NITRO-SDK Crypto ライブラリ

オーバービュー

Ver 1.0.0

任天堂株式会社発行

目次

改訂履歴

版	改訂日	改 訂 内 容	承認者	担当者
1.0.0	2006-4-10	初版発行		

1 はじめに

NITRO-SDK Crypto ライブラリ は、ニンテンドーDS 用ゲームソフトの開発に使用できる、セキュリティライブラリの総称です。ライブラリ・パッケージには下記に示すものが含まれています。

- 暗号化ライブラリ
- 電子署名ライブラリ
- 関数リファレンス及びサンプルデモ

NITRO-SDK ライブラリを適切使用することにより、ゲームで用いるデータを隠蔽／保護したり、データそのものの正当性を保証する事が可能です。ただし、使用方法を誤った場合はこの限りではありませんし、適切に使用していたとしても、絶対の安全が保証されるわけではありませんので、あくまでも自己責任でご使用下さい。

重要な注意事項

NITRO-SDK 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。

2　RC4 アルゴリズムによる暗号化

2.1　目的と制限

RC4 アルゴリズムによる暗号化関数は、暗号化技術をゲーム中で手軽且つ低負荷に利用する事を目的として準備されています。例えば、ゲーム中に動的に生成されたデータを平文のままネットワークに流したくない場合や、バックアップに平文のまま保管したくない場合に使用される事を想定しています。

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**。

なお、データの作成者の認証をしたい場合は、CRYPTO_VerifySignature 関数による電子署名の認証機能が利用可能です。 また、NITRO-SDK WiFi ライブラリを用いてサーバと安全に通信したい場合は、SOC_EnableSsl 関数で有効化できる SSL 通信を利用してください。

2.2　RC4 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 RC4 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。

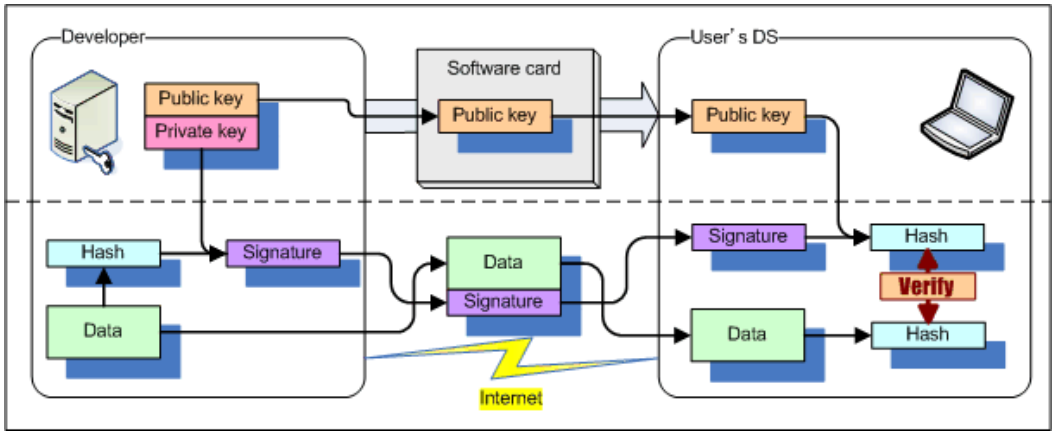
ビット反転攻撃を回避するには、送信するデータに MD5 や SHA-1 などのメッセージダイジェスト値をつけるようにします。攻撃者は元のデータが分かりませんので、任意のビットを変化させることはできても、正しいメッセージダイジェスト値を計算することはできません。MD5 と SHA-1 を求める関数は NITRO-SDK に用意されています。

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。



- 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。

電子著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 NITRO-SDK Crypto ライブラリが提供する電子署名機能

CRYPTO_VerifySignature関数、CRYPTO_VerifySignatureWithHash関数は最も基本的な電子署名の認証の仕組みを提供します。証明書という層がありませんので、証明書の有効期限の管理などの機能はありません。必要に応じて、アプリケーション側で機能を実装してください。なお、NITRO-SDK Crypto ライブラリでは電子署名の作成を行うことはできません。

また、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。平文で送るのを避けたいという程度の要求であれば、CRYPTO_RC4*関数による暗号化機能を使用することが可能です。RC4 アルゴリズムに関しましては、RC4 アルゴリズム概要を参照してください。もしくは、NITRO-SDKWiFi ライブラリを用いてサーバと安全に通信したい場合は SOC_EnableSsl 関数で有効化できる SSL 通信を用いてください。

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

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 1024bit 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*関数で扱う公開指数は 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=EB95BE33197364F2722C87C80AF31CBAE04CE03E1DF6E209AA70F0B3B90C8636622D1
21386FAA53D93CB5F0B45649B7BEBB5C6F942997046F3146D8FF9B9EC3830A01C280D30D9861A
4D1BF2E9051B4306B2C055EDC4BB8E1AA5AB2B54E5DC8D70CFAF9194C9E98F7F9F2928BEE701B
020D4F2715893DB251C26BC98F3A2B347
```

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*関数に 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。
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*関数に渡します。
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*関数に対して、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。

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。