Key-Gated Open Source:
A Practical Approach to Transparent Source Protection

Abstract

We present *key-gated open source*, a software distribution pattern that combines the transparency benefits of open source licensing with practical protection against wholesale code reuse. Unlike source-available licenses that impose legal restrictions, or obfuscation that degrades code quality, key-gated open source encrypts essential modules while maintaining full OSI-compliant licensing. The approach requires no legal innovation—it leverages standard cryptographic primitives and runtime module loading to create a practical barrier to competition while preserving auditability, community engagement, and open source ecosystem benefits. We describe the implementation, analyze the security model, and discuss appropriate use cases.

1. Introduction

Open source software faces a fundamental tension: the transparency that enables trust, security auditing, and community contribution also enables competitors to deploy functionally identical services with minimal investment. Traditional responses to this tension fall into three categories:

1. **Proprietary licensing**: Withholding source code entirely
2. **Source-available licensing**: Publishing source with usage restrictions (BSL, FSL, SSPL)
3. **Open core**: Open sourcing commodity components while keeping differentiating features proprietary

Each approach involves trade-offs. Proprietary licensing sacrifices transparency. Source-available licenses sacrifice OSI compliance and may affect ecosystem benefits (CI/CD pricing, community perception). Open core creates maintenance burden and unclear boundaries.

We propose a fourth approach: **key-gated open source**. The complete source code is published under a standard open source license (e.g., AGPL-3.0), but essential modules are encrypted. Without the decryption key, the software will not execute. Competitors must either obtain the key (kept secret by the maintainer) or re-implement the encrypted modules from scratch.

2. Threat Model

Key-gated open source addresses a specific threat: **low-effort cloning** by competitors who would otherwise fork, rebrand, and deploy with minimal modification.

It does *not* address: determined reverse engineering (always possible given sufficient resources), legal disputes over licensing terms, or nation-state adversaries.

The security goal is **raising the effort bar**, not preventing all competition. For niche SaaS applications, the effort required to re-implement encrypted modules may exceed the effort of building from scratch,

eliminating the advantage of cloning.

3. Implementation

3.1 Cryptographic Scheme

We employ hybrid encryption combining RSA and AES:

1. Generate RSA-4096 keypair (public key published, private key secret)
2. For each protected module:

• Generate random AES-256 key and 96-bit IV

• Encrypt module source with AES-256-GCM (authenticated encryption)

• Encrypt AES key with RSA-OAEP using public key

• Package as: [key_length][encrypted_aes_key][iv][auth_tag][ciphertext]

3.2 Runtime Loading

Protected modules are loaded at runtime via a synchronous decryption and evaluation mechanism. The loader checks cache (decrypt once per process), retrieves the private key from environment or file, decrypts the ciphertext, evaluates the decrypted JavaScript in module context, and returns exports.

3.3 Key Distribution

Private keys are distributed through standard secrets management: file in project root for development (gitignored), environment variable for CI/CD (GitHub Secrets), and cloud secrets manager for production (AWS Secrets Manager, HashiCorp Vault).

4. License Compliance

Key-gated open source is compatible with OSI-approved licenses including copyleft licenses like AGPL-3.0. Encrypted blobs satisfy Open Source Definition requirements: the source code (including encrypted files) is freely redistributable, anyone can fork and modify, and the license imposes no usage restrictions.

The encryption key is operational data, analogous to API credentials or database passwords—necessary for operation but not part of the source code distribution. This interpretation aligns with common practice: many open source projects require external services that are not themselves open source.

5. Security Analysis

5.1 Cryptographic Strength

The scheme uses well-established primitives: RSA-4096 with OAEP padding, AES-256-GCM (authenticated encryption), and cryptographically secure random IV per encryption. Breaking the encryption without the key is computationally infeasible.

5.2 Practical Attacks

Key compromise: If the private key leaks, all encrypted modules are exposed. Mitigation: standard key management hygiene, rotation capability.

Reverse engineering: A determined attacker can analyze wrapper modules, understand the expected interface, and re-implement. This is by design—the goal is effort, not impossibility.

Runtime extraction: An attacker with process access could extract decrypted modules from memory. Mitigation: this requires deployment access, which implies trust.

5.3 Comparison with Alternatives

Approach	Transparency	OSI Compliant	Clone Protection	Effort to Bypass
Proprietary	None	No	Legal	Reverse engineering
Source-available	Full	No	Legal	License violation
Obfuscation	Degraded	Yes	None	De-obfuscation
Key-gated	Full	Yes	Practical	Re-implementation

6. Appropriate Use Cases

Key-gated open source is appropriate when:

1. **Transparency is required** for trust, auditing, or regulatory compliance
2. **OSI compliance matters** for ecosystem benefits or community perception
3. **The protected code is non-trivial** to re-implement
4. **The threat is casual cloning**, not determined adversaries

Examples: niche B2B SaaS, regulated industry software, projects with complex domain logic.

It is *not* appropriate when perfect protection is required (use proprietary), the protected code is trivial (protection provides no value), or legal enforcement is preferred (use source-available licenses).

7. Combining with Copyleft

Key-gated open source pairs effectively with strong copyleft licenses like AGPL-3.0. If a competitor re-implements the encrypted modules and distributes or deploys as a network service, they must publish their implementation under the same license.

This creates a strategic choice for competitors: re-implement and publish (contributing to the ecosystem), build entirely from scratch (no code reuse benefit), or negotiate a commercial license with the maintainer.

8. Conclusion

Key-gated open source provides a middle path between full openness and proprietary protection. By separating legal rights (preserved via open source licensing) from practical executability (restricted via encryption), it enables maintainers to balance transparency with business sustainability.

The approach requires no legal innovation, uses standard cryptographic tools, and integrates with existing development workflows. For appropriate use cases—niche applications where re-implementation effort provides sufficient deterrence—it offers a practical solution to the open source sustainability challenge.

References

[1] Open Source Initiative. "The Open Source Definition." https://opensource.org/osd
[2] MariaDB Corporation. "Business Source License 1.1." https://mariadb.com/bsl11/
[3] Sentry. "Functional Source License." https://fsl.software/
[4] Keygen. "Fair Core License." https://fcl.dev/
[5] NIST. "Recommendation for Block Cipher Modes of Operation: GCM." SP 800-38D.