

TwISDK

CRYPTO マニュアル

ソフトウェア暗号化ライブラリ

2008-09-16

任天堂株式会社発行

このドキュメントの内容は、機密情報であるため、
厳重な取り扱い、管理を行ってください。

目次

1	はじめに	4
1.1	概要	4
1.2	CRYPTOライブラリの構成	4
1.3	各暗号化方式の特徴について	4
1.4	動作環境	5
2	RC4 暗号化	6
2.1	目的と制限	6
2.2	RCアルゴリズムの特徴	6
2.3	RCアルゴリズムの原理	6
3	電子署名	8
3.1	電子署名とは	8
3.2	CRYPTOライブラリが提供する電子署名機能	9
3.3	署名データの形式	9
3.4	署名データの作成例	9
3.4.1	署名用の RSA 鍵を生成する	9
3.4.2	RSA 鍵の内容を確認する	10
3.4.3	電子署名を作成する	10
3.4.4	電子署名を検証する	11
4	RSA暗号化	12
4.1	RSA暗号について	12
4.2	RSA暗号を使うときの注意事項	12
4.3	鍵形式および暗号/復号文字列について	12
4.4	制限事項	13
4.5	鍵作成例	13
4.5.1	RSA 秘密鍵を作成する	13
4.5.2	RSA 公開鍵を作成する	13
4.5.3	鍵の動作確認をする	13



図 3-1	電子署名の概略図	8
-------	----------------	---

改訂履歷

改訂日	改訂内容
2008-09-16	新規作成

1 はじめに

1.1 概要

CRYPTO ライブラリは暗号ライブラリです。ゲームで用いるデータを暗号化することで秘匿することや、改ざんを検出するために電子署名を用いて検証することができます。TwlSDK には暗号ライブラリとして AES ライブラリもありますが、AES ライブラリは TWL のハードウェアが有する AES 暗号化機能を利用するのに対して、それ以外の暗号を利用した機能をソフトウェア処理で提供するのが CRYPTO ライブラリの役割です。

注意事項

CRYPTO ライブラリは、「外国為替及び外国貿易法」「輸出貿易管理令」「外国為替令」などで定められた暗号装置としての機能を搭載しているため、日本から輸出する場合は、事前に経済産業省へ輸出許可申請を行い許可を得ておく必要があります。また、海外から他国へ輸出する場合には各国の輸出規制関連法を遵守しなければなりませんので、ご使用の際はご注意下さい。

1.2 CRYPTOライブラリの構成

CRYPTO ライブラリは以下のモジュールから構成されています

- RC4 暗号化 : RC4 秘密鍵暗号による暗号・復号化を行います
- 電子署名 : 電子署名の検証をおこないます。また、TWL 上では署名の作成も可能です。
- RSA 暗号化 : 公開鍵暗号による暗号・復号化を提供します。
- UTIL : CRYPTO ライブラリの動作に必要な関数のうち上記モジュールに当てはまらないものを提供します。

1.3 各暗号化方式の特徴について

TwlSDK では暗号アルゴリズムとして CRYPTO ライブラリで RC4 と RSA を、AES ライブラリで AES を提供しています。

RC4 は TwlSDK で用意している他の暗号アルゴリズムに比べて処理速度が短く暗号・復号が同一のルーチンで可能なことから手軽に利用しやすいという特徴があります。しかし、暗号・復号の鍵が同一であり一旦鍵が漏洩するとユーザによる暗号解読が容易に行えるため暗号が意味をなさなくなる。そのため、RC4 単体で機密度の高いデータの暗号化をすることはできません。RC4 は Nitro-Crypto から存在した暗号化方式のため、主に NITRO でのデータのやり取りも必要となる場合に使います。

RC4 よりも暗号強度が高いのが AES です。TWL でのみ動作すればいいデータの暗号化には AES を用いるといいです。ただし、AES も暗号・復号の鍵が同一であるため鍵の漏洩には注意する必要があります。

RSA は公開鍵方式を用いた暗号です。公開鍵は漏洩しても問題ないため RC4・AES といった共通鍵方式暗号に比べて鍵の管理が格段に安全になるというメリットがあります。また、CRYPTO の RSA は鍵長を 4096bit まで利用可能なため他の暗号化方式に比べて暗号強度も高いです。しかし、暗号処理速度も格段に遅いため配送リスクのある AES

の鍵のみを RSA で暗号化して配送し、データは AES で暗号化するといった限定的な使い方をするのが一般的です。

1.4 動作環境

RC4・電子署名検証・UTIL は TWL でも NITRO でも動作します。

RSA・電子署名作成 は利用している暗号化コアのライセンスの制約上 NITRO では動作しません。また、HYBRID アプリでも NITRO 上では動作しません。

2 RC4 暗号化

2.1 目的と制限

CRYPTO の RC4 暗号化アルゴリズムによる暗号化関数は、NITRO でも TWL でも利用できることを目的として用意しています。そのため、NITRO でも扱うことが必要なデータをネットワークを介して送信するなどに用いることを想定しています。NITRO で扱うことが不要な場合は AES を利用することを推奨します。

RC4 は共通鍵暗号ですので、基本的にはソフト内に暗号化・復号双方で用いる鍵データを保管しておく必要があります。そのため、ROM バイナリの解析によって鍵が明らかになり、暗号が危殆化される可能性があります。**この関数のみを用いて、機密度の高いデータの暗号化や、データの作成者の認証を行わないでください。**

2.2 RCアルゴリズムの特徴

RC4 アルゴリズムは、以下の特徴を持っています。

- 共通鍵暗号である。
- ストリーム暗号である。
- 暗号化・復号処理が高速である。
- 効果的な解析手法が発表されていない。

ストリーム暗号では入力バイト数と出力バイト数が一致しますので、使用は簡単です。反面、いくつかの注意点に気をつけて使わないと、思わぬ脆弱性が残ったままになることがあります。以下に記述する注意点に留意してください。

2.3 RCアルゴリズムの原理

RC4 アルゴリズムの基本的な動作は、鍵から一意に決定される乱数列を作成し、元データと XOR を取る、というものです。そのため、同一の鍵からは常に同じ暗号用の乱数列が生成されることに注意が必要です。これは以下を意味します。

1. 同じ鍵・同じデータからは常に同じ暗号データが生成される。すなわち、どの暗号文とどの暗号文が同一の平文を持つかが分かる。(辞書攻撃)
2. 同じ鍵から作成された 2 つの暗号データの XOR を取ると、それぞれの平文を XOR したものを取得することができる。(差分攻撃の一種)
3. 暗号文の任意の位置のビットを反転させることで、復号後のデータのビットを反転させることができる。(ビット反転攻撃)

辞書攻撃と差分攻撃を回避するためには、毎回異なる InitializationVector(IV)と呼ばれる値を用意し、共通鍵に IV を加えたものを実際の鍵として RC4 アルゴリズムを適用する必要があります。暗号データを送信する際には、一緒に IV も(暗号化せずに)送るようにします。例えば、RC4 関数に鍵として渡す 128bit のうち、96bit を真の秘密鍵として

扱い、残りの 32bit は IV として毎回異なる数で埋めます。

ビット反転攻撃を回避するには、送信するデータに MD5 や SHA-1 などのメッセージダイジェスト値をつけるようにします。攻撃者は元のデータが分かりませんので、任意のビットを変化させることはできても、正しいメッセージダイジェスト値を計算することはできません。MD5 と SHA-1 を求める関数は NITRO-SDK に用意されています。

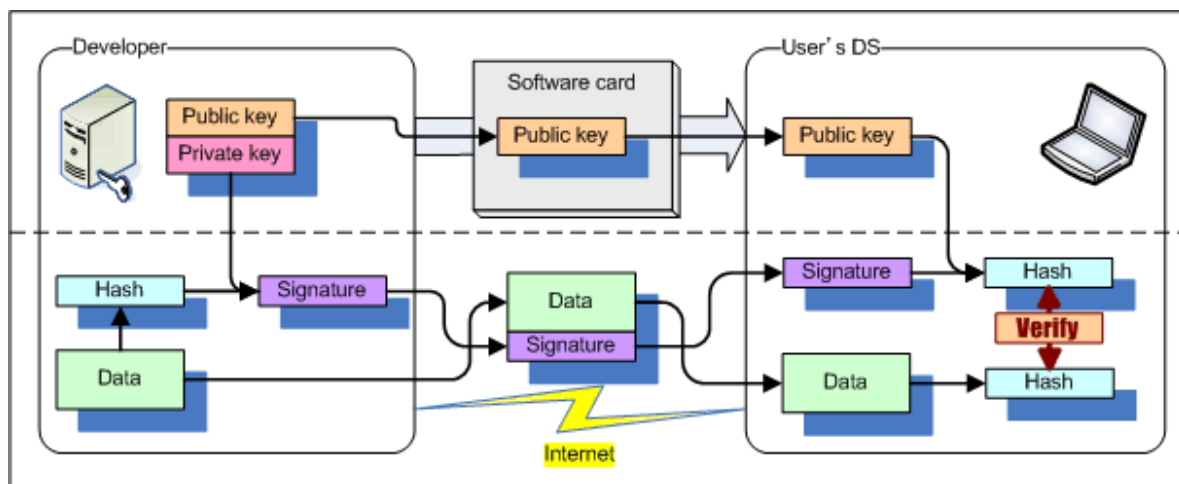
詳細は、一般的な暗号の書籍を参照してください。

3 電子署名

3.1 電子署名とは

電子署名とは、インターネットなどの信頼できない経路から送られてきたデータが、正当なものであることを証明するための機構です。

図 3-1 電子署名の概略図



- 作業主体として署名する人(開発者)・検証する人(ユーザの DS)が関与します。
- データとしては、秘密鍵・公開鍵・送信データ・送信データに対する電子署名、が介在します。
- 秘密鍵は署名する人だけが知っており、秘密にしています。
- 公開鍵は、何らかの信頼できる手段(ゲームカードに焼きこまれている)により、検証する人が前もって入手しているものとします。
- 送信データは任意のサイズのバイナリファイルです。

電子署名を用いてデータの正当性を検証する際の流れは以下の通りです。

1. まず、署名する人が秘密鍵を使用して、送信データから固定長の電子署名を作成します。
2. (実際には、送信データのハッシュ値を取り、それに対して操作を行います。)
3. その後、検証する人が、送信データとそのデータへの電子署名をインターネットなどを経由して受け取ります。
4. 検証する人は、あらかじめ入手していた公開鍵の情報だけを用いて、送信データとその署名から、データが正当なものであることを検証できます。

電子署名という機構の特徴は以下のようになります。

- 公開鍵さえ事前に知っていれば、外部との通信を行わずに、データとそれへの署名だけを用いて正当性を判定できる。

- 公開鍵が明らかになっても、秘密鍵さえ漏洩しなければ、署名の偽造ができない。(すなわち、DS 側の ROM バイナリが万が一解析されたとしても、署名の偽造ができない。)

3.2 CRYPTOライブラリが提供する電子署名機能

電子署名機能としては主に以下の 2 機能があります。

- 実機上での電子署名の検証 (NITRO・TWL 共に可)
- 実機上での電子署名作成 (TWL のみ可)

電子署名検証は `CRYPTO_VerifySignature0`・`CRYPTO_VerifySignatureWithHash0`関数として提供しています。Nitro・Crypto 時代から存在した関数で NITRO でも動作します。

電子署名作成は `CRYPTO_RSA_Sign0`関数等で提供しています。この機能は RSA 社の BSAFE Micro Edition Suite を用いており、ライセンスの都合上 TWL 上でのみ利用可能です。

CRYPTO ライブラリでは証明書の有効期限の管理のような証明書に関する管理機能はありませんので、必要であればアプリケーション側で機能を実装してください。

ちなみに、電子署名はデータの正当性の検証を行うだけの仕組みですので、データの暗号化は行いません。暗号化に関しては CRYPTO ライブラリの RC4・RSA 暗号化機能や AES ライブラリを提供していますので暗号化強度にあわせて利用することが可能です。また TwiWiFi ライブラリを用いてサーバと安全に通信したい場合は NSSL ライブラリ (SSL 通信ライブラリ)を用いてください。

3.3 署名データの形式

`CRYPTO_VerifySignature*`関数に渡す署名データは以下の条件を満たしていればどのような方法で生成しても構いません。

- PKCS#1 に準拠している
- ハッシュアルゴリズムには SHA-1 を使用している
- 公開鍵暗号のアルゴリズムには RSA を使用し、鍵長は 1024bit である
- 使用している公開鍵の公開指数が 65537 である

3.4 署名データの作成例

電子署名は TWL 実機上でも作成可能ですが、一例としてオープンソースの SSL ツールキットである OpenSSL で電子署名を作成する手順を以下に述べます。

3.4.1 署名用の RSA 鍵を生成する

OpenSSL がインストールされている環境のコマンドラインで、以下のコマンドを入力することで、鍵長 1024 bit の RSA 鍵ファイル `privkey.pem` を作成します。

```
> openssl genrsa -out privkey.pem 1024
```

万が一、`privkey.pem` が漏洩すると、誰もが署名できるようになってしまいます。秘密鍵の鍵ファイルは厳重に取り扱うようにしてください。

鍵の作成時に暗号化方式を指定することで、`privkey.pem` にパスワードによる暗号化を行うことも可能です。以下の例では、新規に作成した `privkey.pem` を 3DES アルゴリズムによって暗号化します。

```
> openssl genrsa -des3 -out privkey.pem 1024
```

詳細は `openssl` のリファレンスを参照してください。

3.4.2 RSA 鍵の内容を確認する

以下のコマンドで、`privkey.pem` の内容を確認することができます。

```
> openssl rsa -in privkey.pem -text -noout
```

この中には、署名を行うために必要な秘密情報が含まれていますが、後の検証のために必要な情報(公開鍵)は `modulus` と `publicExponent` の2つの数値です。

以下はコマンドの出力から `modulus` と `publicExponent` を抜粋した例です。

```
modulus:
00:eb:95:be:33:19:73:64:f2:72:2c:87:c8:0a:f3:
1c:ba:e0:4c:e0:3e:1d:f6:e2:09:aa:70:f0:b3:b9:
0c:86:36:62:2d:12:13:86:fa:a5:3d:93:cb:5f:0b:
45:64:9b:7b:eb:b5:c6:f9:42:99:70:46:f3:14:6d:
8f:f9:b9:ec:38:30:a0:1c:28:0d:30:d9:86:1a:4d:
1b:f2:e9:05:1b:43:06:b2:c0:55:ed:c4:bb:8e:1a:
a5:ab:2b:54:e5:dc:8d:70:cf:af:91:94:c9:e9:8f:
7f:9f:29:28:be:e7:01:b0:20:d4:f2:71:58:93:db:
25:1c:26:bc:98:f3:a2:b3:47
publicExponent: 65537 (0x10001)
```

`CRYPTO_VerifySignature*` で扱う公開指数は 65537 で固定ですので、`publicExponent` が 65537 であることを確認してください。

`modulus` の数値は、以下のコマンドで出力することも可能です。

```
> openssl rsa -in privkey.pem -modulus -noout
```

このコマンドでは以下のような文字列が出力されます。

```
Modulus=EB95BE33197364F2722C87C80AF31CBAE04CE03E1DF6E209AA70F0B3B90C8636622D12138
6FAA53D93CB5F0B45649B7BEBB5C6F942997046F3146D8FF9B9EC3830A01C280D30D9861A4D1BF2E9
051B4306B2C055EDC4BB8E1AA5AB2B54E5DC8D70CFAF9194C9E98F7F9F2928BEE701B020D4F271589
3DB251C26BC98F3A2B347
```

この出力中の "Modulus=" の後に続く 16 進数を何らかの方法で C 言語の `u8` の配列に変換して、`CRYPTO_VerifySignature*` に `mod_ptr` として渡してください。なお、この出力例では `Modulus` が 127 バイトですが、このように 128 バイトより少ない場合は 先頭(上位桁)の 0 が省略されていますので、全体が 128 バイトになるように先頭へ 0 を詰めた状態で `mod_ptr` に渡す必要があります。

3.4.3 電子署名を作成する

ここまで準備ができれば、あとは任意のデータを持って来て電子署名を作成するだけです。

電子署名は `CRYPTO_RSA_Sign0` を用いると TWL 上で作成できますし、PC にて作成することもできます。

以下のコマンドで、秘密鍵 `privkey.pem` で `hoge.txt` を署名した `hoge.sign` という署名データが作成されます。

```
> openssl dgst -sha1 -sign privkey.pem -out hoge.sign hoge.txt
```

出来上がったファイルサイズが 128 バイトであることを確認してください。

この 128 バイトのバイナリデータを DS に転送し、`sign_ptr` として `CRYPTO_VerifySignature*()` に渡します。

なお、作成した署名データが正しい電子署名となっているかを PC 上で確認するには、以下のコマンドを実行します。

```
> openssl dgst -sha1 -prverify privkey.pem -signature hoge.sign hoge.txt
```

3.4.4 電子署名を検証する

DS のプログラムに予め公開鍵データを埋め込んでおき、そこへ、通信などの何らかの手段でデータ本体とその電子署名データを送ります。`CRYPTO_VerifySignature*` に対して、そのデータ本体とサイズ、電子署名データ(128 バイト)と、埋め込まれた公開鍵データ(128 バイトの `modulus`)を与えることにより、電子署名を検証し、正当なものと認められれば関数は `TRUE` を返します。

4 RSA暗号化

4.1 RSA暗号について

RSA アルゴリズムによる暗号化関数は公開鍵を用いた暗号を行うために用意しています。

RSA アルゴリズムには以下の特徴があります。

- 公開鍵暗号である。
- TwlSDK で使用できる暗号アルゴリズムの RC4・AES よりも暗号強度が高い
- 暗号化・復号処理は非常に低速である。

公開鍵暗号方式は共通鍵暗号方式に比して鍵の配送に関するリスクが低いというメリットがありますが、他の暗号方式と比較して処理速度が遅いというデメリットも持ち合わせます。そのため暗号化したいデータは他の暗号方式で暗号化し、その暗号化に用いた鍵を RSA で暗号化して配送するという方法を探るのが一般的です。

4.2 RSA暗号を使うときの注意事項

RSA 暗号方式の性質として以下のようなものがあります。

1. 秘密鍵が流出すれば暗号が危殆化する
2. Private Exponent が判明できれば復号ができる(ブルートフォース攻撃)
3. 鍵配送を悪用すればデータの偽装が可能(main-in-the-middle 攻撃)

秘密鍵が流出すると暗号解読も署名偽装も可能となり暗号による安全性は損なわれます。秘密鍵の管理には十分注意してください。

ブルートフォース攻撃に対しては鍵長を長くすれば防ぎやすくなりますが、その分暗号化のための処理速度が増大します。

main-in-the-middle を防ぐためには公開鍵の認証(証明書)が有効です。

詳細は、一般的な暗号の書籍を参照してください。

4.3 鍵形式および暗号/復号文字列について

鍵は公開鍵・秘密鍵共に DER 形式のものを使用します。鍵形式は以下のようにして下さい。

- 鍵のデータフォーマットが ASN.1 形式に準拠しており、DER エンコードされている
- 使用している公開鍵の公開指数が 65537 である

なお鍵長に制限はありませんが、動作確認を取っているのは 1024・2048・4096bit です。

暗号化文字列は以下の形式になっています。

- RSA 暗号にて暗号化
- PKCS#1 v1.5 形式でパディング

暗号化可能な文字列の長さは鍵長よりも 11byte 以上短いものである必要があります。(例えば 1024bit の鍵長の場合は暗号化する文字列は 117byte 以下である必要があります)

CRYPTO ライブラリ以外の方法で暗号化した文字列を CRYPTO ライブラリで復号する場合は上記暗号化文字列形式に従ったものを使用すれば復号できます。

4.4 制限事項

RSA 暗号化アルゴリズムは利用ソフトウェアのライセンスの制限上 TWL でのみ利用可能です。NITRO では使用できません。また、HYBRID アプリで用いた場合でも TWL 上でのみ動作します。

4.5 鍵作成例

一例として、オープンソースの SSL ツールキットである OpenSSL で暗号化に用いる公開・秘密鍵を作成する手順を以下に述べます。

4.5.1 RSA 秘密鍵を作成する

OpenSSL がインストールされている環境のコマンドラインで以下のコマンドを入力することで、PEM 形式・鍵長 1024 bit の RSA 秘密鍵ファイル `privkey.pem` を作成します。

```
> openssl genrsa -out privkey.pem
```

万が一、`privkey.pem` が漏洩すると、誰もが暗号解読および暗号偽造ができるようになってしまいます。秘密鍵の鍵ファイルは厳重に取り扱うようにしてください。

PEM 形式の秘密鍵ができれば DER 形式に変換します。

```
> openssl rsa -outform DER -in privkey.pem -out privkey.der
```

CRYPTO API で秘密鍵を指定するときは、この `privkey.der` の中身を C 言語の `u8` の配列に変換して指定します。なお、`privkey.der` も秘密鍵のため `privkey.pem` 同様厳重に取り扱うようにしてください。

4.5.2 RSA 公開鍵を作成する

以下のコマンドで DER 形式の公開鍵を作成します。

```
> openssl rsa -pubout -inform DER -in privkey.der -outform DER -out pubkey.der
```

CRYPTO API で公開鍵を指定するときは、この `pubkey.der` の中身を C 言語の `u8` の配列に変換して指定します。

4.5.3 鍵の動作確認をする

作成した秘密・公開鍵のペアが正しく機能するかを確認します。

まず、鍵長のよりも短い文字列長の文字列を記載したテキストファイル (`test.txt`) を用意し、公開鍵で暗号化して `test.txt.enc` に変換します。

```
> openssl rsautl -encrypt -in test.txt -out test.txt.enc -pubin -keyform DER -  
inkey pubkey.der
```

作成した秘密・公開鍵のペアが正しく機能するかを確認します。

次に、test.txt.enc を秘密鍵で復号して test.txt.dec に変換します。

```
> openssl rsautl -decrypt -in test.txt.enc -out test.txt.dec -keyform DER -  
inkey privkey.der
```

test.txt と test.txt.dec の中身が一致していたら鍵は正しく動作することが確認できます。

記載されている会社名、製品名等は、各社の登録商標または商標です。

© 2008 Nintendo

任天堂株式会社の許諾を得ることなく、本書に記載されている内容の一部あるいは全部を無断で複製・複写・転写・頒布・貸与することを禁じます。