ソリトン・ランダム・ジェネレータ
ソフトウェア・デベロプメント・キット

Soliton Random Generator - Software Development Kit

SRG-SDK

マニュアル

-- 対象製品 --

SRG-SDK Free Windows32bit版
SRG-SDK Standard Windows32bit版
SRG-SDK Enterprise Windows32bit版

有限会社電机本舗
http://www.dnki.co.jp/

使用承諾書

[1:使用承諾一般条項]
〇有限会社電机本舗(以下『弊社』といいます)は、本製品の使用者(以下『使用者』といいます)に対してこのパッケージに入っている製品を本書に従い使用することを承諾します。製品とは、本パッケージに入っているソフトウェア、プログラムディスク、マニュアルのことです。ただし、サードパーティ製の添付プログラム、マニュアルは製品に含めません。サードパーティ製のプログラムの使用については各メーカの規定に従うものとします。
〇製品に関する著作権およびその他の一切の権利は、本契約で明示的に付与したものを除き全て弊社に帰属します。
〇製品のサポートは購入日から1年間、弊社が実施します。ただし、「SRG-SDK Free 」、サンプル、試用品、オマケとして添付されたものは原則としてサポートの義務を伴わないものとします。

[2:使用制限]
〇使用者は製品を同時に複数のコンピュータで使用できません。また複数の使用者による使用も禁止します。但しマニュアル等において例外的に複数の使用を許可している場合はこの限りではありません。

[3:複製等の禁止]
〇使用者はマニュアルに記載されていない限り製品の全部または一部を複製、解析、改変はできません。
〇使用者は弊社の承諾なしに製品の一部ないし全部を売却、譲渡、貸与のいかなる方法でも第三者に使用できません。
〇使用者の製品の使用のための印刷・コピーは自由ですが、次のコピーライトを必ず各ページに記載してください。その他の転載などについては弊社まで御連絡ください。 コピーライト ”(C)2008, 有限会社電机本舗”

[4:保証]
〇使用者は製品の受け渡しから1ケ月以内に本製品や印刷物の物理的な障害を発見した時は交換を要求できます。

[5:免責]
〇弊社は前条に定める場合を除き製品に関していかなる保証も行いません。
〇弊社は使用者が製品の使用に関して直接または間接に生じる一切の損害(通常損害、特別損害およびその他一切の損害)について責任をおいません。

[6:使用権の消滅]
〇使用者が本承諾書に違反した場合または著作権法その他の命令に違反することによって、製品に関して弊社の著作権およびその他の権利を侵害した場合は[1:使用承諾一般条項]規定の使用権は消滅します。この時はただちに製品を弊社に返還しなければいけません。なお返却のための費用は使用者に負担していただきます。

有限会社電机本舗
〒108-0074 東京都港区高輪1-2-16鈴木ビル6A 電話(03)5449-7057
E-Mail:tec@dnki.co.jp
URL: http://www.dnki.co.jp/

目次

1. SRG-SDK について....4
1.1 概要と特徴....4
1.2 ラインナップ....5
1.3 製品パッケージ....5
1.4 使用環境....5
2. インストール方法....6
3. 簡単な使い方....6
4. 関数リファレンス....7
4.1. 関数一覧....7
4.2. Meta Level functions....8
void _8_lib_inz (void)....8
void _8_inz_meta_seed (unsigned char *seed, int n)....9
void _8_randam_level (int level)....10
4.3. High Level functions....11
short _8_enc_file(char *dist, char *source, int mode)....11
short _8_dec_file(char *dist, char *source, int mode)....13
4.4. Standard Level functions....14
int _8_xopen(_inf_soliton_rnd *rnd_tbl, char *file, int rw_mode, int mode)....14
int _8_xclose(int fd)....15
int _8_xwrite(_inf_soliton_rnd *rnd_tbl, int fd, unsigned char *buf, int no)....16
int _8_xread(_inf_soliton_rnd *rnd_tbl, int fd, unsigned char *buf, int no)....18

1. SRG-SDK について

1.1 概要と特徴

「SRG-SDK」は超長周期擬似乱数発生装置「ソリトン・ランダム・ジェネレータ(SRG)」により10の27000乗の乱数データを生成するソフトウェア開発用のキットです。※特許出願済。
最大で10の27000乗という数の乱数のシードを扱うことができます。

【SRGの特徴】

超長周期

内部に8Kバイトのメモリを使用し10の27000乗という超長周期を持ち、確保する作業用メモリサイズに合わせてスケーラブルに周期が伸びる特性があります。

超高速

乗算・除算命令というCPU負荷の高い命令を一切使用していないため、8ビットや16ビットCPUにおいても効率を落とさずMIPS値に比例した乱数生成を実現できます。 インテル製CPUセレロン1．7GHzのパソコンにおいて秒速1Gbps（125Mバイト）の乱数発生能力を持ちます。

精度

アメリカ政府が無料配布している乱数表の検定プログラムNIST「SP800－22」において。1Gビットファイルを10サンプル検定し、各項目で合格値を出す成績を上げています。
SRGはBlumBlumShub乱数発生方式と同系統の予測不可能型アルゴリズムを採用しており、一定のパターンが続くと先が予測できるという欠点がありません。暗号用途として高水準高品質の乱数を提供できます。

1.2 ラインナップ

SRG-SDK にはファイル操作用のシリーズと、それ以外の用途に用いる特別バージョンがあります。それぞれ Windwos32bit 版、Linux 版が存在しています。

1. SRG-SDK 3種 Free , Standard , Enterprise

ファイル操作向けに特化したSDKです。ファイルを「暗号化して保存（書き込み）」「復号化して開く（読み込む）」操作を行います。 ランダム・シード（乱数種）の数によって3つのラインナップがあります。

SRG-SDK Free

無料配布版です。8ビット（256個）のランダム・シードを提供します。

SRG-SDK Standard

有償製品。32ビット（4,294,967,296個）のランダム・シードを提供します。

SRG-SDK Enterprise

有償製品。10の27000乗のランダム・シード提供します。

2. SRG-SDK Prime

特別バージョンです。 1Gbpsを越す高速、高品質な乱数発生装置を提供します。 各種シュミレータ、モンテカルロ法、通信データの暗号化に最適です。

3. SRG-SDK Environment

特別バージョンです。 C言語標準ファイルライブラリ上位互換の暗号化ライブラリです。 暗号化を意識する事無く、アプリケーションの暗号化を実現します。

1.3 製品パッケージ

製品はCDメディアまたはDVDメディアにて提供されます。本文中の「製品メディア」と書かれている場合には、CD メディアまたは DVD メディアのことを指しています。[SRG-SDK Free]はダウンロード版のみです。本文中に「製品メディア」と書かれている場合には、ダウンロード後に解答したフォルダのことを指しています。メディアの中には擬似乱数発生装置を簡単に使えるように機能をまとめたライブラリが格納され、コンピュータソフトウェアに乱数機能とこれを利用した暗号機能を提供します。 また本マニュアルも含まれます。

1.4 使用環境

対応 OS：32bit の WindowsOS （推奨：WindowsXP 環境）
対応言語：C 言語
必要容量：10 MB
供給形態：DLL ないし共有ライブラリ

2. インストール方法

1. 製品メディア内のFileEncSDKフォルダを開発環境にコピーします。
2. Lib8file.dll,lib8file.libを各自の開発環境にコピーします。
3. プロジェクタに登録後、利用可能になります。

3. 簡単な使い方

1. 共有ライブラリの使い方は次のプロジェクトを参考にしてください。
2. EncFileSDK->Testフォルダに「Test.dsw」というVC6用のプロジェクタがあります。
3. このファイルを起動し、ビルド、実行してください。サンプルプログラムが動作します。

4. 関数リファレンス

擬似乱数を生成する関数群、ならびに乱数の生成関数を説明します。

4.1. 関数一覧

【 Meta Level functions 】

_8_lib_inz()

“_8_”で始まる関数群、本ライブラリを初期化する。本関数か_8_inz_meta_seed ()のどちらかを必ず呼ぶ必要がある。

_8_inz_meta_seed ()

同上。_8_lib_inz()との違いは、ライブラリのシードを不可逆的に強制更新する。次のリロードまで乱数の系列を変更する。

_8_randam_level()

暗号化に利用する乱数の生成を速度優先とするか、乱数性を優先とするかを指定する。

【 High Level functions 】

_8_enc_file ()

平文のファイルを読み、暗号化ファイルを作成する。

_8_dec_file()

暗号化ファイルを読み平文のファイルを作成する。

【 Standard Level functions 】

_8_xopen()

暗号用のストリームファイルをオープンする。

_8_xclose()

暗号用のストリームファイルをクローズする。

_8_xread()

暗号用のストリームファイルを読む。データを自動的に復号化する。

_8_xwrite()

暗号用のストリームファイルへデータを暗号化して書く。

4.2. Meta Level functions

void _8_lib_inz (void)

void _8_lib_inz (void)

本ライブラリ(パッケージ)の初期化を行います。
本ライブラリは初期状態で、この関数を使用します。
本ライブラリはこの関数または、_8_inz_meta_seed ()のどちらかを必ず呼ぶ必要があります。

本関数はラインナップによってシードが異なります。
[SRG-SDK Free]ではシードは1バイトです。
[SRG-SDK Standard]ではシードは4バイト
[SRG-SDK Enterprise]ではシードは9213バイトです。

簡単に使用するのであれば、_8_inz_meta_seed ()の使用を控え、本関数の使用を推奨します。
高い信頼性を得たい場合には、_8_inz_meta_seed ()を推奨します。

■引数

none

■戻り値

none

■Note…

void _8_inz_meta_seed (unsigned char *seed, int n)

void _8_inz_meta_seed (unsigned char *seed, int n)

本関数はライブラリを初期化します。
また、シードを決定します。
ただし、乱数発生毎に９２１３バイトのシードを与える事は速度上不利です。

各乱数はこのシードを元に動作します。
一度実行すると、本ライブラリの中の暗号化ルールは不可逆的に更新されるので注意。
もし2回実行した場合、初回の更新に対して、さらに追加更新がかかり、元に戻すことはできないので注意。
リセットしたい場合は、本体アプリを再起動する必要があります。
本関数はライブラリ起動にあたり一度だけ呼ぶようにすることを推奨します。

■引数

```
unsigned char *seed; // 最大９２１３バイトのバイナリデータ
int n;   // シードの長さを指定する。
         // [ SRG-SDK Free ]では１バイトのみです。
         // [ SRG-SDK Standard ]では１～４バイトの範囲で指定可能です。
         // [ SRG-SDK Enterprise ]では１～９２１３バイトの範囲で指定可能です。
```

■戻り値

none

■Note…

void _8_randam_level (int level)

void _8_randam_level (int level)

本ライブラリ(パッケージ)で使用する乱数発生の品質を指定します。将来の拡張に備えたもので、実装されていません。

■引数

```
int     level   ;               // ゼロ=高速標準(デフォルト)、1=高品質
```

■戻り値

none

■Note…

short _8_enc_file(char *dist, char *source, int mode)

short _8_enc_file(char *dist, char *source, int mode)

指定したファイルを元に暗号化ファイルを作成します。簡易な暗号化機能です。実行後は平文のファイルを消去してください。

■引数

```
char *dist;          // 生成するファイル名
char *source;        // 読み取るファイル名
int mode;            // 暗号化モード, 0=標準、1=トランスペアレントモード
```

■戻り値

0=Normal, else Err

■Note…

標準モードの時は現在時間をシード（暗号化の鍵）として与えファイルを暗号化します。
現在時間はミリ秒谷で与えます。
このシード情報はファイルの先頭に書き込みます。暗号化はメタシードと現在時間の複合により行います。

⇒長所：ファイルの頭にシードが付属するので、暗号化ファイルのバックアップができる。
⇒短所：ファイルサイズが数バイト（通常８バイト）増加する。

◎Linux動作環境時にトランスペアレントモードを使用した場合には、ファイルのiノードの位置情報をシードとします。下記の長所短所があります。繰り返しますが、これはLinuxでのみの性質です。Windows,BSD-Unixではこのようなことはありません。

⇒長所：ファイルサイズが変化しない。
⇒短所：暗号化ファイルはHDD上の位置情報をシードとしているのでバックアップコピーをかけると解読できなくなる。もしくは、専用のバックアッププログラムを自作する必要があります。Windows,BSD-Unixの場合はファイルのバースデイタイムを使用するのでこの欠点はありません。

暗号強度はどちらも十分強力ですが。理論上、ほぼ完全なワンタイムパッド（無限暗号）を満たすのは「標準モード」です。
「トランスペアレント」モードは同じiノードに対しては同じ乱数（暗号ルール）が適用されます。確率論的にワンタイムパッド（無限暗号）にならない場合があります。
ただし、不正コピーした時点でファイルが破壊されるというメリットがあります。

```
---- Sample      short _8_enc_file( char *dist, char *source, int mode )  ---
#include        <stdio.h>
#include        <stdlib.h>
#include        <string.h>

#include        "soliton_rnd_type.h"
#include        "lib8_file.h"

void sample1(void)
{
        _8_enc_file( "enc.txt", "tmp.txt", 1 );                         // tmp.txtからenc.txtを作る
}

int main()
{
        _8_inz_meta_seed( (unsigned char*)"Hello", 5 );     // ライブラリを初期化
                                                           // する(必須)
                                                           // 2重起動しないように注意。
        sample1();
}
```

short _8_dec_file(char *dist, char *source, int mode)

short _8_dec_file(char *dist, char *source, int mode)

指定した暗号化ファイルを元に復号したファイルを作成する。
本関数は簡易な暗号化機能です。実行後は平文のファイルを消去してください。

■引数

```
char *dist;          // 生成するファイル名
char *source;        // 読み取るファイル名
int mode;            // 暗号化モード, 0=標準、1=トランスペアレントモード
                       ※詳細は_8_enc_file()を参照。
```

■戻り値

0=Normal, else Err

■Note…

```
---- Sample   short _8_dec_file( char *dist, char *source, int mode ) ---
#include      <stdio.h>
#include      <stdlib.h>
#include      <string.h>

#include      "soliton_rnd_type.h"
#include      "lib8_file.h"

void sample2(void)
{
      _8_dec_file( "tmp1.txt", "enc.txt", 1 );      // enc.txtを解読しenc.txtを作る
}

int main()
{
      _8_inz_meta_seed( (unsigned char*)"Hello", 5 );    // ライブラリを初期化
                                                        // する(必須)
                                                        // 2重起動しないように注意。
      Sample2();
}
```

4.4. Standard Level functions

int _8_xopen(_inf_soliton_rnd *rnd_tbl, char *file, int rw_mode, int mode)

int _8_xopen(_inf_soliton_rnd *rnd_tbl, char *file, int rw_mode, int mode)

ファイルを暗号モードでOpenしストリームを開く。

■引数

```
_inf_soliton_rnd *rnd_tbl;      // 乱数制御構造体
char *file;             // openするファイル名
int rw_mode;            // openするモード。
                        // 0=Read(復号読み取り)、有値=Write(暗号書き込み)
int mode;               // 暗号化モード, 0=標準、1=トランスペアレントモード
                        // ※詳細は_8_enc_file()を参照。
```

■戻り値

C標準ライブラリのopen()と同じ。ファイルディスクリプタを返す。

0>=Normal, else Err

■Note…

ストリームはシーケンシャルファイルなので注意。シーク、リワインドはサポートしてない。

サンプルプログラムは、_8_xread()/_8_xwrite()を参照。

int _8_xclose(int fd)

int _8_xclose(int fd)

暗号Openしたファイルをクローズする。

■引数

int　　fd　　// ファイルディスクリプタ

■戻り値

C標準ライブラリのclose()と同じ。

0=Normal, else Err

■Note…

サンプルプログラムは、_8_xread()/_8_xwrite()を参照。

int _8_xwrite(_inf_soliton_rnd *rnd_tbl, int fd, unsigned char *buf, int no)

int _8_xwrite(_inf_soliton_rnd *rnd_tbl, int fd, unsigned char *buf, int no)

暗号Openしたファイルへデータを書き込む。

■引数

```
_inf_soliton_rnd *rnd_tbl;      // 乱数制御構造体
int      fd;                    // ファイルディスクリプタ
unsigned char *buf;             // 暗号化して書き込みたいデータ
                                // バッファの内容は破壊されるので注意
int      no;                    // データ数
```

■戻り値

C標準ライブラリのwrite()と同じです。
書き込んだデータ数。
暗号化ストリームの性質上、本関数はnoサイズだけ全て書き込みます。
　もし、noと返り値が異なる時は、ディスクエラーが考えられます。
リトライする時は、一度クローズして最初からやり直してください。

■Note…

C標準ライブラリのwrite()と使い方は同じです。

```
---- Sample  nt _8_xwrite( _inf_soliton_rnd *rnd_tbl, int fd, unsigned char *buf, int no ) ---
#include        <stdio.h>
#include        <stdlib.h>
#include        <string.h>
#include        "soliton_rnd_type.h"
#include        "lib8_file.h"

void sample3(void)
{
    int  fd;
    _inf_soliton_rnd rnd_tbl;
    char        buf[128];

        fd = _8_xopen( &rnd_tbl, "hoge.txt", 1, 1 );          // 暗号化ファイルをopen
        if( fd<=0 ) {
                printf( "hoge.txt open err¥n" );
        }
        else {
                strcpy( buf, "Hello" );
                _8_xwrite( &rnd_tbl, fd, buf, 5 );            // 暗号化して書き込む
                _8_xclose( fd );
        // クローズ
        }
}

int main()
{
        _8_inz_meta_seed( (unsigned char*)"Hello", 5 );    // ライブラリを初期化
                                                          // する(必須)
                                                          // 2重起動しないように注意。
        Sample3();
}
```

int _8_xread(_inf_soliton_rnd *rnd_tbl, int fd, unsigned char *buf, int no)

int _8_xread(_inf_soliton_rnd *rnd_tbl, int fd, unsigned char *buf, int no)

暗号Openしたファイルからデータを読み込む。

■引数

```
_inf_soliton_rnd *rnd_tbl;      // 乱数制御構造体
int      fd                     // ファイルディスクリプタ
unsigned char *buf;             // 復号して読み込みたいデータ
int      no;                    // データ数
```

■戻り値

C標準ライブラリのread()と同じです。

0=Normal, else Err

■Note…

C標準ライブラリのread()と使い方は同じです。

```
---- Sample  int _8_xread( _inf_soliton_rnd *rnd_tbl, int fd, unsigned char *buf, int no )    ---
#include        <stdio.h>
#include        <stdlib.h>
#include        <string.h>

#include        "soliton_rnd_type.h"
#include        "lib8_file.h"

void sample4(void)
{
    int  fd;
_inf_soliton_rnd rnd_tbl;
char    buf[128];
        memset( buf, 0, sizeof(buf) );

        fd = _8_xopen( &rnd_tbl, "hoge.txt", 0, 1 );            // 暗号化ファイルをopen
        if( fd<=0 ) {
                printf( "hoge.txt open err¥n" );
        }
        else {
                _8_xread( &rnd_tbl, fd, buf, 5 );       // 復号して5バイト読む
                printf( "%s ¥n", buf);
                _8_xclose( fd );
        // クローズ
        }
}

int main()
{
        _8_inz_meta_seed( (unsigned char*)"Hello", 5 );    // ライブラリを初期化
                                                          // する(必須)
                                                          // 2重起動しないように注意。
        Sample4();
}
```