paozip for Perl

Perl 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 Perl とは

「Perl で書いた大切なコードを、暗号化して配布できたら...」

そんな願いを叶えるのが paozip for Perl です。

あなたが長年かけて書き上げた Perl のモジュール —
独自のテキスト処理エンジン、正規表現の秘伝のレシピ、業務に特化した複雑なロジック —
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。use や require
するだけで自動復号。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 Perl 実装 — XS モジュール不要。どこでも動きます
- Windows / macOS / Linux 全環境対応
- @INC フック — 暗号化 .ple ファイルを透過的に use / require できる
- Mojolicious / Dancer2 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
- CPAN スタイルのインストール — いつもの perl Makefile.PL && make install でOK

1.3 仕組み（ラッパー方式）

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

```
app.pl ─
  │
  ├── use Paozip;
  ├── Paozip::install_importer();   ← @INC
  │
  └── use SecretLogic;              ← SecretLogic.ple
        │
        └── (.ple)
```

ポイント：

- app.pl（ラッパー）は暗号化しません。これが「入口」になります
- SecretLogic.ple（暗号化ファイル）は use するだけで自動復号
- 利用者からは、普通に Perl 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	Perl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 Perl）
run コマンド	なし	あり	あり
ファイル拡張子	.php	.pye	.ple

Perl版はPython版と同じラッパー方式ですが、Pure Perl 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	Strawberry Perl / ActivePerl
macOS 12+	システム Perl または perlbrew
Ubuntu 20.04+	apt / perlbrew
CentOS/RHEL 8+	dnf / perlbrew
その他 Linux	Perl 5.26+ が動作すれば OK

2.2 対応 Perl バージョン

バージョン	対応状況
Perl 5.26	OK
Perl 5.30	OK
Perl 5.34	OK
Perl 5.36+	OK（推奨）

> Perl 5.24 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
Mojolicious	OK
Dancer2	OK
Catalyst	OK
CGI スクリプト	OK
CLI スクリプト	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-perl-demo
docker-compose build
docker-compose run --rm paozip-demo
```

コンテナの中に入ったら：

```
#
cat lib/SecretLogic.pm          # ←

#
cat lib/SecretLogic.ple         # ←

#
perl app.pl                     # ←

# paozip run
paozip run lib/SecretLogic.ple  # ← 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 CPAN からインストール

```
# CPAN
cpan install Paozip

#  cpanm
cpanm Paozip
```

Pure Perl なので、XS（C コンパイラ）は不要です。

3.3 ソースからビルド

納品された paozip-perl ディレクトリを使って：

```
cd paozip-perl
perl Makefile.PL
make
make test
make install
```

3.4 インストールの確認

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

# Perl API
perl -e "use Paozip; print Paozip::version_string(), qq(\n)"
```

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：

```
# SecretLogic.pm
package SecretLogic;
use strict;
use warnings;

sub calculate {
    my ($x) = @_;
    #
    return ($x * 31337 + 42) % 65536;
}

sub api_key {
    return "sk-very-secret-key-12345";
}

1;
```

暗号化！

```
paozip encrypt SecretLogic.pm
#  SecretLogic.pm → SecretLogic.ple

#
paozip check SecretLogic.ple
# → SecretLogic.ple: Encrypted (paozip format)
```

4.2 ラッパーを作ってみよう

```
#!/usr/bin/env perl
# app.pl —
use strict;
use warnings;
use Paozip;
Paozip::install_importer();

#
use SecretLogic;

#
print SecretLogic::calculate(100), "\n";
print SecretLogic::api_key(), "\n";
```

4.3 実行してみよう

```
#  .pm .ple
rm SecretLogic.pm

#
perl app.pl
# → 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 script.ple
```

5. CLI コマンドリファレンス

5.1 encrypt（暗号化）

```
paozip encrypt <file.pl|file.pm> [-o output]
```

オプション	説明
<file>	暗号化する Perl ファイル（.pl / .pm）
-o output	出力ファイルパス（省略時: .ple に変換）

例：

```
paozip encrypt lib/SecretLogic.pm
# → lib/SecretLogic.ple

paozip encrypt script.pl
# → script.ple

paozip encrypt lib/Secret.pm -o encrypted/Secret.ple
```

5.2 decrypt（復号）

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

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

例：

```
paozip decrypt lib/SecretLogic.ple
# → lib/SecretLogic.pm
```

> 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.ple> [args...]
```

オプション	説明
<file.ple>	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.ple
paozip run script.ple --config production --verbose
```

5.4 check（暗号化チェック）

```
paozip check <file>
```

例：

```
paozip check lib/Secret.ple
# → lib/Secret.ple: Encrypted (paozip format)

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

5.5 version（バージョン表示）

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

6. Perl API リファレンス

```
use Paozip;
```

6.1 暗号化・復号

```
#
my $encrypted = Paozip::encrypt($data, $key);

#
my $decrypted = Paozip::decrypt($data, $key);
```

$key を省略（undef）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

```
#  → .ple
my $output_path = Paozip::encrypt_file($path, $output_path, $key);

#  →
my $decrypted = Paozip::decrypt_file($path, $key);

#  →
my $output_path = Paozip::decrypt_file_to($path, $output_path, $key);
```

6.3 チェック操作

```
#
if (Paozip::is_encrypted($data)) { ... }

#
if (Paozip::is_encrypted_file($path)) { ... }
```

6.4 製品情報

```
Paozip::is_licensed()         # 1 / 0
Paozip::get_license_email()   # "user@example.com"
Paozip::get_product_info()    #
Paozip::version_string()      # "paozip for Perl 2.0.0 [TRIAL]"
```

7. use / require フックの使い方

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

7.1 基本的な使い方

```
use Paozip;
Paozip::install_importer();    # ← 1

#  use
use SecretModule;              # SecretModule.ple
```

たったこれだけ。install_importer() を呼ぶだけで、Perl の @INC に paozip のフックが組み込まれます。

7.2 動作の仕組み

1. use SecretModule が呼ばれる
2. Perl が @INC のパスから SecretModule.pm を探す
3. 見つからない場合、paozip フックが SecretModule.ple を探す
4. SecretModule.ple が見つかれば、復号してモジュールとして評価

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

7.3 注意事項

・Paozip::install_importer() は必ず スクリプトの先頭付近 で呼び出してください
・BEGIN ブロック内で呼び出すと、後続の use 文に間に合います
・require でも .ple ファイルを読み込めます

```
# BEGIN
use Paozip;
BEGIN { Paozip::install_importer(); }

use SecretModule;  #  .ple
```

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 Mojolicious

Mojolicious アプリケーションで使う場合は、スクリプトの先頭でフックをインストールします：

```
#!/usr/bin/env perl
# myapp.pl
use Paozip;
Paozip::install_importer();

use Mojolicious::Lite;
use MyApp::SecretController;  # SecretController.ple

get '/' => sub {
    my $c = shift;
    $c->render(text => MyApp::SecretController::process());
};

app->start;
```

```
#
paozip encrypt lib/MyApp/SecretController.pm
rm lib/MyApp/SecretController.pm

# Mojolicious
perl myapp.pl daemon
```

9.2 Dancer2

```
#!/usr/bin/env perl
# app.pl
use Paozip;
Paozip::install_importer();

use Dancer2;
use MyApp::Routes;  # Routes.ple

dance;
```

```
paozip encrypt lib/MyApp/Routes.pm
rm lib/MyApp/Routes.pm
perl app.pl
```

9.3 CGI スクリプト

```
#!/usr/bin/env perl
# index.cgi
use strict;
use warnings;
use CGI;
use Paozip;
Paozip::install_importer();

require 'secret_logic';  # secret_logic.ple

my $q = CGI->new;
print $q->header;
print SecretLogic::generate_html();
```

```
paozip encrypt secret_logic.pm
rm secret_logic.pm
```

9.4 Docker

```
FROM perl:5.36

WORKDIR /app

# paozip
COPY paozip-perl/ /tmp/paozip-perl/
RUN cd /tmp/paozip-perl && \
    perl Makefile.PL && \
    make && make install && \
    rm -rf /tmp/paozip-perl/

#
COPY . .

#
RUN paozip encrypt lib/SecretLogic.pm && rm lib/SecretLogic.pm

CMD ["perl", "app.pl"]
```

> ビルド時に暗号化し .pm 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
Can't locate SecretModule.pm in @INC	Paozip::install_importer() を暗号化モジュールの use より前に呼んでいるか確認
.pm と .ple の両方がある	.pm が優先されます。暗号化後は .pm 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	make install が成功しているか確認。PATH を確認

11. 使用許諾

paozip for Perl の使用について、paozip for Perl 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 Perl 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 Perl 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 Perl 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 Perl 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 Perl 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 Perl 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 Perl 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 Perl 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 Perl 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 Perl 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 Perl 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 Perl を使用した日より有効です。

12. 代金支払い方法

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