paozip for Ruby

Ruby ソースコードを暗号化して、大切なコードを守るツール

ユーザーズマニュアル

バージョン 2.0.0

2026年2月

有限会社 パオ・アット・オフィス

https://www.pao.ac/

目次

1. はじめに

1.1 paozip for Ruby とは

「大切な Ruby コードを、暗号化して配布できたら…」

そんな願いを叶えるのが paozip for Ruby です。

あなたが心をこめて書いた Ruby のソースコード —
独自のアルゴリズム、長年のノウハウが詰まったビジネスロジック、企業秘密のAPIキー処理 —
それらを暗号化して、安全に配布できます。

しかも、暗号化されたコードはそのまま動きます。利用者はソースコードの中身を見ることなく、プログラムを実行できるのです。

1.2 特長

- Pure Ruby 実装 — C 拡張不要。どこでも動きます
- Windows / macOS / Linux 全環境対応
- require フック — 暗号化ファイルを透過的に require できる
- Rails / Sinatra 対応 — 普段のフレームワークでそのまま使える
- PHP版・Python版と同じ暗号化エンジン — zencode による堅牢な暗号化
- Gem としてインストール — いつもの gem install でOK

1.3 仕組み（ラッパー方式）

paozip for Ruby は ラッパー方式（方式A） を採用しています。

```
app.rb ─
  │
  ├── require 'paozip'
  ├── Paozip.install_importer      ← require
  │
  └── require 'secret_logic'       ← secret_logic.rbe
        │
        └── (.rbe)
```

ポイント：

- app.rb（ラッパー）は暗号化しません。これが「入口」になります
- secret_logic.rbe（暗号化ファイル）は require するだけで自動復号
- 利用者からは、普通に Ruby を実行しているのと変わりません

1.4 PHP/Python版との違い

項目	PHP版	Python版	Ruby版
暗号化方式	同一（zencode）	同一（zencode）	同一（zencode）
実行方式	透過実行	ラッパー方式	ラッパー方式
C 拡張	あり（.so）	あり（.so）	なし（Pure Ruby）
run コマンド	なし	あり	あり
ファイル拡張子	.php	.pye	.rbe

Ruby版はPython版と同じラッパー方式ですが、Pure Ruby で実装されているため、C コンパイラなしでインストールできるのが大きなメリットです。

2. 動作環境

2.1 対応 OS

OS	備考
Windows 10/11	RubyInstaller 推奨
macOS 12+	システム Ruby または rbenv/rvm
Ubuntu 20.04+	apt / rbenv / rvm
CentOS/RHEL 8+	dnf / rbenv / rvm
その他 Linux	Ruby 3.0+ が動作すれば OK

2.2 対応 Ruby バージョン

バージョン	対応状況
Ruby 3.0	OK
Ruby 3.1	OK
Ruby 3.2	OK（推奨）
Ruby 3.3+	OK

> Ruby 2.7 以前は非対応です。

2.3 対応フレームワーク

フレームワーク	対応状況
Ruby on Rails 7+	OK
Sinatra	OK
CLI スクリプト	OK
Rake タスク	OK

3. セットアップ

3.1 Docker で試す（おすすめ）

まずは Docker で手軽に試してみましょう。面倒なインストールは一切不要です。

```
cd paozip-ruby-demo
docker-compose build
docker-compose run --rm paozip-demo
```

コンテナの中に入ったら：

```
#
cat lib/secret.rb              # ← 

#
cat lib/secret.rbe             # ← 

#
ruby app.rb                    # ← 

# paozip run
paozip run lib/secret.rbe      # ← OK
```

暗号化前は丸見えだったコードが、暗号化後はまったく読めない。でも動く。 — これが paozip の魅力です。

3.2 Gem としてインストール

```
gem install paozip
```

これだけです。Pure Ruby なので、追加のビルドツールは不要です。

3.3 ソースからビルド

納品された paozip-ruby ディレクトリを使って：

```
cd paozip-ruby
gem build paozip.gemspec
gem install paozip-*.gem
```

Bundler を使う場合は、Gemfile に追加：

```
gem 'paozip', path: '/path/to/paozip-ruby'
```

```
bundle install
```

3.4 インストールの確認

```
# CLI
paozip version
# → paozip for Ruby 2.0.0 ...

# Ruby API
ruby -e "require 'paozip'; puts Paozip.version_string"
```

4. クイックスタート

3分で暗号化を体験しましょう。

4.1 暗号化してみよう

まず、暗号化キーを作成します：

```
echo "a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6" > .paozip_key
```

次に、秘密のコードを作ります：

```
# secret_logic.rb
module SecretLogic
  def self.calculate(x)
    #
    (x * 31337 + 42) % 65536
  end

  def self.api_key
    "sk-very-secret-key-12345"
  end
end
```

暗号化！

```
paozip encrypt secret_logic.rb
#  secret_logic.rb → secret_logic.rbe

#
paozip check secret_logic.rbe
# → secret_logic.rbe: Encrypted (paozip format)
```

4.2 ラッパーを作ってみよう

```
# app.rb —
require 'paozip'
Paozip.install_importer

#
require_relative 'secret_logic'

#
puts SecretLogic.calculate(100)
puts SecretLogic.api_key
```

4.3 実行してみよう

```
#  .rb .rbe
rm secret_logic.rb

#
ruby app.rb
# → 4272
# → sk-very-secret-key-12345
```

暗号化されたファイルだけで、ちゃんと動きました。

4.4 paozip run で直接実行

ラッパーを書かずに、暗号化ファイルを直接実行することもできます：

```
paozip run secret_logic.rbe
```

5. CLI コマンドリファレンス

5.1 encrypt（暗号化）

```
paozip encrypt <file.rb> [-o output]
```

オプション	説明
<file.rb>	暗号化する Ruby ファイル
-o output	出力ファイルパス（省略時: .rbe に変換）

例：

```
paozip encrypt lib/secret.rb
# → lib/secret.rbe

paozip encrypt lib/secret.rb -o encrypted/secret.rbe
# → encrypted/secret.rbe
```

5.2 decrypt（復号）

```
paozip decrypt <file.rbe> [-o output]
```

オプション	説明
<file.rbe>	復号する暗号化ファイル
-o output	出力ファイルパス（省略時: .rb に変換）

例：

```
paozip decrypt lib/secret.rbe
# → lib/secret.rb

paozip decrypt lib/secret.rbe -o /tmp/secret.rb
```

> 復号には暗号化時と同じキーが必要です。

5.3 run（実行）

```
paozip run <file.rbe> [args...]
```

オプション	説明
<file.rbe>	実行する暗号化ファイル
args...	スクリプトに渡す引数

例：

```
paozip run script.rbe
paozip run script.rbe --config production --verbose
```

> ラッパーを作らずに、暗号化ファイルを直接実行できる便利コマンドです。

5.4 check（暗号化チェック）

```
paozip check <file>
```

例：

```
paozip check lib/secret.rbe
# → lib/secret.rbe: Encrypted (paozip format)

paozip check lib/plain.rb
# → lib/plain.rb: Not encrypted (plain text)
```

5.5 version（バージョン表示）

```
paozip version
# → paozip for Ruby 2.0.0 [TRIAL]
```

6. Ruby API リファレンス

```
require 'paozip'
```

6.1 暗号化・復号

```
#
encrypted = Paozip.encrypt(data, key = nil)

#
decrypted = Paozip.decrypt(data, key = nil)
```

key を省略すると、環境変数 PAOZIP_KEY または .paozip_key ファイルからキーを読み込みます。

6.2 ファイル操作

```
#  → .rbe
output_path = Paozip.encrypt_file(path, output_path = nil, key = nil)

#  →
decrypted = Paozip.decrypt_file(path, key = nil)

#  →
output_path = Paozip.decrypt_file_to(path, output_path = nil, key = nil)
```

6.3 チェック操作

```
#
Paozip.encrypted?(data)       # => true / false

#
Paozip.encrypted_file?(path)  # => true / false
```

6.4 製品情報

```
Paozip.licensed?          # => true / false
Paozip.get_license_email  # => "user@example.com"
Paozip.product_info       # => { product: ..., version: ..., ... }
Paozip.version_string     # => "paozip for Ruby 2.0.0 [TRIAL]"
```

7. require フックの使い方

paozip for Ruby の「キラー機能」です。暗号化ファイルを、普通の require と同じ感覚で読み込めます。

7.1 基本的な使い方

```
# app.rb
require 'paozip'
Paozip.install_importer    # ← 1

#  require
require 'secret_module'    # secret_module.rbe
```

たったこれだけ。install_importer を呼ぶだけで、Ruby の require メカニズムに paozip のフックが組み込まれます。

7.2 require_relative との組み合わせ

```
require 'paozip'
Paozip.install_importer

#  OK
require_relative 'lib/secret'  # lib/secret.rbe
```

7.3 実行フロー

1. require 'secret' が呼ばれる
2. Ruby 標準の require が secret.rb を探す
3. 見つからない場合、paozip フックが secret.rbe を探す
4. secret.rbe が見つかれば、復号してモジュールとして評価

> .rb と .rbe の両方が存在する場合、.rb が優先されます。暗号化後は必ず元の .rb ファイルを削除してください。

注意： Paozip.install_importer は必ず トップレベル で呼び出してください。if __FILE__ == $PROGRAM_NAME ブロックの中では動作しません。

8. 暗号化キーについて

8.1 キーの設定方法

暗号化キーは以下の優先順序で検索されます：

1. API 呼び出し時に直接指定 — Paozip.encrypt(data, "my_key")
2. 環境変数 — PAOZIP_KEY
3. キーファイル — .paozip_key（カレントディレクトリから親ディレクトリを順に検索）

8.2 キーファイルの作成

```
echo "a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6" > .paozip_key
```

環境変数で設定する場合：

```
# Linux / macOS
export PAOZIP_KEY="a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6"

# Windows
set PAOZIP_KEY=a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6
```

8.3 注意事項

・.paozip_key ファイルは 絶対に Git にコミットしないでください
・.gitignore に .paozip_key を追加してください
・暗号化と実行で必ず同じキーを使ってください — キーが違うと復号できません
・キーを紛失すると、暗号化されたファイルは復号できません — 大切に保管してください
・全環境（開発・ステージング・本番）で同じキーを使用してください

9. 各種環境での利用

9.1 Rails

Rails アプリケーションで使う場合は、config/boot.rb の先頭に追加します：

```
# config/boot.rb
require 'paozip'
Paozip.install_importer
```

```
#
paozip encrypt app/models/secret_model.rb
paozip encrypt app/services/billing_service.rb
rm app/models/secret_model.rb app/services/billing_service.rb

# Rails
rails server
```

Rails のオートロードとも問題なく動作します。require の仕組みにフックしているので、Rails が内部で使うファイル読み込みにも対応しています。

9.2 Sinatra

```
# app.rb
require 'paozip'
Paozip.install_importer

require 'sinatra'
require_relative 'lib/secret_logic'  # secret_logic.rbe

get '/' do
  SecretLogic.process(params)
end
```

```
paozip encrypt lib/secret_logic.rb
rm lib/secret_logic.rb
ruby app.rb
```

9.3 Bundler

Gemfile に paozip を追加：

```
source 'https://rubygems.org'

gem 'paozip', path: '/path/to/paozip-ruby'
gem 'rails'   #  gem
```

```
bundle install
```

9.4 Docker

```
FROM ruby:3.2

WORKDIR /app

# paozip
COPY paozip-ruby/ /tmp/paozip-ruby/
RUN cd /tmp/paozip-ruby && \
    gem build paozip.gemspec && \
    gem install paozip-*.gem && \
    rm -rf /tmp/paozip-ruby/

#
COPY . .

#
RUN paozip encrypt lib/secret.rb && rm lib/secret.rb

CMD ["ruby", "app.rb"]
```

ビルド時に暗号化し .rb を削除すれば、配布イメージにソースは含まれません。

10. トラブルシューティング

症状	対処法
Encryption key not found	.paozip_key ファイルを作成するか、PAOZIP_KEY 環境変数を設定
Invalid paozip format	暗号化されていないファイルを復号しようとしている。paozip check で確認
require フックが動作しない	Paozip.install_importer を トップレベル で呼んでいるか確認
.rb と .rbe の両方がある	.rb が優先されます。暗号化後は .rb を削除してください
復号後のファイルが文字化けする	暗号化時と同じキーを使用しているか確認
paozip コマンドが見つからない	gem install paozip が成功しているか確認。gem list paozip で確認

11. 使用許諾

paozip for Ruby の使用について、paozip for Ruby の使用者（以下「利用者様」と称します）と有限会社パオ・アット・オフィス（以下「弊社」と称します）は、以下の各項目についての内容に同意するものとします。

1. 使用許諾書

この使用許諾書は、利用者様が paozip for Ruby を使用する場合に同意しなければならない契約書です。

2. 使用許諾書の同意

利用者様が paozip for Ruby を使用する時点で、本使用許諾書に同意されたものとします。同意されない場合は、paozip for Ruby を使用する事はできません。

3. ライセンス（使用権）の購入

利用者様が paozip for Ruby の製品版を使用して開発を行う場合には、1台の開発用コンピュータで paozip for Ruby を使用するにあたり、1ライセンスを購入する必要があります。

お客様環境等、開発コンピュータでないマシンで paozip for Ruby を使用する場合ライセンスは必要ありません。ランタイムライセンスフリーでございます。

4. 著作権

paozip for Ruby の著作権は、いかなる場合においても弊社に帰属いたします。

5. 免責

paozip for Ruby の使用によって、直接的、あるいは、間接的に生じた、いかなる損害に対しても、弊社は補償賠償の責任を負わないものとします。

6. 禁止事項

paozip for Ruby 及びその複製物を第三者に譲渡・貸与する事は出来ません。paozip for Ruby を開発ツールとして再販/再配布することを禁止します。なお、暗号化されたファイルを配布することは問題ございません。

7. 保証の範囲

弊社は paozip for Ruby の仕様を予告無しに変更することがあります。その場合の利用者様に対する情報提供は、弊社 WEB サイトにて行う事とします。

8. 適用期間

本使用許諾条件は利用者様が paozip for Ruby を使用した日より有効です。

12. 代金支払い方法

paozip for Ruby の製品版をご利用頂ける場合は、ライセンスを購入して頂く必要があります。

体験版について: すべての機能を制限なくお試しいただけます。体験版では [TRIAL] メッセージが表示されます。

必要なライセンス数の数え方

paozip for Ruby で開発を行うパソコンの台数

1ライセンス当たりの価格

11,000円（税込）

バグフィックス等のバージョンアップは原則として無償とさせて頂きます。

お支払方法

11,000円 × ライセンス数 を下記口座へ銀行振込、または、郵便振替による送金をして下さい。

銀行名	支店名（コード）	口座番号	名義
三菱UFJ銀行	新宿支店 (341)	普通 3831891	ユ）パオアットオフィス
PayPay銀行	すずめ支店 (002)	普通 6461359	ユ）パオアットオフィス

郵便口座番号	名義
00150-0-576845	有限会社 パオ・アット・オフィス

※ 振込手数料は利用者様負担でお願い致します。

お支払いの通知と製品の送付

1. 振り込みが完了した時点で、必ず弊社 WEB サイトの入金連絡フォームから入金のご連絡をお願いいたします。
2. 弊社では上記連絡を受けて入金確認を行い、paozip for Ruby の製品版を利用者様へ電子メールにてお送りさせて頂きます。

見積書/納品書/請求書/領収証の発行

見積書/納品書/請求書/領収証の発行は可能でございます。製品サイトでの手続きにより発行いたします。

お問い合わせ

製品に関するお問い合わせは、下記までお願いいたします。

有限会社 パオ・アット・オフィス

・製品サイト: https://www.pao.ac/paozip/

・購入ページ: https://www.pao.ac/paozip/buy.html

・メール: info@pao.ac

paozip for Ruby — あなたの Ruby コードを、シンプルに、確実に、守ります。