Master 1 Algèbre Appliquée: Cryptographie et calcul formel

Juin 2018

LE CHIFFREMENT COMPLETEMENT HOMOMORPHE DE DGHV

présenté par:

- Mouhamadou Thioub

Sous la direction de Ilaria Chilotti

Contents

Remerciement

Je remercie mon encadreuse Ilaria Chilotti pour son aide tout au long du semestre , mais aussi pour toutes les remarques constructives qu'elle a pu faire concernant le présent travail. Elle a notamment contribué à rendre ce travail plus abordable.

RÉSUMÉ

Le chiffrement homomorphe est aujourd'hui une part de la cryptographie si importante qu'elle semble être un phénomène de mode. Si bien que depuis les premiers travaux de Graig Gentry en 2009, des dizaines de schémas ont vu le jour, ce qui rend la tâche difficile à un néophyte pour s'y retrouver dans cette mer de schémas disponibles. Ce travail consiste donc à étudier le chiffrement complètement homomorphe de DGHV et limplémenter en langage C.

ABSTRACT

The homomorphic encryption is nowadays such an important part of the science of cryptography that it seems to be a trend. So much important that since the first Craig Gentry's blueprint in 2009, dozens of schemes were born, hardening the work of a beginner not to be confused in such an amount of available schemes. Our work consisted to study the fully homomorphic encryption of DGHV and to implement it in language C.

1 Introduction

1.1 Motivation

La cryptographie moderne est basée sur des problèmes mathématiques liés aux nombres, par exemple l'algorithme RSA(Rivest Shamir Adelman) qui utilise la factorisation des entiers.

Cependant, avec l'émergence de la théorie des ordinateurs quantiques avec l'algorithme de Shor, ce type de solution devient vulnérable aux attaques quantiques.

Pour protéger les données contre la cryptanalyse quantique, une nouvelle zone a émergé dans la sécurité de l'information appelée cryptographie post-quantique, ce qui étudie des algorithmes résistants aux attaques quantiques dont l'attaque est exponentielle même pour un ordinateur quantique.

Parmi les caractéristiques de la cryptographie post-quantique, le chiffrement homomorphe offre la possibilité de maintenir l'intégrité des données et assurer leur confidentialité.

1.2 Objectif

L'objectif de notre travail est de décrire le système cryptographique de DGHV, un schéma homomorphe très simple en théorie, mais aussi l'implémenter en langage C.

1.3 Quelques définitions

Les schémas cryptographiques homomorphiques nous permettent d'effectuer des opérations sur des messages cryptés. Nous verrons deux définitions nécessaires pour classer les cryptosystèmes homomorphes.

Dans ce qui suit la fonction Enc est une fonction de chiffrement d'un message m avec la clé publique P_k; *Dec est la fonction de déchiffrement correspondante en utilisant la clé secrète* S_k.

1.3.1 Chiffrement homomorphe

Un chiffrement est homomorphe si :
À partir de Enc(a) et Enc(b), il est possible de calculer Enc(f(a, b)) où f peut-être, par exemple : $\oplus, \otimes, \times, +$ *et sans que la clé privée ne soit utilisée.*

Parmi les chiffrements Homomorphes, on distingue, selon l'opération qu'ils permettent d'évaluer sur des clairs, les chiffrements homomorphes additifs (se

basent sur des opérations d'addition uniquement), c'est le cas des cryptosystèmes de Pailler et de Goldwasser-Micalli et les chiffrements homomorphes multiplicatifs (se basent sur des opérations de multiplication uniquement) tels que les cryptosystèmes de RSA et d'El Gamal.

1.3.2 Chiffrement Homomorphe additif

Dans un contexte de chiffrement additif, un serveur distant pourra retourner le résultat d'une opération d'addition sur les messages en clair en faisant le calcul sur des messages chiffrées, sans disposer de la clé secrète.

En d'autres termes, soient :

Enc une fonction de chiffrement.
Dec_s *une* $fonctionde\ déchiffrement$ *à clé secrète* s.
Alors :

$$Dec_s(Enc(m) + Enc(n)) = m + n$$

1.3.3 Chiffrement Homomorphe multiplicatif

Par analogie avec ce qui précède, un système basé sur le chiffrement homomorphe multiplicatif permet de n'effectuer que des produits sur les clairs, sans disposer de la clé secrète.
En d'autres termes, soient :

Enc une fonction de chiffrement.
Dec_s *une* $fonctionde\ déchiffrement$ *à clé secrètes.*
Alors :

$$Dec_s(Enc(m) \times Enc(n)) = m \times n$$

Remarque:

Un chiffrement sera dit partiellement homomorphe lorsqu'il est homomorphe pour une seule opération. Il sera dit complétement homomorphe lorsqu'il est homomorphe pour toutes les deux opérations.

1.4 Un peu d'histoire

1.4.1 Du chiffrement partiellement homomorphe au chiffrement complètement homomorphe

La cryptographie asymétrique est apparue en 1978 lorsque Rivest, Shamir et Adelman publient leur cryptosystème RSA. Ce dernier est révolutionnaire dans le sens où seul le destinataire peut déchiffrer le message envoyé . On remarque deux particularités sur RSA:

- Il est homomorphe pour la multiplication;
- Il est distinguable: étant donné un message m et un chiffré c, il est possible de savoir si Enc(m)=c ou non.
La première particularité fait l'objet de la conjecture de Rivest , Adelman et Dertouzos sur l'existence d'un chiffrement complètement homomorphe.

La seconde particularité bien moins désirable, fait l'objet des attaques à clairs choisis sur le RSA. Le RSA largement utilisé aujourd'hui a été revisité afin de pouvoir résister aux attaques à clairs choisis: une première fois dans une version OAEP (Optimal Asymmetric Encryption Padding), puis une deuxième fois dans une version OAEP+.Cependant ces deux versions perdent le caractère homomorphe.

Deux ans plus tard, El Gammal publie un système de chiffrement qui reprend certaines idées du protocole d'échange des clés de Diffie- Hellman. Ce dernier est aussi homomorphe partiellement (pour la multiplication).
Ce n'est qu'en 1997 que le premier cryptosystème homomorphe pour l'addition est apparue .Il fut publié par Naccache et Stern.
Nombreuses sont les tentatives de chiffrements complètement homomorphes depuis sa conjecture mais jusqu'en 2009, on n'a pas réussi à élaborer un système complètement homomorphe. En résumé, on va donner juste les noms de quelques chercheurs qui ont essayé de proposer un système de chiffrement complètement homomorphe mais sans succès.

Tentatives de résolution :

-1984 : Goldwasser, Micali : XOR sur les bits
-1984 : El Gamal : multiplication modulo p
-1998 : Paillier : Addition modulo n = pq
-2005 : Boneh, Goh, Nissim : 2 additions et 1 multiplication modulo p

1.4.2 La thèse de Gentry,une nouvelle ère

Dans sa thèse, Gentry choisit un bruit r (ou erreur) dans un idéal I d'un anneau A, r=kI.
Le message est alors chiffré en ajoutant ce bruit au message. Formellement, on a:
c= m+kI .
La procédure de déchiffrement consiste à retirer le bruit. Les propriétés homomorphes de ce système sont quasi-immédiates:‘

$$\begin{aligned} \mathrm{c}_1 + c_2 &= m_1 + m_2 + (k_1 + k_2)I \\ \mathrm{c}_1 c_2 &= m_1 m_2 + (m_1 k_2 + m_2 k_1 + k_1 k_2). \end{aligned}$$

Donc le premier schéma complètement homomorphe est du à Gentry 2009.
Un an plus tard,il publie avec Van Dikj , Vaikuntanathan et Halevi un cryptosystème du nom de DGHV. Ce dernier fait l'objet de notre projet.
Dans ce qui suit ,nous allons faire une présentation du schéma de DGHV avec ses propriétés homomorphes et sa sécurité sémantique.

2 Étude du schéma DGHV

Le schéma DGHV est apparue en deux versions : une version symétrique et une version asymétrique.

2.1 Le chiffrement DGHV symétrique

La clé secrète sera un nombre premier p de taille η.
Pour chiffré un message $m \in \{0,1\}$, *on choisit au hasard un entier* $r \in]-2^{\cdot}2^{\gamma}[$ *etunentierq* $\in [0, 2^{\gamma}/p[$

Le chiffré du message est Enc(m)=c=pq+2r+m.
Pour déchiffrer , on calcule Dec(c)=(c mod p)mod 2.

Les propriétés homomorphes sont immédiates . En effet, si
$c_1 = Enc(m_1) = pq_1 + 2r_1 + m_1$ *et*
$c_2 = Enc(m_2) = pq_2 + 2r_2 + m_2$,
ona :
$c_1 + c_2 = (q_1 + q_2)p + 2(r_1 + r_2) + m_1 m_2$ *et*
$c_1 c_2 = (c_2 q_1 + c_1 q_2 + q_1 q_2)p + 2(2r_1 r_2 + r_2 m_1 + r_1 m_2) + m_1 m_2.$

Remarque:

Le bruit doit rester inférieur à p pour que l'on puisse déchiffrer .

2.2 Le chiffrement DGHV asymétrique

Le problème sur lequel repose ce schéma est le SSSP (Sparse Sub set Sum Problem) et le ACGD(Approximate greatest common divisor) que nous expliquerons ultérieurement.
Le chiffrement comporte plusieurs étapes :

2.2.1 Génération de clés

Pour la clé privée ,on génère un nombre impair de taille η, $p \in 2\boldsymbol{Z} \cap \left[2^{\eta-1}\, 2^{\eta}\right[$.
Pour la clé publique , on tire au hasard les x_i *dans une distribution* $D_{\gamma,\rho}(p)$, *pour* $\in \{0, ..., \tau\} i$.
Renommer les x_i *afin que* $x - 0$ *soit le plus grand.*

Recommencer jusqu' à ce que x_0 soit impair et $x_0(mod\ p)$ soit pair.
La clé publique est $pk =< x_0, x_1, ...x_\tau >$.

2.2.2 Chiffrement du message

Choisir aléatoirement S $\subseteq \{1, ..., \tau\} et un nombre aléatoire r \in]-2^\gamma, 2^\gamma[$.
Le chiffré est $c = (m + 2r + \sum_{i \in S} x_i) mod x_o$.

2.2.3 Déchiffrement

Pour déchiffrer, on calcule (c modp) mod 2

2.3 Le boostrapping

La condition nécessaire pour que la propriété homomorphe se conserve est que le bruit soit inférieur à $\frac{p}{2}$. Il est clair que le bruit augmente pour chaque opération mais surtout pour la multiplication. Cette augmentation du bruit limite le nombre d'opérations possibles.

Pour rendre le schéma complètement homomorphe, les auteurs propose le boostrapping qui consiste à "rafraîchir" la taille du bruit . Malheureusement, j'ai pas réussi à coder le boostrapping de DGHV .

2.4 Étude de la sécurité de DGHV

Définition1:(SSSP)

Soit A = $a_1, a_2, ..., a_n$ *un grand ensemble d'entiers*, $t, M \in Z$.
Le problème SSSP consiste à trouver un sous−ensemble S de A tel que la somme des entiers dans S vaille $t(mod M)$.

SSSP est une version particulière du problème du sac à dos.

Définition2:(AGCD)

Soient p un entier impair de taille η *et la distribution* :

$D_{\gamma,\rho}(p) = \{$*choisir* $q \in Z \cap [0, 2^\gamma/p[$, *choisir* $r \in]-2^\rho, 2^\rho/[$, : *outpout* $x = pq + 2r\}$

$D_{\gamma,\rho}(p)$ *est une distribution qui génère des "presque" multiples de p.*

Le problème $(\rho, , \gamma)-AGCD$ *est de retrouver p ayant un nombre polynomiale d'éléments de* $D_{,\rho}(p)$ *pour p un nombre aléatoire donné, entier impair de taille .*

La sécurité du système repose sur les deux problèmes précédents.

SSSP intervient dans le chiffrement du message.Sa difficulté nous garantit qu'un attaquant ne sera pas capable de retrouver le sous ensemble S décrit dans le chiffrement.

AGCD intervient dans la génération de la clé publique .
Sa difficulté nous garantit qu'un attaquant ne sera pas capable de trouver le secret p .

Afin de garantir la vulnérabilité face aux attaques connus, certains auteurs suggèrent d'utiliser les paramètres de sécurité ci-dessous:

parameters	toy	small	meduim	large
λ	42	52	62	72
ρ	16	24	32	39
η	1088	1632	2176	2652
γ	$16*10^4$	$86*10^4$	$42*10^5$	$19*10^6$

3 Enjeux et Applications pratiques du chiffrement homomorphe

3.1 Déléguer un calcul sans divulguer ses données

Si on analyse le fonctionnement du chiffrement homomorphe, c'est un échange de données en toute confidentialité (c.à.d. les données sont chiffrées en local par l'entreprise cliente, et l'application génératrice des clés est responsable du chiffrement et déchiffrement s'exécute sur les ordinateurs de l'entreprise). La Figure 4.1 ci-dessous éclaircit l'échange homomorphe des données entre l'entreprise cliente et le serveur Cloud.

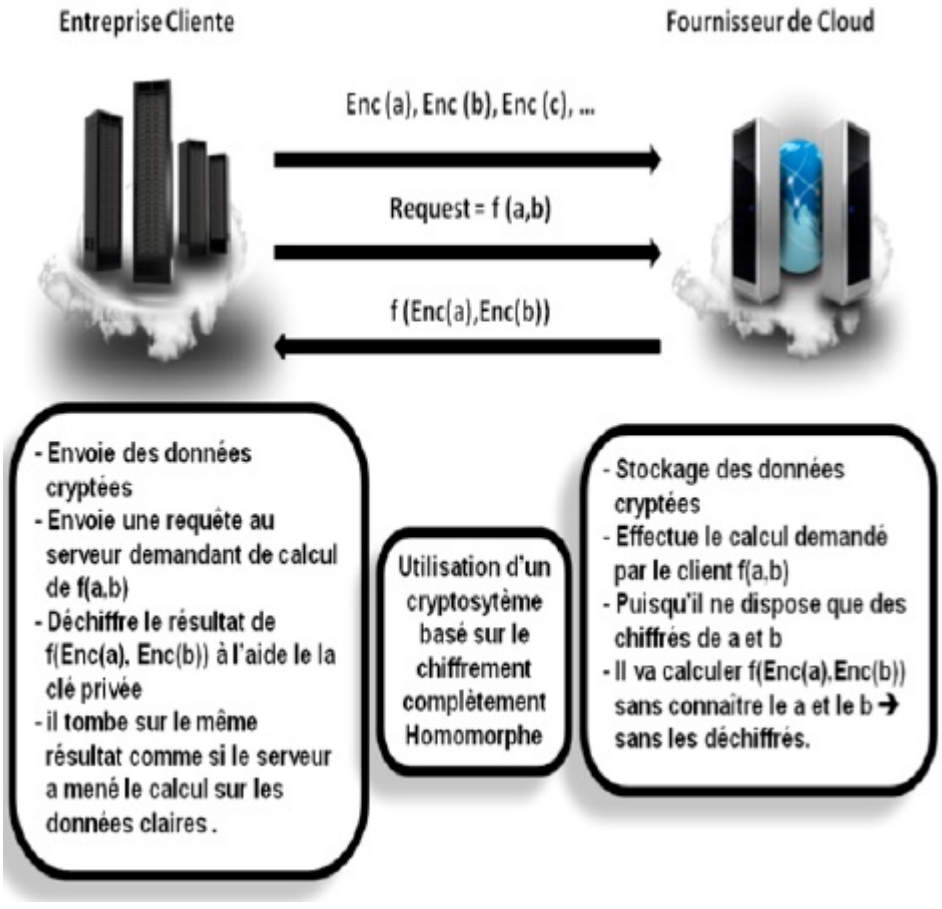

Figure 4.1 Chiffrement Homomorphe au Cloud Computing

Si nous sommes prudent, pour chiffrer nos données avant de les confier à un service de sauvegarde extérieur, dans le cloud par exemple. Comment demander alors à ce service de prendre les données d'un fichier et d'en calculer la moyenne, ou d'en extraire des informations particulières ? Pour calculer et manipuler des données, il semble qu'il faille les connaître.

Non ! C'est tout l'enjeu des systèmes de chiffrements homomorphes : nous permettons à un tiers de calculer avec nos données qu'il ne connaît pas, et il produit un résultat qui lui est illisible, mais que nous savons déchiffrer.

3.2 Un protocole: Le retrait d'information privé

Le contexte **?** est le suivant. On dispose d'une grande base de données que l'on souhaite interroger de façon privée , on ne veut pas que le possesseur de la base de données sache à quel élément de cette base on s'intéresse. Ce peut être le cas par exemple d'un investisseur abonné à un service d'informations financières, et qui ne désire pas révéler à quelles sociétés il s'intéresse.

On suppose que la base de données est organisée de la façon suivante : chaque entrée est indexée par un entier unique, on les note $\mathrm{I}_1, \ldots, I_d$.
Les entrées de la base de données ne sont pas supposés chiffrées, ce sont simplement des nombres entiers.

On souhaite obtenir l'élément I_k *sans que le possesseur de la base de données ne le sache. Voici comment procéder* :

$$Enc(n+m) = Enc(n) + Enc(m), et\ donc\ également$$
$$Enc(2n) = Enc(n) + Enc(n) = 2Enc(n).$$

Par extension, pour tous les entiers $\mathrm{n}_1, \ldots, n_d$ *et* $a_1, \ldots, a_d, ona$:
$Enc(a_1 n_1 + + a_d n_d) = a_1 Enc(n1) + ... + a_d Enc(n_d)$
L'utilisateur commence par calculer des chiffrés m_i, *où* m_i *est un chiffré de* 0 *sauf pour* $i = k$.

Dans ce cas , m_k *est un chiffré de* 1 .*On suppose que notre système de déchiffrement est randomisé, et donc que tous les* m_i *sont différents* : (0 *peut être chiffré de plusieurs façons différentes*).

L' utilisateur envoie alors à l'administrateur de la base de données la requête $(\mathrm{m}_1, \ldots, m_d)$.

Il n'a aucun moyen de savoir quel est celui des m_i *qui correspond à*1 .*Il calcule alors* $I_1 m_1 + + I_d m_d$,
il renvoie ce résultat au client .$Celui - ci$ *calcule*
$Des(I_1 m_1 + + I_d m_d)$ *et récupère* I_k.

En effet, on a:

$$\begin{aligned} \mathrm{I}_1 m_1 + ... I_d m_d = \ & I_1 Enc(0) + ... I_k Enc(1) + ... + I_d Enc(0) \\ & = \mathrm{Enc}(\mathrm{I}_1 * o + ... I_k * 1 + ... I_d * 0) \\ & = \mathrm{Enc}(\mathrm{I}_k) \end{aligned}$$

4 Implémentation

4.1 GMP: Une librairie des grands nombres

La librairie GMP(GNU Multiple Precision) est une librairie utilisée en langage C pour manipuler des grands nombres. Dans cette librairie, plusieurs fonctions arithmétiques y sont prédéfinies.

Pour faciliter au lecteur la compréhension du code, nous donnerons dans la suite les fonctions de la librairie GMP qui ont été utilisées dans le code source ainsi que leur rôle.

Comme toutes les autres librairies en langage C, pour pouvoir accéder aux fonctions prédéfinies de la librairie, on utilise "include ¡gmp.h¿ dans l'entête du programme.

Les variables de types GMP (les grands nombres) sont déclarées comme suit:
mpz_int x;
mpz_init(x);
x est déclaré du type entier et est initialisé à 0.
mpz_add(somme,x,y): calcule x+y et affecte le résultat dans somme.
mpz‘_mul(prod,x,y): calcule x*y et affecte le résultat dans prod.
mpz_set_str (x , "a", b): affecte dans x la valeur de a lu en base b.
mpz_probab_prime_p (n , resp): effectue le test de Miller-Rabin avec une probabilité de 4^{-resp} pour tester si n est premier.
mpz_nextprime(p,p_1) : *cherche (avec un test de primalité probabiliste) le plus petit nombre premier supérieur à p et le met dans p.*
$mpz\_powm(x, a, b, n)$: *calcule* $a^b \ (mod\ n)$ *et* $l'affecte\ à\ x$.
$mpz\_urandomb(x, state, n)$: *génère un entier aléatoire compris entre* 0 *et* 2^{n-1}.

4.2 Réalisation

Cette partie servira à exposer les particularités à implanter DGHV pour les expériences. Les tests et les calculs ont été effectués sur les serveurs de L'UVSQ (https://sage.prism.uvsq.fr/), essentiellement sur Acer ayant 2 processeurs Intel

Core i3 cadencés à 1,8 GHz et 4 Go fonctionnant sous Ubuntu.

Afin de calculer le temps d'exécution de certaines instructions en C, il est pratique d'utiliser la fonction clock() disponible directement dans la librairie ¡time.h¿. Ainsi pour stocker le temps dans une variable time et ensuite connaître le temps depuis time, il suffit de faire :
int time = 0;
// Votre code ...
time = clock();

Ci-dessous la Figure 3.2 qui représente un simple exemple de réalisation d'un calcul en utilisant notre programme " DGHV Homomorphe", ce programme accepte en entrée : "la taille de la clé secrète qu'on veut utiliser" , " la taille de q ", et " la taille du bruit r " ensuite elle génère la Clé privée.

```
void  chiffrement_DGHV(mpz_t chiffre ,mpz_t m,int taille_p , mpz_t p,int taille_q ,
{
    mpz_t q,r,prod,inter;
    mpz_init(prod);
    mpz_init(inter);
    mpz_init(q);
    mpz_init(r);
    mpz_init(chiffre);
    /*initialise random state*/
    gmp_randstate_t state;
    gmp_randinit_default (state);

    mpz_urandomb (r, state, taille_r);
    mpz_urandomb (q, state, taille_q);
    mpz_mul(prod,q,p);
    mpz_add(r,r,r);
    mpz_add(chiffre,prod,r);
    mpz_add(chiffre,chiffre,m);

return;

// Procedure pour dechiffrer
void dechiffrer_DGHV(mpz_t chiffre,mpz_t key)
{
     mpz_t x;
     mpz_t y;
     mpz_init(y);
     mpz_t dechiffre;
     mpz_init(dechiffre);
```

```
      mpz_init(x);
      mpz_set_str(x, "1", 10);
      mpz_set_str(y,"2",10);
      mpz_powm (dechiffre, chiffre , x , key);
      mpz_powm(dechiffre,dechiffre,x,y);
      gmp_printf ("Dechiffrement DGHV (%Zd)= %Zd\n",chiffre,dechiffre);
return;
}

 /*on genere un nombre compris entre min et max*/
void gen_alea_intervalle(mpz_t alea,int n,gmp_randstate_t state)
{
      mpz_t max;
      mpz_init(max);
      mpz_ui_pow_ui(max,2,n);
      mpz_t min;
      mpz_init(min);
      mpz_ui_pow_ui(min,2,n-1);
      mpz_urandomm(alea,state,max);

     do
      {
          mpz_urandomm(alea,state,max);
      }while(mpz_cmp(alea,min) > 0);

     return;
}

// trouver un nombre premier compris entre min et max    l'aide du nombre g  n
void gen_premier_alea_intervalle(mpz_t prime,int n,gmp_randstate_t state)
{
  mpz_t prime2;
  mpz_init(prime2);
  gen_alea_intervalle(prime,n,state);

  do
    {
      mpz_nextprime(prime,prime);


    }while( !mpz_probab_prime_p(prime,MR_REP));

  return;
}
```

5 Conclusion

Le chiffrement homomorphe a connu des avancées théoriques importantes, dont la plus remarquée est celle de Gentry avec la publication du premier système de chiffrement complètement homomorphe (Fully Homomorphic Encryption).

De nombreux cryptosystèmes homomorphes ont des applications uniquement en théorie mais en pratique dès qu'on veut effectuer un calcul surtout dans le cas de la multiplication, la taille du produit des chiffrés explosent, surtout dans le cas du chiffrement complètement homomorphe qui demande encore des optimisations au niveau des paramètres, chose qui rend son application impossible jusqu'au jour d'aujourd'hui.

L'étude du cryptosystème de DGHV m'a permis de comprendre son fonctionnement théorique mais aussi de réviser le langage C grâce aux implémentations de différentes fonctions.

Néanmoins, avec les problèmes rencontrés lors de l'exécution des algorithmes je me suis convaincu que DGHV n'est pas l'algorithme optimal pour rendre un schéma complètement homomorphe.

References

1. Manual GMP disponible sur https://gmplib.org
2. Jean Sebastien CORON, Full Homomorphic Encryption Over the integers with shorter public keys, Édition Eurocrypt 2011. LNCS, ,PP.487-504.
3. Gentry, Halevi, Van Dijk,Vaikuntanathan,V.: 2010, Full Homomorphic Encryption Over the integers . In Gilbert, H. édition Eurocrypt 2010. LNCS, VOL.6110 Pp.24-43.