DC Programming and DCA for some classes of problems in cryptography

Thi Tuyet Trinh Nguyen

UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Programmation DC et DCA pour certaines classes de problèmes en cryptographie

DC Programming and DCA for some classes of problems in cryptography

THÈSE

présentée et soutenue publiquement le 6 décembre 2023

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

NGUYEN Thi Tuyet Trinh

Composition du jury

Président :	Anass NAGIH	Professeur, Université de Lorraine
Rapporteurs :	Viet Hung NGUYEN	Professeur, Université Clermont Auvergne
	Mounir HADDOU	Professeur, INSA de Rennes
Examinateurs :	Van Tien DO	Professeur, Université de technologie et d'économie de Budapest
	Van Dat CUNG	Professeur, Grenoble INP-UGA
Directrice de thèse :	Hoai An LE THI	Professeur, Université de Lorraine

Laboratoire de Génie Informatique, de Production et de Maintenance — EA 3096

Mis en page avec la classe thesul.

Remerciements

Je souhaite ici adresser un grand merci à toutes les personnes qui ont rendu cette thèse possible. Cette thèse a été réalisée au sein du département Informatique & Applications (IA), LGIPM, l'Université de Lorraine.

Tout d'abord, je tiens à exprimer ma grande gratitude à Madame Hoai An Le Thi, qui est ma directrice de thèse et professeur des Universités à l'Université de Lorraine, pour m'avoir donné la chance de travailler au sein du département Informatique & Applications du LGIPM de l'Université de Lorraine. Je ne peux que louer son talent, et c'est un privilège pour moi de collaborer avec elle. Elle m'a introduit au domaine de la recherche et m'a enseigné. Je tiens à exprimer ma sincère gratitude pour sa disponibilité, sa patience, son soutien, et tous les précieux conseils qu'elle m'a prodigués tout au long de cette thèse. Grâce à son accompagnement, sa disponibilité, et ses conseils, j'ai pu achever cette thèse et surpasser mes propres attentes.

Je tiens à exprimer mon respect et mes sincères remerciements au Professeur Tao Pham Dinh de l'INSA de Rouen pour ses précieux conseils et son soutien dans le cadre de mes travaux de recherche. Je suis reconnaissant pour les discussions profondes et captivantes que nous avons eues, ainsi que pour ses suggestions qui m'ont ouvert de nouvelles perspectives de recherche.

Je tiens à exprimer ma gratitude sincère à Monsieur Viet Hung Nguyen, Professeur à l'Université Clermont Auvergne, et à Monsieur Mounir Haddou, Professeur à l'INSA de Rennes, pour avoir accepté d'être les rapporteurs de ma thèse et pour leur temps précieux.

Je souhaite également remercier Professeur Van Tien Do, Université de technologie et d'économie de Budapest, Professeur Anass Nagih, Université de Lorraine, Professeur Van Dat Cung, Grenoble INP-UGA, pour m'avoir fait l'honneur d'accepter d'être membre du jury.

Je voudrais exprimer ma gratitude au docteur Hoang Phuc Hau Luu, Université de Lorraine, pour son aide.

Je voudrais remercier le reader Xuan Vinh Doan, Université de Warwick, pour sa discussion sur la construction du modèle d'optimisation de la construction de l'arbre Merkle dans le système Ethereum.

Je suis très reconnaissant au professeur Loic Colson de l'Université de Lorraine et au professeur François Charoy de l'Université de Lorraine pour leur temps et leur soutien à mon comité de suivi.

Mes sincères remerciements vont également à tous les membres du département d'IA du LGIPM, en particulier le professeur Hoai Minh Le, Phuc Hau, Van Tuan, Vinh Thanh, My Le, Khac Linh, Aurelié, Gaëlle, et Mme Greppi. Je suis reconnaissante pour votre amabilité et votre générosité, qui ont grandement contribué à rendre notre séjour au département d'IA du LGIPM mémorable. En particulier, j'ai consacré la majeure partie de mon temps au département d'IA en compagnie de Hau, Tuan et Le, qui m'ont apporté un soutien précieux tout au long de ma thèse.

Je souhaite exprimer ma gratitude à la Commission de sécurité des informations Gouvernmentales du Vietnam pour le financement de mes études en France. Je tiens à exprimer mes sincères remerciements à la direction de l'académie des techniques de cryptographie, aux enseignants du département de cryptographie, en particulier du département de science cryptographique, pour leur soutien et leur aide pendant ma thèse.

Je voudrais exprimer ma gratitude particulière à mes amis proches à Metz, Mme Hong Truong, Vinh

Nguyen, Huong Nguyen, Vy Phan, Nga Nguyen, Hoang Anh et Trinh Duong, pour leur soutien et leur gentillesse. Ils m'ont grandement aidé à m'adapter à la vie dans un nouveau pays et m'ont très bien traité.

Enfin et surtout, je voudrais exprimer ma profonde gratitude à ma famille au Vietnam, en particulier mon mari Van Toi Bui et ma fille Mai Chi Bui, qui m'ont toujours encouragé et cru pendant mes années de thèse. J'espère qu'ils sauront à quel point leurs encouragements sont précieux et que malgré la distance, leurs pensées m'ont ému.

Je souhaite dédier cette thèse à ma famille
qui m'a toujours soutenue tout au long de ma vie.

NGUYEN Thi Tuyet Trinh

Née le 18 septembre 1988 au Vietnam
E-mail: thi-tuyet-trinh.nguyen@univ-lorraine.fr , et nguyentuyettrinh189@gmail.com
Adresse professionnelle: Bureau UM-AN1-33, LGIPM - Université de Lorraine, 3 rue Augustin Fresnel, BP 45112, 57073 METZ Cedex 03, France

Situation Actuelle

Depuis 01/2021	Doctorante au LGIPM (Laboratoire de Génie Informatique, de Production et de Maintenance - EA 3096), Université de Lorraine, France Encadré par Prof. Hoai An Le Thi Sujet de thèse: **Programmation DC et DCA pour certaines classes de problèmes en cryptographie**

Experience Professionnelle

3/2013 - 12/2020	Enseignante-Chercheuse à l'Académie des Techniques de Cryptographie, Hanoi, Vietnam

Diplôme et Formation

2021 - present	Doctorante en Informatique, LGIPM, Université de Lorraine, Metz, France
2015 - 2017	Master en Techniques Cryptographiques, Académie des Techniques de Cryptographie, Hanoi, Vietnam
2008 - 2012	Diplôme Universitaire en Informatique et Technologie, Université Jiaotong de Pékin, Pékin, Chine

Publications

Referred international journals

[J1] H. A. Le Thi, T. T. T. Nguyen, & H. P. H. Luu (2022). A DC programming approach for solving a centralized group key management problem. *Journal of Combinatorial Optimization*, 1-29. https://doi.org/10.1007/s10878-022-00862-1.

[J2] H. A. Le Thi, & T. T. T. Nguyen (2023). Solving the problem of batch deletion and insertion members in the Logical Key Hierarchy structure by a DC Programming approach. *Submitted & Available on arXiv* arXiv:2305.10131.

Referred international conference papers

[C1] T. T. T. Nguyen, H. P. H. Luu, & H. A. Le Thi (2022). Solving a Centralized Dynamic Group Key Management Problem by an Optimization Approach. In: *Le Thi, H.A., Pham Dinh, T., Le, H.M. (eds) Modelling, Computation and Optimization in Information Systems and Management Sciences. MCO 2021. Lecture Notes in Networks and Systems*, vol 363. Springer, Cham.

[C2] T. T. T. Nguyen, H. A. Le Thi, & X.V. Doan (2023). Optimizing Merkle Tree Structure for Blockchain Transactions by a DC Programming Approach. In: *Nguyen, N.T., et al. Computational Collective Intelligence. ICCCI 2023. Lecture Notes in Computer Science()*, vol 14162. Springer, Cham.

[C3] T. T. T. Nguyen, & H. A. Le Thi (2023). A DCA-Like based algorithm for the Merkle tree construction problem in Ethereum cryptocurrency system. In: *Proceedings of the 4th International Conference and Summer School on Numerical Computations: Theory and Algorithms NUMTA 2023.* **Accepted for publication**.

Communications in national / international conferences

T. T. T. Nguyen, & H. A. Le Thi (2022). Solving the problem of batch deletion and insertion members in the Logical Key Hierarchy structure by a DC Programming approach. *32nd European Conference on Operational Research (EURO 2022)*, Espoo, Finland.

Contents

Introduction générale **xix**

Chapter 1
Preliminaries **1**

1.1 Fundamental convex analysis 1
1.2 DC programming and DCA 5
1.3 Accelerated DCA, DCA-Like and Accelerated DCA-Like 8
1.3.1 Accelerated DCA for solving the problem (1.9) 9
1.3.2 DCA-Like for solving the problem (1.9) 10
1.3.3 Accelerated DCA-Like for solving the problem (1.9) 11
1.4 Cryptography 12
1.4.1 Terms and definitions 12
1.4.2 Objectives of Cryptography 13
1.4.3 Classification of cryptographic algorithms 14
1.4.4 Preliminaries of Centralized group key management 16
1.4.5 Merkle tree in Ethereum system 18

Chapter 2
DC Programming and DCA for Dynamic Centralized Group Key Management problem

2.1 Introduction 22
2.2 Related works 25
2.3 Optimization models for the group key update problem using batch insertion and individual deletion techniques 27
2.3.1 Problem description 27
2.3.2 The first optimization model 27
2.3.3 A two-step algorithm and the second optimization model 32
2.4 DCA for solving the group key update problem using batch insertion and individual deletion techniques 33
2.4.1 DCA for solving the problem (2.5) 33
2.4.2 A two-step DCA based algorithm for solving the problem (2.7) 36
2.5 Two-step algorithm for the group key update problem using batch rekeying technique . . 37

2.5.1 Problem description . . . 37
2.5.2 The optimization model . . . 37
2.6 DCA for solving the group key update problem using batch rekeying technique . . . 41
2.7 Numerical experiments . . . 45
2.7.1 Dataset . . . 45
2.7.2 Comparative algorithms . . . 45
2.7.3 Set up experiments and Parameters . . . 46
2.7.4 Comparative results . . . 46
2.8 Conclusion . . . 52

Chapter 3
Advanced DCA based approaches for optimizing Merkle tree structure in blockchain transaction system

3.1 Introduction . . . 56
3.2 Optimization model for the problem of constructing Merkle tree in blockchain based system . . . 58
3.2.1 Problem definition . . . 58
3.2.2 The optimization model . . . 58
3.3 Solution methods based on DC programming and DCA for solving the problem (3.9) . . . 60
3.3.1 DCA for solving the problem (3.14) . . . 61
3.3.2 Accelerated DCA for solving the problem (3.14) . . . 61
3.3.3 DCA-Like for solving the problem (3.14) . . . 62
3.3.4 Accelerated DCA-Like for solving the problem (3.14) . . . 63
3.3.5 The recursive DCA-based approaches for solving the problem (3.10) . . . 64
3.4 Numerical experiments . . . 66
3.4.1 Experiment setting . . . 66
3.4.2 Comparative algorithms . . . 66
3.4.3 Comparative results . . . 66
3.5 Conclusion . . . 71

Chapter 4
A stochastic DCA based approach to Autoencoder architecture in Text Encryption and Decryption

4.1 Introduction . . . 73
4.2 Autoencoder for text encryption and decryption . . . 76
4.2.1 Autoencoder . . . 76
4.2.2 Proposed autoencoder architecture for text encryption and decryption . . . 79
4.2.3 MCSDCA optimizer [39, 53, 54] . . . 82
4.2.4 Proposed cryptosystem . . . 83
4.3 Numerical experiments . . . 85

4.3.1 Experimental setups 85
4.3.2 Experimental results 86
4.4 Conclusion 93

Chapter 5
Conclusion and perspectives

Bibliography **97**

List of Figures

1.1 A full binary tree structure. 16
1.2 The key tree after the member has been deleted [61]. 17
1.3 Adding a new member at the shallowest position [61]. 18
1.4 Illustration of a Merkle tree of 8 items. Merkle proof of item x_2 (with a dashed blue leaf) includes nodes h_3, h_{0-1}, and h_{4-7} (in solid red) [59]. 19

2.1 The tree structure after inserting new nodes. 29

3.1 Constructing Merkle tree for blockchain transactions using Algorithm 13. 65

4.1 Architecture of an autoencoder [76]. 77
4.2 The first autoencoder architecture of proposed system. 79
4.3 The second autoencoder architecture of proposed system [87]. 81
4.4 Function of text encryption in the GUI of the proposed autoencoder. 84
4.5 Function of text decryption in the GUI of the proposed autoencoder. 85
4.6 Training result over 100 epochs using autoencoder 1. 86
4.7 Training result over 100 epochs using autoencoder 2. 87
4.8 Training result over 100 epochs using autoencoder 3. 87
4.9 Training result over 100 epochs using autoencoder 4. 87
4.10 Training result over 100 epochs using autoencoder 5. 89
4.11 Training result over 100 epochs using autoencoder 6. 89
4.12 Training result over 100 epochs using autoencoder 7. 90
4.13 Training result over 100 epochs using autoencoder 8. 90
4.14 Encryption of 8-bit ASCII with trained weights of the autoencoder with one hidden layer that has a) 8, b) 9, c) 10 hidden neurons. 92
4.15 Encryption of chosen 8-bit ASCII with last bit changed in the autoencoder with one hidden layer that has a) 8, b) 9, c) 10 hidden neurons. 93

List of Tables

2.1 Comparison between DCAEP1 and DCAEP2 in terms of insertion cost, balance, and running time. . . . 47
2.2 Comparison between DCAEP+ and DCAEP2 in terms of rekeying cost, balance (BL), and running time (s). . . . 48
2.3 Comparison results in the case where the binary tree has height $= 8$, balance $= 5$, and the number of departure members $= 100$. . . . 49
2.4 Comparison results in the case where the binary tree has height $= 9$, balance $= 4$, and the number of departure members $= 300$. . . . 50
2.5 Comparison results in the case where the binary tree has height $= 10$, balance $= 4$, and the number of departure members $= 700$. . . . 50
2.6 Comparison results in the case where the binary tree has height $= 11$, balance $= 5$, and the number of departure members $= 1000$. . . . 51
2.7 Comparison results in the case where the binary tree has height $= 12$, balance $= 5$, and the number of departure members $= 3000$. . . . 52

3.1 Comparative results between the recursive DCA approach and recursive DCA-Like approach. Bold values indicate the best results. . . . 67
3.2 Comparative results of the recursive DCA, ADCA, DCA-Like, and ADCA-Like approaches. Bold values indicate the best results. . . . 68
3.3 Comparative results of the recursive ADCA-Like approach with other existing algorithms. Bold values indicate the best results. . . . 70

4.1 The hyperparameter list of proposed autoencoders. . . . 81
4.2 Training metrics for each optimizer in autoencoders with one hidden layer. Bold values indicate the best results. . . . 88
4.3 Validation loss/accuracy for 2669 samples and Testing loss/accuracy for 15865 samples in autoencoders with one hidden layer. Bold values indicate the best results. . . . 88
4.4 Training metrics for each optimizer in autoencoders with more than one hidden layer. Bold values indicate the best results. . . . 90
4.5 Validation loss/accuracy for 2669 samples and Testing loss/accuracy for 15865 samples in autoencoders with more than one hidden layer. Bold values indicate the best results. . . . 91

Abbreviations and Notations

The key abbreviations and notations used in this thesis are listed as follows.

3DES	triple data encryption standard
ADCA	accelerated difference of convex functions algorithm
AES	advanced encryption standard
ASCII	American standard code for information interchange
BCE	binary cross entropy
CGKM	centralized group key management
DC	Difference of convex functions
DCA	DC algorithm
DES	data encryption standard
DSA	digital signature algorithm
ECC	Elliptic curve cyptography
GUI	graphical user interface
LKH	logical key hierarchy
MD5	message-digest algorithm 5
NIST	national institute of standards and technology
RSA	Rivest, Shamir and Adleman
SGD	stochastic gradient descent
SHA-1	secure hash algorithm 1
SHA-2	secure hash algorithm 2
$\mathbb{R}$	set of real numbers
$\mathbb{R}^n$	set of real vectors of size n
$\|\cdot\|_0$	step function defined by $\|s\|_0 = 1$ if $s \neq 0$, 0 otherwise
$\langle .,. \rangle$	scalar product, $\langle x, y \rangle = \sum_{i=1}^{n} x_i.y_i, x, y \in \mathbb{R}^n$
$\chi_C(.)$	indicator function of a set $C, \chi_C(x) = 0$ if $x \in C, +\infty$ otherwise
co$\{C\}$	convex hull of a set of points C
$\nabla f(x)$	gradient of f at x
$\partial f(x)$	subdifferential of f at x
$\mathrm{dom}\, f$	effective domain of a function f
e	the vector of ones
I_a	$a-$by$-a$ identity matrix
$\|S\|$	cardinality of set S
O	big O notation

Introduction générale

Cadre général et nos motivations

La cryptographie est l'art des communications secrètes, c'est une science de la sécurité de l'information et de la communication. Ces techniques sont utilisées depuis des milliers d'années pour transformer un message en un texte chiffré illisible, de sorte que seul le destinataire prévu puisse le lire, évitant ainsi que le message ne soit intercepté par des personnes non autorisées et ne tombe entre de mauvaises mains. La cryptographie joue donc l'un des rôles les plus importants et les plus stimulants dans le monde numérique, où la sécurité des données est omniprésente. En effet, la cryptographie est largement utilisée pour l'authentification sur les cartes bancaires, les réseaux sociaux, le commerce électronique, les services de streaming et la technologie blockchain, etc. Outre les réseaux sociaux, des environnements plus sécurisés, tels que les réseaux militaires, exigent un niveau de confidentialité et de sécurité encore plus élevé pour la transmission des données, la gestion des membres et la gestion des clés. Ainsi, la cryptographie est l'étude des techniques mathématiques liées aux aspects de la sécurité de l'information tels que la confidentialité, l'intégrité des données, l'authentification des entités et l'authentification de l'origine des données.

L'optimisation joue un rôle crucial dans le domaine de la cryptographie, dont l'objectif est de développer et d'utiliser des protocoles et des algorithmes de communication sécurisés. L'objectif de l'optimisation cryptographique est d'améliorer l'efficacité et l'efficience des opérations cryptographiques sans compromettre leur sécurité. Il s'agit de trouver les algorithmes, les protocoles et les mises en œuvre les plus efficaces qui permettent de trouver un équilibre entre la sécurité et les ressources informatiques, de veiller à ce que les opérations cryptographiques soient exécutées efficacement et de fournir une protection solide contre les attaquants. En utilisant des techniques d'optimisation, la cryptographie est capable d'accomplir un chiffrement et un déchiffrement plus rapides, une gestion efficace des clés, une authentification sécurisée et d'autres fonctions essentielles pour la communication numérique sécurisée et la technologie blockchain.

Pour contribuer à ces directions de recherche, la conception de nouvelles techniques d'optimisation pour résoudre certaines classes de problèmes en cryptographie est le principal sujet d'étude de cette thèse.

Le premier problème que nous étudions est l'optimisation de la gestion centralisée des clés de groupe à l'aide de clés symétriques (clés partagées secrètement). En général, les objectifs d'optimisation comprennent la réduction du coût de la recomposition des clés, l'optimisation de la sélection des paramètres dans les systèmes de communication de groupe, l'optimisation de la structure de l'arbre des clés, l'optimisation de l'intervalle de temps entre les recompositions, etc. Toutefois, la majorité de ces objectifs n'ont pas été rigoureusement décrits sous la forme d'un modèle d'optimisation mathématique. En

outre, leurs algorithmes sont principalement basés sur des arguments logiques ou heuristiques.

Le deuxième problème concerne les arbres de Merkle, qui sont des composants essentiels de la technologie blockchain car ils permettent la vérification sécurisée des transactions et garantissent l'intégrité des données. L'arbre de Merkle d'État d'Ethereum, le deuxième plus grand réseau de blockchain, contient le solde actuel et d'autres données pertinentes pour chaque compte. Chaque bloc de la chaîne est associé à la racine de Merkle calculée pour l'état du réseau après l'exécution des transactions du bloc. Le processus d'approbation des blocs comprend la vérification de la validité des transactions et de leur impact sur l'état du réseau. Par exemple, un paiement requiert une valeur minimale du solde du payeur et, s'il est effectué, il implique une modification du solde des deux comptes concernés. En effet, les données de deux comptes et les nœuds correspondants de l'arbre de Merkle doivent être recalculés. Nous avons besoin d'une approche efficace et mathématique pour optimiser la structure de l'arbre de Merkle en fonction de la distribution des transactions. En réalité, différentes approches heuristiques ont été utilisées pour construire un arbre de Merkle pour une distribution de transactions spécifique.

Le troisième problème est le cryptage et le décryptage de texte à l'aide d'un autoencodeur. Les réseaux neuronaux sont apparus comme un outil puissant dans le domaine de la cryptographie, offrant de nouvelles approches pour un certain nombre de tâches cryptographiques. Les réseaux neuronaux sont des modèles informatiques qui s'inspirent de la structure et du fonctionnement du cerveau humain. Ils sont utilisés en cryptographie pour des tâches telles que le cryptage, le décryptage, la génération de clés et l'authentification. En outre, les systèmes de cryptage basés sur les réseaux neuronaux offrent un niveau élevé de résistance aux attaques car ils peuvent s'adapter et évoluer dans le temps. L'utilisation des réseaux neuronaux en cryptographie offre des possibilités intéressantes pour le développement de systèmes cryptographiques sûrs et efficaces, capables de résister aux attaques d'adversaires avancés. De nombreuses recherches ont été menées sur les autoencodeurs pour le cryptage et le décryptage de texte; cependant, ils doivent être améliorés en termes de performance et de sécurité.

La structure non convexe étudiée est la DC (différence de fonctions convexes). La programmation DC et DCA (DC Algorithm) considèrent le problème DC de la forme

$$\min_{x\in\mathbb{R}^n}\{f(x) := g(x) - h(x)\},$$

où $g, h : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ sont convexes, propres, semicontinus inférieurs. La fonction f est appelée fonction DC avec les composantes DC g et h, et $g - h$ est une décomposition DC de f. DCA est basé sur la dualité DC et des conditions d'optimalité locale. La construction de DCA implique les composantes DC g et h et non la fonction DC f elle-même. Chaque fonction DC admet une infinité des décompositions DC qui influencent considérablement sur la qualité (la rapidité, l'efficacité, la globalité de la solution obtenue, etc.) de DCA. Ainsi, au point de vue algorithmique, la recherche d'une "bonne" décomposition DC et d'un "bon" point initial est très importante dans le développement de DCA pour la résolution d'un programme DC.

L'utilisation de la programmation DC et DCA dans cette thèse est justifiée par de multiple arguments [47]:

- DCA a été appliqué avec succès à de nombreux programmes non convexes de grandes dimensions dans divers domaines des sciences appliquées (voir [71, 72, 74, 73, 40, 46, 47] et ses références).
- DCA est une philosophie plutôt qu'un algorithme. Pour chaque problème, nous pouvons con-

cevoir une famille d'algorithmes basés sur DCA. La flexibilité de DCA sur le choix des décomposition DC peut offrir des schémas DCA plus performants que des méthodes standards.

- L'analyse convexe fournit des outils puissants pour prouver la convergence de DCA dans un cadre général. Ainsi tous les algorithmes basés sur DCA bénéficient (au moins) des propriétés de convergence générales du schéma DCA générique qui ont été démontrées.
- DCA est une méthode efficace, rapide et scalable pour la programmation non convexe. A notre connaissance, DCA est l'un des rares algorithmes de la programmation non convexe, non différentiable qui peut résoudre des programmes DC de très grande dimension.

Il est important de noter qu'avec les techniques de reformulation en programmation DC et les décompositions DC appropriées, on peut retrouver la plupart des algorithmes existants en programmation convexe/non convexe comme cas particuliers de DCA.

Nos contributions

Les principales contributions de la thèse sont de construire des modèles d'optimisation et d'appliquer des approches basées sur la DCA pour résoudre des problèmes dans plusieurs aspects de la cryptographie, en particulier la gestion centralisée des clés de groupe, la conception de l'arbre de Merkle dans les systèmes de transaction blockchain, et l'architecture autoencodeur pour le cryptage et le décryptage de texte. Tous ces problèmes peuvent être formulés comme la minimisation d'une fonction DC. Il en résulte un programme DC standard qui peut être résolu efficacement par le DCA et d'autres algorithmes DC avancés.

Dans le deuxième chapitre, nous avons proposé des approches d'optimisation du problème de la mise à jour de la clé de groupe dans la structure LKH (Logical Key Hierarchy en anglais) avec deux techniques de recomposition différentes, à savoir l'insertion par lots et la recomposition par lots. Tout d'abord, nous optons pour la technique d'insertion par lots afin de préserver le secret du problème de mise à jour de la clé de groupe. Il s'agit du premier modèle d'optimisation qui prend en compte simultanément le coût de mise à jour des clés et l'équilibre de l'arbre de clés résultant. Le problème d'optimisation proposé comporte des variables binaires et une fonction objective discontinue. Il est d'abord formulé de manière équivalente pour éliminer les fonctions de pas de l'objectif. Grâce aux résultats récents sur les techniques de pénalités exactes dans la programmation DC, ce dernier problème peut être reformulé comme un programme DC. Nous avons ensuite conçu un DCA efficace pour résoudre ce problème. En outre, nous avons proposé un algorithme en deux étapes plus efficace qui intègre un algorithme pour trouver tous les nœuds feuilles dans la première étape, ce qui simplifie le modèle d'optimisation dans la deuxième étape. Ce problème simplifié peut toujours être résolu en utilisant la même approche basée sur l'DCA que le problème original. En outre, nous avons proposé une approche d'optimisation du problème de la suppression et de l'insertion par lots de membres dans CGKM (Centralized Group Key Management en anglais). Notre objectif principal est de minimiser simultanément le coût de la suppression et de l'ajout de nœuds tout en maintenant un arbre équilibré. Le problème d'optimisation proposé comporte des variables binaires et une fonction objective discontinue.

Dans le troisième chapitre, nous avons proposé un modèle d'optimisation pour la construction de l'arbre de Merkle dans le système de transaction Ethereum. À notre connaissance, il s'agit du premier

modèle mathématique permettant de résoudre le problème de la structure de l'arbre de Merkle sur la base de la distribution des transactions. Le problème d'optimisation proposé est un programme quadratique binaire. Grâce aux résultats récents sur les techniques de pénalités exactes dans la programmation DC, le problème est reformulé comme un programme DC. Nous avons ensuite appliqué une DCA efficace pour résoudre ce problème. En outre, nous déployons ce que l'on appelle le DCA-Like, qui est basé sur une nouvelle méthode efficace d'approximation de la fonction objective DC sans connaître sa décomposition DC [41, 35]. DCA-Like est "comme" DCA dans le sens où il approxime itérativement le programme DC par une séquence de programmes convexes. Cependant, DCA-Like est "différent" de DCA dans la façon dont il approxime la fonction objective, pour laquelle nous n'avons peut-être pas de décomposition DC. Alors que l'DCA standard fonctionne avec une majorisation convexe de la fonction objective sur l'ensemble de l'espace via une décomposition DC disponible, l'DCA-Like recherche une meilleure approximation convexe de la solution actuelle via une décomposition qui n'est pas nécessairement DC. Ainsi, DCA-Like fonctionne même lorsqu'il n'est pas possible de mettre en évidence une décomposition DC. DCA-Like assouplit la condition de convexité de la deuxième composante DC, ce qui conduit à un majorant plus proche de la fonction objective et potentiellement à une meilleure solution. Parallèlement, nous déployons Accelerated DCA (ADCA) et Accelerated DCA-Like (ADCA-Like) pour améliorer DCA et DCA-Like en y incorporant la technique d'accélération de Nesterov [68, 35]. Plus précisément, l'étape d'accélération de l'ADCA vise à trouver un point z^k qui est un point extrapolé de l'itération courante x^k et de l'itération précédente x^{k-1} via la formulation d'accélération de Nesterov.

En outre, nous avons proposé une approche d'optimisation plus efficace pour construire un arbre de Merkle à partir d'un grand nombre de comptes de blockchain. Notre méthode sépare les comptes en petits sous-groupes et construit récursivement l'arbre de Merkle pour tous les comptes en utilisant alternativement DCA et d'autres algorithmes DC avancés (ADCA, DCA-Like, et ADCA-Like). Des expériences numériques sur plusieurs ensembles de données montrent l'efficacité de nos approches en termes de qualité des solutions et de temps d'exécution.

Dans le quatrième chapitre, nous proposons une approche basée sur l'apprentissage profond utilisant des réseaux neuronaux, qui représente le prochain développement dans le domaine de la cryptographie. L'objectif principal de notre travail est de développer un autoencodeur utilisé pour crypter et décrypter en toute sécurité des messages textuels représentés dans un format de 8 bits, qui correspond à un caractère ASCII. Dans l'autoencodeur proposé, nous utilisons la fonction de perte BCE (Binary Cross Entropy) combinée à la fonction de pénalité afin d'obtenir des valeurs de sortie identiques aux entrées binaires. En outre, nous appliquons un nouvel algorithme stochastique appelé Markov chain stochastic DCA (MCSDCA) [39, 53, 54] comme optimiseur dans diverses architectures d'autoencodeur. Les expériences numériques montrent les vertus de MCSDCA par rapport à d'autres optimiseurs de base (tels qu'Adam et SGD) dans l'apprentissage profond. Cela suggère que la méthode proposée a un haut degré de confidentialité parce que chaque processus de formation produit toujours un ensemble différent et important de poids formés.

Organisation de la thèse

La thèse se compose de cinq chapitres. Le chapitre 1 contient des informations préliminaires sur la programmation DC, DCA et d'autres algorithmes DC avancés, ainsi que des informations préliminaires

sur la cryptographie. Dans le chapitre 2, nous étudions le problème d'optimisation de la mise à jour de la clé de groupe dans la gestion centralisée dynamique des clés de groupe avec deux techniques différentes : l'insertion par lots et la recomposition par lots. Le modèle d'optimisation proposé est un problème combinatoire avec une fonction objective non convexe et des variables binaires. En utilisant des techniques de pénalités exactes, ce problème peut être reformulé sous la forme d'un programme DC qui peut être résolu efficacement par la DCA. Le chapitre 3 examine un programme quadratique binaire pour construire la structure de l'arbre de Merkle dans un système de transaction blockchain. Dans le chapitre 4, le DCA stochastique de la chaîne de Markov est utilisé en tant qu'optimiseur dans un autoencodeur sécurisé et efficace pour le cryptage et le décryptage de texte. Enfin, le chapitre 5 conclut la thèse.

Chapter 1

Preliminaries

1.1 Fundamental convex analysis

We begin with a review of Convex Analysis and Nonsmooth Analysis notions that will be beneficial later on (see, e.g., [78, 79, 18, 15]).
Let X represent the Euclidean space $\mathbb{R}^n$, and let $\overline{\mathbb{R}} = \mathbb{R} \cup \{\pm\infty\}$ represent the set of extended real numbers. A subset C of X is *convex* if $(1-\lambda)x + \lambda y \in C$ for any $x, y \in C$ and any $\lambda \in [0,1]$.
$\operatorname{conv} C$ represents the convex hull of a given set C, which is the set of all convex combinations of points in C

$$\operatorname{conv} C := \{\lambda_1 x_1 + \lambda_2 x_2 + ... + \lambda_k x_k : x_i \in C, \lambda_i \geq 0, \forall i = \overline{1,k}, \lambda_1 + \lambda_2 + ... + \lambda_k = 1\}.$$

Let C be a convex set. A function $f : C \to (-\infty, +\infty]$ is convex on C if

$$f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda) f(y), \text{ for all } x, y \in C, \lambda \in [0,1].$$

If the inequality above holds strictly whenever $x \neq y$ and $0 < \lambda < 1$, then a real-valued function f on a convex set C is said to be *strictly convex* on C.
Denoted by $\operatorname{dom} f$, the *effective domain* of a convex function f on C is defined as

$$\operatorname{dom} f := \{x \in X : \quad f(x) < +\infty\}.$$

It is evident that the domain of function f is a convex set in X.
The convex function f is said to be *proper* if $\operatorname{dom} f \neq \emptyset$ and $f(x) > -\infty$ for all $x \in C$.
The function $f : C \to (-\infty, +\infty]$ is said to be *lower semicontinuous* at a point x of C if

$$f(x) \leq \liminf_{y \to x} f(y).$$

Denote by $\Gamma_0(X)$ the set of all proper lower semicontinuous convex functions on X.
Let ρ be a nonnegative number and C be a convex subset of X. A function $\theta : C \to (-\infty, +\infty]$ is $\rho-$ convex if

$$\theta[\lambda x + (1-\lambda)y] \leq \lambda\theta(x) + (1-\lambda)\theta(y) - \frac{\lambda(1-\lambda)}{2}\rho\|x-y\|^2$$

for all $x, y \in C$ and $\lambda \in (0,1)$. It amounts to say that $\theta - (\rho/2)\|\cdot\|^2$ is convex on C.
The modulus of strong convexity of θ on C, denoted by $\rho(\theta, C)$ or $\rho(\theta)$ if $C = X$, is given by

$$\rho(\theta, C) = \sup\{\rho \geq 0 : \theta - (\rho/2)\|\cdot\|^2 \text{ is convex on } C\}.$$

One says that θ is *strongly convex* on C if $\rho(\theta, C) > 0$.
A vector y is said to be a *subgradient* of a convex function f at a point x^0 if

$$f(x) \geq f(x^0) + \langle x - x^0, y\rangle, \quad \forall x \in X.$$

The set of all subgradients of f at x^0 is called the *subdifferential* of f at x^0, denoted by $\partial f(x^0)$,

$$\partial f(x^0) := \{y \in X : f(x) - f(x^0) \geq \langle x - x^0, y\rangle, \quad \forall x \in X\}. \tag{1.1}$$

A function f is considered to be *subdifferentiable* at x if the set $\partial f(x)$ is not empty.
For $\epsilon > 0$, a vector y is said to be an $\epsilon-$*subgradient* of a convex function f at a point x^0 if

$$f(x) \geq (f(x^0) - \epsilon) + \langle x - x^0, y\rangle, \quad \forall x \in X.$$

The set of all $\epsilon-$subgradients of f at x^0 is called the $\epsilon-$*subdifferential* of f at x^0 and is denoted by $\partial_\epsilon f(x^0)$.
For $\epsilon \geq 0$, a point x_ϵ is called an $\epsilon-$solution of the problem $\inf\{f(x) : x \in \mathbb{R}^d\}$ if

$$f(x_\epsilon) \leq f(x) + \epsilon, \quad \forall x \in \mathbb{R}^d.$$

Let us now describe two fundamental notations.

$$\operatorname{dom}\partial f = \{x \in X : \partial f(x) \neq \emptyset\} \text{ and range } \partial f(x) = \cup\{\partial f(x) : x \in \operatorname{dom}\partial f\}.$$

Proposition 1. *Let f be a proper convex function. Then*

1. *$\partial_\epsilon f(x)$ is a closed convex set, for any $x \in X$ and $\epsilon \geq 0$.*
2. *$\operatorname{ri}(\operatorname{dom} f) \subset \operatorname{dom}\partial f \subset \operatorname{dom} f$ where $\operatorname{ri}(\operatorname{dom} f)$ stands for the relative interior of $\operatorname{dom} f$.*
3. *If f is differentiable at $x \in \operatorname{dom} f$ then $\partial f(x) = \{\nabla f(x)\}$.*
4. *$x_0 \in \arg\min\{f(x) : x \in X\}$ if and only if $0 \in \partial f(x_0)$.*

Let C be a nonempty convex subset of $\mathbb{R}^n$. The *indicator function* of C, denoted by χ_C, is the function

$$\chi_C(x) = \begin{cases} 0 & \text{if } x \in C, \\ +\infty & \text{otherwise.} \end{cases}$$

The *normal cone* of C at $x \in C$, denoted $N_C(x)$, is given by

$$N_C(x) = \partial_{\chi_C}(x) = \{u \in \mathbb{R}^n : \langle u, y - x\rangle \leq 0, \quad \forall y \in C\}.$$

A function $f : \mathbb{R}^n \to \mathbb{R}^m$ is considered to be *Lipschitz continuous* on C if there exists a non-negative real number L such that

$$\|f(x_1) - f(x_2)\| \leq L\|x_1 - x_2\|, \quad \forall x_1, x_2 \in C.$$

This number L is called a Lipschitz constant of f on C.

A function $f : \mathbb{R}^n \to (-\infty, +\infty]$ is said to be locally Lipschitz at $x \in \mathbb{R}^n$ if there exists a neighborhood U_x of x such that f is Lipschitz continuous on U_x.

The definition of the *Fréchet subdifferential* of f at $x \in \operatorname{dom} f$ for a lower semicontinuous function is as follows:

$$\partial^F f(x) = \left\{ z \in \mathbb{R}^n : \ \liminf_{h \to 0} \frac{f(x+h) - f(x) - \langle z, h \rangle}{\|h\|} \geq 0 \right\}.$$

For $x \notin \operatorname{dom} f$, we set $\partial^F f(x) = \emptyset$. Note that, even if $x \in \operatorname{dom} f$, $\partial^F f(x)$ can be empty, e.g., take $f(u) = -|u|$ where $u \in \mathbb{R}$. The set $\partial^F f(x)$ is a closed and convex set. Nevertheless, it should be noted that the Fréchet subdifferential does not exhibit the property of being a closed mapping, hence resulting in computational instability [14]. Due to this reason, the subsequent *limiting subdifferential* is viewed as a "limited variant" of the Fréchet subdifferential. By definition, the limiting subdifferential of f at $x \in \operatorname{dom} f$ is

$$\partial f(x) = \{z \in \mathbb{R}^n : \ \exists (x_k, f(x_k)) \to (x, f(x)), \ z_k \in \partial^F f(x_k), \ \ z_k \to z\},$$

and we put $\partial f(x) = \emptyset$ if $x \notin \operatorname{dom} f$. Now, the limiting subdifferential is a closed mapping. However, the set $\partial f(x)$ is closed but not necessarily convex.

Let $f : \mathbb{R}^n \to (-\infty, +\infty]$ be a locally Lipschitz function at a given $x \in \mathbb{R}^n$. The *Clarke directional derivative* and the *Clarke subdifferential* of f at x is given by the following formulas.

$$f^C(x, d) = \limsup_{(t,u) \to (0^+, x)} \frac{f(u + td) - f(u)}{t}$$
$$\text{and } \partial^C f(x) = \{y \in \mathbb{R}^n : \langle y, d \rangle \leq f^C(x, d), \quad \forall d \in \mathbb{R}^n\}.$$

If f is continuously differentiable at x then $\partial^C f(x) = \nabla f(x)$. Then f is a convex function, then $\partial^C f(x)$ coincides with the subdifferential $\partial f(x)$.

The following inclusions describe the relations between Fréchet, limiting, and Clarke subdifferentials:

$$\partial^F f(x) \subset \partial f(x) \subset \partial^C f(x).$$

When f is a convex function, the Fréchet, limiting, and Clarke subdifferential coincide with the Convex Analysis subdifferential (1.1).

Conjugates of convex functions

The *conjugate* of a function $f : X \to \overline{\mathbb{R}}$ is the function $f^* : X \to \overline{\mathbb{R}}$, defined by

$$f^*(y) = \sup_{x \in X}\{\langle x, y\rangle - f(x)\}.$$

Let $f \in \Gamma_0(\mathbb{R}^n)$, then

$$y \in \partial f(x) \Leftrightarrow x \in \partial f^*(y). \tag{1.2}$$

Proposition 2. *Let $f \in \Gamma_0(X)$. Then we have*

1. *$f^* \in \Gamma_0(X)$ and $f^{**} = f$.*
2. *$f(x) + f^*(y) \geq \langle x, y\rangle$, for any $x, y \in X$.*
3. *$f(x) + f^*(y) = \langle x, y\rangle \Leftrightarrow y \in \partial f(x) \Leftrightarrow x \in \partial f^*(y)$.*

Polyhedral Functions

A *polyhedral* set is defined as a closed convex set that is of the form

$$C = \{x \in X : \langle b_i, x\rangle \leq \beta_i, \quad \forall i = \overline{1, m}\},$$

where $b_i \in X$ and $\beta_i \in \mathbb{R}$ for $i = \overline{1, m}$.
A function $f \in \Gamma_0(X)$ is said to be *polyhedral* if it satisfies the following condition:

$$f(x) = \max\{\langle a_i, x\rangle - \alpha_i : i = 1, ..., k\} + \chi_C(x), \quad \forall x \in X, \tag{1.3}$$

where $a_i \in X, \alpha_i \in \mathbb{R}$ for all $i = \overline{1, k}$ and C is nonempty polyhedral set. It is clear that $\operatorname{dom} f = C$.

Proposition 3. *[29] Let f be a polyhedral convex function, and $x \in \operatorname{dom} f$. Then we have*

1. *f is subdifferentiable at x, and $\partial f(x)$ is a polyhedral convex set. In particular, if f is defined by* (1.3) *with $C = X$ then*

$$\partial f(x) = \operatorname{co}\{a_i : i \in I(x)\},$$

where $I(x) = \{i \in \{1, ..., k\} : \langle a_i, x\rangle - \alpha_i = f(x)\}$.

2. *The conjugate f^* is a polyhedral convex function. Moreover, if $C = X$ then*

$$\operatorname{dom} f^* = \operatorname{co}\{a_i : i = 1, ..., k\},$$
$$f^*(y) = \inf\left\{\sum_{i=1}^k \lambda_i \alpha_i : \sum_{i=1}^k \lambda_i a_i = y, \sum_{i=1}^k \lambda_i = 1, \lambda_i \geq 0, \forall i = 1, ..., k\right\}.$$

In particular,

$$f^*(a_i) = \alpha_i, \forall i = 1, ..., k.$$

1.2 DC programming and DCA

Standard DC program

Recall that $\Gamma_0(\mathbb{R}^n)$ denotes the convex cone of all proper, lower semicontinuous, convex functions on $\mathbb{R}^n$. The standard DC program takes the form [71]

$$(P_{dc}) \qquad \alpha := \inf\{f(x) := g(x) - h(x) : x \in \mathbb{R}^n\}$$

where $g, h \in \Gamma_0(\mathbb{R}^n)$. Such a function f is called DC, $g - h$ is DC *decomposition* while g and h are DC *components* of f. It is worth noting that a DC function has infinitely many DC decompositions. We may assume that g and h are strongly convex without losing generality since f can be recast as the difference of two strongly convex functions as follows

$$f = \left(g + \frac{\rho}{2}\|\cdot\|^2\right) - \left(h + \frac{\rho}{2}\|\cdot\|^2\right), \quad \text{with } \rho > 0.$$

According to [46], the class of DC functions is of significant magnitude, making it capable of encompassing a vast majority of real optimization issues. In addition, it has been shown that a sequence of difference of convex (DC) functions may be used to provide a uniform approximation of any continuous function inside a bounded set [5]. The set of DC functions exhibits closure under commonly used operators in optimization, such as linear combination, maximum, and minimum. Specifically, when a finite number of DC functions are subjected to these operators, the resulting functions remain inside the class of DC functions. Nevertheless, it should be noted that the category of DC functions exhibits instability when subjected to pointwise convergence and the supremum operator [19].

A DC program that includes a convex constraint $x \in C$ may be expressed as a standard DC program (P_{dc}) by including the indicator function of C, denoted as χ_C, into the first DC component. This can be achieved by defining $\hat{g}$ as the sum of the first DC component g and χ_C. When the minimization of a DC function is performed over nonconvex DC constraints, the resulting optimization problem deviates from a standard DC program. Instead, it is often known as a general DC program [37, 73]. In this thesis, we are mostly concerned with standard DC programs.

The dual DC program of (P_{dc}) is defined by

$$(D_{dc}) \qquad \beta := \inf\{h^*(y) - g^*(y) : y \in \mathbb{R}^n\}.$$

It can be verified that $\alpha = \beta$ and there is the perfect symmetry of the DC duality: the dual of (D_{dc}) is exactly (P_{dc}).

Polyhedral DC program

In the DC problem (P_{dc}), if one of the DC components g and h is a polyhedral convex function, (P_{dc}) is called a *polyhedral DC program*. Polyhedral DC program is an important class of DC optimization which is often encountered in practice and has interesting properties.

Consider the case where h is a polyhedral convex function defined as

$$h(x) = \max\{\langle a_i, x\rangle - \alpha_i : i = 1, ..., k\}.$$

By Proposition 3, the dual problem (D_{dc}) of (P_{dc}) can be written as

$$\alpha = \inf\{\alpha_i - g^*(a_i) : i = 1, ..., k\}.$$

In the case g is polyhedral convex and h is not polyhedral, the dual problem (D_{dc}) also has the similar formulation as above since g^* is polyhedral.

Optimality conditions for DC optimization

A point x^* is said to be a *local minimizer* of $g - h$ if $x^* \in \operatorname{dom} g \cap \operatorname{dom} h$ and there is a neighborhood U of x^* such that

$$g(x) - h(x) \geq g(x^*) - h(x^*), \quad \forall x \in U. \tag{1.4}$$

A point x^* is said to be a *critical point* of $g - h$ if

$$\partial g(x^*) \cap \partial h(x^*) \neq \emptyset. \tag{1.5}$$

Optimality conditions for DC standard programs are shown in the following theorem (see [71]).

Theorem 4. *i) Global optimality condition: x^* is an optimal solution of the problem (P_{dc}) if and only if*

$$\partial_\epsilon h(x) \subset \partial_\epsilon g(x), \forall \epsilon > 0.$$

ii) Necessary condition for local optimality: if x^* is a local minimizer of $g - h$, then

$$\partial h(x^*) \subset \partial g(x^*).$$

iii) Sufficient condition for local optimality: Let x^* be a critical point of $g - h$ and $y^* \in \partial g(x^*) \cap \partial h(x^*)$. Let U be a neighborhood of x^* such that $(U \cap \operatorname{dom} g) \subset \operatorname{dom} \partial h$. If for any $x \in U \cap \operatorname{dom} g$, there is $y \in \partial h(x)$ such that $h^*(y) - g^*(y) \geq h^*(y^*) - g^*(y^*)$, then x^* is a local minimizer of $g - h$. More precisely,

$$g(x) - h(x) \geq g(x^*) - h(x^*), \quad \forall x \in U \cap \operatorname{dom} g.$$

Remark 1. *a) By the symmetry of the DC duality, Theorem 4 has its corresponding dual part.*

b) The necessary local optimality condition $\partial h^*(x^*) \subset \partial g^*(x^*)$ is also sufficient for many important classes programs, for example, if h is polyhedral convex, or when f is locally convex at x^*, i.e. there exists a convex neighborhood U of x^* such that f is finite and convex on U. We know that a polyhedral convex function is differentiable everywhere except on a set of measure zero. Thus, if h is a polyhedral convex function, then a critical point of $g - h$ is almost always a local solution to (P_{dc}).

c) If $f = g - h$ is actually convex on $\mathbb{R}^n$, we call (P_{dc}) a "false" DC program. In addition, if $\operatorname{ri}(\operatorname{dom} g) \cap \operatorname{ri}(\operatorname{dom} h) \neq \emptyset$ and $x^* \in \operatorname{dom} g$ such that g is continuous at x^*, then $0 \in \partial f(x^*) \Leftrightarrow \partial h(x^*) \subset \partial g(x^*)$. Thus, in this case, the local optimality is also sufficient for the global optimality. If, in addition, h is differentiable, a critical point is also a global solution.

DC Algorithm

The DCA (see [75, 71, 72, 45, 46, 47] and references therein) is a powerful tool for solving DC programs, particularly when the DC decomposition is tailored. DCA is based on local optimality conditions and duality in DC programming, which introduces the nice and elegant concept of approximating a DC program by a sequence of convex ones. At each iteration k, DCA approximates the second DC component h by its affine minorization $h_k(x) := h(x^k) + \langle x - x^k, y^k \rangle$, with $y^k \in \partial h(x^k)$, and solves the resulting convex subprogram

$$(P_k) \quad \alpha_k := \inf\{f(x) := g(x) - h_k(x) : x \in \mathbb{R}^n\} \tag{1.6}$$

whose x^{k+1} is a solution, i.e., $x^{k+1} \in \partial g^*(y^k)$.

Standard DCA.

Initialization: Let $x^0 \in \operatorname{dom} \partial h$ and $k = 0$.

repeat

Step 1: Compute the subgradient $y^k \in \partial h(x^k)$.

Step 2: Solve the convex program $x^{k+1} \in \arg\min\{g(x) - h_k(x) : x \in \mathbb{R}^n\}$.

Step 3: $k = k + 1$.

until Stopping criterion.

The reader is referred to [71, 45] for information regarding the convergence properties of the DCA and the theoretical underpinning of DC programming. In Theorem 5, some fundamental properties of the sequence generated by DCA are described.

Theorem 5. *([71]) The sequences $\{x^k\}$ and $\{y^k\}$ generated by DCA have the following properties:*

1. *The sequences $\{g(x^k) - h(x^k)\}$ and $\{h^*(y^k) - g^*(y^k)\}$ are decreasing and*

   - *$g(x^{k+1}) - h(x^{k+1}) = g(x^k) - h(x^k)$ if and only if $\{x^k, x^{k+1}\} \subset \partial g^*(y^k) \cap \partial h^*(y^k)$ and $[\rho(h) + \rho(g)]\|x^{k+1} - x^k\| = 0$.*

   - *$h^*(y^{k+1}) - g^*(y^{k+1}) = h^*(y^k) - g^*(y^k)$ if and only if $\{y^k, y^{k+1}\} \subset \partial g(x^k) \cap \partial h(x^k)$ and $[\rho(h^*) + \rho(g^*)]\|y^{k+1} - y^k\| = 0$.*

   DCA terminates at the kth iteration if either of the above equalities holds.

2. *If $\rho(h) + \rho(g) > 0$ (resp. $\rho(h^*) + \rho(g^*) > 0$), then the sequence $\{\|x^{k+1} - x^k\|^2\}$ (resp. $\{\|y^{k+1} - y^k\|^2\}$) converges.*

3. *If the optimal value α of the problem (P_{dc}) is finite and the sequences $\{x^k\}$ and $\{y^k\}$ are bounded, then every limit point of $\{x^k\}$ is a critical point of (P_{dc}), and every limit point of $\{y^k\}$ is a critical point of (D_{dc}).*

4. *DCA has a linear convergence for general DC program.*

5. *If (P_{dc}) is a polyhedral DC program, the sequence $\{x^k\}$ contains finitely many elements and DCA has a finite convergence: the sequence $\{x^k\}$ has a subsequence that converges to a DC critical point of (P_{dc}) after a finite number of iterations.*

6. *If DCA converges to a point x^* that admits a convex neighborhood in which the objective function f is finite and convex (i.e. the function f is locally convex at x^*) and if the second DC component h is differentiable at x^*, then x^* is a local minimizer to the problem (P_{dc}).*

A duality point of view [71, 45]

Step 1 of the DCA, $y^k \in \partial h(x^k)$, is equivalent to $x^k \in \partial h^*(y^k)$ thanks to (1.2), or

$$h^*(z) \geq h^*(y^k) + \langle x^k, z - y^k \rangle, \quad \forall z \in \mathbb{R}^n, \tag{1.7}$$

which is further equivalent to

$$y^k \in \arg\min\{h^*(z) - \langle x^k, z\rangle : z \in \mathbb{R}^n\}.$$

Likewise, Step 2 of the DCA, $x^{k+1} \in \arg\min\{g(x) - \langle y^k, x\rangle\}$, is equivalent to $y^k \in \partial g(x^{k+1})$. Thanks to (1.2), it is further equivalent to

$$x^{k+1} \in \partial g^*(y^k). \tag{1.8}$$

From (1.7) and (1.8), we have the following observation: if $\{(x^k, y^k)\}$ is the sequence generated by DCA applied to the primal DC program (P_{dc}), then $\{(y^k, x^{k+1})\}$ is the sequence generated by DCA applied to the dual DC program (D_{dc}). Consequently, for each convergence property of the primal sequence $\{x^k\}$, there is a corresponding convergence property for the dual sequence $\{y^k\}$.

DC programming and DCA are acknowledged as effective nonconvex optimization tools due to their robustness and performance in comparison to existing methods, their speed and scalability, and the adaptability of DC decompositions. Their effectiveness and popularity are contingent upon their extraordinary simplicity, rigorous mathematical foundations, and pervasiveness in optimization (see [47]).

1.3 Accelerated DCA, DCA-Like and Accelerated DCA-Like

Recent works in DC programming and DCA aim to address the following important open questions: improving DCA convergence speed; the development of efficient solvers for solving convex sub-problems; the search for good DC decompositions using regularization and decomposition techniques, and the search for good convex approximations of the DC objective function; the development of DCA schemes adapted to problems with big data; the use of DCA without highlighting a DC decomposition of the objective function; and the extension of DCA to the resolution of non-convex programs beyond DC programming [47]. As a result, advancements in DCA development involve the creation of novel DCA variants that enhance the standard DCA to address some of these questions.

Accelerated DCA (ADCA) was initially introduced in [68], which was the first to incorporate Nesterov's acceleration technique into standard DCA. To improve the convergence speed of DCA, rather than performing the standard DCA calculation of $y^k \in \partial h(x^k)$, the authors aim to identify a point v^k that is "more promising" than x^k in the sense that v^k is better than one of the last q iterations $\{x^{k-q}, ..., x^{k-1}, x^k\}$ in terms of objective function, i.e., $f(v^k) \leq \max_{t=\max(0,k-q),...,k} f(x^t)$. This point is extrapolated from the current iteration x^k and the previous iteration x^{k-1} using Nesterov's formulation. If v^k is not "more promising" than x^k, x^k is maintained to be v^k. Subsequently, $y^k \in \partial h(v^k)$

is obtained. It is evident that the sequence $\{f(x^k)\}$ decreases in accordance with standard DCA when $q = 0$. In addition, when q is greater than zero, ADCA has the capability to enhance the objective function and thus avoid a possible bad local minimum. A high value of q increases the chances that extrapolation points will be utilized in ADCA, in theory.

The second approach, called DCA-Like, is presented in [41, 35]. DCA-Like is an effective approach that applies the DCA philosophy without emphasizing a DC decomposition of the objective function. Instead, it provides a suitable convex surrogate that is adapted for each iteration. It possesses two significant advantages in comparison to standard DCA. It is noted that DCA-Like approximates the DC program (P_{dc}) iteratively using a sequence of convex ones, which is "like" DCA. On the other hand, DCA-Like is "unlike" DCA in that it relaxes two of its most important requirements [41, 35]:

i) In the decomposition $f = g_k - h_k$, h_k is not necessarily convex (i.e. one possibly does not have a DC decomposition of f);

ii) The affine minorant of h_k may not be a lower bound of h_k on the whole space but rather only at the current solution.

It is proved that, fortunately, in spite of these modifications, the whole bounded sequence $\{x_k\}$ generated by DCA-Like still converges to a critical point of the standard DC program under some conditions. Furthermore, under Kurdyka-Łojasiewicz assumption, the convergence rate of DCA-Like is at least sublinear $O(1/k^\alpha)$ with $\alpha > 1$. An acceleration version of DCA-Like, called ADCA-Like, is also introduced to improve the convergence speed.

In [68, 35], the authors considered the sum of two nonconvex functions minimization problem, which takes the form

$$\min_{x\in\mathbb{R}^n} F(x) = f(x) + r(x), \tag{1.9}$$

where $f(x)$ is a differentiable non-convex function with $L-$Lipschitz continuous gradient and $r(x)$ is a DC function.

Since f is a function with $L-$Lipschitz continuous gradient, it can be expressed as a DC function: $f(x) = \frac{\rho}{2}\|x\|^2 - [\frac{\rho}{2}\|x\|^2 - f(x)]$, where $\rho \geq L$. Let $r(x) := g_r(x) - h_r(x)$, then a DC decomposition of $F(x) = f(x) + r(x)$ is given by

$$F(x) = G(x) - H(x), \tag{1.10}$$

where $G(x) = \frac{\rho}{2}\|x\|^2 + g_r(x)$ and $H(x) = \frac{\rho}{2}\|x\|^2 - f(x) + h_r(x)$. Therefore, DCA, Accelerated DCA, DCA-Like, and ADCA-Like can be applied to solve (1.9).

1.3.1 Accelerated DCA for solving the problem (1.9)

The Accelerated DCA, also known as ADCA, involves the integration of Nesterov's acceleration technique, originally designed for convex algorithms, into the standard DCA [68, 35]. To be more explicit, the objective of the acceleration step in ADCA is to determine a point denoted as z^k. This point is obtained by extrapolating the current iteration x^k and the preceding iterate x^{k-1} using Nesterov's acceleration formulation

$$z^k = x^k + \frac{t_k - 1}{t_{k+1}}(x^k - x^{k-1}),$$

where $t_{k+1} = \frac{1+\sqrt{1+4t_k^2}}{2}$. If z_k is better than one of last $q+1$ iterates $\{x^{k-q}, ..., x^{k-1}, x^k\}$ in terms of objective function, i.e., $F(z^k) \leq \max_{t=\max(0,k-q),...,k} F(x^t)$ then z^k will be used instead of x^k to compute y^k in the DCA scheme. This condition allows the objective function $F(x)$ to increase and consequently to escape from a potential bad local minimum. Theoretically, a large-value of q increases the chance of using the extrapolated points z^k in ADCA. Note that if $q = 0$ then $F(z^k) \leq F(x^k)$ and ADCA is a monotone algorithm like DCA. ADCA for solving the problem (1.9) is described in Algorithm 1.

Algorithm 1 ADCA for solving the problem (1.9)

Initialization: Choose an initial point x^0, $z^0 = x^0$, $q \in \mathbb{N}$, $t_0 = (1+\sqrt{5})/2$, $\rho \geq L$, and $k \leftarrow 0$.
repeat
If $F(z^k) \leq \max_{t=\max(0,k-q),...,k} F(x^t)$ then set $v^k = z^k$, otherwise set $v^k = x^k$.
Compute $y^k = \rho v^k - \nabla f(v^k) + \xi^k$, where $\xi^k \in \partial h_r(v^k)$.
Compute x^{k+1} by solving the convex problem

$$\min_{x \in \mathbb{R}^n} \left\{ \frac{\rho}{2}\|x\|^2 + g_r(x) - \langle y^k, x \rangle \right\}. \tag{1.11}$$

Compute $t_{k+1} = \frac{1+\sqrt{1+4t_k^2}}{2}$ and $z^{k+1} = x^{k+1} + \frac{t_k - 1}{t_{k+1}}(x^{k+1} - x^k)$ if $k \geq 1$.
$k \leftarrow k + 1$.
until Stopping criterion.

Note that the convergence results of ADCA are guaranteed by Theorem 2.1 and Theorem 2.2 in [35].

1.3.2 DCA-Like for solving the problem (1.9)

This section presents DCA-Like for solving (1.9) [41, 35]. DCA-Like aims to avoid objective function approximations with excessively large values of ρ. The main idea of DCA-Like is to maintain the parameter ρ as small as possible while seeking a convex approximation of objective function. By employing a small value of ρ_k, it is possible to obtain a more precise majorant of the objective function, which may result in a superior solution. DCA-Like relaxes the key requirement of standard DCA that the minorant $H_{\rho_k}(x, x^k) := H_\rho(x^k) + \langle \rho x^k - \nabla f(x^k) + \xi^k, x - x^k \rangle$ must be a lower bound of the second component $H_{\rho_k}(x)$ on the whole space. More precisely, at each iteration k, it is sufficient to just determine the value of ρ_k such that $H_{\rho_k}(x, x^k)$ is a lower bound of $H_{\rho_k}(x)$ at x^{k+1}, i.e.,

$$H_{\rho_k}(x^{k+1}) \geq H_{\rho_k}(x^{k+1}, x^k). \tag{1.12}$$

The DCA-Like algorithm for solving (1.9) is described in Algorithm 2.

Algorithm 2 DCA-Like for solving the problem (1.9)

Initialization: Choose an initial point x^0, a small enough positive parameter $\rho_0, \eta > 1, 0 < \delta < 1$, and $k \leftarrow 0$

repeat

Compute $\xi^k \in \partial h(x^k)$ and $\nabla f(x^k)$.

Set $\rho_k = \max\{\rho_0, \delta\rho_{k-1}\}$ if $k > 0$.

Compute x^{k+1} by solving the following convex program

$$\min_{x\in\mathbb{R}^n} \left\{ \frac{\rho_k}{2}\|x\|^2 + g(x) - \langle \rho_k x^k - \nabla f(x^k) + \xi^k, x\rangle \right\}. \tag{1.13}$$

while $H_{\rho_k}(x^{k+1}) < H_{\rho_k}(x^{k+1}, x^k)$ **do**

$\rho_k \leftarrow \eta\rho_k$.

Compute x^{k+1} by solving (1.13).

end while

$k = k + 1$.

until Stopping criterion.

The convergence properties and convergence rate of DCA-Like are guaranteed by Theorem 3.6 and Theorem 3.7 in [35].

1.3.3 Accelerated DCA-Like for solving the problem (1.9)

The accelerated version of DCA-Like to solve (1.9) is presented in this section [41, 35]. According to the DCA-Like algorithm, x^{k+1} is computed from x^k at each iteration by solving the convex subproblem (1.13). ADCA-Like (Accelerated DCA-Like) aims to find a point z^k that is superior to x^k for the computation of x^{k+1} in order to accelerate DCA-Like.

Algorithm 3 ADCA-Like for solving the problem (1.9)

Initialization: Choose an initial point $x^0, z^0 = x^0, q \in \mathbb{N}, t_0 = (1+\sqrt{5})/2$, a small enough positive parameter $\rho_0, \eta > 1, 0 < \delta < 1$, and $k \leftarrow 0$.

repeat

If $F(z^k) \leq \max_{t=\max(0,k-q),\ldots,k} F(x^t)$, then set $v^k = z^k$, otherwise set $v^k = x^k$.

Compute $\xi^k \in \partial h(v^k)$ and $\nabla f(v^k)$.

Set $\rho_k = \max\{\rho_0, \delta\rho_{k-1}\}$ if $k > 0$.

Compute x^{k+1} by solving the following convex program

$$\min_{x\in\mathbb{R}^n} \left\{ \frac{\rho_k}{2}\|x\|^2 + g(x) - \langle \rho_k v^k - \nabla f(v^k) + \xi^k, x\rangle \right\}. \tag{1.14}$$

while $H_{\rho_k}(x^{k+1}) < H_{\rho_k}(x^{k+1}, v^k)$ **do**

$\rho_k \leftarrow \eta\rho_k$.

Compute x^{k+1} by solving (1.14).

end while

Compute $t_{k+1} = \frac{1+\sqrt{1+4t_k^2}}{2}$ and $z^{k+1} = x^{k+1} + \frac{t_k - 1}{t_{k+1}}(x^{k+1} - x^k)$ if $k \geq 1$.

$k = k + 1$.

until Stopping criterion.

The convergence properties and convergence rates of ADCA-Like are guaranteed by Theorem 3.8 and Theorem 3.9 in [35].

1.4 Cryptography

Nowadays, information is an indispensable commodity, and accessing it through online resources has become a part of our daily lives. Nonetheless, information transmitted via telecommunications systems, including wireless, is always vulnerable to active and passive attacks [85]. Information security is therefore indispensable for enhancing data protection. This is accomplished using cryptographic methods. Multiple definitions exist for cryptography. According to [85], cryptography is the art and science of secret writing that modifies the original message structure into a distorted one. Similarly, in [56], cryptography is defined more intuitively as the application of computer science and mathematics to facilitate secure communication between two parties in the presence of an eavesdropper. Focused on the principles of confidentiality, data integrity, authentication, and non-repudiation, good cryptography prioritizes these aspects.

Optimization plays a crucial role in the field of cryptography, where the objective is to develop secure and efficient communication protocols and algorithms. By utilizing optimization techniques, cryptography is able to accomplish speedier and more secure encryption and decryption, efficient key management, secure authentication, and data integrity, as well as other essential functions for secure digital communication and blockchain technology.

1.4.1 Terms and definitions

Before proceeding further, we introduce some notations and definitions [1, 98, 56, 2, 3] used in this thesis.

- *Plaintext (cleartext)*: unencrypted information.
- *Ciphertext*: data which has been transformed to hide its information content.
- *Encryption (encipherment)*: (reversible) transformation of data by an encryption algorithm to produce ciphertext, i.e. to hide the information content of the data.
- *Encryption algorithm (encipherment algorithm)*: process which transforms plaintext into ciphertext.
- *Encryption system (encipherment system/cipher)*: cryptographic technique used to protect the confidentiality of data, and which consists of three component processes: an encryption algorithm, a decryption algorithm, and a method for generating keys.
- *Decryption (decipherment)*: reversal of a corresponding encryption.
- *Decryption algorithm (decipherment algorithm)*: process which transforms ciphertext into plaintext.
- *Key*: sequence of symbols that controls the operations of a cryptographic transformation.
- *Key management*: administration and use of generation, registration, certification, deregistration, distribution, installation, storage, archiving, revocation, derivation and destruction of keying material in accordance with a security policy.

- *Secret key*: key used with symmetric cryptographic techniques by a specified set of entities.
- *Individual key*: key shared between the key server and each member of the group.
- *Key encryption key*: cryptographic key that is used for the encryption or decryption of other keys.
- *Shared secret key*: key which is shared with all the active entities via a key establishment mechanism for multiple entities. It is also called group key.
- *Rekeying*: process of updating and redistributing the shared secret key (SSK), and optionally, key encryption key (KEK). This process is executed by the key server.
- *Individual rekeying*: rekeying method in which the shared secret key, and optionally, key encryption key are updated when an entity joins or leaves.
- *Batch rekeying*: rekeying method in which the key server processes both join and leave requests at every rekeying interval T.
- *Batch deletion*: rekeying method in which the key server processes each join request immediately to reduce the delay for a new user to access group communications, but processes leave requests at every rekeying interval T.
- *Batch insertion*: rekeying method in which the key server processes each leave request immediately to reduce the exposure to users who have departed, but processes join requests at every rekeying interval T.
- $e_K(M)$: result of encrypting data M with a symmetric encryption algorithm using the secret key K.
- $X\|Y$: result of concatenating data items X and Y in that order.
- *Hash-function*: a function which maps strings of bits to fixed-length strings of bits, satisfying the following two properties:
  — it is computationally infeasible to find for a given output, an input which maps to this output;
  — it is computationally infeasible to find for a given input, a second input which maps to the same output.
  Note 1: Computational feasibility depends on the specific security requirements and environment.
- *Hash-code*: the string of bits which is the output of a hash-function.
  Note 2: The literature on this subject contains a variety of terms that have the same or similar meaning as hash-code. Modification Detection Code, Manipulation Detection Code, digest, hash-result, hash-value and imprint are some examples.

1.4.2 Objectives of Cryptography

Effective cryptography is centered around the fundamental principles of maintaining confidentiality, ensuring data integrity, establishing authentication, and enabling non-repudiation.

Confidentiality

It is the set of operations developed to guarantee that only authorized individuals have access to data [13]. There are a number of ways to ensure confidentiality, including physical protection and mathematical algorithms that render data unintelligible [16, 56]. Confidentiality is a set of principles that restrict access to specific data [56].

Data Integrity

It can be defined as the set of operations that ensures the delivery of data without modification. Substitution, deletion, and insertion are typical data modifications [16]. In other words, data integrity guarantees the consistency and correctness of data throughout its lifetime [56].

Authentication

It is the set of operations that ensures the information received is only from the real sender. This function is applicable to both entities and information. Two parties communicating with one another must identify themselves. The origin, date of origin, data content, time of transmission, etc. of information transmitted over a channel should be verified. This aspect of cryptography is typically subdivided into two main classes for these reasons: entity authentication and data origin authentication [56]. Data origin authentication implicitly provides data integrity (because if a message is modified, the source has changed, and therefore the message is no longer valid).

Non-repudiation

It is the set of operations that ensures neither the sender nor the receiver can reject the exchange of information. This means that the sender of a message cannot reject it because he or she did not send it [13]. This ensures that it is impossible to compose a piece of information that would prevent data transmission.

1.4.3 Classification of cryptographic algorithms

Once a message has been encrypted, it can be decrypted using either the same key or a different one. These two cryptographic techniques are known, respectively, as symmetric-key cryptography and asymmetric-key cryptography.

Symmetric-key Cryptography

Symmetric-key cryptography is also known as secret-key cryptography since it has only one key for both encryption and decryption and has to be kept secret. Both the sender and receiver need to agree on a single shared secret key for this encryption method. Encrypting a message generates data of approximately the same length as the original text. For deciphering the message, the inverse of the encryption function and the same encryption key are used. The symmetric encryption algorithms AES, DES, and 3DES are well-known [55].

Asymmetric-key Cryptography

Asymmetric-key (or public-key) cryptography is a technological discipline concerned with the secure transmission and reception of information through public networks. Public key cryptography relies on the utilization of mathematical problems and functions that are computationally challenging to solve in a single way, serving as the foundation for encryption. Public-key cryptosystems are characterized by the utilization of distinct keys for encryption and decryption processes. These keys are known as the public key and the private key, respectively. One of the essential considerations in ensuring the security of asymmetric encryption is in the fundamental fact that the derivation of the private key from the public key is infeasible. Consequently, the likelihood of divulging the confidential data is negligible. Prominent examples of asymmetric cryptographic algorithms are RSA, ElGamal, and ECC.

The cryptographic hash function, often referred to informally as a one-way hash function, is one of the fundamental primitives of modern cryptography. While the primary purpose of encryption is to transform data into ciphertext using keys, hash algorithms contribute to the overall security of data. The hash algorithm provides a method for validating the integrity of data. By generating a hash value of the original data prior to encryption and comparing it to a hash value derived from the decrypted data, it is possible to ensure that the data has not been modified during transmission or storage. This procedure provides an additional layer of protection against unauthorized modifications, complementing encryption.

Hash functions

In the 1970s, hash functions were introduced as a powerful tool to address numerous security issues in telecommunications and computer networks [90]. It is a function that converts variable-length messages to fixed-length data, typically between 128 and 512 bits [86]. The majority of hash functions are applied to hash tables. A hash table is a data structure that makes it simple to organize data so that it can be quickly located later. Each position in a hash table is referred to as a slot, and it contains a piece of data. Each slot is allocated a unique integer index. Each slot is referenced by a key, and the data item stored in it is referred to as a value [90].

Three potential properties are listed for a hash function H with inputs x, x' and outputs y, y' [56]. The first one is preimage resistance. For essentially all pre-specified outputs, it is computationally infeasible to find any input that hashes to that output, i.e., to find any preimage x' such that $H(x') = y$ when given any y for which a corresponding input is not known. The hash function has to ensure the second-preimage resistance. It is computationally infeasible to find any second input that has the same output as any specified input, i.e., given x, to find a second-preimage $x' \neq x$ such that $H(x) = H(x')$. Collision resistance is the third property of the hash function. It is computationally infeasible to find any two distinct inputs x, x' that hash to the same output, i.e., such that $H(x) = H(x')$.

Designing a cryptographic hash function was considered straightforward for a number of years. In this way, MD5 and SHA-1, two common hash functions, were widely used in the aim of achieving security. However, MD5 was revealed to have vulnerabilities, but SHA-1, which is founded on the same principles as MD5, was believed to be secure and was therefore widely used. Nonetheless, with the advent of the technological revolution, attacks on MD5 and SHA-1 emerged. The attacks demonstrate that compromising these systems is simpler than anticipated. Therefore, MD5 and SHA-1 are no longer

recommended for use. Alternative options exist, but they are based on the same fundamental principles as MD5 and SHA-1. Changing from one hash function to another also has significant software and hardware implications. Therefore, evidence of safety is required to transfer effectively. In addition, in an effort to increase data security, NIST launched an open competition to define a new set of hash functions known as SHA-3, which is intended to address deficiencies in SHA-2 [86].

1.4.4 Preliminaries of Centralized group key management

Logical Key Hierarchy (LKH) [94, 96] is a common and fundamental technique for centralized group key management. The key tree is a hierarchical structure utilized by the LKH to manage the group key. A full binary tree in which each node has zero or two children is among the most efficient structures. The LKH, as illustrated in Fig. 1.1, shows a single trusted key server to maintain the tree of keys and update the distribution of keys. A shared secret key is assigned to the root node of the tree. The leaf nodes of the key tree correspond to the group members and each leaf node assigns an individual key. Additionally, key encryption keys are assigned to the internal nodes (middle level nodes). The key encryption keys are shared by multiple members whose individual keys are assigned to the descendant of the node to which the key encryption key is assigned. Each member in the group has to maintain all the keys assigned to the nodes on the path from the root node to the leaf node, to which the individual key of the member is assigned. Therefore, the number of keys an entity has is proportional to the logarithm of the total number of entities. When a member joins or leaves, all the keys on the member's key path have to be changed to maintain forward and backward secrecy.

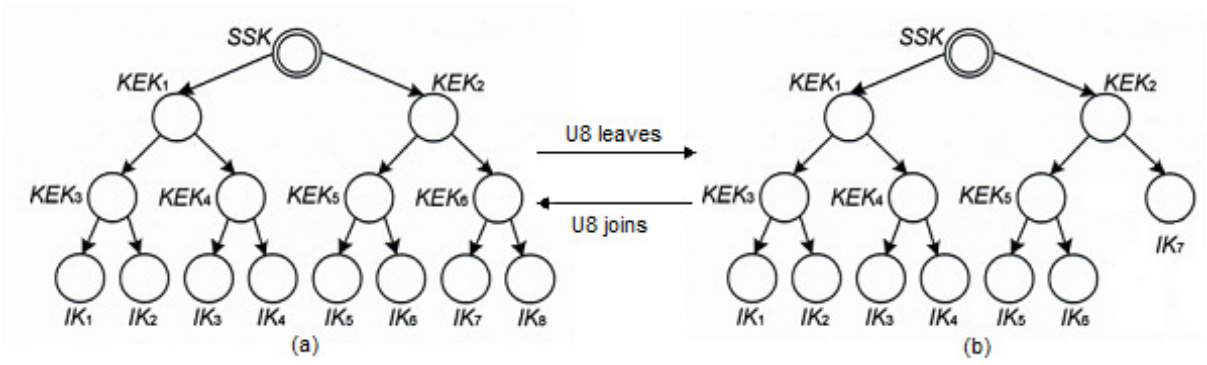

Figure 1.1: A full binary tree structure.

Figure 1.1 shows an example of an LKH, which can be taken as a full binary tree architecture. Each node in a key tree represents a symmetric key. The key server chooses and distributes the keys for the eight members $U_1, U_2, ..., U_8$. The root key SSK represents the group key shared by all group members. The leaf node keys $IK_1, IK_2, ..., IK_8$ are assigned to the group members $U_1, U_2, ..., U_8$, separately, and IK_i is shared by both the key server and individual member U_i. The internal node keys $KEK_1, KEK_2, ..., KEK_6$ are known only by the members that are in the subtree rooted at this node. Therefore, each member holds the keys from its leaf node key to the root node key. For instance, U_1 holds four keys $\{SSK, KEK_1, KEK_3, IK_1\}$ along the path from U_1 to root.

A set of operations and techniques for efficiently updating the tree when members join and leave the group was defined in [61].

Deletion of an existing member

When a member associated with a leaf node, C, leaves the group, if node S (node C's sibling) is also a leaf node, leaf node S is transferred to node P (which is the parent node of C and S). If node S is the root of another subtree, S will be moved to node P to create a new subtree with node S as its root (closer to the root). Figure 1.2 illustrates this operation. The key server has to generate new keys for all nodes in the path from this deleted node to the root.

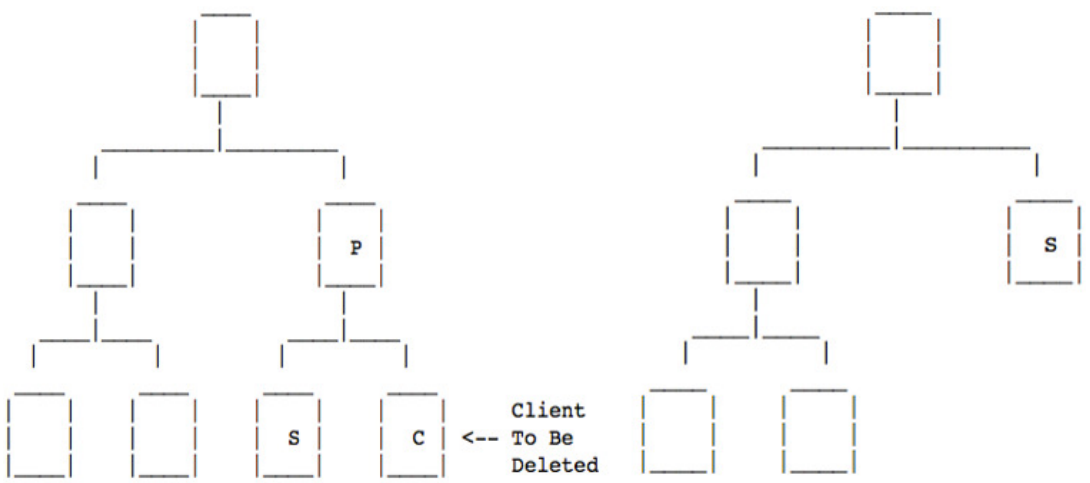

Figure 1.2: The key tree after the member has been deleted [61].

Suppose that member U_8 leaves the group as shown in Fig. 1.1 from (a) to (b), then the form of the updated keys (UpdateKey$_{1-7}$), broadcast by the key server to $U_1, U_2, U_3, U_4, U_5, U_6$ and U_7 is:

$$\text{UpdatedKey}_{1-7} = e_{KEK1}(SSK') \| e_{KEK2'}(SSK') \| e_{KEK5}(KEK2') \| e_{IK7}(KEK2').$$

Addition of a new member

When a new member is added to the group, the key server will choose a leaf node, LS, which is shallowest to the root. Node LS will be modified to be a new interior node NI, make the existing member of node LS be a left child of NI. The new member will become the right child of NI. By inserting the new member at the shallowest place in the tree, we help to keep it balanced and minimize its height. This minimizes the computational cost and bandwidth required during key updates. Figure 1.3 illustrates this operation. The key server has to generate new keys for all nodes in the path from this new node to the root.

Suppose that a member U_8 joins the group as shown in Fig. 1.1 (from (b) to (a)). After gaining the authorization by the key server, the new member U_8 will be assigned a joining node and get a secret key IK_8. Three updated keys (UpdatedKey$_{1-7}$), broadcast by the key server to U_1, U_2, U_3, U_4, U_5, U_6 and U_7 are:

$$\text{UpdatedKey}_{1-7} = e_{SSK}(SSK') \| e_{KEK2}(KEK2') \| e_{IK7}(KEK6').$$

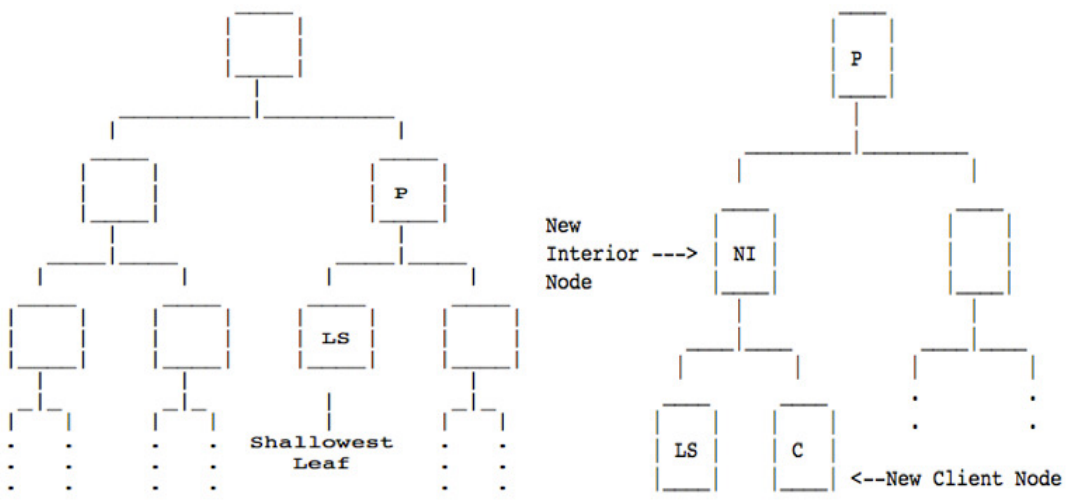

Figure 1.3: Adding a new member at the shallowest position [61].

Three updated keys (UpdatedKey$_8$), sent by the key server to U_8 are:

$$\text{UpdatedKey}_8 = e_{IK8}(SSK' \| KEK2' \| KEK6').$$

In order to further reduce the rekeying cost, there are other approaches proposed to improve the efficiency of key tree-based approaches. The optimization of the hierarchical binary tree called One-Way Function Tree (OFT) was proposed in [84, 89]. Their scheme reduces the rekeying cost from $2\log_2(N)$ to only $\log_2(N)$.

However, there is rarely a balanced tree once members join or leave a group. The height from the root to any leaf node in a balanced binary key tree with N leaf nodes is $\log_2 N$. If the tree is unbalanced, it is possible that some members will have to perform $N-1$ decryptions in order to obtain the group key. Additionally, in an unbalanced key tree, some members may be required to store N keys, while the remaining members may only require a few keys. Hence, maintaining a balanced key tree is a critical problem to solve.

1.4.5 Merkle tree in Ethereum system

A Merkle tree is illustrated in Fig. 1.4. It is computed for eight data items $x_0 - x_7$ (shown as leaf nodes). Internal nodes are associated with hash values computed hierarchically including the Merkle root R. The amount of data needed to update an item, which corresponds to a leaf in the tree, is the number of hash values for the sibling nodes in a path from this assigned leaf node to the root. This can show that the known Merkle root R is computed from a set of data elements with x among them without disclosing the Merkle tree's whole values. To demonstrate the presence of x_2, it is sufficient to give simply three values (shown in red in the figure): (i) h_3 (ii) $h_{0-1} = H(h_0 \| h_1)$ (iii) $h_{4-7} = H(H(h_4 \| h_5) \| H(h_6 \| h_7))$, where for $i \in [0, 7]$, h_i denotes the hash H values of x_i, $X \| Y$ denotes result of concatenating data items X and Y in that order. With these three values and x_2, the root value R can be computed as $R = H(H(h_{0-1} \| H(H(x_2) \| h_3)) \| h_{4-7})$.

In Ethereum system, the State Merkle tree is a key-value map in which the account addresses are the keys and values include the balance and potentially code or storage assigned for the account. Addition-

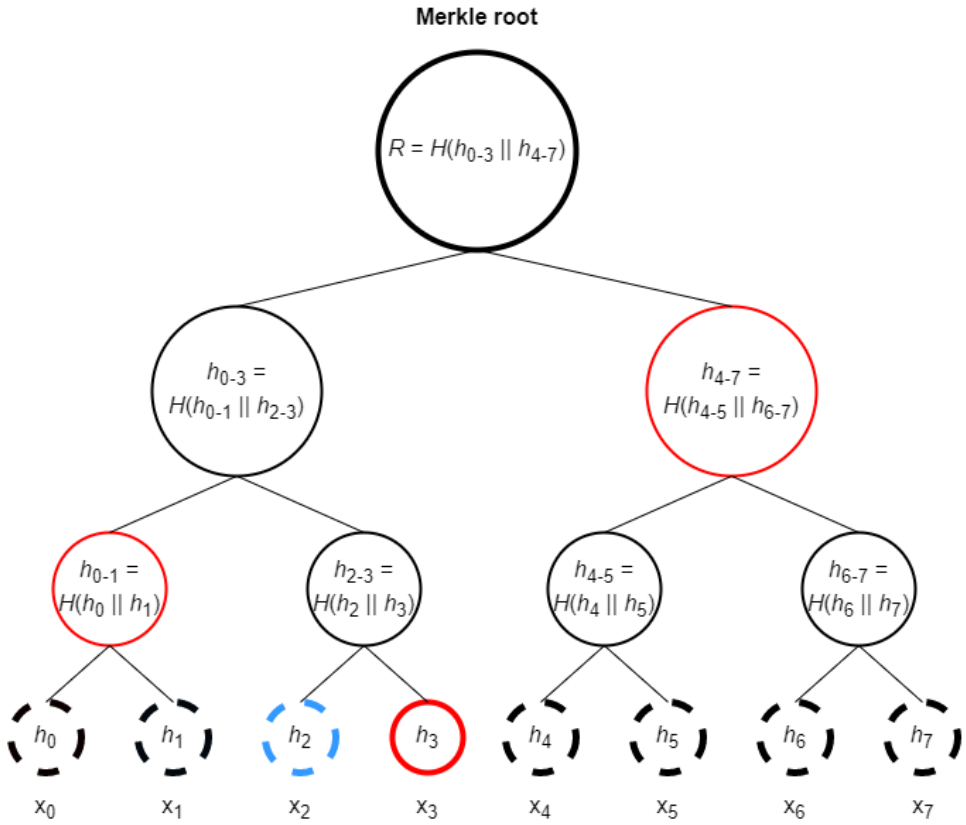

Figure 1.4: Illustration of a Merkle tree of 8 items. Merkle proof of item x_2 (with a dashed blue leaf) includes nodes h_3, h_{0-1}, and h_{4-7} (in solid red) [59].

ally, when the state of a transaction is modified as a consequence of its execution, all nodes on the paths from the modified accounts to the root are adjusted. When frequently used accounts move closer in the state tree due to their overlapping paths to the root, fewer tree portions must be read and updated. For example, account x_1 transfers 0.01 Ethereum (ETH) to account x_2. The data of two accounts, especially the balance assigned to two leaf nodes on the Merkle tree has to be updated. The siblings paths for x_1 (with h_0, h_{2-3}, and h_{4-7}) and x_2 (with h_3, h_{0-1}, and h_{4-7}) share the node h_{4-7} as illustrated in Fig. 1.4. As a result, just $2 \times 3 - 1 = 5$ values on the tree are required to update the Merkle tree. The position of the nodes in the tree has an impact on possible savings. Because the nodes along their paths do not overlap, there are no savings when the data of x_2 and x_5 are modified.

Chapter 2

DC Programming and DCA for Dynamic Centralized Group Key Management problem

In this chapter, we address an important problem of updating group key in centralized group key management (CGKM) using two different techniques, namely batch insertion and batch rekeying. Using batch insertion technique, it consists in finding a set of leaf nodes in a binary key tree to insert new members with minimal insertion cost and keeping the tree as balanced as possible. The two mentioned important objectives are combined into a unified (deterministic) optimization model whose objective function contains discontinuous step functions with binary variables, which is known to be NP-hard. We reformulate the problem as a standard DC (Difference of Convex functions) program that can be solved efficiently by DCA (DC Algorithm). Besides, the insertion nodes must be the leaf nodes, we introduce a two-step algorithm to reduce the model complexity: the first is to find the set of leaf nodes, while the second is to solve the simplified optimization problem. With the batch rekeying technique, we propose a two-step algorithm to minimize the total rekeying cost, which is the cost of deletion and insertion members, while keeping the tree as balanced as possible. The mentioned problem is nonsmooth, nonconvex, and very hard to solve. Numerical experiments have been studied intensively to justify the merit of our proposed models as well as the corresponding DCAs.

The results presented in this chapter were published/submitted in:

H. A. Le Thi, T. T. T. Nguyen, & H. P. H. Luu (2022). A DC programming approach for solving a centralized group key management problem. *Journal of Combinatorial Optimization*, 1-29.

T. T. T. Nguyen, H. P. H. Luu, & H. A. Le Thi (2022). Solving a Centralized Dynamic Group Key Management Problem by an Optimization Approach. In: *Le Thi, H.A., Pham Dinh, T., Le, H.M. (Eds.) Modelling, Computation and Optimization in Information Systems and Management Sciences. MCO 2021. Lecture Notes in Networks and Systems*, vol 363. Springer, Cham.

T. T. T. Nguyen, & H. A. Le Thi (2022). Solving the problem of batch deletion and insertion members in the Logical Key Hierarchy structure by a DC Programming approach. *32nd European Conference on Operational Research (EURO 2022)*, Espoo, Finland.

H. A. Le Thi, & T. T. T. Nguyen (2023). Solving the problem of batch deletion and insertion members in the Logical Key Hierarchy structure by a DC Programming approach. *Submitted & Available on arXiv* `arXiv:2305.10131`.

2.1 Introduction

Secure group communication systems are required for many critical network applications, including teleconferences, information services, and collaborative work. Access to the group can be enforced through encryption techniques. A group secret key is used to encrypt the data, and the corresponding ciphertext is sent to all members. As a result, securing group communications (i.e., ensuring the confidentiality, authenticity, and integrity of messages sent between group members) has emerged as a critical issue in Internet design. Apart from social networks, more secure environments, such as military networks, require an even higher level of confidentiality and security for data transmission, membership management, and key management.

Due to the dynamic nature of group membership, it becomes necessary to change the group key in an efficient and secure manner when members join or leave the group. Typically, the key server (KS) is responsible for updating the group key when the group membership changes. Due to its centralization, this type of scheme is referred to as centralized group key management (CGKM) [77, 83, 8]. Group key management mechanisms can be categorized based on their logical architecture, which includes star-based structures, d-ary tree-based structures, and general tree-based structures [2]. Among the various structures for this type of management, the widely used hierarchical key-tree approach proposed in [94] and independently investigated in [96] is an efficient method for reducing the cost of rekeying [82, 69]. The rekeying cost refers to the number of messages required for distribution among members in order to receive the new group key. In the binary tree, only leaf nodes can be removed and new nodes will be only appended below given leaf nodes. The number of messages is the one of updated keys in the path from the root to the selected leaf node. Consequently, if the binary tree architecture is balanced, the key server needs to perform $2\log_2(N)$ encryptions for a joining and $2(\log_2(N)-1)$ for a leaving, where N is the number of group members and $\log_2(N)$ represents the height of the binary key tree. By definition, a key tree is classified as balanced if the distance from the root to any two leaf nodes does not exceed one.

In the context of group key management systems, it is important to provide two different types of secrecy, namely backward secrecy and forward secrecy. If the system changes the group key after a new user joins, the new member will be unable to decrypt past group communications; this is known as backward secrecy. Similarly, if the system rekeys after a current user departs, or is ejected from the system, the departed user will be unable to access future group communication; this is known as forward secrecy. To accommodate different application requirements and make trade-offs between performance and group security, the key server can operate in three batch modes [98]: 1) batch rekeying, in which the key server processes both join and leave requests periodically in a batch; 2) batch deletion, in which the key server processes each join request immediately to reduce the delay for a new user to access group communications, but processes leave requests in a batch; and 3) batch insertion, in which the key server processes each leave request immediately to reduce the exposure to users who have departed, but processes join requests in a batch. Hence, processing batch deletion provides backward secrecy for group communications, whereas batch insertion assures forward secrecy of the system. The batch rekeying techniques maintain the trade-off between security and performance, decrease the necessary number of messages, enhance operational efficiency, and prevent the generation of duplicate keys. Nevertheless, despite the advancements made in these approaches, there are still obstacles to overcome in order to achieve optimal balance of the key tree, minimize rekeying costs, and maintain the overall security of

the group.

Numerous works in the literature investigate the optimization aspect of centralized group key management using symmetric keys (keys that are shared secretly) [60, 22, 31]. In general, optimization objectives include reducing the cost of rekeying, optimizing parameter selection in group communication systems, optimizing the structure of the key tree, and optimizing the time interval between rekeying, etc. However, the majority of these objectives have not been rigorously described as a mathematical optimization model. Additionally, their algorithms are primarily based on logical/heuristic arguments.

In this chapter, we consider two rekeying techniques: batch insertion and batch rekeying. First, we opt for the third approach which is batch insertion and individual deletion. The advantage of processing each leave request immediately is to maintain the forward secrecy for group communications. In practice, the efficient implementation of backward secrecy is feasible by using the distribution of a new group key via encryption with the old group key, therefore ensuring its availability to the current group users. Since the forward secrecy is harder to implement, this is the remarkable advantage of such an approach in comparison to the former two. However, this approach also has a drawback as the number of updated keys generated by individual deletion can be much more than the sum of those generated by batch rekeying. Therefore, we propose a novel approach to an important problem in centralized dynamic group key management that consists in finding a set of leaf nodes in a binary key tree to insert new members while minimizing the insertion cost. Moreover, as stated in [61], maintaining balanced trees is desirable in practice because membership updates can be performed with logarithmic rekeying costs provided that the tree is balanced. Hence, the tree's balance is a critical property of the Logical Key Hierarchy (LKH) structure. However, because the key server has no control over the departing members' positions, the tree may become unbalanced as a result. It will remain unbalanced until insertions/deletions restore the tree to a balanced state or until some action is taken to rebalance the tree. To resolve this issue, we must exert control over the shape of the LKH's key tree. This objective should be accomplished through the insertion process, as rebalancing the entire tree is an extremely costly procedure. To our knowledge, the majority of approaches in the literature are heuristic in nature, and the two mentioned important objectives - the rekeying cost and the balance of the tree, have not been adequately addressed in a unified optimization framework.

Furthermore, processing each leave request immediately ensure the forward secrecy for group communications. However, the number of updated keys generated by individual leave rekeying can be much more than the sum of those generated by batch rekeying. In this case, key server and group members are assigned to perform more rekeying operations rather than making use of the service. Batch rekeying [50, 65, 70, 100, 99, 93, 69] can alleviate the out-of-sync problem and improve efficiency. That is, when a member joins or leaves the group, the key server does not instantly conduct rekeying; instead, it aggregates the total number of joining and leaving members over a specified time interval and then changes the associated keys. Therefore, batch rekeying techniques are used to enhance efficiency by reducing the amount of messages necessary and benefiting on the potential overlap of new keys for several rekeying requests, and then reduce the possibility of generating redundant new keys. In [50], the authors demonstrated how to use a batching technique on the key server to process join and leave requests to decrease the rekeying cost. Two merging methods that are appropriate for batch joining events to maintain the balance of the tree following insertion were developed in [65]. In [93], the authors introduced rotation-based key tree methods for balancing the tree even when batch leave requests exceed batch join requests.

Existing methods do not take key tree balancing and rekeying costs into account simultaneously, which results in an unbalanced key tree or excessive rekeying costs.

Due to the advantages of batch rekeying, we propose an optimization approach to the problem of updating the group key for both join and leave requests at every rekeying interval. In actuality, the deletion nodes are also the leaf nodes in the tree, and a subtree of new nodes can be appended below a leaf node or replaced by the position of the leaving node on the binary key tree. The latter has a lower updating key cost than the former since, when a member leaves, all the keys on the path from the root to the deletion node must be updated anyway.

There is another recent research direction that is distinct from symmetric key cryptography, and it is focused on the modification of encryption algorithms. Asymmetric key cryptography (also known as public key cryptography) [20, 34, 27] is used in conjunction with the LKH in this line of work, which reduces the cost of updating the group key while increasing the computational complexity. The use of known complexity-theoretic hard problems, as opposed to numeric-theoretic ones, is the main idea of public key cryptography. In a public-key cryptosystem, the public keys can be freely and openly transmitted. Only the corresponding private keys need to be kept secret by their owners. An asymmetric key algorithm, such as RSA, Diffie-Hellman, Elliptic Curve Cryptosystem, etc., can be integrated into the LKH structure. In the key tree, the public key of each member is stored in leaf nodes. All keys from the root to the leaf node will be encrypted using the public key that is assigned to this node, and the only member who has the corresponding private key can decrypt the message to obtain a new group key. Due to the significant computational difficulty of performing modular exponentiation in the multiplicative group, point multiplication, and addition in the elliptic curve with large enough parameters, these schemes run much more slowly than symmetric cryptosystems.

Our contributions. We develop optimization approaches to the problem of updating group key in the LKH structure using two different techniques, namely batch insertion and batch rekeying. Our main contributions are two-fold [67, 43, 42].

Firstly, we opt for the batch insertion and individual deletion techniques due to its advantages over the individual rekeying as discussed above. In the theoretical perspective, the mentioned problem can be formulated as a deterministic optimization model. To our knowledge, this is the first work introducing an optimization model that takes into account simultaneously both objectives: the insertion cost and the balance of the tree. The proposed optimization model is a combinatorial problem with discontinuous step functions and binary variables. It is very challenging to handle such kinds of programs by standard methods where the source of difficulty comes from the discontinuity of the objective and the binary nature of the solutions. The problem is first equivalently formulated to remove the discontinuity of the objective, and then the latter problem is reformulated as a DC program via an exact penalty technique, where the DCA is at our disposal as an efficient algorithm in DC programming [74, 48, 38].

Furthermore, by observing that inserting nodes must be leaf nodes, we proposed an alternative approach to solve the mentioned problem including two steps, which is expected to be more efficient. Therein, the first step of our algorithm consists in finding the set of all leaf nodes of the given tree. Thanks to the first step, the optimization problem can be simplified: the leaf nodes constraints are removed and the dimension of the optimization variables is reduced significantly. Therefore, in the second step, we only need to solve a simplified model (of the original model) which can still be solved by the aforementioned DCA-based approach. Clearly, the first step reduces considerably the complexity of the

optimization model given in the second step, which works only on the leaf nodes.

Secondly, we opt for the batch rekeying to update the group key in CGKM. As the deleting and inserting nodes should be leaf nodes, the first step of our algorithm consists in finding the set of all leaf nodes of the given tree. In the second stage, based on the found leaf nodes, we find a set of leaf nodes to delete leaving members and insert new members while minimizing the total rekeying cost, which is the cost of deletion and insertion members, and taking the balance of the tree into account. The proposed optimization model is also a combinatorial problem with discontinuous step functions and binary variables. The discontinuity of the objectives and the binary nature of the solutions make it extremely difficult to manage these types of programs using conventional techniques. To overcome these challenges, we first equivalently formulate the problem to eliminate the objective's discontinuity, and then we reformulate the latter problem using an exact penalty technique as a DC program, for which the DCA is an effective algorithm.

Overall, our proposed approaches provide a good compromise compared to existing works, producing a more balanced key tree with a low key updating cost.

The remaining sections of the chapter are structured as follows. The survey of many relevant centralized key management schemes for multicast communication is presented in Section 2.2. The optimization models and the solution method based on DC programming and DCA for the problem of updating group key in the LKH structure with batch insertion technique are presented in Section 2.3 and 2.4, respectively. Section 2.5 gives a two-step algorithm for the problem of updating group key in the LKH structure with batch rekeying. Section 2.6 presents the solution method based on DC programming and DCA. The implementation of the algorithms for solving the problem in LKH structure and numerical experiments are presented in Section 2.7. Finally, some conclusions are provided in Section 2.8.

2.2 Related works

In the centralized group key management system, a single organization is in charge of key generation, distribution, rekeying, and group communication. The greatest challenges of centralized methods are scalability overhead, rekeying overhead, communication overhead, computing overhead, storage overhead, maintaining forward and backward secrecy, and independence from collusion. The difficulty of centralized systems to manage multiple membership changes is an additional issue. To address these challenges, it is necessary to investigate the various key management optimization-based approaches described in the literature. Most existing methods rely on the modification of cryptographic algorithms or logical/heuristic arguments to reduce the rekeying cost or maintain the tree's balance following insertion and deletion operations.

In the literature, there are several works that study the optimization aspect of CGKM with symmetric keys (keys that are shared secretly) [50, 25, 60, 84, 52, 65, 22, 93, 31]. As a solution to the scalability problem of group key management, the LKH key tree architecture was developed in [94, 96]. They implemented key graphs to identify secure groups. Additionally, they presented three strategies for the secure distribution of rekeying messages whenever the group membership changes. Nevertheless, the scheme's computational cost increases. In [61], the authors addressed the problem of how to maintain balanced trees in the key management scheme of [94]. Every time a member is to be added in the tree, the key server always finds the shallowest leaf of the tree in which a new member to be inserted.

Furthermore, they proposed two simple tree rebalancing schemes for deletion. One is a modification of the deletion algorithm that is to move a member located at the deepest level to the place occupied by the deleted member. The other method allows the key tree to be unbalanced for a while after a sequence of keys is updated, and then reconstructs a key tree to be in a balanced state. However, this method does not perform well when the leaving members are always on one side (right or left side). In this situation, the key tree becomes a skewed binary tree.

For the establishment of keys in large, dynamic groups, the One-Way Function Tree (OFT) method was suggested in [84]. The computation cost of members is logarithmically dependent on the size of the group. The storage cost of group members is also proportional to the group's size on a logarithmic scale. However, the proposed scheme is vulnerable to collusion attacks, wherein departing and incoming members may collude their keying information to discover the previous and more recent group keys. The authors of [25] proposed three algorithms based on a 2–3 tree (where each internal node has a degree of 2 or 3) to reduce the worst-case communication cost required for updating the group key. They discovered that the height- and weight-balanced 2–3 tree algorithms achieved a reasonable trade-off between restructuring costs and tree structure. In order to balance the B-tree, a node split occurs after a new member is added. In [52], Lu et al. proposed a new Non Split Balancing High Order (NSBHO) tree in which the node splitting is not required for balancing the tree. To perform a leave operation, it requires the same worst case rekeying cost as 2–3 tree does. In addition, NSBHO has superior and far superior rekeying performance in average and worst case in comparison with 2–3 tree.

In [50], the authors demonstrated how to use a batching technique on the key server to process join and leave requests to decrease the rekeying cost. In [65], two merging methods that are appropriate for batch joining events to maintain the balance of the tree following insertion were developed. In [93], a multicast key management scheme was proposed for batch rekeying based on rotation one or twice. The balance factor of a node is the difference between the heights of its right and left subtrees. They reduced the overhead of batch rekeying operations when the batch leave operations are higher than batch join operations. The proposed rotation based algorithms has the ability to control the users join positions. However, they fail to control leaving positions of the users. The performance of the scheme is degraded, if the batch join operations are higher than the batch leave operations.

The key establishment mechanisms for multiple entities in order to provide procedures for managing cryptographic keying material used in symmetric or asymmetric cryptographic algorithms in accordance with the current security policy were specified in [2]. It also defines symmetric key establishment mechanisms based on a general tree structure (based on the LKH structure) with both individual and batch rekeying. However, this standard does not specify how to construct a tree structure or how to insert new members at specific positions in the tree.

Exclusion Basis Systems (EBS), a combinatorial formulation of the group key management problem that produces optimal results with respect to the size of the group, the number of keys stored by each member, and the number of rekeying messages was presented in [60]. They developed a general technique for determining the optimal values of the keys and messages as a function of the group size and described the tradeoff between the two mentioned parameters. Their formulation contains stars, d-ary trees and all such structures as special cases. Additionally, they illustrate the scalability of EBSs and provide an explanation of the algorithms for admitting and evicting group members. In [22], the authors suggested ways to optimize the key management structure in a hybrid group key management scheme.

Both the overall communication cost and the computational cost imposed on client devices are reduced by their method. A hybrid group key management scheme was suggested as a potential resolution to the challenges that are inherent in the star-based and tree-based schemes. The authors of [31] proposed a new batch rekeying scheme which dynamically configure the batch rekeying intervals. To analyze communication cost of a key tree structure, they utilize level-homogeneous key tree structure, whose node in the same level has the same degree.

Another way to solve the problem in group key management is to integrate the tree structure with the public key encryption algorithm. A Cluster-based Elliptic Curve Key Management (CECKM) protocol for secure group communications in wireless sensor networks (WSNs) was proposed in [51]. The CECKM protocol provides the same level of security as RSA and Diffie-Hellman with reduced keys and is effective for sensor node key exchange protocols. It requires significantly less time for system key resynchronization than the Diffie-Hellman and Group Diffie-Hellman protocols. In [30], the authors proposed a pairing-free Identity-based Two-Party Authenticated Key Agreement (ID-2PAKA) protocol using elliptic curve cryptography. The proposed protocol provides an efficient method for two parties to generate a shared session key over an open network and is suitable for efficient and secure peer-to-peer communications. In [34], Kumar et al. proposed a more efficient centralized group key distribution based upon RSA public key cryptosystem. In order to increase the level of the security, the proposed protocol uses an additional key called group key encryption key, which is generated using Chinese Remainder Theorem. In addition, the extended clustered tree structure based key management is more efficient for batch rekeying in terms of computational cost of KS. Finally, for updating the group key, whenever the group membership change, the amount of keying information that needs to be communicated is also reduced.

2.3 Optimization models for the group key update problem using batch insertion and individual deletion techniques

2.3.1 Problem description

We consider the problem of updating group key in the LKH structure. When the membership of a group changes dynamically, the group key and all related keys on the binary tree must be updated. As discussed earlier, the key server can control the new node to be placed at any chosen position, but it cannot control the positions where deletions occur. Moreover, we opt for batch insertion and individual deletion techniques to maintain forward secrecy for group communications. Therefore, our main goal here is to find a set of leaf nodes in a binary key tree to insert new members while minimizing the insertion cost (the number of updated keys) and taking the balance of the tree into account.

2.3.2 The first optimization model

Based on the above problem description, in this subsection, we construct an optimization model that is defined on the entire set of tree nodes.

Given a full binary tree that can be represented as an ordered set $T = \{2^a + b$: there exists a node at the b-th horizontal position of level a on the tree, $0 \leq a \leq h, 0 \leq b \leq 2^a - 1, a, b \in \mathbb{N}\}$, where h is the height of the tree, we sort T in ascending order for convenience. The distance from the root to the tree

node $T[i]$ is given by $d_i = \lfloor \log_2 T[i] \rfloor$. Let the set of joining members be $M = \{1, 2, ..., m\}$.

Let A_{ik} be binary variables defined by $A_{ik} = 1$ if there exists an edge that connects vertex $T[i]$ to vertex $T[k]$ in the key tree T, where $i < k, T[i], T[k] \in T$, $A_{ik} = 0$ otherwise, $\forall i, k = \overline{1, n}, n = |T|$.

To determine whether a node in the tree T is a leaf node or not, we define the following variable: $B_i = 1$ if $\sum_{k=1}^{n} A_{ik} = 0$, $B_i = 0$ if $\sum_{k=1}^{n} A_{ik} = 2$. It means that if $B_i = 1$, the corresponding node $T[i]$ is a leaf node and if $B_i = 0$, it is an interior node that cannot be inserted new member.

Remark 2. *i) After inserting m members into the tree, each leaf node will be either assigned with a certain number of new members or not changed (illustrated in Fig. 2.1). If a leaf node is assigned some new members, it becomes a key interior node, otherwise, it remains a leaf node.*

ii) A valid insertion procedure will create a full binary subtree below each assigned leaf node. According to Handshaking lemma [92] and the properties of a full binary tree, we have the following assertion: The cost of appending new nodes at the position of a leaf node on the full binary tree only depends on the number of new nodes to be inserted at that leaf node but it does not depend on the configuration of the full binary subtree.

Now we propose the optimization model for minimizing the key updating cost. Let x_{ij} be binary variables defined by $x_{ij} = 1$ if a new member $j \in M$ is inserted into the subtree below the node $T[i]$, $x_{ij} = 0$ otherwise. Since every new member is appended below only one leaf node of the original tree, $\sum_{i=1}^{n} x_{ij} = 1, \forall j = \overline{1, m}$. For a given value $i \in \{1, 2, ..., n\}$, the number of new nodes j being inserted at the leaf node $T[i]$ with $B_i = 1$ is calculated as $m_i = \sum_{j=1}^{m} x_{ij}$. Obviously, $\sum_{i=1}^{n} B_i \times m_i = m$. It means that a subtree at the leaf node $T[i]$ includes $m_i + 1$ leaf nodes (including m_i new nodes and the old leaf node $T[i]$) as illustrated in Fig. 2.1. For the leaf node $T[i]$ that is chosen to append some new members (equivalent to $\sum_{j=1}^{m} x_{ij} > 0$), the cost to build a corresponding subtree is $2\sum_{j=1}^{m} x_{ij} - 1$. Besides, the cost to update keys from the root to $T[i]$ is d_i. More precisely, we have

$$\text{cost at } T[i] = \begin{cases} d_i + 2\sum_{j=1}^{m} x_{ij} - 1, & \text{if } \sum_{j=1}^{m} x_{ij} > 0 \\ 0, & \text{if } \sum_{j=1}^{m} x_{ij} = 0, \end{cases}$$

$$= d_i \times 1_{\sum_{j=1}^{m} x_{ij} > 0} + 2\sum_{j=1}^{m} x_{ij} \times 1_{\sum_{j=1}^{m} x_{ij} > 0} - 1_{\sum_{j=1}^{m} x_{ij} > 0}.$$

In batch insertion, with two leaf nodes being inserted new members, there is an advanced technique that we can reuse the updated key encryption keys on the overlap part of one path (the path from the root to a leaf node) to another path. It also leads to a complicated recursive relation based on the structure of the binary tree to compute the total cost. We therefore alleviate this by ignoring the overlap keys,

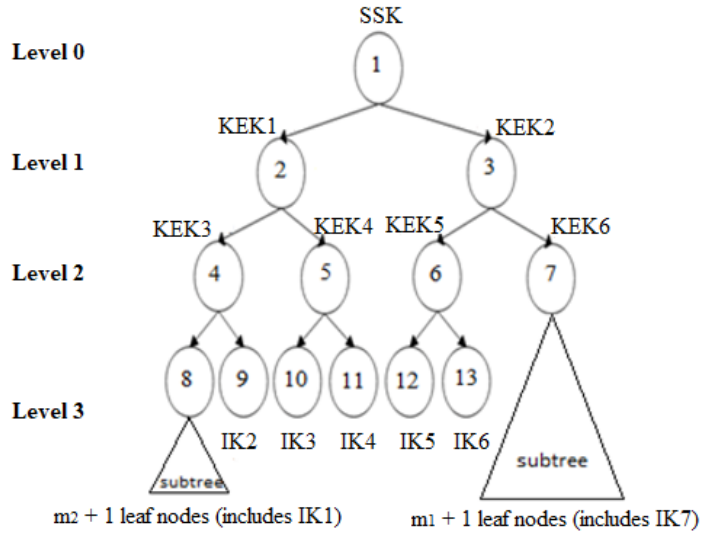

Figure 2.1: The tree structure after inserting new nodes.

resulting in the approximation cost function as follows:

$$\begin{aligned} F(x) &= \sum_{i=1}^{n} d_i \times 1_{\sum_{j=1}^{m} x_{ij}>0} + 2\sum_{i=1}^{n}\sum_{j=1}^{m} x_{ij} \times 1_{\sum_{j=1}^{m} x_{ij}>0} - \sum_{i=1}^{n} 1_{\sum_{j=1}^{m} x_{ij}>0} \\ &= \sum_{i=1}^{n} d_i \times 1_{\sum_{j=1}^{m} x_{ij}>0} + 2\sum_{i=1}^{n} 1_{\sum_{j=1}^{m} x_{ij}>0}\sum_{j=1}^{m} x_{ij} - \sum_{i=1}^{n} 1_{\sum_{j=1}^{m} x_{ij}>0} \\ &= \sum_{i=1}^{n} d_i \times 1_{\sum_{j=1}^{m} x_{ij}>0} + 2\sum_{i=1}^{n}\sum_{j=1}^{m} x_{ij} - \sum_{i=1}^{n} 1_{\sum_{j=1}^{m} x_{ij}>0} \\ &= \sum_{i=1}^{n} (d_i - 1) \times 1_{\sum_{j=1}^{m} x_{ij}>0} + 2m = \sum_{i=1}^{n} (d_i - 1) \times \left|\sum_{j=1}^{m} x_{ij}\right|_0 + 2m, \end{aligned} \tag{2.1}$$

where $|\cdot|_0$ denotes the step function defined by $|s|_0 = 1$ if $s \neq 0$, 0 otherwise.

At the same time, we append new nodes in such a way that the balance of the tree is considered based on the given tree structure. The distance from the root to the deepest leaf node is defined by $\max_{i=\overline{1,n}} \log_2(T[i] \times B_i \times (\sum_{j=1}^{m} x_{ij} + 1))$. The distance from the root to the shallowest leaf node is defined by $\min_{i=\overline{1,n}:B_i=1} \log_2(T[i] \times B_i \times (\sum_{j=1}^{m} x_{ij} + 1)) = \min_{i=\overline{1,n}} \log_2(T[i] \times B_i \times (\sum_{j=1}^{m} x_{ij} + 1) + (1 - B_i) \times C)$, where C is a large constant which satisfies $C \geq 2^{h+1} \times m$. Formally, the balance condition can be expressed as

$$\begin{aligned} &\max_{i=\overline{1,n}} \log_2 \left(T[i] \times B_i \times (\sum_{j=1}^{m} x_{ij} + 1) \right) \\ &\quad - \min_{i=\overline{1,n}} \log_2 \left(T[i] \times B_i \times (\sum_{j=1}^{m} x_{ij} + 1) + (1 - B_i) \times C \right) \leq 1. \end{aligned} \tag{2.2}$$

It is evident that given an arbitrary tree structure and the number of new members, the balance condition (2.2) is not always possible. Therefore, we do not impose this condition as a constraint, rather,

we find a tree as balanced as possible by putting the balance term in the objective function.

Furthermore,

$$\begin{aligned}
&\max_{i=\overline{1,n}} \log_2 \left(T[i] \times B_i \times (\sum_{j=1}^{m} x_{ij} + 1) \right) \\
&\qquad - \min_{i=\overline{1,n}} \log_2 \left(T[i] \times B_i \times (\sum_{j=1}^{m} x_{ij} + 1) + (1 - B_i) \times C \right) \\
&= \log_2 \max_{i=\overline{1,n}} \left(T[i] \times B_i \times (\sum_{j=1}^{m} x_{ij} + 1) \right) \\
&\qquad - \log_2 \min_{i=\overline{1,n}} \left(T[i] \times B_i \times (\sum_{j=1}^{m} x_{ij} + 1) + (1 - B_i) \times C \right).
\end{aligned}$$

By mean value theorem (see Theorem 5.9 in [80]), there exists $c \in (c_1, c_2)$ where

$$\begin{aligned}
c_1 &= \min_{i=\overline{1,n}} \left(T[i] \times B_i \times (\sum_{j=1}^{m} x_{ij} + 1) + (1 - B_i) \times C \right), \\
c_2 &= \max_{i=\overline{1,n}} \left(T[i] \times B_i \times (\sum_{j=1}^{m} x_{ij} + 1) \right),
\end{aligned}$$

such that

$$\begin{aligned}
&\log_2 \max_{i=\overline{1,n}} \left(T[i] \times B_i \times (\sum_{j=1}^{m} x_{ij} + 1) \right) \\
&\qquad - \log_2 \min_{i=\overline{1,n}} \left(T[i] \times B_i \times (\sum_{j=1}^{m} x_{ij} + 1) + (1 - B_i) \times C \right) \\
&= \frac{1}{c \ln 2} \left(\max_{i=\overline{1,n}} \left(T[i] \times B_i \times (\sum_{j=1}^{m} x_{ij} + 1) \right) \right. \\
&\qquad \left. - \min_{i=\overline{1,n}} \left(T[i] \times B_i \times (\sum_{j=1}^{m} x_{ij} + 1) + (1 - B_i) \times C \right) \right) \\
&\leq \frac{1}{\ln 2} \left(\max_{i=\overline{1,n}} \left(T[i] \times B_i \times (\sum_{j=1}^{m} x_{ij} + 1) \right) \right. \\
&\qquad \left. - \min_{i=\overline{1,n}} \left(T[i] \times B_i \times (\sum_{j=1}^{m} x_{ij} + 1) + (1 - B_i) \times C \right) \right).
\end{aligned}$$

We introduce the notion of *balance coefficient* that is defined as the difference between the index of the deepest leaf node and the index of the shallowest leaf node, $\max_{i=\overline{1,n}}(T[i] \times B_i \times (\sum_{j=1}^{m} x_{ij} + 1)) -$

$\min_{i=\overline{1,n}}(T[i] \times B_i \times (\sum_{j=1}^{m} x_{ij} + 1) + (1 - B_i) \times C)$. By the above analysis, we can use the balance coefficient to control the balance of the tree: if the balance coefficient is small, the tree is encouraged to be more balanced. This coefficient is integrated into the objective function as a penalty term. It is noteworthy that the balance coefficient is much simpler than the original definition of the balance which is based on the distances from the root to the shallowest and the deepest nodes: the $\log_2$ function has been removed. This implies that our model will be simplified, and - consequently - is easier to handle. Finally, our optimization problem takes the following form:

$$\begin{aligned}
&\min \sum_{i=1}^{n}(d_i - 1) \times \left|\sum_{j=1}^{m} x_{ij}\right|_0 + \lambda \left[\max_{i=\overline{1,n}} \left(T[i] \times B_i \times \left(\sum_{j=1}^{m} x_{ij} + 1\right)\right)\right. \\
&\qquad \left. -\min_{i=\overline{1,n}} \left(T[i] \times B_i \times \left(\sum_{j=1}^{m} x_{ij} + 1\right) + (1 - B_i) \times C\right)\right] \qquad (2.3) \\
&\text{subject to } \sum_{i=1}^{n} x_{ij} = 1, \forall j = \overline{1,m}, x_{ij} \in \{0,1\}, \\
&\qquad \sum_{i=1}^{n} B_i \times \sum_{j=1}^{m} x_{ij} = m, \forall i = \overline{1,n}, \forall j = \overline{1,m},
\end{aligned}$$

where λ is a positive parameter controlling the trade-off between the cost of inserting new members and the balance coefficient of the tree after insertion.

It is observed that (2.3) is an optimization problem with binary variables and discontinuous objective. However, it can be reformulated as a combinatorial program with continuous objective by using new binary variables u_i as follows [36, 40]. Let $K = \{x : \sum_{i=1}^{n} x_{ij} = 1, \forall j = \overline{1,m}, x_{ij} \in [0,1], \sum_{i=1}^{n} B_i \times \sum_{j=1}^{m} x_{ij} = m, \forall i = \overline{1,n}, \forall j = \overline{1,m}\}$. Problem (2.3) and the following problem have the same optimal value (refer to Proposition 6 below)

$$\begin{aligned}
&\min \left\{ \sum_{i=1}^{n}(d_i - 1) \times u_i + \lambda(\max_{i=\overline{1,n}}(T[i] \times B_i \times (\sum_{j=1}^{m} x_{ij} + 1)) - \min_{i=\overline{1,n}}(T[i] \times B_i \times (\sum_{j=1}^{m} x_{ij} + 1)\right. \\
&\left. +(1 - B_i) \times C)) : x \in K, x \in \{0,1\}^{n \times m}, u \in \{0,1\}^n, 0 \le \sum_{j=1}^{m} x_{ij} \le m u_i, i = \overline{1,n} \right\} \qquad (2.4)
\end{aligned}$$

$$\begin{aligned}
&\Leftrightarrow \min \left\{ \sum_{i=1}^{n}(d_i - 1) \times u_i + \lambda(\max_{i=\overline{1,n}}(T[i] \times B_i \times (\sum_{j=1}^{m} x_{ij} + 1)) + \max_{i=\overline{1,n}}(-T[i] \times B_i \times \right. \\
&\left. (\sum_{j=1}^{m} x_{ij} + 1) + (B_i - 1) \times C)) : x \in K, x \in \{0,1\}^{n \times m}, u \in \{0,1\}^n, 0 \le \sum_{j=1}^{m} x_{ij} \le m u_i, i = \overline{1,n} \right\}. \qquad (2.5)
\end{aligned}$$

Proposition 6. *For $\lambda > 0$ the problems (2.3) and (2.4), (2.5) are equivalent, in the sense that they have the same optimal value and $x^* \in K$ is a solution of (2.3) iff there is $u^* \in \{0,1\}^n$ such that (x^*, u^*) is a solution of (2.4) and (2.5).*

It is observed that (2.5) is still a very difficult optimization problem with nonsmooth objective function and binary variables. We can - however - reformulate this problem as a DC program via an exact penalty technique [74, 48, 38]. Therefore, the reformulated program can be handled by DCA which is an efficient algorithm for DC programming.

This model is defined on the entire set of tree nodes instead of on only the set of leaf nodes, and we must store the matrix A of dimension $n \times n$ where n is the number of tree nodes. Therefore, the total space complexity of this optimization model is $O(n^2)$, obviously more complicated as the number of nodes in the tree increases. Because the new nodes should be inserted to leaf nodes, we propose an alternative algorithm including two steps described below. Therein, the first step of our algorithm consists in finding the set of all leaf nodes of the given tree. In the second step, based on the found leaf nodes, we consider an optimization problem which minimizes the insertion cost of new members while keeping the tree as balanced as possible. Clearly, the first step reduces considerably the complexity of the optimization model given in the second step which works only on the leaf nodes.

2.3.3 A two-step algorithm and the second optimization model

As the new nodes to be inserted should be leaf nodes, we first find all the leaf nodes in a full binary tree. Second, we propose an optimization model based on the found leaf nodes that minimizes the cost of adding new members while maintaining the tree's balance. The following describes the second optimization model.

Given a full binary tree that can be represented as an ordered set $T = \{2^a + b$: there exists a node at the b-th horizontal position of level a on the tree, $0 \leq a \leq h, 0 \leq b \leq 2^a - 1, a, b \in \mathbb{N}\}$, where h is the height of tree, we seek a set of the leaf nodes $L \subseteq T$ that is defined as $L = \{t \in T : 2t \notin T, 2t + 1 \notin T\}, l = |L|$. For convenience, we sort T, L in ascending order. The distance from the root to the leaf node $L[i]$ is given by $d_i = \lfloor \log_2 L[i] \rfloor$. Let the set of joining members be $M = \{1, 2, ..., m\}$.

Let x_{ij} be binary variables defined by $x_{ij} = 1$ if a new member $j \in M$ is inserted into the subtree below the leaf node $L[i]$, $x_{ij} = 0$ otherwise. The total cost for inserting all new members is calculated as same as described in Sect. 2.3.2.

$$F(x) = \sum_{i=1}^{l} (d_i - 1) \times \left| \sum_{j=1}^{m} x_{ij} \right|_0 + 2m, \tag{2.6}$$

Besides, the notion of balance coefficient is defined as the difference between the index of the deepest leaf node and the index of the shallowest leaf node, $\max_{i=\overline{1,l}}(L[i] \times (\sum_{j=1}^{m} x_{ij} + 1)) - \min_{i=\overline{1,l}}(L[i] \times (\sum_{j=1}^{m} x_{ij} + 1))$. Let $K' = \{x : \sum_{i=1}^{l} x_{ij} = 1, \forall j = \overline{1,m}, x_{ij} \in [0,1], \forall i = \overline{1,l}, \forall j = \overline{1,m}\}$. Finally, our second optimization model takes the following form:

$$\min \left\{ \sum_{i=1}^{l} (d_i - 1) \times u_i + \lambda(\max_{i=\overline{1,l}}(L[i] \times (\sum_{j=1}^{m} x_{ij} + 1)) + \max_{i=\overline{1,l}}(-L[i] \times (\sum_{j=1}^{m} x_{ij} + 1))) : x \in K', \right.$$
$$\left. x \in \{0,1\}^{l \times m}, u \in \{0,1\}^l, 0 \leq \sum_{j=1}^{m} x_{ij} \leq m u_i, i = \overline{1,l} \right\}. \tag{2.7}$$

Due to the fact that the optimization model of the two-step algorithm is defined on the set of leaf nodes rather than the entire set of tree nodes, its space complexity is calculated as $O(l)$, where l is the number of leaf nodes on the key tree, which is significantly less than the total number of tree nodes.

2.4 DCA for solving the group key update problem using batch insertion and individual deletion techniques

2.4.1 DCA for solving the problem (2.5)

Let $p(x,u)$ be the penalty function define by

$$p(x,u) := \sum_{i=1}^{n}\sum_{j=1}^{m} \min\{x_{ij}, 1-x_{ij}\} + \sum_{i=1}^{n} \min\{u_i, 1-u_i\}.$$

Notice that, there are some other possibilities to choose the penalty function. This function is purposefully chosen to get a polyhedral DC program, which enjoys interesting convergence properties.

Then (2.5) can be rewritten as

$$\min\left\{\sum_{i=1}^{n}(d_i-1)\times u_i + \lambda(\max_{i=\overline{1,n}}(T[i]\times B_i\times(\sum_{j=1}^{m}x_{ij}+1)) + \max_{i=\overline{1,n}}(-T[i]\times B_i\times(\sum_{j=1}^{m}x_{ij}+1)\right.$$
$$\left.+(B_i-1)\times C)) : x\in K, x\in[0,1]^{n\times m}, u\in[0,1]^n, 0\le\sum_{j=1}^{m}x_{ij}\le mu_i, i=\overline{1,n}, p(x,u)\le 0\right\}. \tag{2.8}$$

According to [74, 48, 38], there exists $\bar{t}$ such that the problem (2.8) and the following penalized problem are equivalent for all $t\ge\bar{t}$:

$$\min\left\{\sum_{i=1}^{n}(d_i-1)\times u_i + \lambda(\max_{i=\overline{1,n}}(T[i]\times B_i\times(\sum_{j=1}^{m}x_{ij}+1)) + \max_{i=\overline{1,n}}(-T[i]\times B_i\times(\sum_{j=1}^{m}x_{ij}+1)\right.$$
$$\left.+(B_i-1)\times C)) + tp(x,u) : x\in K, x\in[0,1]^{n\times m}, u\in[0,1]^n, 0\le\sum_{j=1}^{m}x_{ij}\le mu_i, i=\overline{1,n}\right\}. \tag{2.9}$$

Note that, in practice, the number $\bar{t}$ is generally hard to compute. Therefore, a common strategy is to use a quite large t at the beginning and adaptively increase this value to encourage the solutions to be binary. Once the solution is binary, we stop increasing t. In particular, for our following algorithm, when a binary solution is achieved at an iteration k, the solution will remain binary at all latter iterations (refer to Theorem 7).

Now let Δ be the feasible set of Problem (2.9), i.e. $\Delta := \{(x,u) : x\in K, u\in[0,1]^n, 0\le$

$\sum_{j=1}^{m} x_{ij} \leq mu_i, i = \overline{1,n}\}$ and

$$\zeta(x,u) = \lambda(\max_{i=\overline{1,n}}(T[i]\times B_i \times (\sum_{j=1}^{m} x_{ij} + 1)) + \max_{i=\overline{1,n}}(-T[i] \times B_i \times (\sum_{j=1}^{m} x_{ij} + 1) + (B_i - 1) \times C)).$$

Since p is concave, the following DC formulation of (2.9) seems to be natural:

$$\min\{f(x,u) = g(x,u) - h(x,u) : (x,u) \in \mathbb{R}^{n\times m} \times \mathbb{R}^n\} \tag{2.10}$$

where $g(x,u) := \zeta(x,u) + \chi_\Delta(x,u)$, $h(x,u) := -\sum_{i=1}^{n}(d_i - 1) \times u_i - tp(x,u)$,
are clearly convex functions. In addition, the function h is polyhedral convex and therefore (2.10) is a polyhedral DC program.
According to the above-described general DCA scheme, applying DCA to (2.10) amounts to computing two sequences $\{(x^k, u^k)\}$ and $\{(y^k, v^k)\}$ in the way that $(y^k, v^k) \in \partial h(x^k, u^k)$ and (x^{k+1}, u^{k+1}) solves the convex program of the form (P_k). Since $(y^k, v^k) \in \partial h(x^k, u^k)$ is equivalent to

$$y_{ij}^k = \begin{cases} t, & \text{if } x_{ij}^k \geq 0.5 \\ -t, & \text{if } x_{ij}^k < 0.5, \end{cases} \quad \forall i = 1...n, \forall j = 1...m, \tag{2.11}$$

$$v_i^k = \begin{cases} (1 - d_i) + t, & \text{if } u_i^k \geq 0.5 \\ (1 - d_i) - t, & \text{if } u_i^k < 0.5, \end{cases} \quad \forall i = 1...n, \tag{2.12}$$

the algorithm can be described as follow.

Algorithm 4 DCAEP1 for solving the problem (2.10)

Initialization: Let $(x^0, u^0) \in [0,1]^{n\times m} \times [0,1]^n$ be a guess, set $k := 0, \epsilon > 0, t_k > 0, \theta > 0$.
repeat
Compute $(y^k, v^k) \in \partial h(x^k, u^k)$ via (2.11) and (2.12).
Solve the following convex program to obtain (x^{k+1}, u^{k+1})

$$\min\{\zeta(x,u) - \langle x, y^k\rangle - \langle u, v^k\rangle : (x,u) \in \Delta\}. \tag{2.13}$$

if $p(x^{k+1}, u^{k+1}) > 0$ **then**
$t_{k+1} \leftarrow t_k + \theta$.
end if
$k \leftarrow k + 1$.
until $|f(x^{k+1}, u^{k+1}) - f(x^k, u^k)| \leq \epsilon(|f(x^k, u^k)| + 1)$

Note that, the convex problem (2.13) can be solved as follows. By introducing new variables $\xi = \max_{i=\overline{1,n}}(T[i] \times B_i \times (\sum_{j=1}^{m} x_{ij} + 1)), \mu = \max_{i=\overline{1,n}}(-T[i] \times B_i \times (\sum_{j=1}^{m} x_{ij} + 1) + (B_i - 1) \times C)$, we can reformulate (2.13) as the following linear program which can be solved efficiently by existing optimization packages

$$\min \left\{ \lambda(\xi+\mu) - \langle x, y^k\rangle - \langle u, v^k\rangle : (x,u) \in \Delta,\ T[i] \times B_i \times (\sum_{j=1}^{m} x_{ij} + 1) \le \xi, i = \overline{1,n}, \right.$$
$$\left. -T[i] \times B_i \times (\sum_{j=1}^{m} x_{ij} + 1) + (B_i - 1) \times C \le \mu, i = \overline{1,n} \right\}. \tag{2.14}$$

The sequence $\{(x^k, u^k)\}$ generated by DCAEP1 enjoys the following property: if the penalty parameter is large enough and the solution at an iteration is binary, then the solution remains binary for the succeeding iterations.

The convergence of DCAEP1 is stated as follows.

Theorem 7. *(Convergence properties of DCAEP1)*

(i) DCAEP1 generates a sequence $\{(x^k, u^k)\}$ contained in $V(\Delta)$ such that the sequence $\{f(x^k, u^k)\}$ is decreasing.

(ii) There exists t^* such that: if $t_k > t^*$ and (x^k, u^k) is binary, then (x^s, u^s) remains binary for all $s \ge k$.

(iii) The sequence $\{(x^k, u^k)\}$ converges to $(x^*, u^*) \in V(\Delta)$ after a finite number of iterations. The point (x^*, u^*) is a critical point of Problem (2.10)*. Moreover if $u_i^* \ne \frac{1}{2}$ for all $i = \overline{1,n}$, then (x^*, u^*) is a local solution to* (2.10)*.*

Proof. (i) is consequence of DCA's convergence theorem for a generic standard DC program.

(ii) Let $(\xi^{k+1}, \mu^{k+1}, x^{k+1}, u^{k+1})$ be the optimal solution of the problem (2.14). We denote

$$\Xi := \left\{ (\xi, \mu, x, u) : (x,u) \in \Delta, T[i] \times B_i \times (\sum_{j=1}^{m} x_{ij} + 1) \le \xi, i = \overline{1,n}, \right.$$
$$\left. -T[i] \times B_i \times (\sum_{j=1}^{m} x_{ij} + 1) + (B_i - 1) \times C \le \mu, i = \overline{1,n} \right\}$$

the polyhedral convex set of (2.14). Since the objective of (2.14) is linear, it is well known that for all $k \ge 0$, $(\xi^{k+1}, \mu^{k+1}, x^{k+1}, u^{k+1})$ belongs to $V(\Xi)$ - the vertex set of Ξ. We denote $\bar{V} = \{(x,u) : \exists \xi, \mu \text{ such that } (\xi, \mu, x, u) \in V(\Xi)\} \cup \{(x^0, u^0)\}$.

If $p(x,u) = p(x',u') \quad \forall (x,u), (x',u') \in \bar{V}$, the assertion is trivial (if $p(x^k, u^k) = 0$ then $p(x^s, u^s) = 0,\ \forall s$). Otherwise, let $\tilde{\zeta}(x,u) = \zeta(x,u) + \sum_{i=1}^{n}(d_i - 1)u_i$ and

$$t^* = \sup \left\{ \frac{\tilde{\zeta}(x,u) - \tilde{\zeta}(x',u')}{p(x',u') - p(x,u)} : (x,u), (x',u') \in \bar{V}, p(x,u) < p(x',u') \right\} > -\infty. \tag{2.15}$$

Since $V(\Xi)$ is finite, so is $\bar{V}$. Consequently, $t^* < +\infty$.

By using similar arguments to (Theorem 8 in [74]), we proceed as follows. Assume by contradiction, let $t_k > t^*$ and (x^k, u^k) is binary but (x^{k+1}, u^{k+1}) is not binary. Then, $p(x^k, u^k) = 0$ and

$p(x^{k+1}, u^{k+1}) > 0$, so $p(x^k, u^k) < p(x^{k+1}, u^{k+1})$. From (2.15),

$$t_k > t^* \geq \frac{\tilde{\zeta}(x^k, u^k) - \tilde{\zeta}(x^{k+1}, u^{k+1})}{p(x^{k+1}, u^{k+1}) - p(x^k, u^k)}, \tag{2.16}$$

equivalently,

$$\tilde{\zeta}(x^{k+1}, u^{k+1}) + t_k p(x^{k+1}, u^{k+1}) > \tilde{\zeta}(x^k, u^k) + t_k p(x^k, u^k). \tag{2.17}$$

However, since (x^{k+1}, u^{k+1}) is the optimal solution of (2.13), we have

$$\tilde{\zeta}(x^{k+1}, u^{k+1}) + t_k p(x^{k+1}, u^{k+1}) \leq \tilde{\zeta}(x^k, u^k) + t_k p(x^k, u^k), \tag{2.18}$$

which is a contradiction.

(iii) (2.10) is a polyhedral DC program. By applying the finite convergence of DCA for polyhedral DC programs [71], DCAEP1 has a finite convergence, say, the sequence $\{(x^k, u^k)\}$ has a subsequence that converges to a critical point $(x^*, u^*) \in V(\Delta)$ after a finite number of iterations.

Moreover if $u_i^* \neq \frac{1}{2}$ for all $i = \overline{1, n}$, then the function h is differentiable at (x^*, u^*). Therefore, the necessary local condition

$$\emptyset \neq \partial h(x^*, u^*) \subset \partial g(x^*, u^*)$$

holds. Since h is a polyhedral convex function and $(x^*, u^*) \in \mathcal{P}_l = \{(x, u) \in \Delta : \partial h(x, u) \subset \partial g(x, u)\}$, then (x^*, u^*) is a local minimizer of (2.10) [71]. The proof is then complete.

□

2.4.2 A two-step DCA based algorithm for solving the problem (2.7)

As the problem (2.7) is defined on the set of leaf nodes instead of the entire set of tree nodes, we first find all leaf nodes in a full binary tree. A node $t \in T$ is a leaf node if and only if there are no nodes $2t$ and $2t + 1$ on the tree. To check if node x is in the set T, we perform the binary search algorithm. The algorithm for finding leaf nodes in a full binary tree is described as follows.

Algorithm 5 Algorithm of finding leaf nodes of a given tree T

Initialization: $k = 1, L = []$.
for $a = 0$ to h **do**
  for $b = 0$ to $2^a - 1$ **do**
    if $(2^{(a+1)} + 2b \notin T)$ **and** $(2^{(a+1)} + 2b + 1 \notin T)$ **then**
      $L[k] = 2^a + b$
      $k = k + 1$
    end if
  end for
end for

After finding the set of leaf nodes on the key tree, we can use the same exact penalty techniques and DCA based algorithm described in Sect. 2.4.1 to solve the problem (2.7).

Algorithm 6 DCAEP2 for solving the problem (2.7)

Step 1: Apply the algorithm for finding leaf nodes in a full binary key tree.
Step 2:
Initialization: Let $(x^0, u^0) \in [0,1]^{l\times m} \times [0,1]^l$ be a guess, set $k := 0, \epsilon > 0, t_k > 0, \theta > 0$.
repeat
Compute $(y^k, v^k) \in \partial h(x^k, u^k)$ as $y_{ij}^k = t_k$ if $x_{ij}^k \geq 0.5$, $y_{ij}^k = -t_k$ otherwise, $v_i^k = (1-d_i)+t_k$ if $u_i^k \geq 0.5$, $v_i^k = (1-d_i) - t_k$ otherwise, $\forall i = \overline{1,l}, \forall j = \overline{1,m}$.
Solve the following convex program to obtain (x^{k+1}, u^{k+1})

$$\min \{\zeta(x,u) - \langle x, y^k\rangle - \langle u, v^k\rangle : (x,u) \in \Delta'\} \tag{2.19}$$

where $\Delta' := \{(x,u) : x \in K', u \in [0,1]^l, 0 \leq \sum_{j=1}^m x_{ij} \leq mu_i, i = \overline{1,l}\}$.
if $p(x^{k+1}, u^{k+1}) > 0$ **then**
$t_{k+1} \leftarrow t_k + \theta$.
end if
$k = k + 1$.
until $|f(x^{k+1}, u^{k+1}) - f(x^k, u^k)| \leq \epsilon(|f(x^k, u^k)| + 1)$

2.5 Two-step algorithm for the group key update problem using batch rekeying technique

2.5.1 Problem description

We consider the problem of batch deletion and insertion members in the LKH structure. When the membership of a group changes dynamically, the group key and all related keys on the binary tree must be updated. In batch rekeying, the key server aggregates the total number of joining and leaving members over a specified time interval and then changes the associated keys. Therefore, batch rekeying techniques increase efficiency in the number of required messages, take advantage of the possible overlap of new keys for multiple rekeying requests, and then reduce the possibility of generating redundant new keys. In actuality, the deletion nodes are also the leaf nodes in the tree, and a subtree of new nodes can be appended below a leaf node or replaced by the position of the leaving node on the binary key tree. The latter has a lower updating key cost than the former since, when a member leaves, all the keys on the path from the root to the deletion node must be updated anyway. Our main goal here is to find a set of leaf nodes in a binary key tree to delete leaving members and insert new members while minimizing the rekeying cost (the number of updated keys) and keeping the tree as balanced as possible.

2.5.2 The optimization model

As the deletion and insertion nodes should be leaf nodes, we first find all the leaf nodes in a full binary tree. Second, we propose an optimization model based on the found leaf nodes that minimizes the cost of deleting departure members and adding new members while maintaining the tree's balance. The following describes the optimization model.

Given a full binary tree that can be represented as an ordered set $T = \{2^a + b$: there exists a node at the b-th horizontal position of level a on the tree, $0 \leq a \leq h, 0 \leq b \leq 2^a - 1, a, b \in \mathbb{N}\}$, where h is the height of tree, we seek a set of the leaf nodes $L \subseteq T$ that is defined as $L = \{t \in T : 2t \notin T, 2t+1 \notin T\}, l = |L|$. A is a set of leaving members and $A \subseteq L, l_2 = |A|$. The set of remaining leaf nodes on

the tree T is denoted as $L' = L \setminus A, l_1 = |L \setminus A|$. For convenience, we sort T, L, A, L' in ascending order. The distance from the root to the leaf node $L'[i]$ and the departure leaf node $A[k]$ are given by $d_i = \lfloor \log_2 L'[i] \rfloor, i = \overline{1, l_1}$ and $d_k = \lfloor \log_2 A[k] \rfloor, k = \overline{1, l_2}$, respectively. Let the set of joining members be $M = \{1, 2, \ldots, m\}$.

Remark 3. *Each leaving node will be either replaced with a certain number of new members or only deleted from the key tree (illustrated in Fig. 2.1). If a subtree of new nodes is replaced the position of leaving nodes, the rekeying cost gets reduced by the number of all keys on the path from the root to the deletion node which are certainly required to be updated when a member assigned to this node is leaving from the group.*

Now we propose the optimization model for minimizing the key updating cost. Let x_{ij} be binary variables defined by $x_{ij} = 1$ if a new member $j \in M$ is inserted into the subtree below the leaf node $L'[i]$, $x_{ij} = 0$ otherwise, $i = \overline{1, l_1}, j = \overline{1, m}$. Let y_{kj} be binary variables defined by $y_{kj} = 1$ if a leaf node $A[k]$ is deleted and a new member $j \in M$ is inserted into the subtree at the leaf node $A[k]$, $y_{kj} = 0$ if the leaf node $A[k]$ is only deleted, $k = \overline{1, l_2}, j = \overline{1, m}$.

Since every new member is appended below only one leaf node of the original tree, $\sum_{i=1}^{l_1} x_{ij} + \sum_{k=1}^{l_2} y_{kj} = 1$, $\forall j = \overline{1, m}$. For a given value $i \in \{1, 2, ..., l_1\}$, the number of new nodes j being inserted at the leaf node $L'[i]$ is calculated as $m_i = \sum_{j=1}^{m} x_{ij}$. It means that a subtree at the leaf node $L'[i]$ includes $m_i + 1$ leaf nodes (including m_i new nodes and the old leaf node $L'[i]$) as illustrated in Fig. 2.1. For the leaf node $L'[i]$ that is chosen to append some new members (equivalent to $\sum_{j=1}^{m} x_{ij} > 0$), the cost to build a corresponding subtree is $2\sum_{j=1}^{m} x_{ij} - 1$. Besides, the cost to update keys from the root to $L'[i]$ is d_i. More precisely, we have

$$\begin{aligned} \text{cost at } L'[i] &= \begin{cases} d_i + 2\sum_{j=1}^{m} x_{ij} - 1, & \text{if } \sum_{j=1}^{m} x_{ij} > 0 \\ 0, & \text{if } \sum_{j=1}^{m} x_{ij} = 0, \end{cases} \\ &= d_i \times 1_{\sum_{j=1}^{m} x_{ij} > 0} + 2\sum_{j=1}^{m} x_{ij} \times 1_{\sum_{j=1}^{m} x_{ij} > 0} - 1_{\sum_{j=1}^{m} x_{ij} > 0}. \end{aligned}$$

In Sect. 2.3.2, we define the approximation inserting cost function as follows:

$$F(x) = \sum_{i=1}^{l_1} (d_i - 1) \times \left| \sum_{j=1}^{m} x_{ij} \right|_0 + 2\sum_{i=1}^{l_1}\sum_{j=1}^{m} x_{ij}, \tag{2.20}$$

where $|\cdot|_0$ denotes the step function defined by $|s|_0 = 1$ if $s \neq 0$, 0 otherwise.

Moreover, we calculate the cost for a set of leaving nodes. For a given value $k \in \{1, 2, ..., l_2\}$, the number of new nodes j being inserted at the leaf node $A[k]$ is calculated as $m_k = \sum_{j=1}^{m} y_{kj}$. It means that a subtree at the leaf node $A[k]$ includes m_k leaf nodes as illustrated in Fig. 2.1. For the leaf node $A[k]$ that is chosen to append some new members (equivalent to $\sum_{j=1}^{m} y_{kj} > 0$), the cost to build a corresponding subtree is $2\sum_{j=1}^{m} y_{kj} - 1$. Besides, the cost to update keys from the root to $A[k]$ is $d_k - 1$. More precisely, we have

$$\text{cost at } A[k] = \begin{cases} (d_k - 1) + 2\sum_{j=1}^{m} y_{kj} - 1, & \text{if } \sum_{j=1}^{m} y_{kj} > 0 \\ (d_k - 1), & \text{if } \sum_{j=1}^{m} y_{kj} = 0, \end{cases}$$
$$= (d_k - 1) + 2\sum_{j=1}^{m} y_{kj} \times 1_{\sum_{j=1}^{m} y_{kj} > 0} - 1_{\sum_{j=1}^{m} y_{kj} > 0}.$$

Therefore, the approximation cost function for a set of departure nodes as follows:

$$F(y) = \sum_{k=1}^{l_2} (d_k - 1) + 2\sum_{k=1}^{l_2}\sum_{j=1}^{m} y_{kj} - \sum_{k=1}^{l_2} \left| \sum_{j=1}^{m} y_{kj} \right|_0 . \tag{2.21}$$

We obtain the following total cost for deleting leaving members and inserting new members

$$\begin{aligned} F(x,y) &= \sum_{i=1}^{l_1} (d_i - 1) \times \left| \sum_{j=1}^{m} x_{ij} \right|_0 + 2\sum_{i=1}^{l_1}\sum_{j=1}^{m} x_{ij} \\ &+ \sum_{k=1}^{l_2} (d_k - 1) + 2\sum_{k=1}^{l_2}\sum_{j=1}^{m} y_{kj} - \sum_{k=1}^{l_2} \left| \sum_{j=1}^{m} y_{kj} \right|_0 \\ &= \sum_{k=1}^{l_2} (d_k - 1) + \sum_{i=1}^{l_1} (d_i - 1) \times \left| \sum_{j=1}^{m} x_{ij} \right|_0 - \sum_{k=1}^{l_2} \left| \sum_{j=1}^{m} y_{kj} \right|_0 \\ &+ 2\sum_{i=1}^{l_1}\sum_{j=1}^{m} x_{ij} + 2\sum_{k=1}^{l_2}\sum_{j=1}^{m} y_{kj} \\ &= \sum_{k=1}^{l_2} (d_k - 1) + \sum_{i=1}^{l_1} (d_i - 1) \times \left| \sum_{j=1}^{m} x_{ij} \right|_0 - \sum_{k=1}^{l_2} \left| \sum_{j=1}^{m} y_{kj} \right|_0 + 2m. \end{aligned} \tag{2.22}$$

At the same time, we append new nodes in such a way that the balance of the tree is considered based on the given tree structure. We introduce the notion of *balance coefficient* that is defined as the difference between the index of the deepest leaf node and the index of the shallowest leaf node, $\max_{i=\overline{1,l_1},k=\overline{1,l_2}}(L'[i] \times (\sum_{j=1}^{m} x_{ij} + 1), \frac{A[k]}{2} \times (\sum_{j=1}^{m} y_{kj} + 1)) - \min_{i=\overline{1,l_1},k=\overline{1,l_2}}(L'[i] \times (\sum_{j=1}^{m} x_{ij} + 1), \frac{A[k]}{2} \times (\sum_{j=1}^{m} y_{kj} + 1))$. By the above analysis, we can use the balance coefficient to control the balance of the tree: if the balance coefficient is small, the tree is encouraged to be more balanced. This coefficient is integrated into the objective function as a penalty term. It is noteworthy that the balance coefficient is much simpler than the original definition of the balance which is based on the distances from the root to the shallowest and the deepest nodes. This implies that our model will be simplified, and - consequently - is easier to handle. Finally, our optimization problem takes the following form:

$$\begin{aligned}
&\min \sum_{i=1}^{l_1}(d_i-1)\times\left|\sum_{j=1}^{m}x_{ij}\right|_0-\sum_{k=1}^{l_2}\left|\sum_{j=1}^{m}y_{kj}\right|_0 \\
&+\lambda\left[\max_{i=\overline{1,l_1},k=\overline{1,l_2}}\left(L'[i]\times\left(\sum_{j=1}^{m}x_{ij}+1\right),\frac{A[k]}{2}\times\left(\sum_{j=1}^{m}y_{kj}+1\right)\right)\right. \\
&\left.-\min_{i=\overline{1,l_1},k=\overline{1,l_2}}\left(L'[i]\times\left(\sum_{j=1}^{m}x_{ij}+1\right),\frac{A[k]}{2}\times\left(\sum_{j=1}^{m}y_{kj}+1\right)\right)\right] \\
&\text{subject to } \sum_{i=1}^{l_1}x_{ij}+\sum_{k=1}^{l_2}y_{kj}=1, j=\overline{1,m}, \\
&\qquad x_{ij}\in\{0,1\}, y_{kj}\in\{0,1\}, i=\overline{1,l_1}, k=\overline{1,l_2}, j=\overline{1,m},
\end{aligned} \tag{2.23}$$

where λ is a positive parameter controlling the trade-off between the rekeying cost and the balance coefficient of the tree after insertion and deletion.

It is observed that (2.23) is an optimization problem with binary variables and discontinuous objective. However, it can be reformulated as a combinatorial program with continuous objective by using new binary variables u_i and v_k as follows. Let $K = \{(x,y) : \sum_{i=1}^{l_1} x_{ij} + \sum_{k=1}^{l_2} y_{kj} = 1, \forall j = \overline{1,m}, x_{ij} \in [0,1], y_{kj} \in [0,1], \forall i = \overline{1,l_1}, \forall k = \overline{1,l_2}, \forall j = \overline{1,m}\}$. Problem (2.23) and the following problem have the same optimal value (refer to Proposition 8 below)

$$\begin{aligned}
&\min\left\{\sum_{i=1}^{l_1}(d_i-1)\times u_i-\sum_{k=1}^{l_2}v_k+\lambda\left(\max_{i=\overline{1,l_1},k=\overline{1,l_2}}\left(L'[i]\times(\sum_{j=1}^{m}x_{ij}+1),\frac{A[k]}{2}\times(\sum_{j=1}^{m}y_{kj}+1)\right)\right.\right. \\
&\left.-\min_{i=\overline{1,l_1},k=\overline{1,l_2}}\left(L'[i]\times(\sum_{j=1}^{m}x_{ij}+1),\frac{A[k]}{2}\times(\sum_{j=1}^{m}y_{kj}+1)\right)\right):(x,y)\in K, x\in\{0,1\}^{l_1\times m}, \\
&\left.y\in\{0,1\}^{l_2\times m}, u\in\{0,1\}^{l_1}, v\in\{0,1\}^{l_2}, 0\le\sum_{j=1}^{m}x_{ij}\le mu_i, i=\overline{1,l_1}, \sum_{j=1}^{m}y_{kj}\ge v_k, k=\overline{1,l_2}\right\}
\end{aligned} \tag{2.24}$$

$$\begin{aligned}
&\Leftrightarrow\min\left\{\sum_{i=1}^{l_1}(d_i-1)\times u_i-\sum_{k=1}^{l_2}v_k+\lambda\left(\max_{i=\overline{1,l_1},k=\overline{1,l_2}}\left(L'[i]\times(\sum_{j=1}^{m}x_{ij}+1),\frac{A[k]}{2}\times(\sum_{j=1}^{m}y_{kj}+1)\right)\right.\right. \\
&\left.+\max_{i=\overline{1,l_1},k=\overline{1,l_2}}\left(-L'[i]\times(\sum_{j=1}^{m}x_{ij}+1),-\frac{A[k]}{2}\times(\sum_{j=1}^{m}y_{kj}+1)\right)\right):(x,y)\in K, x\in\{0,1\}^{l_1\times m}, \\
&\left.y\in\{0,1\}^{l_2\times m}, u\in\{0,1\}^{l_1}, v\in\{0,1\}^{l_2}, 0\le\sum_{j=1}^{m}x_{ij}\le mu_i, i=\overline{1,l_1}, \sum_{j=1}^{m}y_{kj}\ge v_k, k=\overline{1,l_2}\right\}.
\end{aligned} \tag{2.25}$$

Proposition 8. *For $\lambda > 0$ the problems (2.23) and (2.24), (2.25) are equivalent, in the sense that they have the same optimal value and $(x^*, y^*) \in K$ is a solution of (2.23) iff there is $u^* \in \{0,1\}^{l_1}, v^* \in \{0,1\}^{l_2}$ such that (x^*, y^*, u^*, v^*) is a solution of (2.24) and (2.25).*

It is observed that (2.25) is still a very difficult optimization problem with nonsmooth objective function and binary variables. We can - however - reformulate this problem as a DC program via an exact penalty technique [74, 48, 38]. Therefore, the reformulated program can be handled by DCA which is an efficient algorithm for DC programming.

2.6 DCA for solving the group key update problem using batch rekeying technique

As the problem (2.25) is defined on the set of leaf nodes included all leaving nodes, we first find leaf nodes in a full binary tree using Algorithm 5 in Sect. 2.4.2. After finding the set of leaf nodes on the key tree, we can use the exact penalty techniques to solve the problem (2.25). Let $p(x, y, u, v)$ be the penalty function define by

$$p(x,y,u,v) := \sum_{i=1}^{l_1}\sum_{j=1}^{m} \min\{x_{ij}, 1-x_{ij}\} + \sum_{k=1}^{l_2}\sum_{j=1}^{m} \min\{y_{kj}, 1-y_{kj}\} + \sum_{i=1}^{l_1} \min\{u_i, 1-u_i\} + \sum_{k=1}^{l_2} \min\{v_k, 1-v_k\}.$$

Notice that, there are some other possibilities to choose the penalty function. This function is purposefully chosen to get a polyhedral DC program, which enjoys interesting convergence properties.

Then (2.25) can be rewritten as

$$\min \left\{ \sum_{i=1}^{l_1}(d_i-1)\times u_i - \sum_{k=1}^{l_2} v_k + \lambda \left(\max_{i=\overline{1,l_1},k=\overline{1,l_2}} \left(L'[i]\times(\sum_{j=1}^{m} x_{ij}+1), \frac{A[k]}{2}\times(\sum_{j=1}^{m} y_{kj}+1) \right) \right.\right.$$
$$\left.\left. + \max_{i=\overline{1,l_1},k=\overline{1,l_2}} \left(-L'[i]\times(\sum_{j=1}^{m} x_{ij}+1), -\frac{A[k]}{2}\times(\sum_{j=1}^{m} y_{kj}+1) \right) \right) : (x,y)\in K, x\in[0,1]^{l_1\times m}, \right.$$
$$\left. y\in[0,1]^{l_2\times m}, u\in[0,1]^{l_1}, v\in[0,1]^{l_2}, 0\le \sum_{j=1}^{m} x_{ij} \le m u_i,\ i=\overline{1,l_1}, \sum_{j=1}^{m} y_{kj} \ge v_k, k=\overline{1,l_2}, p(x,y,u,v)\le 0 \right\}. \tag{2.26}$$

According to [74, 48, 38], there exists $\bar{t}$ such that the problem (2.26) and the following penalized problem are equivalent for all $t \ge \bar{t}$:

$$\min \left\{ \sum_{i=1}^{l_1}(d_i-1)\times u_i - \sum_{k=1}^{l_2} v_k + \lambda \left(\max_{i=\overline{1,l_1},k=\overline{1,l_2}} \left(L'[i]\times(\sum_{j=1}^{m} x_{ij}+1), \frac{A[k]}{2}\times(\sum_{j=1}^{m} y_{kj}+1) \right) \right.\right.$$
$$\left.\left. + \max_{i=\overline{1,l_1},k=\overline{1,l_2}} \left(-L'[i]\times(\sum_{j=1}^{m} x_{ij}+1), -\frac{A[k]}{2}\times(\sum_{j=1}^{m} y_{kj}+1) \right) \right) + tp(x,y,u,v) : (x,y)\in K,$$
$$x\in[0,1]^{l_1\times m}, y\in[0,1]^{l_2\times m}, u\in[0,1]^{l_1}, v\in[0,1]^{l_2}, 0\le \sum_{j=1}^{m} x_{ij}\le m u_i, i=\overline{1,l_1}, \sum_{j=1}^{m} y_{kj}\ge v_k, k=\overline{1,l_2} \right\}. \tag{2.27}$$

Note that, in practice, the number $\bar{t}$ is generally hard to compute. Therefore, a common strategy is to use a quite large t at the beginning and adaptively increase this value to encourage the solutions to be binary. Once the solution is binary, we stop increasing t. In particular, for our following algorithm, when a binary solution is achieved at an iteration k, the solution will remain binary at all latter iterations (refer to Theorem 9).

Now let Δ be the feasible set of Problem (2.27), i.e. $\Delta := \{(x,y,u,v) : (x,y)\in K, u\in[0,1]^{l_1}, v\in [0,1]^{l_2}, 0\le\sum_{j=1}^{m} x_{ij}\le m u_i, i=\overline{1,l_1}, \sum_{j=1}^{m} y_{kj}\ge v_k, k=\overline{1,l_2}\}$ and

$$\zeta(x,y,u,v) = \lambda \left(\max_{i=\overline{1,l_1},k=\overline{1,l_2}} \left(L'[i]\times(\sum_{j=1}^{m} x_{ij}+1), \frac{A[k]}{2}\times(\sum_{j=1}^{m} y_{kj}+1) \right) \right.$$
$$\left. + \max_{i=\overline{1,l_1},k=\overline{1,l_2}} \left(-L'[i]\times(\sum_{j=1}^{m} x_{ij}+1), -\frac{A[k]}{2}\times(\sum_{j=1}^{m} y_{kj}+1) \right) \right).$$

Since p is concave, the following DC formulation of (2.27) seems to be natural:

$$\begin{aligned} \min\{ f(x,y,u,v) =& g(x,y,u,v) - h(x,y,u,v) : \\ & (x,y,u,v)\in \mathbb{R}^{l_1\times m}\times\mathbb{R}^{l_2\times m}\times\mathbb{R}^{l_1}\times\mathbb{R}^{l_2}\} \end{aligned} \tag{2.28}$$

where $g(x,y,u,v) := \zeta(x,y,u,v) + \chi_\Delta(x,y,u,v)$,
$h(x,y,u,v) := -\sum_{i=1}^{l_1}(d_i-1)\times u_i + \sum_{k=1}^{l_2} v_k - tp(x,y,u,v)$, are clearly convex functions. In addition, the function h is polyhedral convex and therefore (2.28) is a polyhedral DC program.

According to the above-described general DCA scheme, applying DCA to (2.28) amounts to computing two sequences $\{(x^l,y^l,u^l,v^l)\}$ and $\{(\alpha^l,\beta^l,\gamma^l,\sigma^l)\}$ in the way that $(\alpha^l,\beta^l,\gamma^l,\sigma^l)\in\partial h(x^l,y^l,u^l,v^l)$ and $(x^{l+1},y^{l+1},u^{l+1},v^{l+1})$ solves the convex program of the form (P_{dc}). Since $(\alpha^l,\beta^l,\gamma^l,\sigma^l)\in\partial h(x^l,y^l,u^l,v^l)$ is equivalent to

$$\alpha_{ij}^l = \begin{cases} t, & \text{if } x_{ij}^l \ge 0.5 \\ -t, & \text{if } x_{ij}^l < 0.5, \end{cases} \quad \forall i=1...l_1, \forall j=1...m, \tag{2.29}$$

$$\beta_{kj}^l = \begin{cases} t, & \text{if } y_{kj}^l \ge 0.5 \\ -t, & \text{if } y_{kj}^l < 0.5, \end{cases} \quad \forall k=1...l_2, \forall j=1...m, \tag{2.30}$$

$$\gamma_i^l = \begin{cases} (1-d_i)+t, & \text{if } u_i^l \geq 0.5 \\ (1-d_i)-t, & \text{if } u_i^l < 0.5, \end{cases} \forall i = 1...l_1, \tag{2.31}$$

$$\sigma_k^l = \begin{cases} 1+t, & \text{if } v_k^l \geq 0.5 \\ 1-t, & \text{if } v_k^l < 0.5, \end{cases} \forall k = 1...l_2, \tag{2.32}$$

the algorithm can be described as follow.

Algorithm 7 DCAEP+ for solving the problem (2.28)

Step 1: Apply the algorithm for finding leaf nodes in a full binary key tree.
Step 2:
Initialization: Let $(x^0, y^0, u^0, v^0) \in [0,1]^{l_1 \times m} \times [0,1]^{l_2 \times m} \times [0,1]^{l_1} \times [0,1]^{l_2}$ be a guess, set $l := 0, \epsilon > 0, t_l > 0, \theta > 0$.
repeat
Compute $(\alpha^l, \beta^l, \gamma^l, \sigma^l) \in \partial h(x^l, y^l, u^l, v^l)$ via (2.29), (2.30), (2.31), and (2.32).
Solve the following convex program to obtain $(x^{l+1}, y^{l+1}, u^{l+1}, v^{l+1})$

$$\min \{\zeta(x,y,u,v) - \langle x, \alpha^l \rangle - \langle y, \beta^l \rangle - \langle u, \gamma^l \rangle - \langle v, \sigma^l \rangle : (x,y,u,v) \in \Delta\}. \tag{2.33}$$

if $p(x^{l+1}, y^{l+1}, u^{l+1}, v^{l+1}) > 0$ **then**
$t_{l+1} \leftarrow t_l + \theta$.
end if
$l = l + 1$.
until $|f(x^{l+1}, y^{l+1}, u^{l+1}, v^{l+1}) - f(x^l, y^l, u^l, v^l)| \leq \epsilon(|f(x^l, y^l, u^l, v^l)| + 1)$

Note that, the convex problem (2.33) can be solved as follows. By introducing new variables $\xi = \max_{i=\overline{1,l_1}, k=\overline{1,l_2}}(L'[i] \times (\sum_{j=1}^m x_{ij} + 1), \frac{A[k]}{2} \times (\sum_{j=1}^m y_{kj} + 1)), \mu = \max_{i=\overline{1,l_1}, k=\overline{1,l_2}}(-L'[i] \times (\sum_{j=1}^m x_{ij} + 1), -\frac{A[k]}{2} \times (\sum_{j=1}^m y_{kj} + 1))$, we can reformulate (2.33) as the following linear program which can be solved efficiently by existing optimization packages

$$\min \Bigg\{ \lambda(\xi+\mu) - \langle x, \alpha^l \rangle - \langle y, \beta^l \rangle - \langle u, \gamma^l \rangle - \langle v, \sigma^l \rangle : (x,y,u,v) \in \Delta, L'[i] \times (\sum_{j=1}^m x_{ij} + 1) \leq \xi,$$
$$-L'[i] \times (\sum_{j=1}^m x_{ij} + 1) \leq \mu, i = \overline{1,l_1}, \frac{A[k]}{2} \times (\sum_{j=1}^m y_{kj} + 1) \leq \xi, -\frac{A[k]}{2} \times (\sum_{j=1}^m y_{kj} + 1) \leq \mu, k = \overline{1,l_2} \Bigg\}. \tag{2.34}$$

The sequence $\{(x^l, y^l, u^l, v^l)\}$ generated by DCAEP+ enjoys the following property: if the penalty parameter is large enough and the solution at an iteration is binary, then the solution remains binary for the succeeding iterations.

The convergence of DCAEP+ is stated as follows.

Theorem 9. *(Convergence properties of DCAEP+)*

(i) DCAEP+ generates a sequence $\{(x^l, y^l, u^l, v^l)\}$ contained in $V(\Delta)$ such that the sequence $\{f(x^l, y^l, u^l, v^l)\}$ is decreasing.

(ii) There exists t^* such that: if $t_l > t^*$ and (x^l, y^l, u^l, v^l) is binary, then (x^s, y^s, u^s, v^s) remains binary for all $s \geq l$.

(iii) The sequence $\{(x^l, y^l, u^l, v^l)\}$ converges to $(x^*, y^*, u^*, v^*) \in V(\Delta)$ after a finite number of iterations. The point (x^*, y^*, u^*, v^*) is a critical point of Problem (2.28)*. Moreover if $u_i^* \neq \frac{1}{2}, v_k^* \neq \frac{1}{2}$ for all $i = \overline{1, l_1}, k = \overline{1, l_2}$, then (x^*, y^*, u^*, v^*) is a local solution to* (2.28)*.*

Proof. (i) is consequence of DCA's convergence theorem for a generic standard DC program.

(ii) Let $(\xi^{l+1}, \mu^{l+1}, x^{l+1}, y^{l+1}, u^{l+1}, v^{l+1})$ be the optimal solution of the problem (2.34). We denote

$$\begin{aligned}\Xi := \{(\xi, \mu, x, y, u, v) : (x, y, u, v) \in \Delta, L'[i] \times (\sum_{j=1}^{m} x_{ij} + 1) \leq \xi,\\ -L'[i] \times (\sum_{j=1}^{m} x_{ij} + 1) \leq \mu, i = \overline{1, l_1}, \frac{A[k]}{2} \times (\sum_{j=1}^{m} y_{kj} + 1) \leq \xi,\\ -\frac{A[k]}{2} \times (\sum_{j=1}^{m} y_{kj} + 1) \leq \mu, k = \overline{1, l_2}\}\end{aligned}$$

the polyhedral convex set of (2.34). Since the objective of (2.34) is linear, for all $l \geq 0$, $(\xi^{l+1}, \mu^{l+1}, x^{l+1}, y^{l+1}, u^{l+1}, v^{l+1})$ belongs to $V(\Xi)$ - the vertex set of Ξ. We denote $\bar{V} = \{(x, y, u, v) : \exists \xi, \mu$ such that $(\xi, \mu, x, y, u, v) \in V(\Xi)\} \cup \{(x^0, y^0, u^0, v^0)\}$.

If $p(x, y, u, v) = p(x', y', u', v') \quad \forall (x, y, u, v), (x', y', u', v') \in \bar{V}$, the assertion is trivial. Otherwise, let $\tilde{\zeta}(x, y, u, v) = \zeta(x, y, u, v) + \sum_{i=1}^{l_1}(d_i - 1)u_i - \sum_{k=1}^{l_2} v_k$ and

$$t^* = \sup\left\{\frac{\tilde{\zeta}(x, y, u, v) - \tilde{\zeta}(x', y', u', v')}{p(x', y', u', v') - p(x, y, u, v)} : (x, y, u, v), (x', y', u', v') \in \bar{V}, \right.$$
$$\left. p(x, y, u, v) < p(x', y', u', v')\right\} > -\infty. \tag{2.35}$$

Since $V(\Xi)$ is finite, so is $\bar{V}$. Consequently, $t^* < +\infty$.

By using similar arguments to Theorem 8 in [74], we proceed as follows. Assume by contradiction, let $t_l > t^*$ and (x^l, y^l, u^l, v^l) is binary but $(x^{l+1}, y^{l+1}, u^{l+1}, v^{l+1})$ is not binary. Then $p(x^l, y^l, u^l, v^l) = 0$ and $p(x^{l+1}, y^{l+1}, u^{l+1}, v^{l+1}) > 0$, so $p(x^l, y^l, u^l, v^l) < p(x^{l+1}, y^{l+1}, u^{l+1}, v^{l+1})$. From (2.35),

$$t_l > t^* \geq \frac{\tilde{\zeta}(x^l, y^l, u^l, v^l) - \tilde{\zeta}(x^{l+1}, y^{l+1}, u^{l+1}, v^{l+1})}{p(x^{l+1}, y^{l+1}, u^{l+1}, v^{l+1}) - p(x^l, y^l, u^l, v^l)}, \tag{2.36}$$

equivalently,

$$\tilde{\zeta}(x^{l+1}, y^{l+1}, u^{l+1}, v^{l+1}) + t_l p(x^{l+1}, y^{l+1}, u^{l+1}, v^{l+1}) > \tilde{\zeta}(x^l, y^l, u^l, v^l)$$
$$+ t_l p(x^l, y^l, u^l, v^l). \tag{2.37}$$

However, since $(x^{l+1}, y^{l+1}, u^{l+1}, v^{l+1})$ is the optimal solution of (2.33), we have

$$\tilde{\zeta}(x^{l+1}, y^{l+1}, u^{l+1}, v^{l+1}) + t_l p(x^{l+1}, y^{l+1}, u^{l+1}, v^{l+1}) \leq \tilde{\zeta}(x^l, y^l, u^l, v^l)$$
$$+ t_l p(x^l, y^l, u^l, v^l), \tag{2.38}$$

which is a contradiction.

(iii) (2.28) is a polyhedral DC program. By applying the finite convergence of DCA for polyhedral DC programs [71], DCAEP+ has a finite convergence, say, the sequence $\{(x^l, y^l, u^l, v^l)\}$ has a subsequence that converges to a critical point $(x^*, y^*, u^*, v^*) \in V(\Delta)$ after a finite number of iterations.

Moreover if $u_i^* \neq \frac{1}{2}, v_k^* \neq \frac{1}{2}$ for all $i = \overline{1, l_1}, k = \overline{1, l_2}$, then the function h is differentiable at (x^*, y^*, u^*, v^*). Therefore the necessary local condition

$$\emptyset \neq \partial h(x^*, y^*, u^*, v^*) \subset \partial g(x^*, y^*, u^*, v^*)$$

holds. Since h is a polyhedral convex function and $(x^*, y^*, u^*, v^*) \in \mathcal{P}_l = \{(x, y, u, v) \in \Delta : \partial h(x, y, u, v) \subset \partial g(x, y, u, v)\}$, then (x^*, y^*, u^*, v^*) is a local minimizer of (2.28) [71]. The proof is then complete.

□

2.7 Numerical experiments

To study the performance of our approaches, we perform them on several full binary trees with different configurations.

2.7.1 Dataset

We construct randomly full binary trees with the height of $8, 9, 10, 11, 12, 13$ and the balance of 5, 4, 4, 5, 5, 3, respectively. First, we generate a fixed number of random integers in the appropriate value range. Second, we build a full binary tree with the given height based on that set of values. Therefore, the constructed trees have different configurations. A certain number of leaving members is generated randomly among a set of leaf nodes in the binary key tree.

2.7.2 Comparative algorithms

We will compare our optimization approaches with some heuristic-based schemes that were proposed in [50], [65], and [93]. We use the following notations: D denotes the number of departing members, J denotes the number of joining members, and H denotes the height of the key tree.

In [50], the authors presented a very simple Marking algorithm that updates the key tree and generates a rekey subtree after each rekey interval. In this algorithm, there are four distinct cases that need consideration. If $J = D$, then every departing member is replaced by a new member. If $J < D$, the J shallowest leaf nodes of the departing members are selected and replaced with those of the joining members. In their terminology, "shallowest node" refers to the leaf node at the lowest height. In the case where $J > D$ and $D = 0$, the algorithm selects and removes the leaf node with the shallowest depth. This leaf node, together with the joining members, combine to create a novel key tree, which is then put in the original position of the least deep leaf node. Subsequently, in the scenario where $J > D$ and $D > 0$, it follows that the whole of the leaving individuals are substituted by the incoming individuals. From these replacements, the shallowest leaf node is chosen and deleted from the key tree. This leaf

node and the additional joining nodes comprise a new key tree, which is then inserted in place of the removed leaf node. The key server then generates and distributes the essential keys to the members.

In [65], two Merging algorithms were introduced as feasible options for performing batch joining of events. In order to accommodate batch departure requests more effectively, the two Merging algorithms have been expanded into a Batch Balanced algorithm. The performance of the Batch Balanced algorithm outperforms that of current algorithms in scenarios where the number of joining members exceeds the number of leaving members. Additionally, it demonstrates superior performance when the number of departing members is almost equal to N/d, where N represents the total number of group members and d represents the degree of the key tree without any joining members. In situations where the number of members entering a group and the number of members leaving the group are about equal, the Batch Balanced algorithm exhibits a similar performance to that of current methodologies.

In [93], Vijayakumar et al. presented Rotation based key tree algorithm to make the tree balanced. One limitation of their approach is that it works better for batch leave operations than batch join operations. When the batch join operations are greater than the batch leave operations, the performance is degraded. In this case, find the insertion point to insert new nodes. It is the shallowest leaf node from the left or right subtree, where the insertion of new nodes does not increase the height of the key tree. When $J = D$, replace all the leaving nodes with joining nodes. When $J < D$ and $J > 0$, find all the nodes where the leaving operation is going to take place. They pick J shallowest nodes of the D leaving nodes in the key tree. These selected shallowest nodes are replaced by the newly joining members. The remaining departure nodes are simply deleted from the key tree.

2.7.3 Set up experiments and Parameters

The optimization approaches were implemented in the Matlab R2017b, the comparative algorithms were implemented in the Python 3.8, and performed on a PC Intel i7 $-$ 7700 CPU, 3.60GHz of 16GB RAM. CPLEX 12.6 was used for solving linear programs. We stop the DCA scheme with the tolerance $\epsilon = 10^{-5}$.

Concerning the parameter t, as t_0 is hard to compute, we take a quite large value t_0 at the beginning and use an adaptive procedure for updating t during our scheme.

2.7.4 Comparative results

Compare Algorithm 4 with Algorithm 6

We compare the insertion cost, balance, and running time of DACEP1 for the first optimization model and DCAEP2 for the two-step algorithm to the problem of the updating group key in the LKH structure (refer to Table 2.1). The insertion cost is the number of updated keys in actuality after inserting new members. In our algorithms, it is calculated by the number of updated keys based on the approximation cost function minus the overlap keys. Moreover, the balance is measured by the difference of the distance from the root to the deepest leaf node and the shallowest one. The time ratio is Time 1 divided by Time 2.

Comments on numerical results:

In our experiments, when setting the initial set of chosen leaf nodes (to append new members) the same for two algorithms, we obtain identical solutions' qualities (insertion cost and balance). However,

Table 2.1: Comparison between DCAEP1 and DCAEP2 in terms of insertion cost, balance, and running time.

No	H	J	Cost		Balance		Total time (s)		Time ratio
			Cost1	Cost2	Balance1	Balance2	Time1	Time2	
1	8	50	227	227	3	3	0.2433	0.1475	1.6
2	8	100	352	352	3	3	0.4051	0.1744	2.3
3	8	200	575	575	4	4	0.7371	0.3385	2.2
4	8	300	785	785	4	4	1.0824	0.3975	2.7
5	8	400	991	991	4	4	1.5090	0.5062	3.0
6	8	500	1192	1192	4	4	2.4876	0.6380	3.9
7	9	200	716	716	4	4	1.6521	0.4974	3.3
8	9	300	937	937	4	4	2.6066	0.7257	3.6
9	9	400	1160	1160	4	4	3.6804	0.9813	3.8
10	9	600	1584	1584	4	4	5.3517	1.5684	3.4
11	9	800	1989	1989	4	4	7.1779	2.0982	3.4
12	9	1000	2391	2391	4	4	9.0751	2.6864	3.4
13	10	500	1679	1679	4	4	13.719	2.8754	4.8
14	10	1000	2771	2771	5	5	28.728	6.0403	4.8
15	10	1500	3790	3790	5	5	42.606	10.267	4.1
16	10	2000	4796	4796	4	4	66.562	13.392	5.0
17	10	2500	5796	5796	5	5	80.241	18.589	4.3
18	10	3000	6798	6798	5	5	94.436	20.459	4.6
19	11	500	2054	2054	4	4	50.262	7.3267	6.9
20	11	1000	3359	3359	4	4	98.652	15.122	6.5
21	11	1500	4489	4489	5	5	152.01	23.365	6.5
22	11	2000	5535	5535	5	5	203.99	31.053	6.6
23	11	2500	6576	6576	5	5	245.76	39.217	6.3
24	11	3000	7591	7591	5	5	293.07	47.586	6.2
25	12	1000	4145	4145	4	4	334.90	46.419	7.2
26	12	2000	6711	6711	5	5	714.18	95.727	7.5
27	12	4000	11125	11125	5	5	1445.6	204.73	7.1
28	12	6000	15230	15230	5	5	2240.7	326.72	6.9
29	12	8000	19246	19246	6	6	2787.8	423.26	6.9
30	12	10000	23261	23261	6	6	3431.8	555.64	6.2
31	13	2000	8301	8301	5	5	2921.5	281.98	10.4
32	13	4000	13489	13489	5	5	5979.1	603.95	9.9
33	13	6000	17974	17974	5	5	17353.5	1718.2	10.1
34	13	8000	22267	22267	5	5	43811.6	4295.3	10.2
35	13	9000	24346	24346	5	5	63140.4	6442.9	9.8
36	13	10000	26379	26379	6	6	119451.2	11597.2	10.3

the first model is more complex than the second one of two-step algorithm defined on only a set of leaf nodes. The experimental results prove that DCAEP2 is more efficient than DCAEP1 in terms of running time due to its lower computational complexity.

Regarding the running time, DCAEP2 runs approximately $2.6, 3.5, 4.6, 6.5, 7.0, 10.1$ times faster than DCAEP1 in cases where the height of the tree is $8, 9, 10, 11, 12, 13$, respectively.

Compare Algorithm 6 with Algorithm 7

Table 2.2: Comparison between DCAEP+ and DCAEP2 in terms of rekeying cost, balance (BL), and running time (s).

No	H	D	J	DCAEP+			DCAEP2				
				Cost	BL	Time	Insert cost	Delete cost	Cost	BL	Time
1	8	100	30	277	3	0.19	118	255	373	3	0.18
2	8	100	50	314	3	0.24	179	255	434	4	0.22
3	8	100	100	407	4	0.26	293	255	548	4	0.26
4	8	100	200	595	4	0.34	504	255	759	4	0.41
5	8	100	300	789	4	0.45	703	255	958	5	0.49
6	8	100	400	983	4	0.56	904	255	1159	5	0.64
7	8	100	500	1183	4	0.76	1104	255	1359	5	0.82
8	9	300	200	904	3	0.91	593	661	1254	4	0.52
9	9	300	300	1073	3	0.56	804	661	1465	4	0.63
10	9	300	400	1304	5	1.52	1007	661	1668	5	0.76
11	9	300	600	1689	5	1.53	1412	661	2073	5	1.07
12	9	300	800	2065	5	1.99	1813	661	2474	5	1.34
13	9	300	1000	2455	5	2.38	2213	661	2874	5	1.61
14	10	700	500	2103	3	5.94	1423	1478	2901	3	3.61
15	10	700	1000	3023	6	10.64	2447	1478	3925	6	6.53
16	10	700	1500	3942	5	13.27	3451	1478	4929	5	10.79
17	10	700	2000	4914	5	20.85	4452	1478	5930	5	13.30
18	10	700	2500	5881	6	26.56	5452	1478	6930	6	19.01
19	10	700	3000	6872	6	31.70	6452	1478	7930	6	20.73
20	11	1000	500	3228	4	10.98	1770	2465	4235	4	8.93
21	11	1000	1000	4054	4	18.52	2903	2465	5368	4	16.98
22	11	1000	1500	4910	6	26.49	3944	2465	6490	6	25.53
23	11	1000	2000	6271	6	33.88	4953	2465	7418	6	33.33
24	11	1000	2500	7231	6	40.16	5961	2465	8426	6	40.76
25	11	1000	3000	8180	7	49.36	6962	2465	9427	7	48.67
26	12	3000	2000	8782	4	117.6	5755	6215	11970	4	106.7
27	12	3000	4000	12213	7	228.9	9859	6215	16074	7	209.2
28	12	3000	5000	14061	6	271.2	11866	6215	18081	6	265.6
29	12	3000	6000	15867	6	361.2	13868	6215	20083	6	334.7
30	12	3000	8000	19664	7	441.1	17872	6215	24087	7	436.8
31	12	3000	10000	23571	7	583.9	21872	6215	28087	7	561.9
32	13	4000	2000	13820	4	366.9	7252	10039	17291	4	312.9
33	13	4000	4000	17738	5	650.3	11818	10039	21857	5	606.4
34	13	4000	6000	21567	5	1649	16031	10039	26070	6	1569
35	13	4000	8000	25373	5	4257	20883	10039	30922	5	4135
36	13	4000	9000	27283	5	6351	22100	10039	32139	5	6227
37	13	4000	10000	29352	5	11628	24109	10039	34148	5	10835

We compare the performance of DCAEP+ (Algorithm 7) with DCAEP2 (Algorithm 6) in terms of the following three criteria: the rekeying cost (insertion cost and deletion cost), the balance of the tree after rekeying and the running time of the algorithm. The rekeying cost is the number of updated keys in actuality after deleting departure members and inserting new members. In our algorithms, it is calculated by the number of updated keys based on the approximation cost function minus the overlap

keys. The rekeying cost of DCAEP2 is calculated as the total cost of individual deletion and batch insertion. Moreover, the balance is measured by the difference of the distance from the root to the deepest leaf node and the shallowest one. The running time of DCAEP2 is measured as the total execution time of processing batch insertion and individual deletion. The result is summarized in Table 2.2.

Comments on numerical results:

- DCAEP+ has a lower rekeying cost than DCAEP2 in all cases. The cost of DCAEP+ on average is less than 18.6%, 19.6%, 17.9%, $18.0\%, 20.5\%$, and 16.8% of DCAEP2 in cases where the height of the tree is $8, 9, 10, 11, 12$, and 13, respectively.

- The balance of the key tree is the same for both algorithms.

- The running time of DCAEP+ and DCAEP2 is the same in the case of a tree whose height is $8, 11, 12, 13$. DCAEP2 runs approximately 1.5 times faster than DCAEP+ in cases where the tree's height is either 9 or 10.

Compare with other existing algorithms

Table 2.3: Comparison results in the case where the binary tree has height $= 8$, balance $= 5$, and the number of departure members $= 100$.

No	J	DCAEP+		DCAEP2		Rotation		Marking		Merging	
		Cost	Balance	Cost	Balance	Cost	Balance	Cost	Balance	Cost	Balance
1	30	277	3	373	3	699	5	255	5	258	5
2	50	314	3	434	4	699	5	255	5	298	5
3	100	407	4	548	4	699	5	255	5	398	5
4	150	496	4	656	4	827	1	354	3	498	6
5	180	570	3	717	4	950	1	414	4	558	8
6	200	595	4	759	4	997	1	454	4	598	8
7	220	639	4	798	4	1037	1	494	4	638	8
8	250	693	4	859	5	1097	1	554	5	698	10
9	260	711	4	877	5	1125	1	574	5	718	10
10	270	734	4	899	5	1145	1	594	5	738	10
11	280	746	4	918	5	1173	1	614	5	758	10
12	300	789	4	958	5	1229	1	654	5	798	10
13	320	829	4	999	5	1301	1	694	5	838	10
14	350	887	4	1059	5	1369	1	754	5	898	10
15	370	921	4	1099	5	1417	1	794	6	938	12
16	380	946	4	1119	5	1437	1	814	6	958	12
17	400	983	4	1159	5	1477	1	854	6	998	12
18	420	1019	4	1199	5	1517	1	894	6	1038	12
19	450	1085	4	1259	5	1577	1	954	6	1098	12
20	480	1142	4	1319	5	1637	1	1014	6	1158	12
21	500	1183	4	1359	5	1677	1	1054	6	1198	12

The three mentioned heuristic-based algorithms give explicit solutions for how to delete leaving members and insert new members into the key tree, so the running time is insignificant. On the other hand, we compare the performance of DCAEP+ and DCAEP2 with other existing schemes in terms of the following two criteria: the rekeying cost (insertion cost and deletion cost) and the balance of the tree after rekeying.

Table 2.4: Comparison results in the case where the binary tree has height $= 9$, balance $= 4$, and the number of departure members $= 300$.

No	J	DCAEP+		DCAEP2		Rotation		Marking		Merging	
		Cost	Balance	Cost	Balance	Cost	Balance	Cost	Balance	Cost	Balance
1	200	904	3	1254	4	1959	4	661	4	998	4
2	250	986	3	1361	4	1959	4	661	4	1098	4
3	300	1073	3	1465	4	1959	4	661	4	1198	4
4	350	1165	4	1568	5	2090	1	760	5	1298	5
5	400	1304	5	1668	5	2214	1	860	6	1398	5
6	450	1406	5	1772	5	2354	1	960	7	1498	6
7	500	1500	5	1872	5	2454	1	1060	7	1598	6
8	550	1600	5	1974	5	2554	1	1160	7	1698	6
9	600	1689	5	2073	5	2672	1	1260	8	1798	8
10	650	1773	5	2173	5	2772	1	1360	8	1898	8
11	680	1839	5	2232	5	2832	1	1420	8	1958	8
12	700	1868	5	2273	5	2872	1	1460	8	1998	8
13	750	1955	6	2373	5	2972	1	1560	8	2098	8
14	780	2018	5	2434	5	3041	1	1620	8	2158	8
15	800	2065	5	2474	5	3081	1	1660	8	2198	8
16	850	2155	6	2573	5	3190	1	1760	9	2298	10
17	900	2251	5	2674	5	3308	1	1860	9	2398	10
18	950	2352	5	2774	5	3408	1	1960	9	2498	10
19	980	2413	5	2834	5	3468	1	2020	9	2558	10
20	1000	2455	5	2874	5	3508	1	2060	9	2598	10

Table 2.5: Comparison results in the case where the binary tree has height $= 10$, balance $= 4$, and the number of departure members $= 700$.

No	J	DCAEP+		DCAEP2		Rotation		Marking		Merging	
		Cost	Balance	Cost	Balance	Cost	Balance	Cost	Balance	Cost	Balance
1	400	1960	3	2675	3	4927	4	1478	4	2198	4
2	500	2103	3	2901	3	4927	4	1478	4	2398	4
3	600	2268	3	3109	3	4927	4	1478	4	2598	4
4	700	2435	3	3319	3	4927	4	1478	4	2798	4
5	800	2689	5	3520	5	3596	1	1677	6	2998	5
6	900	2837	6	3728	6	3823	1	1877	7	3198	5
7	1000	3023	6	3925	6	4059	1	2077	8	3398	6
8	1200	3400	5	4329	5	4459	1	2477	8	3798	6
9	1400	3765	5	4729	5	4859	1	2877	9	4198	8
10	1500	3942	5	4929	5	5069	1	3077	9	4398	8
11	1600	4141	5	5129	5	5279	1	3277	9	4598	8
12	1800	4529	6	5530	6	5679	1	3677	10	4998	10
13	2000	4914	5	5930	5	6079	1	4077	10	5398	10
14	2200	5291	5	6330	5	6479	1	4477	10	5798	10
15	2400	5692	6	6730	6	6879	1	4877	10	6198	10
16	2500	5881	6	6930	6	7079	1	5077	10	6398	10
17	2700	6280	6	7330	6	7479	1	5477	10	6798	10
18	2800	6473	6	7530	6	7679	1	5677	11	6998	12
19	2900	6675	6	7730	6	7879	1	5877	11	7198	12
20	3000	6872	6	7930	6	8079	1	6077	11	7398	12

Table 2.6: Comparison results in the case where the binary tree has height $= 11$, balance $= 5$, and the number of departure members $= 1000$.

No	J	DCAEP+		DCAEP2		Rotation		Marking		Merging	
		Cost	Balance	Cost	Balance	Cost	Balance	Cost	Balance	Cost	Balance
1	500	3228	4	4235	4	8865	5	2465	5	2998	5
2	700	3550	4	4697	4	8865	5	2465	5	3398	5
3	800	3701	4	4919	4	8865	5	2465	5	3598	5
4	1000	4054	4	5368	4	8865	5	2465	5	3998	5
5	1200	4383	6	5790	6	6203	1	2864	7	4398	6
6	1500	4910	6	6490	6	7383	1	3464	8	4998	6
7	1600	5459	7	6690	7	7616	1	3664	9	5198	8
8	1800	5844	7	7015	7	8060	1	4064	9	5598	8
9	2000	6271	6	7418	6	8592	1	4464	9	5998	8
10	2200	6634	6	7824	6	9058	1	4864	10	6398	10
11	2300	6827	6	8021	6	9346	1	5064	10	6598	10
12	2500	7231	6	8426	6	9845	1	5464	10	6998	10
13	2800	7832	6	9026	6	10555	1	6064	10	7598	10
14	2900	7963	7	9227	7	10832	1	6264	10	7798	10
15	3000	8180	7	9427	7	11043	1	6464	10	7998	10
16	3200	8590	6	9827	6	11487	1	6864	11	8398	12
17	3500	9189	6	10427	6	12230	1	7464	11	8998	12
18	3800	9756	7	11028	7	12841	1	8064	11	9598	12
19	3900	9958	7	11228	7	13041	1	8264	11	9798	12
20	4000	10167	7	11427	7	13241	1	8464	11	9998	12

The specific test scenario is executed as follows: Deleting a certain number of leaving members and inserting several numbers of new members in full binary trees that have different heights according to the chosen algorithm. We compare the performance of DCAEP+ and DCAEP2 with three existing schemes: the Rotation algorithm, the Marking algorithm, and the Merging algorithm. The rekeying cost and balance of the original trees that have the heights of $8, 9, 10, 11$, and 12, and the balance of $5, 4, 4, 5, 5$ are summarized in Tables 2.3–2.7, respectively.

Comments on numerical results:

- In terms of updating key cost, DCAEP+ results in a lower rekeying cost than the Rotation algorithm in all cases. The cost of DCAEP+ on average is less than 36.3%, 36.4%, 25.4%, 32.1%, and 32.6% of the Rotation algorithm corresponding to the height of the tree at $8, 9, 10, 11$, and 12, respectively. For cases where the number of new members is greater than the number of leaving members $(J > D)$, DCAEP+ has a rekeying cost that is lower than the Merging algorithm. While the Marking algorithm gives the lowest updating key cost, it cannot maintain the tree balance after deletion and insertion.

DCAEP2 also has a lower cost than the Rotation algorithm, namely 22.8%, 21.9%, 8.7%, 19.6%, and 15.1% less for trees with heights of $8, 9, 10, 11$, and 12, respectively. The rekeying cost of DCAEP2 is slightly higher than that of the Marking and Merging algorithms.

- As for the balance of the key tree after deletion and insertion, DCAEP+ always keeps this factor lower than the Marking algorithm. In particular, the balance of our approach is less than 24.3%, 34.5%, 39.5%, 32.8%, and 44.8% of the Marking algorithm, where the tree has a height of $8, 9, 10, 11$, and 12, respectively. DCAEP+ is more efficient in terms of balance than the Merging algorithm in the

Table 2.7: Comparison results in the case where the binary tree has height $= 12$, balance $= 5$, and the number of departure members $= 3000$.

No	J	DCAEP+		DCAEP2		Rotation		Marking		Merging	
		Cost	Balance	Cost	Balance	Cost	Balance	Cost	Balance	Cost	Balance
1	2000	8782	4	11970	4	24258	5	6215	5	9998	5
2	2500	9561	4	13023	4	24258	5	6215	5	10998	5
3	3000	10375	4	14043	4	24258	5	6215	5	11998	5
4	3500	11193	5	15064	5	17039	1	7214	8	12998	6
5	4000	12213	7	16074	7	18182	1	8214	9	13998	6
6	4200	12585	7	16472	7	18582	1	8614	10	14398	8
7	4500	13131	6	17081	6	19182	1	9214	10	14998	8
8	4800	13648	7	17684	7	19782	1	9814	10	15598	8
9	5000	14061	6	18081	6	20182	1	10214	10	15998	8
10	5500	14929	7	19082	7	21182	1	11214	11	16998	10
11	5800	15510	6	19684	6	21818	1	11814	11	17598	10
12	6000	15867	6	20083	6	22230	1	12214	11	17998	10
13	6500	16803	7	21084	7	23230	1	13214	11	18998	10
14	7000	17771	6	22087	6	24242	1	14214	11	19998	10
15	7500	18760	6	23086	6	25254	1	15214	12	20998	12
16	8000	19664	7	24087	7	26266	1	16214	12	21998	12
17	8500	20650	6	25087	6	27266	1	17214	12	22998	12
18	9000	21595	7	26087	7	28266	1	18214	12	23998	12
19	9500	22575	7	27086	7	29266	1	19214	12	24998	12
20	10000	23571	7	28087	7	30266	1	20214	12	25998	12

case of a binary tree with a height of 8. In these instances, DCAEP+ has an average balance of 5 less than 59.3% of the Merging algorithm's. In the case of a binary tree with a height of $9, 10, 11, 12$, where the number of new members is greater than or equal to $450, 1200, 1600, 4200$ and greater than the number of departure members (on $15, 13, 14, 15/20$ instances), our approach DCAEP+ is more efficient in terms of balance than the Merging algorithm. In these instances, DCAEP+ has an average balance of $5.1, 5.5, 6.5, 6.5$, less than $37.9\%, 42.9\%, 36.8\%, 36.4\%$ of the Merging algorithm's where the tree has a height of $9, 10, 11$ and 12, respectively. The Marking and Merging algorithms keep the balance of the tree after rekeying the same as the original tree when the number of members joining is equal to or less than the number of those departing. Meanwhile, our algorithm improves the balance of the tree after updating the group key in these cases. Moreover, DCAEP+ and DCAEP2 keep the same balance of the key tree after rekeying.

Overall, our proposed approaches simultaneously take into account both objectives: the rekeying cost and the balance of the tree. It is efficient in both scenarios when the number of leaving members exceeds the number of new ones and the number of new members is greater than the number of departing ones.

2.8 Conclusion

In this chapter, we proposed optimization approaches to the problem of updating group key in the LKH structure with two different rekeying techniques, namely batch insertion and batch rekeying. First, we

opt for the batch insertion technique to maintain the forward secrecy of the group key update problem. This is the first optimization model that considers simultaneously the insertion cost and the balance of the resulting key tree. The suggested optimization problem has binary variables and an objective function that is discontinuous. It is first equivalently formulated to eliminate the objective's step functions. By using recent results on exact penalty techniques in DC programming, the latter problem can be reformulated as a DC program. We then designed an efficient DCA to solve this problem. Furthermore, we proposed a more effective two-step algorithm that integrated an algorithm to find all leaf nodes in the first step, which simplifies the optimization model in the second step. This simplified problem can still be solved by the DCA-based approach, as for the original problem.

In addition, we proposed an optimization approach to the problem of batch deletion and insertion members in CGKM. Our primary objective is to simultaneously minimize the cost of removing and adding nodes while maintaining a balanced tree. The suggested optimization problem has binary variables and an objective function that is discontinuous. Numerical experiments have been conducted to justify the merits of our proposed model as well as the corresponding DCA. It has been shown that our approach obtains a better trade-off between two considered criteria in comparison with existing approaches. In other words, our method allows for a good compromise between the rekeying cost, which is the cost of deleting and inserting members, and the tree's balance after rekeying.

Chapter 3

Advanced DCA based approaches for optimizing Merkle tree structure in blockchain transaction system

In this chapter, we take advantage of typical transaction characteristics to better construct the Merkle tree to improve blockchain network performance. It consists of identifying a tree structure with the minimum number of hash values required to update the account data associated with each transaction based on the distribution of all transactions. The proposed optimization model is a binary quadratic program, which is very hard to solve. By using the exact penalty techniques, we reformulate the problem as a conventional DC program that is efficiently solvable by the DCA. To get a better convex approximation of the objective function without knowing a DC decomposition, DCA-Like, a novel extension of DCA, is applied. Furthermore, we deploy Accelerated DCA (ADCA) and Accelerated DCA-Like (ADCA-Like) to improve DCA and DCA-Like by incorporating Nesterov's acceleration technique into them. In addition, we alternately combine DCA and other advanced variants of DCA (ADCA, DCA-Like, and ADCA-Like) with the divide-and-conquer algorithm to build a Merkle tree for a large number of blockchain accounts. Numerical experiments on several datasets illustrate the efficiency of our approaches.

The results presented in this chapter were published/submitted in:
T. T. T. Nguyen, H. A. Le Thi, & X.V. Doan (2023). Optimizing Merkle Tree Structure for Blockchain Transactions by a DC Programming Approach. In: *Nguyen, N.T., et al. Computational Collective Intelligence. ICCCI 2023. Lecture Notes in Computer Science()*, vol 14162. Springer, Cham.
T. T. T. Nguyen, & H. A. Le Thi (2023). A DCA-Like based algorithm for the Merkle tree construction problem in Ethereum cryptocurrency system. In: *Proceedings of the 4th International Conference and Summer School on Numerical Computations: Theory and Algorithms NUMTA 2023.* **Accepted for publication**.

3.1 Introduction

In recent years, blockchain technology has attracted considerable interest as a decentralized, secure ledger with no central authority. It first gained popularity as the technology behind Bitcoin [63], an innovative cryptocurrency, and has subsequently been expanded to include electronic voting, supply chain communications, and medical informatics [12, 24, 33]. It is impossible to change a block without also modifying all subsequent blocks, as each block after the original (genesis) block carries the result of a cryptographic hash function computed on the content of the previous block.

The Merkle tree is a well-known tool in cryptography, first suggested by Ralph Merkle [58], which enables efficiently verifying the validation of a data element in a set without revealing the entire set. Additionally, it guarantees the integrity of data stored in this kind of data structure. In a Merkle tree, every node is assigned a hash value. The label of a leaf node corresponds to the hash value of a specific data item. At the same time, the label of a non-leaf node is derived by concatenating the labels of their respective child nodes. The hash value ensures the integrity and immutability of the transaction which is used as the input of hash function. To ensure the inclusion of certain data in a Merkle tree, it is necessary for every node to acquire a label denoted as R for the root of the tree, often referred to as the Merkle root, from a reliable and trustworthy source. A Merkle proof is a cryptographic technique used to verify the inclusion of a certain data element, denoted as x, inside a Merkle tree. This proof comprises the sibling path of the leaf node containing x, and it contains the labels associated with the sibling nodes along the path from the leaf to the root of the tree.

Merkle trees are critical components of blockchain technology, as they enable the secure verification of transactions. The State Merkle tree in Ethereum, the second-largest blockchain network, contains the current balance and other relevant data for each account. Each block in the chain is associated with the Merkle root computed for the network state following the execution of the block transactions. The block approval process includes verifying the validity of transactions and their impact on the state. For instance, a payment requires a minimum value of the payer's balance, and when successful, it implies a change in the balance of the two involved accounts. Indeed, the data from two accounts and the corresponding nodes on the Merkle tree need to be recalculated.

In actuality, the distribution of accounts involved in transactions is not uniform. The majority of accounts transfer value to a small number of other accounts, while a minority of accounts are linked to several others. Similarly, when a transaction is seen as a collection of accessible addresses, a small number of unique transactions account for a significant share of the transactions. The transaction frequency is characterized by a few transactions being repeated several times, whereas the majority of transactions occur just once. Furthermore, as frequently used accounts move closer to the root of the State tree due to overlapping paths, fewer tree nodes must be read and updated.

Different heuristic-based approaches have been used to build a Merkle hash tree for a specific transaction distribution in [59, 62, 97]. The algorithms differ in several aspects, including (i) the codeword lengths they generate (fixed or variable length) and (ii) the data they rely on, such as the complete distribution of transactions or only the distribution of accounts. The first random algorithm allocated fixed-length codewords of the minimal possible length of $\lceil \log_2(n) \rceil$ bits for all n accounts [97]. Therefore, every account was presented at the same tree height. The second approach, which was based on the Huffman-Merkle Hash Tree (HuffMHT), considered the account distribution inferred by the transaction distribution input [62]. Based on the probability of each account, the method assigned variable-length

codewords as Huffman codes. In [59], the authors proposed the method of building a Merkle tree based on the account's information in blockchain networks to reduce the number of required hash values used in proofs for data membership. They attempted to modify the Huffman algorithm to consider the distribution of transactions rather than just accounts. However, they have not constructed a mathematical optimization model for this transaction encoding problem. Additionally, their algorithms are primarily based on logical/heuristic arguments.

Our contributions. In this chapter, we propose an optimization model for constructing the Merkle tree based on the transaction distribution in Ethereum system [66]. The objective is to minimize the number of changed hash values in the Merkle tree required for updating the information of accounts involved in each transaction. To our knowledge, this is the first work introducing an optimization model that constructs the Merkle tree based on the transaction distribution. The proposed optimization model is a binary quadratic program. Consequently, it is very challenging to handle such kinds of programs by standard methods where the source of difficulty comes from the nonconvexity of the objective and the binary nature of the solutions. By using the exact penalty techniques [44, 74, 38], we reformulate the problem as a DC program, where the DCA is at our disposal as an efficient algorithm in DC programming.

In practice, it is difficult to determine the exact value of parameter ρ in the objective function (refer to Problem (1.10)), and one usually estimates by a quite large number. However, this could lead to a bad convex approximation of the objective function, then DCA may converge rapidly to a biased critical point. Therefore, we deploy the DCA-Like, which relies on a novel and effective method to estimate the objective function of the DC programming without prior knowledge of its DC decomposition [41, 35]. DCA-Like may be considered similar to DCA since it involves the iterative approximation of the DC program via a sequence of convex programs. However, DCA-Like differs from DCA in its approach to approximating the objective function. In some cases, we may not possess a DC decomposition for the objective function. While the standard DCA involves using a convex majorization of the objective function throughout the whole space using an available DC decomposition, DCA-Like aims to improve the convex approximation of the existing solution by employing a decomposition that may not necessarily be DC. Therefore, DCA-Like is capable of functioning even in situations where it is not possible to identify and emphasize a DC decomposition. The primary purpose of DCA-Like is to keep the parameter ρ as small as possible while simultaneously seeking a convex approximation of the target function. DCA-Like relaxes the convexity criterion associated with the second DC component, resulting in a majorant of the objective function that is closer in proximity and potentially yielding an improved solution.

Moreover, we apply Accelerated DCA (ADCA) and ADCA-Like for solving the problem of constructing Merkle tree in Ethereum system. In ADCA [68, 35], the authors incorporated Nesterov's acceleration technique to enhance the effectiveness of standard DCA. ADCA differs from the conventional line search acceleration based on the Armijo type rule, which can be computationally expensive. The acceleration step in ADCA, which consists of using an extrapolated point from the current and previous iterations, seeks to locate a point z^k that is superior to x^k for the computation of x^{k+1}. In addition, Accelerated DCA-Like incorporates Nesterov's acceleration technique, which potentially leads to a better solution in comparison to DCA-Like [41, 35].

In fact, the large number of accounts makes the size of the coefficient matrix in the binary quadratic program too enormous (the size of the matrix increases exponentially according to the number of accounts), which causes the error of out-of-memory in the programming software. Therefore, our proposed

problem cannot be directly solved by any algorithm. To tackle this difficulty, we alternately combine the divide-and-conquer algorithm [49] with DCA, ADCA, DCA-Like, and ADCA-Like, referred to as recursive DCA, ADCA, DCA-Like, and ADCA-Like approaches, to solve the problem of building a Merkle tree for blockchain transactions when the number of accounts is significant. A divide-and-conquer algorithm is characterized by its recursive nature, as it systematically decomposes a given problem into several sub-problems that are either identical or closely related. This process continues until the sub-problems reach a level of simplicity that allows for direct resolution. The solutions to the sub-problems are then integrated to provide a solution to the initial problem.

The remainder of this chapter is structured as follows. The optimization model for the problem of constructing the Merkle tree in Ethereum system is developed in Section 3.2. Section 3.3 presents the solution methods based on DCA and other advanced variants of DCA (ADCA, DCA-Like and ADCA-Like). The implementation of the algorithms for solving the problem and numerical experiments are presented in Section 3.4. Finally, some conclusions are provided in Section 3.5.

3.2 Optimization model for the problem of constructing Merkle tree in blockchain based system

3.2.1 Problem definition

According to [17, 88], the distribution of accounts engaging in transactions is not consistent, and its bias can be expressed in two major properties. In fact, the majority of accounts only transfer value to a small number of other accounts, whereas a small number of accounts are linked to a large number of others. As seen with Ethereum [17], these highly connected nodes, which are frequently exchanged or mining pool accounts, are significantly influencing the structure of the network. Similarly, if a transaction is defined as its collection of accessible addresses, a small number of unique transactions account for a substantial fraction of transactions. A subsequent study [88] found comparable findings for other blockchain networks. The duration of the measurement interval influences both distributions. Therefore, we make use of common transaction features to better organize Merkle trees and propose an optimization approach to solve this problem. Assume that a transaction has two distinct accounts, as in the common case of payment transactions. We investigate an optimization model for constructing Merkle trees associated with the transaction distribution. The objective is to minimize the number of hash values required for account information modification in each transaction.

3.2.2 The optimization model

Based on the above problem description, we propose an optimization model in this subsection. Given $A = \{1, 2, ..., n\}$ is the set of accounts and (TX, D) is the transaction distribution included the frequency of a transaction between two accounts. A full binary tree can be represented as an ordered set $\mathbb{T} = \{2^l + m$: there exists a node at the m-th horizontal position of level l on the tree, $0 \leq l \leq h, 0 \leq m \leq 2^l - 1, l, m \in \mathbb{N}\}$, where h is the height of tree, $T = |\mathbb{T}|$.

Let z_{it} be binary variables defined by $z_{it} = 1$ if an account $i \in A$ is added into node t of the Merkle tree, $z_{it} = 0$ otherwise, $\forall i = \overline{1, n}, t = \overline{1, T}, T = 2^n - 1$. Since every account is appended at only one node of the tree, $\sum_{t=1}^{T} z_{it} = 1, \forall i = \overline{1, n}$. The Merkle tree is a full binary tree in which every node has

0 or 2 children. If $z_{it} = 1$ then $z_{jt'} = 0, i = \overline{1,n}, j = \overline{1,n}$, for all $t^{'}$ belongs to the path from the node t to the root (index of the root is 1). Let $P(t)$, which is defined by Algorithm 8, be the set of all nodes on the path from the node t to the root.

Algorithm 8 Algorithm of finding all nodes on the path from the node t to the root

Initialization: $P(t) = [], t \in \mathbb{T}$.
while $t > 1$ **do**
  $t' = \left[\frac{t}{2}\right]$.
  $P(t)$.append(t').
  $t = t'$.
end while

The computability of a node in the tree is defined as follows: A node t is computable if and only if there exists a node t in the Merkle tree or both children of node t are computable. We define a binary variable c_t by $c_t = 1$ if there is a node t in the tree, $c_t = 0$ otherwise. Therefore, the following constraints ensure the computability of a node in the tree $\sum_{i=1}^{n} z_{it} \leq c_t, t = \overline{1,T}, (c_{2t} + c_{2t+1}) - 1 \leq c_t, t = \overline{1,T}, c_t \leq \sum_{i=1}^{n} z_{it} + \frac{1}{2}(c_{2t} + c_{2t+1}), t = \overline{1,T}$.

On the other hand, there exists node t on the tree, so the sibling node of t is also one node of this tree because the Merkle tree is a full binary tree. If $z_{it} = 1$ then $c_{s(t)} = 1$, where $s(t)$ is denoted as the sibling node of node t. Therefore, we obtain the constraint as $\sum_{i=1}^{n} z_{it} \leq c_{s(t)}$, where $s(t) = t - 1$ if t is odd, $s(t) = t + 1$ otherwise. The objective of the problem is to minimize the total cost for the transaction verification between two accounts. If two accounts are added into two leaf nodes in the Merkle tree as $\sum_{i=1}^{n} z_{it} = 1$ and $\sum_{j=1}^{n} z_{js} = 1, \forall t = \overline{1,T}, s = \overline{1,T}$, we can define the cost of validating a transaction between accounts i and j that is assigned to node t and s, respectively, as a value C_{ts}. The total cost for a transaction distribution D over n accounts is defined as $\sum_{t=1}^{T} \sum_{s=1}^{T} C_{ts} \bar{q}_{ts}$, where $\bar{q}_{ts}$ is the frequency of a transaction that involves two accounts i and j that are assigned to node t and s on the tree, $\bar{q}_{ts} = \sum_{i=1}^{n} \sum_{j=1}^{n} q_{ij} z_{it} z_{js}$. Finally, our optimization problem takes the following form:

$$\min \sum_{t=1}^{T} \sum_{s=1}^{T} C_{ts} \sum_{i=1}^{n} \sum_{j=1}^{n} q_{ij} z_{it} z_{js} \tag{3.1}$$

$$\text{subject to} \sum_{t=1}^{T} z_{it} = 1, \forall i = \overline{1,n}, \tag{3.2}$$

$$\sum_{i=1}^{n} z_{it} + \sum_{j=1}^{n} z_{jt'} \leq 1, t = \overline{1,T}, t' \in P(t) \tag{3.3}$$

$$\sum_{i=1}^{n} z_{it} \leq c_t, t = \overline{1,T}, \tag{3.4}$$

$$(c_{2t} + c_{2t+1}) - 1 \leq c_t, t = \overline{1,T}, \tag{3.5}$$

$$c_t \leq \sum_{i=1}^{n} z_{it} + \frac{1}{2}(c_{2t} + c_{2t+1}), t = \overline{1,T}, \tag{3.6}$$

$$\sum_{i=1}^{n} z_{it} \leq c_{s(t)}, t = \overline{1,T}, \tag{3.7}$$

$$z_{it} \in \{0,1\}, c_t \in \{0,1\}, \forall i = \overline{1,n}, t = \overline{1,T}. \tag{3.8}$$

It is observed that (3.1) is a binary quadratic program. Let

$$K = \Bigg\{ (z,c) : \sum_{t=1}^{T} z_{it} = 1, \sum_{i=1}^{n} z_{it} + \sum_{j=1}^{n} z_{jt'} \leq 1, \sum_{i=1}^{n} z_{it} \leq c_t, (c_{2t} + c_{2t+1}) - 1 \leq c_t,$$
$$c_t \leq \sum_{i=1}^{n} z_{it} + \frac{1}{2}(c_{2t} + c_{2t+1}), \sum_{i=1}^{n} z_{it} \leq c_{s(t)}, z_{it} \in [0,1], c_t \in [0,1], i = \overline{1,n}, t = \overline{1,T}, t' \in P(t) \Bigg\}.$$

Problem (3.1) can be rewritten as follows

$$\min \left\{ \sum_{t=1}^{T} \sum_{s=1}^{T} C_{ts} \sum_{i=1}^{n} \sum_{j=1}^{n} q_{ij} z_{it} z_{js} : (z,c) \in K, z \in \{0,1\}^{n \times T}, c \in \{0,1\}^{T} \right\}. \tag{3.9}$$

3.3 Solution methods based on DC programming and DCA for solving the problem (3.9)

In [44, 74, 38], a continuous approach based on DC programming and DCA was proposed to solve binary quadratic programs and this method is also available to solve the problem (3.9).

Then (3.9) can be rewritten as

$$\min \left\{ z^T Q z : (z,c) \in K, z \in \{0,1\}^{n \times T}, c \in \{0,1\}^{T} \right\}, \tag{3.10}$$

where Q is a matrix $((n \times T) \times (n \times T))$. According to [44, 74, 38], the problem (3.10) and the following problem are equivalent:

$$\min \left\{ \rho z^T e - z^T (\rho I - Q) z : (z,c) \in K, z \in \{0,1\}^{n \times T}, c \in \{0,1\}^{T} \right\}, \tag{3.11}$$

where $e \in \mathbb{R}^{n \times T}$ is the vector of ones, I is the identity matrix of order $n \times T$ and $\rho \geq \lambda(Q)$, the largest eigenvalue of matrix Q.

The concave quadratic function

$$p(z,c) := \sum_{i=1}^{n} \sum_{t=1}^{T} z_{it}(1 - z_{it}) + \sum_{t=1}^{T} c_t(1 - c_t) = z^T[e - z] + c^T[e - c],$$

can be used as exact penalty function for (3.11) [44, 74, 38]. Then (3.11) can be rewritten as

$$\min \left\{ \rho z^T e - z^T (\rho I - Q) z : (z,c) \in K, p(z,c) \leq 0 \right\}. \tag{3.12}$$

There exists $\bar{\tau}$ such that the problem (3.12) and the following penalized problem are equivalent for all $\tau \geq \bar{\tau}$:

$$\min \left\{ (\rho + \tau) z^T e + \tau c^T e - (z^T (\rho I - Q) z + \tau z^T z + \tau c^T c) : (z,c) \in K \right\}. \tag{3.13}$$

The following DC formulation of (3.13) seems to be natural:

$$\min\{ f(z,c) = g(z,c) - h(z,c) : (z,c) \in \mathbb{R}^{n \times T} \times \mathbb{R}^{T} \}, \tag{3.14}$$

where $g(z,c) := (\rho+\tau)z^Te + \tau c^Te + \chi_K(z,c)$, $h(z,c) := z^T(\rho I - Q)z + \tau z^Tz + \tau c^Tc$, are clearly convex functions.

3.3.1 DCA for solving the problem (3.14)

Applying the general DCA scheme to (3.14) amounts to computing two sequences $\{(z^l,c^l)\}$ and $\{(y^l,b^l)\}$ in the way that $(y^l,b^l) \in \partial h(z^l,c^l)$. Since $(y^l,b^l) \in \partial h(z^l,c^l)$ is equivalent to

$$y^l = 2(\rho+\tau-Q)z^l, \tag{3.15}$$

$$b^l = 2\tau c^l, \tag{3.16}$$

following is a description of the algorithm.

Algorithm 9 DCA for solving the problem (3.14)

Initialization: Choose $(z^0,c^0) \in [0,1]^{n\times T} \times [0,1]^T$ as an initial value, set $l := 0, \epsilon > 0, \rho > 0, \tau_l > 0, \theta > 0$.
repeat
Compute $(y^l,b^l) \in \partial h(z^l,c^l)$ via (3.15) and (3.16).
Solve the following linear program to obtain (z^{l+1},c^{l+1})

$$\min\{(\rho+\tau_l)z^Te + \tau_l c^Te - \langle z,y^l\rangle - \langle c,b^l\rangle : (z,c) \in K\}. \tag{3.17}$$

if $p(z^{l+1},c^{l+1}) > 0$ **then**
$\tau_{l+1} \leftarrow \tau_l + \theta$.
end if
$l = l+1$.
until $\|(z,c)^{l+1} - (z,c)^l\| \le \epsilon$

The convergence properties of Algorithm 9 are stated as follows [74].

Theorem 10. *For $\rho \ge \lambda(Q)$, the largest eigenvalue of matrix Q, there hold*

(i) Exists τ^* such that: if $\tau_l > \tau^*$ and (z^l,c^l) is binary, then (z^s,c^s) remains binary for all $s \ge l$.

(ii) Algorithm 9 has a finite convergence: the sequence $\{(z^l,c^l)\}$ has a subsequence that converges to a DC critical point of $f = g - h$ after a finite number of iterations.

3.3.2 Accelerated DCA for solving the problem (3.14)

In [68, 35], the authors proposed the Accelerated DCA (ADCA in short) for a special case of the standard DC program whose objective function is the sum of a differentiable function with $L-$Lipschitz continuous gradient and a DC function. The proposed algorithm consists in incorporating the Nesterov's acceleration technique into standard DCA. According to Algorithm 9, ADCA for solving the problem (3.14) is described in Algorithm 10.

Algorithm 10 ADCA for solving the problem (3.14)

Initialization: Choose an initial point $(z^0, c^0) \in [0,1]^{n\times T} \times [0,1]^T$, $(\alpha^0, \beta^0) = (z^0, c^0)$, $q \in \mathbb{N}$, $t_0 = (1+\sqrt{5})/2$, set $l := 0, \epsilon > 0, \rho > 0, \tau_l > 0, \theta > 0$.

repeat

If $f(\alpha^l, \beta^l) \leq \max_{t=\max(0,l-q),\ldots,l} f(z^t, c^t)$ then set $(u^l, v^l) = (\alpha^l, \beta^l)$, otherwise set $(u^l, v^l) = (z^l, c^l)$.

Compute $(y^l, b^l) \in \partial h(u^l, v^l)$ via (3.15) and (3.16).

Solve the following linear program to obtain (z^{l+1}, c^{l+1})

$$\min \{(\rho + \tau_l) z^T e + \tau_l c^T e - \langle z, y^l \rangle - \langle c, b^l \rangle : (z, c) \in K\}. \tag{3.18}$$

if $p(z^{l+1}, c^{l+1}) > 0$ **then**

$\tau_{l+1} \leftarrow \tau_l + \theta$.

end if

Compute $t_{l+1} = \frac{1+\sqrt{1+4t_l^2}}{2}$ and $\alpha^{l+1} = z^{l+1} + \frac{t_l - 1}{t_{l+1}}(z^{l+1} - z^l)$, $\beta^{l+1} = c^{l+1} + \frac{t_l - 1}{t_{l+1}}(c^{l+1} - c^l)$ if $l \geq 1$.

$l = l + 1$.

until $\|(z, c)^{l+1} - (z, c)^l\| \leq \epsilon$

Note that the convergence results of Algorithm 10 are guaranteed by Theorem 2.1 and Theorem 2.2 in [35].

3.3.3 DCA-Like for solving the problem (3.14)

In practice, it is difficult to determine the exact value of ρ, and one usually estimates ρ by a quite large value. Nevertheless, if the value of ρ is quite high, it might result in a bad convex approximation of f, hence causing DCA to converge quickly towards a biased critical point. To overcome this drawback and improve the standard DCA, we apply DCA-Like, which is a novel and effective way to approximate the DC objective function without knowledge of its DC decomposition [41, 35]. At each iteration l of DCA-Like, ρ_l is found such that $h^l_{\rho_l}(z, c) = h_{\rho_l}(z^l, c^l) + \langle (z, c) - (z^l, c^l), (y^l, b^l) \rangle$ is a lower bound of $h_{\rho_l}(z, c)$ at (z^{l+1}, c^{l+1}) but not on the whole space, i.e.,

$$h_{\rho_l}(z^{l+1}, c^{l+1}) \geq h^l_{\rho_l}(z^{l+1}, c^{l+1}), \tag{3.19}$$

with $(y^l, b^l) \in \partial h_{\rho_l}(z^l, c^l)$, and solves the following convex problem to obtain (z^{l+1}, c^{l+1})

$$\min\{ f^l_{\rho_l}(z, c) := g_{\rho_l}(z, c) - h^l_{\rho_l}(z, c) : (z, c) \in K\}. \tag{3.20}$$

Since $(y^l, b^l) \in \partial h_{\rho_l}(z^l, c^l)$ is equivalent to

$$y^l = 2(\rho_l + \tau_l - Q) z^l, \tag{3.21}$$

$$b^l = 2\tau_l c^l, \tag{3.22}$$

the algorithm can be described as follow.

Algorithm 11 DCA-Like for solving the problem (3.14)

Initialization: Choose $(z^0, c^0) \in [0,1]^{n\times T} \times [0,1]^T$ as an initial value, set $l := 0, \epsilon > 0$, a small enough positive parameter $\rho_0, \tau_0 > 0, \theta > 0, \eta > 1$ and $0 < \delta < 1$.
repeat
Set $\rho_l = \max\{\rho_0, \delta\rho_{l-1}\}$ if $l > 0$.
Compute (z^{l+1}, c^{l+1}) by solving the following linear program with $\rho = \rho_l$ and $(y^l, b^l) \in \partial h_\rho(z^l, c^l)$ via (3.21) and (3.22).

$$\min\{(\rho_l + \tau_l)z^T e + \tau_l c^T e - \langle z, y^l\rangle - \langle c, b^l\rangle : (z,c) \in K\}. \tag{3.23}$$

while $h_{\rho_l}(z^{l+1}, c^{l+1}) < h^l_{\rho_l}(z^{l+1}, c^{l+1})$ **do**
$\rho_l \leftarrow \eta\rho_l$.
Update (z^{l+1}, c^{l+1}) by solving (3.23) with $\rho = \rho_l$ and $(y^l, b^l) \in \partial h_\rho(z^l, c^l)$.
end while
if $p(z^{l+1}, c^{l+1}) > 0$ **then**
$\tau_{l+1} \leftarrow \tau_l + \theta$.
end if
$l = l + 1$.
until $\|(z,c)^{l+1} - (z,c)^l\| \le \epsilon$

The convergence properties of Algorithm 11 are stated as follows [35].

Theorem 11. *Let $\{(z^l, c^l)\}$ be the sequence generated by Algorithm 11. For $\rho_k \ge \|Q\|$, the following statements hold.*

i) Exists τ^* such that: if $\tau_l > \tau^*$ and (z^l, c^l) is binary, then (z^s, c^s) remains binary for all $s \ge l$.

ii) Algorithm 11 has a finite convergence: the sequence $\{(z^l, c^l)\}$ has a subsequence that converges to a DC critical point of $f = g - h$ after a finite number of iterations.

3.3.4 Accelerated DCA-Like for solving the problem (3.14)

Following the same idea of Accelerated DCA, we apply an accelerated version of DCA-Like (ADCA-Like) [41, 35], which is described in Algorithm 12. In theory, a higher value of the parameter q enhances the chance of using the extrapolated points (α^l, β^l) in ADCA, consequently increasing the probability of acceleration.

Algorithm 12 ADCA-Like for solving the problem (3.14)

Initialization: Choose an initial point $(z^0, c^0) \in [0,1]^{n\times T} \times [0,1]^T, (\alpha^0, \beta^0) = (z^0, c^0), q \in \mathbb{N}, t_0 = (1+\sqrt{5})/2$, set $l := 0, \epsilon > 0$, a small enough positive parameter $\rho_0, \tau_0 > 0, \theta > 0, \eta > 1$ and $0 < \delta < 1$.

repeat

If $f(\alpha^l, \beta^l) \leq \max_{t=\max(0,l-q),\ldots,l} f(z^t, c^t)$, then set $(u^l, v^l) = (\alpha^l, \beta^l)$, otherwise set $(u^l, v^l) = (z^l, c^l)$.

Compute $(y^l, b^l) \in \partial h_{\rho_l}(u^l, v^l)$ via (3.21) and (3.22).

Set $\rho_l = \max\{\rho_0, \delta\rho_{l-1}\}$ if $l > 0$.

Compute (z^{l+1}, c^{l+1}) by solving the following linear program with $\rho = \rho_l$

$$\min \{(\rho_l + \tau_l)z^T e + \tau_l c^T e - \langle z, y^l\rangle - \langle c, b^l\rangle : (z,c) \in K\}. \tag{3.24}$$

while $h_{\rho_l}(z^{l+1}, c^{l+1}) < h^l_{\rho_l}(z^{l+1}, c^{l+1})$ **do**

$\rho_l \leftarrow \eta\rho_l$.

Update (z^{l+1}, c^{l+1}) by solving (3.24) with $\rho = \rho_l$ and $(y^l, b^l) \in \partial h_\rho(z^l, c^l)$.

end while

if $p(z^{l+1}, c^{l+1}) > 0$ **then**

$\tau_{l+1} \leftarrow \tau_l + \theta$.

end if

Compute $t_{l+1} = \frac{1+\sqrt{1+4t_l^2}}{2}$ and $\alpha^{l+1} = z^{l+1} + \frac{t_l - 1}{t_{l+1}}(z^{l+1} - z^l), \beta^{l+1} = c^{l+1} + \frac{t_l-1}{t_{l+1}}(c^{l+1} - c^l)$ if $l \geq 1$.

$l = l + 1$.

until $\|(z,c)^{l+1} - (z,c)^l\| \leq \epsilon$

The convergence properties and rates of convergence for ADCA-Like are presented in Theorem 3.8 and 3.9 in [35].

3.3.5 The recursive DCA-based approaches for solving the problem (3.10)

It is noted that $T = 2^n - 1$ is exponentially large when the number of accounts n is large. In the problem (3.10), the size of the matrix Q is $(n \times T) \times (n \times T)$. In fact, for a period of 1 hour, the number of active accounts ranges between $5160 - 31182$ with an average of 18270 [59]. Consequently, the large number of accounts makes the size of the matrix Q too enormous, which causes the error of out-of-memory in the programming software. According to our experiments, $n = 12$ makes the size of Q equal to $2,415,919,104$, so the problem (3.10) cannot be solved directly by any algorithm. Therefore, we alternatively combine DCA and other advanced variants of DCA (ADCA, DCA-Like, and ADCA-Like) with the divide-and-conquer algorithm [49] for solving the problem (3.10).

The primary objective of this strategy (illustrated in Fig. 3.1) is to divide the huge number of accounts into a subgroup of m members (m should be small, for instance $m = 6, 7, 8, 9, 10$). Then, we apply each of our optimization approaches to construct the Merkle tree for each subgroup based on the specific transaction distribution. Gather m subtrees sequentially into a set and calculate the correlation coefficients of transaction frequency between the accounts in each bunch of m subtrees. The correlation coefficient is defined as $\sum_{tx=\{a_i,a_j\}\in TX} q_{ij}$ where q_{ij} is the frequency of a transaction between each pair of accounts a_i and a_j in two subtrees. Then it is continued to apply our optimization approach to build a Merkle tree for the above set of m Merkle trees (the root value represents each group). The above process repeats until the Merkle tree is built for all n accounts. The number of iterations to construct a Merkle

tree for a group of n accounts is $\lceil log_m(n) \rceil$. Note that the construction of Merkle trees for subgroups can be done in parallel. The size of matrix Q in our approach is $(m \times T) \times (m \times T) = (m \times 2^m) \times (m \times 2^m)$, where m is the size of subgroup. The detailed algorithm is described in Algorithm 13.

Algorithm 13 The recursive DCA-based approaches to solve the problem (3.10)

Input: Transaction distribution (TX, D); Accounts $A = \{1, 2, ..., n\}$. The size of subgroup m.
Output: Merkle tree for n accounts.
Divide n accounts into each subgroup including m members (the last subgroup can have a smaller number of members than m).
Apply alternatively DCA and other advanced DC algorithms (ADCA, DCA-Like, ADCA-Like) to construct the Merkle tree for each $\lceil \frac{n}{m} \rceil$ subgroup based on (TX, D).
loop = 1.
repeat
Gather m subtrees sequentially into a set and calculate the correlation coefficients of transaction frequency between the accounts in each $\lceil \frac{n}{m^{loop+1}} \rceil$ set of m subtrees.
Apply alternatively DCA and other advanced DC algorithms (ADCA, DCA-Like, ADCA-Like) to build a Merkle tree for the above set of m subtrees.
loop = loop $+1$.
until loop $\leq \lceil log_m(n) \rceil$
return Merkle tree for n accounts.

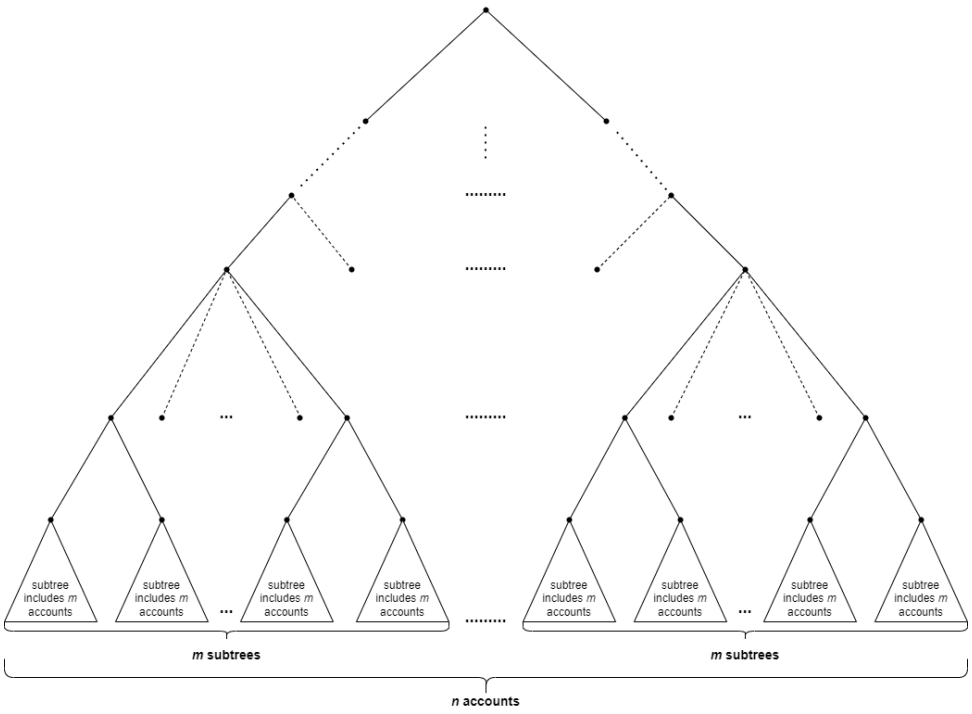

Figure 3.1: Constructing Merkle tree for blockchain transactions using Algorithm 13.

3.4 Numerical experiments

3.4.1 Experiment setting

We used Ethereum transaction data from August 28, 2022, 00:00:00 to August 29, 2022, 00:00:00 [26]. We refer to lengths of 1 minute, 5 minutes, 10 minutes, 20 minutes, 30 minutes, 6 hours, 12 hours, and 1 day to collect the transactions. We implemented Algorithm 13 to construct the Merkle tree for Ethereum transactions involving a large number of accounts.

Our optimization approaches were implemented in Matlab R2019b and performed on a PC Intel Xeon Gold 5118 CPU with two processors (2.30GHz and 2.29GHz) and 128GB of RAM. CPLEX 12.9 was used for solving linear programs. We stop DCA and other advanced DC algorithms (ADCA, DCA-Like, and ADCA-Like) with the tolerance $\epsilon = 10^{-5}$. Concerning the parameters ρ and τ, as ρ_0 and τ_0 are hard to compute, we take a small value of ρ_0 and τ_0 at the beginning and use an adaptive procedure for updating ρ and τ during our algorithm. We set $\eta = 2$ and $\delta = 0.3$. In practice, it is difficult to determine the efficient size of the subgroup because it depends on the specific datasets. We choose the range of $m = 6, 7, 8, 9, 10$.

3.4.2 Comparative algorithms

We will compare our optimization approaches with some heuristic-based schemes that were proposed in [59, 62, 97]. In [97], accounts were sorted randomly in the Merkle tree, for instance based on the order of their associated addresses. Therefore, every account was presented at the same tree height. The first approach assigned fixed-length codewords of the shortest possible length of $\lceil \log_2(n) \rceil$ bits to each of n accounts. This method arbitrarily assigned codewords to accounts.

The second method, based on the Huffman-Merkle Hash Tree (HuffMHT), took into consideration the account distribution deduced from the input transaction distribution [62]. The approach assigned variable-length codewords as Huffman codes based on the probability of each account. Shorter codewords were given to accounts that take part in more transactions in order to minimize the average codeword length based on account distribution.

In [59], the authors modified the Huffman algorithm to consider the distribution of transactions rather than just accounts to build Merkle trees efficiently. They tended to assign long common prefixes to pairs of accounts with frequent joint transactions, in addition to assigning short codewords to common accounts. The authors determined the upper bounds for the optimal communication cost $OPT(D)$ for a given distribution D. The optimal communication cost for a transaction distribution D defined over n accounts satisfies $OPT(D) \leq 2 \times \lceil \log_2(n) \rceil$. These comparative algorithms were implemented in the Python 3.8.

3.4.3 Comparative results

Comparison of the recursive DCA approach with the recursive DCA-Like approach

We compare the performance of the recursive DCA approach (Algorithm 9 is used in Algorithm 13) and the recursive DCA-Like approach (Algorithm 11 is used in Algorithm 13) in terms of communication cost (measured in bits) and processing time (measured in seconds) to solve the problem of designing Merkle tree structures for blockchain transactions in cases where the number of accounts (no of accounts,

in short) and transactions (no of trans, in short) is large. The number of variables and constraints in the recursive DCA approach and recursive DCA-Like approach are defined as $\frac{m}{m-1}(m^{\lceil \log_m n \rceil} - 1)(m + 1)(2^m - 1)$ and $\frac{m}{m-1}(m^{\lceil \log_m n \rceil} - 1)(3 \times (2^m - 1) + m)$ respectively, where n is the number of accounts and m is the chosen size of the subgroup. The results are summarized in Table 3.1.

Table 3.1: Comparative results between the recursive DCA approach and recursive DCA-Like approach. Bold values indicate the best results.

No of accounts	No of trans	No of variables	No of constraints	Size of subgroup	Recursive DCA		Recursive DCA-Like	
					Cost	Time	Cost	Time
1305	1002	125,020,830	34,207,690	10	19.14	353.1	17.54	374.5
		37,711,800	11,379,960	9	18.90	210.9	18.40	180.3
		10,740,600	3,617,640	8	18.75	31.0	18.09	42.7
		2,844,800	1,086,400	7	18.14	9.2	**17.16**	9.5
		4,114,530	1,819,350	6	18.45	6.3	17.97	**5.9**
5998	5245	125,020,830	34,207,690	10	22.96	1579.9	21.35	1616.4
		37,711,800	11,379,960	9	22.51	923.8	22.42	852.9
		85,943,160	28,947,304	8	24.51	178.8	23.89	181.6
		19,920,712	7,607,516	7	22.02	37.4	**21.32**	37.6
		4,114,530	1,819,350	6	21.82	**27.5**	21.51	28.8
73106	74656	1,250,320,830	342,107,690	10	26.25	18752.2	25.65	8108.5
		3,055,115,700	921,915,540	9	26.98	12092.1	25.28	6408.3
		687,563,640	231,584,616	8	26.83	2558.5	25.87	2335.6
		139,452,096	53,255,328	7	25.55	498.4	**25.06**	489.1
		148,141,602	65,504,790	6	26.15	318.5	25.28	**302.7**
78750	77654	1,250,320,830	342,107,690	10	26.47	20285.2	24.91	8797.8
		3,055,115,700	921,915,540	9	26.75	13115.6	25.33	6953.1
		687,563,640	231,584,616	8	25.82	2686.1	25.45	2499.6
		139,452,096	53,255,328	7	25.24	487.3	**24.84**	474.7
		148,141,602	65,504,790	6	25.58	343.9	25.12	**316.4**
81371	80653	1,250,320,830	342,107,690	10	26.09	20836.1	25.37	9926.4
		3,055,115,700	921,915,540	9	26.37	13404.8	25.35	7845.8
		687,563,640	231,584,616	8	25.44	2777.3	24.64	2514.7
		139,452,096	53,255,328	7	25.86	583.9	**24.59**	456.5
		148,141,602	65,504,790	6	25.92	374.5	25.06	**368.8**
137546	180435	12,503,320,830	3,421,107,690	10	28.78	29739.9	27.96	14168.4
		3,055,115,700	921,915,540	9	28.62	17763.2	28.10	11198.6
		687,563,640	231,584,616	8	28.80	4463.2	28.37	4321.4
		976,171,784	372,790,012	7	28.42	940.3	**27.46**	926.8
		148,141,602	65,504,790	6	28.55	626.9	28.07	**555.7**
227977	329067	12,503,320,830	3,421,107,690	10	29.97	56503.8	29.28	24713.5
		3,055,115,700	921,915,540	9	29.85	30352.9	29.45	19503.7
		687,563,640	231,584,616	8	30.09	7742.4	29.35	7024.2
		976,171,784	372,790,012	7	29.64	2024.6	**28.99**	1491.3
		148,141,602	65,504,790	6	30.29	909.7	29.42	**886.9**
445407	662847	12,503,320,830	3,421,107,690	10	32.24	115901.7	30.47	49153.8
		3,055,115,700	921,915,540	9	32.87	56536.6	31.53	38791.4
		5,500,527,480	1,852,683,112	8	33.54	16194.1	32.43	14293.8
		976,171,784	372,790,012	7	31.33	3419.8	**30.05**	3372.8
		888,852,258	393,029,910	6	32.57	1924.2	31.71	**1824.0**

Comments on numerical results:

In cases where the number of accounts is small ($n = 1305, 5998$), the recursive DCA approach is slightly faster than the recursive DCA-Like approach, while the latter is better than the former in terms of objective value.

In datasets where the number of accounts is medium or large, the recursive DCA-Like approach is competitive with the recursive DCA approach in all two criteria. In terms of running time, the recursive DCA-Like approach runs on average $2.3, 1.8, 1.1, 1.12$, and 1.06 times faster than the recursive DCA approach in cases where the size of the subgroup is 10, 9, 8, 7, and 6, respectively. Furthermore, the recursive DCA-Like approach provides approximately 5% lower cost than the recursive DCA approach.

As a result, the DCA-Like approach enhances the standard DCA approach in terms of both solution quality and rapidity in large-scale settings.

In fact, it is challenging to determine the appropriate size of the subgroup since it relies on the particular datasets. In most cases, the communication cost is the lowest, with a subgroup size of $m = 7$. Moreover, the smaller the subgroup's size is, the faster the execution time is. Our approaches, in particular, yield the shortest running time when the subgroup size is 6. An experimentally good choice for these datasets could be $m = 7$.

Comparison of the recursive DCA, ADCA, DCA-Like, and ADCA-Like approaches

Table 3.2 summarizes the experimental results of comparing the recursive DCA, ADCA, DCA-Like, and ADCA-Like approaches for optimizing the Merkle tree structure based on the transaction distribution. We in turn combine our algorithms (Algorithm 9, 10, 11, and 12) with the divide and conquer algorithm for solving the Merkle tree construction problem with the large accounts in the Ethereum system. The chosen size of subgroup is 7.

Table 3.2: Comparative results of the recursive DCA, ADCA, DCA-Like, and ADCA-Like approaches. Bold values indicate the best results.

No of accounts	**No of trans**	**No of variables**	**No of constraints**	**Algorithm**	**Cost (bits)**	**Running time (s)**
1305	1002	2,844,800	1,086,400	Recursive DCA	18.14	9.2
				Recursive ADCA	**17.01**	**7.6**
				Recursive DCA-Like	17.16	9.5
				Recursive ADCA-Like	17.05	8.9
5998	5245	19,920,712	7,607,516	Recursive DCA	22.02	37.4
				Recursive ADCA	21.42	37.2
				Recursive DCA-Like	21.32	37.6
				Recursive ADCA-Like	**21.28**	**36.5**
73106	74656	139,452,096	53,255,328	Recursive DCA	26.15	498.4
				Recursive ADCA	25.06	444.2
				Recursive DCA-Like	25.06	489.1
				Recursive ADCA-Like	**24.07**	**403.7**
78750	77654	139,452,096	53,255,328	Recursive DCA	25.24	487.3
				Recursive ADCA	24.83	451.3

Continued on next page

Table 3.2 – continued from previous page

No of accounts	No of trans	No of variables	No of constraints	Algorithm	Cost (bits)	Running time (s)
				Recursive DCA-Like	24.84	474.7
				Recursive ADCA-Like	**24.35**	**441.1**
81371	80653	139,452,096	53,255,328	Recursive DCA	25.86	583.9
				Recursive ADCA	24.59	469.3
				Recursive DCA-Like	24.59	456.5
				Recursive ADCA-Like	**24.45**	**448.2**
137546	180435	976,171,784	372,790,012	Recursive DCA	28.42	940.3
				Recursive ADCA	27.96	930.5
				Recursive DCA-Like	27.46	926.8
				Recursive ADCA-Like	**26.08**	**850.3**
227977	329067	976,171,784	372,790,012	Recursive DCA	29.64	2024.6
				Recursive ADCA	28.99	1371.6
				Recursive DCA-Like	28.99	1491.3
				Recursive ADCA-Like	**27.16**	**1285.1**
445407	662847	976,171,784	372,790,012	Recursive DCA	31.33	3419.8
				Recursive ADCA	30.47	3018.6
				Recursive DCA-Like	30.05	3372.8
				Recursive ADCA-Like	**29.59**	**2944.9**

Comments on numerical results:

- We evaluate the advantages of using acceleration techniques by conducting a comparative analysis between two pairings: the recursive ADCA approach versus the recursive DCA approach and the recursive ADCA-Like approach versus the recursive DCA-Like approach.

As for the comparison of the recursive DCA and ADCA approaches, in most of the cases, the recursive ADCA approach is better than the recursive DCA approach. In eight datasets, the ADCA approach is superior to the DCA approach in terms of communication cost and running time. The ratio of gains in terms of running time is from 1.2 to 1.5 times. The cost of the recursive ADCA approach is consistently 3.3% less than that of the recursive DCA approach.

For all datasets, the recursive ADCA-Like approach provides the lower cost than the recursive DCA-Like approach by 2.5% on average. The gains in running time are also competitive, the ADCA-Like approach is faster than the DCA-Like approach from 1.1 to 1.5 times. Therefore, we can say that ADCA-Like approach further improves the performance of DCA-Like approach.

- Among the four comparative approaches, the recursive ADCA-Like approach gives the best results. In terms of communication cost, the recursive ADCA-Like approach gives the best result in 7 cases, followed by the recursive ADCA approach in 1 case. As for the running time, the ADCA-Like approach, followed by ADCA approach, is the fastest in seven over eight datasets. On another hand, the recursive ADCA and DCA-Like approaches furnish the same objective value on four datasets (when the number of accounts is 73106, 78750, 81371 and 227977). This illustrates the superior of the recursive ADCA-Like approach versus the other approaches.

Comparison of the recursive ADCA-Like approach with other existing algorithms

We compare the performance of our optimization approach, named Recursive ADCA-Like (Algorithm 12 is integrated in Algorithm 13), with three other existing algorithms (Random_tree [97], Huffman_tree [62], and Pairs-first Huffman_tree [59]) in terms of the following two criteria: the communication cost (bits) and the running time (seconds). Baseline is the upper bound for the optimal communication cost for a given transaction distribution [59].

Table 3.3: Comparative results of the recursive ADCA-Like approach with other existing algorithms. Bold values indicate the best results.

Number of accounts	Number of transactions	Algorithm	Communication cost (bits)	Running time (s)
1305	1002	Recursive ADCA-Like	**17.05**	8.9
		Baseline	22.00	
		Random_tree	19.60	**0.2**
		Huffman_tree	20.75	0.7
		Pairs-first Huffman_tree	19.47	1.7
5998	5245	Recursive ADCA-Like	**21.28**	36.5
		Baseline	27.79	
		Random_tree	23.76	**1.3**
		Huffman_tree	23.31	14.6
		Pairs-first Huffman_tree	23.22	40.9
73106	74656	Recursive ADCA-Like	**24.07**	403.7
		Baseline	32.94	
		Random_tree	28.68	**76.9**
		Huffman_tree	27.73	1080.9
		Pairs-first Huffman_tree	26.64	4650.2
78750	77654	Recursive ADCA-Like	**24.35**	441.1
		Baseline	33.83	
		Random_tree	29.19	**84.7**
		Huffman_tree	27.88	1127.1
		Pairs-first Huffman_tree	26.98	5164.2
81371	80653	Recursive ADCA-Like	**24.45**	448.2
		Baseline	33.37	
		Random_tree	28.97	**91.7**
		Huffman_tree	27.53	1266.3
		Pairs-first Huffman_tree	26.47	4534.6
137546	180435	Recursive ADCA-Like	**26.08**	850.3
		Baseline	36.00	
		Random_tree	32.40	**299.3**
		Huffman_tree	31.37	5140.7
		Pairs-first Huffman_tree	30.09	20478.2

Continued on next page

Table 3.3 – continued from previous page

Number of accounts	Number of transactions	Algorithm	Communication cost (bits)	Running time (s)
227977	329067	Recursive ADCA-Like	**27.16**	1285.1
		Baseline	36.00	
		Random_tree	33.71	**791.6**
		Huffman_tree	32.12	15654.6
		Pairs-first Huffman_tree	30.94	65227.5
445407	662847	Recursive ADCA-Like	**29.59**	2944.9
		Baseline	38.01	
		Random_tree	35.66	**2921.5**
		Huffman_tree	34.12	34561.3
		Pairs-first Huffman_tree	33.90	151374.9

Comments on numerical results:

- In terms of communication cost, our approach Recursive ADCA-Like results in the lowest cost of the existing methods to construct a Merkle tree in all cases. The cost of our approach on average is less than the Random algorithm, Huffman algorithm, and Pairs-first Huffman method $15.9\%, 13.6\%$ and 10.8%, respectively.

- In cases where the number of accounts is not large $n = 1305, 5998$, the execution time of the Random algorithm is always significantly faster than other methods. When the number of accounts is medium or large, the running time of Random algorithm is approximately 3.5 times faster than our algorithm. As for the comparison of the Huffman algorithm and our approach, the running time of our approach is faster than the Huffman algorithm by 2.5 to 12.2 times when the number of accounts is 73106, 78750, 81371, 137546, 227977, and 445407. Similarly, our method is on average 26.6 times quicker than the Pairs-first Huffman method in all instances in which the number of accounts is medium or large.

Overall, our optimization approach offers a competitive trade-off between communication cost and running time for constructing a Merkle tree based on the transaction distribution of Ethereum accounts.

3.5 Conclusion

In this chapter, we have proposed an optimization model for constructing the Merkle tree in the Ethereum transaction system. To our knowledge, this is the first mathematical model for solving the Merkle tree structure problem based on transaction distribution. The suggested optimization problem is a binary quadratic program. Thanks to the recent results on exact penalty techniques in DC programming, the problem has been reformulated as a DC program. We then applied an efficient DCA to solve this problem.

To find a better convex approximation to the current solution through a decomposition that is not necessarily DC, we applied the DCA-Like to the problem of constructing a tree structure with the minimal number of hash values required to update the account data associated with each transaction. DCA-Like

improves the standard DCA in both quality of solution and running time in large-scale settings. At the same time, we deployed Accelerated DCA (ADCA) and Accelerated DCA-Like (ADCA-Like) to improve DCA and DCA-Like by incorporating Nesterov's acceleration technique into them.

Furthermore, we suggested the more effective optimization approach to building a Merkle tree from a large number of blockchain accounts. Our method separates accounts into small subgroups and recursively builds the Merkle tree for all accounts using alternately DCA and other advanced variants of DCA (ADCA, DCA-Like, and ADCA-Like). Numerical experiments show the benefits of our suggested model and the corresponding approaches.

Chapter 4

A stochastic DCA based approach to Autoencoder architecture in Text Encryption and Decryption

Cryptography is the science and practice of ensuring the security of communications and data by converting them into an unreadable format, thereby preventing unauthorized parties from understanding the content. At the same time, autoencoders play a fundamental role in unsupervised learning and in deep architectures for transfer learning and other tasks. Using an autoencoder, we will generate a stable method of text encryption and decryption. It is an alternative cryptosystem, avoiding the use of traditional sequential algorithms. In this chapter, we propose a symmetric-key encryption system using an autoencoder that is used to securely encrypt and decrypt text messages represented in an 8-bit format, which corresponds to an ASCII (American Standard Code for Information Interchange) character. In this proposed autoencoder, we use the Binary Cross Entropy loss function combined with the penalty function in order to obtain output values that are the same as the binary inputs. Moreover, we apply a novel stochastic algorithm, known as Markov chain stochastic DCA (MCSDCA), as the optimizer in various autoencoder architectures. The proposed cryptosystem's performance and confidentiality are demonstrated via a variety of analyses.

4.1 Introduction

Cryptography is the art and science that focuses on the analysis and development of techniques and methodologies for securing communications, ensuring that only those with proper authorization may access and comprehend the content. The basic objective of cryptography is to ensure the confidentiality, privacy, integrity, and validity of data. The confidentiality of information is achieved by using a key to encrypt and decrypt communications. Consequently, cryptography plays one of the most crucial and challenging roles in the digital age, where data security is ubiquitous. Indeed, encryption algorithms are frequently utilized in banking, social networks, e-commerce, and streaming media.

Encryption algorithms are applied to data (often known as plaintext) to produce encrypted data (or ciphertext). This method is called encryption. The encryption should be constructed so that the ciphertext contains no information about the plaintext, with the possible exception of its length. Every encryption algorithm has a related decryption algorithm that converts ciphertext back to its plaintext form. Encryption systems operate in conjunction with a key. In a symmetric encryption system, both

the encryption and decryption algorithms use the same key. Encryption and decryption are performed using distinct but related keys in an asymmetric encryption system.

Numerous symmetric key algorithms are used to encrypt and decrypt text, including AES, CAMELLIA, DES, 3DES, Blowfish, RC6, CAST, RC4, Salsa20, etc. The Advanced Encryption Standard (AES) algorithm is the most extensively adopted algorithm for modern data encryption in industry. The U.S. National Institute of Standards and Technology (NIST) announced AES in 2001 to supersede the Data Encryption Standard (DES). It uses a series of substitution, permutation, and combining operations to convert plaintext to ciphertext in a secure manner. It employs a 16-byte fixed data length and variable key lengths. AES utilizes ten rounds for 128-bit keys, twelve rounds for 192-bit keys, and fourteen rounds for 256-bit keys. Numerous attacks on AES, including brute force attacks, differential cryptanalysis, known-plaintext and chosen-plaintext attacks, side-channel attacks, etc., have been published [10, 91, 95].

Another technique for text encryption in cryptography is using asymmetric key algorithms. Popular public key algorithms include RSA, ElGamal, and Elliptic Curve Cryptography (ECC). RSA stands for Rivest, Shamir, and Adleman, who introduced the RSA algorithm in 1977. In RSA, a user generates a pair of keys: a public key, which can be shared on a public channel, and a private key, which must be kept confidential. Messages encrypted with the public key can only be decrypted with the corresponding private key, ensuring the confidentiality of the data. RSA is also used for digital signatures, where the private key is used to sign documents and the public key is used to verify the signature's authenticity, providing data integrity and authentication. The security of RSA relies on the difficulty of factoring the product of two large prime numbers, making it computationally infeasible for attackers to derive the private key from the public key [56]. Despite its age, RSA remains a crucial component of secure communication, particularly in key exchange and digital certificate systems, although its key lengths have had to increase over time to withstand advances in computing power and cryptanalysis. NIST has recommended key lengths for RSA that are suitable for many levels of security: 2048 bits, 3072 bits, and 4096 bits. ElGamal, developed by Taher ElGamal in the 1980s, is based on the difficulty of the discrete logarithm problem in finite fields [56]. It is primarily used for encryption and digital signatures. ElGamal is known for its security but is generally slower and requires larger key sizes compared to ECC. On the other hand, ECC is a more recent cryptographic approach that leverages the mathematical properties of elliptic curves to provide strong security with smaller key sizes and faster operations compared to traditional methods like RSA and ElGamal [56]. ECC is widely used in various security applications, including secure communications, digital signatures, and secure key exchange. Its efficiency makes it especially suitable for resource-constrained environments like mobile devices and IoT devices, where computational resources are limited. The security of ECC relies on the elliptic curve discrete logarithm problem.

Due to the impending threat posed by quantum computers, post-quantum algorithms for encryption are a crucial area of cryptography research [11, 7]. Quantum computers, when realized on a sufficiently large scale, are anticipated to break many of the current cryptographic algorithms used to secure data, such as RSA, ElGamal, and ECC, by solving mathematical problems such as integer factorization and discrete logarithms efficiently. Post-quantum cryptography focuses on developing encryption algorithms that are resistant to attacks by quantum computers. These new algorithms rely on mathematical problems that quantum computers cannot solve significantly faster than classical computers. Among the nu-

merous post-quantum encryption techniques are lattice-based cryptography, code-based cryptography, multivariate polynomial cryptography, and others. The objective is to assure the long-term security of sensitive data in an era when quantum computers are practical, thereby protecting the confidentiality and integrity of data in a world powered by quantum computing. NIST initiated a solicitation, evaluation, and standardization process for one or more quantum-resistant public-key cryptographic algorithms.

In the digital world, a relatively new collection of computational tools, artificial neural networks (ANNs), has been developed to handle and solve several difficult real-life issues. ANNs are useful because they can process information in a nonlinear manner, are highly parallelizable, fault- and noise-tolerant, and have the potential to learn [9]. Owing to the stochastic nature of the training phase of neural networks, there are numerous conceivable combinations of weights and connections, allowing for the concealment and compression of information in a variety of ways. Indeed, neural networks may be widely applied to cryptography. Neural networks can also be applied in preliminary steps, such as the creation of chaotic binary sequences to be deployed with other trapdoor functions.

A feed-forward, multilayer neural network that learns to recreate identical input data is an autoencoder. It is composed of an input layer, an output layer (both with the same number of neurons), and one or more intermediate layers, also known as hidden layers, so that a network bottleneck forces a compressed representation of the original input [87]. Due to their intrinsic encoding and decoding capabilities, autoencoders integrated with cryptography provide a novel way for encrypting and decrypting data.

Many research investigations have been conducted on autoencoders for the purpose of text encryption and decryption [23, 87, 76, 57]. In [23], the authors proposed methods of encryption and decryption with a deep learning approach that utilizes two Autoencoder neural networks (AENNs) with multilayer architecture as the secret key generator and one AENN as a hash value generator. The error function was declared with Sum of Squared Errors (SSE) and they used the gradient descent algorithm to solve the optimization problem. In [87], it was implemented to code and decode characters represented in an 8-bit format, which corresponds to the size of the ASCII representation. The back-propagation algorithm was used in order to perform the learning process with two different variants depending on when the rounding procedure was carried out—during or after the learning phase. The architecture employed for these autoencoders was 8 input neurons in order to represent ASCII characters, n neurons in the first hidden layer, 8 neurons in the second hidden layer to represent the codification of the data, n neurons in the third hidden layer, and finally 8 neurons in the output layer to recover the initial data. Several tests (with $n = 7, 8, 9, 10$) were conducted to determine the best autoencoder architectures to encrypt and decrypt, taking into account that a good encrypt method corresponds to a process that generates a new code with uniqueness and a good decrypt method successfully recovers the input data.

In [76], a sparse autoencoder architecture was chosen, consisting of an input layer with 8 neurons, a hidden layer with 9 neurons, and an output layer with 8 neurons. The rationale behind selecting a single layer is rooted in the need to maintain simplicity and effectiveness within the system, particularly when including a greater number of neurons compared to the input layer. In this implementation, the activation function utilized was a sigmoid function with a gain parameter denoted as β. The use of a variable gain value facilitates the incorporation of layers that employ distinct variations of the sigmoid function. Such a mechanism enables the gradient descent problem to be mitigated. In [57], a hybrid autoencoder architecture was proposed and studied to encrypt and decrypt digital information represented with binary

sequences in a secure manner. The architecture employed for the autoencoder was 8 input neurons in order to represent ASCII characters, 10 neurons in the first hidden layer, 8 neurons in the second hidden layer to represent the codification of the data, 10 neurons in the third hidden layer, and finally 8 neurons in the output layer to recover the initial data. Five activation functions, including Sigmoid, ReLU, Tanh, Sine, and LeakyReLU, were compared in terms of training loss, testing accuracy, encryption time, and decryption time. Mean Squared Error (MSE) was used as the loss function, and the optimizer was Nadam.

Our contributions. In this chapter, we propose a symmetric-key encryption system using an autoencoder that is used to securely encrypt and decrypt text messages represented in an 8-bit format, which corresponds to an ASCII (American Standard Code for Information Interchange) character. Eight different autoencoder architectures are proposed for text encryption and decryption. In these proposed autoencoders, we use the Binary Cross Entropy loss function combined with the penalty function in order to obtain output values that are the same as the binary inputs. Furthermore, we deploy a novel stochastic algorithm known as Markov chain stochastic DCA (MCSDCA) [39, 53, 54] as the optimizer in different autoencoder architectures. Numerical experiments show the virtues of MCSDCA in comparison with other baseline optimizers (such as Adam and stochastic gradient descent (SGD)) in training autoencoders. This suggests that the proposed method has a high degree of confidentiality because each training epoch always produces a different and large set of trained weights.

4.2 Autoencoder for text encryption and decryption

4.2.1 Autoencoder

The autoencoder was first proposed in the 1980s by Hinton and the PDP group [81]. It is a neural network design used in unsupervised learning to effectively extract significant features and patterns from data. Furthermore, autoencoders have become increasingly significant in recent years within the realm of deep learning methodologies [21, 28, 32]. In this approach, the individual layers of an autoencoder are trained independently in an unsupervised manner and subsequently combined to optimize the entire network, resulting in a restricted Boltzmann machine. The network configuration described has shown significant advancements in many current classification and regression issues [6]. Autoencoders are extensively used in several domains, including but not limited to data compression, natural language processing, image denoising, and analogous data generation.

Autoencoder architecture

An artificial neural network with autoencoder architecture consists of a first layer that presents input data and an output layer that recovers input data. Both layers must consist of the same number of neuron units (neurons) plus one or more hidden layers [64]. Furthermore, for the purpose of using the autoencoder in encryption and decryption applications, it is essential that the design maintains symmetry. The autoencoder is trained in an unsupervised manner via the use of the back-propagation technique, where the input values are employed as the target values. Therefore, the primary objective of an autoencoder is to approximate the function $h_{W,b}(x) \approx x$. In other words, the objective of this process is to train a model to rebuild an input vector at the output layer in a manner that ensures a high degree of similarity

between the original vector x and the reconstructed vector $\hat{x}$. The architectural design of an autoencoder consists of three hidden layers, each containing a different number of neurons denoted as m, h, and m accordingly. The autoencoder takes in an input vector of size n, as seen in Fig. 4.1.

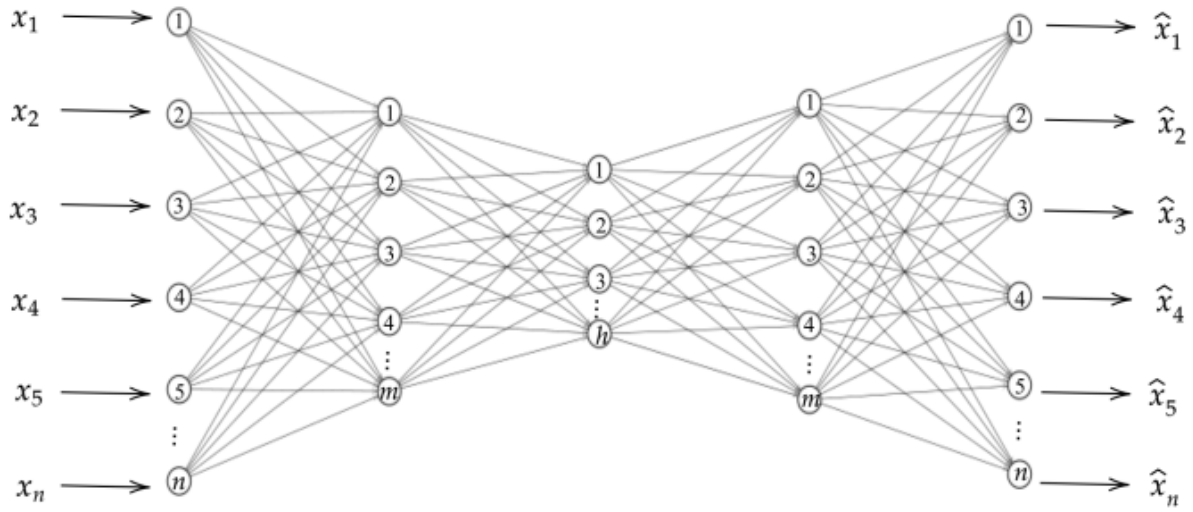

Figure 4.1: Architecture of an autoencoder [76].

Although the approximation of the identity function seems not very interesting, an autoencoder can discover fascinating structures based on the input data when imposing constraints on the number of hidden neurons. One illustrative case to consider is an image compression system. Let us consider the inputs as the pixel values of a 10×10 image, so $n = 100$ and $x \in \mathbb{R}^{100}$ and that there are only 50 neurons in one hidden layer. In this situation, the system is required to compress the input from $\mathbb{R}^{100}$ to $\mathbb{R}^{50}$ before reconstructing it to $\mathbb{R}^{100}$. Working with independent and identically distributed (i.i.d.) random variables would make the compression procedures extremely challenging. Nonetheless, if there are patterns in the data, this algorithm will identify them [64]. Occasionally, compressing the input is not the only method for discovering data set features. Even if other constraints, such as a large number of hidden neurons, exist in the hidden layers, the autoencoder can still find intriguing structures; this is the concept of sparsity.

Back-Propagation algorithm

The Back-Propagation algorithm [4] is a supervised learning method for training multilayer artificial neural networks, and even if the algorithm is very well-known, we summarize in this section the main equations in relation to the implementation of the Back-Propagation algorithm, as they are important in order to understand the current work.

Consider a neural network architecture comprising several hidden layers. The activation of the neurons belonging to a hidden or output layer, denoted by y_i, can be written as:

$$y_i = g\left(\sum_{j=1}^{L} w_{ij} \times s_j\right) = g(h), \tag{4.1}$$

where w_{ij} are the synaptic weights between neuron i in the current layer and the neurons of the previous layer with activation s_j. In the previous equation, h has been introduced as the synaptic potential of a neuron. The activation function used, g, is the logistic function given by the following equation:

$$g(x) = \frac{1}{1 + e^{-\beta x}}. \tag{4.2}$$

The objective of the Back-Propagation supervised learning algorithm is to minimize the difference between given outputs (targets) for a set of input data and the output of the network. This error depends on the values of the synaptic weights, and so these should be adjusted in order to minimize the error. The error function computed for all output neurons can be defined as:

$$E = \frac{1}{2} \sum_{k=1}^{p} \sum_{i=1}^{n} (z_i(k) - y_i(k))^2, \tag{4.3}$$

where the first sum is performed on the data set's p patterns and the second sum is performed on the n output neurons. $z_i(k)$ is the target value of output neuron i for pattern k, while $y_i(k)$ is the network's response output. By using the method of gradient descent, the Back-Propagation attempts to minimize this error in an iterative process by updating the synaptic weights upon the presentation of a given pattern. The synaptic weights between the two last layers of neurons are updated as follows:

$$\Delta w_{ij}(k) = -\eta \frac{\partial E}{\partial w_{ij}(k)} = \eta [z_i(k) - y_i(k)] g_i'(h_i) s_j(k), \tag{4.4}$$

where the learning rate (η) is a predetermined parameter. The derivative of the sigmoid function (g') and the synaptic potential (h) are also utilized. The remaining weights are adjusted using equations similar to those introduced in the algorithm, incorporating a set of values known as "deltas" (δ). These deltas propagate the error from the last layer to the inner layers and are computed based on equations (4.5) and (4.6).

The delta value for each neuron i of the last of the N hidden layers is computed as:

$$\delta_i^{(N)} = -[z_i - y_i^{(N)}](y_i^{(N)})'. \tag{4.5}$$

The delta values for the rest of the hidden layer neurons are computed according to:

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} w_{ji} \delta_j^{(l+1)} \right) (y_i^{(l)})', \tag{4.6}$$

where s_{l+1} is denoted as the number of nodes in layer $l + 1$.

Therefore, the desired partial derivatives are given as:

$$\frac{\partial E}{\partial w_{ij}^{(l)}} = y_j^{(l)} \delta_i^{(l+1)}.$$

After the weights are properly adjusted, the system is ready to correlate unknown data.

4.2.2 Proposed autoencoder architecture for text encryption and decryption

Autoencoder with one hidden layer

The proposed autoencoder network can be divided into two major components, namely the encoding network (encoder) and the decoding network (decoder). The encoding network takes in an 8-bit ASCII value as input and hence has 8 neurons, one for each bit. The inputs are then passed through a hidden layer consisting of $n = 7, 8, 9, 10$ units whose values, after training, form the encrypted array. This encrypted array can be transmitted to the receiver. Therefore, the encoding network carries out the encryption procedure. The rectified linear activation function, or ReLU for short, has been used in the encoding network. It is a piecewise linear function that will output the input directly if it is positive; otherwise, it will output zero, as shown below:

$$\text{ReLU}(x) = \max(0, x). \tag{4.7}$$

It has become the default activation function for many types of neural networks because a model that uses it is easier to train due to the value of its gradient being 1 for input values greater than 0, thus allowing for large update steps.

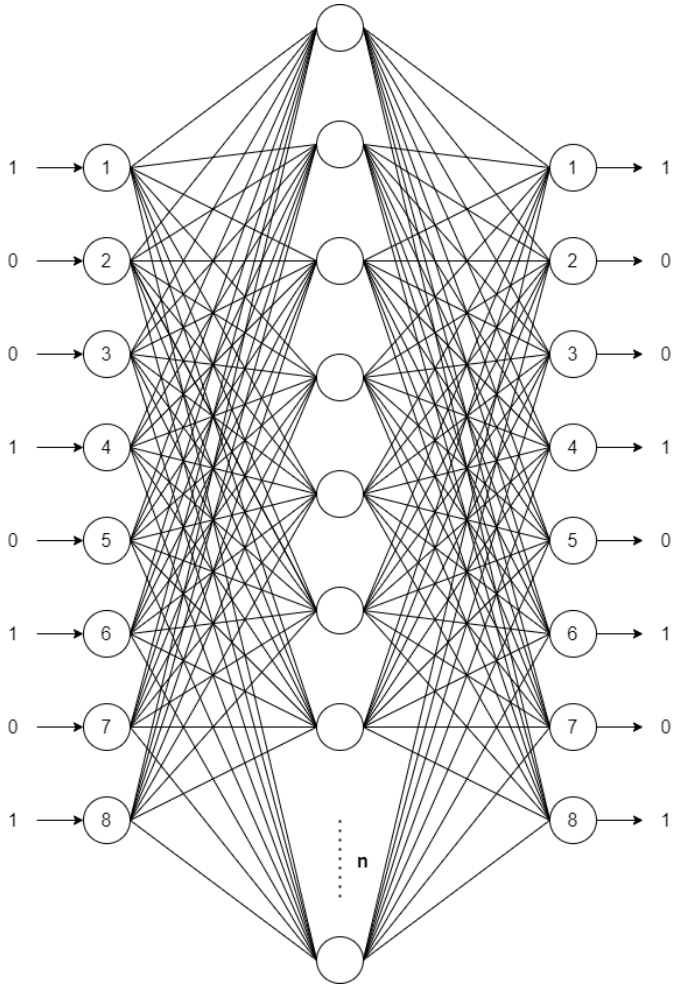

Figure 4.2: The first autoencoder architecture of proposed system.

The decoding network takes in the encrypted array (consisting of n encrypted values) as input and hence contains n neurons. An output layer includes 8 neurons. This output layer, after training, will reproduce the original 8-bit ASCII that was given as input to the encoding network. The decoding network mirrors the architecture of the encoding layer and hence carries out decryption at the receiver side. The encoding network as well as the decoding network have been combined during the training phase, as shown in Fig. 4.2, to ensure minimum loss in decrypted data and maximum replication of the input values. We select the sigmoid function as the activation function of the decoding network. The sigmoid activation function, or logistic function, is one of the most used and popular activation functions for neural networks. The input values get mapped between values 0.0 and 1.0. Due to this property, the sigmoid function is a common choice for networks that work on probability.

$$\text{Sigmoid}(x) = \frac{1}{1+e^{-x}}. \tag{4.8}$$

The loss function used in the autoencoder architecture is binary cross-entropy (BCE). The binary characteristics of the input in the proposed autoencoder make BCE a suitable choice to measure the similarity between the input data and the reconstructed output from the autoencoder. For non-binary data or regression tasks, other loss functions like mean squared error (MSE) or mean absolute error (MAE) may be more suitable. In particular, the definition of the binary cross-entropy loss function is defined as:

$$\text{BCELoss} = -\frac{1}{m}\sum_{k=1}^{m}\sum_{i=1}^{8}\left[x_i(k)\times\log(\hat{x_i}(k)) + (1-x_i(k))\times\log(1-\hat{x_i}(k))\right], \tag{4.9}$$

where m is the number of data samples in a batch, $x_i(k)$ is the value of input neuron i for training sample k, and $\hat{x_i}(k)$ is the value of output neuron i for training sample k.

In fact, the input and output values are binary sequences, so we propose to use the penalty function combined with the loss function in order to obtain the binary output values. Let the penalty function be defined by

$$p(\hat{x_i}(k)) := \sum_{k=1}^{m}\sum_{i=1}^{8}\hat{x_i}(k)(1-\hat{x_i}(k)). \tag{4.10}$$

Therefore, the objective function is the following:

$$\begin{aligned} F(x,\hat{x}) = -\frac{1}{m}\sum_{k=1}^{m}\sum_{i=1}^{8}&\left[x_i(k)\times\log(\hat{x_i}(k)) + (1-x_i(k))\times\log(1-\hat{x_i}(k))\right] \\ &+\lambda\sum_{k=1}^{m}\sum_{i=1}^{8}\hat{x_i}(k)(1-\hat{x_i}(k)), \hat{x_i}(k)\in[0,1], k=\overline{1,m}, i=\overline{1,8}, \end{aligned} \tag{4.11}$$

where λ is a positive parameter controlling the trade-off between the loss function and the penalty function.

Table 4.1: The hyperparameter list of proposed autoencoders.

Autoencoder	Number of neurons in each layer	Number of parameters in encoding network	Number of parameters in decoding network
Autoencoder1	8-7-8	63	64
Autoencoder2	8-8-8	72	72
Autoencoder3	8-9-8	81	80
Autoencoder4	8-10-8	90	88
Autoencoder5	8-7-8-7-8	127	127
Autoencoder6	8-8-8-8-8	144	144
Autoencoder7	8-9-8-9-8	161	161
Autoencoder8	8-10-8-10-8	178	178

Autoencoder with more than one hidden layer

We apply the autoencoder structure proposed in [87]. The architecture employed for the autoencoder has been 8 input neurons in order to represent ASCII characters, n neurons in the first hidden layer, 8 neurons in the second hidden layer to represent the ciphertext, n neurons in the third hidden layer, and finally 8 neurons in the output layer to recover the initial data. The architecture is shown in Fig. 4.3, taking into account that n has been changed to analyze the best architecture. Specifically, n was chosen to be equal to $7, 8, 9,$ and 10.

The activation function is chosen as a sigmoid function. The objective function is the combination of the BCE function and the penalty function, as mentioned above.

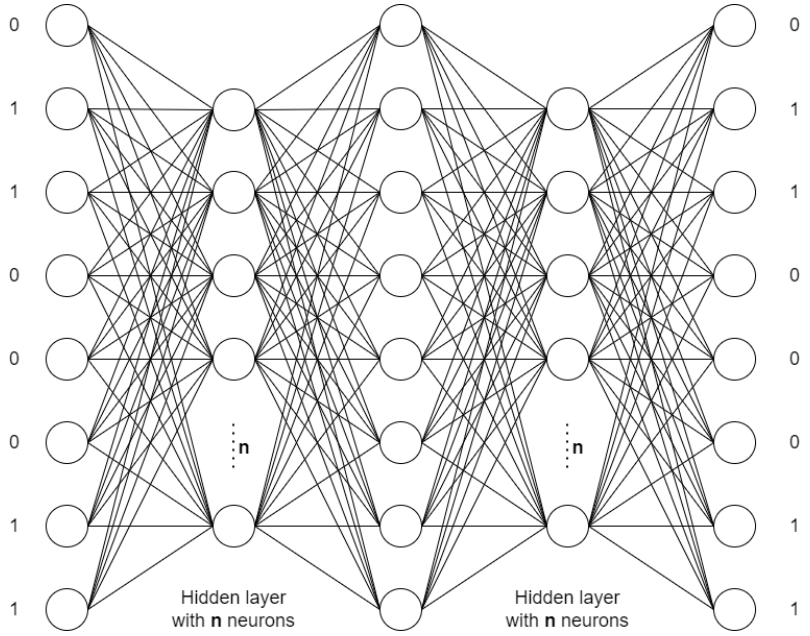

Figure 4.3: The second autoencoder architecture of proposed system [87].

Table 4.1 is a summary of the autoencoder architectures used for the encryption and decryption of text data.

Simultaneously, stochastic optimization algorithms are frequently used for training autoencoders and neural networks in general. In comparison to batch optimization methods, stochastic optimization techniques, such as stochastic gradient descent (SGD), Adam, Nadam, RMSprop, etc., are computationally

efficient and permit faster convergence. This is especially essential when working with large datasets, as modifying the model parameters on a per-sample or mini-batch basis is significantly quicker than processing the entire dataset at once. In addition, stochastic optimization incorporates randomization into the training procedure by selecting random subsets of data for each parameter update. This variability prevents the model from getting stuck in local minimums and can lead to more accurate generalizations on data that is unseen. In this study, we apply a novel stochastic technique known as Markov chain stochastic DCA (MCSDCA) [39, 53, 54] in training autoencoder for text encryption and decryption.

4.2.3 MCSDCA optimizer [39, 53, 54]

We apply the optimizer named MCSDCA underdamped Langevin dynamics (MCSDCA-udLD) [39, 53, 54] where Markov chains are developed based on the underdamped Langevin diffusion.

In [39, 53, 54], the authors considered the optimization problem of deep learning,

$$\min f(x),$$

where f is a large sum of compositions between a neural network and a loss function, and x represents the network's trainable parameters. The advantage of using MCSDCA is its ability to minimize the smoothed version of the objective function $\tilde{f}$. This smoothed function aims to capture the main characteristics of f and is designed to generalize well on new, unseen data. In contrast, directly minimizing f might be challenging because to its rough nature and its tendency to exclusively reflect the training data.

In [53, 54], the authors pointed out the DC structure of $\tilde{u}(x,t)$ as follows,

$$\tilde{u}(x,t) = \underbrace{\frac{1}{2t}x^\top \mathcal{H} x}_{G(x)} - \underbrace{\epsilon \log \int_{\mathbb{R}^n} \exp\left(-\frac{1}{\epsilon}\left[f(x') + \frac{1}{2t}x'^\top \mathcal{H} x' - \frac{1}{t}x^\top \mathcal{H} x'\right]\right) dx'}_{H(x)}. \tag{4.12}$$

It is obvious that G and H are convex. The detailed MCSDCA-udLD is presented in Algorithm 14 [53, 54].

Algorithm 14 MCSDCA underdamped Langevin dynamics

Initialization. A starting point x^0, a sequence of positive numbers $\{\gamma_k\}$, a sequence of Markov chains' length $\{n_k\}$, the number of burn-in samples b, $\epsilon > 0$, $\delta > 0$, $t > 0$, set $k = 0$.
repeat
Set the starting state of the Markov chain $x_0^k := x^k$, and $v_0^k := 0$.
for $i = 0$ to $n_k - 2$ **do**
1. Receive a minibatch of data $D_{\text{minibatch}}$.
2. Compute a stochastic gradient $\tilde{\nabla} f(x_i^k)$ of f using $D_{\text{minibatch}}$.
3. Compute the conditional expectation of v_{i+1}^k and x_{i+1}^k given x_i^k and v_i^k as follows

$$\mathbb{E}(v_{i+1}^k) = e^{-2\delta} v_i^k - \frac{1}{2}(1 - e^{-2\delta}) \left(\tilde{\nabla} f(x_i^k) + \frac{1}{t}(x_i^k - x^k) \right),$$

$$\mathbb{E}(x_{i+1}^k) = x_i^k + \frac{1}{2}(1 - e^{-2\delta}) v_i^k - \frac{1}{2} \left(\delta - \frac{1}{2}(1 - e^{-2\delta}) \right) \left(\tilde{\nabla} f(x_i^k) + \frac{1}{t}(x_i^k - x^k) \right).$$

4. Sample $v_{i+1}^k \sim \mathbb{E}(v_{i+1}^k) + \sqrt{c_2}\mathcal{N}(0, I)$.
5. Sample $x_{i+1}^k | v_{i+1}^k \sim \mathbb{E}(x_{i+1}^k) + c_3 c_2^{-1}(v_{i+1}^k - \mathbb{E}(v_{i+1}^k)) + \sqrt{(c_1 - c_3^2 c_2^{-1})}\mathcal{N}(0, I)$.
end for
Compute $y^k = \dfrac{1}{n_k - b} \sum_{i=b}^{n_k - 1} x_i^k$.
Solve the following convex problem,

$$x^{k+1} = \arg\min_x \left\{ \frac{1}{2t}\|x\|^2 - \frac{1}{t}\langle x, y^k \rangle + \frac{\gamma_k}{2}\|x - x^k\|^2 \right\}, \tag{4.13}$$

which has the closed-form solution

$$x^{k+1} = \frac{t\gamma_k}{1 + t\gamma_k} x^k + \frac{1}{1 + t\gamma_k} y^k.$$

$k = k + 1$.
until Stopping criterion

4.2.4 Proposed cryptosystem

Consider a scenario where Alice (the sender) wants to send a confidential message to Bob (the receiver) securely without letting an eavesdropper, Eve, gain access to the information. In order to ensure secure communication, Alice would utilize the proposed framework as follows.

Training sets

- Alice trains the autoencoder, which consists of a single hidden layer (refer to Table 4.1) on a dataset of 1024 randomly generated 8-bit sequences. When selecting an autoencoder with several hidden layers (refer to Table 4.1), Alice proceeds to train this autoencoder using a dataset consisting of 16384 randomly generated 8-bit sequences. This is to make sure that all possible 8-bit combinations of 0s and 1s are in the dataset, along with some repetitions for an efficient and inclusive training process. For 100 epochs of training, Alice should observe a set of training weights.

- After the training is completed, the encoder network's weights and architecture act as the encryp-

tion key, while the decoder network acts as the decryption key.

- Alice sends the decoder network along with its weights (the decryption key) to Bob over a secure channel.

At this stage, both parties have access to the decryption key.

Encryption

- The 8-bit binary ASCII values of every character in Alice's secret message are fed as input into the trained encoder model.
- The encoder uses the trained weights to transform the 8-bit sequences into $n-$dimensional floating-point vectors for the autoencoder with one hidden layer or $8-$dimensional floating-point vectors for the autoencoder with more than one hidden layer. These vectors represent the encrypted data for each character.
- These encrypted vectors (also called as ciphertext) are then transmitted to Bob via a public channel.

Fig. 4.4 illustrates the GUI for text encryption using the proposed autoencoder. The user can choose a document (.doc or .docx) to encrypt, and the encrypted file is stored in .txt format.

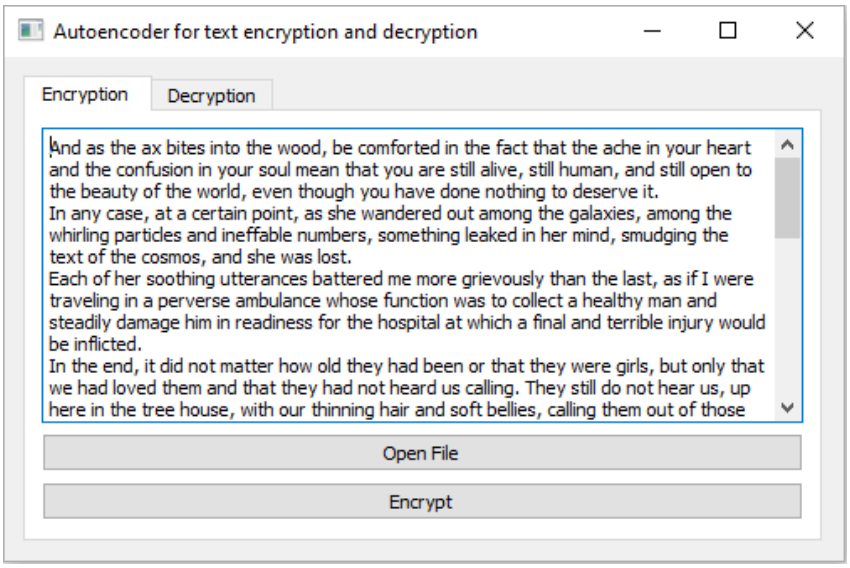

Figure 4.4: Function of text encryption in the GUI of the proposed autoencoder.

Decryption

- Bob receives the ciphertext and feeds them as input to his trained decoder model (shared previously by Alice).
- The decoder model uses its trained weights to recover the encrypted vector.
- Following the processing of each block, the output layer generates an array of 8-dimensional floating-point vectors. These vectors get discretized by the application of a threshold function,

as described in Equation (4.14). Subsequently, the discretized vectors are further turned into characters using the ASCII table.

$$\phi(x) = \begin{cases} 0, & \text{if } x < \epsilon \\ 1, & \text{if } x \geq 1 - \epsilon, \end{cases} \tag{4.14}$$

where ϵ is much smaller than 0.5, for example, $\epsilon = 0.05$. This implies that the autoencoders we have proposed have a high level of accuracy in decrypting numerical data.

Fig. 4.5 depicts the GUI for text decryption using the proposed autoencoder. The user can select a ciphertext file (.txt format) to decrypt, and the application displays the decrypted content.

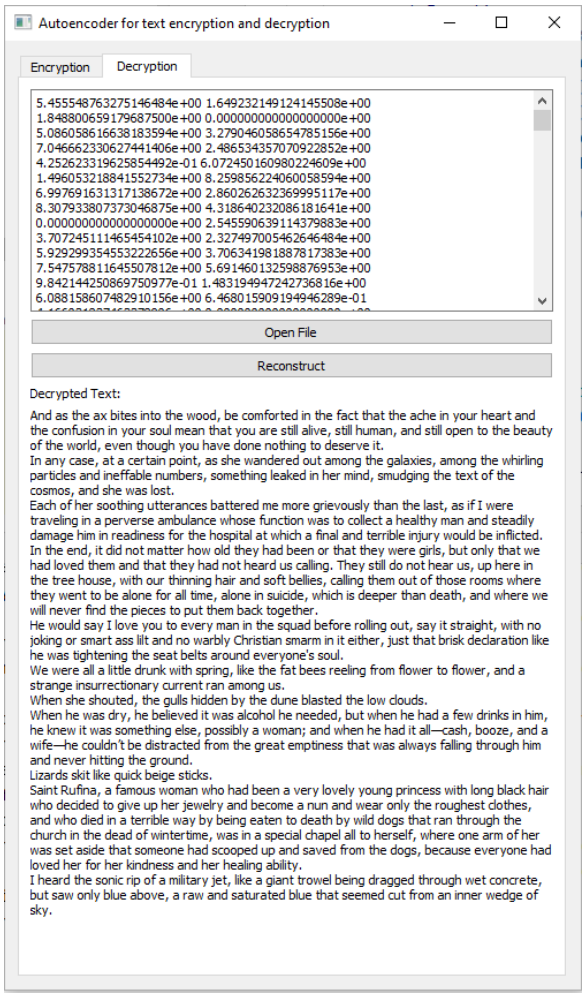

Figure 4.5: Function of text decryption in the GUI of the proposed autoencoder.

4.3 Numerical experiments

4.3.1 Experimental setups

The proposed system was implemented in Pytorch using a PC with an Intel i7-7700 CPU, 3.60GHz of 16GB RAM. The user interface of the autoencoder for text encryption and decryption was implemented using QWidget.

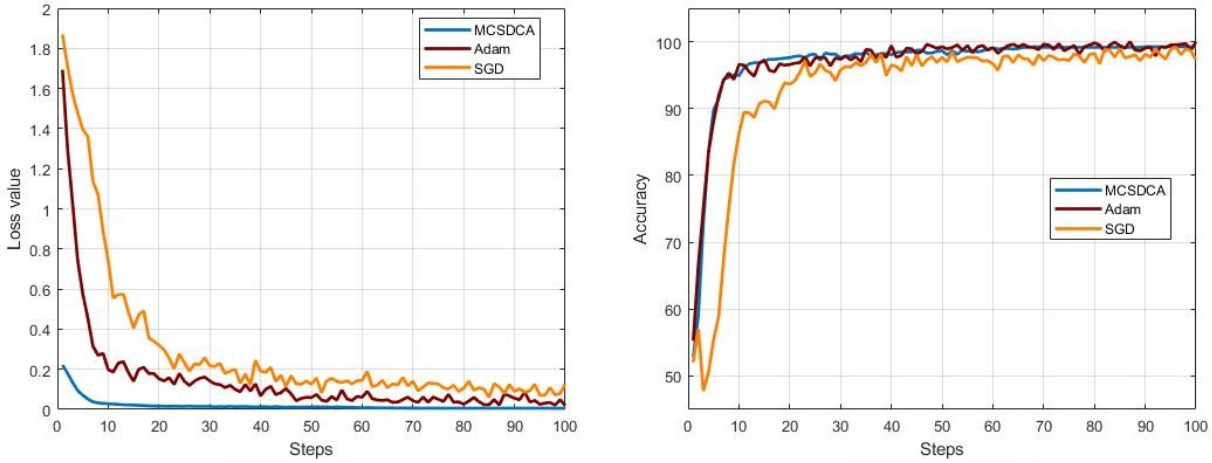

Figure 4.6: Training result over 100 epochs using autoencoder 1.

Comparative optimizers We will compare MCSDCA optimizer with Adam and SGD in eight autoencoder architectures.

Training procedure For the baseline algorithms (Adam, SGD), the learning rate is set as Adam(0.001), and SGD(0.01). For MCSDCA, all parameters are set to the same values as in [53, 54]. The autoencoder, which consists of a single hidden layer or more than one hidden layer (refer to Table 4.1), was first trained on a random set of 1024 or 16384, respectively, 8-bit sequences for multiple epochs and saved locally.

4.3.2 Experimental results

Training loss and testing accuracy

The benchmarking was done over 100 epochs using three optimizers, such as MCSDCA (blue line), Adam (red line), and SGD (orange line). There is a need for the system's performance to decrease after modifying all the weights inside the network. In these experiments, the accuracy of the autoencoder is measured by the complete similarity ratio of the output value and the input value for 8-bit sequences and is calculated in percentages.

Autoencoder with one hidden layer

In Fig. 4.6, 4.7, 4.8, and 4.9, we plot the training loss and training accuracy over 100 epochs using the autoencoder, which the number of neurons in the hidden layer is set to 7, 8, 9, and 10, respectively. Table 4.2 contains the training metrics for each optimizer. The loss value and accuracy of validation set and testing set are summarized in Table 4.3.

Comments on numerical results:

- The autoencoder 4, where the number of neurons in the hidden layer is $n = 10$, has the best performance. Using the MCSDCA optimizer, the loss value in the validation and testing sets is the lowest, and the accuracy is equal to 100% for both validation and testing sets. The testing loss value of MCSDCA is lower than that of Adam and SGD by 18.0% and 37.8%, respectively, in the autoencoder 4. At the same time, the testing accuracy of MCSDCA is 7.2% higher than that of SGD, and it is the

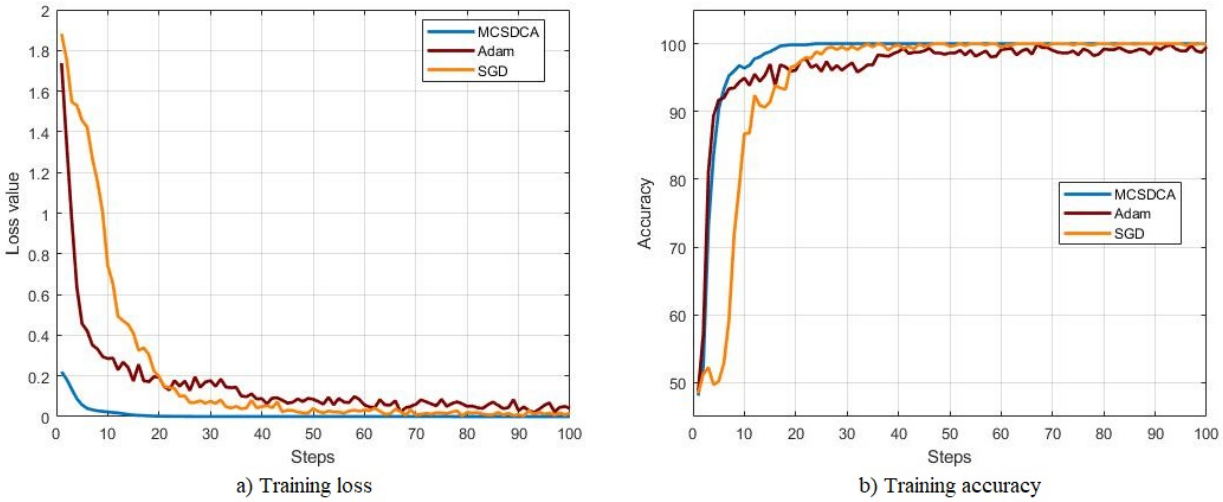

Figure 4.7: Training result over 100 epochs using autoencoder 2.

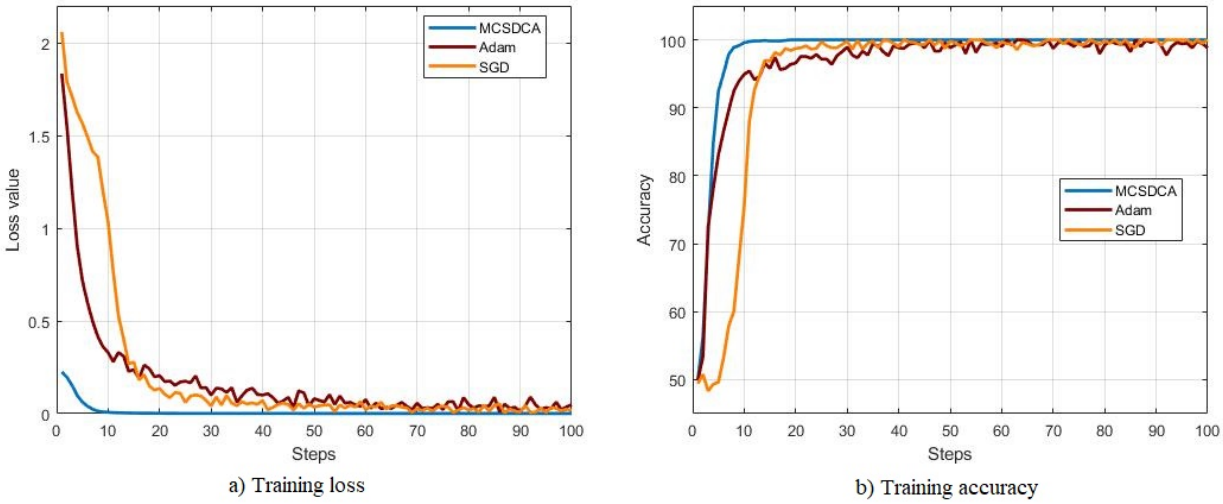

Figure 4.8: Training result over 100 epochs using autoencoder 3.

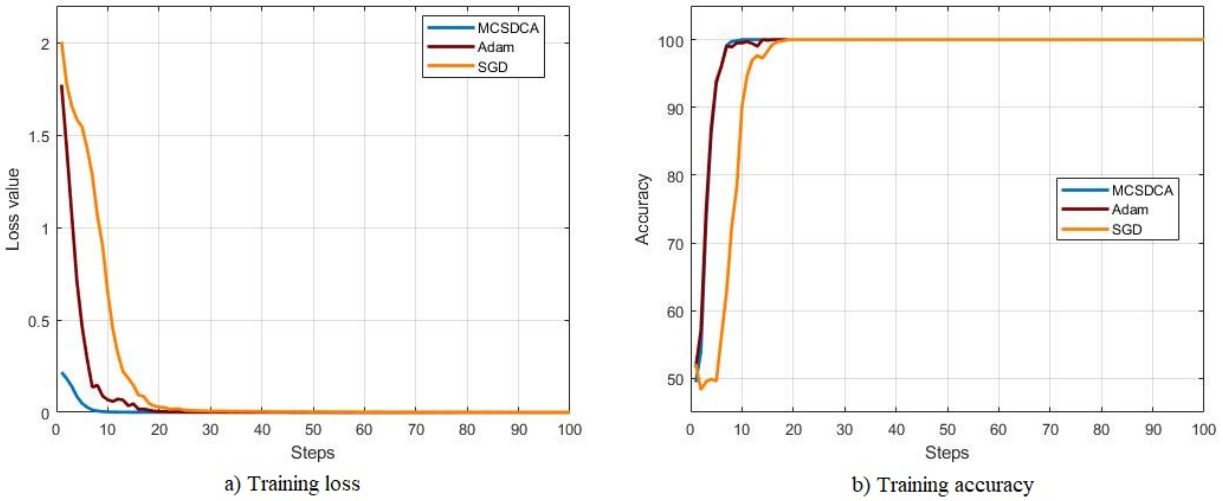

Figure 4.9: Training result over 100 epochs using autoencoder 4.

Table 4.2: Training metrics for each optimizer in autoencoders with one hidden layer. Bold values indicate the best results.

Optimizer	**Autoencoder**	**Epochs**	**Minimum training loss**	**Training accuracy**	**Training time (seconds)**
MCSDCA	Autoencoder1	100	0.005893	99.2347%	13.8540
	Autoencoder2		0.000085	**100%**	12.1445
	Autoencoder3		0.000063	**100%**	11.9253
	Autoencoder4		**0.000048**	**100%**	11.7743
Adam	Autoencoder1	100	0.019505	98.6784%	9.9375
	Autoencoder2		0.040183	99.5117%	9.7532
	Autoencoder3		0.000125	98.8281%	9.2765
	Autoencoder4		0.000104	**100%**	9.3412
SGD	Autoencoder1	100	0.127418	97.2656%	8.3172
	Autoencoder2		0.018794	99.9023%	**7.3776**
	Autoencoder3		0.001269	**100%**	7.6728
	Autoencoder4		0.001178	**100%**	7.5139

Table 4.3: Validation loss/accuracy for 2669 samples and Testing loss/accuracy for 15865 samples in autoencoders with one hidden layer. Bold values indicate the best results.

Optimizer	**Autoencoder**	**Validation loss**	**Validation accuracy**	**Testing loss**	**Testing accuracy**
MCSDCA	Autoencoder1	0.043335	95.4749%	0.037958	96.0164%
	Autoencoder2	0.000514	**100%**	0.001091	99.9622%
	Autoencoder3	0.000651	**100%**	0.000648	**100%**
	Autoencoder4	**0.001575**	**100%**	**0.001516**	**100%**
Adam	Autoencoder1	0.009220	**100%**	0.009390	99.9559%
	Autoencoder2	0.002763	99.8878%	0.004747	99.7416%
	Autoencoder3	0.001340	**100%**	0.000954	99.9685%
	Autoencoder4	0.002158	**100%**	0.001849	**100%**
SGD	Autoencoder1	0.330839	74.2708%	0.323979	75.6823%
	Autoencoder2	0.003865	**100%**	0.003583	**100%**
	Autoencoder3	0.001211	**100%**	0.001195	98.9874%
	Autoencoder4	0.002461	91.5436%	0.002438	93.8639%

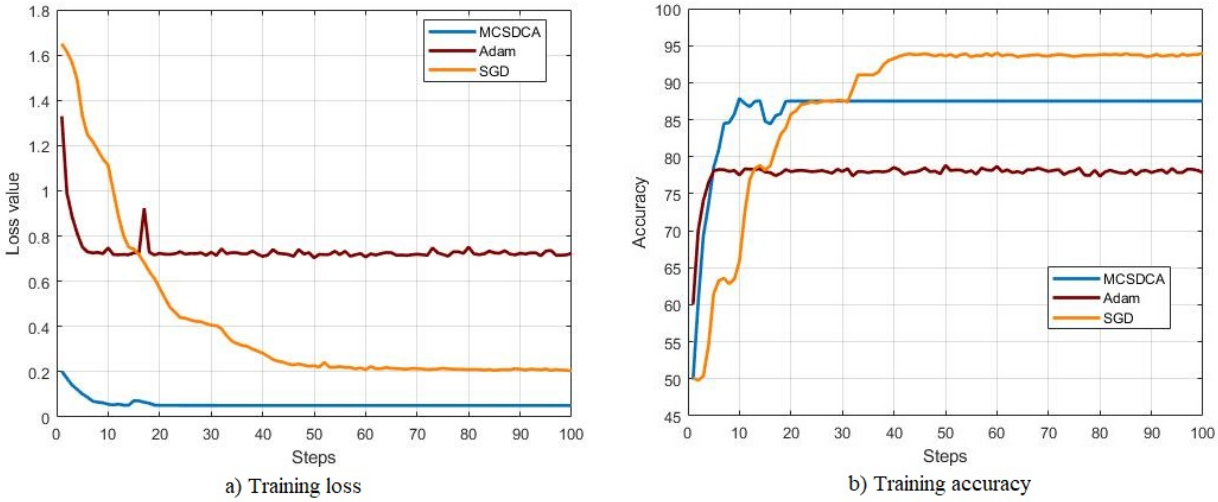

Figure 4.10: Training result over 100 epochs using autoencoder 5.

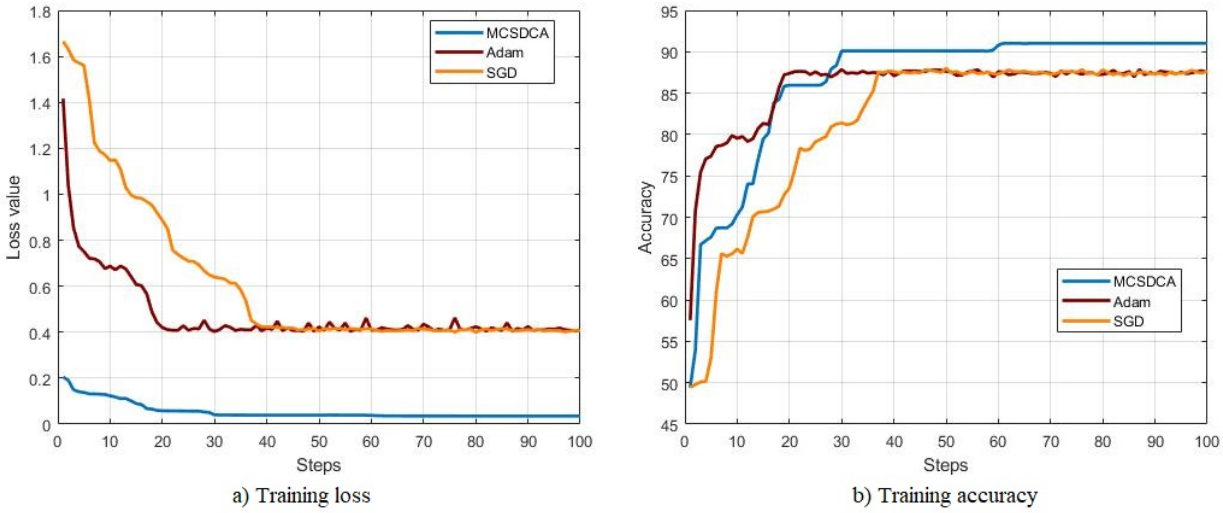

Figure 4.11: Training result over 100 epochs using autoencoder 6.

same with the testing accuracy of Adam when the number of neurons in the hidden layer is 10.

- Compared with Adam and SGD, the optimizer MCSDCA gives competitive results in terms of loss value and accuracy in validation and test data. However, the training time for Adam and SGD is approximately 1.3 and 1.6 times faster than that of MCSDCA.

- The autoencoder 1, where the number of neurons in the hidden layer is $n = 7$, has the worst performance. The validation and testing accuracy of this autoendoer using all three optimizers do not reach 100%. Therefore, it is important to thoroughly evaluate the autoencoder 1 before using it for the encryption and decryption of textual data.

Autoencoder with more than one hidden layer

In Fig. 4.10, 4.11, 4.12, and 4.13, we plot the training loss and training accuracy over 100 epochs using the autoencoder, where the number of neurons in the hidden layers is set to (7,8,7), (8,8,8), (9,8,9), and (10,8,10), respectively. Table 4.4 contains the training metrics for each optimizer. The loss value and accuracy of the validation set and testing set are summarized in Table 4.5.

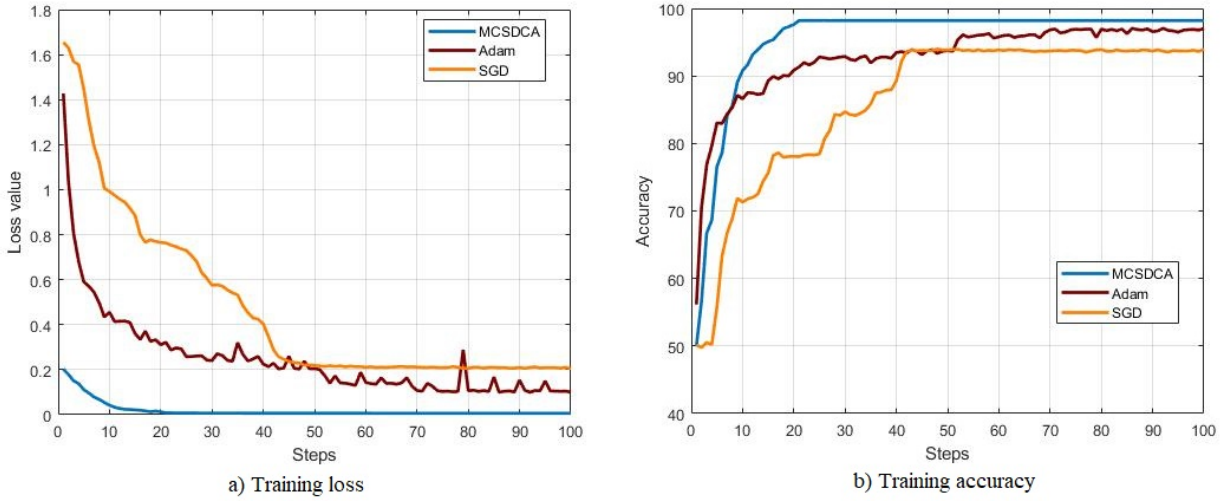

Figure 4.12: Training result over 100 epochs using autoencoder 7.

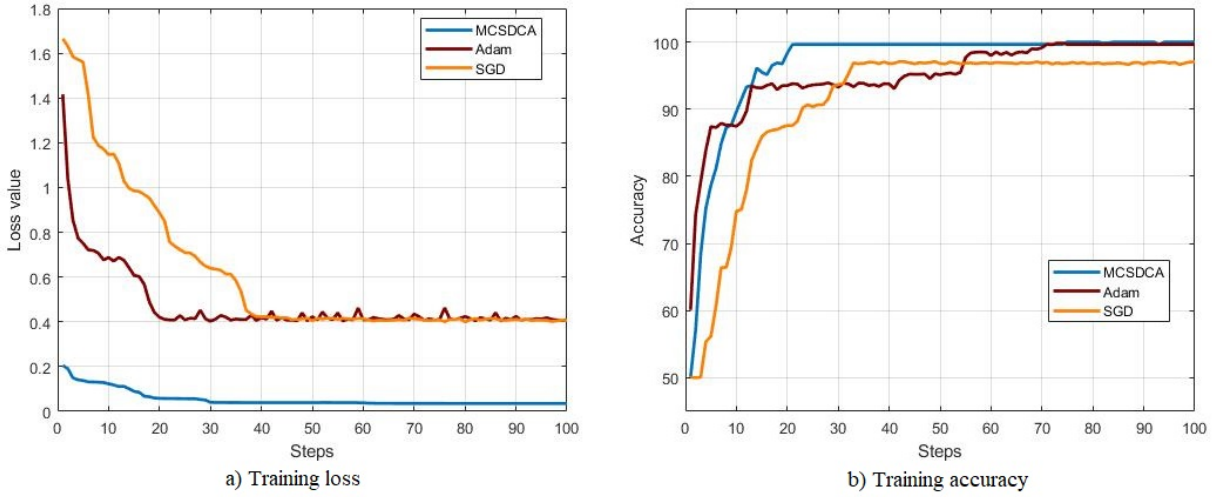

Figure 4.13: Training result over 100 epochs using autoencoder 8.

Table 4.4: Training metrics for each optimizer in autoencoders with more than one hidden layer. Bold values indicate the best results.

Optimizer	Autoencoder	Epochs	Minimum training loss	Training accuracy	Training time (seconds)
MCSDCA	Autoencoder5	100	0.050786	87.5121%	375.8933
	Autoencoder6		0.035224	91.0199%	390.9791
	Autoencoder7		0.006357	98.2068%	318.5714
	Autoencoder8		**0.000083**	**100%**	324.9840
Adam	Autoencoder5	100	0.723974	77.8564%	234.8839
	Autoencoder6		0.409104	87.5488%	199.3420
	Autoencoder7		0.100046	97.0276%	199.3461
	Autoencoder8		0.000125	**100%**	206.4019
SGD	Autoencoder5	100	0.203155	93.9758%	**118.0992**
	Autoencoder6		0.406008	87.6587%	153.7570
	Autoencoder7		0.203059	93.9697%	159.6565
	Autoencoder8		0.097949	97.0642%	160.5798

Table 4.5: Validation loss/accuracy for 2669 samples and Testing loss/accuracy for 15865 samples in autoencoders with more than one hidden layer. Bold values indicate the best results.

Optimizer	**Autoencoder**	**Validation loss**	**Validation accuracy**	**Testing loss**	**Testing accuracy**
MCSDCA	Autoencoder5	0.507406	75.1122%	0.510290	76.7438%
	Autoencoder6	0.317590	73.0366%	0.320740	72.8679%
	Autoencoder7	0.079572	87.5243%	0.077486	88.2288%
	Autoencoder8	**0.000787**	**100%**	**0.000770**	**100%**
Adam	Autoencoder5	0.642317	73.0516%	0.636030	73.6086%
	Autoencoder6	0.341578	72.8721%	0.340226	73.9552%
	Autoencoder7	0.155136	79.1249%	0.150515	80.3971%
	Autoencoder8	0.001728	**100%**	0.001541	**100%**
SGD	Autoencoder5	0.331465	71.1122%	0.328804	71.1976%
	Autoencoder6	0.338735	70.9274%	0.337335	71.6924%
	Autoencoder7	0.254032	72.2364%	0.254604	72.1273%
	Autoencoder8	0.067222	90.8527%	0.069528	90.4097%

Comments on numerical results:

- The autoencoder 8, where the number of neurons in each hidden layer is $(10, 8, 10)$, has the best performance. Using the MCSDCA optimizer, the loss value in the validation and testing sets is the lowest, and the accuracy is equal to 100% for both validation and testing sets. The testing loss value of MCSDCA is lower than that of Adam and SGD by 50.1% and 90.4%, respectively, in the autoencoder 8. At the same time, the testing accuracy of MCSDCA is 9.6% higher than that of SGD, and it is the same with the testing accuracy of Adam when the number of neurons in the hidden layer is $(10, 8, 10)$.

- Compared with Adam and SGD, the optimizer MCSDCA gives competitive results in terms of loss value and accuracy in validation and test data. However, the training time for Adam and SGD is approximately 1.7 and 2.4 times faster than that of MCSDCA.

- The autoencoders $5, 6,$ and 7 do not reach 100% in terms of validation and testing accuracy using all three optimizers. It is shown that adding multiple hidden layers to the autoencoder architecture for text encryption and decryption requires considerable analysis. Increasing the complexity of the autoencoder's structure does not always result in an improvement in the efficiency of the encryption and decryption processes.

Security analysis

As described in the section on the proposed cryptosystem, the plaintext cannot be decrypted unless the recipient has access to the trained decoder of the autoencoder. Consider a scenario where an eavesdropper intercepts the ciphertext on the public channel. To recover the plaintext, the eavesdropper must be aware of the autoencoder architecture and the trained weights of the decoder. Assuming that the eavesdropper is aware of the network's architecture, he or she will still require accurate knowledge of the weights assigned to each neuron in the decoding network. Even if the training dataset remains unchanged, the weight configurations determined at the conclusion of the training procedure are extremely variable due to the stochastic nature of the training phase. In addition, the training is conducted on a dataset chosen at random, which increases the unpredictability of the weights determined during training and makes it more difficult for an unauthorized interceptor to access confidential data.

In fact, the autoencoder was trained on a random sequence of 1024 bits for the autoencoder with one hidden layer or a sequence of 16384 bits for the autoencoder with more than one hidden layer for 100 epochs and saved locally. The proposed cryptosystem's key consists of the model architecture, the training dataset, and trainable parameters (including weights and biases). The number of trainable weights in the proposed autoencoder is quite large (refer to Table 4.1) and takes random values as real floating-point numbers. This exhaustive attack resistance is attributed to the model architecture, the large number of trainable parameters, and the stochastic nature by which these parameters are determined. Without a trained decoder, it is virtually impossible to decipher the ciphertext. Text is also encoded as floating-point vectors, which provides additional security by making decoding more difficult.

- Weight Space Analysis

Any cryptosystem's key space should be exceedingly large in order to give an attacker a very large number of key combinations from which to obtain the plaintext. In consideration of the fact that the primary element for the suggested autoencoder is a seed sequence, an analysis of the key space was conducted to assess the robustness of the proposed system against brute force attacks [76]. Furthermore, in the case that an attacker opts for breaking the system via a trial-and-error approach using the synaptic weights of the autoencoder, a probabilistic analysis was conducted to determine the potential methods in which the attacker may choose the correct weights of the neural network in [76].

For the proposed autoencoder with one hidden layer, the number of trainable variables (weights), i.e., the number of keys in the decoder/decryption system, is $64, 72, 80$ or 88 according to the chosen number of hidden neurons as $7, 8, 9$ or 10 (refer to Table 4.1). For the proposed autoencoder with more than one hidden layer, the number of trainable variables, i.e., the number of keys in the decoder/decryption system, is $127, 144, 161$ or 178 (refer to Table 4.1). The trainable weights are initialized with randomly generated 32-bit floating-point values. The 32-bit floating-point number is characterized by a minimum value of $1.175494351 \times 10^{-38}$ and a maximum value of $3.402823466 \times 10^{38}$.

As a result, the suggested weight space is large enough to withstand brute force attacks.

- Confusion (1 bit change) [57]

To evaluate the security of the proposed cryptosystem, the level of confusion in the encrypted 8-dimensional vectors was computed. To accomplish this, two 8-bit binary ASCII sequences (with a single-bit difference) were passed through the encoders that had been trained. The encrypted vectors obtained for each of the sequences are shown in Fig. 4.14 and Fig. 4.15.

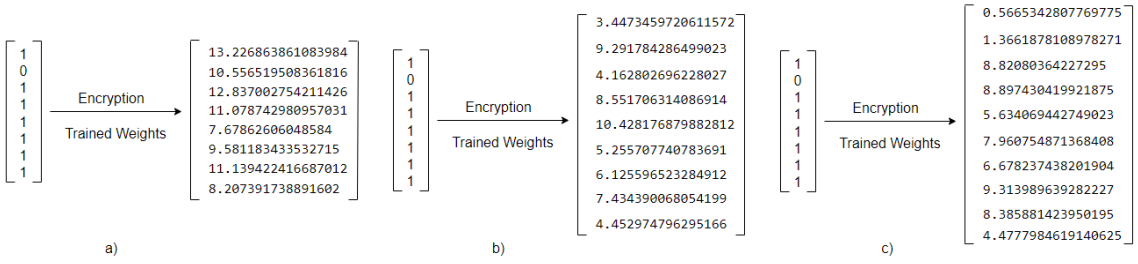

Figure 4.14: Encryption of 8-bit ASCII with trained weights of the autoencoder with one hidden layer that has a) 8, b) 9, c) 10 hidden neurons.

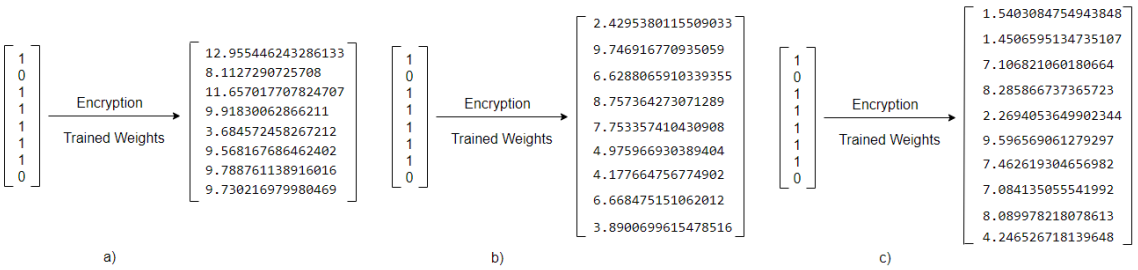

Figure 4.15: Encryption of chosen 8-bit ASCII with last bit changed in the autoencoder with one hidden layer that has a) 8, b) 9, c) 10 hidden neurons.

Even though the plaintexts differ by only one bit, none of the encrypted vectors' components are related. This indicates that cryptanalysis techniques based on similarity and frequency analysis will be rendered ineffective, indicating that this cryptosystem exhibits a high level of confusion.

4.4 Conclusion

In this chapter, we have proposed a deep learning-based approach using neural networks, which represents the next development in cryptography. The main aim of our work is to develop an autoencoder that is used to securely encrypt and decrypt text messages represented in an 8-bit format, which corresponds to an ASCII character. Eight different autoencoder architectures are proposed for text encryption and decryption. In these proposed autoencoders, we use the BCE loss function combined with the penalty function in order to obtain output values that are the same as the binary inputs. Moreover, we apply a novel stochastic algorithm known as Markov chain stochastic DCA (MCSDCA) [39, 53, 54] as the optimizer in eight autoencoder architectures. Experimental results demonstrate that MCSDCA is competitive with other baseline optimizers (e.g., Adam and SGD) in training the autoencoders. This suggests that the proposed method has a high degree of confidentiality because each training epoch always produces a different and large set of trained weights.

As future work, given that cryptography is utilized not only for text files but also for other multimedia formats, we will modify the proposed scheme so that it will handle images and audio data. Furthermore, there are a number of attacks on the cryptographic algorithms. Each attack employs unique methods and requires varying quantities of time to compromise the system. Additionally, each attack exploits information about the encryption system that is already known. In consideration of this, multiple methodologies will be explored in order to cryptanalyze the suggested system. In addition, since there are many programming languages that are specifically designed for the purpose of developing artificial intelligence applications, an investigation into the efficacy of the proposed system under various implementations will be conducted.

Chapter 5

Conclusion and perspectives

In this thesis, we have studied three optimization problems in cryptography: dynamic centralized group key management, the optimization problem of constructing Merkle tree in blockchain transaction system, and optimizing autoencoder architecture for text encryption and decryption. To our knowledge, there is no rigorous mathematical optimization model for solving these classes of problems in cryptography. The mentioned algorithmic designs are mainly based on logical/heuristic arguments.

In centralized group key management, we developed optimization approaches to the problem of updating group key in the LKH structure using two different techniques, namely batch insertion and batch rekeying. Using the batch insertion technique, it consists of finding a set of leaf nodes in a binary key tree to insert new members with minimal insertion cost and keeping the tree as balanced as possible. This is the first optimization model that simultaneously considers the insertion cost and the balance of the resulting key tree. The suggested optimization problem has binary variables and an objective function that is discontinuous. It is first equivalently formulated to eliminate the objective's step functions. Thanks to the recent results on exact penalty techniques in DC programming, the latter problem can be reformulated as a DC program. We then designed an efficient DCA for solving this problem. Furthermore, we proposed a more effective two-step algorithm that integrated an algorithm to find all leaf nodes in the first step, which simplifies the optimization model in the second step. This simplified problem can still be solved using the same DCA-based approach as the original problem. In addition, we proposed an optimization approach to the problem of batch deletion and insertion of members in CGKM. Our primary objective is to simultaneously minimize the cost of removing and adding nodes while maintaining a balanced tree. The suggested optimization problem has binary variables and an objective function that is discontinuous. Numerical experiments have been conducted to justify the merits of our proposed model as well as the corresponding DCA. It has been shown that our approach obtains a better trade-off between two considered criteria in comparison with existing algorithms.

In the Ethereum cryptocurrency system, we proposed an optimization model for constructing the Merkle tree based on the distribution of transactions. To our knowledge, this is the first mathematical model for solving this Merkle tree structure problem based on the transaction distribution. The suggested optimization problem is a binary quadratic program. Thanks to the recent results on exact penalty techniques in DC programming, the problem can be reformulated as a DC program. We then applied an efficient DCA to solve this problem. To overcome the difficulty in parameter estimation and improve the standard DCA, we applied the DCA-Like algorithm to the problem of constructing a tree structure with the minimal number of hash values required to update the account data associated with

each transaction. DCA-Like improves the standard DCA in both quality of solution and running time in large-scale settings. At the same time, we applied Accelerated DCA (ADCA) and Accelerated DCA-Like (ADCA-Like), which consist of incorporating Nesterov's acceleration technique into DCA and DCA-Like. Furthermore, we combined the divide-and-conquer algorithm with each of our optimization approaches (DCA, ADCA, DCA-Like, and ADCA-Like) to solve the problem of building a Merkle tree when the number of accounts is large. Numerical experiments have been conducted in order to confirm the benefits of our suggested model and the corresponding DCA based approaches.

In the context of neural network based cryptography, we developed an efficient symmetric-key encryption system with an autoencoder architecture that is used to securely encrypt and decrypt text messages represented in an 8-bit format, which corresponds to an ASCII (American Standard Code for Information Interchange) character. In eight proposed autoencoder architectures, we use the binary cross-entropy loss function combined with the penalty function in order to obtain output values that are the same as the binary inputs. Moreover, we apply a new stochastic algorithm called Markov chain stochastic DCA (MCSDCA) [39, 53, 54] as the optimizer in various autoencoder architectures. Numerical experiments show the virtues of MCSDCA in comparison with other baseline optimizers (such as Adam and SGD) in training the autoencoders. This suggests that the proposed method has a high degree of confidentiality because each training epoch always produces a different and large set of trained weights.

Our objective for future work is to identify further optimization problems in the field of cryptography so that we may develop optimization models and suggest efficient DC programming and DCA-based approaches to address these problems.

Bibliography

[1] ISO/IEC: 10118-1. Information technology - Security techniques - Hash-functions - Part 1: General, 2000.

[2] ISO/IEC: 11770-5. Information technology - Security techniques - Key management - Part 5: Group key management, 2020.

[3] ISO/IEC: 18033-1. Information technology - Encryption algorithms - Part 1: General, 2021.

[4] S. Agatonovic-Kustrin and R. Beresford. Basic concepts of artificial neural network (ANN) modeling and its application in pharmaceutical research. *Journal of pharmaceutical and biomedical analysis*, 22(5):717–727, 2000.

[5] M. Bačák and J. M. Borwein. On difference convexity of locally Lipschitz functions. *Optimization*, 60(8-9):961–978, 2011.

[6] P. Baldi. Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML workshop on unsupervised and transfer learning*, pages 37–49. JMLR Workshop and Conference Proceedings, 2012.

[7] W. Barker, W. Polk, and M. Souppaya. Getting ready for post-quantum cryptography: explore challenges associated with adoption and use of post-quantum cryptographic algorithms. *The Publications of NIST Cyber Security White Paper (DRAFT), CSRC, NIST, GOV*, 26, 2020.

[8] R. Barskar and M. Chawla. A survey on efficient group key management schemes in wireless networks. *Indian J. Sci. Technol*, 9(14):1–16, 2016.

[9] I. A. Basheer and M. Hajmeer. Artificial neural networks: fundamentals, computing, design, and application. *Journal of microbiological methods*, 43(1):3–31, 2000.

[10] D. J. Bernstein. Cache-timing attacks on AES. 2005.

[11] D. J. Bernstein and T. Lange. Post-quantum cryptography. *Nature*, 549(7671):188–194, 2017.

[12] U. Bodkhe, S. Tanwar, K. Parekh, P. Khanpara, S. Tyagi, N. Kumar, and M. Alazab. Blockchain for industry 4.0: A comprehensive review. *IEEE Access*, 8:79764–79800, 2020.

[13] M. U. Bokhari and Q. M. Shallal. A review on symmetric key encryption techniques in cryptography. *International journal of computer applications*, 147(10), 2016.

[14] J. Bolte, A. Daniilidis, A. Lewis, and M. Shiota. Clarke subgradients of stratifiable functions. *SIAM Journal on Optimization*, 18(2):556–572, 2007.

[15] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[16] S. Bruce. Applied Cryptography: Protocols, Algorithms, and Source Code in C. 2nd edition, 1996.

[17] T. Chen, Z. Li, Y. Zhu, J. Chen, X. Luo, J. C.-S. Lui, X. Lin, and X. Zhang. Understanding Ethereum via graph analysis. *ACM Trans. Internet Technol.*, 20(2):1–32, 2020.

[18] F. H. Clarke. *Optimization and nonsmooth analysis*. John Wiley & Sons, 1983.

[19] R. Correa, M. A. Lopez, and P. Pérez-Aros. Necessary and sufficient optimality conditions in DC semi-infinite programming. *SIAM Journal on Optimization*, 31(1):837–865, 2021.

[20] M. Elhoseny, H. Elminir, A. Riad, and X. Yuan. A secure data routing schema for WSN using elliptic curve cryptography and homomorphic encryption. *J King Saud Univ - Comput Inf Sci*, 28(3):262–275, 2016.

[21] D. Erhan, A. Courville, Y. Bengio, and P. Vincent. Why does unsupervised pre-training help deep learning? In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 201–208. JMLR Workshop and Conference Proceedings, 2010.

[22] K. Fukushima, S. Kiyomoto, T. Tanaka, and K. Sakurai. Optimization of group key management structure with a client join-leave mechanism. *Inf Process Manag*, 16:130–141, 2008.

[23] A. F. O. Gaffar, A. B. W. Putra, and R. Malani. The multi layer autoencoder neural network (ML-AENN) for encryption and decryption of text message. In *2019 5th International Conference on Science in Information Technology (ICSITech)*, pages 128–133. IEEE, 2019.

[24] W. Gao, W. G. Hatcher, and W. Yu. A survey of blockchain: Techniques, applications, and challenges. In *2018 27th international conference on computer communication and networks (ICCCN)*, pages 1–11. IEEE, 2018.

[25] J. Goshi and R. E. Ladner. Algorithms for dynamic multicast key distribution trees. In *Proceedings of the twenty-second annual symposium on Principles of distributed computing*, pages 243–251, 2003.

[26] M. Gregoriadis, R. Muth, and M. Florian. Analysis of Arbitrary content on blockchain-based systems using BigQuery. In *Companion Proceedings of the Web Conference 2022*, pages 478–487, 2022.

[27] P. Hillmann, M. Knüpfer, T. Guggemos, and K. Streit. Cake: An efficient group key management for dynamic groups. *arXiv preprint arXiv:2002.10722*, 2020.

[28] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[29] J.-B. Hiriart-Urruty and C. Lemaréchal. *Fundamentals of convex analysis*. Springer Science & Business Media, 2004.

[30] S. H. Islam and G. Biswas. A pairing-free identity-based two-party authenticated key agreement protocol for secure and efficient communication. *Journal of King Saud University-Computer and Information Sciences*, 29(1):63–73, 2017.

[31] D.-H. Je, H.-S. Kim, Y.-H. Choi, and S.-W. Seo. Dynamic configuration of batch rekeying interval for secure multicast service. In *2014 International Conference on Computing, Networking and Communications (ICNC)*, pages 26–30. IEEE, 2014.

[32] R. Jogdand and S. S. Bisalapur. Design of an efficient neural key generation. *International Journal of Artificial Intelligence & Applications (IJAIA)*, 2(1):60–69, 2011.

[33] N. Kshetri and J. Voas. Blockchain-enabled e-voting. *IEEE Software*, 35(4):95–99, 2018.

[34] V. Kumar, R. Kumar, and S. K. Pandey. A computationally efficient centralized group key distribution protocol for secure multicast communications based upon RSA public key cryptosystem. *J King Saud Univ - Comput Inf Sci*, 32(9):1081–1094, 2020.

[35] H. M. Le. *Contribution au développement des méthodes avancées de la programmation DC et DCA pour certaines classes de problèmes d'optimisation non-convexes. Applications en apprentissage automatique*. Habilitation à diriger des recherches, Université de Lorraine, Mar. 2022.

[36] H. A. Le Thi, T. P. Dinh, H. M. Le, and X. T. Vo. DC approximation approaches for sparse optimization. *European Journal of Operational Research*, 244(1):26–46, 2015.

[37] H. A. Le Thi, V. N. Huynh, and T. Pham Dinh. DC programming and DCA for general DC programs. In *Advanced Computational Methods for Knowledge Engineering*, pages 15–35. Springer, 2014.

[38] H. A. Le Thi, V. N. Huynh, and T. Pham Dinh. Error bounds via exact penalization with applications to concave and quadratic systems. *Journal of Optimization Theory and Applications*, 171(1):228–250, 2016.

[39] H. A. Le Thi, V. N. Huynh, T. Pham Dinh, and H. P. H. Luu. Stochastic difference-of-convex-functions algorithms for nonconvex programming. *SIAM Journal on Optimization*, 32(3):2263–2293, 2022.

[40] H. A. Le Thi, H. M. Le, and T. Pham Dinh. Feature selection in machine learning: an exact penalty approach using a Difference of Convex function Algorithm. *Machine Learning*, 101:163–186, 2015.

[41] H. A. Le Thi, H. M. Le, D. N. Phan, and B. Tran. Novel DCA based algorithms for a special class of nonconvex problems with application in machine learning. *Applied Mathematics and Computation*, 409:125904, 2021.

[42] H. A. Le Thi and T. T. T. Nguyen. Solving the problem of batch deletion and insertion members in the Logical Key Hierarchy structure by a DC programming approach. *arXiv preprint arXiv:2305.10131*, 2023.

[43] H. A. Le Thi, T. T. T. Nguyen, and H. P. H. Luu. A DC programming approach for solving a centralized group key management problem. *Journal of Combinatorial Optimization*, 44(5):3165–3193, 2022.

[44] H. A. Le Thi and T. Pham Dinh. A continuous approach for globally solving linearly constrained quadratic zero-one programming problems. *Optimization*, 50(1-2):93–120, 2001.

[45] H. A. Le Thi and T. Pham Dinh. The DC (difference of convex functions) programming and DCA revisited with DC models of real world nonconvex optimization problems. *Annals of operations research*, 133(1-4):23–46, 2005.

[46] H. A. Le Thi and T. Pham Dinh. DC programming and DCA: thirty years of developments. *Mathematical Programming, Special Issue dedicated to : DC Programming - Theory, Algorithms and Applications*, 169(1):5–68, 2018.

[47] H. A. Le Thi and T. Pham Dinh. Open issues and recent advances in DC programming and DCA. *Journal of Global Optimization*, pages 1–58, 2023.

[48] H. A. Le Thi, T. Pham Dinh, and V. N. Huynh. Exact penalty and error bounds in DC programming. *Journal of Global Optimization*, 52(3):509–535, 2012.

[49] A. Levitin. *Introduction To the Design And Analysis Of Algorithms, 3rd edition*. Pearson Education, Inc., 2012.

[50] X. S. Li, Y. R. Yang, M. G. Gouda, and S. S. Lam. Batch rekeying for secure group communications. In *Proceedings of the 10th international conference on World Wide Web*, pages 525–534, 2001.

[51] H.-Y. Lin, M.-Y. Hsieh, and K.-C. Li. The cluster-based key management mechanism with secure data transmissions scheme in wireless sensor networks. *DEStech Transactions on Engineering and Technology Research, AMMA*, 2017.

[52] H. Lu. A novel high-order tree for secure multicast key management. *IEEE Trans Comput*, 54(2):214–224, 2005.

[53] H. P. H. Luu. *Advanced machine learning techniques based on DCA and applications to predictive maintenance*. Theses, Université de Lorraine, Oct. 2022.

[54] H. P. H. Luu, H. M. Le, and H. A. Le Thi. Markov chain stochastic DCA and applications in deep learning with PDEs regularization. *Neural Networks*, 2023.

[55] T. Medina and A. Miranda. Comparison of algorithms based cryptography symmetric DES, AES and 3DES. *Rev. Mundo Fesc*, 9:14–21, 2015.

[56] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone. *Handbook of applied cryptography*. CRC press, 2018.

[57] U. Menon, A. R. Menon, A. Hudlikar, A. Sharmila, and P. Mahalakshmi. A hybrid autoencoder architecture for text encryption. In *2021 Innovations in Power and Advanced Computing Technologies (i-PACT)*, pages 1–7. IEEE, 2021.

[58] R. C. Merkle. A digital signature based on a conventional encryption function. In *Conference on the theory and application of cryptographic techniques*, pages 369–378. Springer, 1987.

[59] A. Mizrahi, N. Koren, and O. Rottenstreich. Optimizing Merkle proof size for blockchain transactions. In *2021 International Conference on COMmunication Systems & NETworkS (COMSNETS)*, pages 299–307. IEEE, 2021.

[60] L. Morales, I. Sudborough, M. Eltoweissy, and M. Heydari. Combinatorial optimization of multicast key management. In *Proceedings of the 36th Annual Hawaii International Conference on System Sciences*, 2003. 9 pages.

[61] M. J. Moyer, J. Rao, and P. Rohatgi. Maintaining Balanced Key Trees for Secure Multicast. Internet-draft, Internet Engineering Task Force, 1999. 16 pages.

[62] J. L. Muñoz, J. Forné, O. Esparza, and M. Rey. Efficient certificate revocation system implementation: Huffman Merkle Hash Tree (HuffMHT). In *International Conference on Trust, Privacy and Security in Digital Business*, pages 119–127. Springer, 2005.

[63] S. Nakamoto. A peer-to-peer electronic cash system. *https://bitcoin.org/bitcoin.pdf*, 2008.

[64] A. Ng et al. Sparse autoencoder. *CS294A Lecture notes*, 72(2011):1–19, 2011.

[65] W. H. D. Ng, M. Howarth, Z. Sun, and H. Cruickshank. Dynamic balanced key tree management for secure multicast communications. *IEEE Trans Comput*, 56(5):590–605, 2007.

[66] T. T. T. Nguyen, H. A. Le Thi, and X. V. Doan. Optimizing Merkle Tree Structure for Blockchain Transactions by a DC Programming Approach. In *International Conference on Computational Collective Intelligence*, pages 405–417. Springer, 2023.

[67] T. T. T. Nguyen, H. P. H. Luu, and H. A. Le Thi. Solving a Centralized Dynamic Group Key Management Problem by an Optimization Approach. In *Le Thi, H.A., Pham Dinh, T., Le, H.M. (Eds.) Modelling, Computation and Optimization in Information Systems and Management Sciences. MCO 2021. Lecture Notes in Networks and Systems*, volume 363, pages 375–385. Springer, 2022.

[68] P. D. Nhat, H. M. Le, and H. A. Le Thi. Accelerated difference of convex functions algorithm and its application to sparse binary logistic regression. In *IJCAI*, pages 1369–1375, 2018.

[69] A. S. Pande and R. C. Thool. Survey on Logical Key Hierarchy for secure group communication. In *2016 international conference on automatic control and dynamic optimization techniques (ICACDOT)*, pages 1131–1136. IEEE, 2016.

[70] J. Pegueroles and F. Rico-Novella. Balanced batch LKH: New proposal, implementation and performance evaluation. In *Proceedings of the Eighth IEEE Symposium on Computers and Communications. ISCC 2003*, pages 815–820. IEEE, 2003.

[71] T. Pham Dinh and H. A. Le Thi. Convex analysis approach to DC programming: theory, algorithms and applications. *ACTA mathematica vietnamica*, 22(1):289–355, 1997.

[72] T. Pham Dinh and H. A. Le Thi. A DC optimization algorithm for solving the trust-region subproblem. *SIAM J Optim*, 8(2):476–505, 1998.

[73] T. Pham Dinh and H. A. Le Thi. Recent advances in DC programming and DCA. *Transactions on computational intelligence*, 8342:1–37, 2014.

[74] T. Pham Dinh, C. N. Nguyen, and H. A. Le Thi. An efficient combined DCA and B&B using DC/SDP relaxation for globally solving binary quadratic programs. *J Glob Optim*, 48(4):595–632, 2010.

[75] T. Pham Dinh and E. B. Souad. Algorithms for solving a class of nonconvex optimization problems. Methods of subgradients. In *North-Holland Mathematics Studies*, volume 129, pages 249–271. Elsevier, 1986.

[76] F. Quinga-Socasi, L. Zhinin-Vera, and O. Chang. A Deep Learning Approach for Symmetric-Key Cryptography System. In *Proceedings of the Future Technologies Conference (FTC) 2020*, volume 1, pages 539–552. Springer, 2020.

[77] S. Rafaeli and D. Hutchison. A survey of key management for secure group communication. *ACM Computing Surveys (CSUR)*, 35(3):309–329, 2003.

[78] R. T. Rockafellar. *Convex analysis*. Princeton university press, 1970.

[79] R. T. Rockafellar. *The theory of subgradients and its applications to problems of optimization: Convex and nonconvex functions*. Berlin Heldermann., 1981.

[80] W. Rudin. *Principles of mathematical analysis*, volume 3. McGraw-hill, New York, 1964.

[81] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning Internal Representations by Error Propagation*, page 318–362. MIT Press, Cambridge, MA, USA, 1986.

[82] N. Sakamoto. An efficient structure for LKH key tree on secure multicast communications. In *15th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pages 1–7. IEEE, 2014.

[83] R. Seetha and R. Saravanan. A survey on group key management schemes. *Cybernetics and information technologies*, 15(3):3–25, 2015.

[84] A. T. Sherman and D. A. McGrew. Key establishment in large dynamic groups using one-way function trees. *IEEE transactions on Software Engineering*, 29(5):444–458, 2003.

[85] P. I. Singh and P. Shende. Symmetric key cryptography: Current trends. 2014.

[86] R. Sobti and G. Geetha. Cryptographic hash functions: A review. *International Journal of Computer Science Issues (IJCSI)*, 9(2):461, 2012.

[87] F. Q. Socasi, R. Velastegui, L. Zhinin-Vera, R. Valencia-Ramos, F. Ortega-Zamorano, and O. Chang. Digital Cryptography Implementation using Neurocomputational Model with Autoencoder Architecture. In *ICAART 2020-12th International Conference on Agents and Artificial Intelligence*, volume 2, pages 865–872, 2020.

[88] S. Somin, G. Gordon, and Y. Altshuler. Social signals in the Ethereum trading network, 2018. Preprint at `https://arxiv.org/abs/1805.12097`.

[89] Y. Sun, M. Chen, A. Bacchus, and X. Lin. Towards collusion-attack-resilient group key management using one-way function tree. *Computer Networks*, 104:16–26, 2016.

[90] E. Swathi, G. Vivek, and G. S. Rani. Role of hash function in cryptography. *Int. J. Adv. Eng. Res. Sci.(IJAERS)*, 2016.

[91] E. Tromer, D. A. Osvik, and A. Shamir. Efficient cache attacks on AES, and countermeasures. *Journal of Cryptology*, 23:37–71, 2010.

[92] C. Vasudev. *Graph theory with applications*. New Age International, India, 2006.

[93] P. Vijayakumar, S. Bose, and A. Kannan. Rotation based secure multicast key management for batch rekeying operations. *Netw Sci*, 1(1-4):39–47, 2012.

[94] D. Wallner, E. Harder, R. Agee, et al. Key management for multicast: Issues and architectures. Technical report, RFC 2627, 1999.

[95] R. Wang, H. Wang, E. Dubrova, and M. Brisfors. Advanced far field EM side-channel attack on AES. In *Proceedings of the 7th ACM on Cyber-Physical System Security Workshop*, pages 29–39, 2021.

[96] C. K. Wong, M. Gouda, and S. S. Lam. Secure group communications using key graphs. *IEEE/ACM transactions on networking*, 8(1):16–30, 2000.

[97] G. Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151:1–32, 2014.

[98] Y. R. Yang, X. S. Li, X. B. Zhang, and S. S. Lam. Reliable group rekeying: A performance analysis. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 27–38, 2001.

[99] X. B. Zhang, S. S. Lam, D.-Y. Lee, and Y. R. Yang. Protocol design for scalable and reliable group rekeying. *IEEE ACM Trans Netw*, 11(6):908–922, 2003.

[100] S. Zhu, S. Setia, and S. Jajodia. Performance optimizations for group key management schemes. In *23rd International Conference on Distributed Computing Systems, 2003. Proceedings.*, pages 163–171. IEEE, 2003.

Résumé

La cryptographie, l'art et la science de la communication sécurisée, a une longue et illustre histoire. Elle utilise des algorithmes mathématiques pour convertir des messages en clair en texte chiffré illisible, protégeant ainsi la confidentialité, l'intégrité et l'authenticité des données. Cette thèse vise à déployer des techniques d'optimisation avancées pour résoudre certaines classes de problèmes en cryptographie. Le thème principal de la thèse est d'appliquer des approches efficaces basées sur la programmation DC (Différence de fonctions Convexes) et DCA (algorithme DC) pour résoudre les problèmes de gestion dynamique centralisée des clés de groupe, la construction de l'arbre de Merkle dans les systèmes de transaction blockchain, et l'architecture autoencoder pour le cryptage et le décryptage de texte.

La thèse se compose de cinq chapitres. Le chapitre 1 présente des préliminaires sur la programmation DC et DCA, ainsi que sur la cryptographie. Le chapitre 2 étudie le problème de la mise à jour de la clé de groupe lorsque l'appartenance change dynamiquement dans la gestion centralisée des clés de groupe avec deux techniques : l'insertion par lots et la recomposition par lots. Nous proposons des modèles d'optimisation pour minimiser le coût de la remise en clé et maintenir l'arbre aussi équilibré que possible en même temps. Les deux objectifs importants mentionnés sont combinés dans un modèle d'optimisation unifié dont la fonction objective contient des fonctions discontinues avec des variables binaires, ce qui est connu pour être NP-hard. Nous reformulons le problème comme un programme DC standard qui peut être résolu efficacement par DCA. De plus, les nœuds d'insertion et de suppression doivent être des nœuds feuilles, nous introduisons un algorithme en deux étapes pour réduire la complexité du modèle. Dans le chapitre 3, nous proposons un modèle d'optimisation pour résoudre le problème de la construction d'une structure d'arbre de Merkle basée sur la distribution des transactions Ethereum. L'objectif est de minimiser le nombre de valeurs de hachage nécessaires pour mettre à jour les données de compte associées à chaque transaction et d'assurer l'intégrité des données dans le système Ethereum. En utilisant les techniques de pénalités exactes, nous reformulons ce programme quadratique binaire en un programme DC conventionnel qui peut être résolu efficacement par le DCA. Pour obtenir une meilleure approximation convexe de la fonction objective sans connaître la décomposition DC, nous appliquons DCA-Like, une nouvelle extension de DCA. En outre, nous déployons la DCA accélérée (ADCA) et la DCA-Like accélérée (ADCA-Like) pour améliorer la DCA et la DCA-Like en y incorporant la technique d'accélération de Nesterov. Nous combinons séparément les approches DCA, ADCA, DCA-Like et ADCA-Like avec l'algorithme "diviser pour régner" pour résoudre le problème lorsque le nombre de comptes est élevé. Un autoencodeur efficace et sûr pour le cryptage et le décryptage de texte est examiné au chapitre 4. Il s'agit d'une approche basée sur l'apprentissage profond utilisant des réseaux neuronaux, qui présente un degré élevé de confidentialité et représente le prochain développement de la cryptographie. Nous appliquons un nouvel algorithme stochastique appelé DCA stochastique à chaîne de Markov (MCSDCA) comme optimiseur dans diverses architectures d'autoencodeurs. Enfin, le chapitre 5 conclut la thèse.

Mots-clés: Gestion centralisée des clés de groupe, L'arbre de Merkle, transaction de la blockchain, Autoencoder, Programmation DC et DCA

Abstract

Cryptography, the art and science of secure communication, has a long, illustrious history. It uses mathematical algorithms to convert original messages into unreadable ciphertext, thereby protecting the data's confidentiality, integrity, and authenticity. Cryptography plays a crucial role in securing our information systems in the interconnected world of today, where immense quantities of sensitive data are transmitted and stored electronically. This thesis aims to deploy advanced optimization techniques for solving certain classes of problems in Cryptography. The main theme of the thesis is to propose the optimization model and apply efficient approaches based on DC (Difference of Convex functions) programming and DCA (DC Algorithm) to solve the problems of dynamic centralized group key management, Merkle tree construction in blockchain transaction systems, and autoencoder architecture for text encryption and decryption.

The thesis consists of five chapters. Preliminaries on DC programming and DCA, and cryptography are presented in Chapter 1. Chapter 2 studies the problem of updating the group key when membership changes dynamically in centralized group key management with two techniques: batch insertion and batch rekeying. We propose optimization models for minimizing the rekeying cost and keeping the tree as balanced as possible at the same time. The two mentioned important objectives are combined into a unified (deterministic) optimization model whose objective function contains discontinuous step functions with binary variables, which is known to be NP-hard. We reformulate the problem as a standard DC program that can be solved efficiently by DCA. Moreover, the insertion and deletion nodes must be the leaf nodes, we introduce a two-step algorithm to reduce the model's complexity. In Chapter 3, we propose an optimization model for solving the problem of constructing a Merkle tree structure based on the Ethereum transaction distribution. The objective is to minimize the number of hash values required to update the account data associated with each transaction and to ensure the integrity of data in the Ethereum system. Using the exact penalty techniques, we reformulate this binary quadratic program as a conventional DC program that is efficiently solvable by the DCA. To get better convex approximation of the objective function without knowing a DC decomposition, DCA-Like, a novel extension of DCA, is applied. Furthermore, we deploy Accelerated DCA (ADCA) and Accelerated DCA-Like (ADCA-Like) to improve DCA and DCA-Like by incorporating the Nesterov's acceleration technique into it. We separately combine DCA, ADCA, DCA-Like, and ADCA-Like approaches with the divide-and-conquer algorithm to solve the problem when the number of accounts is large. An efficient and secure autoencoder for text encryption and decryption is discussed in Chapter 4. This is a deep learning-based approach using neural networks, which has a high degree of confidentiality and represents the next development in cryptography. We apply a new stochastic algorithm called Markov chain stochastic DCA (MCSDCA) as the optimizer in various autoencoder architectures. Finally, Chapter 5 brings the thesis to a close.

Keywords: Centralized group key management, Merkle tree, blockchain transaction, Autoencoder, DC (Difference of Convex functions) programming and DCA (DC Algorithms)