多重アファイン鍵暗号

鈴木 秀一

東京電機大学

〒 270-1382 千葉県印西市武西学園台 2-1200

ssuzuki@chiba.dendai.ac.jp

概 要

数論的な代数幾何学は、既に有効に暗号理論に活用されている。筆者は数論的でない代数幾何学 (双有理幾何学) を暗号理論に活用することをしばらく研究していた。その副産物として多項式写像を活用した秘密鍵暗号を考案した。この暗号の安全性について検討した結果、この多項式写像は、アファインであっても充分安全であることが分かってきた。この暗号を高速化していくことによって、今回の「多重アファイン鍵暗号」にたどり着いた。本暗号はストリーム暗号である。技術的な核心は、非常に周期の長い乱数の系列をほとんど無数に生成するアルゴリズムである。この暗号の特徴として、アファイン写像であるため、1 バイトあたり整数の掛け算を 0.5 回程度しか実行しないことが挙げられる。したがって非常に高速である。パソコン (Pentium III 600MHZ, Delphi 5.0 のみで記述されたプログラム) 上で 280 MBits/Sec 以上の暗号化速度をソフトウェアのみで実現できる。平成 11 年 5 月に暗号理論の専門家たちに本暗号を公表したが、今のところ解読の手がかりは見出されていない。

キーワード

ストリーム暗号, 多重アファイン鍵システム, 一方向関数, 使い捨て署名, デジタル署名, マスタキー.

1 多重アファイン鍵に至るまで

筆者は、[1] などを参考に、射影空間の双有理写像を具体的に構成することを調べた。例えば以下のような公開鍵暗号を構成してみた。

$p,\ q$ を十分大きな素数とする。$R = \mathbf{Z}/(n) \quad (n = pq)$ として $\mathbf{Z}/(p)$ 上のアファイン空間 $\mathbf{A}^m$ の上の双有理写像を整数係数多項式写像

$$\mu : \mathbf{A}^m \longrightarrow \mathbf{A}^m$$

で構成する。任意の多項式写像 $g : \mathbf{A}^m \longrightarrow \mathbf{A}^m$ を用いて $f(x) = \mu(x) + p\ g(x) \quad (x \in \mathbf{A}^m)$ で多項式写像を f を定義する。

公開鍵 n, 多項式写像 $f(x)$

秘密鍵 $p,\ q,\ \mu^{-1}(x)$

ここで、f は双有理写像でないので逆写像を構成できない。$p > q$ のとき $\sqrt{n}$ より小さな非負整数で平文を構成し、暗号文を $f(x) \mod n$ で計算することで素因数分解の困難さを根拠とする暗号を構成してみた。この暗号は n が 1000 桁程度の場合、RSA 暗号に比較して 300 倍以上高速であった。

しかしこの暗号は $f(x)$ の係数をたくさん公開するので、この情報から簡単に n を因数分解できてしまうという欠点があった。そこでこの速度を生かして秘密鍵暗号を構成することを考えた。当時、インターネットなどにカオス暗号という新しいストリーム暗号がアナウンスされていた。この暗号の詳細は分からなかったが、実数演算を用いることは確実であり、整数係数の多項式写像で、この暗号より高速で安全な暗号を構成できると直感した。

2　カオス暗号と相互作用する多項式写像暗号

筆者が推測するカオス暗号は、おおむね次のようなものである。実数 x_n に対して、

$$x_{n+1} = ax_n^2 + bx_n + c$$

という 2 次式で実数列 $\{x_n\}$ を計算し、平文 $\{m_n\}$ に対して暗号文 $\{c_n\}$ を

$$c_n = m_n \oplus [x_n] \quad ([x_n] \text{は } x_n \text{の整数部分})$$

のように計算して暗号化するストリーム暗号であろう。ここで $\oplus$ は排他的論理和を表す。筆者は、この暗号は実数演算を使用するので、より高速な暗号が必ず存在するはずであると感じた。整数係数の多項式写像を使って擬似乱数を生成し、ストリーム暗号を構成するのであるから、一次式で安全なものを構成すればカオス暗号より高速になる。カオス暗号の安全性は実数演算から一部をマスクして整数を取り出すところにある。例えば 16 ビット符号無し整数を用いて

$$x_{n+1} = ax_n + b$$

で整数の擬似乱数 $\{x_n\}$ を生成する Lehmer 法を考える。安全性のために x_n の半分をマスクして 8 ビットの擬似乱数を使ってストリーム暗号を構成する場合、4 バイト以上暗号化すると未知数 $a,\ b$ が特定される可能性が出てくる。そこで 3 回この式を使用したら、別の式に変更する必要が出てくる。そこで、複数の一次式を準備して、係数が特定される危険性のある式を別の一次式で書き換えてしまうアイディア生まれた。すなわち、寿命を持った複数の一次式と、その一次式の間の相互作用を定義すれば、高速で安全な暗号を構成できると考えた。ここで、暗号化に使用した 8 ビットの乱数の残りの部分を鍵選択に活用すると、暗号の外部からは、使用された鍵の番号をまったく特定できなくなる。このようにして、アファイン関数だけでも安全な暗号を構成できる、という構想が固まった。

3　多重アファイン鍵システム

ここで多重アファイン鍵システムの定義を述べる。

3.1　アファイン鍵

アファイン鍵とは有限環の元や整数を成分とする次のような構造体 K である。

$$K = \{a,\ b,\ c,\ n\}$$

ここで、$a,\ b$ はアファイン関数の係数、c はこの鍵が何回使用されたかをカウントするカウンタ、n はこのアファイン鍵を使用できる回数の上限、すなわちアファイン鍵の寿命である。

整数または有限環の元 x に対するアファイン鍵 K の作用を $K(x) = ax + b$ で定義する。

3.2 多重アファイン鍵システム

多重アファイン鍵システムとは、複数のアファイン鍵 $\{K[i]\}$ と、そのアファイン鍵同士の間の相互作用を定義する鍵書き換えのプロシージャ $w(i,j)$ の組

$$\mathcal{K} = \{\{K[i]\}, w(i,j)\}$$

である。ここで、$w(i,j)$ は $K[i]$ を $K[j]$ で書き換えることを意味する。これはどのようなものであってもよいので無数の多重アファイン鍵システムを構成できる。最も単純なものは**基本型**と言われる以下のものである。

```
procedure w(i,j:integer):integer;
begin
   K[i].a:=K[i].a*K[j].a+K[j].b;
   K[i].b:=K[i].b*K[j].a+K[j].b;
end;
```

実は、この基本型には、後述するように、問題点があり、実際に暗号化に使用されることは無い。しかし、このような鍵書き換えのプロシージャを一つ準備して、以下のアルゴリズムで多重アファイン鍵暗号が構成される。

3.3 多重アファイン鍵暗号のアルゴリズム

以下では、32 ビットの符号無し整数を用いた 32 個のアファイン鍵からなる多重アファイン鍵システム $\{\{K[i]\}, w(i,j)\}$ と初期値 x_0 を用いて、平文 $\{m_k\}$ を暗号化し暗号文 $\{c_k\}$ を得るものとする。

1. `i=0, k=1`
2. `x0=K[i](x0) , K[i].c=K[i].c+1`
3. `c[k]=m[k] xor (x0 and 65535)` (encription)
4. `j=(x0 shr 16) and 31`
5. `if K[i].c` $\geq$ `K[i].n then w(i,j) , K[i].c=0`
6. `i=j, k=k+1 goto 2.`

アルゴリズムを直感的に説明すると以下のようになる。32 ビット符号無し整数の初期値 x_0 から、繰り返しアファイン鍵を作用させて 32 ビット符号無し整数の擬似乱数 $\{x_k\}$ を生成する。この x_k の下位 16 ビットを使用して暗号化を行う。すると擬似乱数の上位 16 ビットが使用されずに残っている。もちろんこれは無駄ではない。暗号の安全性のためには十分役に立っている。この部分は暗号化に関わらないので、暗号の外部から見ることができない。そこでこれを使って、秘密裏に鍵の番号を変更するのである。擬似乱数の上半分と下半分には通常まったく相関が無いので、鍵の番号を特定することができない。アファイン鍵の寿命が尽きた場合にも、この情報を使って鍵を書き換えるのである。生成された乱数から、アファイン鍵の係数を未知数と見て連立方程式を立てたとしても、特定不可能な鍵によって別のものに書き換えられてしまう。ある程度良質な乱数が生成されれば、本暗号を解読することは非常に困難であろうと考えられる。しかも本暗号は単なる整数係数の一次式の計算のみで構成されているので非常に高速である。

解読不可能なことが証明できる唯一の暗号として知られる Vernam 暗号は真正乱数 $\{r_k\}$ を用いて

$$c_k = m_k \oplus r_k$$

として定義されることを思い出すと、ここで生成される擬似乱数が十分良質であれば本暗号は解読できない。したがって、本暗号の安全性の全ての議論は、このシステムで生成される擬似乱数がどのようなものであるか、という問題になる。

3.4 基本型の問題点

著者が山形大学での ISEC で多重アファイン鍵暗号に関する講演した際、基本形のアルゴリズムを説明した時点で時間切れとなった。[3] には高速形として次節に示す鍵書き換えプロシージャも記載されている。このアルゴリズムは発表当初から公表しているものである。基本形を高速化する目的のほかに、これを考案した動機として、基本形には重要な問題点があったことをここに述べておく。

基本形の多重アファイン鍵システムで擬似乱数を生成していくと、かなり高い確率で多重アファイン鍵の係数が 0 になり、0 ばかりを出力する事態に陥る。すなわち、基本形には弱鍵がかなり高い割合で存在する、という事実がある。この傾向は以下の高速形では、かなり改善されている。鍵書き換えにわざわざ if 文を導入したのは、この問題に対する対策であったが、筆者は、迂闊にもこの事情の説明を怠った。[7] では、この点が攻撃された。後述するように、多重アファイン鍵暗号の弱鍵は、現在では原理的にもほぼ排除できたと考えられる。

3.5 いろいろな鍵書き換えのプロシージャモデル

ここでは、多重アファイン鍵システムの本質的な部分として、アファイン鍵の間の鍵書き換えのプロシージャモデルをいくつか定義する。ここでは、鍵の個数を 64 個とするが、実際には鍵の個数はいくつでもかまわない。ここで便宜上、$p = 65521,\ q = 2^{15} - 1 = 32767$ であるとするが、別の値でもかまわない。多重アファイン鍵を $K = \{K[0], K[1], ..., K[63]\}$ とする。

3.5.1 高速型 1：

基本形で、$K[i].b$ の書き換えを `K[i].b:=(K[j].a*K[i].b+K[j].b) and q;` のように変更したものである。`mod` 演算が高速な `and` 演算に置き換わる。値 b の範囲が 1 ビット程度狭くなるが、生成される乱数には本質的な変化はないと考えられる。

3.5.2 高速型 2：

乱数を生成するときに、アファイン鍵の数列への作用を

$$x_{n+1} = (K[i](x_n) = K[i].a * x_n + K[i].b) \bmod p$$

とするところを

$$x_{n+1} = (K[i](x_n) = K[i].a * x_n + K[i].b) \text{ and } q$$

とするとさらに高速化される。ここで、鍵の相互作用を以下のように変更する場合もある。17 は素数 $p = 65521$ に対する原始根である。他の原始根でもその効果は同様である。$K[i]$ を書き換えた直後、

```
if K[i].b=0 then
begin
  K[i].b:=1;
```

```
    if K[i].a=0 then
      K[i].a:=17;
  end;
```

ここで、K[i].a の書き換えのような係数の調整を行う。係数変更の分遅くなるが、乱数生成が高速化されるので、結果的にこの方式は高速である。

3.5.3 安全形：

上記では、あるアファイン鍵を、別の一つのアファイン鍵で書き換えているが、この部分の安全性を高めるために、以下のように二つ以上の鍵を用いて書き換えることも出来る。

```
        K[i].a:=(K[j].a*K[i].a+K[j].b) mod p;
        inc(j);
（または j:=j+c; （ c は多重アファイン鍵の値から適当に定められる整数））
        j:=j and 63;
        K[i].b:=(K[j].a*K[i].b+K[j].b) and q;
(または K[i].b:=(K[j].a*K[i].b+K[j].b) mod p;)
```

さらに、安全性を高める目的で、一部のアファイン鍵 K の数列への作用を

```
K(x)=(x*(x+K.a)+K.b) mod p;
```

のように、2 次式以上の多項式によって計算することも可能である。本暗号では、このような高次式が部分的に混入していていもほとんど実質的に暗号化速度が低下しないのも大きな特徴である。さらにアファイン鍵の個数を増やすと安全性が増加するが、これでも実行速度は低下しない。

ここで説明した多重アファイン鍵暗号は、最も高速な場合がたまたま複数のアファイン写像で実現できるのであって、相互作用する複数の多項式写像によっても同様の暗号を構成できることは明白である。良質な乱数を生成する目的で、多重多項式写像システムが今後研究される可能性は残されている。

3.6 沖縄モデル

このモデルは 2000 年 1 月 25 日の沖縄日韓合同情報セキュリティシンポジュームで発表された。32 ビットの符号無し整数の演算を用いて 2 バイトずつ暗号化し、mod も使用しないので、かなり高速である。このモデルでは整数のアファイン演算 $ax+b$ を以下のようなような意味で計算する。この演算を**安定化演算** (**stable multiplication**) ということにする。

```
a*x+b=((a*x+b shr 16)) xor (a*x+b)
```

鍵の書き換えのとき、アファイン鍵の係数がともに偶数であると、この鍵は偶数のみ出力するので、次第に鍵や生成される乱数の最下位ビットが 0 になる確率が高くなる。この効果が累積されて、鍵が下のほうから次第に 0 になっていく傾向がある。生成される乱数の上位半分は 0 になり難い。これを下半分に排他的論理和してこの傾向を拡散させるわけである。この演算を用いて鍵書き換えのプロシージャ $w(i,j)$ を以下のように記す。

```
procedure w(i,j:integer):integer;
begin
```

```
    K[i].a:=K[i].a*K[j].a+K[j].b;
    j:=(j+1) and 31;
    K[i].b:=K[i].b*K[j].a+K[j].b;
end;
```

ここでの計算は安定化演算を用いるものとする。沖縄モデルは、実験的に弱鍵が少なく、安定して長い周期の擬似乱数を生成する。しかし、初期値と初めの 6 個程度のアファイン鍵の成分がすべて 0 であるようなタイプの鍵は弱鍵になる。しかし、自明でない弱鍵を見出すのは非常に困難であり、実用的にはこのモデルが使用されるのではないかと考えられる。

4 多重アファイン鍵暗号の弱鍵とそれを排除する方法

沖縄モデルには弱鍵見出された。この点を改良できるであろうか。アファイン鍵の係数が 0 になると弱鍵が出現しやすい傾向がある。そこで前節のプロシージャで、鍵が強制的に 0 にならないように

```
procedure w(i,j:integer):integer;
begin
    K[i].a:=(K[i].a*K[j].a+K[j].b) or 2;
    j:=(j+1) and 31;
    K[i].b:=(K[i].b*K[j].a+K[j].b) or 1;
end;
```

のようにする対策も考えられる。このタイプの鍵書き換えを使用した場合、弱鍵を見出すことは大変難しい。短い周期の擬似乱数生成に陥る鍵は今のところ見出せない。

しかし、もっと本質的に弱鍵を排除する方法がある。多重アファイン鍵暗号の弱鍵は、一つ (または 2, 3 個程度の少数) の鍵のみを連続的に使用する状況で発生する。安定して自明でない擬似乱数を生成している状況では、すべての鍵が一様に使用されている。そこで少数の鍵のみが使用される状況を排除すれば弱鍵を排除できる。例えば大域的に鍵回転変数　`vkey:integer;` を確保し、鍵書き換えの直前に、次に使用する鍵の番号 `j` を強制的に全ての鍵を使用するようにするのである。

1. `i=0, k=1, vkey=0`
2. `x0=K[i](x0) , K[i].c=K[i].c+1`
3. `c[k]=m[k] xor (x0 and 65535)` (encription)
4. `j=(x0 shr 16) and 31`
5. `if K[i].c` $\geq$ `K[i].n then vkey=(vkey+1) and 31, j=(j+vkey) and 31, w(i,j) , K[i].c=0`
6. `i=j, k=k+1 goto 2.`

このようにすると、弱鍵でない通常状態では `vkey` の影響はほとんど無い。しかし、少数の鍵のみを使用する弱鍵に近い疑わしい状況になると、強制的に全体の鍵を使うように仕向けられる。多数のアファイン鍵が同時に弱鍵になる確率は極めて小さい。

5 多重アファイン鍵システムが生成する擬似乱数の周期

本節では、前節で扱った形式の多重アファイン鍵システムが生成する擬似乱数の周期を計測してみる。もとより、多重アファイン鍵システムによって生成される擬似乱数の周期は非常に長いので、鍵

の個数は 4 個程度、鍵の係数も 4〜12 ビット程度のミニチュア版を用いて計測してみた。この表の鍵の成分の最初の値は乱数生成の初期値である。

鍵の個数 n	鍵の長さ L bits	鍵の成分	周期 p	$p^{1/nL}$
3	4	9,8,14,4,12,7,3	13165	2.2
4	4	14,11,10,7,4,3,13,11,4	56693	1.98
3	4	7,7,3,12,11,3,9	12971	2.2
2	10	725,998,779,252,152	2251617	2.08
5	4	12,13,6,12,8,13,6,6,12,11,9	359431	1.9
2	12	2420,3316,2463,284,1109	131994432	2.18
4	6	16,51,50,29,55,48,14,37,1	51487411	2.1
4	6	8,59,62,42,31,24,48,21,18	11512780	1.97
3	8	251,42,183,236,7,40,119	152418175	2.19
2	8	126,65,143,8,142	268091	2.18
6	4	6,7,10,12,13,1,3,3,3,14,6,8	83674821	2.14

表 1:

表中の $p^{1/nL}$ の値はかなり安定した値をとる。多数の周期の可能性があることと、多重アファイン鍵の個数と係数の長さは生成される擬似乱数の周期に指数的に影響することが読み取れる。このことから、実用的な鍵のサイズとしてアファイン鍵の個数 32 個、係数は 32 ビットの符号無し整数を用いた場合、生成される擬似乱数の周期 p は $p > 2^{32\times32} = 1.798 \times 10^{308}$ となると予想できる。これは暗号用の擬似乱数としては十分な長さである。M 系列の乱数のように長周期の擬似乱数生成法も知られている。しかし多重アファイン鍵システムによる擬似乱数の生成法はランダムな鍵を用いても、安定して長周期の擬似乱数を生成する点に特徴がある。このような性質は、暗号用の擬似乱数として望ましいと考えられる。

6　デジタル署名とマスタキー

多重アファイン鍵システムを用いると、非常に多くの乱数を共有できる。多重アファイン鍵システムによって多重アファイン鍵を生成すると、一方向関数を構成できる。この一方向関数を用いて Lamport の使い捨てデジタル署名を構成できるが、この署名は署名ファイルが巨大化するという欠点を持っていた。そこで、署名の秘密鍵の部分を一つの多重アファイン鍵から組織的に生成する工夫をした。これがマスタキーの原形となった。一つの多重アファイン鍵から無数の多重アファイン鍵を系統的に生成し、樹形のダイアグラム (**鍵位相　key topology**) で下位の鍵を管理できる。すなわちマスタキーの概念が成立する。上位の鍵を持つ者は、下位の鍵による暗号をすべて解読できる。マスタキーという言葉は以前から使用されていたが、それは単にセッションキーを生成するもとの鍵を意味していた。ここで導入されたマスタキーを使用するメリットは、地球規模のネットワークの**セキュリティを自己組織化できる**点にある。管理者は以下のことを行うだけである。

1. マスタキーとなる多重アフィン鍵を生成する。
2. いくつかの子鍵を生成し、その鍵位相を公開する。
3. 子鍵を配布する。

以下、ユーザ (下位の管理者) は子鍵を受け取ったら、これを用いて、上位の管理者と同様に子鍵の生成と鍵位相の公開を再帰的に繰り返す。

これだけの手間で、自律的に地球規模のネットワークのセキュリティを組織化できる。地球規模のネットワークを一元管理できる可能性は、このような単純化された手法でなければ実現できない。

多重アファイン鍵システムを使用して、本質的に異なる二つの一方向関数を構成する。つまり、擬似乱数を生成していく過程で、多重アファイン鍵をそのまま使用し続ける場合と、多重アファイン鍵が生成する乱数を新しい多重アファイン鍵と見なして乗り換える、という 2 種類である。この結果、Lamport の使い捨て署名と、鍵を生成する情報 (鍵位相) を公開しても安全なマスタキーを構成できる。この方式は鍵供託の問題の一つの解答にもなっていると考えられる。また、大きな組織で VPN などを使用しているとき、その構成員が退職したりする場合にも速やかに混乱なく業務を引き継ぐことが可能になる。鍵を紛失しても簡単に再発行できる。また、上位の管理者と信頼関係があれば、異なる鍵を使用するユーザ間でも流暢な通信が可能である。上位の管理者は、下位のユーザの通信権を自由にコントロールできる。

ここで定義したマスタキーは、多重アファイン鍵システムとは独立な概念であるが、多重アファイン鍵システムを用いると実例を構成することができる。

6.1 Lamport の使い捨て署名

Lamport は任意の一方向関数 $f : X \longrightarrow Y$ を用いて, 使い捨て署名を構成できることを示した [4]。これは以下のようなものである。

$x_{00}, x_{01}, x_{10}, x_{11}, \cdots \in X$ に対して $y_{00}, y_{01}, y_{10}, y_{11}, \cdots \in Y \quad (y_{ij} = f(x_{ij}))$ とし、この f と $\{y_{ij}\}$ を公開する。$K = (x_{00}, x_{01}, x_{10}, x_{11}, \cdots)$ とする。

メッセージ M をビット列 $\{M_0 M_1 M_2 \cdots\}$ で表し、これに次のような署名を添付する。

$$(M, sig_K(M))$$

ただし

$$sig_K(M_0, M_1, M_2, \cdots) = (x_{0M_0}, x_{1M_1}, x_{2M_2}, \cdots)$$

この署名は原則的には一回しか使用できないが、署名の確認は何回でもできる。

6.2 多重アファイン鍵システムによる一方向関数と使い捨て署名

多重アファイン鍵システム [2] を用いて、一方向関数を定義する。多重アファイン鍵システムの全体を Ω とし、$K \in \Omega,\ K = \{K[i], w(i,j)\}$ とする。K の生成する擬似乱数を $\{x_0, x_1, x_2, \cdots\}$ とする。このとき、適当な正整数 v を用いて

$$f(K) = [K]_v = (x_v, x_{v+1}, x_{v+2}, \cdots, x_{v+N-1}) \quad (N \text{ は多重アファイン鍵のサイズ})$$

多重アファイン鍵システムから生成された擬似乱数から、もとの多重アファイン鍵を特定することは困難であろう、と考えるならば、我々は $(f(K), w(i,j))$ を新たに多重アファイン鍵とみなすことによって、一方向関数

$$f : \Omega \longrightarrow \Omega$$

を構成することができる。記号 $[K]_v$ を K の v 切片ということにする。v 切片を用いて Lamport の使い捨て署名を構成することができる。この署名は高速であるが、署名ファイルが大きくなるという欠点も持つ。1 ビットの署名が 516 バイト程度になる。しかし公開しないデータ $(x_{00}, x_{01}, x_{10}, x_{11}, \cdots)$ は 258 バイト程度に押さえることが可能である。これは一つの多重アファイン鍵から組織的に多くの多重アファイン鍵を生成することで実現できるからである。このアイディアから以下のマスタキーの概念が出てくる。

6.3 マスタキーの概念

鍵空間 Ω は十分大きい集合であるとする。更にこの Ω から自分自身への複数の一方向関数

$$f_i : \Omega \longrightarrow \Omega \quad (i = 0, 1, 2, \cdots, m-1)$$

があるとする。

このとき、任意の $K \in \Omega$ と $f = \{f_i\}$ に対して (K, f) を**マスタキー**ということにする。$K,\ H \in \Omega$ に対して $H = f_{\epsilon_{k-1}}(f_{\epsilon_{k-2}}(\cdots f_{\epsilon_1}(f_{\epsilon_0}(K))))$ のとき

$$top(H) = (\epsilon_{k-1}, \cdots, \epsilon_1, \epsilon_0)_{m-addic} \quad (m \text{ 進整数})$$

とし、これを鍵 H の鍵位相 (key topology) とここでは言っておく。逆に鍵位相 m が $(\epsilon_{k-1}, \cdots, \epsilon_1, \epsilon_0)$ という m 進数表示を持つとき、$m(K) = f_{\epsilon_{k-1}}(f_{\epsilon_{k-2}}(\cdots f_{\epsilon_1}(f_{\epsilon_0}(K))))$ と定義することで鍵を構成できる。一方向関数なので鍵位相を公開しても安全である。

マスタキー (K, f) は、ある鍵位相 m に対して、$m \neq n$ かつ $m(K) = n(K)$ となる鍵位相 n を見つけることが計算量的に困難であるとき、**弱い意味で衝突なし**ということにする。同様に、マスタキー (K, f) は、任意の鍵位相 m に対して、$m \neq n$ かつ $m(K) = n(K)$ となる鍵位相 n を見つけることが計算量的に困難であるとき、**強い意味で衝突なし**ということにする。

強い意味で衝突なしのマスタキー (K, f) が構成できたとすると、世界中に (K, f) から生成された鍵を配布し、鍵位相を公開して組織的に鍵を管理できる。鍵を紛失したり変更したりする場合も混乱なく対応できる。弱い意味での衝突なしのマスタキーが構成できた場合でも鍵の一元管理ができる。

組織的に鍵を管理する目的でマスタキーまたはマスタ鍵という名称の概念が [5], [6] には記述されている。鍵位相のような情報を公開する方式は新しいのではないかと著者は考えている。

6.4 多重アファイン鍵システムによるマスタキー

多重アファイン鍵の全体を Ω とする。Ω には複数の一方向関数が定義できる。$K \in \Omega$ に対して、K の生成する擬似乱数を $x_0, x_1, x_2, \cdots$ とする。

$$f_0(K) = x_0, x_1, x_2, \cdots, x_{v-1} \text{を生成した時点での多重アファイン鍵 } K$$

また、

$$f_1(K) = [K]_v (\text{すなわち生成した擬似乱数を新しい多重アファイン鍵として乗り換える。})$$

ここで、正整数 v の値は適宜設定する。実験的に見て、アファイン鍵の個数をある程度以上たくさん使用した場合、これらの二つの一方向関数で構成したマスタキー $(K, f),\ (f = (f_0, f_1))$ は強い意味で衝突なしであると考えられる。その理由は以下のようである。

- 多重アファイン鍵システムは、その係数や初期値に対して極めて敏感に反応する。任意の 2 進整数 n によって $n(K)$ は Ω 上に均等に分布すると予想できる。
- 32 ビット整数を用いて、多重アファイン鍵の個数が 64 個程度の場合、Ω は 10^{1233} 個程度の元を持つ。
- この場合、誕生日攻撃によって衝突が起こる確率が 10^{-600} になるためには、多重アファイン鍵を 4.57×10^{316} 個以上使用しなければならない。

すなわち、516 バイト程度の多重アファイン鍵一つで、世界中の鍵を組織的に一元管理できると考えられる。

参考文献

[1] Hudson, H.:Cremona Transformation, 1911,Cambridge UP.

[2] Suzuki, Shuichi: The hash function using the multi-affine key system , JWISC 2000, Okinawa,2000.

[3] Suzuki, Shuichi: The cryptography using the multi-affine key system (In Japanese.), technical report of IEICE, ISEC 1999-32, Yamagata University 1999.

[4] Stinson, Douglas: Cryptography:Theory and practice, CRC press, inc, USA, 1995.

[5] 辻井重男, 笠原正雄: 暗号と情報セキュリティ, 昭晃堂, 1990.

[6] 松井 甲子雄: コンピュータのための暗号組立法入門, 森北出版, 1986.

[7] 盛合、宮野、下山: 「多重アファイン鍵暗号について」, SCIS2000 技術報告集 (沖縄), 2000.