2013年5月8日

**秘密分散とは何？**

**１．秘密分散とは**

　秘密にしたいデータを暗号化する。情報漏えいなどの対策で暗号化は必須である。その一方で、暗号化に用いた暗号鍵の紛失、盗難などに見舞われると、暗号化したデータにアクセスできなくなる、即ち可用性が損なわれてしまう。暗号化を補間する技術が暗号鍵の秘密分散である。

　鍵の分散化とは、対象の鍵に所定の変換を行い、２つ以上の新たな鍵に分ける処理を示す。鍵の分散化で生成した鍵は、単独では有効な鍵でなく、生成した複数の鍵に対して分散化の逆変換を行うと、有効な鍵（元の鍵）を得る（復元する）ことができる。最も簡単な鍵の分散化の例は、鍵を半分に分割し、２つの鍵とする“電子割符”がある。この電子割符の仕組みを応用（分割設計図に従い分割し、分割設計図を割符に付加する）すると、文書ファイルの秘密分散が実現できる。

　秘密分散にはいろいろな手法があるが、まず、秘密分散の一例として、多数決論理関数を利用した実装例の紹介、次に、一般によく利用されている (k, n)しきい値秘密分散法についても実装サンプルを通して紹介し、秘密分散の原理を示す。

**２．多数決論理関数とは**

多数決論理関数は、majority(x, y, z) = ( (x&y) | (y&z) | (z&x) )の式で表わされる論理演算である。簡単に言えば、ｘ、ｙ、ｚのビットの多数決で、即ちビットが２個以上であればON、１個以下であればOFFとなる論理演算を示す。具体的には、次の通りである。

　　　ｋ：１００１

　　　ｘ：０１０１　　０１０１

　　　ｙ：１０１１　　１０１１

　　　ｚ：１０００　　１００１（複数ある）

majority：１００１　　１００１

　上記の例の通り、主鍵ｋから多数決論理関数に基づき発生した副鍵ｘ、ｙ、ｚの内、副鍵ｘ、ｙを固定しても複数の副鍵ｚを生成できることを示している。

この多数決論理関数を利用すると、主鍵ｋは、副鍵ｘと副鍵ｙと副鍵ｚの鍵対に分散して保持され、主鍵ｋの秘密分散が実現できる。

**2.1　実験データ**

　主鍵ｋに対応する多数決論理を満たす副鍵ｘ、ｙ、ｚの生成、や主鍵ｋと任意に与えた副鍵ｘに対応する多数決論理を満たす副鍵ｙ、ｚの生成が可能である。次の式の中で、f(0)は主鍵ｋのビット情報、majorityFunc(0,0,0)は副鍵ｘ、ｙ、ｚのビット情報を、ビット情報0はfalse、1はtrueを示す。

　　ｋに対応するｘ、ｙ、ｚを求める場合

この場合、主鍵ｋのビット情報に対して、次のような8通りの組み合わせがある。

f(0) = majorityFunc(0,0,0), majorityFunc(0,0,1), majorityFunc(0,1,0), majorityFunc(1,0,0)

f(1) = majorityFunc(0,1,1), majorityFunc(1,0,1), majorityFunc(1,1,0), majorityFunc(1,1,1)

　　ｋとｘに対応する、ｙ、ｚを求める場合

主鍵ｋと副鍵ｘのビット情報が”01”の場合、副鍵ｙ、ｚは”00”の値しかなく、同様に”10”の場合、"11"しかない。主鍵ｋと副鍵ｘのビット情報が”00”の場合、副鍵ｙ、ｚは"00"、"01"と”10”の３つ、同様に”11”の場合、”01”、”10”、”11”の３つの値が存在する（ゆらぎが存在する）。

f(0) = majorityFunc(1,0,0)

f(1) = majorityFunc(0,1,1)

f(0) = majorityFunc(0,0,0), majorityFunc(0,0,1), majorityFunc(0,1,0)

f(1) = majorityFunc(1,0,1), majorityFunc(1,1,0), majorityFunc(1,1,1)

　主鍵ｋ、副鍵ｘに対応する、多数決論理を満たす副鍵ｙ、副鍵ｚの生成させる実験を行った。主鍵ｋと副鍵ｘのビット状態は、"00"、"01"、"10"、”11”の組み合わせで、先述の通り、"01"と”10”では副鍵ｙ、ｚは一意に決まり、"00"、”11”にはゆらぎがある。主鍵ｋ、副鍵ｘが"00"、”11”となる確率は0.5である。鍵長32ビット長で、パソコンの疑似乱数で実験（10000回のデータ発生）したところ、"00"、”11”の出現頻度は約0.3334であった。これは、32ビット長の中で約10個の割合で"00"、”11”が発生し、10の6乗個の異なる鍵が発生できることが分かった。鍵長を長くすると更に組み合わせも増え、秘密分散に多数決論理関数も十分利用可能と考えるに至った。本格的に利用する場合、鍵データの圧縮などの処理の実装が必要である。

**2.2　実装例**

　Javaによる多数決論理関数の実装例。メソッドgenerateKeyは、与えらた主鍵、副鍵に対応する任意の副鍵2個を発生させる、メソッドmajorityは、3個の副鍵の多数決論理演算の結果を求めるものである。

// ２つの鍵b1（主鍵）、b2（副鍵）から多数決論理関数に合致する２つの副鍵a2、a3を生成

public BitSet[ ] generateKey( BitSet b1, BitSet b2 ) {

BitSet a2 = new BitSet( ) ;

BitSet a3 = new BitSet( ) ;

for ( int i = 0; i < KEY\_LENGTH; i++ ) {

//b1(0)=majority(b2(1),a2(0),a3(0))

if ( !b1.get( i ) && b2.get( i ) ) {

a2.set( i, false ) ;

a3.set( i, false ) ;

}

//b1(1)=majority(b2(0),a2(1),a3(1))

else if ( b1.get( i ) && !b2.get( i ) ) {

a2.set( i, true ) ;

a3.set( i, true ) ;

}

//b1(0)=majority(b2(0),a2(0),a3(0)),

　　　majority(b2(0),a2(0),a3(1)), majority(b2(0),a2(1),a3(0))

else if ( !b1.get( i ) && !b2.get( i ) ) {

int v = (int)( Math.floor( Math.random( ) \* 3 ) ) ;

switch ( v ) {

case 0 :

a2.set( i, false ) ;

a3.set( i, false ) ;

break ;

case 1 :

a2.set( i, false ) ;

a3.set( i, true ) ;

break ;

case 2 :

a2.set( i, true ) ;

a3.set( i, false ) ;

break ;

}

}

//b1(1)=majority(b2(1),a2(0),a3(1)),

majority(b2(1),a2(1),a3(0)), majority(b2(1),a2(1),a3(1))

else if ( b1.get( i ) && b2.get( i ) ) {

int v = (int)( Math.floor( Math.random( ) \* 3 ) ) ;

switch ( v ) {

case 0 :

a2.set( i, false ) ;

a3.set( i, true ) ;

break ;

case 1 :

a2.set( i, true ) ;

a3.set( i, false ) ;

break ;

case 2 :

a2.set( i, true ) ;

a3.set( i, true ) ;

break ;

}

}

}

BitSet x[ ] = new BitSet[ 2 ] ;

x[ 0 ] = a2 ;

x[ 1 ] = a3 ;

return x ;

}

// BitSetの多数決論理関数 majority(x, y, z) = ( (x&y) | (y&z) | (z&x) )

public BitSet majority( BitSet x, BitSet y, BitSet z ) {

BitSet b12 = (BitSet)x.clone( ) ;

b12.and ( y ) ;

BitSet b23 = (BitSet)y.clone( ) ;

b23.and ( z ) ;

BitSet b31 = (BitSet)z.clone( ) ;

b31.and ( x ) ;

b23.or( b31 ) ;

BitSet b123 = (BitSet)b12.clone( ) ;

b123.or( b23 ) ;

return b123 ;

}

**2.3　秘密分散の冗長化**

　この関数の応用例として、データｋを３つに分散したｘ、ｙ、ｚを用い、それぞれ２つづつ組み合わせたｘｙ、ｙｚ、ｚｘによる秘密分散を行う。それぞれを別のデータセンターに預け、データを保護する方式がある。この方式によれば、データ量は全体で元の６倍となるが、１つのデータセンターからデータが漏えいしても元のデータが復元できないこと、１つのデータセンターが地震で壊滅しても残りのデータセンターからデータを取り寄せれば元のデータが復元できること、など秘密分散の冗長化が可能となる。

**３．(k, n)しきい値秘密分散法**

一般に秘密分散と言えば、Shamirが開発したSecret Sharing Scheme がある。この方法は、(k, n)しきい値秘密分散法とも言われている。(k, n)しきい値秘密分散法は、秘密情報をn個の分散情報に分け，任意のk (≦n) 個集めると秘密情報が復元できると言う特徴、分散情報の一部が漏えいしても元の秘密情報の復元はできないと言う特徴を持つ。

この秘密分散法について、大変分かり易い解説がネットにあり、こちらを参照ください。

しきい値秘密分散法(SSS/SSSS : Shamir's Threshold Secret Sharing Scheme)

<http://www.atmarkit.co.jp/fsecurity/special/53tsss/tsss02.html>

**3.1　HTMLによる(k, n)しきい値秘密分散法の実装サンプル**

　次のサンプルはHTML＋Javascriptで手軽に体験できる。Secure Sharesのテキストボックスに分散する数分だけ分散情報が表示される。この分散情報を編集し、閾値前後のデータ量にして、復号できるか否かを確認してください。

Shamir Secure Secret Sharing

<http://www.rlr-uk.com/tools/SecSplit/SecureSplit.aspx>

<http://127.0.0.1/sss/sss.html>(ダウンロードした日本語バージョン)

**3.2　node.jsによる(k, n)しきい値秘密分散法の実装サンプル**

node.jsの実装では、ライブラリamper5and / secrets.js（<https://github.com/amper5and/secrets.js/>）を利用した。”secrets.js”は優れたライブラリで、(k, n)しきい値秘密分散が簡単に実装できた。

**3.2.1　秘密分散と復号の概略コード**

//－－－－－秘密情報から分散情報の生成－－－－－

var secrets = require('./secrets');//secret.jsのインスタンス生成

//秘密情報をhex stringに変換

var pwHex = secrets.str2hex(pw);

//sssによる分散情報の作成

var shares = secrets.share(pwHex, sh, th);//sh:シェア数、th:閾値

//分散情報のファイル書き込み

for(var i=0;i<sh;i++){

fileSaveContents( "./s"+i+".txt" , shares[i] );

}

//－－－－－分散情報の復号－－－－－

var shares=new Array();

for(var i=2;i<argv.length;i++){

var d=fileGetContents( argv[i] );

shares.push(d);

}

//読み込んだん分散情報からsssオブジェクトを生成

var newShare = secrets.newShare(1, shares);

//分散情報からテキスト情報の復号

var comb = secrets.combine( shares );

//hex stringをテキストに変換

comb = secrets.hex2str(comb);

console.log( "秘密情報は\n"+comb );

**3.2.2　実行結果**

●分散情報の生成

C:\node\v0.8.23\test>node sssEncode.js 6 2 "(k, n)しきい値秘密分散法は、秘密情報をn個の分散情報に分け，任意のk (≦n) 個集めると秘密情報が復元できると言う特徴、分散情報の一部が漏えいしても元の秘密情報の復元はできないと言う特徴を持つ。"

share=6

threshold=2

(k, n)しきい値秘密分散法は、秘密情報をn個の分散情報に分け，任意のk (≦n) 個集めると秘密情報が復元できると言う特徴、分散情報の一部が漏えいしても元の秘密情報の復元はできないと言う特徴を持つ。

30023064630130925fb4727930468a0030683044306a304d3067306f51435fa9306e583160c55bc679d8306e5143308230663057304430486f0f304c90e84e00306e583160c56563520630015fb4727930468a003068308b304d306751435fa9304c583160c55bc679d83068308b308196c6500b00200029006e226600280020006b306e610f4efbff0c30515206306b583160c565635206306e500b006e3092583160c55bc679d83001306f6cd5656352065bc679d850243044304d30570029006e0020002c006b0028

秘密情報をシェア6、閾値2で、分散保存しました。

●秘密情報の復号

C:\node\v0.8.23\test>node sssDecode.js s0.txt s5.txt

秘密情報は

(k, n)しきい値秘密分散法は、秘密情報をn個の分散情報に分け，任意のk (≦n) 個集めると秘密情報が復元できると言う特徴、分散情報の一部が漏えいしても元の秘密情報の復元はできないと言う特徴を持つ。

●分散情報s0.txtの内容

C:\node\v0.8.23\test>type s0.txt

801446cb3c72248e26fd7c1c56c7ed71131d510d761b14135a97574962d3abd74f829507685887feba002feaa2b402369b3bd7059cf4460dad1c54caf225d0e16fcc37c14c6ca8d34044682c6f32bea38fcf6c11b2be55cc96f9deb65c5e7f25a6424b74ee516f8437324eb8047e445d2319c8d9ab2cf277bccb6b79074eca7ee3b9ce23dec0db0cd8b0f01120601ac700d3b771dcde6c2d25bf89a7c73165ed9e31387d39f6d7e4c8631d580c431fc2a6b707125a7ed180b72dd198d0b6a2d151be8f0c5869eb0748dc29b0e

●閾値に満たない場合、秘密情報が正しく復号できない。

C:\node\v0.8.23\test>node sssDecode.js s0.txt

秘密情報は

鬎跂끴蚞ᯨⴕ୪ᦍ狝᠋꟭焥歰ﰪ쐱햀蘱繌齭蟓廙猖驼寸싒췦眝഻거؁Ē謏냍㮜꟮瓬랐

䟤猤띎搤엧澝峉⯥섛ﳶ苆ц贴웊簔ﳃถ≝䲯퇅惚콄灙뎽⍩⭀ﺪꀂ翫薈偶뵴

汾쇅濗䣢윢河

●s5.txtの内容を編集した場合、秘密情報が正しく復号できない。

C:\node\v0.8.23\test>node sssDecode.js s0.txt s5.txt

秘密情報は

䳦䝍껺嵁⣡땇싵䘡卓鞫븑퉢咛ﮡ먽蝨礷Ɫ砎ӊ㗅ᦦ渱췝⬡猎榸堲쑎炈휀넢⪍

顀๐程璜᪦늲枛蔙耳덆⸆ᆿ霭ꨜ᠝뺖಻鮂끎མ遢೴镂蚖禎ᾛ褷悔躂⸕鿡돻Ⅻ힡뉮黝໴芶≛똝

蕓ꡇ傫㢕碎

**3.2.3　ソースプログラム**

sssEncode.js(<http://127.0.0.1/sss/sssEncode.js>)

sssDecode.js(<http://127.0.0.1/sss/sssDecode.js>)

作成　大坂　哲司