SCIS 2014 The 31st Symposium on
Cryptography and Information Security
Kagoshima, Japan, Jan. 21 - 24, 2014
The Institute of Electronics,
Information and Communication Engineers

タイプ 1 の双線形写像群の暗号方式からタイプ 3 への変換アルゴリズムの実装

Implementation of Automated Translation for Schemes on Symmetric Bilinear Groups

丹後偉也* 阿部正幸 * † 岡本龍明 * †
TangoTakeya AbeMasayuki OkamotoTatsuaki

あらまし 双線形写像群には 3 つのタイプがあり，そのうちのタイプ 1 は構造が単純なため容易に暗号を構成できるという利点がある．しかし，最近の離散対数問題の進展により，セキュリティと効率が落ちてしまった．対してタイプ 3 は効率的な方式を構成できるが，全ての演算が正しく計算できるように設計者は二つの群上の要素の展開に注意を払わなければいけないため，構成が複雑になってしまうという問題がある．そこで，タイプ 1 の群で構成された暗号方式をタイプ 3 の群に変換するアルゴリズムが提案された．本論文では，より効率的に変換できるアルゴリズムについて考察し，実装した．これにより，既存のタイプ 1 の暗号方式を変換することによって効率的な暗号方式に変換でき，また，双線形写像に基づく暗号方式の設計者はタイプ 1 の暗号方式を構成してからこのアルゴリズムにより変換することでより暗号方式の設計が容易になった．

キーワード 双線形写像群，ペアリング，Bilinear Groups，Pairings

1 はじめに

1.1 背景

双線形写像群は，双線形写像の 3 つの群のことであり，三者間公開鍵共有法と ID ベース暗号が応用され，多くの暗号化方式で使われてきた．Galbraith，Paterson，Smart らは [1] で，この双線形写像群を 3 つのタイプに分類した．そのうちのタイプ 1 は $\mathbb{G} = \tilde{\mathbb{G}}$ で，タイプ 3 は $\mathbb{G} \neq \tilde{\mathbb{G}}$ であり，$\mathbb{G}, \tilde{\mathbb{G}}$ 間の効率的に計算できる同型写像が無い．現在では，楕円曲線上の双線形群を構築するためにタイプ 3 を用いることによって，他のタイプと較べて群要素のサイズは最小となり，効率が最大となる．しかし，タイプ 1 では群同士が対称的であるという単純な構造であるため，多くの暗号方式ではタイプ 1 が用いられている．

しかし，最近の離散対数問題が進展してきたことにより，タイプ 1 の双線形写像群ではより大きいビット数が求められるようになり，効率が悪くなってしまった．タイプ 3 ではその影響を受けないため，ビット数を抑えられる．そのため，ある目的のための方式がタイプ 1 で実現可能であることが示されている場合，同じ目的を実現できる方式をタイプ 3 で構成できる必要が有る．考えられるアプローチとして元の方式とは異なる手順で構成し直し，タイプ 3 で用いることができる仮定に帰着させるという方法がある．この場合 SXDH 仮定のような強力な仮定に基いてより効率的な方式を構成できる．例えばこのアプローチによる Waters09 暗号に対する変換を [2] が行っている．本論文で用いるもう一つのアプローチは，そのまま元の方式の手順を維持し，双線形写像群をタイプ 1 からタイプ 3 に変換させる方法である．例えばこの後者のアプローチによる Waters09 暗号に対する変換を [3] が行っている．この後者のアプローチの一般的な変換手順が未公開論文である [4] で提案された．

1.2 目的

本論文では，[4] の変換アルゴリズムに従って変換を行うプログラムの実装を行い，既にあるタイプ 1 の暗号方式をタイプ 3 へと効率的に変換させ，考察する．さらに，既存の変換アルゴリズムでは入力させる形式が暗号方式の記述に対して厳密すぎるという欠点を補い，柔軟性を持たせるため，変換アルゴリズムを改善させる．

* 京都大学大学院 情報学研究科 社会情報学専攻
† NTT セキュアプラットフォーム研究所

2　変換アルゴリズム

2.1　依存関係グラフによる変換アルゴリズム

[4] で述べられている，依存関係グラフによる変換アルゴリズムについて説明する．全ての変数を $\mathbb{G}, \hat{\mathbb{G}}$ で二重化すれば、タイプ 3 の群でタイプ 1 の群をシミュレートすることができるが、それには問題が 2 つある。1 つ目は，全ての演算が二重になるため、演算・空間ともに効率が悪くなることである．2 つ目は，ハッシュの出力を群要素へ乗せる演算を利用できないということである．これらの問題点を解決するために，依存関係グラフを使って必要なところだけを二重化するというアプローチを採る．

ある群を変換させた場合，その群を計算するために用いる群も変換させなければいけない．このように，変換させる際には，暗号方式内での群同士の依存関係について着目しなければいけない．そこで，群をノード，依存関係をエッジとした有向グラフによって表現する．例えば，ある群 G_1で群 G_2が導かれる場合，ノード G_1，ノード G_2があり，G_1から G_2へ向かうエッジがある．さらに，どの群がペアリングの計算に用いられるかの情報も必要となるので，ペアリングのノードとして $P_i[0], P_i[1]$ を追加し，計算に使われる群のノードからペアリングのノードへのエッジを追加する．i は定義順の番号であり，添字としての意味を持つ．以降，このペアリングのノードをペアリングノードと呼ぶ．また，暗号方式がある仮定に基いて構成されている場合，その仮定では群が二重化されると成立しない場合もある．そのことを考慮するためにも，仮定に依存する群に関して，仮定ノードを追加し，仮定ノードから群へのエッジを追加する．

以降では，この依存関係グラフをベースにして考えていく．基本アイデアとしてはこの依存関係グラフ $\mathcal{G}$ を，変換しない $\mathbb{G}$ 側のグラフ $\mathcal{G}_0$と，変換する$\hat{\mathbb{G}}$側のグラフ $\mathcal{G}_1$ に振り分けていく．ただし，ある群のノードをどちらかのグラフに入れると，そのノードの祖先にあたるノードも必ずそのグラフに必要となる．そうすれば，どちらのグラフにも同じノードが含まれる場合があり，その場合はそのノードの群に $\mathbb{G}$ も$\hat{\mathbb{G}}$も必要ということになる．これを群の二重化と呼ぶこととする．

ここで，アルゴリズム内で用いる演算子や関数の定義をする．二つのグラフ $\mathcal{G} = (V, E)$ と $\mathcal{G}' = (V', E')$ について，$\mathcal{G} \oplus \mathcal{G}'$ はグラフ $(V \cup V', E \cup E')$ になるとする．$\mathcal{G}'$が $\mathcal{G}$ のサブグラフであるとき，$\mathcal{G}' \subseteq \mathcal{G}$ と表記する.$\mathcal{G}$ のあるノード X に対して，$\mathsf{Anc}(\mathcal{G}, X)$ は，X へ到達する全てのパス上のエッジとノードで構成された $\mathcal{G}$ のサブグラフである．X 自身も $\mathsf{Anc}(\mathcal{G}, X)$ に含まれる．Eval 関数は各選択の二重化の度合いについて評価する．NoDup は群の二重化を禁止するノードの集合である．$\mathsf{bit}_i(a)$ は，a を 2 進数 としてみたときの i 番目のビットである．

この依存関係グラフを用いた変換によって，変換前の方式と変換後の方式が安全性に関して等価であることは，ある暗号方式の依存関係グラフ $\mathcal{G}$ を $\mathcal{G}_0 = (V_0, E_0)$ と $\mathcal{G}_1 = (V_1, E_1)$ に分けた後，以下の 4 条件を満たすということである．$P = \{P_0[0], P_0[1], \ldots, P_{n_p}[0], P_{n_p}[1]\}$ とする．この証明は [4] で行われている．

- $\mathcal{G}_0 \oplus \mathcal{G}_1 = \mathcal{G}$ である．
- それぞれの $i \in \{0, 1\}$ に対して
全ての $X \in V_i \setminus P$ は
$\mathsf{Anc}(\mathcal{G}, X) \subseteq \mathcal{G}_i$である．
- それぞれの $i \in \{0, \ldots, n_p\}$ に対して
$P_i[b] \in V_0$かつ $P_i[1-b] \in V_1$となる
$b \in \{0, 1\}$ が存在する．
- $\mathsf{NoDup} \cap V_0 \cap V_1 = \emptyset$ である．

二つのグラフに群を割り振っていく際，可能な限り二重化される群を減らす必要がある．例えば，$e(S_1, S_2)$, $e(S_3, S_4)$ のペアリングの計算があり，$S_2 := g^x, S_3 := g^y$ としたとき，もし S_1と S_3を変換させた場合，つまり S_2 と S_4をグラフ $\mathcal{G}_0$に入れ, S_1と S_3をグラフ $\mathcal{G}_1$に入れる場合, S_2は g に依存しているため g は $\mathcal{G}_0$に入れる必要があり, S_3も g に依存しているため, g は $\mathcal{G}_1$に入れる必要がある．よって, $\mathcal{G}_0$と $\mathcal{G}_1$に g が存在することになり，これは群の二重化が生じることを意味する．このように，効率的に変換するためには，暗号方式内での群同士の依存関係について注意し，変換する群を選択する必要がある．

さらに，変換するかどうかを選択するのはペアリングノードだけではない．ペアリングノードを選択していっても，どちらのグラフにも属さないノードが生じる可能性があるためである．このようなノードを取りこぼさないように，Bottom ノードを定義する．Bottom ノードとは，あるノードから他のあるノードへのエッジが無いノードである．ただしペアリングノードは除く．Bottom ノードをどちらのグラフに属するかによって，同様に群の二重化が生じる可能性があるため，Bottom ノードについても変換させるかどうかの選択を考慮する必要がある．

ペアリングの計算に用いられる 2 つの群のどちらを変換させるかという選択を考えると，ペアリングの数が増えるとその分選択パターンが増えていく．以下の [4] の変換アルゴリズム，FindSplit では，依存関係グラフと二重化禁止ノードの集合を受け取り，全ての選択パターンについてそれぞれ二重化の度合いについて評価し，最も二重化の度合いの小さい選択を出力するものとなっている．

Algorithm: FindSplit(G, NoDup)

1. 集合 L を初期化する．

2. Bottom ノードの集合を
$\{B_1, \ldots, B_{n_b}\} \subseteq \{X_1, \ldots, X_k\}$ とする．

3. $For\ \ell = 0, \ldots, 2^{n_p+n_b} - 1$ として
以下の手順を繰り返す．

   (a) $\mathcal{G}_0 = (V_0, E_0), \mathcal{G}_1 = (V_1, E_1)$ として初期化する．

   (b) $For\ i = 1, \ldots, n_p$ として，
   $\mathcal{G}_0 \leftarrow \mathcal{G}_0 \oplus \mathsf{Anc}(\mathcal{G}, P_i[\mathsf{bit}_i(\ell)])$
   $\mathcal{G}_1 \leftarrow \mathcal{G}_1 \oplus \mathsf{Anc}(\mathcal{G}, P_i[1 - \mathsf{bit}_i(\ell)])$ とする．

   (c) $For\ j = 1, \ldots, n_b,\ i = \mathsf{bit}_{n_p+j}(\ell)$ として，
   $\mathcal{G}_i \leftarrow \mathcal{G}_i \oplus \mathsf{Anc}(\mathcal{G}, B_j)$ とする．

   (d) $V_0 \cap V_1 \cap \mathsf{NoDup} = \emptyset$ なら
   $(\mathcal{G}_0, \mathcal{G}_1)$ を L に追加する．

4. $\mathsf{Eval}(L)$ を出力する．

3. で変化していく ℓ の値によってペアリングノードのどちらかが $\mathcal{G}_0$に分けられ，片方が $\mathcal{G}_1$ に割り振られるかが決定される．その際，そのノードの祖先となるノードも共に割り振られる．また，Bottom ノードも ℓ の値によって，$\mathcal{G}_0$に割り振られるのか, $\mathcal{G}_1$に割り振られるのかが決定される. 割り振るときは，既に割り振られたものに統合していく．全てのペアリングノードと Bottom ノードを割り振って統合した後は，(d) で $\mathcal{G}_0$と $\mathcal{G}_1$のノード，つまり V_0と V_1について積をとり，二重化してしまうノードの集合を求め，さらに二重化が禁止されているノードの集合 NoDup と積を取り，空集合になるかどうかをみることによって，二重化禁止されているノードが二重化してしまうかどうかを判定する．判定が通れば，集合 L に $(\mathcal{G}_0, \mathcal{G}_1)$ を追加していき，最後に 4.で $Eval$ 関数によって，もっとも二重化の度合いが小さい変換パターンを出力する．

この FindSplit は上記で述べた 4 つの条件を満たすことが，[4] で証明されている．

2.2 依存関係グラフの正準化

2.1 節で述べられたアルゴリズムでは，入力させる依存関係グラフに関して，暗号方式での群の依存関係を入力者が決める．その際，依存関係は暗号方式に対して一意に決まるものではない場合がある．例えば，群 X_1と X_2を乗算してから X_3と乗算する場合のグラフと，群 X_2と X_3を乗算してから X_1と乗算する場合のグラフ の 2 パターンが考えられる．このように，入力者は依存関係グラフを定義するとき入力に柔軟性があると言える．しかし，この柔軟性により，暗号方式としては同様にみることができるのに，グラフが異なることによって，FindSplit の結果が異なる可能性がある．ここでは，入力者がどのような依存関係グラフを定義したとしても，結果が同じになるように，依存関係グラフを正準化させる．

ある依存関係グラフと二重化禁止ノードの集合を FindSplit に与えたとき，依存関係グラフの正準化を行った上で全ての選択パターンにおいて二重化禁止ノードを二重化せざるを得なかった場合，つまり変換できなかった場合，正準化を行わずにいかなる依存関係グラフのパターンを構成し直してもその暗号方式は変換ができないという不可能性が示されていると言える．

ここでも関数の定義をする．あるグラフ $\mathcal{G}$ に対して，$\mathsf{In}(\mathcal{G})$ は，あるノードに対して向かうようなエッジが存在しないノードの集合である．

依存関係グラフの正準化アルゴリズム

依存関係グラフ $\mathcal{G} = (V, E)$ を $\bar{\mathcal{G}} = (V, \bar{E})$ へ正準化させる．
$P = \{P_1[0], P_1[1], \ldots, P_n[0], P_n[1]\}$ とする．

1. $\bar{E}$ を初期化する．

2. 全ての $X \in V \setminus P$, $Y \in \mathsf{In}(\mathsf{Anc}(\mathcal{G}, X))$ について，
(Y, X) を $\bar{E}$ に追加する．

3. 全ての $X \in P$ について，
$(*, X) \in E$ となるエッジを $\bar{E}$に追加する．

このアルゴリズムでは，In 関数の定義により，依存関係グラフのペアリングのノードを除く全てのノードに関してエッジをつなぎなおしている．これにより，依存関係グラフの形に関わらず，依存関係グラフを正準化させることができる．

ただし，2.1 節で述べられたアルゴリズムでは暗号方式全体の依存関係について一つのグラフにまとめていたが，依存関係グラフを正準化させることによってエッジがつなぎかえられるので，暗号方式内のアルゴリズムの構成が変わってしまう可能性がある．よって，この正準化は，暗号方式内のアルゴリズム別に行い，その後グラフを統合するようにする．

また，このアルゴリズムでは，依存関係グラフの正準化を行うことによって，Bottom ノードが正準化を行わない場合よりも増加し，試行パターン数が急激に増加してしまうという問題がある．そこで，Bottom ノードの統合を行う．依存関係グラフの正準化によって関係の推移律が生じることがないことが明らかであるので，単純な構造としてみなすことができる．ある Bottom ノードに対して向かっているエッジがありその始点となるノードの集合を G_1 としたとき，他のある Bottom ノードに対して向かっているエッジの始点となるノードの集合 G_2 が G_1 の部分集合となる場合，後者の Bottom ノードは

前者の Bottom ノードに統合することができる．これにより，試行パターン数を大幅に削減することができる．

2.3 ペアリングの正準化

ペアリングノードに関しても正準化が必要となる場合がある．例えば，e(AB,C) のときに変換が不可能である場合でも，e(AB,C) を e(A,C)e(B,C) とした場合でも変換が不可能かどうかはわからない．

ある依存関係グラフと二重化禁止ノードの集合を FindSplit に与えたとき，ペアリングの正準化を行った上で全ての選択パターンにおいて二重化禁止ノードを二重化せざるを得なかった場合，つまり変換できなかった場合，正準化を行わない場合でもその暗号方式は変換ができないという不可能性が示されていると言える．

ペアリングの正準化は以下のアルゴリズムを通して行う．

ペアリングの正準化アルゴリズム

依存関係グラフ $\mathcal{G} = (V, E)$ を $\bar{\mathcal{G}} = (V, \bar{E})$ へ正準化させる．$P = \{P_1[0], \ldots, P_n[1]\}$ とする．

1. $V' = V \setminus P$, $E' = \emptyset$ とする．
2. 全ての $X \in V \setminus P$, $Y \in \mathsf{In}(\mathsf{Anc}(\mathcal{G}, X))$ について，(Y, X) を $\bar{E}$ に追加する．
3. $w = 0$ とする．
4. $i = 1, \ldots, n_p$ として以下の手順を繰り返す．
   (a) $(X_1, \ldots, X_{t_0}) := \mathsf{In}(\mathsf{Anc}(\mathcal{G}, P_i[0]))$, $(Y_1, \ldots, Y_{t_1}) := \mathsf{In}(\mathsf{Anc}(\mathcal{G}, P_i[1]))$ とする．
   (b) $u = 1, \ldots, t_0$ かつ $v = 1, \ldots, t_1$として以下を繰り返す．
   - $x = w + (u-1) * t_1 + v$.
   - $P_x[0]$ と $P_x[1]$ を V'に追加する．
   - $(X_u, P_x[0])$ と $(Y_v, P_x[1])$ を E'に追加する．

   (c) $w = w + t_0 \times t_1$.

以上のアルゴリズムは，依存関係グラフの正準化をした後，ペアリングの正準化を行っている．例えば，ペアリングが e(ABC, DE) であった場合，e(A,D), e(A,E),(B,D), e(B,E),(C,D), e(C,E) に分解され，新たなペアリングノードに番号を付け，分解されたノードからそのペアリングノードへとエッジをつないでいく．

3 実装

2 章で述べた 3 つの変換アルゴリズムを Javascript によって実装した．

3.1 依存関係グラフによる変換アルゴリズムの実装

一つのテキストボックスに，入力形式に従って暗号方式の依存関係等の情報を入力した上で決定ボタンを押すと，二重化の度合いが最小となる選択の結果を出力するという構成になっている．入力形式を図 1 に示す．

```
[Dependencies]
%コメント
G_1              %G_1 はどの群にも依存しない
G_2              %G_2 はどの群にも依存しない
Z : G_1, G_2     %Z は G_1, G_2 に依存する
...

[Pairings]
G_1, G_2     %G_1 と G_2 がペアとなり
             %ペアリングの計算に用いられる
...

[Prohibits]
H, S     %H と S は二重化禁止

[Priority]
G_1     %G_1 は優先度 1
G_2     %G_2 は優先度 2
%指定していない群は最低の優先度となる
%この例では指定していない群は優先度 3 となる

[Weight]
G_i : 10     %群 G_i は変換の重みが 10
H_i : 10     %群 H_i は変換の重みが 10
%指定していない群は変換の重みは 1 となる
```

図 1: 依存関係グラフによる変換の入力形式

Dependencies では群同士の依存関係の定義を，Pairings ではペアリングの依存関係の定義を，Prohibits では二重化禁止の群の定義を，Priority では優先度の定義を，Weight ではノードの重みの定義を行っている．特に，Prohibits は，2.1 節の FindSplit アルゴリズムに入力する NoDup に対応している．Priority，Weight は Eval 関数，つまり二重化の度合いを評価する部分で用いられる．Priority で優先度 1 以下の群が二重化される場合と優先度 2 以下の群が二重化される場合とでは，評価結果は後者の方が高いとみなす．どの群が二重化されないことが望ましいのかを設定するために用意されている．Weight では，Priority に従って，その優先度の重みとして加算

される．例えば優先度 2 の重みが 2 となる群が二重化される場合と，優先度 2 の重みが 1 となる群が 2 つ二重化される場合とでは評価結果は同等とみなす．群の集合を設定するため等に用意されている．

また，2.1 節のアルゴリズムでは，二つのグラフに分割する際にグラフ同士に意味の違いはないにも関わらず，全通り探索するとグラフが対照的なものまで探索してしまうため無駄になってしまう．よって，実際には対照的なものについては探索しなくて良いので，探索数を 1/2 に減らす事ができる．具体的には 2.1 節の FindSplit の 3. の for$\ell = 0, \ldots, 2^{n_p+n_b} - 1$ は for$\ell = 0, \ldots, 2^{n_p+n_b-1} - 1$ とすることができる．

3.2 依存関係グラフの正準化の実装

3.1 節と同じ構成になっている．入力形式を図 2 に示す．異なるアルゴリズム間でも同じ群を用いるならば同じ群の名前にする必要があり，また，ペアリングの番号については異なる定義であれば，番号が重複しないようにしなければならない．

```
[Dependencies]
%アルゴリズムの名前は自由
 <Setup algorithm>
G_1   %G_1 はどの群にも依存しない
G_2   %G_2 はどの群にも依存しない
msk : G_2   %msk は G_2 に依存する
@1 : G_1, G_2   %G_1 と G_2 がペアとなり
                %ペアリングの計算に用いられる
                %番号は 1 番
%次のアルゴリズムについて記入する
 <Key Gen algorithm>
...

%以下は依存関係グラフによる変換の入力形式と
%同様
[Prohibits]
[Priority]
[Weight]
```

図 2: 依存関係グラフの正準化の入力形式

3.3 ペアリングの正準化の実装

3.2 節と同じ構成になっており，入力形式も同じで図 2 の通りとなっている．変わるのは内部でペアリングの正準化を行っている点のみである．ペアリングを再構成した順に番号が連番で振られる．このアルゴリズムでもペアリングの正準化により試行パターン数が増大する．

4 実行結果

2 つの暗号方式についてタイプ 3 への効率的な変換を行った．試行パターン数が少ない暗号方式に関しては，手作業による結果と比較し本当にアルゴリズムを反映した実装ができているかどうかの確認も行った．実行環境は，OS が OS X，プロセッサが 2.6GHz Intel Core i7，メモリが 8GB，ブラウザが Google Chrome となっている．

4.1 Waters IBE scheme [5]

Waters IBE scheme の依存関係，二重化禁止，評価基準を作成した入力例を以下に示す．

Waters IBE scheme の入力例

```
[Dependencies]
%construction
g_1
g_2
u'
u_i
msk : g_2
%msk  = master secret key
d_1 : g_2, u', u_i
d_2
%d_v = (d_1, d_2)
C_2
C_3 : u', u_i
%C_1 ∈ G_T
%proof
A
B
C
g_1 : A
g_2 : B
u' : g_2
u_i : g_2
d_1 : g_1, u', u_i
d_2 : g_1
%In the paper, d_1 is desacribed
%as d_0,  d_2 is described as d_1
C_2 : C
C_3 : C
[Pairings]
g_1, g_2
d_2, C_3
d_1, C_2
```

[Prohibits]
[Priority]
%Smaller assumption is preferred 1st
A, B, C
%Smaller ciertext is perferred 2nd
C_2, C_3
%Smaller public-key is preferred 3rd
g_1, g_2, u', u_i
[Weight]
$u\_i : 128$

A,B,C はそれぞれ，BDH 仮定の g^a, g^b, g^cに対応する．この入力例を実装したプログラムに入力すると，$\mathcal{G}_0$のノードとして $11, 21, 31, G\_2, u', u\_i, C\_2, C\_3, B, C$ が，$\mathcal{G}_1$ のノードとして $10, 20, 30, g\_1, g\_2, u', u\_i, msk$, d_1, d_2, A, B が出力される．二重化される群は g_2, u', u_i, B と出力される．数字の 10,11 はそれぞれ，$P_1[0], P_1[1]$ に対応する．実行時間は 7 ミリ秒となった．

上記の入力例の [Prihibits] の部分に A,B,C を入れた場合，つまり仮定の二重化を禁止させた場合は，グラフは出力されない．実行時間は同様に 7 ミリ秒となった．

4.2 VRL Group Signature Scheme [6]

VRL の入力例について示す．

VRL Group Signature Scheme の入力例

[Dependencies]
%construction
%%KeyGen
g_1
g_2
$w : g\_2$
A_i
%%SigGen
M
$u : g\_1, g\_2, w, M$
$v : g\_1, g\_2, w, M$
$T\_1 : u$
$T\_2 : A\_i, v$
$R\_1 : u$
$R\_3 : T\_1, u$
%%Verify
$R\_1 : u, T\_1$
$R\_3 : T\_1, u$
%Proof
%%Lemma6.1
u_0
u_1
v'
$h\_0 : u\_0$
$h\_1 : u\_1$
Z
%v' is described as v in the paper.
%The above is a set of instances.
$w : g\_2$
W
$A\_i0 : Z, W, v'$
$A\_i1 : W, v'$
$T\_1 : h\_0, u\_0$
$T\_2 : Z, W, v', h\_0, u\_0$
$u : u\_0$
$v : v', u\_0$
$T\_1 : u$
$T\_2 : A\_i0, v'$
$T\_1 : h\_1, u\_1$
$T\_2 : W, h\_1, u\_1, v$
$u : u\_1$
$v : u\_1, v'$
$T\_1 : u$
$T\_2 : A\_i1, v$
%%Lemma6.2
$w : g\_2$
$T\_1 : u$
$T\_2 : A\_i, g\_1$
%A_i is described as A in the paper
A^*
S
$u : g\_1, g\_2, w, S$
$v : g\_1, g\_2, w, S$
%%exculpability
$T'\_2 : T\_2, A\_i$
$\%T'\_2 = T\_2/A\_i$
[Pairings]
g_1, g_2
T_2, g_2
v, w
v, g_2
T_2, g_2
u, w
v, g_2
T_2, w
T_2, u
A_i, u
T_1, v
A_i, w
A_i, g_2
T_1, v

A_i, u
T_2, u
[Prohibits]
u, v
[Priority]
[Weight]
$A\_i : 10$

上記の入力例に対して，$\mathcal{G}_0$のノードとして $11, 21, 31, 41, 51, 60, 71, 81, 91, 101, 110, 121, 131, 140, 151, 161, g\_1, g\_2, w, M, u, T\_1, R\_1, R\_3, u\_0, u\_1, h\_0, h\_1, A^*, S$ が，$\mathcal{G}\_1$ のノードとして $10, 20, 30, 40, 50, 61, 70, 80, 90, 100, 111, 120, 130, 141, 150, 160, g\_1, g\_2, w, A\_i, M, v, T\_2, u\_0, u\_1, v', h\_0, h\_1, Z, W, A\_i0, A\_i1, S, T'\_2$ が出力される．二重化される群は $g\_1, g\_2, w, M, u\_0, u\_1, h\_0, h\_1, S$ と出力される．実行時間は 43 分 38 秒となった．

この実行結果からタイプ 3 の双線形写像群へ変換した場合の暗号方式について以下に示す．

KeyGen(n)**:**生成元 $g \in \mathbb{G}$ と $\tilde{g} \in \tilde{\mathbb{G}}$を選択する．次に $\gamma \leftarrow \mathbb{Z}_p^*$を選択し，$w = g^\gamma, \tilde{w} = \tilde{g}^\gamma \in \tilde{\mathbb{G}}$とする．$x_i \leftarrow \mathbb{Z}_p^*$ をランダムに選択し，$A_i \leftarrow g^{1/(\gamma + x_i)}$とする．

そして，公開鍵 $gpk = (g, \tilde{g}, w, \tilde{w}) \in \mathbb{G}^2 \times \tilde{\mathbb{G}}^2$，秘密鍵 $gsk[i] = (A_i, x_i) \in \mathbb{G} \times \mathbb{Z}_p^*$とする．

Sign$(gpk, gsk[i], M)$**:**入力されたメッセージを $M \in \{0,1\}^*$とし，以下の手順を行う．ランダムに $r \leftarrow \mathbb{Z}_p$ を取り，$(\tilde{u}, v) \leftarrow H_0(gpk, M, r) \in \tilde{\mathbb{G}} \times \mathbb{G}$ を計算する．H_0 はハッシュ関数である．ランダムに$\alpha \leftarrow \mathbb{Z}_p$を取り，$\tilde{T}_1 \leftarrow \tilde{u}^\alpha \in \tilde{\mathbb{G}}, T_2 \leftarrow A_i v^\alpha \in \mathbb{G}$ を計算する．$\delta \leftarrow x_i \alpha \in \mathbb{Z}_p$とし，$r_\alpha, r_x, r_\delta \leftarrow \mathbb{Z}_p$をランダムに取る．$\tilde{R}_1 \leftarrow \tilde{u}^{r_\alpha} \in \tilde{\mathbb{G}}, \tilde{R}_3 \leftarrow \tilde{T}_1^{r_x} u^{-r_\delta} \in \tilde{\mathbb{G}}$，$R_2 \leftarrow e(T_2, \tilde{g})^{r_x} e(v, \tilde{w})^{-r_\alpha} e(v, \tilde{g})^{-r_\delta}$を計算する．ハッシュ関数 H を使って $c \leftarrow H(gpk, M, r, \tilde{T}_1, T_2, \tilde{R}_1, R_2, \tilde{R}_3) \in \mathbb{Z}_p$として計算する．$s_\alpha = r_\alpha + c\alpha, s_x = r_x + cx_i, s_\delta = r_\delta + c\delta \in \mathbb{Z}_p$を計算する．

最後に，署名$\sigma \leftarrow (r, \tilde{T}_1, T_2, c, s_\alpha, s_x, s_\delta) \in \mathbb{G} \times \tilde{\mathbb{G}} \times \mathbb{Z}_p^5$ を出力する．

Verify(gpk, RL, σ, M)**:**署名の検証を行う．RL は失効リストであり，σはグループ署名である．

$(\tilde{u}, v) \leftarrow H_0(gpk, M, r) \in \tilde{\mathbb{G}} \times \mathbb{G}$ を計算し，$\tilde{R}'_1 \leftarrow \tilde{u}^{s_\alpha} / \tilde{T}_1^c, \tilde{R}'_3 \leftarrow \tilde{T}_1^{s_x} u^{-s_\delta}$，$R'_2 \leftarrow e(T_2, \tilde{g})^{s_x} e(v, \tilde{w})^{-s_\alpha} e(v, \tilde{g})^{-s_\delta} \left(\frac{e(T_2, \tilde{w})}{e(g, \tilde{g})}\right)^c$を求め，$c \stackrel{?}{=} H(gpk, M, r, \tilde{T}_1, T_2, \tilde{R}'_1, R'_2, \tilde{R}'_3)$ を検証する．

以上が，タイプ 1 の双線形写像群による VRL Group Signature Scheme をタイプ 3 の双線形写像群を用いた暗号方式に変換した結果である．

5 考察

手作業による結果との相違もなく，正しくアルゴリズムの反映をした実装ができていると言える．そして，実装によりコンピュータで実行できるようになったので，より効率的に，正確に最も二重化の度合いが小さくなる変換の選択について求められるようになった．結果として思考パターン数が多い暗号方式は手作業では求めることが難しかったが，実装により求められるようになった．

特に，Waters IBE scheme に関しては仮定の二重化をせずにタイプ 3 へ変換させることは不可能であるということがわかった．

また，VRL Group Signature Scheme に関しては [7] ではタイプ 1 からタイプ 3 への変換は困難であると述べられており，また [8] では HashToPoint や準同型写像を使うため変換することができないと述べられていた．しかし，今回の実行結果から実際に変換できるということを発見できた．

また，依存関係グラフの正準化とペアリングの正準化により，入力者は群の計算順序を意識することなく依存関係について定義することができるようになった．正準化した場合に変換が出来なかった場合は，正準化しなかった場合も変換できないという不可能性を示す事もできていると言える．

6 おわりに

本論文では，双線形写像群のタイプ 1 からタイプ 3 へと変換させるアルゴリズムの実装を行い，実際に様々な暗号方式について変換を試みた．その結果，それぞれのタイプ 1 からタイプ 3 への効率的な変換について明らかになった．また，変換が困難であると思われていた暗号方式の変換ができることも分かった．このように，タイプ 1 の暗号方式はセキュリティ，効率の面で実用的でなかったが，タイプ 3 への効率的な変換を，群同士の依存関係について示すことによって機械的に行えるようになり，より効率的に正確に，手作業では困難だった変換の選択が可能になった．また，設計者は設計の容易なタイプ 1 で暗号方式を構成してから，この変換アルゴリズムを用いてタイプ 3 に変換させることも可能となった．

7 謝辞

有益な議論をして頂いた UCL の Groth 様と，入力ファイルを作成を行って頂いた NICT の大久保様に感謝致します．

参考文献

[1] S. D. Galbraith, K. G. Paterson, and N. P. Smart. “Pairings for cryptographers.” Discrete Applied Mathematics, 156(16):3113-3121, 2008.

[2] Jie Chen, Hoon Wei Lim, San Ling, Huaxiong Wang, Hoeteck Wee “ePrint 2012/224 and Pairing 2012,” LNCS 7708, pp.122-140, 2013.

[3] Ramanna, Chatterjee, Sarkar “ePrint 2012/024, PKC’12”

[4] Masayuki Abe, Jens Grohth, Jens Grohth, Miyako Ohkubo, Takeya Tango. “Automated Translation for Schemes on Symmetric Bilinear Groups.”

[5] B. Waters. “Efficient identity-based encryption without random oracles.” In Advances in Cryptology - Eurocrypt 2005, volume 3494 of LNCS, pages 114-127. Springer-Verlag, 2005.

[6] D. Boneh, H. Shacham. “Group signatures with verier-local revocation. In V. Atluri, B. Pfitzmann, and P. D. McDaniel, editors, ” ACM Conference on Computer and Communications Security, pages 168-177. ACM, 2004.

[7] N. P. Smart, F. Vercauteren. “On computable isomorphisms in efficient asymmetric pairing-based systems.” Discrete Applied Mathematics, 155(4):538-547, 2007.

[8] S. Chatterjee, A. Menezes. “On cryptographic protocols employing asymmetric pairings - the role of revisited.” Discrete Applied Mathematics, 159(13):1311-1322, 2011.