extraction module queries the operating system for four hardware identifiers, designed to be cross-platform supporting both Windows and Linux environments. The extraction relies on native OS command-line tools to eliminate heavy third-party dependencies:

- Motherboard ID: On Windows, this is retrieved via PowerShell using Get-CimInstance Win32_BaseBoard. On Linux, it reads from DMI tables at /sys/class/dmi/id/board_serial or product_uuid. This represents the most unique and least frequently changed hardware component.
- CPU Info: On Windows, it is retrieved via Win32_Processor.ProcessorId. On Linux, the module parses /proc/cpuinfo. This is a hardware-burned identifier specific to the CPU model.
- Disk Serial: On Windows, it uses Win32_DiskDrive.SerialNumber. On Linux, it utilizes the lsblk -no SERIAL command. This value changes upon primary disk replacement.
- Machine GUID: On Windows, it is extracted from the Registry at HKLM\SOFTWARE\Microsoft\Cryptography\MachineGuid. On Linux, it reads from /etc/machine-id. This is a software-generated identifier that changes upon OS reinstallation.

These four raw values are concatenated in a deterministic order and passed to SHA-256 to produce the 256-bit machine fingerprint. The use of SHA-256 ensures that the raw hardware identifiers are never stored or transmitted in plaintext, preserving user privacy. The pseudocode for fingerprint generation is as follows :

raw = concat(motherboard_id, cpu_id, disk_serial, machine_guid) fingerprint = SHA256(raw)

B. License Data Structure

The license is stored as a structured JSON file containing two main sections: the license identity block and the hardware policy block.

The license identity block contains: the license owner's identifier, the licensed product name and version, the issue date, the expiry date, and the machine fingerprint captured at the time of issuance. The hardware policy block defines the weight assigned to each hardware component for similarity evaluation, and the minimum score threshold required for access.

Based on empirical analysis of hardware uniqueness and replacement frequency, the following weight distribution was selected:

Hardware Component	Assigned Weight
Motherboard ID	45 points
CPU Info	30 points
Disk Serial	10 points
Machine GUID	15 points
Total (Full Match)	**100 points**

TABLE I. Hardware Component Weight Distribution

This weight distribution reflects the following rationale: the Motherboard ID receives the highest weight (45 points) because it is the most architecturally central component. The CPU receives 30 points as it is similarly persistent. The Machine GUID receives 15 points, and the Disk Serial receives 10 points. This lower weight for the disk accommodates common user maintenance actions; upgrading storage should not penalize a legitimate user, provided the core computing components remain unchanged. The access threshold is set at 85 points.

The access threshold is set at 85 points. This threshold was chosen to allow a single minor change (such as OS reinstallation, losing 15 points) while requiring that the most critical components,Motherboard, CPU, and Disk,remain present.

To further illustrate the practical implementation of this data structure, the following snippet demonstrates the generated signed_license.json payload. The cryptographic hashes are truncated for readability. This JSON structure ensures that both the hardware policy and the identity block are tightly coupled and protected by the base64-encoded RSA-PSS signature.

```
{
  "license": {
    "license_id": "LIC-88CBE6",
    "owner": "Darryl",
    "product": "ProtectedApp",
    "issued_at": "2026-06-17",
    "expiry": "2027-06-17",
    "fingerprint": {
      "motherboard_hash": "3e7c3c1468c8a09f...",
      "cpu_hash": "2577d0c9a70af3a5...",
      "disk_hash": "f181204cf0f01f1b...",
      "machine_guid_hash": "d0e8ff16e3674974..."
    },
    "fingerprint_policy": {
      "motherboard_weight": 45,
      "cpu_weight": 30,
      "disk_weight": 10,
      "machine_guid_weight": 15,
      "threshold": 85
    }
  },
  "signature":
"RcjLCOk4LW+j3/UheEMCiu7cfK338B/gqD..."
```

```
}
```

C. Digital Signature Scheme

The license signing process employs the RSA-PSS signature scheme. The license issuer performs the following operations :

- Serialize the license data (excluding the signature field itself) into a canonical string representation.
- Compute the SHA-256 digest of the serialized license data.
- Encrypt the digest using the vendor's RSA-2048 private key to produce the digital signature.
- Embed the Base64-encoded signature into the license JSON file.

The License Verifier performs the inverse: it extracts the signature field, reconstructs the canonical serialization of the license data, recomputes the SHA-256 digest, and uses the embedded RSA-2048 public key to verify the signature. If verification fails, the system rejects the license immediately, regardless of the similarity score. This provides a strong cryptographic guarantee of integrity and authenticity: any modification to the license data,however minor,invalidates the signature.

D. Weighted Similarity Evaluation Algorithm

After a successful signature verification, the Verifier proceeds to the Weighted Similarity Evaluation. The algorithm compares each hardware component extracted from the current machine against the corresponding component stored in the license. For each matching component, the assigned weight is added to a running score. The pseudocode is:

```
score = 0
for each component c in {MB, CPU, Disk,
GUID}:
  if current[c] == license[c]:
    score += weight[c]
return score >= THRESHOLD (85)
```

The comparison is not performed on the raw hardware identifier strings, but rather on the individual SHA-256 hashes of each component (e.g., comparing the current motherboard_hash against the licensed motherboard_hash). This preserves user privacy because the raw serial numbers are never stored in the license file, while still enabling per-component scoring rather than a binary all-or-nothing aggregate match.

E. Implementation Environment

The prototype implementation was developed in Python 3.10+ with a cross-platform architecture supporting both Windows 11 and Linux distributions. Hardware queries were performed using native OS commands (PowerShell cmdlets for Windows and bash/shell commands for Linux) executed via Python's built-in subprocess module, ensuring fallback mechanisms are in place if permission errors occur. Cryptographic operations were implemented using the cryptography library (version 45.x), which provides OpenSSL-backed RSA-2048 and SHA-256 primitives. License files are stored in JSON format and accessed via Python's standard json module. The signing infrastructure (key generation, license issuance) was developed as a separate command-line tool to simulate vendor-side operations, ensuring clear separation between the issuer and verifier components.

IV. Testing And Evaluation

The experimental scenarios were executed to validate the verifier's response against various hardware configurations and attack vectors. **Due to hardware availability constraints, component replacement scenarios were emulated by modifying the corresponding hardware identifiers in the testing environment.** The tests were run using an automated testing script to efficiently execute these simulations and attacks. A summary screenshot of this automated execution is provided in the Appendix. For a comprehensive, step-by-step demonstration of each test case, please refer to the video presentation linked at the end of this paper. The evaluation results, based on the configured weight distribution (Threshold: 85), are as follows:

1. E1 (Valid License): The signed license file was executed on the authorized client machine without any hardware modifications. The verifier successfully calculated a full score of 100/85. Access Granted.
2. E2 (License Copy Attack): The license from PC-A was executed on PC-B (simulating entirely different hardware identifiers for all four components). The similarity score dropped to 0/85. Access Denied.
3. E3 (License Tampering): The contents of the signed_license.json file were modified (e.g., extending the expiry date). The verifier rejected the license immediately due to an RSA-PSS signature verification failure, completely bypassing the hardware score check. Access Denied.
4. E4 (OS Reinstallation): Simulated a Machine GUID change due to an OS reinstallation. The machine lost 15 points, but the total score of 85 exactly met the required threshold. Access Granted (License Remains Valid).
5. E5 (SSD Replacement): Simulated a disk replacement causing the primary disk serial to change. The score lost 10 points, making the final score 90/85. Since 90 comfortably exceeds the threshold of 85, Access Granted (License Remains Valid). (Evaluation Note: This outcome successfully highlights the fault tolerance of the framework, allowing legitimate users to perform common storage upgrades without triggering false-positive piracy detections).
6. E6 (Motherboard Replacement): Simulated a Motherboard UUID replacement, removing 45 points of the most architecturally critical component. The score plummeted to 55/85, which is strictly below the threshold. Access Denied.

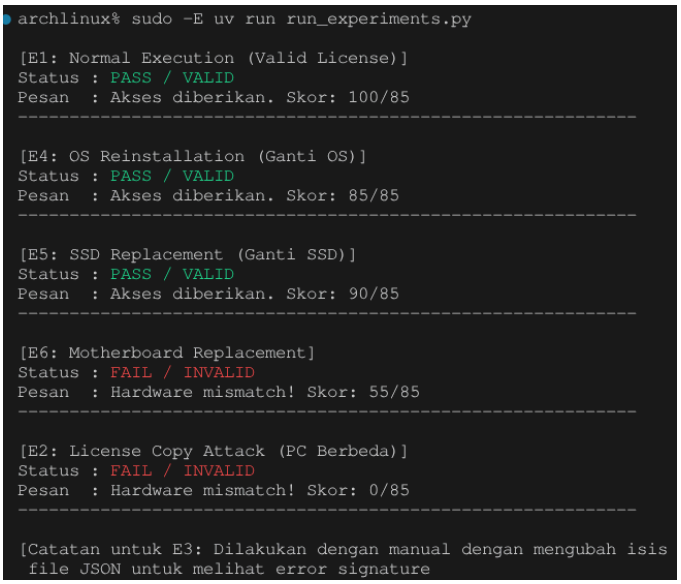

FIG 2.Experimental Evaluation of License Verification Scenarios

V. Discussion

A. Security Analysis

The system's security is grounded in the cryptographic strength of RSA-2048 and SHA-256. As of the time of writing, no practical attacks against either algorithm exist for the key sizes used. The confidentiality of the vendor's private key is the single most critical security assumption; if the private key is compromised, an adversary can generate valid licenses for any hardware configuration. Mitigating this risk falls within the key management domain and is outside the scope of this framework, but standard practices,such as Hardware Security Module (HSM) storage of the private key are recommended.

The Weighted Similarity Evaluation introduces a deliberate trade-off: by allowing partial hardware matches, the system creates a narrow attack surface where an adversary who possesses knowledge of the licensed machine's hardware identifiers might attempt to replicate enough components to exceed the threshold. This risk is mitigated by three factors: (1) the SHA-256 hashing of raw identifiers in the aggregate fingerprint prevents direct leakage of hardware data, (2) physical hardware spoofing at the WMI level requires administrator privilege and specialized tools, and (3) the threshold of 85 requires the attacker to replicate at least three of the four hardware components, including the motherboard.

B. Fault Tolerance Analysis

The weighted model provides predictable and configurable fault tolerance. A system administrator or vendor can tune the threshold and component weights to match their threat model and user base. For example, a software product deployed in enterprise environments with frequent OS reimaging might lower the Machine GUID weight or raise the threshold to only 70, while a high-security product might require a full 100-point match. The current default configuration (threshold: 85) strikes a balance appropriate for general commercial software.

C. Comparison with Existing Approaches

Traditional product key systems provide no hardware binding whatsoever and are trivially defeated by key sharing. Online activation systems,such as those used by major commercial software vendors,solve the copying problem but introduce a dependency on network connectivity and require server-side infrastructure. The proposed framework provides hardware binding without requiring online verification, making it suitable for deployment in air-gapped or offline environments. However, it does not provide the revocation capability that online systems offer.

Hardware-locked systems that use cryptographic tokens or dongles provide strong protection but impose hardware costs and logistics on both the vendor and the user. The proposed framework achieves comparable binding strength through software-only means, eliminating hardware cost entirely.

D. Limitations

The framework has several known limitations. First, it is susceptible to kernel-level hardware identifier spoofing, which is an out-of-scope attack class but represents a ceiling on the system's protection level. Second, the fault-tolerance window, while necessary for usability, introduces a deterministic score that an informed attacker with partial hardware access could potentially exploit. Third, the framework does not currently implement license revocation or expiry enforcement beyond the expiry date field in the license data,a revocation mechanism would require periodic online validation, which contradicts the offline design goal.

VI. Conclusion

This paper presented the design and implementation of a Hardware-Bound Licensing Framework that combines RSA-2048 digital signatures with SHA-256 machine fingerprinting and a Weighted Similarity Evaluation mechanism. The framework provides cryptographic guarantees of license integrity and authenticity while accommodating common, legitimate hardware changes through a configurable fault-tolerance model.

The weight distribution assigns 45 points to the Motherboard ID, 30 to CPU, 10 to Disk Serial, and 15 to Machine GUID, with an access threshold of 85 points. This configuration successfully allows minor system changes (such as OS reinstallation or storage drive upgrades) while strictly requiring the presence of the most critical core hardware components to prevent unauthorized license copying.

The experimental evaluation has been completed and successfully validated the system's effectiveness across six distinct scenarios, including valid operations, license copy attacks, license tampering, and legitimate hardware upgrades. The test results conclusively demonstrate the framework's ability to balance robust security measures with an accommodating user experience.

Future work may explore extending the framework with secure online revocation capabilities, additional hardware components (e.g., network interface MAC address), and anti-tamper measures to protect the verifier binary itself.

Video Link at Youtube

https://youtu.be/_AXzb4CytCM

Link Github

https://github.com/RylRizqi16/Hardware-Bound-Licensing

Acknowledgment

The author would like to thank the faculty of the Department of Information Systems and Technology, School of Electrical Engineering and Informatics, Institut Teknologi Bandung, for the guidance and framework provided in the II4021 Cryptography course.

References

[1] W. Diffie and M. E. Hellman, "New directions in cryptography," IEEE Transactions on Information Theory, vol. 22, no. 6, pp. 644–654, Nov. 1976.

[2] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," Communications of the ACM, vol. 21, no. 2, pp. 120–126, Feb. 1978.

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

[4] V. Kumar and K. Paul, "Device fingerprinting for cyber-physical systems: A survey," ACM Comput. Surv., vol. 55, no. 14s, Art. 302, pp. 1–41, Jul. 2023.

[5] A. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna, "Cookieless monster: Exploring the ecosystem of web-based device fingerprinting," in Proc. IEEE Symposium on Security and Privacy, San Francisco, CA, 2013, pp. 541–555.

[6] R. Anderson, Security Engineering: A Guide to Building Dependable Distributed Systems, 3rd ed. Hoboken, NJ: Wiley, 2020.

[7] B. Schneier, Applied Cryptography: Protocols, Algorithms, and Source Code in C, 2nd ed. New York, NY: Wiley, 1996.

[8] M. Bellare and P. Rogaway, "The exact security of digital signatures—How to sign with RSA and Rabin," in Advances in Cryptology — EUROCRYPT '96, U. Maurer, Ed., Lecture Notes in Computer Science, vol. 1070. Berlin, Germany: Springer, 1996, pp. 399–416.

[9] K. Moriarty, Ed., B. Kaliski, J. Jonsson, and A. Rusch, PKCS #1: RSA Cryptography Specifications Version 2.2, IETF RFC 8017, Nov. 2016.

[10] R. Munir, "Algoritma RSA," Bahan Kuliah II4031 Kriptografi dan Koding, Program Studi Sistem dan Teknologi Informasi, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung, 2024. [Online]. Tersedia: https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi-dan-Koding/ 2023-2024/11-Algoritma-RSA-2024.pdf

[11] R. Munir, "Tanda-tangan Digital," Bahan Kuliah II4031 Kriptografi dan Koding, Program Studi Sistem dan Teknologi Informasi, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung, 2021. [Online]. Tersedia: https://informatika.stei.itb.ac.id/~rinaldi.munir/ Kriptografi-dan-Koding/2020-2021/Tanda-tangan-digital-2021.pdf

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, 19 Juni 2026

Darryl Rizqi Ramadhan

18223084