ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Αξιολόγηση και παραλληλοποίηση αλγορίθμων μίξης για χρήση σε συστήματα ηλεκτρονικών ψηφοφοριών

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Κωνσταντίνου Ι. Μαμασούλα

Επιβλέπων: Παναγιώτης Τσανάκας
Καθηγητής ΕΜΠ

Αθήνα, Μάρτιος 2015

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ
ΣΥΣΤΗΜΑΤΩΝ

Αξιολόγηση και παραλληλοποίηση αλγορίθμων μίξης για χρήση σε συστήματα ηλεκτρονικών ψηφοφοριών

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Κωνσταντίνου Ι. Μαμασούλα

Επιβλέπων: Παναγιώτης Τσανάκας
Καθηγητής ΕΜΠ

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 27η Μαρτίου 2015.

.....	.....	.....
Παναγιώτης Τσανάκας Καθηγητής ΕΜΠ	Νεκτάριος Κοζύρης Καθηγητής ΕΜΠ	Γεώργιος Γκούμας Λέκτορας ΕΜΠ

Αθήνα, Μάρτιος 2015

.....
Κωνσταντίνος Ι. Μαμασούλας
Διπλωματούχος Ηλεκτρολόγος Μηχανικός & Μηχανικός Υπολογιστών

Abstract

The present thesis is a result of studying the various mixing algorithms designed to meet the needs of a modern E-voting system. There is a lot of effort in this field today and it is quite interesting to look deeper into what science and technology combined together has to offer us.

Initially, the general concept of electronic voting is presented, along with the principles that need to be followed if anyone attempts to construct a complete electronic voting system in the real world. Furthermore, the advantages and disadvantages of such systems are stressed, to point out the progress so far and the challenges waiting to be dealt with in the future.

Next, all the necessary cryptographic primitives and methods are explained in order to introduce the main scope of the present thesis: the mixing algorithms involved in an E-voting protocol. In particular, a survey was carried out in which several algorithms were chosen and evaluated, in terms of computational complexity. In order to improve the running time of those algorithms, a significant part of this thesis is the distribution of their computational load among parallel tasks.

From all the algorithms included in the survey, that of Furukawa and Sako was chosen to be implemented and tested with the view of replacing the current algorithm used in the Zeus e-voting system. The specific implementation was slightly optimized and, in the end, was extended to embody the functionality of parallel execution.

Finally, the experimental results from the execution of the aforementioned algorithm constitute the last part of the present thesis, which lead to several conclusions about the mixing phase of an E-voting protocol, as well as to a general theoretical model that determines a practical number of CPUs needed for the parallel execution.

Keywords: Electronic voting, mixnet, permutation, Zero-Knowledge Proof, parallel execution, scalability, speedup.

Περίληψη

Η παρούσα εργασία είναι αποτέλεσμα μελέτης των διαφόρων αλγορίθμων μίξης οι οποίοι σχεδιάστηκαν για να συναντούν τις ανάγκες των σύγχρονων συστημάτων ηλεκτρονικών εκλογών. Η προσπάθεια στον τομέα αυτό είναι μεγάλη σήμερα και είναι αρκετά ενδιαφέρον να κοιτάξει κανείς βαθύτερα στο τι η επιστήμη σε συνδυασμό με την τεχνολογία έχουν να μας προσφέρουν.

Αρχικά, παρουσιάζεται η έννοια των ηλεκτρονικών εκλογών, μαζί με τις αρχές που πρέπει να ακολουθούνται αν οποιοσδήποτε επιχειρήσει να κατασκευάσει ένα πλήρες σύστημα ηλεκτρονικών εκλογών στην πραγματικό κόσμο. Επιπλέον, τονίζονται τα πλεονεκτήματα και τα μειονεκτήματα τέτοιων συστημάτων, ώστε να επισημανθεί η μέχρι τώρα πρόοδος και οι προκλήσεις που περιμένουν να αντιμετωπιστούν.

΄Επειτα, όλες οι απαραίτητες κρυπτογραφικές βάσεις και μέθοδοι εξηγούνται, για να εισαγάγουν το κύριο πεδίο της παρούσας εργασίας: τους αλγορίθμους μίξης που εμπλέκονται σε ένα πρωτόκολλο ηλεκτρονικών εκλογών. Πιο συγκεκριμένα, διεξήχθη μια έρευνα στην οποία επιλέχθηκαν και αξιολογήθηκαν αρκετοί αλγόριθμοι, σε όρους υπολογιστικής πολυπλοκότητας. Με σκοπό να βελτιωθεί ο χρόνος εκτέλεσης αυτών των αλγορίθμων, ένα σημαντικό κομμάτι αυτής της εργασίας είναι η κατανομή του υπολογιστικού τους φορτίου σε παράλληλες εργασίες.

Από όλους τους αλγορίθμους που περιλαμβάνονται στην έρευνα, εκείνος των Furukawa και Sako επιλέχθηκε για να υλοποιηθεί και να δοκιμαστεί με προοπτική αντικατάστασης του τωρινού αλγορίθμου που χρησιμοποιείται στο ηλεκτρονικό σύστημα ψηφοφοριών Ζευς. Η συγκεκριμένη υλοποίηση βελτιστοποιήθηκε ελαφρώς και, στο τέλος, επεκτάθηκε ώστε να ενσωματώσει τη λειτουργικότητα της παράλληλης εκτέλεσης.

Τέλος, τα πειραματικά αποτελέσματα από την εκτέλεση του προαναφερθέντος αλγορίθμου αποτελούν το τελευταίο κομμάτι της παρούσας εργασίας, τα οποία οδηγούν σε αρκετά συμπεράσματα για τη διαδικασία μίξης ενός πρωτοκόλλου ηλεκτρονικών εκλογών, καθώς επίσης και σε ένα γενικό θεωρητικό μοντέλο το οποίο καθορίζει ένα πρακτικό αριθμό επεξεργαστών που χρειάζονται για την παράλληλη εκτέλεση.

Λέξεις κλειδιά: Ηλεκτρονική Ψηφοφορία, δίκτυο μίξης, μετάθεση, απόδειξη μηδενικής γνώσης, παράλληλη εκτέλεση, κλιμακωσιμότητα, επιτάχυνση.

Ευχαριστίες

Η παρούσα διπλωματική εργασία εκπονήθηκε στο Εθνικό Μετσόβιο Πολυτεχνείο, υπό την επίβλεψη του Καθηγητή Παναγιώτη Τσανάκα.

Θα ήθελα να ευχαριστήσω τον καθηγητή μου, Παναγιώτη Τσανάκα, που με εισήγαγε στο αντικείμενο των Συστημάτων και της Αρχιτεκτονικής Υπολογιστών και για την ευκαιρία που μου έδωσε να εργαστώ σε αυτό το ενδιαφέρον επιστημονικό πεδίο. Επίσης, τον ευχαριστώ για την εμπιστοσύνη που μου έδειξε με την παρούσα εργασία και για τη δυνατότητα να συνεισφέρω σε ένα πραγματικό σύστημα ηλεκτρονικών ψηφοφοριών.

Ιδιαίτερα θα ήθελα να ευχαριστήσω τον Γεώργιο Τσουκαλά που μοιράστηκε τις γνώσεις του μαζί μου όλους αυτούς τους μήνες, τις συμβουλές, την υπομονή του και το χρόνο που αφιέρωσε, τα οποία αποδείχθηκαν ανεκτίμητα για την ολοκλήρωση αυτής της εργασίας. Η βαθειά γνώση και η μεγάλη εμπειρία του στα Συστήματα Υπολογιστών ήταν εκείνο που με βοήθησε, μέσα από τις συζητήσεις μας, να λύσω τα προβλήματα που προέκυψαν.

Θα ήθελα επίσης να ευχαριστήσω τον Πάνο Λουρίδα για την καθοδήγηση και τη συμβολή του από την αρχή μέχρι το τέλος της παρούσας διπλωματικής.

Θέλω να ευχαριστήσω τους φίλους μου για τα υπέροχα φοιτητικά χρόνια που περάσαμε και όλες τις εμπειρίες που απέκτησα τόσο σε ακαδημαϊκό όσο και σε προσωπικό επίπεδο.

Τέλος, θέλω να ευχαριστήσω την οικογένειά μου για τη συνεχή στήριξή τους όλα αυτά τα χρόνια και την εμπιστοσύνη τους στις επιλογές μου.

Κωνσταντίνος Ι. Μαμασούλας

Contents

List of Tables

List of Figures

Κεφάλαιο 1

Εισαγωγή

Από τότε που γεννήθηκε η δημοκρατία, διάφορα εκλογικά συστήματα έχουν σχεδιαστεί, επιτρέποντας στους διαφόρους πληθυσμούς να εκφράζουν τη γνώμη τους πάνω σε κρίσιμες αποφάσεις. ΄Ετσι, με τον όρο *ψηφοφορία*, εννοοούμε ότι ένα πρόσωπο μπορεί να εκφράσει έναν αριθμό επιλογών ελεύθερα ανάμεσα σε ένα ευρέως γνωστό σύνολο υποψηφίων. Το πιο διαδεδομένο σύστημα ψηφοφοριών σήμερα είναι το παραδοσιακό (paper-based). Σε αυτό το σύστημα, υπάρχουν συγκεκριμένες ημέρες που ένας ψηφοφόρος μπορεί να έρθει σε ένα *εκλογικό κέντρο*, να περάσει από μια διαδικασία πιστοποίησης, να λάβει ένα ψηφοδέλτιο και να διαλέξει την ψήφο του μέσα σε ένα *θάλαμο ψηφοφορίας (παραβάν)*. Ακολούθως, καταθέτει την ψήφο του σε ένα διαφανές κουτί (κάλπη), μπροστά στην αρμόδια επιτροπή, και καταχωρείται, κάτι το οποίο αποδεικνύει ότι έλαβε μέρος στις εκλογές. ΄Οταν η διαδικασία της ψηφοφορίας τελειώνει, το κουτί που περιέχει όλες τις ψήφους ‘ανοίγει’ από την (εφορευτική) επιτροπή, μετρούνται οι ψήφοι και τα εκλογικά αποτελέσματα ανακοινώνονται. Με αυτό τον τρόπο, ο ψηφοφόρος πείθεται ότι η ανωνυμία, η ιδιωτικότητα και η επαληθευσιμότητα της ψήφου ικανοποιούνται. Δυστυχώς, υπάρχουν παραδείγματα που δείχνουν αυτές τις ιδιότητες να αποτυγχάνουν και διάφορες απάτες έχουν έρθει στο φως ανά τα χρόνια όπου οι εκλογές έχουν εδραιωθεί.

Αν και ο παραδοσιακός τρόπος εκλογών χρησιμοποιείται ευρέως σήμερα, ένα μειονέκτημα αυτού είναι ότι χρειάζεται αρκετό χρόνο για να ολοκληρωθεί. Αφήνοντας στην άκρη το γεγονός ότι οι ψηφοφόροι πρέπει πραγματικά να πάνε στα εκλογικά κέντρα, η καταμέτρηση των ψήφων μπορεί να πάρει ώρες, ή και μέρες, οδηγώντας σε μια χρονοβόρα ανακοίνωση των αποτελεσμάτων. Αυτό ακριβώς είναι που ένα σύστημα ηλεκτρονικών ψηφοφοριών (E-Voting system) σκοπεύει να βελτιώσει, αντί για ώρες ή μέρες μετρήματος ψήφων, τα αποτελέσματα των εκλογών να ανακοινώνονται στο διάστημα μιας ώρας για πα-

ράδειγμα. Φυσικά, υπάρχουν πολλές παράμετροι ασφαλείας οι οποίοι χρειάζεται να μελετηθούν πολύ προσεκτικά πριν την πραγματική χρήση ενός τέτοιου συστήματος και, σήμερα, υπάρχουν πολλές αξιόλογες προσπάθειες σε αυτό το πεδίο.

Στα επόμενα κεφάλαια, παρουσιάζονται περισσότερα για τις ιδιότητες που πρέπει να έχει ένα σύστημα ηλεκτρονικν ψηφοφοριών.

Επισκόπηση

Ηλεκτρονική Ψηφοφορία (ή E-Voting) είναι η μεταφορά του συμβατικού παραδοσιακού τρόπου ψηφοφορίας στον ψηφιακό/ηλεκτρονικό κόσμο. ΄Οπως υποδεικνύει ο όρος, η χρήση ενός ηλεκτρονικού συστήματος είναι απαραίτητη τόσο για την κατάθεση όσο και για το μέτρημα των ψήφων. Σαν κάθε άλλο σύστημα, τα e-voting συστήματα στοχεύουν στην επιτάχυνση της εκλογικής διαδικασίας, εκμεταλλευόμενα ό,τι έχει να μας προσφέρει η τεχνολογία σήμερα. Υπάρχουν δύο τύποι ηλεκτρονικής ψηφοφορίας: η απομακρυσμένη και εκείνη που απαιτεί τη φυσική παρουσία των ψηφοφόρων στα εκλογικά κέντρα, χρησιμοποιώντας ειδικές ηλεκτρονικές συσκευές. Εστιάζουμε στην πρώτη. Με απλά λόγια, έχοντας ένα απομακρυσμένο e-voting σύστημα σημαίνει ότι ο ψηφοφόρος μπορεί να υποβάλλει την ψήφο του χρησιμοποιώντας ένα πρόγραμμα που τρέχει στον προσωπικό του υπολογιστή. Αυτή η προσέγγιση έχει πολλά πλεονεκτήματα, αλλά και σοβαρά μειονεκτήματα που χρειάζεται να αντιμετωπιστούν.

Οι ηλεκτρονικές επιλογές για ψηφοφορίες, φυσικά, ικανοποιούν την ανάγκη των ψηφοφόρων για άνεση/ευκολία. Μπορούμε να φανταστούμε ότι κανένας δε θα χρειάζεται να είναι παρών σε ένα εκλογικό κέντρο για να καταθέσει την ψήφο του και, συνεπώς, αυτό είναι μια αποτελεσματική εναλλακτική για κάποιες ευπαθείς κοινωνικές ομάδες, πχ άνθρωποι με σωματική αναπηρία. Επίσης, το κόστος για την εκτύπωση και ταχυδρόμηση των ψηφοδελτίων μειώνεται σημαντικά, καθώς επίσης και τα έξοδα προσωπικού κατά τη διάρκεια των εκλογών.

Στην άλλη πλευρά, ο σχεδιασμός και η υλοποίηση τέτοιων συστημάτων είναι μια αρκετά δύσκολη διαδικασία λόγω των αρχών που υπαγορεύει το παραδοσιακό μοντέλο ψηφοφοριών σε συνδυασμό με τους βαθύτερους κινδύνους του Διαδικτύου. Η ιδιωτικότητα, η επαληθευσιμότητα, η ορθότητα και πολλές ακόμη ιδιότητες πρέπει να προσαρμοστούν ή ακόμα και να διευρυνθούν προκειμένου να προάγουν ένα σύστημα ηλεκτρονικής ψηφοφορίας.

Υπάρχουν αρκετά στάδια τα οποία απαρτίζουν ένα e-voting πρωτόκολλο, αλλά η παρούσα εργασία επικεντρώνεται σε ένα συγκεκριμένο, το οποίο λέγεται ***μίξη*** και εξηγείται με περισσότερη λεπτομέρεια

στο επόμενο κεφάλαιο. Απλά, αντιπροσωπεύει την εγγύηση ότι οι ψήφοι δε μπορούν να κινδυνεύσουν, όπως σε ένα παραδοσιακό πρωτόκολλο εκλογών.

Σήμερα, πολλοί ερευνητές και μηχανικοί εργάζονται μαζί για την παραγωγή τέτοιων συστημάτων και υπάρχουν αρκετές αξιόλογες υλοποιήσεις. Στην Ελλάδα, υπάρχει ήδη μια πλήρης και χρησιμοποιούμενη e-voting υπηρεσία, το ***'Ζευς'***, το οποίο έχει αναπτυχθεί από το Εθνικό Δίκτυο Έρευνας και Τεχνολογίας (GRNET) και βασίζεται στο σύστημα Helios.

Κίνητρα και σκοπός

Όπως αναφέρθηκε προηγουμένως, ένα e-voting σύστημα αποτελείται από πολλές συνιστώσες. Το κλειδί για μια επιτυχημένη υπηρεσία ηλεκτρονικών εκλογών έγκειται σε ένα ισχυρό κρυπτογραφικό σχήμα, το οποίο αποτελεί και τον πυρήνα της διαδικασίας, και ένα ισχυρό κρυπτογραφικό σχήμα μεταφράζεται σε μεγάλη υπολογιστική πολυπλοκότητα, σε όρους επιστήμης υπολογιστών. Τέτοια διαδικασία είναι το προαναφερθέν στάδιο της μίξης: απαιτεί πολλούς υπολογιστικούς πόρους ώστε να παράγει ένα έγκυρο αποτέλεσμα και όσο μεγαλύτερος είναι ο αριθμός των ψηφοφόρων, τόσο περισσότερους πόρους χρειάζεται. Γι΄ αυτό, αξίζει να μελετήσουμε τρόπους επιτάχυνσης της διαδικασίας αυτής.

Ο σκοπός αυτής της εργασίας είναι η μελέτη, η υλοποίηση και η σύγκριση διαφόρων αλγορίθμων που μπορούν να χρησιμοποιηθούν για το στάδιο της μίξης, στο πλαίσιο του συστήματος ηλεκτρονικών εκλογών Zeus. Το Zeus χρησιμοποιεί έναν πολύ ακριβό υπολογιστικά αλγόριθμο για να κάνει μίξη και, ως αποτέλεσμα, αποτελεί και το πιο αργό στάδιο της ροής εργασιών του. Γι΄ αυτό το λόγο, πραγματοποιήθηκε μια έρευνα πάνω στους διαθέσιμους αλγορίθμους που υπάρχουν στη βιβλιογραφία, ώστε να εξετάσουμε τη δυνατότητα αντικατάστασης του ήδη υπάρχοντος. Μετά το πέρας αυτής της έρευνας, επιλέγεται ένας αλγόριθμος για υλοποίηση και αποτιμάται η επίδοσή του, οδηγώντας σε συμπεράσματα σχετικά με τον κύριο στόχο μας, την επιτάχυνση μίξης.

Κεφάλαιο 2

Βασικές αρχές του E-voting

2.1 Αρχές και ιδιότητες

Για να καταλάβουμε πώς μπορούμε να κατασκευάσουμε ένα σύστημα ηλεκτρονικής ψηφοφορίας σε σύγκριση με το παραδοσιακό μοντέλο, παρακάτω παρουσιάζονται τα δομικά στοιχεία τα οποία πρέπει να ενσωματώνει το καθένα. Φυσικά, ένα ηλεκτρονικό σύστημα ψηφοφοριών θα πρέπει να πετυχαίνει *τουλάχιστον* ό,τι και τα παραδοσιακά σχήματα [9]:

Μυστικότητα

Αυτή είναι μια από τις πιο θεμελιώδεις ιδιότητες για μια ψηφοφορία. Κανείς άλλος συμμετέχων εκτός από τον ίδιο τον ψηφοφόρο δεν πρέπει να μπορεί να αποφανθεί για το περιεχόμενο της ψήφου του.

Ορθότητα

Αν όλοι οι συμμετέχοντες σε μια διαδικασία εκλογών είναι τίμιοι και συμπεριφέρονται όπως έχει προγραμματιστεί, τότε τα τελικά αποτελέσματα είναι το μέτρημα των ψήφων που υποβλήθηκαν. Αυτή η ιδιότητα εξασφαλίζει ότι το αποτέλεσμα των εκλογών είναι ακριβές.

Receipt-Freeness

Οι ψηφοφόροι δεν πρέπει να μπορούν ούτε να λαμβάνυν ούτε να κατασκευάζουν κάποια απόδειξη η οποία μπορεί να (απο)δείξει το περιεχόμενο της ψήφου τους. Αυτό σημαίνει ότι ο ψηφοφόρος δε μπορεί να πουλήσει την ψήφο του ούτε να εξαναγκαστεί από άλλους, επειδή δε μπορεί να παρέχει κανένα στοιχείο για το πώς ψήφισε.

Ευρωστία

Αντίσταση σε εσφαλμένη συμπεριφορά και σε συνασπισμό ψηφοφόρων: οποιοσδήποτε ψηφοφόρος που 'κλέβει' θα ανιχνεύεται και η διαδικασία της ψηφοφορίας δε θα διασπάται.

Επαληθευσιμότητα

Μια σωστή διαδικασία εκλογών πρέπει να είναι επαληθεύσιμη ώστε να εμποδίζεται οποιοδήποτε εσφαλμένο αποτέλεσμα. Αυτό μπορεί να πραγματοποιηθεί είτε **ατομικά**, δηλαδή κάθε ψηφοφόρος να επαληθεύει αν η ψήφος του λήφθηκε υπόψιν, είτε **καθολικά**, δηλαδή οποιοσδήποτε συμμετέχων ή παθητικός παρατηρητής να μπορει να πειστεί για την εγκυρότητα των ψήφων και το τελικό μέτρημα της ψηφοφορίας.

Δημοκρατία

΄Ολοι οι έγκυροι συμμετέχοντες σε μια διαδικασία εκλογών μπορούν να ψηφίσουν **μόνο μία φορά**, έτσι ώστε να έχουν ίση συνεισφορά στο τελικό αποτέλεσμα των εκλογών.

Δικαιοσύνη

Οποιαδήποτε πληροφορία για την καταμέτρηση της κάλπης δε μπορεί να αποκτηθεί πριν τη δημοσίευσή της.

2.2 Κρυπτογραφικές Αρχές

΄Οπως έχουμε ήδη τονίσει στο προηγούμενο κεφάλαιο, ένα ασφαλές e-voting σύστημα πρέπει να εφαρμόζει όλες τις κρυπτογραφικές ιδιότητες που χρειάζονται για τη διατήρηση των αρχών που διέπουν μια εκλογική διαδικασία, όπως είδαμε παραπάνω.

2.2.1 Κρυπτοσύστημα ElGamal

Το πιο διαδεδομένο κρυπτογραφικό σχήμα που χρησιμοποιείται στις ηλεκτρονικές ψηφοφορίες είναι ElGamal. Αποτελεί ένα ομομορφικό σχήμα κρυπτογράφησης όπου επιλέγονται μια κυκλική ομάδα $\mathbb{G}$ με γεννήτορα g και δύο μεγάλοι πρώτοι αριθμοί, η τάξη q και το modulus p, έτσι ώστε $p = 2q + 1$.

Κρυπτογράφηση ElGamal

Είναι ένας ασύμμετρος αλγόριθμος κρυπτογράφησης δημοσίου κλειδιού, που βασίζεται στην ανταλλαγή κλειδιού Diffie-Hellman [20]. ΄Ενα κρυφό (ιδιωτικό) κλειδί $x \in \mathbb{Z}_q$ διαλέγεται τυχαία και το δημόσιο κλειδί του κρυπτογραφικού σχήματος προκύπτει ως $(p, g, y = g^x \bmod p)$. Συνεπώς, το κρυπτοκείμενο ενός μηνύματος M (plaintext) ορίζεται ως ένα ζεύγος ElGamal ως εξής:

$$C = (a, b) = (g^r, y^r M)$$

όπου r διαλέγεται τυχαία από το $\mathbb{Z}$ (encryption randomness) κι έτσι,

$$\text{ENCRYPT}(M, r) = (g^r, y^r M) = C.$$

Αποκρυπτογράφηση ElGamal

Για να αποκρυπτογραφήσουμε ένα κρυπτογραφημένο ζεύγος ElGamal (a, b) και να λάβουμε το αρχικό μήνυμα M, υπολογίζουμε:

$$M = \text{DECRYPT}(C) = \frac{b}{a^x} \bmod p$$

2.2.2 Αποδείξεις Μηδενικής Γνώσης

΄Ενα από τις πιο σημαντικές (κρυπτογραφικές) έννοιες που εντάσσονται στα συστήματα ηλεκτρονικών ψηφοφοριών είναι η ***Απόδειξη Μηδενικής Γνώσης*** (Zero-Knowledge Proof) [9, 22]. Αυτές οι αποδείξεις αποτελούνται από μεθόδους τις οποίες μια ομάδα **(prover)** μπορεί να χρησιμοποιήσει για να αποδείξει σε μια άλλη ομάδα **(verifier)** ότι μια πρόταση είναι αλήθεια, χωρίς όμως να αποκαλύπτει καμία άλλη πληροφορία εκτός από το ότι η πρόταση είναι όντως αληθής. Κάθε Απόδειξη Μηδενικής γνώσης ακολουθεί κάποιες βασικές αρχές επίσης:

- **Πληρότητα:** Δεδομένου ότι εκείνος που αποδεικνύει την αλήθεια - prover - είναι έντιμος, δηλαδή ότι η πρόταση είναι πραγματικά αληθής, τότε εκείνος που επαληθεύει την αλήθεια - verifier - θα δεχτεί την απόδειξη. Ακριβέστερα, η πληρότητα διασφαλίζει ότι ο verifier θα δεχτεί την απόδειξη με πολύ μεγάλη πιθανότητα (κοντά στο 1).
- **Ορθότητα:** Αν ο prover δεν είναι έντιμος, δηλαδή αν η πρόταση είναι ψευδής, τότε ο verifier δεν θα πρέπει να δεχτεί την απόδειξη, παρά μόνο με πολύ μικρή πιθανότητα.

- **Μηδενική γνώση:** Ο verifier δεν μπορεί να μάθει τίποτα παραπάνω από την αλήθεια (ή όχι) της πρότασης την οποία παράγει ο prover.

Μια *Απόδειξη Μηδενικής Γνώσης* μπορεί να χωριστεί σε δύο κατηγορίες: **διαδραστική** (interactive) [22] και **μη διαδραστική** (non-interactive).

Προφανώς, μια διαδραστική απόδειξη μηδενικής γνώσης χρειάζεται αλληλεπίδραση μεταξύ των prover και verifier. Αντίθετα, μια μη διαδραστική απόδειξη δεν το απαιτεί και σημειώνεται ότι και οι δύο τηρούν τις ιδιότητες που αναφέρθηκαν παραπάνω.

2.2.3 Fiat-Shamir Heuristic

Μια μη διαδραστική απόδειξη μηδενικής γνώσης απαιτεί τη χρήση μιας **κοινής συμβολοσειράς αναφοράς** (common reference string), η οποία μοιράζεται ανάμεσα στον prover και στον verifier. Είναι δυνατό να μετατρέψουμε μια διαδραστική απόδειξη μηδενικής γνώσης σε μια μη διαδραστική με χρήση του Fiat-Shamir Heuristic [8].

Αυτο το heuristic κάνει χρήση μιας κρυπτογραφικής συνάρτησης κατακερματισμού (hash function) για να παραγάγει το *κοινό μυστικό* μεταξύ του prover και του verifier.

Γενικά, τα συστήματα ηλεκτρονικών ψηφοφοριών επιθυμούν τη χρήση μιας μη διαδραστικής απόδειξης μηδενικής γνώσης. Για παράδειγμα, ας σκεφτούμε την περίπτωση όπου η διεξάγουσα αρχή μιας διαδικασίας εκλογών (prover) οφείλει να πείσει πολλές πλευρές (verifiers) ότι η καταμέτρηση των ψήφων είναι σωστή. Δεν υπάρχει λόγος να παράγει μια ξεχωριστή απόδειξη για κάθε verifier, αλλά αντίθετα, παράγει μία μόνο απόδειξη την οποία μπορεί ο κάθε verifier να ελέγξει.

2.2.4 Μίξη

΄Οπως αναφέρθηκε στο προηγούμενο κεφάλαιο, η *μυστικότητα* είναι μια απαραίτητη ιδιότητα που δε μπορεί να λείπει από ένα πρωτόκολλο ηλεκτρονικής ψηφοφορίας. Η **Μίξη** είναι το στάδιο το οποίο είναι υπεύθυνο για τη διατήρηση της ταυτότητας των ψηφοφόρων κρυφή σε μια διαδικασία εκλογών, κάνοντας χρήση ενός *αλγορίθμου μίξης*. Το στοιχείο του συστήματος το οποίο εκτελεί αυτόν τον αλγόριθμο καλείται **mixnet** ή **δίκτυο μίξης**. Με άλλα λόγια, ένα mixnet λαμβάνει έναν αριθμό από (κρυπτογραφημένες) ψήφους που υποβλήθηκαν από τους έγκυρους ψηφοφόρους και παράγει ένα νέο σύνολο ψήφων, η οποίες είναι **επανακρυπτογραφημένες** και **τυχαίως ανακατεμένες**. Αυτό το είδος δικτύου μίξης ονομάζεται 're-encryption mixnet'. Αν κάποιος είχε στη διάθεσή του μόνο το δεύτερο σύνολο

από ψήφους στη διάθεσή του, είναι υπολογιστικά αδύνατο να συνδυάσει μια τέτοια ψήφο με ένα συγκεκριμένο ψηφοφόρο. Μια τέτοια διαδικασία όμως, πρέπει να **αποδεικνύεται** ότι έγινε σωστά, όπως θα δούμε παρακάτω.

Η ιδέα των mixnets προτάθηκε πρώτα από τον David Chaum, το 1981 [6]. Για να επιτευχθεί ανωνυμία μεταξύ δύο πλευρών που ανταλλάσσουν αλληλογραφία (e-mails), ορίζεται η έννοια ενός ***mix-server*** (ή 'mix'). Ο ρόλος του είναι η επεξεργασία των e-mails πριν την παράδοσή τους από έναν αποστολέα σε έναν παραλήπτη. Κάνοντας χρήση της κρυπτογράφησης δημοσίου κλειδιού, ο αποστολέας προετοιμάζει ένα μήνυμα M και μετά το κρυπτογραφεί με το δημόσιο κλειδί του παραλήπτη K_a. Το αποτέλεσμα επανακρυπτογραφείται χρησιμοποιώντας το δημόσιο κλειδί του mix-server και έπειτα στέλνεται σε αυτόν. Ακολούθως, ο mix-server χρησιμοποιεί το κρυφό κλειδί του για να λάβει το κρυπτογραφημένο μήνυμα M και τη διεύθυνση-στόχο A του παραλήπτη, στην οποία θα παραδώσει το μήνυμα. Η διαδικασία αυτή αντικατοπτρίζεται στην είσοδο και στην έξοδο ενός mix-server:

$$K_1\left(R_1, K_a\left(R_0, M\right), A\right) \rightarrow K_a\left(R_0, M\right), A$$

όπου R_0, R_1 είναι κάποιες τυχαίες ακολουθίες που προσκολλώνται σε κάθε μήνυμα με σκοπό την κρυπτογράφησή του, για να εξασφαλίσουμε ότι κανένας δε μπορεί να εξάγει τα περιεχόμενά του, πχ δοκιμάζοντας κάποιες τιμές X τέτοιες ώστε $K_a\left(X\right) = K_a\left(M\right)$, με K_1, K_a να είναι τα δημόσια κλειδιά του παραλήπτη και του mix-server, αντίστοιχα.

Για να επεκτείνει την ιδέα του mixnet, ο Chaum πρότεινε μια αλυσίδα από ξεχωριστά 'mixes'. Σε αυτή την περίπτωση υπάρχουν N mix-servers διατεταγμένοι ως μια ακολουθία, μαζί με τα ιδιωτικά/δημόσια κλειδιά τους. Ο αποστολέας κρυπτογραφεί το αρχικό μήνυμα M με όλα τα δημόσια κλειδιά, για κάθε διαδοχικο mix-server στην αλυσίδα. Για παράδειγμα, η είσοδος και η έξοδος του πρώτου κόμβου είναι

$$K_N\left(R_N, K_{N-1}\left(R_{N-1}, \ldots, K_2\left(R_2, K_1\left(R_1, K_a\left(R_0, M\right), A\right)\right)\ldots\right)\right) \rightarrow \\ K_{N-1}\left(R_{N-1}, \ldots, K_2\left(R_2, K_1\left(R_1, K_a\left(R_0, M\right), A\right)\ldots\right)\right)$$

και η τελική έξοδος της αλυσίδας είναι $[K_a\left(R_0, M\right), A]$, όπως αναμένεται.

Αυτό είναι ένα κατάλληλο σχήμα για να χρησιμοποιηθεί στα συστήματα ηλεκτρονικών εκλογών, για τη διατήρηση της ανωνυμίας της ψήφου. Κάθε ψήφος, που κρυπτογραφείται και καταχωρείται από τους νόμιμους ψηφοφόρους, θα συμπεριλαμβάνεται στην είσοδο ενός mixnet (αλυσίδα από mix-servers) και η τελική λίστα των αποκρυπτογραφημένων ψήφων θα δημιουργείται, χωρίς να αποκαλύπτει κάποια αντιστοίχιση με αυτούς. Αυτό απαιτεί είτε κάποια τυχαία μετάθεση των

ψήφων πριν την έξοδο κάθε βήματος, ή ακόμα και μια αλφαβητική σειρά αυτών, όσο όμως δεν υπάρχουν ίδια αντικείμενα στην είσοδο.

Αυτό το δίκτυο μίξης εμπίπτει στην κατηγορία των λεγόμενων *decryption mixnets* (mixnets αποκρυπτογράφησης). Αυτού του είδους τα δίκτυα έχουν αρκετές αδυναμίες. Πρώτα απ΄ όλα, αν ένας mix-server δεν είναι έντιμος, μπορεί να ανταλλάξει το μήνυμα με ένα δικό του, κάτι που δεν είναι αποδεκτό σε μια διαδικασία ψηφοφορίας. Επιπλέον, ο τελευταίος mix-server στην αλυσίδα μπορεί να αποκρυπτογραφήσει το περιεχόμενο των ψήφων και, αν δεν του 'αρέσει' η έξοδος, μπορεί να εγκαταλήψει τη διαδικασία. Γι΄ αυτό το λόγο, ένα re-encryption mixnet (δίκτυο επανακρυπτογράφησης) χρησιμοποιείται πιο συχνά.

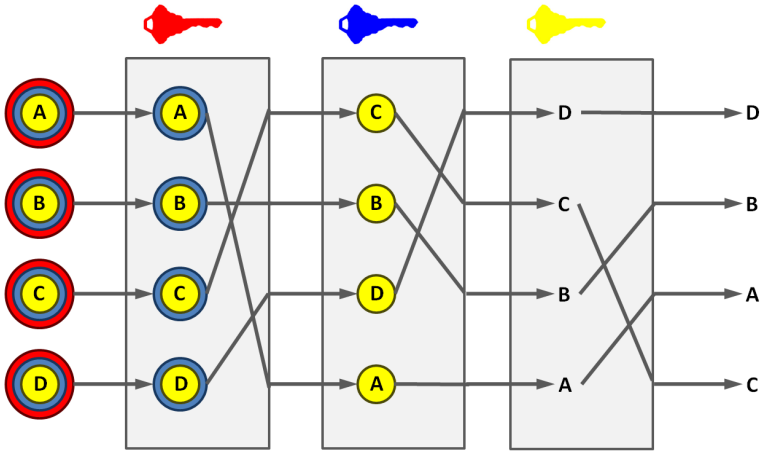

Figure 2.1: Example of a decryption mixnet

Από την άλλη μεριά, ένα δίκτυο επανακρυπτογράφησης [16] είναι υπεύθυνο για την επανακρυπτογράφηση και τη μετάθεση τον ψήφων εισόδου χωρίς να τις αποκρυπτογραφεί μερικώς. Γι΄ αυτό το λόγο, χρησιμοποιείται συχνά ένα ομομορφικό σχήμα κρυπτογράφησης, όπως το ElGamal. ΄Ετσι, κάθε mix-server κάνει αυτή ακριβώς τη διαδικασία και προωθεί το αποτέλεσμα στον επόμενο κόμβο του δικτύου. Η φάση της αποκρυπτογράφηση εκτελείται ξεχωριστά, μετά από τη μίξη.

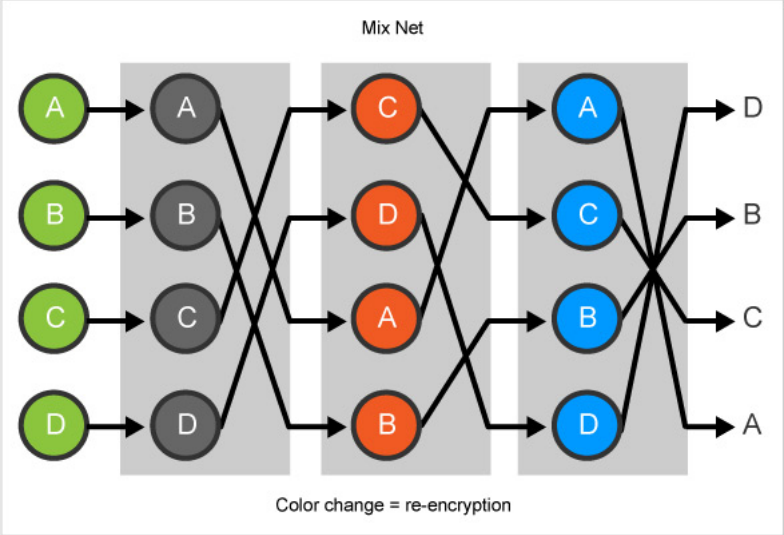

Figure 2.2: Example of a re-encryption mixnet

2.3 Εξαναγκασμός ψήφου

Μια σημαντική παράμετρος που πρέπει να ληφθεί υπόψιν για ένα σύστημα ηλεκτρονικών ψηφοφοριών είναι ο *εξαναγκασμός ψήφου* (coercion). Αν ένας αντίπαλος (adversary) καταφέρει να αναγκάσει έναν ψηφοφόρο να υποβάλει την ψήφο του με συγκεκριμένο τρόπο, ή ακόμα να υποβάλει και μια τυχαία ψήφο, τότε η ψηφοφορία μπορεί να εξαγοραστεί. Τέτοια συμπεριφορά δεν είναι αποδεκτή από ένα σύστημα ψηφοφορίας. ΄Ενας άλλος τύπος εξαναγκασμού είναι η *πώληση ψήφου* (vote selling), όπου ένας ψηφοφόρος εθελοντικά προσπαθεί να αποκαλύψει το περιεχόμενο της ψήφου του σε τρίτους. Αν και έχουν διαφορές, και οι δύο περιπτώσεις μοιάζουν ίδιες όταν πρόκειται για την αντιμετώπισή τους.

Γενικά, δεν είναι εύκολη διαδικασία να γίνει ένα e-voting σύστημα ανθεκτικό σε εξαναγκασμούς ψήφων (coercion-resistant). Λόγω της ανάπτυξης της τεχνολογίας, είναι σχετικά εύκολο για όλους να αποκτήσουν μια (κρυφή) βιντεοκάμερα, να καταγράψουν την ταυτότητά τους και όλες τις ενέργειές τους με ένα εκλογικό κέντρο. Με αυτόν τον τρόπο, ένας ψηφοφόρος μπορεί να πουλήσει την ψήφο του ή να αποδείξει το περιεχόμενό της σε τρίτους. Συνεπώς, ένα σύστημα ανθεκτικό στον εξαναγκασμό ψήφου, θα καθοδηγούσε τους ψηφοφόρους του να κλείσουν οποιαδήποτε συσκευή καταγραφής κατα τη διάρκεια

της ψηφοφορίας και δε θα παρείχε καμία απόδειξη (receipt) που να αποδεικνύει πώς ψήφισαν. ΄Οπως αναφέρθηκε, η ιδιότητα του receipt-freeness ισχυρίζεται ότι ο ψηφοφόρος δε μπορεί να λάβει οποιαδήποτε πληροφορία από το σύστημα για να αποδείξει το περιεχόμενο της ψήφου του. Ως αποτέλεσμα, εκείνος που θα εξανάγκαζε κάποιον δε μπορεί να είναι σίγουρος για την επιλογή του ψηφοφόρου και, συνεπώς, δε μπορεί να τον επηρεάσει.

΄Ενα άτομο που μπορεί να (ή έχει σκοπό να) εξαναγκάσει κάποιον ψηφοφόρο (coercer) μπορεί να δράσει είτε τοπικά είτε απομακρυσμένα. Προφανώς, είναι ευκολότερο να είναι παρών σε ένα εκλογικό κέντρο, όπου μπορεί να αλληλεπιδράσει με τον ψηφοφόρο, σε αντίθεση με το αν ήταν απομακρυσμένος, όπου ο ψηφοφόρος έχει το ελεύθερο της επιλογής του. Ωστόσο, αν υπάρχει οποιοσδήποτε τρόπος καταγραφής ή αν παρέχονται αποδείξης ψήφου, δεν υπάρχει σημαντική διαφορά ανάμεσα στους δύο τύπους.

Εκτός από την τοποθεσία ενός coercer, ο χρόνος που ενεργεί είναι επίσης πολύ κρίσιμος. Τα σύγχρονα συστήματα μπορούν να σχεδιαστούν για να αποφεύγουν οποιοδήποτε *μετεκλογικό εξαναγκασμό*, ενώ είναι πρακτικά αδύνατο να αντιμετωπίσουν *προεκλογικό εκαναγκασμό*. Πέρα από τις παραπάνω προσεγγίσεις που μπορεί να ακολουθήσει ένα σύστημα ηλεκτρονικών ψηφοφοριών, χρειάζεται και ένα κανάλι που δε θα μπορεί να καταγράφεται "untappable channel" [4], ως μέτρο αντιμετώπισης του προεκλογικού εξαναγκασμού ψήφου.

2.4 Προκλήσεις

Σε συνδυασμό με τις αρχές τις οποίες πρέπει να διατηρεί ένα ισχυρό e-voting σύστημα, υπάρχουν διάφορες ακόμα προκλήσεις που πρέπει να αντιμετωπιστούν, σχετικές με την επίδοση και την εφαρμογή του. Μερικές από αυτές είναι:

- **Ταχύτητα:** Είναι ένα κύριο χαρακτηριστικό για την επίδοση μιας e-voting υπηρεσίας. ΄Ενα σύστημα το οποίο παράγει αποτελέσματα, δηλαδή η καταμέτρηση των ψήφων, μετά από ένα πολύ μεγάλο χρονικό διάστημα από τι στιγμή που υποβάλλονται οι ψήφοι των συμμετεχόντων, δεν είναι επιθυμητό.

- **Κλιμακωσιμότητα:** Ιδανικά, θα θέλαμε ένα σύστημα ηλεκτρονικών ψηφοφοριών να χειρίζεται οποιοδήποτε αριθμό ψήφων ή, τουλάχιστον, τον αριθμό συμμετεχόντων σε έκταση μιας χώρας.

- **Ευκολία στη χρήση:** ΄Ενα e-voting σύστημα απευθύνεται σε όλους τους συμμετεχόντες μιας διαδικασίας ψηφοφοριών, πχ στα άτομα μιας ομάδας ή τους πολίτες μιας χώρας. Οποιοσδήποτε

έχει το δικαίωμα να ψηφίσει, πρέπει να έχει πρόσβαση σε ένα εύκολο στη χρήση τεχνολογικό μέσο, χωρίς να επηρεάζεται από παράγοντες όπως η γλώσσα, η ηλικία κλπ.

- **Χαμηλό κόστος:** ΄Ενα e-voting σύστημα δεν είναι χρήσιμο αν είναι πολύ ακριβό. Αν συμβαίνει αυτό, άσχετα με το αν είναι ασφαλές, δεν είναι καθόλου ελκυστικό. Γι΄ αυτό, ενδέχεται να προτιμάται ένα λιγότερο ασφαλές σύστημα, αλλά φθηνότερο, αλλιώς οι άνθρωποι είναι πιθανότερο να μείνουν στο παραδοσιακό μοντέλο εκλογών.

Κεφάλαιο 3

Μια μελέτη για τα Mixnets

΄Εχοντας παρουσιάσει τις πιο σημαντικές ιδιότητες και απαιτήσεις ενός αξιόπιστου συστήματος ηλεκτρονικών ψηφοφοριών, εστιάζουμε στη φάση της Μίξης. Υπάρχουν αρκετοί αλγόριθμοι που έχουν προταθεί από διαφόρους συγγραφείς και ο στόχος σε αυτό το κεφάλαιο είναι η αναζήτηση και η σύγκριση αυτών των αλγορίθμων, από τους οποίους θα επιλέξουμε έναν προς μελέτη, υλοποίηση και αξιολόγηση, τα οποία ακολουθούν σε επόμενο κεφάλαιο. ΄Οπως έχουμε ήδη αναφέρει, θα προσπαθήσουμε να βρούμε έναν εναλλακτικό αλγόριθμο μίξης για το σύστημα ηλεκτρονικών εκλογν Ζευς.

΄Ολα τα mixnets που παρουσιάζονται ανήκουν στην κατηγορία των re-encryption mixnets [16], τα οποία βασίζονται στην ιδέα του Chaum [6].

3.1 Sako-Kilian Mixnet

Οι Sako και Kilian πρότειναν το πρώτο πρωτόκολλο ηλεκτρονικής ψηφοφορίας χωρίς αποδείξεις ψήφου (receipt-free protocol) το 1995 [18]. Το πρωτόκολλο αυτό βασίζεται στην ύπαρξη n κέντρων μίξης (mix-centers), τα οποία παράγουν αποδείξεις μηδενικής γνώσης ώστε να αποδείξουν ότι η έξοδός τους είναι μια *επανακρυπτογράφηση* (re-encryption) και μετάθεση (permutation) της εισόδου τους. Αυτά τα κέντρα-κόμβοι, εργάζονται σε αλυσίδα, δηλαδή η έξοδος του κέντρου i είναι η είσοδος για το κέντρο $i + 1$ και το τελευταίο κέντρο παράγει την αποκρυπτογράφηση των αρχικών κρυπτοκειμένων, χωρίς να κινδυνεύει η ιδιωτικότητα των ψήφων. Για να μπορέσει ένας κόμβος να αποδείξει την ορθότητα της μίξης, παρέγεται και ένα 'δευτερεύον ανακάτεμα' και ανάλογα με τη δοκιμασία ενός διαδραστικού ελεγκτή (verifier), ένα κέντρο μίξης αποκαλύπτει είτε τη μετάθεση και τις παραμέτρους επανακρυπτογράφησης αυτής της δευτερέυουσας μίξης ε-

ίτε τη διαφορά μεταξύ των δύο μίξεων [2].

Το σχήμα μπορεί να χρησιμοποιηθεί με οποιοδήποτε ομομορφικό σύστημα κρυπτογράφησης (περισσότερο σύνηθες το ElGamal) και επιβάλλει μια παράμετρο ασφαλείας (security parameter) ℓ, η οποία υποδηλώνει το πόσες φορές ο prover επαναλαμβάνει μια απόδειξη μηδενικής γνώσης για τη μίξη, έτσι ώστε η πιθανότητα παρατυπίας να γίνει αμελητέα.

Τέλος, η πολυπλοκότητα του πρωτοκόλλου είναι $642N$, όπου N είναι ο αριθμός των αρχικών κρυπτοκειμένων. Οι υπολογισμοί εκθετικών ανά κέντρο μίξης μπορούν να κατανεμηθούν σε παράλληλες εργασίες και έτσι ο συνολικός χρόνος του πρωτοκόλλου να βελτιωθεί σημαντικά.

3.2 Jakobsson's Mixnet

Ο Markus Jakobsson πρότεινε ένα mixnet το 1998, περιγράφοντας μία ισχυρή μέθοδο για τη μίξης κρυπτοκειμένων ElGamal [13]. Το σχήμα του βασίζεται στην ύπαρξη ενός συγκεκριμένου αριθμού mix-servers. Κάθε mix-server είναι υπεύθυνος για τη μίξη ενός **αντίγραφου** των αρχικών κρυπτοκειμένων εισόδου του mixnet και, την ίδια στιγμή, για το 'κρύψιμο' οποιονδήποτε ιδιωτικών δεδομένων που χρησιμοποιούνται γι΄ αυτό το σκοπό (blinding). Τα αντίγραφα των κρυπτοκειμένων είναι χρήσιμα προκειμένου να ανιχνευθεί τυχόν αλλοίωση ανάμεσα στους mix-servers. Τόσο τα παραγώμενα (και επεξεργασμένα) αντίγραφα όσο και τα αρχικά ταξινομούνται και το πρωτόκολλο απαιτεί να είναι εντελώς ίδια. Αν όχι, τότε σημαίνει ότι τουλάχιστον ένας mix-server 'έκλεψε', ο οποίος μπορεί να βρεθεί με μια ξεχωριστή διαδικασία εντοπισμού. Συνεπώς, το πρωτόκολλο πετυχαίνει την ορθότητα, την ιδιωτικότητα και την ευρωστία.

΄Οσον αφορά στην αποδοτικότητά του, ο συνολικός αριθμός εκθετικών που απαιτείται για μια μίξη-αποκρυπτογράφηση N κρυπτοκειμένων είναι περίπου $3\kappa kN + 7k^2N$, όπου κ είναι ο αριθμός των γύρων επανάληψης των υπολογισμών, για να επιτευχθεί ο επιθυμητός βαθμός ασφάλειας.

Τέλος, αξίζει να σημειώσουμε ότι οι Yvo Desment ανδ Kaoru Kurosawa στο [7] παρουσίασαν μια επίθεση γι΄ αυτό το mixnet η οποία παραβιάζει την ευρωστία του.

3.3 Abe's Mixnet

Το 1999, ο Abe πρότεινε δύο σχήματα για τη μίξη κρυπτοκειμένων ElGamal, το MiP-1 και MiP-2, αντίστοιχα [1]. Η κύρια διαφορά μεταξύ

τους είναι ότι το πρώτο αποτελείται μόνο από μια φάση (τυχαιοποιημένη αποκρυπτογράφηση - randomized decryption) και το δεύτερο από δύο (μετάθεσ και αποκρυπτογράφηση). Ο πυρήνας των δύο πρωτοκόλλων είναι ένα *δίκτυο μετάθεσης* (permutation network) [1], το οποίο είναι υπεύθυνο για την τυχαιοποίηση, τη μετάθεση και, τέλος, την αποκρυπτογράφηση των κρυπτοκειμένων εισόδου, χρησιμοποιώντας εύρωστες και καθολικώς επαληθεύσιμες μεθόδους. Η κατασκευή καθενός πρωτοκόλλου περιλαμβάνει την ύπαρξη m mix-servers και στον καθένα ανατίθεται ένα υποσύνολο από τις πύλες μεταγωγής (switching gates) του επιλεγμένου δικτύου μετάθεσης. ΄Επειτα, κάθε mix-server, M_k, παράγει τα ανακατεμένα κρυπτοκείμενα εισόδου κάθε πύλης που είναι υπεύθυνος (η είσοδος του M_k είναι έξοδος του M_{k-1}) μαζί με μια απόδειξη μηδενικής γνώσης για την ορθότητα της διαδικασίας. Η έξοδος του M_k επαληθεύεται από όλους τους άλλους mix-servers, όπου και φαίνεται αν είναι έντιμος ή όχι. Και τα δυο πρωτόκολλα ακολουθούν την αυτή τη βασική ιδέα, με ελαφρώς διαφορετικό τρόπο.

Ο αριθμός των εκθετικών που χρειάζεται για την εκτέλεση των δυο σχημάτων αντιπροσωπεύει την αποδοτικότητα τους: $O(mN \log N)$ εκθετικά, όπου N είναι ο αριθμός των κρυπτοκειμένων εισόδου.

3.4 Millimix

Το Millimix είναι το όνομα του mixnet που προτάθηκε από τους Markus Jakobsson και Ari Juels το 1999 [14]. Τον πυρήνα του αποτελεί ένα δίκτυο μετάθεσης, όπως και στο [1], το οποίο αναλαμβάνει την επανακρυπτογράφηση και μετάθεση των κρυπτοκειμένων στη είσοδό του. ΄Οπως σε κάθε δίκτυο επανακρυπτογράφησης, τόσο οι παράμετροι επανακρυπτογράφησης όσο και η μετάθεση παραμένουν κρυφά και δημιουργούνται οι αντίστοιχες αποδείξεις μηδενικής γνώσεις. Το mixnet σχεδιάστηκε για κρυπτοσυστήματα ElGamal και δουλεύει ως εξής: υπάρχουν k mix-servers που δρουν σε ένα δίκτυο μετάθεσης, αποτελούμενο από $O(n \log n)$ πύλες μεταγωγής, των οποίων ο ρόλος είναι η *κατά ζεύγη* επανακρυπτογράφηση και μετάθεση. Κάθε πύλη πρέπει να αποδείξη την ορθότητα των αποτελεσμάτων, με την παραγωγή μιας *Απόδειξης Ισοδυναμίας Μηνυμάτων* (Plaintext Equivalence Proof - PEP) [14], η οποία δείχνει ότι ένα ζεύγος (α', β') είναι έγκυρη επανακρυπτογράφηση ενός ζεύγους (α', β'), και μια *Διαζευκτική Απόδειξη Ισοδυναμίας Μηνυμάτων* (Disjunctive Plaintext Equivalence Proof - DISPREP), η οποία δείχνει ότι ένα ζεύγος (α', β') είναι έγκυρη επανακρυπτογράφηση είτε του ζεύγους (α_1, β_1) είτε του (α_2, β_2). Αυτή η διαδικασία επαναλαμβάνεται από κάθε διαθέσιμο mix-server και λειτουργούν σε ακολουθία, δηλαδή η είσοδος του server i είναι η έξο-

δος του server $i+1$. Αν ο server i κάνει κάποια παρατυπία κατά την εκτέλεση του πρωτοκόλλου, αποβάλλεται και ο server $i+1$ λαμβάνει την είσοδό του από τον server $i-1$.

Το σχήμα χρειάζεται $O(kn \log n)$ εκθετικά για τη μίξη N κρυπτοκειμένων και την επαλήθευσή της (κάθε server επαληθεύει τις αποδείξεις των υπολοίπων). Επίσης τεχνικές για μαζική επαλήθευση μπορεί να εφαρμοστεί για να μειώσει το υπολογιστικό κόστος.

3.5 Furukawa-Sako Mixnet

Οι Furukawa και Sako, το 2001, πρότειναν μια σχετικά πιο αποδοτική απόδειξη μίξης μηδενικής γνώσης από τους Sako και Kilian, απαιτώντας $18N$ εκθετικά αντί για $642N$ για μίξη N κρυπτοκειμένων. Αρχικά, διεύρυναν την έννοια της γραμμικής μετάθεσης χρησιμοποιώντας έναν *πίνακα μετάθεσης* (permutation matrix), ώστε να επανακρυπτογραφούν και να ανακατεύουν τα ψηφοδέλτια εισόδου. Οι παράμετροι επανακρυπτογράφησης και ο πίνακας μετάθεσης παραμένουν κρυφά και παράγονται τα απαραίτητα επιχειρήματα μηδενικής γνώσης για να πειστεί ο οποιοσδήποτε ελεγκτής για την ύπαρξή τους. Επίσης, αυτές οι αποδείξεις μπορούν να κατασκευαστούν μη διαδραστικά χρησιμοποιώντας το Fiat-Shamir Heuristic [8].

Το πρωτόκολλο χρησιμοποιεί κρυπτοσύστημα ElGamal και τα ψηφοδέλτια εισόδου αναμένεται να έχουν τη μορφή ζευγών ElGamal, αντίστοιχα.

3.6 Boneh-Golle mixnet

Οι Dan Boneh και Philippe Golle, το 2002, κατέληξαν σε ένα 'σχεδόν καθολικά σωστό' δίκτυο επανακρυπτογράφησης, εννοώντας ότι μπορεί να εγγυηθεί μια σωστή μίξη με *μεγάλη πιθανότητα* και όχι συντριπτική [5], Η ιδέα πίσω από αυτό είναι ότι η διαδικασία της μίξης μπορεί να πραγματοποιηθεί πολύ γρήγορα και τα αποτελέσματα μιας ψηφοφορίας να ανακοινωθούν σχεδόν αμέσως. Αντίθετα, ένα συνηθισμένο mixnet που πρώτα κατασκευάζει όλες τις απαραίτητες αποδείξεις μηδενικής γνώσης για να αποδείξει την ορθότητα, περιλαμβάνει μια χρονοβόρα διαδικασία ανακοίνωσης αποτελεσμάτων. Οι συγγραφείς προτείνουν μια τεχνική επαλήθευσης με *σταθερό κόστος*, άσχετα από τον αριθμό των servers που μπορεί να χρησιμοποιούνται. Εναλλακτικά, μπορεί να εφαρμοστεί μια άλλη, διαφορετική και πολύ πιο αργή μέθοδος επαλήθευσης, ώστε να εξαλειφθεί οποιαδήποτε αβεβαιότητα στο τέλος των εκλογών.

Το συγκεκριμένο πρωτόκολλο επιτυγχάνει πληρότητα, ορθότητα (σχεδόν καθολικά - κάποιος server που δεν είναι έντιμος μπορεί να ανιχνευθεί με μεγάλη πιθανότητα, όχι συντριπτική) και ευρωστία. Δεν μπορεί να πετύχει καθολική επαληθευσιμότητα, πράγμα που μπορεί να πετύχει με μια διαφορετική και πιο αργή μέθοδο.

Σχετικά με την επίδοσή τους, το mixnet αυτό απαιτεί ένα μεταβλητό και ένα σταθερό αριθμό εκθετικών ανά mix-server, $2N + 2\alpha(2k - 1)$, για να αποδείξει μία 'σχεδόν σωστή' μίξη, όπου α είναι μια παράμετρος ασφαλείας. Αν δεν υπάρχει κάποιος ανέντιμος mix-server, τότε προχωράμε στη διαδικασία της αποκρυπτογράφησης με κόστος $(2 + 4k)N$ εκθετικά, όπου k είναι ο αριθμός των διαθέσιμων servers. Το σταθερό κόστος της επαλήθευσης είναι σημαντικό πλεονέκτημα για το χρόνο εκτέλεσης της ψηφοφορίας, αλλά, ως μειονέκτημα, ανάλογα και με την παράμετρο ασφαλείας που επιλέγεται, μπορεί να διαρρεύσουν πληροφορίες σχετικά με την (κρυφή) μετάθεση που χρησιμοποιεί κάποιος server. Αυτό είναι αποδεκτό για μεγάλης κλίμακας εκλογές, όπου ο αριθμός των κρυπτοκειμένων εισόδου είναι πολύ μεγάλος.

3.7 Neff's Mixnet

Στο κείμενό του, ο Neff παρουσίασε δύο διαφορετικά είδη μίξης [15]. Η πρώτη είναι η *απλή k-μίξη*, όπου τα κρυπτοκείμενα εισόδου απλά στοιχεία μιας κυκλικής ομάδας, και η δεύτερη αφορά σε ζεύγη ElGamal ακολούθως. Και στις δύο περιπτώσεις, υπάρχουν δύο πλευρές - ο prover και ο verifier - που αλληλεπιδρούν, όπου ο πρώτος οφείλει να πείσει το δεύτερο για την ύπαρξη μιας μετάθεσης π και κάποιων παραμέτρων επανακρυπτογράφησης σύμφωνα με τους οποίους επεξεργάστηκαν τα κρυπτοκείμενα εισόδου. Η πλευρά της επαλήθευσης (verifier) δημιουργεί και στέλνει δοκιμασίες στην πλευρά της απόδειξης (prover), για να μπορεί να προσδιορίσει την ακειραιότητα της διαδικασίας.

Η επίδοση του κάθε πρωτοκόλλου αποτυπώνεται στον παρακάτω πίνακα:

	Simple k-Shuffle	**Shuffle of ElGamal Pairs**
Proof Construction	$2k$	$8k + 4$
Verification	$4k + 2$	$12k + 4$

Table 3.1: Neff's mixnet complexity

όπου k υποδηλώνει τον αριθμό των κρυπτοκειμένων εισόδου.

3.8 Furukawa's Mixnet

Ο Jun Furukawa, το 2005, πρότεινε επίσης δύο πρωτόκολλα, τα οποία επεξεργάζονται κρυπτοκείμενα ElGamal. Το πρώτο παράγει μια αποδοτική απόδειξη για μίξη αποκλειστικά, ενώ το δεύτερο για μίξη και αποκρυπτογράφηση [10]. ΄Οπως και στο [11], χρησιμοποιείται η έννοια του *πίνακα μετάθεσης*. Το πρώτο πρωτόκολλο απαιτεί τρεις γύρους για να κατασκευάσει την αντίστοιχη απόδειξη και το δεύτερο απαιτεί πέντε γύρους. ΄Οσον αφορά στην ασφάλειά του, ο συγγραφέας τονίζει ότι το πρωτόκολλο για τη μίξη-αποκρυπτογράφηση δεν είναι μηδενικής γνώσης, αλλά μπορεί να δειχθεί ότι ένας έντιμος ελεγκτής δε μπορεί να μάθει τίποτα για την κρυφή μετάθεση, με συντριπτική πιθανότητα (Complete Permutaion Hiding [10]).

Η πολυπλοκότητα των πρωτοκόλλων για τις αντίστοιχες αποδείξεις μίξης είναι ως εξής, όπου k είναι ο αριθμός των κρυπτοκειμένων εισόδου.

		Protocol I	**Protocol II**
Shuffle Argument	Proof Construction	$9k$	-
	Verification	$6k$	-
Shuffle-Decryption Argument	Proof Construction	$10k$	$8k$
	Verification	$8k$	$6k$

Table 3.2: Furukawa's mixnet complexity

3.9 Scytl Mixnet

Οι Jordi Puiggalí Allepuz και Sandra Guasch Castelló, το 2010, πρότειναν ένα ευριστικώς ασφαλές δίκτυο επανακρυπτογράφησης, χρησιμοποιώντας τις ομομορφικές ιδιότητες των κρυπτοσυστημάτων ElGamal [17]. Το σχήμα τους υλοποιήθηκε και χρησιμοποιήθηκε σε πραγματικές εκλογές, οι ιδιότητες του οποίου εξετάστηκαν εκτενώς στο [;]. Η διαδικασία της μίξης συμπεριλαμβάνει έναν αριθμό από mix-servers, όπου ο καθένας αναλαμβάνει, όπως έχουμε ήδη δει, την επανακρυπτογράφηση και μετάθεση τον κρυπτοκειμένων στην είσοδό του. Φυσικά, η μετάθεση και οι παράμετροι επανακρυπτογράφησης πρέπει να μένουν κρυφά και οι servers λειτουργούν σε σειρά, καθένας δηλαδή προωθεί τα αποτελέσματά του σε επόμενο server. Ο τελευταίος server δημοσιεύει την τελική μίξη των ψηφοδελτίων και μετά ακολουθεί η διαδικασία της επαλήθευσης.

Μόλις ολοκληρωθεί η μίξη, ξεκινάει η διαδικασία της επαλήθευσης, όπου ο ελεγκτής ορίζει τυχαία ομάδες στις οποίες χωρίζονται τα

κρυπτοκείμενα εισόδου του κάθε server. Στη συνέχεια, ο κάθε server παρέχει πληροφορίες στον ελεγκτή σχετικά με την τοποθεσία στην έξοδό του των ψήφων κάθε ομάδας της εισόδου του, χωρίς να αποκαλύπτει περαιτέρω στοιχεία για την αρχική τους θέση στις ομάδες. ΄Επειτα, ο ελεγκτής επεξεργάζεται τα στοιχεία κάθε ομάδας (τα πολλαπλασιάζει) ώστε να λάβει μια *Απόδειξη Ακεραιότητας Εισόδου* και κάνει το ίδιο για την έξοδο, ώστε να λάβει μια *Απόδειξη Ακεραιότητας Εξόδου*. ΄Ετσι εντέλει, παράγεται μια απόδειξη μηδενικής γνώσης για κάθε ομάδα του mix-server ότι η απόδειξη ακεραιότητας εξόδου είναι μια επανακρυπτογράφηση της απόδειξης ακειραιότητας εισόδου, χρησιμοποιώντας τις ομομορφικές ιδιότητες του κρυπτοσυστήματος.

΄Οσον αφορά στην αποδοτικότητά του, ο αριθμός των εκθετικών για κάθε διαδικασία είναι ο εξής:

Mixing	**Zero-Knowledge proofs**	**Verification**
$2mk$	$2(m/n)k$	$4(m/n)k$

Table 3.3: Scytl's mixnet complexity

όπου m είναι ο αριθμός των κρυπτοκειμένων εισόδου, n είναι ο αριθμός των ψήφων ανά ομάδα εισόδου και k είναι ο αριθμός των διαθέσιμων mix-servers.

3.10 Groth's Mixnet

Το 2010, ο Jens Groth πρότεινε ένα σχήμα για μίξη ενός δεδομένου συνόλου κρυπτοκειμένων, χρησιμοποιώντας τις ομομορφικές ιδιότητες κρυπτοσυστημάτων ElGamal και των μεθόδων δέσμευσης (commitment schemes) [12]. Η διαδικασίας της μίξης του ακολουθεί το παράδειγμα του Neff, που έχει να κάνει με το αναλλοίωτο των πολυωνύμων υπό τη μετάθεση των ριζών τους, σε αντίθεση με το σχέδιο των Furukawa-Sako, όπου ο prover δεσμεύεται σε έναν πίνακα μετάθεσης [11], για να αποδείξει την ορθότητά της. ΄Ολα τα επιχειρήματα που παράγει το πρωτόκολλο είναι μηδενικής γνώσης και το σχήμα επιτρέπει την εφαρμογή τεχνικών, όπως η *πολλαπλή εκθετικοποίηση* (multi-exponent-iation) και μαζική επαλήθευση batch verification, να βελτιώσουν τη συνολική του επίδοση.

Το τελικό πρωτόκολλο που προτείνει ο συγγραφέας για μίξη ομομορφικών κρυπτοκειμένων είναι ένα επιχείρημα μηδενικής γνώσης *εφτά κινήσεων*, βασισμένο σε ένα απλότερο τεσσάρων κινήσεων που χρησιμοποιείται για τη μίξη στοιχείων με γνωστό περιεχόμενο (plain messages). Και τα δύο δεν αποκαλύπτουν καμία πληροφορία για την

κρυφή μετάθεση ή τις παραμέτρους επανακρυπτογράφησης που χρησιμοποιούν.

Σχετικά με την επίδοσή του, ο αριθμός των εκθετικών που χρειάζεται για την παραγωγή των αποδείξεων μίξης είναι $6n$ και για τον έλεγχο της είναι άλλα $6n$, το οποίο καθιστά το πρωτόκολλο πιο γρήγορο από αυτό τα [11] και Furu05, τόσο σε υπολογιστικές απαιτήσεις όσο και σε απαιτήσεις επικοινωνίας. Αυτή η πολυπλοκότητα δεν περιέχει τη διαδικασία της αποκρυπτογράφσης των κρυπτοκειμένων. Αν θέλουμε και αυτό, μπορούμε να συνδυάσουμε τη μίξη με την αποκρυπτογράφηση σε μία ενιαία λειτουργία, που απαιτεί $6n$ εκθετικά για την απόδειξη και $7n$ εκθετικά για τον έλεγχό της.

3.11 Bayer-Groth Mixnet

Το 2012, οι Bayer και Groth πρότειναν ένα αρκετά αποδοτικό mixnet, σε σχέση με τα υπόλοιπα, για την κατασκευή αποδείξεων μίξης [3]. Η ιδέα για την επανακρυπτογράφηση και μετάθεση των κρυπτοκειμένων εισόδου διατηρείται ίδια, όπως έχουμε δει. Οποιοσδήποτε ελεγκτής πείθεται για την ορθότητα των αποτελεσμάτων, με τη δέσμευση commitment που κάνει ο prover σε μια κρυφή μετάθεση και τις αντίστοιχες τυχαίες παραμέτρους. Ο τελευταίος κατασκευάζει δύο επιχειρήματα για την απόδειξη, ένα πολλαπλής εκθετικοποίησης, όπου ένα σύνολο κρυπτοκειμένων υψωμένο σε ένα σύνολο (κρυφών) τιμών αποδίδει ένα συγκεκριμένο κρυπτοκείμενο, και ένα επιχείρημα γινομένου, όπου ένα σύνολο τιμών έχει ένα συγκεκριμένο γινόμενο.

Για την επίδοση του πρωτοκόλλου έχουμε τα εξής. Ο υπολογιστικός χρόνος του prover μπορεί να μειωθεί σημαντικά εκτελώντας όλους τους υπολογισμούς στον εκθέτη. Επιπλέον, τεχνικές για γρήγορο πολλαπλασιασμό πολυωνύμων μπορούν να εφαρμοστούν, για περαιτέρω βελτίωση της διαδικασίας. Οι συγγραφείς μέσα από τα θεωρητικά και πειραματικά αποτελέσματα της μελέτης τους, έδειξαν ότι για $N = mn$ κρυπτοκείμενα, διατεταγμένα σε έναν δι-διάστατο πίνακα, η πολυπλοκότητα του prover είναι $O(N \log m)$ εκθετικά ή $O(N)$ εκθετικά, αν εφαρμοστούν κι άλλες αριθμητικές βελτιστοποιήσεις, για παράδειγμα. Η διαδικασία της επαλήθευσης χρειάζεται $O(N)$ εκθετικά, αντίστοιχα. Για $N = 100,000$ κρυπτοκείμενα, ο καλύτερος χρόνος εκτέλεσης για το επιχείρημα της μίξης είναι περίπου δύο λεπτά (βελτιστοποιημένο), χρησιμοποιώντας σύστημα κρυπτογράφησης ElGamal.

Ο παρακάτω πίνακας συγκεντρώνει τις πολυπλοκότητες των mixnets που παρουσιάστηκαν σε αυτό το κεφάλαιο, με μια ενιαία σημειογραφία για καθαρότητα:

Scheme	Year	Proof in expos	Verification in expos
Sako-Kilian	1995	$O(lsN)$	$O(lsN)$
Jakobsson	1998	$3\ell sN + 7s^2N$	
Abe	1999	$O(N \log N)$	
Millimix	1999	$O(sN \log N)$	
Furukawa-Sako	2001	$8N$	$10N$
Boneh-Golle	2002	$2N + 2\ell(2s - 1)$	$2 + 4s$
Neff	2004	$8N + 4$	$12N + 4$
Furukawa	2005	$8N$	$6N$
Scytl	2010	$(2N/v + 2N)s$	$4(N/v)s$
Groth	2010	$6N$	$7N$
Bayer-Groth	2012	$O(N \log m)$	$O(N)$

Table 3.4: Comparison for ElGamal cryptosystems $(N = mn)$, s: number of mix-servers, ℓ: security parameter, v: number of votes per group

Ο αλγόριθμος που επιλέχθηκε για ανάλυση και υλοποίηση είναι εκείνος των Furukawa-Sako. Στο επόμενο κεφάλαιο παρουσιάζεται η παραλληλοποίησή του και η αξιολόγηση της επίδοσής του σε πραγματικές συνθήκες.

Κεφάλαιο 4

Παραλληλοποίηση του Furukawa-Sako Mixnet

Σε αυτό το κεφάλαιο, παρουσιάζουμε την *παραλληλοποίηση* του αλγορίθμου των Furukawa και Sako. Αρχικά, δίνεται το σχήμα σχεδόν όπως προτείνεται από τους συγγραφείς, με μερικές αλλαγές, πχ τη μετατροπή του σε μη διαδραστικό χρησιμοποιώντας την τεχνική Fiat-Shamir. Πιο συγκεκριμένα, η φάσεις της μίξης και της επαλήθευσης διαχωρίζονται ρητά και η καθεμία διαιρείαι σε μικρότερες εργασίες, οι οποίες μοιράζονται σε ένα αριθμό από παράλληλους εργάτες (workers). Εντέλει, τα αποτελέσματα από όλους τους εργάτες συνδυάζονται στο τελευταίο στάδιο για να δημιουργήσουν είτε μια απόδειξη μίξης είτε μια δήλωση επαλήθευσης.

Σκοπός μας σε αυτό το κεφάλαιο είναι τόσο η θεωρητική όσο και η πειραματική μελέτη της παράλληλης υλοποίησης του αλγορίθμου αυτού, ώστε να μπορέσουμε να δούμε τα όριά του και την κλιμακωσιμότητά του για ηλεκτρονικές εκλογές διαφόρων μεγεθών. ΄Ετσι, θα μπορέοουμε να καθορίσουμε τα διάφορα tradeoffs που υπάρχουν και μας ενδιαφέρουν και να κατασκευάσουμε ένα γενικό μοντέλο για μια πρακτική επιλογή επεξεργαστών για μια παράλληλη εκτέλεση.

4.1 Ο αλγόριθμος

Το πρωτότυπο σχήμα σχεδιάστηκε για διαδραστική επαλήθευση μιας απόδειξης μίξης. Στην παρούσα εργασία και η μίξη και η επαλήθευση θεωρούνται διαφορετικοί αλγόριθμοι και μελετώνται ξεχωριστά. Οι δύο φάσεις περιγράφονται παρακάτω, παρουσιάζοντας τις αλλαγές που κάναμε πριν την υλοποίησή τους:

Algorithm 1 Shuffle

input: $p, q, g, y, \{C_1, \ldots, C_N\}$ ▷ p, q, g, y : *the parameters of the cryptosystem (ElGamal),* $\{C_i\} = \{(g_i, m_i)\}$ *the input ciphertexts*

$\phi \leftarrow random\_permutation(N)$ ▷ *create permutation*

$\phi^{-1} \leftarrow inverse\_permutation(N, \phi)$ ▷ *create inverse permutation*

$r_1, \ldots, r_N \leftarrow random(N, 1, q)$ ▷ *create N random values* $\in [1, q]$

$C'_1, \ldots, C'_N \leftarrow permute\_ciphertexts(N, \phi^{-1}, \{C_i\}, \{r_i\}, p, g, q, y)$
▷ *premutation and re-encryption*

$\tilde{g}, \tilde{g_1}, \ldots, \tilde{g_N} \leftarrow random\_elements(N + 1, p, g, q)$ ▷ *generate random basis*

$\sigma, \rho, \tau, \alpha, \lambda \leftarrow random(5, 1, q)$
$\{\alpha_i\}, \{\lambda_i\} \leftarrow random(2N, 1, q)$ ▷ *generate 2N + 5 random values*

Computations

$t = g^{\tau}, \upsilon = g^{\rho}, w = g^{\sigma}, u = g^{\lambda}, u_i = g^{\lambda_i}$ mod p ▷ $i = 1, \ldots, N$

$\tilde{g_i}' = \tilde{g}^{r_i} \tilde{g}_{\phi^{-1}(i)}$ mod p, ▷ $i = 1, \ldots, N$

$$\tilde{g}' = \tilde{g}^{\alpha} \prod_{i=1}^{N} \tilde{g}_i^{\alpha_i} \text{ mod p}$$

$$g' = g^{\alpha} \prod_{i=1}^{N} g_i^{\alpha_i} \text{ mod p}$$

$$m' = y^{\alpha} \prod_{i=1}^{N} m_i^{\alpha_i} \text{ mod p}$$

$t_i = g^{3\alpha_{\phi^{-1}(i)} + \tau\lambda_i}$ mod p ▷ $i = 1, \ldots, N$

$\upsilon_i = g^{3\alpha^2_{\phi^{-1}(i)} + \rho r_i}$ mod p ▷ $i = 1, \ldots, N$

$\dot{\upsilon} = g^{(\sum_{i=1}^{N} \alpha_i^3) + \tau\lambda + \rho\alpha}$ mod p ▷ $i = 1, \ldots, N$

$w_i = g^{2\alpha_{\phi^{-1}(i)} + \sigma r_i}$ mod p ▷ $i = 1, \ldots, N$

$\dot{w} = g^{\left(\sum_{i=1}^{N} \alpha_i^2\right) + \sigma\alpha}$ mod p ▷ $i = 1, \ldots, N$

$c_1, \ldots, c_N \leftarrow generate\_random\_challenge(p, g, q, y, N, \tilde{g}, \{\tilde{g}_i\}, \{(g_i, m_i)\}, \{(g'_i, m'_i)\})$ ▷ *Fiat-Shamir Heuristic*

$s = \sum_{i=1}^{N} r_i c_i + \alpha \bmod q$ ▷ *additional values*

$s_i = c_{\phi(i)} + \alpha_i \bmod q$

$\lambda' = \sum_{i=1}^{N} \lambda_i {c_i}^2 + \lambda \bmod q$

$return\ t, \upsilon, w, u, \{u_i\}, \{\tilde{g_i}'\}, \tilde{g}', g', m', \{t_i\}, \{\upsilon_i\}, \dot{\upsilon}, \{w_i\}, \dot{w}, s, \{s_i\}, \lambda'$

Algorithm 2 Verification

input: $p, g, q, y, \{(g_i, m_i)\}, \{(g'_i, m'_i)\}, \tilde{g}, \{\tilde{g}_i\}, t, \upsilon, w, u, \{u_i\}, \{\tilde{g_i}'\}, \tilde{g}', g', m', \{t_i\}, \{\upsilon_i\}, \dot{\upsilon}, \{w_i\}, \dot{w}, s, \{s_i\}, \lambda'$

$c_1, \ldots, c_N \leftarrow generate\_random\_challenge(p, g, q, y, N, \tilde{g}, \{\tilde{g}_i\}, \{(g_i, m_i)\}, \{(g'_i, m'_i)\})$ ▷ *Fiat-Shamir Heuristic*

$e_1 \leftarrow \left(\tilde{g}^s \prod_{i=1}^{N} {\tilde{g}_i}^{s_i} = \tilde{g}' \prod_{i=1}^{N} {\tilde{g}_i}'^{c_i} \right) \bmod p$

$e_2 \leftarrow \left(g^s \prod_{i=1}^{N} g_i^{s_i} = g' \prod_{i=1}^{N} g_i'^{c_i} \right) \bmod p$

$e_3 \leftarrow \left(y^s \prod_{i=1}^{N} m_i^{s_i} = m' \prod_{i=1}^{N} m_i'^{c_i} \right) \bmod p$

$e_4 \leftarrow \left(g^{\lambda'} = u \prod_{i=1}^{N} {u_i}^{c_i^2} \right) \bmod p$

$e_5 \leftarrow \left(t^{\lambda'} \upsilon^s g^{\sum_{i=1}^{N} (s_i^3 - c_i^3)} = \dot{\upsilon} \prod_{i=1}^{N} {\upsilon_i}^{c_i} {t_i}^{c_i^2} \right) \bmod p$

$e_6 \leftarrow \left(w^s g^{\sum_{i=1}^{N} (s_i^2 - c_i^2)} = \dot{w} \prod_{i=1}^{N} {w_i}^{c_i} \right) \bmod p$

$outcome \leftarrow e_1 \wedge e_2 \wedge e_3 \wedge e_4 \wedge e_5 \wedge e_6$

$return\ e_1, e_2, e_3, e_4, e_5, e_6, outcome$

΄Οπως βλέπουμε, έξοδος του πρώτου αλγορίθμου αποτελεί την απόδειξη μιας σωστής μίξη, η οποία αποτελεί την είσοδο για τον αλγόριθμο της επαλήθευσης, ώστε ένας (μη διαδραστικός) ελεγκτής να αποφανθεί για την ορθότητα της διαδικασίας.

4.2 Παραλληλοποίηση

Σε αυτή την ενότητα, παρουσιάζεται η ανάλυση της παράλληλης εκδοχής των παραπάνω αλγορίθμων. Αρχικά, η μίξη και η επαλήθευση χωρίζονται σε μικρότερες λειτουργίες, για να μπορέσει το φορτίο τους να κατανεμηθεί σε έναν αριθμό από παράλληλες εργασίες (parallel workers). Ωστόσο, υπάρχουν κάποιες εξαρτήσεις μεταξύ των δεδομένων του προβλήματος και η τεχνική παραλληλοποίησης πρέπει να διασφαλίσει την εγκυρότητα των υπολογισμών. Οι λειτουργίες αυτές παρουσιάζονται ως συναρτήσεις παρακάτω, όπου η είσοδος και η έξοδός τους φαίνετα καθαρά.

4.2.1 Shuffle

- random_permutation()
  - input: N
  - output: $\phi(i)$
- inverse_permutation()
  - input: $N, \phi(i)$
  - output: $\phi^{-1}(i)$
- generate_randomizers()
  - input: N, q
  - output: $\{r_i\}$
- permute_ciphertexts()
  - input: $N, p, g, q, y, \phi^{-1}(i), \{C_i\}, \{r_i\}$
  - output: $\{C'_i\}$
- generate_random_basis()
  - input: N, p, g, q
  - output: $\tilde{g}, \{\tilde{g}_i\}$
- generate_initial_values()
  - input: N, q
  - output: $\sigma, \rho, \tau, \alpha, \lambda, \{\alpha_i\}, \{\lambda_i\}$
- first_expos()
  - input: $N, \sigma, \rho, \tau, \lambda, \{\lambda_i\}, p$

  - output: $t, v, w, u, \{u_i\}$
- new_g_tilde_list()
  - input: $N, \phi^{-1}, \tilde{g}, \{\tilde{g}_i\}, \{r_i\}, p$
  - output: $\{\tilde{g}_i{}'\}$
- new_g_tilde()
  - input: $N, p, g, \tilde{g}, \{\tilde{g}_i\}, \alpha, \{\alpha_i\}$
  - output: $\tilde{g}'$
- new_g()
  - input: $N, p, g, \{C_i\}, \alpha, \{\alpha_i\}$
  - output: g'
- new_m()
  - input: $N, p, y, \{C_i\}, \alpha, \{\alpha_i\}$
  - output: m'
- t_list()
  - input: $N, p, g, \phi^{-1}(i), \{\alpha_i\}, \tau, \{\lambda_i\}$
  - output: $\{t_i\}$
- v_list()
  - input: $N, p, g, \phi^{-1}, \{\alpha_i\}, \rho, \{r_i\}$
  - output: $\{v_i\}$
- new_v()
  - input: $N, p, g, \{\alpha_i\}, \tau, \rho, \alpha, \lambda$
  - output: $\dot{v}$
- w_list()
  - input: $N, p, g, \phi^{-1}, \{\alpha_i\}, \sigma, \{r_i\}$
  - output: $\{w_i\}$
- new_w()
  - input: $N, p, g, \phi^{-1}, \{\alpha_i\}, \sigma, \alpha$
  - output: $\dot{w}$
- generate_random_challenge()

  - input: $N, p, g, q, y, \tilde{g}, \{\tilde{g_i}\}, \{C_i\}, \{C_i'\}$
  - output: $\{c_i\}$

- additional_values()
  - input: $N, q, \phi, \{r_i\}, \{c_i\}, \alpha, \{\alpha_i\}, \lambda, \{\lambda_i\}$
  - output: $s, \{s_i\}, \lambda'$

4.2.2 Verification

- validate_eq_1()
  - input: $N, p, \tilde{g}, \{\tilde{g_i}\}, s, \{s_i\}, \tilde{g}', \{\tilde{g_i}'\}, \{c_i\}$
  - output: $True/False$
- validate_eq_2()
  - input: $N, p, g, \{C_i\}, s, \{s_i\}, g', \{C_i'\}, \{c_i\}$
  - output: $True/False$
- validate_eq_3()
  - input: $N, p, y, \{C_i\}, s, \{s_i\}, m', \{C_i'\}, \{c_i\}$
  - output: $True/False$
- validate_eq_4()
  - input: $N, p, g, \lambda', u, \{u_i\}, \{c_i\}$
  - output: $True/False$
- validate_eq_5()
  - input: $N, p, g, t, \lambda', \upsilon, \dot{\upsilon}, \{\upsilon_i\}, s, \{t_i\}, \{s_i\}, \{c_i\}$
  - output: $True/False$
- validate_eq_6()
  - input: $N, p, g, w, \dot{w}, \{w_i\}, s, \{s_i\}, \{c_i\}$
  - output: $True/False$

4.2.3 Γράφοι εξαρτήσεων και παράλληλη μέθοδος

Μετά τον ορισμό όλων των λειτουργιών τα οποία απαρτίζουν τους δύο αρχικούς αλγορίθμους, πρέπει να εξετάσουμε όλες τις εξαρτήσεις δεδομένων μεταξύ τους, για την ορθότητα των αποτελεσμάτων. Γι-΄ αυτό το σκοπό, δημιουργούμε τους *γράφους εξαρτήσεων* οι οποίοι οπτικοποιούν τις ροές δεδομένων του προβλήματος.

Shuffle proof

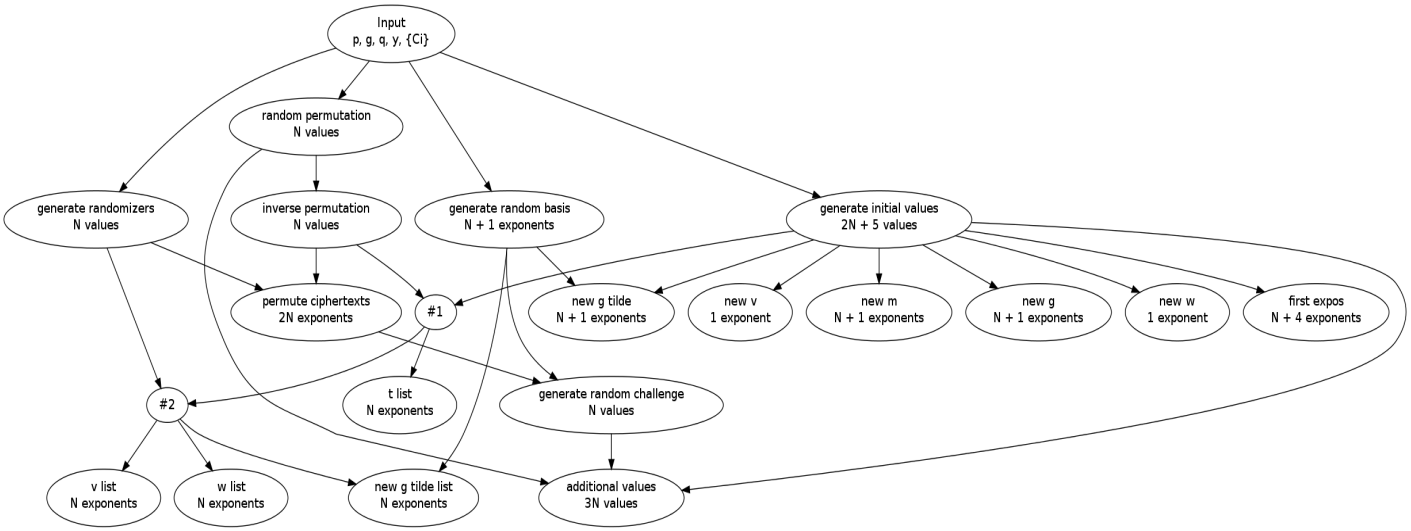

Figure 4.1: Shuffle algorithm dependencies

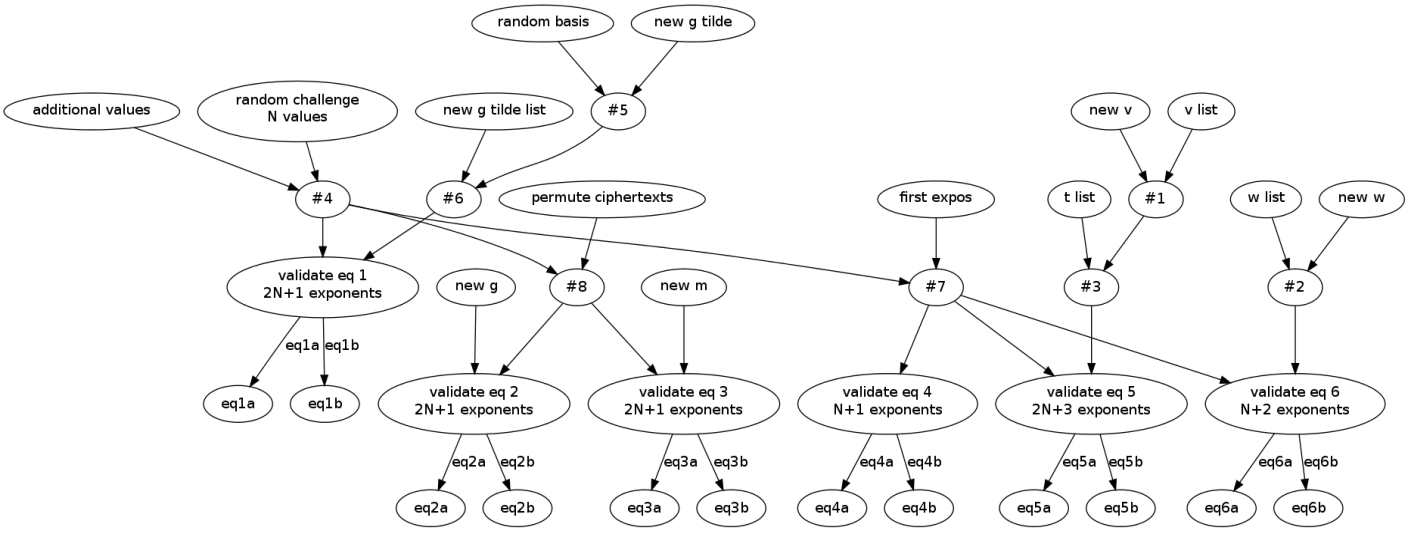

Figure 4.2: Verification algorithm dependencies

Από τα παραπάνω γραφήματα, συμπεραίνουμε ότι κάποιες λειτουργίες εκτελούνται πριν από άλλες, όπως υπαγορεύουν οι αλυσίδες δεδομένων του προβλήματος. Για παράδειγμα, δε μπορούμε να κάνουμε

επανακρυπτογράφηση, αν δεν υπολογίσουμε πρώτα τις παραμέτρους αυτής και την αντίστροφη μετάθεση. Επιπρόσθετα, αναγράφεται και το κόστος της κάθε λειτουργίας, εκφρασμένο σε αριθμό εκθετικών. ΄Οπως έχουμε ήδη αναφέρει, τα εκθετικά είναι τάξεις πιο αργοί υπολογισμοί από όλους τους υπόλοιπους στο παρόν πρωτόκολλο. Οι λειτουργίες που δεν περιλαμβάνουν εκθετικά, μπορούν να εκτελεστούν σειριακά, αλλά επιλέξαμε να γίνουν και αυτές παράλληλα, καθώς θα παίξουν καθοριστικό ρόλο στην επίδοση του προγράμματος σε μεγάλες εισόδους και κατ΄ επέκταση στη μέγιστη επιτάχυνση της μίξης.

Οπότε, η διαδικασία της παραλληλοποίησης συνοψίζεται σε τρία βήματα: πρώτον, η είσοδος διαβάζεται και δημιουργείται ο επιθυμητός αριθμός (ανεξάρτητων) εργασιών. Δεύτερον, κάθε εργασία εκτελεί παράλληλους υπολογισμούς και, τρίτον, όλα τα αποτελέσματα συγκεντρώνονται σε μια ενιαία απόδειξη ή σε μία πρόταση επαλήθευσης, ακολούθως.

Η ακριβής μέθοδος που χρησιμοποιήσαμε για να κατανείμουμε το συνολικό αριθμό εκθετικών ανά εργασία φαίνεται παρακάτω, όπου η καρδιά της διαδικασίας είναι στην ακόλουθη συνάρτηση:

Algorithm 3

function GET_TASK_INPUTS(length, nr_tasks, task_no)
/* *take the number of input items, number of tasks and return the indices for each task_no* */

$current \leftarrow task\_no$
$idxs \leftarrow \{\}$
while $current < lentgh$ **do**
$idxs \leftarrow idxs \cup \{current\}$
$current = current + nr\_tasks$
return idxs

Η συνάρτηση επιστρέφει μια λίστα με δείκτες για την εκάστοτε εργασία, που υπαγορεύουν σε ποιες θέσεις αντιστοιχεί κάθε αποτέλεσμα-υπολογισμός της. ΄Οπως είναι προφανές, η συνάρτηση υπολογίζει τους δείκτες με κυκλικό τρόπο (cyclic). Επομένως, έχουμε να κάνουμε με πράξεις πάνω σε διανύσματα (vector calculations) των οποίων υποσύνολα χρησιμοποιούν διαφορετικές παράλληλες εργασίες. Στο μοντέλο του παράλληλου προγραμματισμού, αυτό αντιστοιχεί με μια SIMD (single instruction - multiple data) λογική (ταξινόμηση του Flynn).

Algorithm 4 Break tasks

input: nr_tasks, infile, directory, mode

/* *read the input ciphertexts and the cryptosystem parameters* */
$p, g, q, y, \{C_i\}, N \leftarrow$ READ_INPUT(infile)

if $num\_tasks > N$ **then**
  $num\_tasks \leftarrow N$ ▷ each operation has at most N exponents
if mode is 'shuffle' **then**
  /* *calculate random values and any common data sequentially* */
  $\phi \leftarrow$ RANDOM_PERMUTATION(N)
  $\phi^{-1} \leftarrow$ INVERSE_PERMUTATION(ϕ)
  $\sigma, \rho, \tau, \alpha, \lambda, \{\alpha_i\}, \{\lambda_i\} \leftarrow$ GENERATE_RANDOM(2, q)
  $\tilde{g_0} \leftarrow$ GENERATE_RANDOM_ELEMENT(p, g, q)
  $sum1 \leftarrow$ SUM($\{\alpha_i^2\}$)
  $sum2 \leftarrow$ SUM($\{\alpha_i^3\}$)
  $\{R_i\} \leftarrow$ CREATE_TASK_FILES($nr\_tasks, directory$)
  for $i = 0$ to $num\_tasks - 1$ **do**
    $my\_idxs \leftarrow$ GET_TASK_INPUTS(N, nr_tasks, i)
    for $k \in my\_idxs$ **do**
      SAVE_IN_MY_TASK($C_k, C_{\phi_k^{-1}}, \alpha_k, \alpha_{\phi_k^{-1}}, \lambda_k, \phi_k, \phi_k^{-1}, R_i$)
  /**we have not assigned these operations to any task yet**/
  $remaining \leftarrow \{\dot{v}, \dot{w}, \tilde{g}^\alpha, g^\alpha, y^\alpha, t, v, w, u\}$
  for $i = 0$ to LENGTH($remaining$) -1 **do**
    $target = i$ mod nr_tasks
    SAVE_IN_MY_TASK($remaining_i, R_{target}$)
else if mode is 'verify' **then**
  $proof \leftarrow$ READ_INPUT(infile)
  $\{c_i\} \leftarrow$ GENERATE_RANDOM_CHALLENGE($proof$)
  $sum1 \leftarrow$ SUM($\{s_i^3 - c_i^3\}$)
  $sum2 \leftarrow$ SUM($\{s_i^2 - c_i^2\}$)
  $\{R_i\} \leftarrow$ CREATE_TASK_FILES($nr\_tasks, proof$)
  for $i = 0$ to $nr\_tasks - 1$ **do**
    $my\_idxs \leftarrow$ GET_TASK_INPUTS(N, nr_tasks, i)
    for $k \in my\_idxs$ **do**
      SAVE_IN_MY_TASK($\tilde{g}_k, s_k, \tilde{g}'_k, c_k, g_k, g'_k, m_k, m'_k, u_k, c_k^2, v_k, t_k, w_k, R_i$)
  $remaining \leftarrow (\{\tilde{g}^s, \tilde{g}'), (g^s, g'), (y^s, m'), (g^{\lambda'}, u), (t^{\lambda'}, \dot{v}), v^s, g^{sum1}, (w^s, \dot{w}), g^{sum2}\}$
  for $i = 0$ to LENGTH($remaining$) -1 **do**
    $target = i$ mod nr_tasks
    SAVE_IN_MY_TASK($remaining_i, R_{target}$)

Βλέπουμε ότι η διαδικασία της κατανομής των υπολογισμών και στις δύο περιπτώσεις είναι ίδια, χρησιμοποιώντας τη συνάρτηση που ορίσαμε προηγουμμένως. Ο διαχωρισμός τους γίνεται με μία λέξη-κλειδί (mode) και ο μέγιστος αριθμός παράλληλων εργασιών δεν μπορεί να ξεπεράσει τον αριθμό των κρυπτοκειμένων εισόδου, που δίνονται ως αρχείο (infile). Το όρισμα 'directory' αναφέρεται για πληρότητα, είναι σχετικό με την υλοποίηση και μπορεί να αγνοηθεί. Επίσης, η κατανομή των υπολογισμών γίνεται σε δύο βήματα, στο πρώτο μοιράζονται εκείνοι που είναι της τάξης του N, και έπειτα όλοι οι μεμονωμένοι (σταθερός αριθμός) που ορίζει το πρωτόκολλο (remaining). ΄Αρα, όλες οι εργασίες δέχονται ομάδες των (N/nr_tasks) εκθετικών για υπολογισμό και οι πρώτες $(N \bmod nr\_tasks)$ λαμβάνουν ένα παραπάνω (υποθέτουμε ότι οι εργασίες αριθμούνται - 0, 1, ...). Το μειονέκτημα αυτής της κατανομής είναι ότι αν μοιράζουμε πολλές λειτουργίες των N εκθετικών, τότε αν δε διαιρείται ακριβώς το N με τον αριθμό των παράλληλων εργασιών, κάποιες εργασίες θα καταλήξουν να υπολογίζουν ελαφρώς περισσότερα εκθετικά από τις υπόλοιπες, άρα, η κατανομή δεν είναι ομοιόμορφη.

Τώρα που οι εργασίες δημιουργήθηκαν μπορούν να τρέξουν παράλληλα χρησιμοποιώντας τον παρακάτω αλγόριθμο:

Algorithm 5 Compute

input: task_file, directory, mode

$p, g, q, y, \{C_i\}, N \leftarrow$ READ_INPUT($task\_file$)
$data \leftarrow$ READ_INPUT($task\_file$)
if mode is 'shuffle' **then**
  /* *read all the necessary data for the computations as input e.g. cryptosystem parameters, other expected values, etc* */
  $my\_idxs, data \leftarrow$ READ_INPUT(task_file)
  $R \leftarrow$ CREATE_MY_RESULTS_FILE($directory$)
  /* *run each shuffle operation for this task file's indices* */
  $\{r_i\} \leftarrow$ GENERATE_RANDOMIZERS($my\_idxs, data$)
  $\{C'_i\} \leftarrow$ PERMUTE($my\_idxs, \{r_i\}, data$)
  $\{\tilde{g}_i\} \leftarrow$ GENERATE_RANDOM_BASIS($my\_idxs, data$)
  $\{\tilde{g}'_i\} \leftarrow$ NEW_G_TILDE_LIST($my\_idxs, data$)
  $\tilde{g}' \leftarrow$ NEW_G_TILDE($my\_idxs, data$)
  $g' \leftarrow$ NEW_G($my\_idxs, data$)
  $m' \leftarrow$ NEW_M($my\_idxs, data$)
  $\{t_i\} \leftarrow$ T_LIST($my\_idxs, data$)

$\{v_i\} \leftarrow$ V_LIST($my\_idxs, data$)
$\{w_i\} \leftarrow$ W_LIST($my\_idxs, data$)
$\{u_i\} \leftarrow$ U_LIST($my\_idxs, data$)
/ save results to file */*
SAVE_IN_MY_RESULTS_FILE($\{r_i\}, \ldots, \{w_i\}, R$)
$remaining \leftarrow \{\tilde{g}^{\alpha}, g^{\alpha}, y^{\alpha}, \dot{v}, \dot{w}, t, v, w, u\}$
for op in $remaining$ **do**
  if op in $task\_file$ **then**
    $res \leftarrow$ COMPUTE_OP(op)
    SAVE_IN_MY_RESULTS_FILE(res, R)
else if mode is 'verify' **then**
  $R \leftarrow$ CREATE_MY_RESULTS_FILE($directory$)
  $my\_idxs, data \leftarrow$ READ_INPUT(task_file)
  / prodX{1,2} indicate the left-hand and right-hand products of equalities 1, ..., 6 of the verification phase */*
  $prod_{11}, prod12 \leftarrow$ COMPUTE_EQ$_1$($my\_idxs, data$)
  $prod_{21}, prod_{22} \leftarrow$ COMPUTE_EQ$_2$($my\_idxs, data$)
  $prod_{31}, prod_{32} \leftarrow$ COMPUTE_EQ$_3$($my\_idxs, data$)
  $prod_{42} \leftarrow$ COMPUTE_EQ$_4$($my\_idxs, data$)
  $prod_{52} \leftarrow$ COMPUTE_EQ$_5$($my\_idxs, data$)
  $prod_{62} \leftarrow$ COMPUTE_EQ$_6$($my\_idxs, data$)
  SAVE_IN_MY_RESULTS_FILE($prod_{11}, \ldots, prod_{62}, R$)
  $remaining \leftarrow \{(\tilde{g}^s, \tilde{g}'), (g^s, g'), (y^s, m'), (g^{\lambda'}, u), (t^{\lambda'}, \dot{v}), v^s, g^{sum1}, (w^s, \dot{w}), g^{sum2}\}$
  for op in $remaining$ **do**
    if op in $task\_file$ **then** $res \leftarrow$ COMPUTE_OP(op)
      SAVE_IN_MY_RESULTS_FILE(res, R)

Οι εργασίες μπορούν είτε να αποθηκεύουν τα αποτελέσματα είτε σε κάποιο αρχείο, όπως φαίνεται παραπάνω, είτε επικοινωνούν κατευθείαν με άλλες εργασίες που χρειάζονται αυτά τα δεδομένα (πχ η εργασία που εκτελεί τις λειτουργίες 'break' και 'combine').

Το τελευταίο στάδιο είναι η *συλλογή* των αποτελεσμάτων από τις παράλληλες εργασίες, που φαίνεται στον παρακάτω αλγόριθμο:

Algorithm 6 Combine results

input: result_files, outfile, mode

$p, g, q, y, N \leftarrow$ READ_INPUT($task\_file$)
if mode is 'shuffle' **then**
  /* *proof contains the empty lists and values* $\{r_i\}$, $\{C_i\}, \{C'_i\}, m', g', \{\tilde{g}_i\}, \ldots, \{t_i\}, \{u_i\}$ */

  $proof \leftarrow$ INITIALIZE_PROOF(N)
  for f in $result\_files$ **do**
    /* *read results from a file and put them in the right positions (my_idxs) of the proof lists* */

    $my\_idxs, my\_\{r_i\}, my\_\{C_i\}, my\_\{C'_i\}, my\_m',$
    $my\_g', my\_\{\tilde{g}_i\}, \ldots, my\_\{t_i\}, my\_\{u_i\} \leftarrow$ READ_INPUT(f)
    INSERT_TO_PROOF($proof, my\_idxs, my\_\{r_i\}, \ldots, my\_\{u_i\}$)
  $\{c_i\} \leftarrow$ GENERATE_RANDOM_CHALLENGE($proof$)
  $s, \{s_i\}, \lambda' \leftarrow additional\_values(\{c_i\})$
  INSERT_TO_PROOF($proof, s, \{s_i\}, \lambda'$)
  SAVE_PROOF_TO_FILE($outfile$)
else if mode is 'verify' **then**
  /* *initialize products* */
  $prod_{11}, prod_{12}, \ldots, prod_{61}, prod_{62} \leftarrow 1$
  for f in $result\_files$ **do**
    $p_{11}, p_{12}, \ldots, p_{61}, p_{62} \leftarrow$ READ_INPUT(f)
    $update\_products(p_{11}, p_{12}, \ldots, p_{61}, p_{62}, prod_{11}, prod_{12}, \ldots, prod_{61}, prod_{62})$
  /* *check if all verification equalities stand* */
  if $prod_{11} = prod_{12} \wedge \ldots \wedge prod_{61} = prod_{62}$ **then**
    SAVE_TO_FILE($outfile$, $VALID$)
  else
    SAVE_TO_FILE($outfile$, $INVALID$)

Τονίζεται ότι τα περισσότερα σειριακά μέρη που φαίνονται στις παραπάνω διαδικασίες (αθροίσματα, αντίστροφη μετάθεση) γίνονται και αυτά παράλληλα, γιατί θα παίξουν σημαντικό ρόλο στην επίδοση του προγράμματος, όπως θα φανεί σε επόμενη ενότητα. Ο λόγος που παρουσιάζονται σειριακά είναι για απλότητα, ώστε να τονιστεί καλύτερα η ιδέα της παραλληλοποίησης. Τελικά, ο τρόπος που θα εκτελεστούν παράλληλα αυτά τα 'σειριακά' τμήματα, δε διαφέρει από εκείνον που παρουσιάζεται παραπάνω.

Για να καταλάβουμε καλύτερα πώς τα δεδομένα μοιράζονται σε nr_tasks παράλληλες εργασίες, ας θεωρήσουμε το παρακάτω παράδειγμα για μία από τις λειτουργίες που περιγράφηκαν πριν, όπου υπο-

θέτουμε μια είσοδο 10 κρυπτοκειμένων και τριών παράλληλων εργασιών.

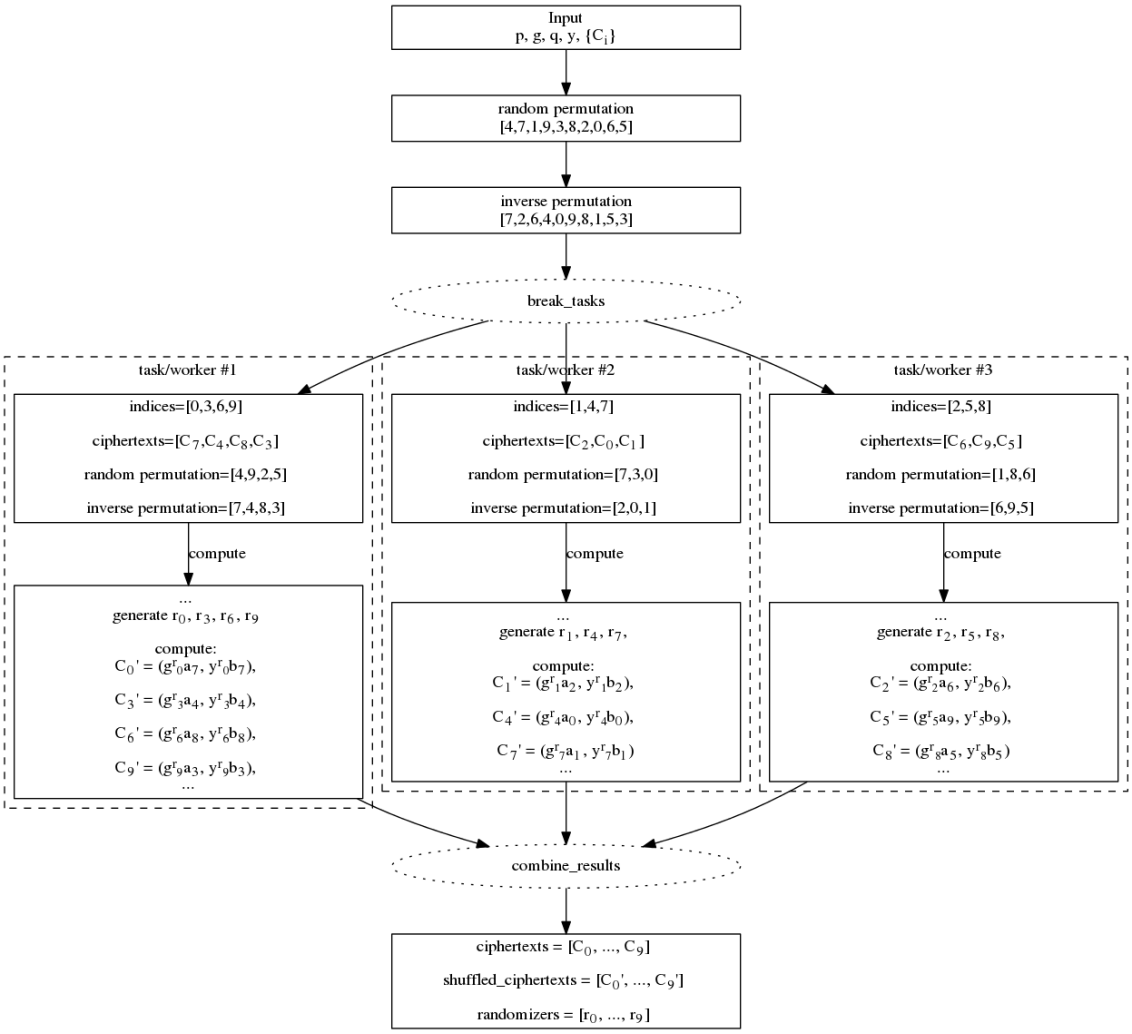

Figure 4.3: Example of permutation and re-encryption in parallel

Εδώ φαίνεται η κυκλική κατανομή με βάση την οποία σχεδιάσαμε την παραλληλοποίηση του παρόντος σχήματος. Κάθε εργασία εκτελεί υπολογισμούς ανεξάρτητα η μία από την αάλλη και τα μόνα σημεία επικοινωνίας είναι στις λειτουργίες του ‘break’ και ‘combine’ με μία master διεργασία (worker). Επίσης, μπορούμε να δούμε ότι μερικές λειτουργίες μπορούν να συγχωνευθούν, πχ η δημιουργία των τυχαίων παραμέτρων επανακρυπτογράφησης μπορεί να γίνει μέσα στην

ίδια τη λειτουργία της επανακρυπτογράφησης και να μην συγκαταλέγεται ξεχωριστά. Επιπλέον, και πάλι για απλότητα, η αντίστροφη μετάθεση υπολογίζεται σειριακά. Αυτό το παράδειγμα δείχνει ακριβώς ποια είναι η έννοια του όρου '*παραλληλοποιημένοι υπολογισμοί διανυσμάτων*' (parallelized vector calculations).

Στην επόμενη ενότητα ποσοτικοποιούμε την ανάλυσή μας με την πρόβλεψη του χρόνου εκτέλεσης και της επιτάχυνσης του παραλληλοποιημένου πρωτοκόλλου. Πριν όμως από αυτό, είναι σημαντικό να αιτιολογήσουμε τον τρόπο της παραλληλοποίησης. ΄Οπως ειπώθηκε στην αρχή, η εργασία αυτή πραγματοποιήθηκε στα πλαίσια του συστήματος ηλεκτρονικών εκλογών 'Ζευς'. ΄Ετσι, **επιλέγουμε** να μην παραλληλοποιήσουμε όλες τις λειτουργίες του προβλήματος, όπως η παραγωγή τυχαίων μεταθέσεων ή διαδικασίες κατακερματισμού (hashing operations). Αυτό συμβαίνει λόγω περιορισμών και θεμάτων ασφαλείας που υποβάλλονται από το Ζευς. Η αρχιτεκτονική του θα κρίνει πώς θα (και αν μπορεί να) χρησιμοποιήσει την υλοποίησή μας στη ροή εργασιών του. Ωστόσο, αυτό που κάναμε, ήταν να παραλληλοποιήσουμε σχεδόν ολόκληρο τον υπόλοιπο αλγόριθμο (ακόμα και τα 'μεγάλα' αθροίσματα και πολλαπλασιασμούς) και ειδικότερα όλους τους υπολογισμούς εκθετικών, που είναι και το κέντρο προσοχής.

Χρόνος εκτέλεσης και πρόβλεψη επιτάχυνσης

Σε αυτό το κομμάτι της ανάλυσής, θα επιχειρήσουμε να δημιουργήσουμε μια εκτίμηση για τον χρόνο εκτέλεσης και, τελικά, για την επιτάχυνση που μπορούμε να πετύχουμε με το παραλληλοποιημένο πρωτόκολλο. Πριν από αυτό όμως, είναι απαραίτητο να αναφέρουμε κάποιες θεμελιώδεις ιδιότητες των παράλληλων προγραμμάτων.

Ο νόμος του Amdahl

Οι παράλληλοι αλγόριθμοι αποτελούν την προσπάθεια για βελτίωση της επίδοσης μιας λύσης σε ένα συγκεκριμένο πρόβλημα με τη χρήση περισσότερων από έναν πυρήνων/επεξεργαστών. Η πιο κοινή μετρική γι΄ αυτή τη σχετική επίδοση είναι η *επιτάχυνση* (speedup), η οποία εδραιώθηκε από τον Amdahl. Με απλά λόγια ορίζει την επιτάχυνση ως

$$S = \frac{T_s}{T_p} = \frac{1}{f + \frac{1-f}{P}}$$

όπου T_s υποδεικνύει το χρόνο εκτέλεσης του (καλύτερου) σειριακού αλγόριθμου που λύνει ένα πρόβλημα και T_p είναι ο συνολικός χρόνος εκτέλεσης χρησιμοποιώντας P πυρήνες για τον αλγόριθμο αυτό. Εναλλακτικά, το f είναι το ποσοστό του προγράμματος που είναι αυ-

στηρά σειριακό (δε μπορεί να παραλληλοποιηθεί) και $1 - f$ είναι το ποσοστό του προγράμματος που μπορεί να εκτελεστεί παράλληλα, με T_s να είναι κανονικοποιημένο στο 1. Η σημαντικότητα του τύπου αυτού φαίνεται καλύτερα με ένα παράδειγμα: αν καταφέρουμε να παραλληλοποιήσουμε το 90% ενός προγράμματος με χρήση δέκα πυρήνων και το υπόλοιπο 10% εκτελείται σειριακά, τότε ο νόμος του Amdahl προβλέπει μία μέγιστη επιτάχυνση περίπου 5.2. Αυτό σημαίνει ότι πρέπει να κρατήσουμε το σειριακό μέρος ενός προγράμματος όσο πιο μικρό μπορούμε, για να μπορέσουμε να πετύχουμε μια μεγάλη επιτάχυνση.

Αν και ο νόμος του Amdahl είναι ευρέως χρησιμοποιούμενος στα παράλληλα συστήματα, δεν περιλαμβάνει άλλες επιβαρύνσεις σε ένα πρόγραμμα, όπως την επικοινωνία μεταξύ επεξεργαστών, συγχρονισμό κλπ. Τα αποτελέσματά του μπορούν να οπτικοποιηθούν στο παρακάτω γράφημα.

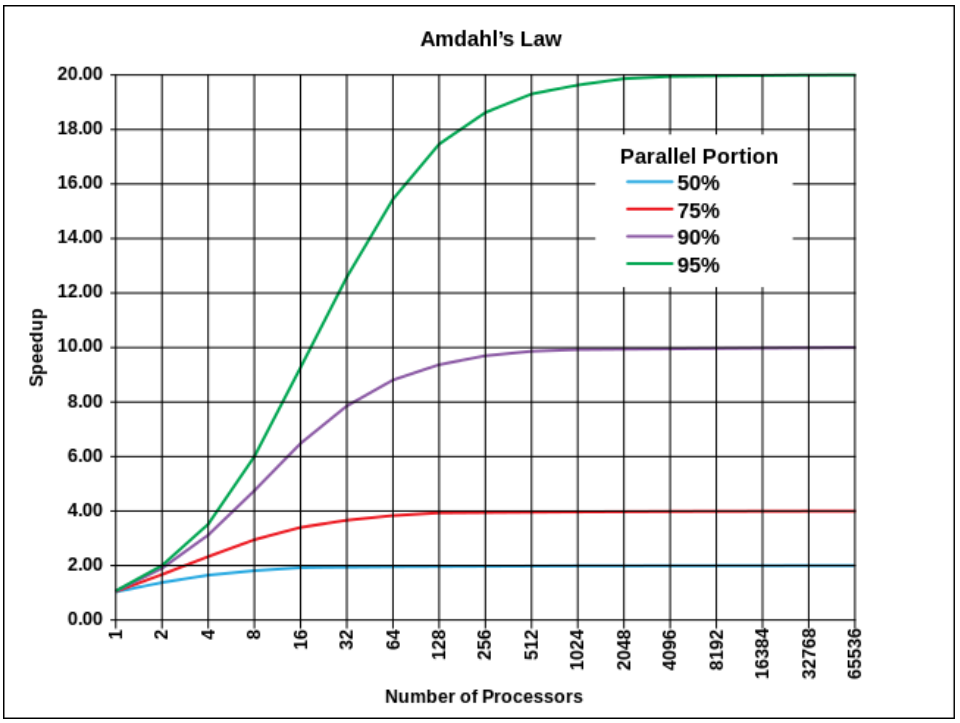

Figure 4.4: Amdahl's Law for speedup

Συνεπώς, στόχος του παράλληλου προγραμματισμού είναι να παραλληλοποιήσει όσο περισσότερο μπορεί τα σειριακά τμήματα, αν και είναι ένα από τις πιο δύσκολες εργασίες ως κλειδί για την επιτάχυνση.

Πρόβλεψη χρόνου εκτέλεσης

Αρχικά, θεωρούμε ότι ο χρόνος εκτέλεσης της παράλληλης υλοποίησής μας διαιρείται σε τρία μέρη: το σειριακό, το παράλληλο και το μέρος της επικοινωνίας. Το σειριακό κομμάτι προέρχεται από τις λειτουργίες 'break' και 'combine' που περιγράφτηκαν στην προηγούμενη ενότητα, το παράλληλο κομμάτι, προφανώς, αντιπροσωπεύει τον κώδικα που τρέχει πάνω σε έναν αριθμό επεξεργαστών και είναι κυρίως ο χρόνος που κάνει το πρωτόκολλο να υπολογίσει τις δυνάμεις (εκθετικά) και, τέλος, το κομμάτι της επικοινωνίας αφορά στο χρόνο που οφείλεται στις μεταφορές δεδομένων μεταξύ των επεξεργαστών/πυρήνων. Το σχήμα των Furukawa-Sako έχει μεγάλο βαθμό παραλληλίας (δεδομένων), το οποίο μπορούμε να εκμεταλλευτούμε για να δημιουργήσουμε απομονωμένα σετ υπολογισμών ανάμεσα στους επεξεργαστές, όπως έχει δειχθεί και από την παραλληλοποίηση στην αρχή του κεφαλαίου.

Για την περίπτωση του παράλληλου τμήματος του χρόνου εκτέλεσης, προχωρούμε ως εξής. Αν χωρίσουμε N εκθετικά σε nr_tasks παράλληλες εργασίες (workers), τότε η καθεμία θα λάβει $N/nr\_tasks > 0$ υπολογισμούς, αν $N > nr\_tasks$, και μένουν ακόμη $N \bmod nr\_tasks$, οι οποίοι δίνονται στις εργασίες $0 \leq i \leq (N \bmod nr\_tasks - 1)$. Αν $N < nr\_tasks$, τότε ο μέγιστος αριθμός εργασιών που δημιουργείται είναι N, εξαιτίας του ότι μοιράζουμε 'ομάδες' από N εκθετικά κάθε φορά (μία ξεχωριστή λειτουργία). Αν υπάρχουν P διαθέσιμοι επεξεργαστές, η μέγιστη παραλληλία δεδομένων επιτυγχάνεται όταν $nr\_tasks = P$. Από τους γράφους εξαρτήσεων, απορρέει ότι υπάρχουν έντεκα 'ομάδες' από N εκθετικά και εννέα 'εναπομείναντα' (remaining) για τη μίξη και δέκα 'ομάδες' N εκθετικών και εννέα 'εναπομείναντα' για τη φάση της επαλήθευσης. Επομένως, κάθε εργασία-εργάτης P_i (εδώ υποθέτουμε ότι οι εργάτες είναι επεξεργαστές), λαμβάνει φορτίο:

$$\text{SHUFFLE}(P_i, P) = 11 \cdot \left[\left[\frac{N}{P}\right] + \begin{cases} 1, & \text{if } i < (N \bmod P) \\ 0, & \text{otherwise} \end{cases}\right] + \left[\frac{9}{P}\right] + \begin{cases} 1, & \text{if } i < (9 \bmod P) \\ 0, & \text{otherwise} \end{cases}$$

$$\text{VERIFY}(P_i, P) = 10 \cdot \left[\left[\frac{N}{P}\right] + \begin{cases} 1, & \text{if } i < (N \bmod P) \\ 0, & \text{otherwise} \end{cases}\right] + \left[\frac{9}{P}\right] + \begin{cases} 1 & \text{, if } i < (9 \bmod P) \\ 0, & \text{otherwise} \end{cases}$$

΄Ομοια με την παραπάνω ιδέα, η πολυπλοκότητα επικοινωνίας σε bits ανά επεξεργαστή μπορεί να κατασκευαστεί με τον ακόλουθο τρόπο,

όπου *"send"* δηλώνει τα δεδομένα που δίνονται σε μια μονάδα επεξεργασίας (CPU) ως είσοδο και *"receive"* δηλώνει τα δεδομένα που η μονάδα επεξεργασίας επιστρέφει ως έξοδο. Για καθαρότητα, οι τύποι έχουν αντικατασταθεί από συναρτήσεις, προσαρμοσμένες στο σχήμα παραλληλοποίησης. Επίσης, θεωρούμε ότι $P > 1$ και αριθμούς B bits.

---

function SHUFFLE_SEND(P_i, P, N)
    /* *return the numbers sent to processor* P_i */
    if $P < 2$ **then**
        return 0
    $numbers \leftarrow 10 + 7 \cdot \left[\frac{N}{P}\right]$
    if $i < (N \bmod P)$ **then**
        $numbers \leftarrow numbers + 7$
    $idxs \leftarrow$ GET_TASK_INPUTS$(9, P, i)$
    if $2 \in idxs$ **then**
        $numbers \leftarrow numbers + 1$
    if $3 \in idxs$ **then**
        $numbers \leftarrow numbers + 1$
    $numbers \leftarrow numbers + 2 + (N/P)$
    return $numbers \cdot B$

function SHUFFLE_RECEIVE(P_i, P, N)
    /* *return the numbers received from processor* P_i */
    if $P < 2$ **then**
        return 0
    $numbers \leftarrow 3 + 9 \cdot \left[\frac{N}{P}\right]$
    if $i < (N \bmod P)$ **then**
        $numbers \leftarrow numbers + 9$
    $idxs \leftarrow$ GET_TASK_INPUTS$(9, P, i)$
    for $i = 2, \ldots, 7$ **do**
        if $i \in idxs$ **then**
            $numbers \leftarrow numbers + 1$
    $numbers \leftarrow numbers + (N/P) + 3$
    return $numbers \cdot B$

```
function VERIFICATION_SEND(P_i, P, N)
    /* return the numbers sent to processor P_i */
    if P < 2 then
        return 0
    numbers ← 4 + 12 · [N/P]
    if i < (N mod P) then
        numbers ← numbers + 12
    idxs ← GET_TASK_INPUTS(9, P, i)
    /* do not send the same number more than once */
    sent ← {}
    if 0 ∈ idxs then
        sent ← sent ∪ {s}
        numbers ← numbers + 3
    if 1 ∈ idxs then
        if s ∉ sent then
            sent ← sent ∪ {s}
            numbers ← numbers + 1
        numbers ← numbers + 1
    if 2 ∈ idxs then
        if s ∉ sent then
            sent ← sent ∪ {s}
            numbers ← numbers + 1
        numbers ← numbers + 1
    if 3 ∈ idxs then
        sent ← sent ∪ {l'}
        numbers ← numbers + 2
    if 4 ∈ idxs then
        if l' ∉ sent then
            numbers ← numbers + 1
        numbers ← numbers + 2
    if 5 ∈ idxs then
        if s ∉ sent then
            sent ← sent ∪ {s}
            numbers ← numbers + 1
        numbers ← numbers + 1
    if 6 ∈ idxs then
        numbers ← numbers + 1
    if 7 ∈ idxs then
        if s ∉ sent then
            numbers ← numbers + 1
        numbers ← numbers + 2
```

```
    if 8 ∈ idxs then
        numbers ← numbers + 1
    numbers ← numbers + 2(N/P) + 1
    return numbers · B

function VERIFY_RECEIVE(P_i, P, N)
    /* return the numbers received from processor i */
    if P < 2 then
        return 0
    return 14 · B
```

Η τελευταία παράμετρος για να ολοκληρώσει την εκτίμηση του χρόνου εκτέλεσης του παράλληλου πρωτοκόλλου είναι τα σειριακά τμήματα της κάθε φάσης (“break” και “combine”). Οι λειτουργίες που εμπλέκονται, όπως παρουσιάστηκαν νωρίτερα, υποδεικνύουν μία γραμμική σχέση ($O(N)$) μεταξύ του χρόνου των σειριακών αυτών τμημάτων και των κρυπτοκειμένων εισόδου. Για μικρό αριθμό κρυπτοκειμένων, μερικές από αυτές δεν έχουν σημαντικό αντίκτυπο στο συνολικό χρόνο εκτέλεσης, όμως για μεγάλο N είναι αυτές που θα καθορίσουν την επίδοση. Εκείνες που λήφθηκαν υπόψη σε αυτή την ανάλυση είναι η δημιουργία τυχαίων αριθμών και δοκιμασιών, η δημιουργία των τυχαίων μεταθέσεων, οι αρχικοποιήσεις και διασχίσεις λιστών και ο υπολογισμός αθροισμάτων. Επομένως, η σχέση για την κάθε φάση γίνεται:

$$\text{SHUFFLE\_SERIAL\_TIME}(N) = \alpha N + \text{E}$$

$$\text{VERIFY\_SERIAL\_TIME}(N) = \beta N$$

όπου E είναι ο μέσος χρόνος ανά εκθετικό.

΄Ετσι λοιπόν, συνδυάζοντας όλες τις προηγούμενες σχέσεις, ο συνολικός χρόνος εκτέλεσης για κάθε φάση διαμορφώνεται ως εξής:

$$
\begin{aligned}
\text{time(shuffle)} = & \max_{0 \le i \le (P-1)} (\textsc{shuffle}(P_i, P)) \cdot \mathrm{E} + \\
& \sum_{i=0}^{P-1} \textsc{shuffle\_send}(P_i, P, N) \cdot \frac{1}{B} \cdot \mathrm{D} + \\
& \sum_{i=0}^{P-1} \textsc{shuffle\_receive}(P_i, P, N) \cdot \frac{1}{B} \cdot \mathrm{D} + \\
& \textsc{shuffle\_serial\_time}(N) \Rightarrow \\
\text{time(shuffle)} = & \left[11 \left(\left[\frac{N}{P} \right] + 1 \right) + \left[\frac{9}{P} \right] + 1 \right] \cdot \mathrm{E} + \\
& (18N + 18P + 8) \cdot \mathrm{D} + \\
& \alpha N + \mathrm{E},\ P > 1.
\end{aligned}
$$

$$
\begin{aligned}
\text{time(verification)} = & \max_{0 \le i \le (P-1)} (\textsc{verify}(P_i, P)) \cdot \mathrm{E} + \\
& \sum_{i=0}^{P-1} \textsc{verify\_send}(P_i, P, N) \cdot \frac{1}{B} \cdot \mathrm{D} + \\
& \sum_{i=0}^{P-1} \textsc{verify\_receive}(P_i, P, N) \cdot \frac{1}{B} \cdot \mathrm{D} + \\
& \textsc{verify\_serial\_time}(N) \Rightarrow \\
\text{time(verification)} = & \left[10 \left(\left[\frac{N}{P} \right] + 1 \right) + \left[\frac{9}{P} \right] + 1 \right] \cdot \mathrm{E} + \\
& \mathrm{D} \cdot \left[12 + 14N + \begin{cases} 4, & \text{if } P = 2 \\ 5, & \text{if } P = 3 \\ 6, & \text{if } P = 4 \\ 5, & \text{if } P = 5 \\ 6, & \text{if } P = 6 \\ 6, & \text{if } P = 7 \\ 7, & \text{otherwise} \end{cases} + \right. \\
& \left. 19P \right] + \beta N,\ P > 1.
\end{aligned}
$$

όπου υποθέτωνται αριθμοί των B bits, E είναι ο μέσος χρόνος ανά εκθετικό και D είναι ο μέσος χρόνος για την αποστολή/λήψη ενός $B - bit$ αριθμού προς/από μία διεργασία χρησιμοποιώντας κάποιο κανάλι επικοινωνίας.

Συνεχίζοντας, μπορούμε σε αυτό το σημείο να ορίσουμε και την επιτάχυνση για τα δύο στάδια του πρωτοκόλλου, όπου T_s είναι ο χρόνος που κάνει το συνολικό πρόγραμμα με χρήση ενός επεξεργαστή ($P = 1$).

$$S_s = \frac{\text{time(shuffle)}|_{P=1}}{\text{time(shuffle)}|_P}$$

$$S_v = \frac{\text{time(verification)}|_{P=1}}{\text{time(verification)}|_P}$$

΄Εχοντας εισάγει το νόμο του Amdahl, περιμένουμε ότι η επιτάχυνση και για τις δύο φάσεις θα κλιμακώνει μέχρι ένα συγκεκριμένο σημείο, όπου θα παρουσιάζει και μια μέγιστη τιμή. Αυτό οφείλεται και στο σειριακό και στο κομμάτι επικοινωνίας των αλγορίθμων. Συγκεκριμένα, το σειριακό τμήμα είνα σταθερό, το παράλληλο μειώνεται και το κομμάτι της επικοινωνίας αυξάνεται με την προσθήκη περισσότερων επεξεργαστών για μια συγκεκριμένη τιμή της εισόδου (κρυπτοκείμενα). Ο αριθμός επεξεργαστών όπου η μέγιστη τιμή της επιτάχυνσης παρατηρείται προκύπτει όταν ο συνολικός χρόνος της παράλληλης εκτέλεσης φτάνει την ελάχιστη τιμή του ή, εναλλακτικά, οι ρυθμοί μεταβολής του παράλληλου και του τμήματος της επικοινωνίας είναι ίσοι και αντίθετοι (σταθερή είσοδος):

$$\frac{dS_s}{dP} = 0 \Rightarrow P = \sqrt{\frac{(11N+9)E}{18D}}$$

$$\frac{dS_v}{dP} = 0 \Rightarrow P = \sqrt{\frac{(10N+9)E}{19D}}$$

Στην επόμενη ενότητα αυτές οι δύο εκφράσεις αποτυπώνονται σε ξεχωριστά γραφήματα, μέσα από τα οποία θα βλέπουμε την ανάγκη σε υπολογιστική δύναμη που χρειαζόμαστε για τη μίξη για πραγματικές συνθήκες. Περισσότερα συμπεράσματα για τα διαγράμματα επιτάχυνσης εξηγούνται παρακάτω.

Επίδοση

Σε αυτό το μέρος, παρουσιάζεται η επίδοση της *παράλληλης* μίξης και επαλήθευσης. Οι δύο αλγόριθμοι δοκιμάστηκαν σε ένα εικονικό μηχάνημα με 8 πυρήνες στα 2.1 GHz για διάφορες τιμές των κρυπτοκειμένων εισόδου, χρησιμοποιώντας αριθμούς των 2048 bits.

Τα πρώτα γραφήματα που παρουσιάζονται εδώ αφορούν σε μετρήσεις για την εκτίμηση των σειριακών τμημάτων των αλγορίθμου, ενώ αμέσως μετά θα φανούν τα αποτελέσματα των μετρήσεων της πραγματικής εκτέλεσης.

Εκτίμηση παραμέτρων για κάθε συνιστώσα χρόνου εκτέλεσης

Χρησιμοποιώντας τις συναρτήσεις που ορίσαμε πριν, θεωρούμε ένα μέσο όρο των $5.6msec$ ανά εκθετικό και περίπου $34\mu sec$ για τη μεταφορά ενός αριθμού από και προς μια διεργασία. Μια 'απλή' δύναμη, όπως ένα τετράγωνο ή ένας κύβος, θεωρούμε ότι κάνει 10^{-5} και $2 \cdot 10^{-5}$ δευτερόλεπτα αντίστοιχα. Στην παρούσα υλοποίηση, χρησιμοποιήσαμε *ουρές* (Queues) για την επικοινωνία μεταξύ διεργασιών και όλες οι τιμές που αναφέρονται εδώ μετρήθηκαν με ένα στοιχειώδες benchmarking, χρησιμοποιώντας κάποια (Python) scripts στο μηχάνημά μας.

Για τον υπολογισμό των εκθετικών, ένα script δημιουργεί δύο αριθμούς των 2048 bits και μετράται ο χρόνος του αντίστοιχου εκθετικού, όπου ο ένας λειτουργεί ως βάση και ο άλλος ως εκθέτης. Αυτό επαναλαμβάνεται NUM φορές (πχ 1000) και στο τέλος υπολογίζουμε ένα μέσο από αυτούς τους χρόνους.

Για το χρόνο μεταφοράς αριθμών, ένα δεύτερο script δημιουργεί μια ουρά και μια διεργασία-παιδί, για να υπάρχει επικοινωνία. Για $n = 1, \ldots, L$ η διεργασία-γονιός γράφει/στέλνει n αριθμούς των 2048 bits στην ουρά και μετράει το χρόνο μέχρι η διεργασία-παιδί να τους διαβάσει. Αυτό επαναλαμβάνεται NUM φορές για να έχουμε μια μέση τιμή για κάθε n. Εξετάζουμε τις περιπτώσεις για $1, \ldots, L$ αριθμούς γιατί ενδέχεται να έχουμε κάποιο σταθερό overhead όταν γράφουμε σε μία ουρά (παρατηρήθηκε κατά τη διάρκεια των μετρήσεων). Γι΄ αυτό, θεωρούμε το χρόνο που μετρήσαμε για $n = 1$ αριθμό σαν βάση και αφότου έχουμε μετρήσει τους χρόνους για $n = 2, \ldots, L$ αριθμούς, διαιρούμε τον καθένα με το $n - 1$ για να πάρουμε μια εκτίμηση για τη μεταφορά ενός αριθμού σε μια ουρά. Η μέση τιμή αυτών, θα μας δώσει τον τελικό μεταφοράς ενός αριθμού μέσω μιας ουράς.

Επίσης, ένα τελευταίο script χρησιμοποιήθηκε για να μετρήσει τα σειριακά τμήματα που αναφέραμε προηγουμένως, πχ η παραγωγή μιας τυχαίας μετάθεσης, η αντίστροφή της, η δημιουργία τυχαίων αριθμών, καθώς επίσης και οποιαδήποτε άλλη διαδικασία επιδρά στον (παράλληλο) χρόνο εκτέλεσης. Ακριβέστερα, κάθε διαδικασία 'τρέχει' NUM φορές για φορ $L = 1000, \ldots, 10000$, στην περίπτωσή μας, κρυπτοκείμενα και λαμβάνουμε ένα μέσο όρο για κάθε τιμή. Τα επόμενα διαγράμματα φανερώνουν αυτές ακριβώς τις μετρήσεις και επιβεβαιώνουν την γραμμική σχέση που υπάρχει ανάμεσα στους αντίστοιχους χρόνους και το μήκος της εισόδου:

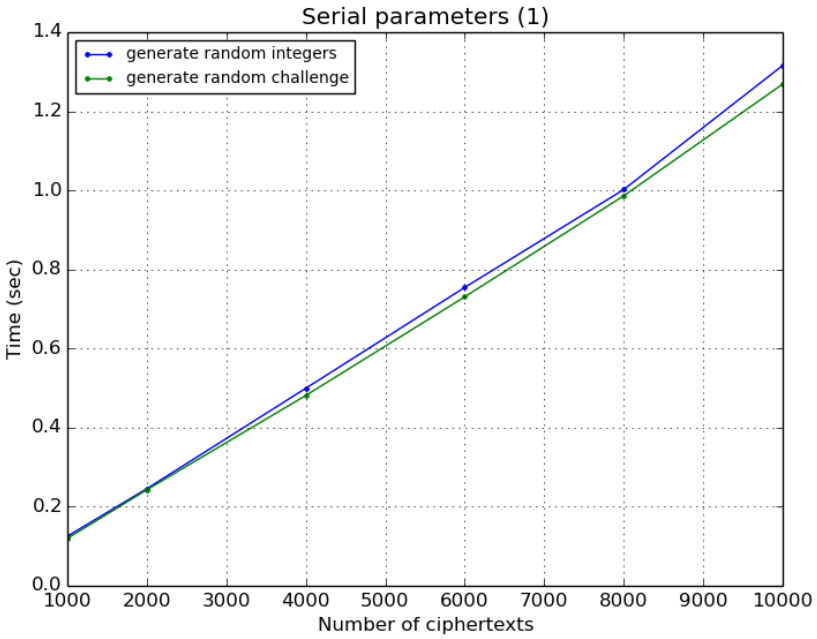

Figure 4.5: Time elapsed for generating random integers and challenges

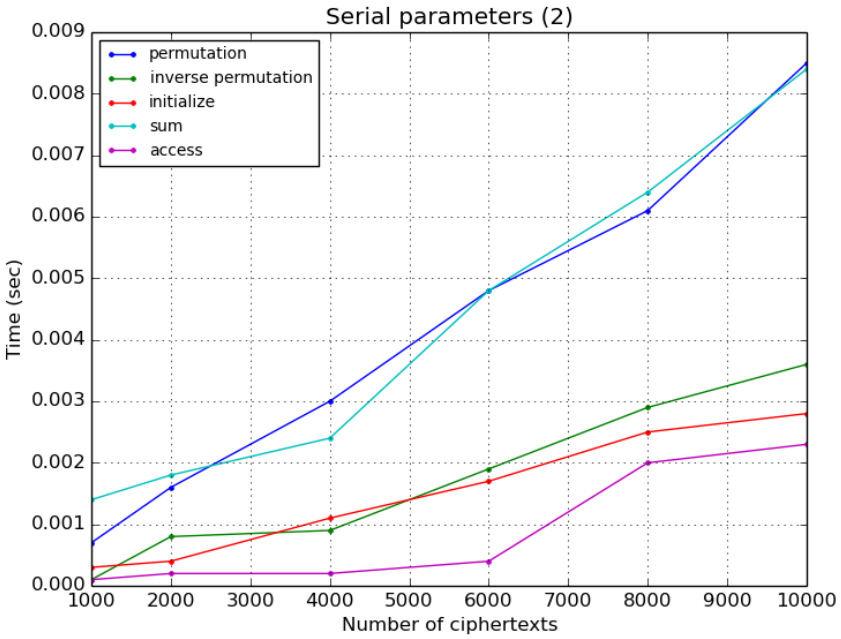

Figure 4.6: Time elapsed for the rest parameters in the serial parts

Οι διακυμάνσεις που παρατηρούνται στα παραπάνω διαγράμματα οφείλονται στην ακρίβεια που χρησιμοποιείται (6 δεκαδικά ψηφία) για το χρόνο, όπου μόνο μικρές διαφορές μπορούν να ανιχνευθούν μεταξύ των λειτουργιών αυτών, ειδικά σε μικρή είσοδο. Στο τέλος, μια γραμμική εφαρμόζεται μια γραμμική παρεμβολή και βγαίνει ο (γραμμικός) παράγοντας για όλες τις λειτουργίες που μας ενδιαφέρουν. Ο επόμενος πίνακας δείχνει το αποτέλεσμα αυτού:

Operation	**Linear factor** $(\frac{seconds}{ciphertext})$
permutation	$0.8365 \cdot 10^{-6}$
inverse permutation	$0.3814 \cdot 10^{-6}$
initialize list	$0.2997 \cdot 10^{-6}$
access list	$0.2603 \cdot 10^{-6}$
sum	$0.7956 \cdot 10^{-6}$
generate ints	$0.1306 \cdot 10^{-3}$
generate challenge	$0.1267 \cdot 10^{-3}$

Table 4.1: Linear factors for serial parameters

Τελικά, φτάνουμε να κατασκευάσουμε ένα γενικό τύπο για τα σειριακά τμήματα της μίξης και της επαλήθευσης, αντίστοιχα:

$$\text{SHUFFLE\_SERIAL\_TIME}(N) = (permutation + 22 \cdot access + 17 \cdot init + challenge) \cdot N + 5 \cdot generate\_int + e \Rightarrow$$

$$\boxed{\text{SHUFFLE\_SERIAL\_TIME}(N) = 0.00014 \cdot N + 0.0056 \text{ sec }, N > 0, P > 1}$$

$$\text{VERIFY\_SERIAL\_TIME}(N) = (challenge + 12 \cdot access + 12 \cdot init) \cdot N \Rightarrow$$

$$\boxed{\text{VERIFY\_SERIAL\_TIME}(N) = 0.000133 \cdot N \text{ sec}}$$

Αποτελέσματα

Αρχικά, τα πρώτα διαγράμματα των μετρήσεών μας είναι της *πραγματικής επιτάχυνσης* της που πετυχαίνουμε με την παράλληλη μίξη/επαλήθευση, για 8 πυρήνες. Εκτός από την επιτάχυνση, μια επιπλέον μετρική επίδοσης αναγράφεται, τα κρυπτοκείμενα που επεξεργαζόμαστε ανά δευτερόλεπτο. Προφανώς, όσο μεγαλύτερη αυτή η τιμή, τόσο καλύτερη επίδοση έχουμε:

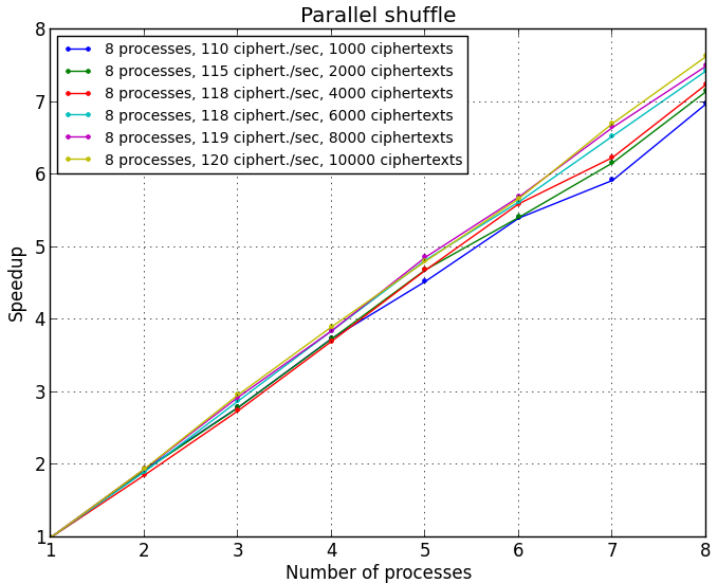

Figure 4.7: Speedup for parallel shuffle

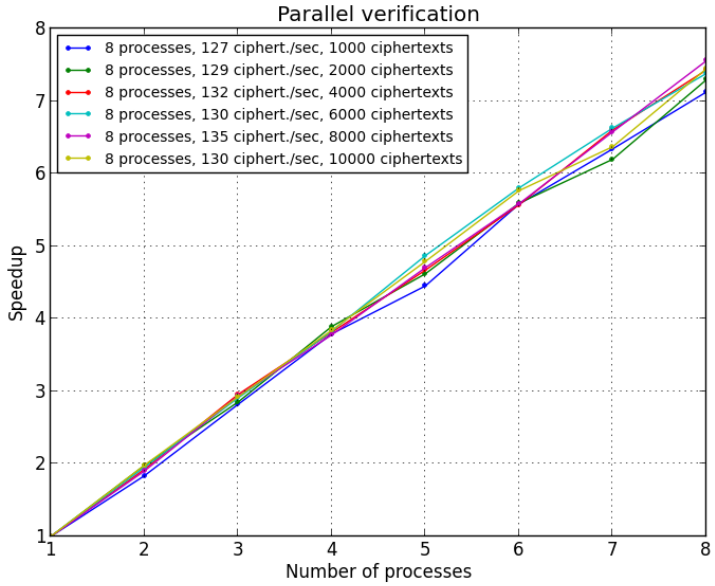

Figure 4.8: Speedup for parallel verification

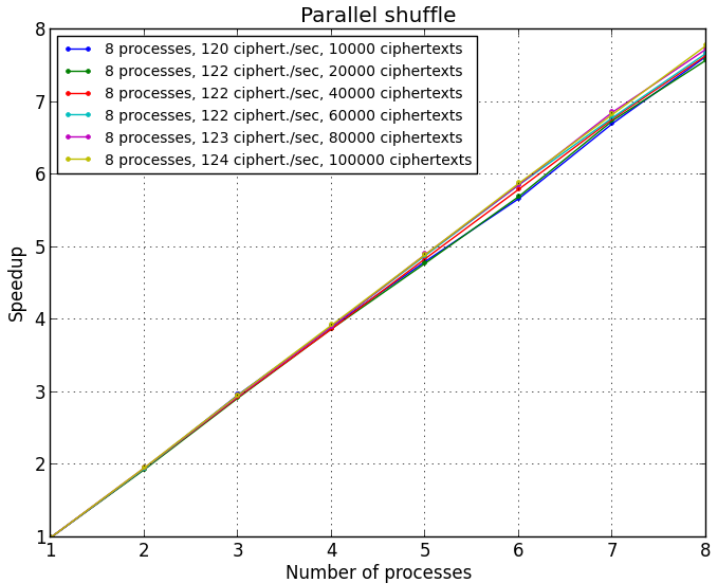

Figure 4.9: Speedup for parallel shuffle - larger input

Για πληρότητα, εμφανίζουμε τα αποτελέσματα που προκύπτουν για είσοδο μέχρι και 100000 κρυπτοικείμενα. Σημειώνεται ότι στις αναλύσεις, έχουμε αναφορά τα 10000 κρυπτοκείμενα για απλότητα, και δε θα ήταν διαφορετικές αν είχαμε κάποια άλλη αναφορά από τις μετρήσεις μας.

Μπορούμε να δούμε ότι η επιτάχυνση για τους δύο παράλληλους αλγόριθμους φτάνει σχεδόν την ιδανική της τιμή, $S_{ideal} = P$. Επομένως, η κλιμακωσιμότητα είναι αρκετά 'καλή', καθώς με την προσθήκη 8 πυρήνων πετυχαίνουμε μια επιτάχυνση κοντά στο 8. Ο νόμος του Amdahl υποδεικνύει όμως ότι τα σειριακά τμήματα και το κόστος επικοινωνίας θέτουν ένα (άνω) φράγμα στη συνεχή αύξηση της επιτάχυνσης. Εδώ, αυτές οι συνιστώσες δεν έχουν σημαντική επίδραση στο χρόνο εκτέλεσης, ακόμα, γιατί ο αριθμός των επεξεργαστών που διαθέτουμε είναι σχετικά μικρός. Επίσης, ο μεγάλος βαθμός παραλληλίας που προσφέρει το σχήμα των Furukawa-Sako συμβάλλει στο παραπάνω αποτέλεσμα, καθώς οι πιο ακριβοί υπολογισμοί (εκθετικά) μπορούν να τρέξουν απομονωμένα στους διαθέσιμους επεξεργαστές, περιλαμβάνοντας μόνο ένα κόστος επικοινωνίας με ένα master process. Στην πραγματικότητα, ακόμα και μέχρι τα 100000 είσοδο, η υλοποίησή μας πετυχαίνει μια επιτάχυνση μεγαλύτερη του 7 για 8 πυρήνες. Παρακάτω όμως θα δούμε ότι οι καμπύλες ξεφεύγουν από την ιδανική καμπύλη, όσο αυξάνουμε τον αριθμό των επεξεργαστών.

Αξίζει να αναφερθεί βέβαια και η χωρική πολυπλοκότητα του προγράμματός μας: για 10000 κρυπτοκείμενα, παράγει περίπου 73MB σε μορφή text στο δίσκο, ενώ για 100000 γύρω στα 740MB. Πιο γενικά, για αριθμούς των 2048 bit, έχουμε περίπου 617 δεκαδικά ψηφία ανά αριθμό. Επομένως, η απόδειξη μίξης μας μπορεί να μεταφραστεί, σε όρους χρήσης δίσκου, ως:

$$U(N) = (12N + 14) \cdot 617 \text{ bytes.}$$

όπου N είναι ο αριθμός κρυπτοκειμένων εισόδου. ΄Οπως φαίνεται, υπάρχει γραμμική σχέση με την είσοδο.

Στα επόμενα σχήματα γίνεται καθαρότερη η σχέση μεταξύ του συνολικού χρόνου εκτέλεσης και των συνιστωσών του, δείχνοντας την καθεμία ξεχωριστά, δηλαδή το παράλληλο τμήμα, το σειριακό και το κομμάτι της επικοινωνίας για μεταβλητό αριθμό κρυπτοκειμένων.

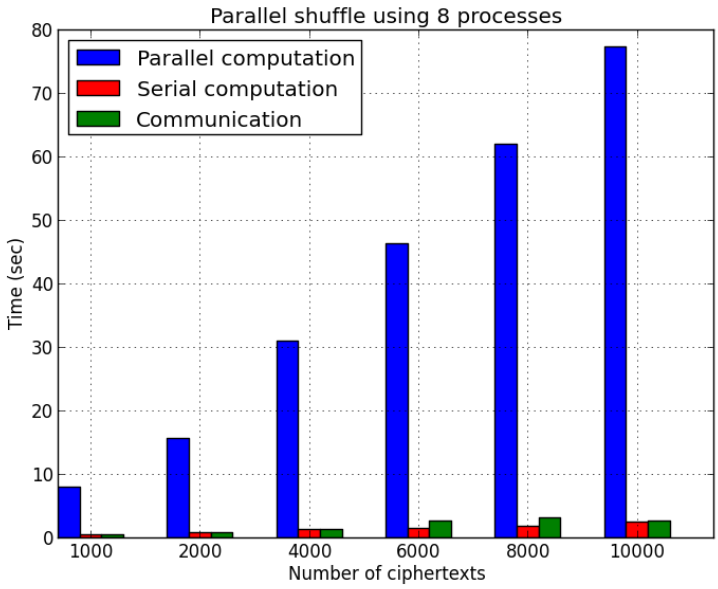

Figure 4.10: Parallel, serial and communication time for shuffle

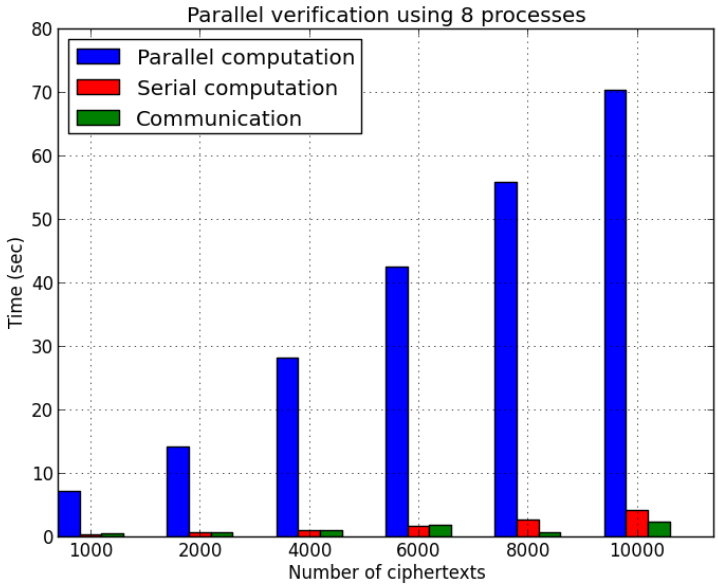

Figure 4.11: Paralllel, serial and communication time for verification

΄Οπως αναμένουμε, ο υπολογισμός των εκθετικών κυριαρχεί στο συνολικό χρόνο. Τα σειριακά τμήματα αυξάνονται καθώς αυξάνεται ο αριθμός των κρυπτοκειμένων, ενώ το κομμάτι της επικοινωνίας ναι μεν αυξάνει, αλλά έχει κάποιες διακυμάνσεις, που πιθανόν να οφείλονται από το περιβάλλον εκτέλεσης του προγράμματος. Μερικές παράμετροι που μπορεί να επηρεάζουν το χρόνο είναι ο συγχρονισμός σε κάποιο μοιραζόμενο αντικείμενο επικοινωνίας (πχ μια ουρά) και άλλες διαδικασίες διαχείρισης νημάτων/διεργασιών (πχ χρονοδρομολόγηση). Αυτοί οι χρόνοι περιλαμβάνονται ήδη στον χρόνο επικοινωνίας του παράλληλου πρωτοκόλλου, καθώς υπολογίζεται ως η διαφορά μεταξύ του συνολικού χρόνου εκτέλεσης και του αθροίσματος του παράλληλου με το σειριακό κομμάτι.

Το επόμενο σημείο προσοχής, και το πιο σημαντικό, είναι τα διαγράμματα που παρουσιάζουν τη σύγκριση μεταξύ της προβλεπόμενης και της πραγματικής τιμής της επιτάχυνσης. ΄Οπως έχει αναφερθεί από την αρχή της εργασίας, ο λόγος που χρειαζόμαστε να προβλέπουμε τη συμπεριφορά του πρωτοκόλλου είναι ότι αναζητούμε τη μέγιστη τιμή επεξεργαστών, δεδομένης της παραλληλοποίησής του, για να κάνουμε παράλληλη μίξη. Εδώ, υποθέτουμε μια είσοδο 10000 (σημειώνεται ότι μέχρι και τα 100000 που δοκιμάσαμε το πρωτόκολλο πειραματικά, τα διαγράμματα έχουν την ίδια μορφή):

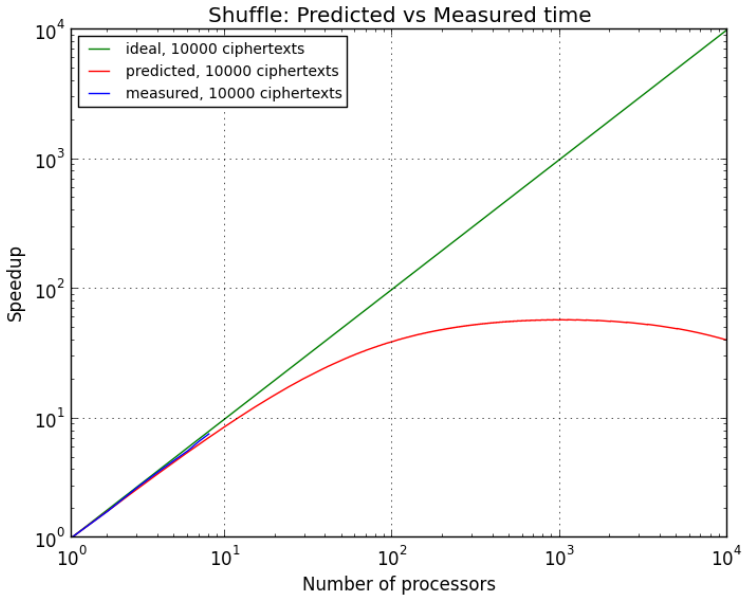

Figure 4.12: Shuffle: theoretical, measured and ideal speedup comparison

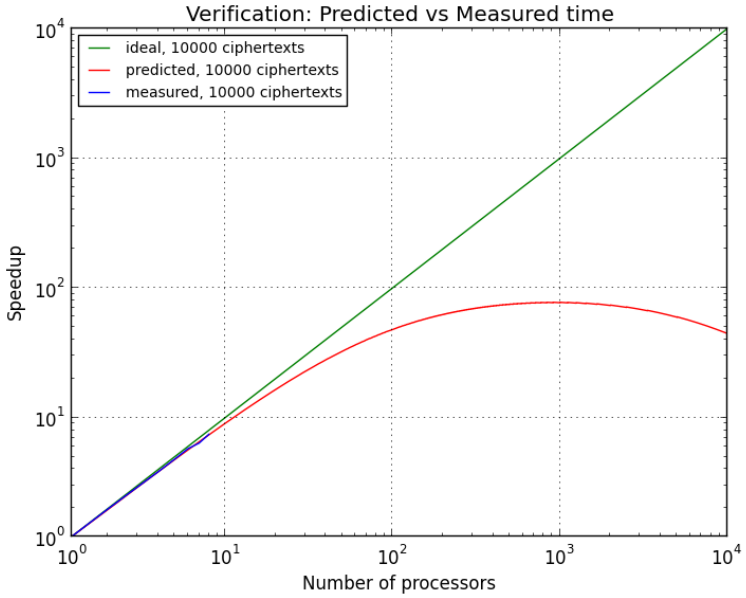

Figure 4.13: Verification: theoretical, measured and ideal speedup comparison

΄Οπως περιμέναμε, οι θεωρητικές καμπύλες φτάνουν μια μέγιστη τιμή, γύρω στους $P = 1000$ επεξεργαστές. Η προβλεπόμενη και η πραγματική επιτάχυνση βρίσκονται πάρα πολύ κοντά και για τις δύο φάσεις και αυτό οφείλεται κυρίως στο μικρό αριθμό των διαθέσιμων επεξεργαστών. Από τον προβλεπόμενο (θεωρητικό) χρόνο εκτέλεσης βλέπουμε ότι το κόστος επικοινωνίας επιδρά σημαντικά στην επίδοση καθώς προσθέτουμε επεξεργαστές. Αργότερα, θα δούμε ότι πώς επηρεάζουν τα σειριακά τμήματα τη συμπεριφορά επίσης. ΄Οπως αναφέρθηκε πριν, μέχρι το σημείο όπου οι ρυθμοί μεταβολής της παράλληλης συνιστώσας και της συνιστώσας επικοινωνίας γίνονται ίσοι και αντίθετοι, η επιτάχυνση συνεχίζει να αυξάνει (με φθίνοντα ρυθμό) και πέρα από αυτό το σημείο, η επικοινωνία κυριαρχεί του παράλληλου χρόνου. Ως αποτέλεσμα, η επιτάχυνση αρχίζει να μειώνεται. Αυτή η ιδιότητα γίνεται καλύτερα αντιληπτή στα επόμενα σχήματα, όπου η καμπύλες του κάθε μέρους φαίνονται ξεχωριστά:

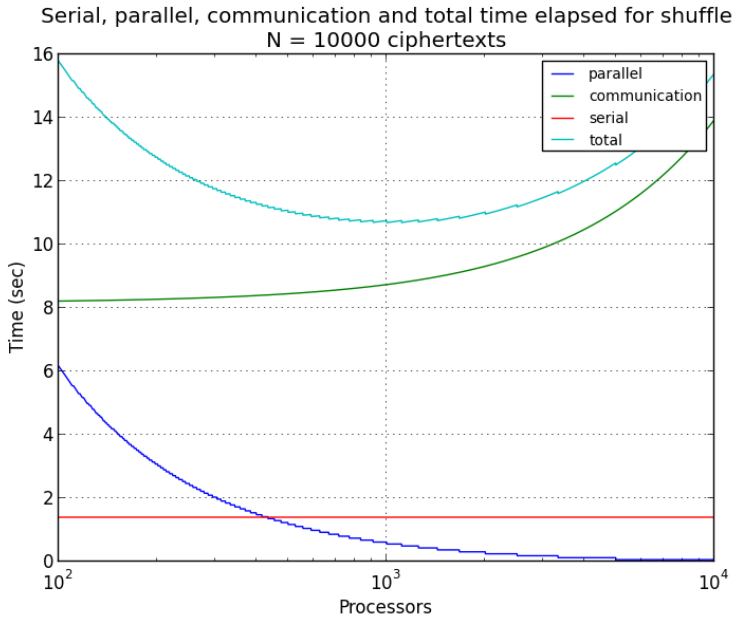

Figure 4.14: Shuffle: serial, parallel and communication times

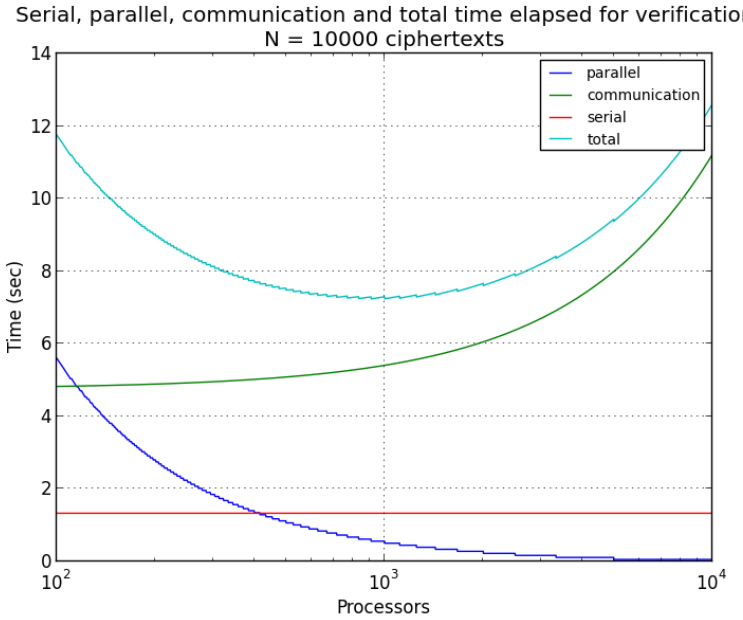

Figure 4.15: Verification: serial, parallel and communication times

Προφανώς, το σειριακό κομμάτι είναι σταθερό, μιας και έχουμε την περίπτωση για 10000 κρυπτοκείμενα και οι άλλες δύο συνιστώσες συνάδουν με την προηγούμενη εξήγηση για το μέγιστο της επιτάχυνσης.

Στη συνέχεια, είναι πολύ σημαντικό να παρουσιάσουμε τον ακριβή τύπο που χρησιμοποιήσαμε για την πρόβλεψη του χρόνου εκτέλεσης. Τα προηγούμενα διαγράμματα (πρόβλεψη επιτάχυνσης) είναι ο κύριος οδηγός μας ώστε να εκθέσουμε τα όρια του προβλήματος και, ετσι, να αποφασίσουμε για ένα πρακτικό αριθμό επεξεργαστών που χρειαζόμαστε για τη μίξη ενός αριθμού ψήφων (κρυπτοκειμένων). Επειδή μας ενδιαφέρει η συμπεριφορά του προγράμματος σε μεγάλες εισόδους (τις οποίες δεν δοκιμάζουμε), χρησιμοποιούμε ένα άνω όριο στις σχέσεις που ορίζουν τα κλάσματα του συνολικού χρόνου εκτέλεσης, οι οποίες αποδεικνύονται πιο χρήσιμες στην κατανόηση του προβλήματος. Αυτό επίσης μεταφράζεται σε μία συνθήκη όπου $N >> P$, αλλιώς δε θα είχε νόημα να παραλληλοποιήσουμε τον αλγόριθμο. ΄Αρα, έχουμε κάποιες *προσεγγιστικές* σχέσεις που περιγράφουν τους χρόνους και, κατ΄ επέκταση, την επιτάχυνση της υλοποίησης:

$$\text{time(shuffle)} \approx 11\frac{N}{P} \cdot E + N \cdot (18 \cdot D + a), N >> P.$$

$$\text{time(verification)} = 10\frac{N}{P} \cdot E + N \cdot (14 \cdot D + b), N >> P.$$

Πλέον γίνεται σαφές ότι το κόστος επικοινωνίας και τα σειριακά τμήματα του σχήματος θα παίξουν καθοριστικό ρόλο στην επιτάχυνση, τα οποία θα είναι και οι παράγοντες που την εμποδίζουν να κλιμακώσει για μεγάλο αριθμό επεξεργαστών.

Οι αντίστοιχες εκφράσεις για τη *σειριακή* εκδοχή του πρωτοκόλλου είναι:

$$\text{serial(shuffle)} = 11NE + 9 + a'N \approx (11 + a') \cdot N, N \text{sufficiently large.}$$

$$\text{serial(shuffle)} = 10NE + 9 + b'N \approx (10 + b') \cdot N, N \text{sufficiently large.}$$

΄Οπως βλέπουμε, υπάρχει γραμμικότητα μεταξύ των κρυπτοκειμένων εισόδου και τον συνολικό (γνήσια σειριακό) χρόνο. Μερικές τιμές για τα a, a', b, b' που υπολογίστηκαν είναι:

	a	a'	b	b'
Shuffle	0.00014	0.0004	-	-
Verification	-	-	0.000127	0.000133

Table 4.2: Linear factors for the parallel and serial versions of the protocol

΄Επειτα, η επιτάχυνση παίρνει την εξής μορφή:

$$S_s = \frac{\text{serial(shuffle)}_N}{\text{time(shuffle)}_{N,P}} = \frac{(11 + a')N}{11\frac{N}{P} \cdot E + (18D + a) \cdot N}, N >> P.$$

$$S_s = \frac{\text{serial(verification)}_N}{\text{time(verification)}_{N,P}} = \frac{(10 + b')N}{10\frac{N}{P} \cdot E + (14D + b) \cdot N}, N >> P.$$

Τώρα, για να μπορούμε να διαλέγουμε ένα κατάλληλο αριθμό επεξεργαστών για την παράλληλη εκτέλεση του σχήματός μας, πρέπει να κοιτάξουμε καλύτερα τα τις καμπύλες της επιτάχυνσης, ιδιαίτερα της θεωρητικής, εφόσον έχουν οριστεί όλες οι σχέσεις που αφορούν σε αυτές. Η επιλογή δε στηρίζεται μόνο σε απλά μαθηματικά, αλλά υπάρχουν πολλές πτυχές που πρέπει να λάβουμε υπόψη (engineering criteria). Για παράδειγμα, αναφέρθηκε ότι η καμπύλες φτάνουν ένα μέγιστο για περίπου 1000 επεξεργαστές. Θα μπορούσε να πει κάποιος ότι αυτός είναι ο αριθμός που ψάχνουμε. ΄Ομως, ένας παράγοντας που θα μπορούσε να επηρεάσει την επιλογή μας είναι το ***bandwidth*** που χρειάζονται τόσα μηχανήματα για ένα δίκτυο επικοινωνίας. Ανάλογα λοιπόν με διάφορα κριτήρια, είτε από τη σκοπιά της διαθεσιμότητας πόρων είτε από οικονομικής πλευράς (που εντέλει εδώ καταλήγουμε),

μπορεί να είμαστε πιο ευέλικτοι ή πιο περιορισμένοι ως προς τον καλύτερο αριθμό που μας εξυπηρετεί. Με απλά λόγια, είμαστε τόσο ευχαριστημένοι όσο μας επιτρέπει το κόστος που είμαστε διατεθειμένοι να καλύψουμε για να πετύχουμε αντίστοιχη επιτάχυνση.

Επομένως, θα πρέπει να ορίσουμε και τα tradeoffs που υπάρχουν και να τα κρίνουμε με ένα μοντέλο που ταιριάζει στο πρόβλημά μας. Από τα διαγράμματα της πρόβλεψης της επιτάχυνσης, μπορούμε να δούμε ότι σταδιακά πρέπει να αυξάνουμε ολοένα και περισσότερο τον αριθμό τον επεξεργαστών για να έχουμε σταθερή αύξηση της επιτάχυνσης (πχ κατά μία μονάδα). Ιδανικά θα θέλαμε η αύξηση των επεξεργαστών κατά 1 να οδηγεί σε ίδια αύξηση της επιτάχυνσης. Αυτό δε μπορεί όμως να συμβαίνει, όπως ορίζει ο νόμος του Amdahl, μιας και δε γίνεται να έχουμε 100% παράλληλο πρόγραμμα και εφόσον το κόστος επικοινωνίας αυξάνεται καθώς αυξάνονται οι επεξεργαστές.

΄Αρα, έχουμε να κάνουμε με ένα κύριο tradeoff, βάσει του οποίου πρέπει να δώσουμε απάντηση για τη μέγιστη παράλληλη μίξη: το κόστος επεξεργαστών ανά μονάδα επιτάχυνσης με την επιτάχυνση. Διαισθητικά, αν κοιτάξουμε τα διαγράμματα, βλέπουμε πως καθώς προσθέτουμε επεξεργαστές η καμπύλη ανεβαίνει 'γραμμικά' μέχρις ότου δημιουργήσει μία καμπή ('σπάσιμο'). Είναι εύλογο λοιπόν να μας ενδιαφέρει εκείνη η συγκεκριμένη περιοχή αν θέλουμε ένα ελάχιστο κόστος ανά μονάδα επιτάχυνσης. Το σημείο αυτό θα δούμε ότι θα το προσδιορίζουμε κατευθείαν στην επόμενη ενότητα. Συνεπώς, για μια μίξη 10000 κρυπτοκειμένων το 'σπάσιμο' της καμπύλης φαίνεται να προκύπτει γύρω στους 30 επεξεργαστές και για την επαλήθευση γύρω στους 40. Επομένως θα πρέπει να διαλέξουμε τιμές κοντά σε αυτές για να έχουμε την καλύτερη επιτάχυνση, με την υπόθεση ότι θέλουμε ένα κόστος το πολύ 2 επεξεργαστές ανά μονάδα.

Επίσης, αν το bandwidth αποτελεί κριτήριο, τότε ισχύουν τα ακόλουθα:

$$Bandwidth = 2P \cdot \frac{1}{D} \frac{numbers}{sec} = 2P \cdot \frac{B}{D} \frac{bits}{sec}$$

όπου θα πρέπει να δούμε πότε $Bandwidt < K$, για μια ζητούμενη τιμή του, K.

Για τα δεδομένα μας, το ζητούμενο bandwidth προκύπτει περίπου $\approx 3.8 \frac{Gbit}{sec}$, για $P = 32$ επεξεργαστές.

Προτείνοντας το μέγιστο αριθμό επεξεργαστών

Σε αυτό το σημείο, μπορούμε να κατασκευάσουμε ένα μοντέλο το οποίο να μας επιτρέπει να καθορίζουμε έναν κατάλληλο αριθμό επεξεργαστών για την παράλληλη εκτέλεση αυτού του πρωτοκόλλου.

Πρακτικά, σκοπός αυτού του μοντέλου είναι να απαντήσει στην επόμενη ερώτηση:

'Υποθέτωντας ότι το κόστος ανά μονάδα επιτάχυνσης δε μπορεί να είναι περισσότερο από M, πόσους επεξεργαστές θα πρέπει να διαλέξουμε για να κάνουμε μίξη N κρυπτοκείμενα;'

΄Οπως σκιαφραφήθηκε στο προηγούμενο μέρος, η πρόβλεψη για το παράλληλο, σειριακό και το κομμάτι επικοινωνίας και, κατ΄ ακολουθία, της επιτάχυνσης για ένα δεδομένο πρόβλημα απαιτεί τη γνώση κάποιων παραμέτρων για το περιβάλλον εκτέλεσης. Τις αναφέρουμε συνοπτικά εδώ:

- E sec/exponentiation
- D sec/number (transfer time to and from a processor)
- $A(N) = aN + b$: serial fraction of the parallel program (N ciphertexts)
- $T_s(N)$: (serial) execution time for N ciphertexts
- $T(P)$: parallel time for shuffle/verification accordingly

Με τις αντίστοιχες σχέσεις για την επιτάχυνση:

$$S_{shuf}(N,P) = \frac{T_s}{T_{shuf}(P)} = \frac{T_s}{T_p + T_c + A_{shuf}} \Rightarrow$$

$$\boxed{S_{shuf}(N,P) \approx \frac{(11+a')N}{11\frac{N}{P} \cdot E + (18D + a) \cdot N}, N >> P.}$$

$$S_{ver}(N,P) = \frac{T_s}{T_{ver}(P)} = \frac{T_s}{T_p + T_c + A_{ver}} \Rightarrow$$

$$\boxed{S_{ver}(N,P) \approx \frac{(10+b')N}{11\frac{N}{P} \cdot E + (14D + b) \cdot N}, N >> P.}$$

Τότε, μπορούμε να σκεφτούμε την απάντηση στο προηγούμενο ερώτημα ως εξής:

'Υπάρχουν P', P τέτοια ώστε $S(P') - S(P) = 1$ και $\Delta P = P' - P = M$.'

΄Αρα, η απάντηση αποτελεί τη λύση σε αυτή την εξίσωση:

$$S(P) = \frac{T_s(N)}{T(P)} \Rightarrow$$
$$S(P') - S(P) = 1 \Rightarrow$$
$$T(P) - T(P') - \frac{T(P)T(P')}{T_s} = 0 \Rightarrow$$

$$\boxed{T(P) - T(P+M) - \frac{T(P)T(P+M)}{T_s} = 0} \quad (4.1)$$

Αν υπάρχει λύση P_a στην παραπάνω εξίσωση, αντιπροσωπεύει το *μέγιστο* αριθμό επεξεργαστών P, οι οποίοι ικανοποιούν τον περιορισμό των M επεξεργαστών ανά μονάδα επιτάχυνσης. Το γεγονός ότι αναπαριστά το μέγιστο αριθμό, απορρέει από τα γραφήματα της προηγούμενης ενότητας. Το κόστος ανά μονάδα επιτάχυνσης αυξάνεται όσο προσθέτουμε επεξεργαστές, ξεκινώντας από μια τιμή μεγαλύτερη του 1. Επομένως, ανάλογα με το τι M διαλέγουμε, μπορούμε να βρούμε πού η αύξηση της επιτάχυνσης γίνεται ακριβώς ίση με 1 και να διαλέξουμε έναν αριθμό επεξεργαστών μεταξύ $2 \leq P \leq \text{FLOOR}(P_a) + M$, βασιζόμενοι ίσως σε κριτήρια για το bandwidth, αν υπάρχουν. Η FLOOR() διαδικασία χρησιμοποιείται γιατί η ρίζα της εξίσωσης 4.1 μπορεί να έχει πραγματική τιμή και οποιαδήποτε τιμή πάνω από αυτή απαιτεί περισσότερους από M επεξεργαστές, κάτι το οποίο δεν είναι αποδεκτό.

Για παράδειγμα, ας θεωρήσουμε έναν αριθμό 10000 κρυπτοκειμένων, όπως φαίνεται στα επόμενα διαγράμματα. Αν απαιτήσουμε ότι το μέγιστο κόστος επεξεργαστών ανά μονάδα επιτάχυνσης να είναι 2, τότε η 4.1 αποδίδει 35 επεξεργαστές για να κάνουμε τη μίξη και 38 για την επαλήθευση. Αυτό σημαίνει ότι μέχρι 37 επεξεργαστές (αντίστοιχα 40 για την επαλήθευση), το κόστος ανά μονάδα επιτάχυνσης για τη μίξη είναι 2 το πολύ. Το αποτέλεσμα αυτό επιβεβαιώνει τη διαίσθησή μας σε προηγούμενη εκτίμηση, όπου κατευθείαν από τα διαγράμματα υποθέσαμε περίπου 30-32 επεξεργαστές για τη μίξη ('γραμμική περιοχή'), με την απαίτηση για ελάχιστο κόστος ανά μονάδα επιτάχυνσης. Σημειώνεται ότι η εξίσωση δεν έχει λύσει αν ζητήσουμε κόστος 1 επεξεργαστή ανά μονάδα.

΄Οσο για το bandwidth, αν υπάρχει απαίτηση να μην υπερβαίνει τα K bits/sec, τότε ο μέγιστος αριθμός επεξεργαστών είναι:

$$R = 2 \cdot P\frac{B}{D} \leq K \Rightarrow P \leq \frac{K \cdot D}{2B}$$

Εφαρμόζοντας το μοντέλο αυτό για διάφορες τιμές της εισόδου και του κόστους ανά μονάδα επιτάχυνσης, μπορούμε δούμε τι συμπεριφορά έχει τελικά το παράλληλο πρόγραμμά μας όταν $N >> P$, καθώς

επίσης και τα όρια που φτάνει τόσο στην ίδια την επιτάχυνση όσο και στην κλιμακωσιμότητα. Οι παράμετροι του συστήματός μας είναι οι εξής:

- $D = 0.000034 \frac{sec}{number}$
- $E = 0.0056$ sec
- $A_{shuf}(N) = 0.00014N + 0.0056$ sec
- $A_{ver}(N) = 0.000133N$ sec

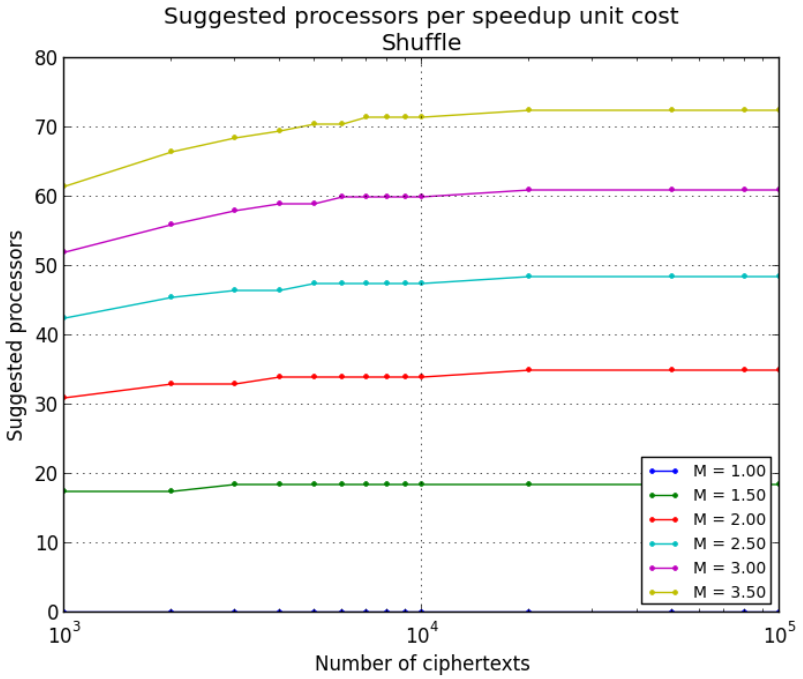

Figure 4.16: Shuffle: suggested number of processors per speedup unit cost

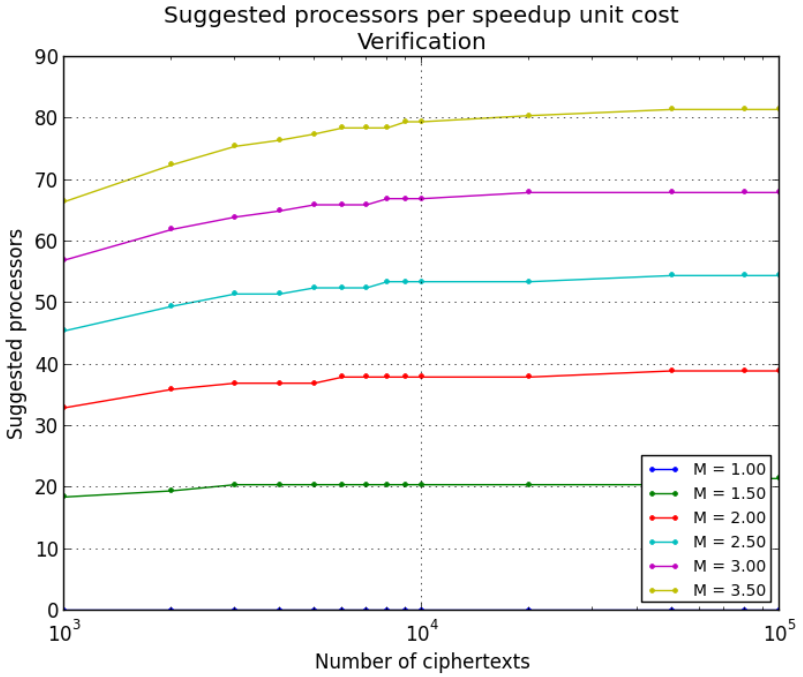

Figure 4.17: Verification: suggested number of processors per speedup unit cost

Εδώ υποθέτουμε ότι δεν υπάρχουν περιορισμοί σε bandwidth και έτσι προτείνεται ο μέγιστος αριθμός επεξεργαστών που ικανοποιεί την 4.1 για κάθε περίπτωση.

Τελικά, χρησιμοποιώντας το μοντέλο που αναφέρθηκε, μαζί με τις προβλέψεις που έγιναν για το χρόνο εκτέλεσης, το σύστημά μας εικάζεται ότι μπορεί να κάνει μίξη 1000000 ψήφων με 32 πυρήνες σε περίπου 45 λεπτά, δημιουργώντας μια απόδειξη γύρω στα 7.5GB δεδομένα στο δίσκο ως απόδειξη.

Προτείνοντας επεξεργαστές για σενάρια κατανεμημένης μίξης

Εφόσον έχουμε τα πειραματικά δεδομένα για το πρόγραμμά μας, τα οποία επιβεβαιώνουν τις θεωρητικές προβλέψεις, μπορούμε να εφαρμόσουμε το μοντέλο μας για σενάρια όπου πλέον υπάρχει ένα δίκτυο με κατανεμημένους κόμβους. Μέχρι στιγμής, οι μετρήσεις μας αφορούσαν σε *single node*, δηλαδή ένα μόνο κόμβο όπου οι πυρήνες επικοινωνούν μέσω μνήμης με χρήση κάποιας ουράς (Queue). Αυτό μας δίνει μια τιμή ρυθμού μεταφοράς γύρω στα 7.8 MB/sec, το οποίο είναι αρκετά αργό. Θεωρούμε λοιπόν δύο περιπτώσεις που, αντί για ουρά, οι επεξεργαστές στο δίκτυο επικοινωνούν με γραμμές των

1GB/sec και 10GB/sec. ΄Ετσι, παρακάτω παρουσιάζεται ένα γράφημα με τον αριθμό επεξεργαστών που προτείνει το μοντέλο μας, μαζί με τις αντίστοιχες επιταχύνσεις για κάθε περίπτωση. Επίσης, θεωρείται ένας αριθμός 1000000 κρυπτοκειμένων ως είσοδος και τρεις τιμές για κόστος ανά μονάδα επιτάχυνσης ($M = 1.2, 1.5, 2$).

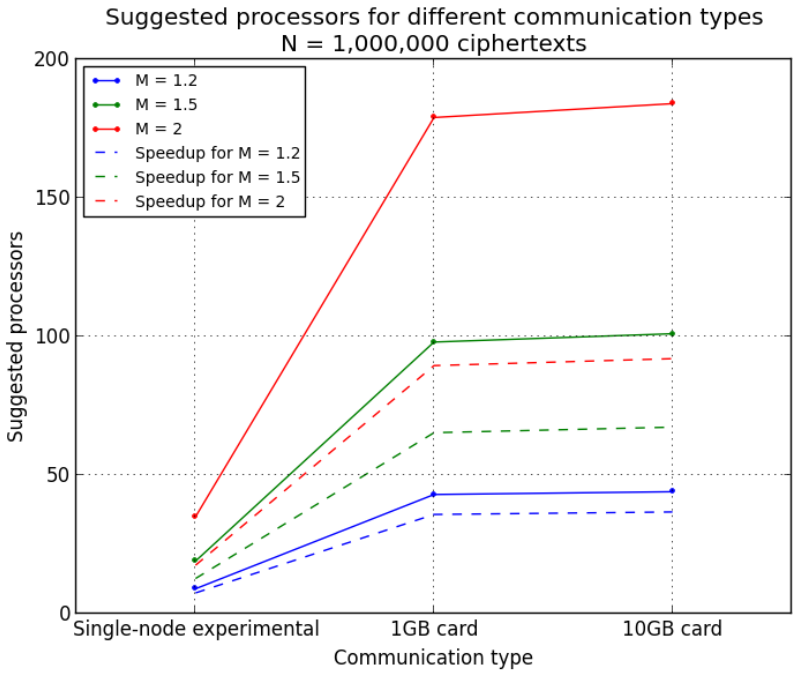

Figure 4.18: Model predictions for two cases of distributed mixing

Παρατηρούμε ότι φεύγοντας από την περίπτωση του ενός κόμβου, ο αριθμός των προτεινόμενων επεξεργαστών αυξάνεται σημαντικά (πχ από 35 σε 179 για $M = 2$) για γραμμή 1GB/sec. Αν δεκαπλασιάσουμε όμως το ρυθμό μεταφοράς (το δεύτερο σενάριο), τοτε ο προτεινόμενος αριθμός έχει ελάχιστη αυξηση (από 179 σε 185 για $M = 2$). Ο λόγος είναι ότι φτάνουμε στο όριο που θέτει το σειριακό κομμάτι του αλγορίθμου (επιβεβαίωση νόμου του Amdahl) και δεν μπορούμε να πετύχουμε παραπάνω επιτάχυνση.

Επιπροσθετα, με τη χρήση του μοντέλου αυτού, μπορούμε να προτείνουμε ένα συνδυασμό παραμέτρων για την παράλληλη μίξη σε ένα κατανεμημένο δίκτυο, πέρα από την πρόβλεψη της επίδοσής της. Για παράδειγμα, αν θεωρήσουμε ότι διαθέτουμε 185 επεξεργαστές και 10GB/sec γραμμή (εντέλει δεν υπάρχει διαφορά με 179 επεξεργαστές και γραμμή 1GB/sec), τότε μπορούμε να επεξεργαστούμε έως και 10500000 περίπου κρυπτοκείμενα σε μία ώρα. Αντίθετα, αν πάρουμε την περίπτωση **ενός κόμβου** με 9 πυρήνες και χρήση Ουράς, τότε μπορούμε να

επεξεργαστούμε περίπου 500000 κρυπτοκείμενα σε μία ώρα.

Συμπερασματικά, χρησιμοποιώντας το θεωρητικό μοντέλο για περιπτώσεις όπως τις παραπάνω, μπορούμε να δούμε τελικά ποια είναι τα όρια του προβλήματός μας και έτσι να πετύχουμε ένα καλό trade-off.

Κεφάλαιο 5

Συμπεράσματα

5.1 Σύνοψη

Σε αυτή την εργασία, διεξήχθη μια εκτενής μελέτη των δικτύων μίξης, ώστε να βρεθεί μια εναλλακτική στον τωρινό αλγόριθμο που χρησιμοποιείται από το σύστημα ηλεκτρονικών ψηφοφοριών Ζευς. Το Ζευς έχει έναν πολύ αργό αλγόριθμο για μίξη στη ροή εργασιών του, ο οποίος είναι και η αιτία για μια χρονοβόρα διαδικασία εκλογών. Γι-΄ αυτό, αναλύσαμε και αξιολογήσαμε, τόσο στη θεωρία όσο και στην πράξη, το σχήμα των Furukawa-Sako και εκτιμήσαμε την επίδοσή του για μια πιθανή χρήση από το Ζευς.

Αυτό το σχήμα έχει την προοπτική να βελτιώσει κατα πολύ τη διαδικασία της μίξης και έχει 'βολικές' ιδιότητες (παραλληλία δεδομένων) τις οποίες μπορούμε να εκμεταλλευτούμε για να πετύχουμε όσο περισσότερη επιτάχυνση γίνεται. Επομένως, παραλληλοποιήσαμε σχεδόν ολόκληρο τον αλγόριθμο και κυρίως τον αριθμό των εκθετικών που είναι και ο λόγος για μια χρονοβόρα μίξη. ΄Ομως, δεν αναζητήσαμε την ιδανική παραλληλοποίηση, αφήνοντας έξω από αυτή ελάχιστες λειτουργίες, λόγω της αρχιτεκτονικής του Ζευς. Ο στόχος της εργασίας αυτής είναι να ορίσει τα tradeoffs που υπάρχουν ανάμεσα στην επιτάχυνση της μίξης και την υπολογιστική δύναμη που χρειαζόμαστε για να την πετύχουμε, ώστε να μπορέσουμε να βρούμε έναν πρακτικό αριθμό CPUs για το πρόβλημά μας, μέσω ενός θεωρητικού μοντέλου. Το μοντέλο αυτό τονίζει τα όρια του προβληματός μας και δίνει μια γενική ιδέα για την αναμενόμενη επίδοση του αντίστοιχου παράλληλου προγράμματος.

Το σχήμα των Furukawa-Sako ήταν μια σημαντική προσέγγιση προς την ιδέα της γρήγορης μίξης και όπως είδαμε, μπορούμε να πετύχουμε και μια επιτάχυνση γύρω στο 37x (single-node), το οποίο είναι ένα θετικό αποτέλεσμα. Πρακτικά, με αυτό το σχήμα μπορούμε να κάνουμε μίξη για εκλογές μέχρι και της τάξης των 100000 σε λιγότερο από 15

λεπτά, με μικρές απαιτήσεις σε χώρο.

5.2 Μελλοντικές κατευθύνσεις

Υπάρχουν περισσότερες προσεγγίσεις για την υλοποίηση του παρόντος πρωτοκόλλου, η οποίες εκμεταλλεύονται ελαφρώς διαφορετικά τις εξαρτήσεις και την παραλληλία των δεδομένων. Για παράδειγμα, θα μπορούσαμενα να δοκιμάσουμε να μην μοιράζουμε ομάδες από N εκθετικά, αλλά, αφαιρετικά, να θεωρούμε το χώρο τους σαν ενιαία ακολουθία η οποία θα οδηγεί ίσως σε καλύτερη κατανομή του φορτίου. Αυτό θα μπορούσε να εφαρμοστεί και με οποιονδήποτε αλλον υπολογισμό διανύσματος μήκους N.

Επιπλέον, η μέθοδος της παραλληλοποίησης μπορεί να επεκταθεί. Στην εργασία αυτή, έχουμε να κάνουμε μία SIMD λογική, όπου μοιράζουμε κομμάτια ίδιων διανυσμάτων σε διαφορετικούς επεξεργαστές. Αντίθετα, θα μπορούσαμε να ακολουθήσουμε μια MIMD λογική όπου αντί για παραλληλία υπολογισμών σε ίδια διανύσματα να είχαμε παραλληλία ολόκληρων λειτουργιών (tasks). Αυτό ενδέχεται να αγγίζει και την ιδέα μιας δυναμικής κατανομής φορτίου κατά την εκτέλεση, με τη χρήση κάποιων αντίστοιχων μηχανισμών.

Τέλος, η παρούσα υλοποίηση δε δοκιμάστηκε πάνω σε κατανεμημένο δίκτυο επεξεργαστών. ΄Ομως, μπορούμε να προβλέψουμε τέτοια σενάριο με το μοντέλο που προτάθηκε σε αυτή την εργασία. Είναι ενδιαφέρουσα η δοκιμή σε τέτοιο πλαίσιο και θα οδηγήσει σε περισσότερα συμπεράσματα για την κλιμακωσιμότητα και την επιτάχυνση της μίξης με τον συγκεκριμένο αλγόριθμο.

NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

DIVISION OF COMPUTER SCIENCE
COMPUTING SYSTEMS LABORATORY

Evaluation and parallelization of mixing algorithms for use in E-voting systems

DIPLOMA THESIS

of

Konstantinos I. Mamasoulas

Supervisor: Panayiotis Tsanakas
Professor NTUA

Athens, March 2015

NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING
COMPUTING SYSTEMS LABORATORY

Evaluation and parallelization of mixing algorithms for use in E-voting systems

DIPLOMA THESIS

of

Konstantinos I. Mamasoulas

Supervisor: Panayiotis Tsanakas
Professor NTUA

Approved by the commitee on the 27th of March 2015.

.....	.....	.....
Panayiotis Tsanakas Professor NTUA	Nectarios Koziris Professor NTUA	Georgios Goumas Lecturer NTUA

Athens, March 2015

.....
Konstantinos I. Mamasoulas
Electrical & Computer Engineer

Chapter 1

Introduction

Since the birth of democracy, a lot of voting systems have been designed, allowing populations to express an opinion about critical decisions. So, by *voting*, we mean that an individual can express a number of choices freely, between a publicly known set of candidates. The most common voting scheme nowadays is the traditional paper-based elections. In this system, there are certain days that a voter can come to a *polling place*, pass through an authentication process, receive a ballot and choose his vote in a *voting booth*. Subsequently, he casts his vote in a transparent box, in front of the necessary authority, and he is then registered, which proves that he took part in the elections. When the voting phase is finished, the box containing all the ballots is publicly opened by the authority, the ballots are counted and the results of the elections are announced. In this way, the voter is convinced that anonymity, privacy and verifiability of the votes are satisfied. Unfortunately, there are examples of these properties failing to preserve and frauds have been exposed throughout the years voting is established.

Although the paper-based system is widely used today, one drawback of it is that it requires quite some time to complete. Leaving aside the fact that voters have to actually go to the voting places, the tallying of the votes can take hours, even days, leading to a lengthy publication of the results. This is exactly what *E-voting* aims to improve, instead of hours or days of counting votes, the results should be announced within an hour for example. Ofcourse, there are many security aspects that need to be studied very carefully before the actual use of such schemes and, today, there are several remarkable efforts in this field.

In the next chapters, more about the properties an electronic voting system needs to have are presented.

Overview

Electronic Voting (or e-voting) is the transfer of the conventional paper-based voting, to the digital/electronic world. As the term indicates, the use of an electronic system is essential for both casting and counting votes. Just like any other electronic system, e-voting systems aim at accelerating the voting process by taking advantage of what technology has to offer us today. There are two main types of e-voting: remote e-voting and e-voting that demands the physical attendance of voters in the polling stations, using dedicated electronic devices. We focus on the first one. Simply put, having a remote e-voting system means that a voter can cast his/her vote by using a program running on his/her personal computer. This approach has many advantages, but great drawbacks that need to be dealt with as well.

E-voting options, ofcourse, satisfy the voters' need for convenience. Imagine that nobody will have to be present in a voting place in order to cast his vote and, what is more, this is an effective alternative for some vulnerable social groups, e.g. the physically disabled. Furthermore, it is means less operational expenses. Also, the cost for printing and mailing the ballots are significantly reduced, along with the personnel expenses during the elections.

On the other hand, the design and implementation of such systems is quite a hard task due to the principles imposed by the traditional paper-based voting scheme in conjuction with the underlying dangers of the Internet. Privacy, verifiability, correctness and many more properties must be adapted or even extended in order to promote the use of an electronic voting protocol.

There are several stages that make up an e-voting protocol, but this thesis focuses on a certain one, called ***mixing***, which is explained in more detail in the next chapter. It simply represents the guarantee that votes cannot be compromised, just like the traditional paper-based voting.

Today, many researchers and engineers are working together to produce such systems and there are quite a few notable implementations. In Greece, there already is a complete and running e-voting service called ***"Zeus"***, which was developed by the *Greek Research and Technology Network (GRNET)* and it is based on the Helios e-voting system.

Motivations and purpose

As mentioned before, an e-voting system consists of several components. The key for a successful e-voting service lies in a strong

cryptographic scheme, that constitutes the core of the process, and a strong cryptographic scheme translates to a lot of computational complexity, in terms of computer science. Such process is the aforementioned mixing stage of an e-voting system: it demands a lot of computational resources in order to produce a valid result and the higher the number of voters, the higher the resources needed. Therefore, it is worth studying ways to accelerate this process.

In particular, the purpose of this thesis is to study, implement and compare various algorithms that can be used to perform the *mixing* stage, in the context of the Zeus e-voting system. Zeus uses a very expensive, in terms of computer science, algorithm for the mixing phase and, as a result, it constitutes the slowest stage in its workflow. For that reason, a survey on the available algorithms that exist in the bibliography was carried out, so that we can examine the possibility of replacing the current one. After the survey is completed, an algorithm is chosen to be impemented and its overall performance is evaluated, leading to conclusions related to the our main goal, the acceleration of the mixing phase.

Chapter 2

E-Voting fundamentals

2.1 Principles and properties

In order to understand how we can construct an e-voting system compared to a traditional paper-based election scheme, the building blocks that each must incorporate are presented below. Ofcourse, an electronic voting scheme must achieve at least what paper-based schemes already do [9]:

Secrecy

This is one of the most fundamental properties of an election. No other participant other than the voter should be able to determine the vote cast by that voter.

Correctness

If all the participants of an election are honest and behave as it is scheduled, then the final results are effectively the tally of casted votes. This property guarantees the result of the election is accurate.

Receipt-freeness

Voters must neither be able to obtain nor construct a receipt which can prove the content of their vote. This means that a voter cannot sell his vote or cannot be coerced by other parties, because he cannot provide any evidence of how he/she voted.

Robustness

Resistance to faulty behaviour and coalition of voters: any cheating voter will be detected and voting process cannot be disrupted.

Verifiability

A correct voting process must be verifiable in order to prevent any incorrect result. This can be achieved either **individually**, i.e. every voter can verify that his vote was counted, or **universally**, i.e. any participant or passive observer can convince himself of the validity of individual votes and of the final tally of the election.

Democracy

All the eligible voters in an election can vote only **once**, so that they have equal power in deciding the outcome of the voting process.

Fairness

Any information about the tally result cannot be obtained by the voters before its publication (counting).

2.2 Cryptographic primitives

As we have already stressed in the previous chapter, a secure e-voting system must apply all the cryptographic primitives needed in order to ensure the preservation of the principles that define a voting process, as seen above.

2.2.1 ElGamal cryptosystem

The most common encryption scheme used in e-voting is ElGamal. It constitutes a homomorphic encryption scheme where a cyclic group $\mathbb{G}$ with a generator g, a large prime order q and another large prime p (modulus) are chosen ($p = 2q + 1$).

ElGamal encryption

It is an asymmetric key encryption algorithm for public-key cryptography, which is based on the Diffie-Hellman key exchange [20]. A secret key $x \in \mathbb{Z}_q$ is chosen at random and the public key of the encryption scheme becomes $(p, g, y = g^x \bmod p)$. Therefore, the ciphertext of

a message M (plaintext) is defined as an ElGamal encryption pair as follows:

$$C = (a, b) = (g^r, y^r M)$$

where r is chosen from $\mathbb{Z}$ at random (encryption randomness). Thus,

$$\text{ENCRYPT}(M, r) = (g^r, y^r M) = C$$

ElGamal decryption

In order to decrypt an ElGamal ciphertext (a, b) and retrieve the plaintext M, we compute:

$$M = \text{DECRYPT}(C) = \frac{b}{a^x} \bmod p$$

2.2.2 Zero Knowledge Proofs

One of the most important (cryptographic) concepts applied in e-voting systems is the ***Zero Knowledge Proof*** [9, 22]. Zero Knowledge Proofs consist of methods which one party (**prover**) can use to prove to another party (**verifier**) that a given statement is true, without revealing any other information apart from the fact that the statement is indeed true. Every form of a Zero Knowledge Proof follows some principles as well:

- **Completeness:** Given that the prover is honest, i.e. if the statement is indeed true, then the verifier will accept the proof. More precisely, completeness ensures that the verifier will accept the proof with a very high probability (close to 1).
- **Soundness:** If the prover is dishonest, i.e. if the statement is false, then the verifier should not accept the proof except with some small probability.
- **Zero knowledge:** The verifier should learn nothing more than the truth of the statement produced by the prover.

A *Zero Knowledge Proof* can be divided in two categories: ***interactive***[22] and ***non-interactive*** [21].

Obviously, an *interactive* zero knowledge proof requires interaction between the prover and the verifier. On the contrary, a *non-interactive* zero knowledge proof does not. Both types, though, abide by the above three properties: completeness, soundness and zero knowledge.

2.2.3 Fiat-Shamir Heuristic

A non-interactive zero knowledge proof demands the use of a **common reference string**, which is shared between the prover and the verifier. It is possible to transform an interactive to a non-interactive proof of knowledge with the *Fiat-Shamir Heuristic*[8].

The heuristic requires the use of a cryptographic hash-function to produce the *shared secret* between the prover and the verifier.

Generally, non-interactive proofs are desired in (remote) e-voting. For instance, consider the case when the authority party (prover) of an election has to convince many verifiers that the tally or mixing is correct. There is no need to compute a separate proof for each verifier. Instead, the prover computes only one proof, which every verifier can validate.

2.2.4 Mixing

As introduced in the previous chapter, *secrecy* is an essential property that cannot be absent from an e-voting protocol. ***Mixing*** is the process responsible for keeping the identity of voters participating in an election secret with the use of a *mixing algorithm*. The component of the system that executes the corresponding mixing algorithm is called a ***mixnet***, accordingly. In other words, a *mixnet* takes a number of votes cast by the eligible voters of the election and it produces a new set of **re-encrypted** and randomly **permuted (mixed)** votes. This type of mixnet is called a "re-encryption mixnet". If someone had only the new set of votes at his disposal, it is computationally infeasible to match any vote with a certain voter. Such a process, however, **must** prove its correctness, as we will see later on.

The concept of *mixnets* was first introduced by David Chaum, in 1981 [6]. In order to achieve anonymity between two correspondents who exchange mails, the existence of a ***mix-server*** **(or "mix")** is defined. Its role is to process mails before they are delivered from a sender to a recipient. Using public key cryptography, the sender prepares a message M and then encrypts it with the recipient's public key K_a. The result is further encrypted using the public key of the mix-server and then it is sent to it. Subsequently, the mix-server uses its corresponding private key to get the encrypted message M and the target address A of the recipient, which delivers the message to. The process described above can be reflected to the the input and output of the mix-server:

$$K_1\left(R_1, K_a\left(R_0, M\right), A\right) \rightarrow K_a\left(R_0, M\right), A$$

where R_0, R_1 are some random strings attached to every message intended to be encrypted, to ensure that nobody is able to retrieve them. e.g. by testing values X such that $K_a(X) = K_a(M)$) with K_1, K_a being the mix's and recipient's public keys, accordingly.

To extend the idea of a *mixnet*, Chaum proposed a cascade of single mixes. In that case, there are N mix-servers arranged in a sequence, along with their own private/public keys. The sender encrypts the initial message M with all the public keys, for each succeeding mix-server in the "chain". For example, the input and output of the first node is

$$K_N(R_N, K_{N-1}(R_{N-1}, \ldots, K_2(R_2, K_1(R_1, K_a(R_0, M), A))\ldots)) \rightarrow \\ K_{N-1}(R_{N-1}, \ldots, K_2(R_2, K_1(R_1, K_a(R_0, M), A)\ldots))$$

and the final output of the cascade is $[K_a(R_0, M), A]$, as expected.

This is a suitable scheme for use in electronic voting systems, in order to preserve the anonymity of a vote. Each vote, that is encrypted and cast by any eligible voter, will be included in the input of a mixnet (cascade of mix-servers) and a final list of decrypted votes will be produced, without revealing any correspondence with them. This requires either a random permutation of the votes before producing an output in each stage, or even an alphabetical order of them, as long as there are no duplicate items in the input.

This type of mixnet is included in the category *decryption mixnets*, as they are called. These types of mixnets have several drawbacks. First of all, if a mix-server is not honest, it can substitute an encrypted message with one of its own, which is not acceptable during the voting process. Furthermore, the last mix-server in the chain is able to decrypt the content of the votes, and if it does not like the output, it may abort the process. That is why a *re-encryption mixnet* is more often used.

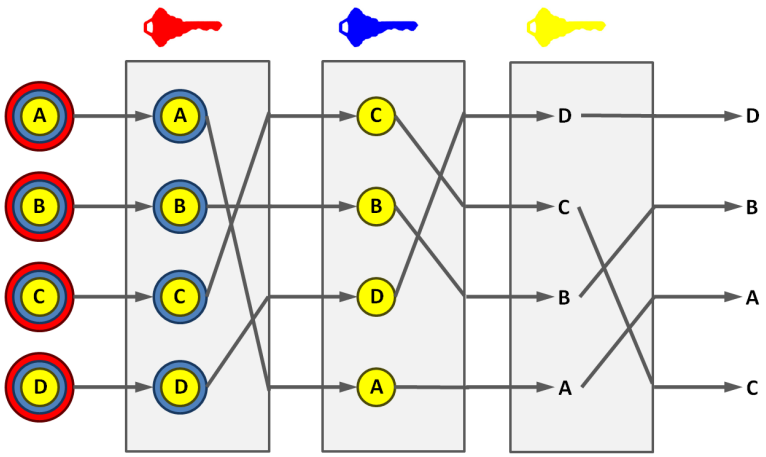

Figure 2.1: Example of a decryption mixnet

On the other hand, a re-encryption mixnet[16] is responsible for re-encrypting and mixing the input votes only and not partially decrypting them. For that reason, a homomorphic encryption scheme, like El-Gamal, is often used. So, every mix-server does exactly that and then forwards the result to the node in the mixnet. The decryption phase is performed separately, after the mixing phase.

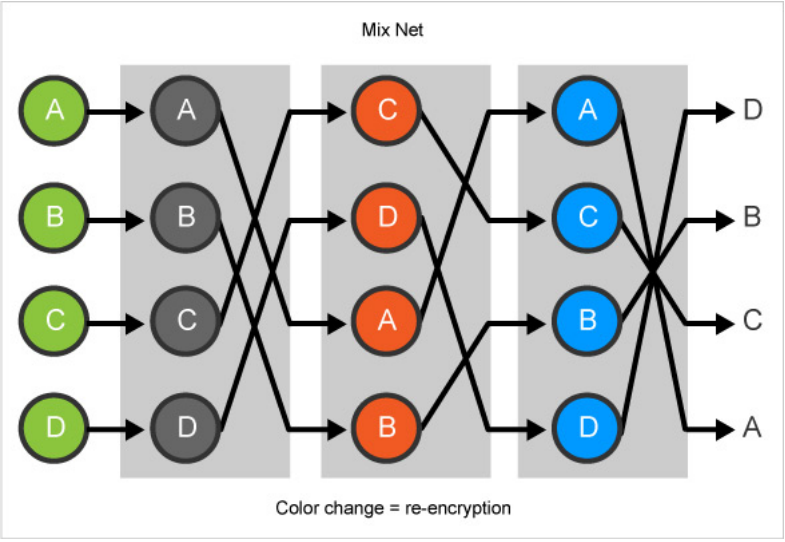

Figure 2.2: Example of a re-encryption mixnet

Finally, in both types, a voter can verify that his vote was indeed counted. He is provided with a (signed) receipt he can use to prove that the final tally contains his vote.

2.3 Coercion

An important parameter of an electronic system that needs to be considered is *coercion*[4]. If an adversary is able to force a voter to cast his vote in a certain way, or even cast a random vote, then the election can be bought. Such behaviour is unacceptable for a voting system. Another type of coercion is *vote selling*, in which a voter voluntarily tries to give away the content of his vote to a third party. Although there are some differences, both cases are quite similar when it comes down to mitigating them.

Generally, it is not an easy task to make an e-voting system coercion-resistant. Due to the advances in current technology, it is quite easy for everyone to acquire a (hidden) video camera, capture their identity and record all the interactions with a poll site. In that way, a voter can sell his vote or prove its content to a third party. Consequently, a coercion-resistant system should instruct its voters to turn off any recording devices during the voting process and not providing them with any receipt that prove how they voted. As mentioned, *receipt-freeness* states that a voter does not gain any information from the system which can be used to prove the content of his vote. As a result, a coercer cannot be sure about the choice of the voter, thus, he cannot influence him.

A coercer can act either locally or remotely. Obviously, it is easier for a coercer to be present at a poll site, where he can interact with the voter, in contrast to a remote coercer, where the voter can make a choice on his own. However, if there is any video recording device or vote receipts are provided, there is no significant difference between the two types.

Apart from the location of a coercer, the time of coercion is very crucial as well. Modern systems can be designed to mitigate *post-election coercion*, while it is practically impossible to deal with *pre-election coercion*. Other than the aforementioned approaches an e-voting system can follow, the availability of an "untappable channel"[4] is demanded as a countermeasure against pre-election coercion.

2.4 Challenges

In conjuction with the principles that a strong e-voting system must preserve, there are some more challenges that must be dealt with, concerning its implementation and efficiency. Some of them are:

- **Speed:** A major concern for the performance of an e-voting system. A system that produces results, i.e. counting the votes, after a very long time interval, from the moment the votes are cast, is not desired.
- **Scalability:** Ideally, an e-voting system is desired to handle any number of voters, or at least the number of particpants that make up a whole country.
- **Ease of use:** A large-scale e-voting system addresses to all the participants of a voting process, e.g. the citizens of a country. Everyone eligible to vote must be provided with an easy-to-use technological means, not affected by limitations, such as language, age etc.
- **Low cost:** An electronic type of voting system is not useful, if it is very expensive. If so, irrelespective to its security, it is not attractive. Therefore, it is likely that a less secure, but cheaper, system is prefered and people possibly will stick to the traditional paper-based model.

Chapter 3

A survey on Mixnets

Having presented all the properties and requirements of a reliable electronic voting system, we focus on the *Mixing*. There are several mixnets proposed by various authors and the goal of this chapter is to search and compare these algorithms, from which one will be chosen for more analysis, implementation and evaluation in the next chapter. As we have already mentioned, we will try to find an alternative mixing algorithm for the Zeus e-voting system.

Every mixnet presented below is a *re-encryption mixnet*[16], which are based on Chaum's idea[6].

3.1 Sako-Kilian Mixnet

Sako and Kilian proposed the first receipt-free electronic voting protocol in 1995 [18]. The protocol relies on the existence of n mix-centers, which produce a zero-knowledge proofs in order to prove that their output is a *re-encryption* and *permutation* of their input. These centers work in a chain, i.e. the output of center i is the input to center $i + 1$ and the last center outputs the decryptions of the initial input ciphertexts, without compromising the privacy of the votes. In order for a node to prove correct mix, another "secondary mix" is produced and depending on the challenge bit of an interactive verifier, a mix-center (prover) reveals either the permutation and the re-encryption parameters of this mix or the differences between the two [2].

The scheme can be used with any homomorphic cryptosystem (most commonly ElGamal) and imposes a security parameter ℓ, which denotes how many times a prover must repeat a zero-knowledge proof of shuffle, so that the possibility of cheating becomes negligible.

Finally, the complexity of the protocol is $642N$, where N is the number of the initial input ciphertexts. The exponentiations per center

can be distributed to a number of parallel workers and, thus, the execution time of the protocol can be significantly improved.

3.2 Jakobsson's Mixnet

Markus Jakobsson proposed a mixnet in 1998, describing a robust method for shuffling ElGamal ciphertexts [13]. His scheme is based on the existence of a certain number of mix-servers. Every server is responsible for permuting a **copy** of the initial input of the mixnet and, at the same time, "hiding" any private data used for that cause (blinding). The copies of the ciphertexts are useful in order to detect any corruption among the participating servers. Both the produced (and processed) copies and the input are sorted and the protocol demands that they are identical. If not, it means that at least one of the mix-servers cheated, which can be found through a separate tracing process. Consequently, the protocol achieves correctness, privacy and robustness.

Regarding its efficiency, the total number of exponentiations demanded to mix-decrypt N ciphertexts is approximately $3\kappa kN + 7k^2N$, where κ is the number of rounds the (blinded) computations are repeated, to obtain the desired degree of security.

Finally, it is worth noting that Yvo Desment and Kaoru Kurosawa in [7] presented an attack for this mixnet that breaches its robustness property.

3.3 Abe's Mixnet

In 1999, Abe proposed two schemes for mixing ElGamal ciphertexts, MiP-1 and MiP-2 accordingly [1]. The main difference between the two is that the former consists of only one phase (randomized decryption) and the latter consists of two phases (permutation and decryption). The core of the two protocols is a *permutation network* [1], which is responsible for the randomization, permutation and, finally, decryption of the input ciphertexts, using robust and universally verifiable methods. The construction of each protocol includes the existence of m mix-servers and they are assigned to a subset of the switching gates of the chosen permutation network. Then, each server, M_k, outputs the permuted (input) ciphertexts of every gate he is in charge of (the input of M_k is the output of M_{k-1}) and produces a zero-knowledge proof for correctness of the process. The output of M_k is verified by all the other servers, where it is deduced whether

M_k is faulty or not. Both protocols follow this basic idea, in a slightly different way.

The exponentiations required for the execution of the two schemes represents their efficiency: $O(mN \log N)$ exponentiations, where N is the number of input ciphertexts.

3.4 Millimix

Millimix is the name of the mixnet proposed by Markus Jakobsson and Ari Juels in 1999 [14]. The core of this mixnet is a *permutation network*, as seen in [1], which is responsible for the re-encryption and permutation of its input ciphertexts. As in every (re-encryption) mixnet, both the re-encryption parameters and the permutation must remain secret and the corresponding zero-knowledge proofs should be provided. The mixnet was designed for ElGamal cryptosystems and works as follows: there are k mix-servers that execute a permutation network, comprised of $O(n \log n)$ switching gates, the role of which is a pairwise re-encryption and permutation. Each gate has to prove correctness of its results, by producing a *Plaintext Equivalence Proof (PEP)* [14], which proves that a ciphertext (α', β') is a valid re-encryption of ciphertext (α, β), and a *Disjunctive Plaintext Equivalence Proof (DISPREP)* [14], which proves that a ciphertext (α', β') is a valid re-encryption of either ciphertext (α_1, β_1) or (α_2, β_2) (ElGamal notation). This process is repeated by every mix-server available and they operate in order, i.e. the input of server i is the output of server $i - 1$. If server i cheated during the execution of the protocol, it is expelled and server $i + 1$ takes its input from server $i - 1$.

The protocol needs $O(kn \log n)$ exponentiations to mix n ciphertexts and to verify the process (each server verifies the proofs of the rest). Also, techniques like batch verification can be applied to reduce the computational cost.

3.5 Furukawa-Sako Mixnet

Furukawa and Sako, in 2001, proposed a more efficient zero-knowledge shuffle proof than Sako and Kilian, requiring $18N$ exponentiations instead of $642N$ for mixing N ciphertexts.

To begin with, they expanded the notion of a (linear) permutation by using a *permutation matrix* in order to re-encrypt and permute the input ballots. The re-encryption parameters and the permutation matrix remain secret and the necessary zero-knowledge arguments

are produced so that a verifier is convinced for their existence. Also, these arguments can be constructed non-interactively by applying the Fiat-Shamir Heuristic [8].

The protocol uses the ElGamal cryptosystem and the input ciphertexts are expected to be ElGamal pairs accordingly.

3.6 Boneh-Golle mixnet

Dan Boneh and Philippe Golle, in 2002, came up with an "almost entirely correct" re-encryption mixnet, meaning that it can only guarantee a correct mixing with a *high probability*, not overwhealming [5]. The idea behind this is that the mixing process can be performed very fast and the results of an election can be announced almost immediately. On the contrary, a common mixnet that constructs all the necessary zero-knowledge proofs first, in order to prove correctness of a shuffle, includes a rather time consuming procedure to announce the results. The authors propose a verification technique with a *constant cost*, despite the number of available servers that might be used. Alternatively, a different and much slower method than the one proposed can be applied, so that every uncertainty at the end of the election is eliminated.

The protocol preserves soundness, correctness (almost entirely - a cheating server can be detected with a high probability) and robustness. Universal verifiability is not achieved in this scheme but it can be checked with a slower proof, as mentioned above.

Concerning its efficiency, the mixnet demands a variable and a constant number of exponentiations per mix-server, $2N + 2\alpha(2k - 1)$, to prove an"almost entirely correct", where α is a security parameter [5]. If no cheating server is found in the process, then we move on to the decryption phase with a cost of $(2 + 4k)N$ exponentiations, where k is the total number of available mix-servers. The constant cost of the verification method is a great advantage for the execution time of the elections, but, as a downside, depending on the chosen security parameter for the execution as well, some information about the (hidden) permutation used by a mix-server might get leaked. This is acceptable for large-scale elections, where there is a great number of input ciphertexts.

3.7 Neff's Mixnet

In his paper, Neff presented two different types of mixing [15]. The first one is the *simple k-Shuffle*, where the input ciphertexts are indi-

vidual cyclic group elements, and the second one regards a number of ElGamal pairs respectively. In both cases there is a prover and a verifier (interactive), where the former has to prove knowledge of the a permutation π and the re-encryption parameters according to which the input elements were shuffled. The verifier generates and sends challenges to the prover so that he can determine the integrity of the process.

The efficiency of each protocol is depicted in the following table:

	Simple k-Shuffle	**Shuffle of ElGamal Pairs**
Proof Construction	$2k$	$8k + 4$
Verification	$4k + 2$	$12k + 4$

Table 3.1: Neff's mixnet complexity

where k denotes the number of input ciphertexts.

3.8 Furukawa's Mixnet

Jun Furukawa, in 2005, proposed two protocols, which process ElGamal pairs [10]. The first one produces an efficient argument for shuffling alone, while the second one for both shuffling and decryption. Like in [11], the notion of a *permutation matrix* is used. The first protocol (shuffle) requires three rounds to produce the corresponding argument and the second one (shuffle and decryption) requires five rounds. Concerning its security, the author stresses that the shuffle-decryption protocol is not zero-knowledge, but it can be proved that an honest verifier cannot learn anything about the hidden permutation, with an overwhelming probability (*Complete Permutation Hiding* [10]).

The efficiency of the protocols (number of total exponentiations) per argument is the following, where k is the number of the input ciphertexts.

		Protocol I	**Protocol II**
Shuffle Argument	Proof Construction	$9k$	-
	Verification	$6k$	-
Shuffle-Decryption Argument	Proof Construction	$10k$	$8k$
	Verification	$8k$	$6k$

Table 3.2: Furukawa's mixnet complexity

3.9 Scytl Mixnet

Jordi Puiggalí Allepuz and Sandra Guasch Castelló, in 2010, proposed a heuristically secure re-encryption mixnet, using the homomorphic properties of ElGamal cryptosystems [17]. Their scheme was implemented and used in real elections, the properties of which (mainly security) were thoroughly examined in [19]. The mixing process includes a certain number of mix-servers, each being responsible for the re-encryption and permutation of its corresponding input ciphertexts, as we have already seen. Ofcourse, both the re-encryption parameters and the permutation remain secret and the mix-servers operate in order, i.e. each passes the processed ciphertexts to the next server in chain. The last server publishes the final mix of the ciphertexts and then the verification phase takes place.

After the mixing is complete, the verification phase starts, where the verifier defines random groups of the input ciphertexts of each mix-server. Subsequently, each mix-server provides the verifier with information about the location in its output of the votes of each input group, without revealing any information about the initial location of the votes in the input groups. Next, the verifier processes the elements of each group (he multiplies them) in order to obtain an *Input Integrity Proof* and does the same for the output, to obtain an *Output Integrity Proof*. In the end, a zero-knowledge proof is constructed for each group of the mix-server to prove that the output integrity proof is a re-encryption of the input integrity proof, using the homomorphic properties of the cryptosystem.

As for its efficiency, the number of exponentiations required for each operation are the following:

Mixing	**Zero-Knowledge proofs**	**Verification**
$2mk$	$2(m/n)k$	$4(m/n)k$

Table 3.3: Scytl's mixnet complexity

where m is the number of the input ciphertexts, n is the number of votes per (input) group and k is the number of the available mix-servers.

3.10 Groth's mixnet

In 2010, Jens Groth suggested a scheme for shuffling a given set of ciphertexts, using the homomorphic properties of ElGamal cryptosystems and commitment schemes [12]. His shuffle procedure follows

Neff's paradigm, concerning the invariance of polynomials under permutation of their roots, in constrast to Furukawa and Sako's scheme [11], where the prover commits to a secret permutation matrix, in order to prove a correct shuffle. All the arguments produced by the protocol are zero-knowledge and the scheme allows techniques such as multi-exponentiation and batch verification to improve its overall efficiency.

The author's final protocol for shuffling homomorphic ciphertexts is a seven-move zero-knowledge argument, based on a simpler four-move zero-knowledge argument for shuffling items with known content (plain messages). Both protocols prove correctness without revealing any information about the secret permutation or the re-encryption parameters that are used.

As for the efficiency, the total number of exponentiations the protocol demands in order to produce the shuffle argument is $6n$ for the prover and another $6n$ for the verifier, which makes it more efficient than [11] and [10], both in computational and communication complexity. This complexity does not include the decryption of the ciphertexts. If we want to shuffle and decrypt a number of ciphertexts with this scheme, we can combine the two operations into one that demands a total of $6n$ exponentiations for the prover and $7n$ exponentiations for the verifier.

3.11 Bayer-Groth Mixnet

In 2012, Bayer and Groth designed an efficient scheme for proving a shuffle [3]. The idea of re-encryption and permutation of the input ciphertexts is preserved, just like we have seen so far. Every verifier is convinced about the correctness of the results, with the prover's commitment to a secret permutation and the corresponding random parameters. The prover constructs two arguments for the proof , a multi-exponentiation argument, where a set of ciphertexts raised to a set of (hidden) values yields a specific ciphertext, and a product argument, where a set of values has a particular product.

Concerning the efficiency of the protocol, we have the following. The prover's computational time can be significantly reduced by carrying most of the operations in the exponent. Furthermore, techniques for fast polynomial multiplication might be applied as well, to further improve the procedure. The authors, through the theoretical and experimental results of their work, showed that for $N = mn$ ciphertexts, arranged in a two-dimentional matrix, the prover's computational complexity is $O(N \log m)$ exponentiations or $O(N)$ exponentiations if further arithmetic optimizations are used. The verification

phase demands $O(N)$ exponentiations respectively. For $N = 100,000$ ciphertexts, their best running time for the shuffle argument is about two minutes (optimized),

The table below sums up the complexities of the mixnets presented in this chapter, with a unified notation for clearness:

Scheme	**Year**	**Proof in expos**	**Verification in expos**
Sako-Kilian	1995	$O(lsN)$	$O(lsN)$
Jakobsson	1998	$3\ell sN + 7s^2N$	
Abe	1999	$O(N \log N)$	
Millimix	1999	$O(sN \log N)$	
Furukawa-Sako	2001	$8N$	$10N$
Boneh-Golle	2002	$2N + 2\ell(2s - 1)$	$2 + 4s$
Neff	2004	$8N + 4$	$12N + 4$
Furukawa	2005	$8N$	$6N$
Scytl	2010	$(2N/v + 2N)s$	$4(N/v)s$
Groth	2010	$6N$	$7N$
Bayer-Groth	2012	$O(N \log m)$	$O(N)$

Table 3.4: Comparison for ElGamal cryptosystems ($N = mn$), s: number of mix-servers, ℓ: security parameter, v: number of votes per group

The algorithm we chose to analyse and implement is the one proposed by Furukawa and Sako. In the next chapter we present its parallelization and evaluation in real conditions.

Chapter 4

Parallelization of Furukawa-Sako Mixnet

In this chapter, we present the *parallelization* of Furukawa-Sako mixnet algorithm. Initially, the scheme is given as written by the authors, along with some necessary modifications, e.g. the transformation into a non-interactive protocol using the Fiat-Shamir technique. In particular, the shuffle and verification phase are explicitly separated and each one is divided into a number of operations, which can be assigned to a number of different parallel workers (tasks). Eventually, the outputs of all workers are combined in a final stage to produce either a proof of shuffle or a verification statement.

The objective in this chapter is both the theoretical and the experiment analysis of the parallel implementation of this algorithm, in order to view its limits and scalability for electronic elections of variable scale. Hence, we will be able to define the tradeoffs there are, which we are interested in, and construct a general model for a practical choice of processors for a parallel execution.

4.1 The algorithm

The original scheme was designed with an interactive verification of a proof shuffle. In the implementation of this thesis both the prover's and the verifier's computations constitute a different algorithm and they are analysed independently from each other. The two phases are described below, presenting the changes we made before we implement them:

Algorithm 7 Shuffle

input: $p, q, g, y, \{C_1, \ldots, C_N\}$ ▷ p, q, g, y : *the parameters of the cryptosystem (ElGamal),* $\{C_i\} = \{(g_i, m_i)\}$ *the input ciphertexts*

$\phi \leftarrow random\_permutation(N)$ ▷ *create permutation*

$\phi^{-1} \leftarrow inverse\_permutation(N, \phi)$ ▷ *create inverse permutation*

$r_1, \ldots, r_N \leftarrow random(N, 1, q)$ ▷ *create N random values* $\in [1, q]$

$C'_1, \ldots, C'_N \leftarrow permute\_ciphertexts(N, \phi^{-1}, \{C_i\}, \{r_i\}, p, g, q, y)$
▷ *premutation and re-encryption*

$\tilde{g}, \tilde{g_1}, \ldots, \tilde{g_N} \leftarrow random\_elements(N + 1, p, g, q)$ ▷ *generate random basis*

$\sigma, \rho, \tau, \alpha, \lambda \leftarrow random(5, 1, q)$
$\{\alpha_i\}, \{\lambda_i\} \leftarrow random(2N, 1, q)$ ▷ *generate 2N + 5 random values*

Computations

$t = g^{\tau}, \upsilon = g^{\rho}, w = g^{\sigma}, u = g^{\lambda}, u_i = g^{\lambda_i}$ mod p ▷ $i = 1, \ldots, N$

$\tilde{g_i}' = \tilde{g}^{r_i} \tilde{g}_{\phi^{-1}(i)}$ mod p, ▷ $i = 1, \ldots, N$

$$\tilde{g}' = \tilde{g}^{\alpha} \prod_{i=1}^{N} \tilde{g}_i^{\alpha_i} \text{ mod p}$$

$$g' = g^{\alpha} \prod_{i=1}^{N} g_i^{\alpha_i} \text{ mod p}$$

$$m' = y^{\alpha} \prod_{i=1}^{N} m_i^{\alpha_i} \text{ mod p}$$

$t_i = g^{3\alpha_{\phi^{-1}(i)} + \tau\lambda_i}$ mod p ▷ $i = 1, \ldots, N$

$\upsilon_i = g^{3\alpha^2_{\phi^{-1}(i)} + \rho r_i}$ mod p ▷ $i = 1, \ldots, N$

$\dot{\upsilon} = g^{(\sum_{i=1}^{N} \alpha_i^3) + \tau\lambda + \rho\alpha}$ mod p ▷ $i = 1, \ldots, N$

$w_i = g^{2\alpha_{\phi^{-1}(i)} + \sigma r_i}$ mod p ▷ $i = 1, \ldots, N$

$\dot{w} = g^{\left(\sum_{i=1}^{N} \alpha_i^2\right) + \sigma\alpha}$ mod p ▷ $i = 1, \ldots, N$

$c_1, \ldots, c_N \leftarrow generate\_random\_challenge(p, g, q, y, N, \tilde{g}, \{\tilde{g}_i\}, \{(g_i, m_i)\}, \{(g'_i, m'_i)\})$ ▷ *Fiat-Shamir Heuristic*

$s = \sum_{i=1}^{N} r_i c_i + \alpha \bmod q$ ▷ *additional values*

$s_i = c_{\phi(i)} + \alpha_i \bmod q$

$\lambda' = \sum_{i=1}^{N} \lambda_i {c_i}^2 + \lambda \bmod q$

$return\ t, \upsilon, w, u, \{u_i\}, \{\tilde{g_i}'\}, \tilde{g}', g', m', \{t_i\}, \{\upsilon_i\}, \dot{\upsilon}, \{w_i\}, \dot{w}, s, \{s_i\}, \lambda'$

Algorithm 8 Verification

input: $p, g, q, y, \{(g_i, m_i)\}, \{(g'_i, m'_i)\}, \tilde{g}, \{\tilde{g}_i\}, t, \upsilon, w, u, \{u_i\}, \{\tilde{g_i}'\}, \tilde{g}', g', m', \{t_i\}, \{\upsilon_i\}, \dot{\upsilon}, \{w_i\}, \dot{w}, s, \{s_i\}, \lambda'$

$c_1, \ldots, c_N \leftarrow generate\_random\_challenge(p, g, q, y, N, \tilde{g}, \{\tilde{g}_i\}, \{(g_i, m_i)\}, \{(g'_i, m'_i)\})$ ▷ *Fiat-Shamir Heuristic*

$e_1 \leftarrow \left(\tilde{g}^s \prod_{i=1}^{N} {\tilde{g}_i}^{s_i} = \tilde{g}' \prod_{i=1}^{N} {\tilde{g}_i}'^{c_i} \right) \bmod p$

$e_2 \leftarrow \left(g^s \prod_{i=1}^{N} g_i^{s_i} = g' \prod_{i=1}^{N} g_i'^{c_i} \right) \bmod p$

$e_3 \leftarrow \left(y^s \prod_{i=1}^{N} m_i^{s_i} = m' \prod_{i=1}^{N} m_i'^{c_i} \right) \bmod p$

$e_4 \leftarrow \left(g^{\lambda'} = u \prod_{i=1}^{N} {u_i}^{c_i^2} \right) \bmod p$

$e_5 \leftarrow \left(t^{\lambda'} \upsilon^s g^{\sum_{i=1}^{N} (s_i^3 - c_i^3)} = \dot{\upsilon} \prod_{i=1}^{N} {\upsilon_i}^{c_i} {t_i}^{c_i^2} \right) \bmod p$

$e_6 \leftarrow \left(w^s g^{\sum_{i=1}^{N} (s_i^2 - c_i^2)} = \dot{w} \prod_{i=1}^{N} {w_i}^{c_i} \right) \bmod p$

$outcome \leftarrow e_1 \wedge e_2 \wedge e_3 \wedge e_4 \wedge e_5 \wedge e_6$

$return\ e_1, e_2, e_3, e_4, e_5, e_6, outcome$

As we can see, the output of the first algorithm constitutes the proof of a correct shuffle, which is the input to the verification algorithm, in order for any (non-interactive) verifier to decide whether the process is valid or not.

4.2 Parallelization

In this section, the analysis of a parallel version of the previous algorithms is presented. To start with, the shuffle and verification phases are split into smaller operations, the load of which can be distributed to a number of parallel workers (tasks). However, there are some dependencies among the problem data and the parallelization technique must ensure the validity of the calculated data. These operations are represented as functions below, where their input and output are clearly stated.

4.2.1 Shuffle

- random_permutation()
  - input: N
  - output: $\phi(i)$
- inverse_permutation()
  - input: $N, \phi(i)$
  - output: $\phi^{-1}(i)$
- generate_randomizers()
  - input: N, q
  - output: $\{r_i\}$
- permute_ciphertexts()
  - input: $N, p, g, q, y, \phi^{-1}(i), \{C_i\}, \{r_i\}$
  - output: $\{C'_i\}$
- generate_random_basis()
  - input: N, p, g, q
  - output: $\tilde{g}, \{\tilde{g}_i\}$
- generate_initial_values()
  - input: N, q
  - output: $\sigma, \rho, \tau, \alpha, \lambda, \{\alpha_i\}, \{\lambda_i\}$
- first_expos()
  - input: $N, \sigma, \rho, \tau, \lambda, \{\lambda_i\}, p$

  - output: $t, v, w, u, \{u_i\}$
- new_g_tilde_list()
  - input: $N, \phi^{-1}, \tilde{g}, \{\tilde{g}_i\}, \{r_i\}, p$
  - output: $\{\tilde{g}_i{}'\}$
- new_g_tilde()
  - input: $N, p, g, \tilde{g}, \{\tilde{g}_i\}, \alpha, \{\alpha_i\}$
  - output: $\tilde{g}'$
- new_g()
  - input: $N, p, g, \{C_i\}, \alpha, \{\alpha_i\}$
  - output: g'
- new_m()
  - input: $N, p, y, \{C_i\}, \alpha, \{\alpha_i\}$
  - output: m'
- t_list()
  - input: $N, p, g, \phi^{-1}(i), \{\alpha_i\}, \tau, \{\lambda_i\}$
  - output: $\{t_i\}$
- v_list()
  - input: $N, p, g, \phi^{-1}, \{\alpha_i\}, \rho, \{r_i\}$
  - output: $\{v_i\}$
- new_v()
  - input: $N, p, g, \{\alpha_i\}, \tau, \rho, \alpha, \lambda$
  - output: $\dot{v}$
- w_list()
  - input: $N, p, g, \phi^{-1}, \{\alpha_i\}, \sigma, \{r_i\}$
  - output: $\{w_i\}$
- new_w()
  - input: $N, p, g, \phi^{-1}, \{\alpha_i\}, \sigma, \alpha$
  - output: $\dot{w}$
- generate_random_challenge()

  - input: $N, p, g, q, y, \tilde{g}, \{\tilde{g_i}\}, \{C_i\}, \{C_i'\}$
  - output: $\{c_i\}$

- additional_values()
  - input: $N, q, \phi, \{r_i\}, \{c_i\}, \alpha, \{\alpha_i\}, \lambda, \{\lambda_i\}$
  - output: $s, \{s_i\}, \lambda'$

4.2.2 Verification

- validate_eq_1()
  - input: $N, p, \tilde{g}, \{\tilde{g_i}\}, s, \{s_i\}, \tilde{g}', \{\tilde{g_i}'\}, \{c_i\}$
  - output: $True/False$
- validate_eq_2()
  - input: $N, p, g, \{C_i\}, s, \{s_i\}, g', \{C_i'\}, \{c_i\}$
  - output: $True/False$
- validate_eq_3()
  - input: $N, p, y, \{C_i\}, s, \{s_i\}, m', \{C_i'\}, \{c_i\}$
  - output: $True/False$
- validate_eq_4()
  - input: $N, p, g, \lambda', u, \{u_i\}, \{c_i\}$
  - output: $True/False$
- validate_eq_5()
  - input: $N, p, g, t, \lambda', \upsilon, \dot{\upsilon}, \{\upsilon_i\}, s, \{t_i\}, \{s_i\}, \{c_i\}$
  - output: $True/False$
- validate_eq_6()
  - input: $N, p, g, w, \dot{w}, \{w_i\}, s, \{s_i\}, \{c_i\}$
  - output: $True/False$

4.2.3 Dependency graphs and parallel method

After the definition of all the operations that make up the two initial algorithms, we need to examine all the data dependencies among them, for the validity of the results. For this reason we create the dependency graphs which visualize the data flows of the problem:

Shuffle proof

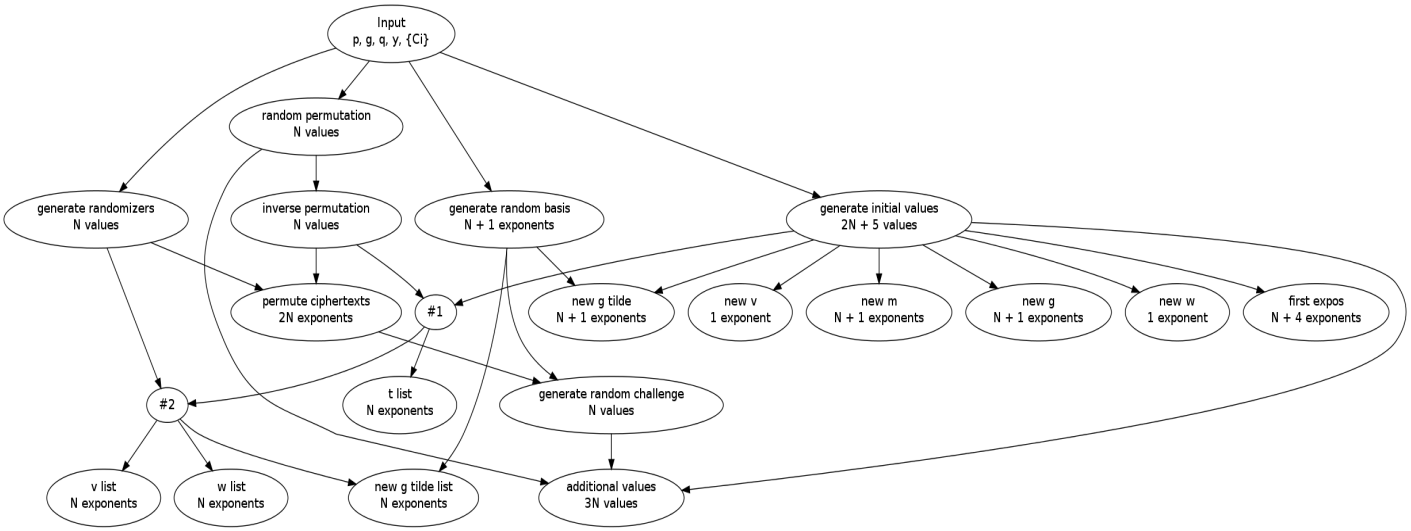

Figure 4.1: Shuffle algorithm dependencies

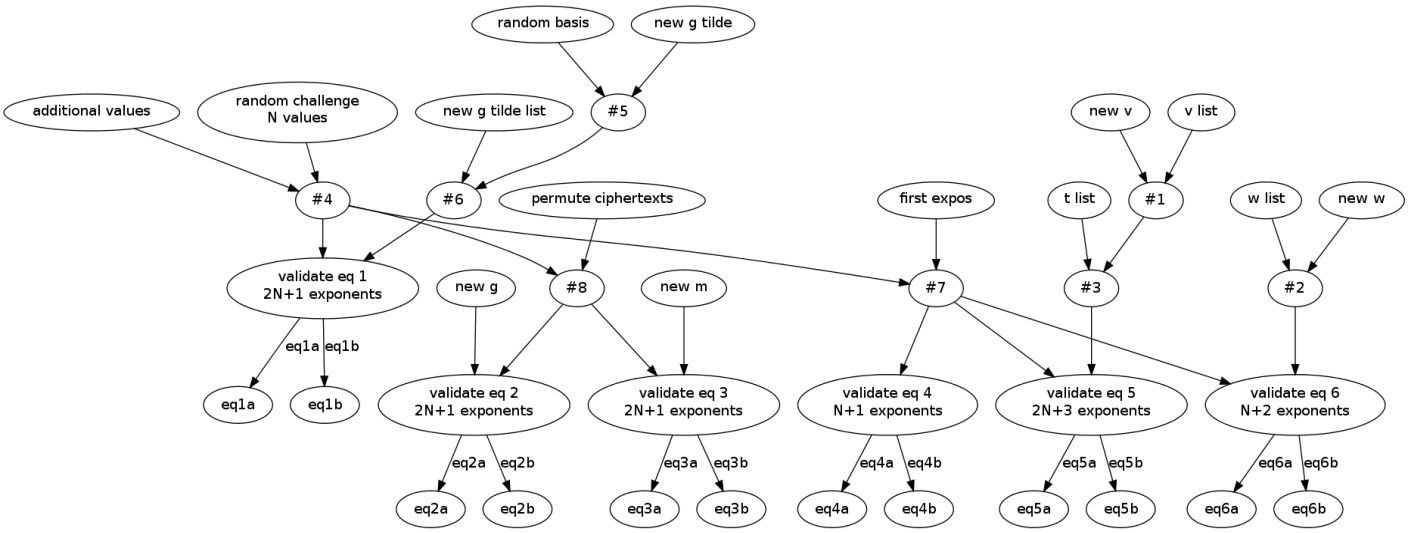

Figure 4.2: Verification algorithm dependencies

From the above figures, it is deduced that some operations precede others, as the the dependency chains of the problems indicate. For

example, we cannot permute (and re-encrypt) the input ciphertexts before we compute the *randomizers* and the *inverse_permutation*. In addition, the cost of every operation is shown, expressed in number of exponentiations. As we have already stressed, exponentiations are way slower than any other operation in this protocol. Those which do not include any exponentiations can be computed serially, but we chose to implement them in parallel, because they will have a significant role in the performance of the program (for a large input) and, consequently, in the maximum acceleration of the mixing phase.

Thus, the process of parallelization is summarized in three steps: first, the input is read and the desired number of independent tasks are created (*break_tasks*); second, every task executes parallel computations (*compute*) and, third, all the results are collected in a single proof of shuffle or a verification statement accordingly (*combine*).

The precise method we used to distribute the total number of exponentiations per task is shown below, where the core of the process lies in the following function:

Algorithm 9

function GET_TASK_INPUTS(length, nr_tasks, task_no)
/* *take the number of input items, number of tasks and return the indices for each task_no* */

$current \leftarrow task\_no$
$idxs \leftarrow \{\}$
while $current < lentgh$ **do**
$idxs \leftarrow idxs \cup \{current\}$
$current = current + nr\_tasks$
return idxs

This function returns a list with indices for each worker, which indicate the positions the results of that worker correspond to. Obviously, the function calculates the indices in a cyclic way. Thus, we are dealing with vector calculations and subsets ("slices") of these vectors are used by different parallel workers. In the context of parallel programming, this corresponds to a SIMD logic (Flynn's taxonomy).

Algorithm 10 Break tasks

input: nr_tasks, infile, directory, mode

/* *read the input ciphertexts and the cryptosystem parameters* */
$p, g, q, y, \{C_i\}, N \leftarrow$ READ_INPUT(infile)

if $num\_tasks > N$ **then**
  $num\_tasks \leftarrow N$ ▷ each operation has at most N exponents
if mode is 'shuffle' **then**
  /* *calculate random values and any common data sequentially* */
  $\phi \leftarrow$ RANDOM_PERMUTATION(N)
  $\phi^{-1} \leftarrow$ INVERSE_PERMUTATION(ϕ)
  $\sigma, \rho, \tau, \alpha, \lambda, \{\alpha_i\}, \{\lambda_i\} \leftarrow$ GENERATE_RANDOM(2, q)
  $\tilde{g_0} \leftarrow$ GENERATE_RANDOM_ELEMENT(p, g, q)
  $sum1 \leftarrow$ SUM($\{\alpha_i^2\}$)
  $sum2 \leftarrow$ SUM($\{\alpha_i^3\}$)
  $\{R_i\} \leftarrow$ CREATE_TASK_FILES($nr\_tasks, directory$)
  for $i = 0$ to $num\_tasks - 1$ **do**
    $my\_idxs \leftarrow$ GET_TASK_INPUTS(N, nr_tasks, i)
    for $k \in my\_idxs$ **do**
      SAVE_IN_MY_TASK($C_k, C_{\phi_k^{-1}}, \alpha_k, \alpha_{\phi_k^{-1}}, \lambda_k, \phi_k, \phi_k^{-1}, R_i$)
  /**we have not assigned these operations to any task yet**/
  $remaining \leftarrow \{\dot{v}, \dot{w}, \tilde{g}^\alpha, g^\alpha, y^\alpha, t, v, w, u\}$
  for $i = 0$ to LENGTH($remaining$) -1 **do**
    $target = i$ mod nr_tasks
    SAVE_IN_MY_TASK($remaining_i, R_{target}$)
else if mode is 'verify' **then**
  $proof \leftarrow$ READ_INPUT(infile)
  $\{c_i\} \leftarrow$ GENERATE_RANDOM_CHALLENGE($proof$)
  $sum1 \leftarrow$ SUM($\{s_i^3 - c_i^3\}$)
  $sum2 \leftarrow$ SUM($\{s_i^2 - c_i^2\}$)
  $\{R_i\} \leftarrow$ CREATE_TASK_FILES($nr\_tasks, proof$)
  for $i = 0$ to $nr\_tasks - 1$ **do**
    $my\_idxs \leftarrow$ GET_TASK_INPUTS(N, nr_tasks, i)
    for $k \in my\_idxs$ **do**
      SAVE_IN_MY_TASK($\tilde{g}_k, s_k, \tilde{g}'_k, c_k, g_k, g'_k, m_k, m'_k, u_k, c_k^2, v_k, t_k, w_k, R_i$)
  $remaining \leftarrow (\{\tilde{g}^s, \tilde{g}'), (g^s, g'), (y^s, m'), (g^{\lambda'}, u), (t^{\lambda'}, \dot{v}), v^s, g^{sum1}, (w^s, \dot{w}), g^{sum2}\}$
  for $i = 0$ to LENGTH($remaining$) -1 **do**
    $target = i$ mod nr_tasks
    SAVE_IN_MY_TASK($remaining_i, R_{target}$)

We can see that the process of "breaking" the calculations are similar for both cases, using the function defined before. They can be distinguished with an argument "mode" and the maximum number of parallel tasks cannot exceed the number of input ciphertexts, which are given as a file (infile). The "directory" argument is mentioned for completeness, it is implementation-specific and can be ignored. What is more, the distribution of calculations happens in two steps, first, the ones of the order of N are given to the parallel workers and second the rest (fixed number) defined by the protocol. Therefore, all tasks get batches of (N/nr_tasks) exponents for computation and the first $(N$ mod $nr\_tasks)$ get one extra (we assume that the parallel workers are numbered - 0, 1, ...). The disadvantage of this distribution is that if we break many tasks of N exponents, then if N is not divided by the number of workers evenly, some workers will end up computing slightly more exponentiations than the rest, so, the distribution will not be uniform.

Now that the workers have been created, they can be run in parallel using the following algorithm:

Algorithm 11 Compute

input: task_file, directory, mode

$p, g, q, y, \{C_i\}, N \leftarrow$ READ_INPUT($task\_file$)
$data \leftarrow$ READ_INPUT($task\_file$)
if mode is 'shuffle' **then**
/* *read all the necessary data for the computations as input e.g. cryptosystem parameters, other expected values, etc* */
$my\_idxs, data \leftarrow$ READ_INPUT(task_file)
$R \leftarrow$ CREATE_MY_RESULTS_FILE($directory$)
/* *run each shuffle operation for this task file's indices* */
$\{r_i\} \leftarrow$ GENERATE_RANDOMIZERS($my\_idxs, data$)
$\{C'_i\} \leftarrow$ PERMUTE($my\_idxs, \{r_i\}, data$)
$\{\tilde{g}_i\} \leftarrow$ GENERATE_RANDOM_BASIS($my\_idxs, data$)
$\{\tilde{g}'_i\} \leftarrow$ NEW_G_TILDE_LIST($my\_idxs, data$)
$\tilde{g}' \leftarrow$ NEW_G_TILDE($my\_idxs, data$)
$g' \leftarrow$ NEW_G($my\_idxs, data$)
$m' \leftarrow$ NEW_M($my\_idxs, data$)
$\{t_i\} \leftarrow$ T_LIST($my\_idxs, data$)
$\{v_i\} \leftarrow$ V_LIST($my\_idxs, data$)

$\{w_i\} \leftarrow$ W_LIST($my\_idxs, data$)
$\{u_i\} \leftarrow$ U_LIST($my\_idxs, data$)
/ save results to file */*
SAVE_IN_MY_RESULTS_FILE($\{r_i\}, \ldots, \{w_i\}, R$)
$remaining \leftarrow \{\tilde{g}^{\alpha}, g^{\alpha}, y^{\alpha}, \dot{v}, \dot{w}, t, v, w, u\}$
for op in $remaining$ **do**
  if op in $task\_file$ **then**
    $res \leftarrow$ COMPUTE_OP(op)
    SAVE_IN_MY_RESULTS_FILE(res, R)
else if mode is 'verify' **then**
  $R \leftarrow$ CREATE_MY_RESULTS_FILE($directory$)
  $my\_idxs, data \leftarrow$ READ_INPUT(task_file)
  / prodX{1,2} indicate the left-hand and right-hand products of equalities 1, ..., 6 of the verification phase */*
  $prod_{11}, prod12 \leftarrow$ COMPUTE_EQ$_1$($my\_idxs, data$)
  $prod_{21}, prod_{22} \leftarrow$ COMPUTE_EQ$_2$($my\_idxs, data$)
  $prod_{31}, prod_{32} \leftarrow$ COMPUTE_EQ$_3$($my\_idxs, data$)
  $prod_{42} \leftarrow$ COMPUTE_EQ$_4$($my\_idxs, data$)
  $prod_{52} \leftarrow$ COMPUTE_EQ$_5$($my\_idxs, data$)
  $prod_{62} \leftarrow$ COMPUTE_EQ$_6$($my\_idxs, data$)
  SAVE_IN_MY_RESULTS_FILE($prod_{11}, \ldots, prod_{62}, R$)
  $remaining \leftarrow \{(\tilde{g}^s, \tilde{g}'), (g^s, g'), (y^s, m'), (g^{\lambda'}, u), (t^{\lambda'}, \dot{v}), v^s, g^{sum1}, (w^s, \dot{w}), g^{sum2}\}$
  for op in $remaining$ **do**
    if op in $task\_file$ **then** $res \leftarrow$ COMPUTE_OP(op)
    SAVE_IN_MY_RESULTS_FILE(res, R)

The workers can either save the results in a file or, as shown above, communicate with directly with other workers that need their data (e.g. the worker that performs the "break" and "combine" operations).

The last stage is the combination of the *combination* of the results from the workers, which is shown below:

Algorithm 12 Combine results

input: result_files, outfile, mode

$p, g, q, y, N \leftarrow$ READ_INPUT($task\_file$)
if mode is 'shuffle' **then**
  /* *proof contains the empty lists and values* $\{r_i\}$, $\{C_i\}, \{C'_i\}, m', g', \{\tilde{g}_i\}, \ldots, \{t_i\}, \{u_i\}$ */

  $proof \leftarrow$ INITIALIZE_PROOF(N)
  for f in $result\_files$ **do**
    /* *read results from a file and put them in the right positions (my_idxs) of the proof lists* */

    $my\_idxs, my\_\{r_i\}, my\_\{C_i\}, my\_\{C'_i\}, my\_m',$
    $my\_g', my\_\{\tilde{g}_i\}, \ldots, my\_\{t_i\}, my\_\{u_i\} \leftarrow$ READ_INPUT(f)
    INSERT_TO_PROOF($proof, my\_idxs, my\_\{r_i\}, \ldots, my\_\{u_i\}$)
  $\{c_i\} \leftarrow$ GENERATE_RANDOM_CHALLENGE($proof$)
  $s, \{s_i\}, \lambda' \leftarrow additional\_values(\{c_i\})$
  INSERT_TO_PROOF($proof, s, \{s_i\}, \lambda'$)
  SAVE_PROOF_TO_FILE($outfile$)
else if mode is 'verify' **then**
  /* *initialize products* */
  $prod_{11}, prod_{12}, \ldots, prod_{61}, prod_{62} \leftarrow 1$
  for f in $result\_files$ **do**
    $p_{11}, p_{12}, \ldots, p_{61}, p_{62} \leftarrow$ READ_INPUT(f)
    $update\_products(p_{11}, p_{12}, \ldots, p_{61}, p_{62}, prod_{11}, prod_{12}, \ldots, prod_{61}, prod_{62})$
  /* *check if all verification equalities stand* */
  if $prod_{11} = prod_{12} \wedge \ldots \wedge prod_{61} = prod_{62}$ **then**
    SAVE_TO_FILE($outfile$, $VALID$)
  else
    SAVE_TO_FILE($outfile$, $INVALID$)

We stress that most of the serial parts shown in the above functions (sums, inverse permutation etc) are run in parallel as well, because they will play an important role in the performance of the program, which will become clearer in a following section. The reason they are presented as serial operations is for simplicity, so that the idea of the parallelization is better understood. Ultimately, how we execute these "serial" parts in parallel, is no different than the methods above.

In order to understand how the data is split into nr_tasks parallel workers, let us consider the following example for one of the operations described earlier, where an input of 10 ciphertexts and three

parallel workers/tasks is assumed.

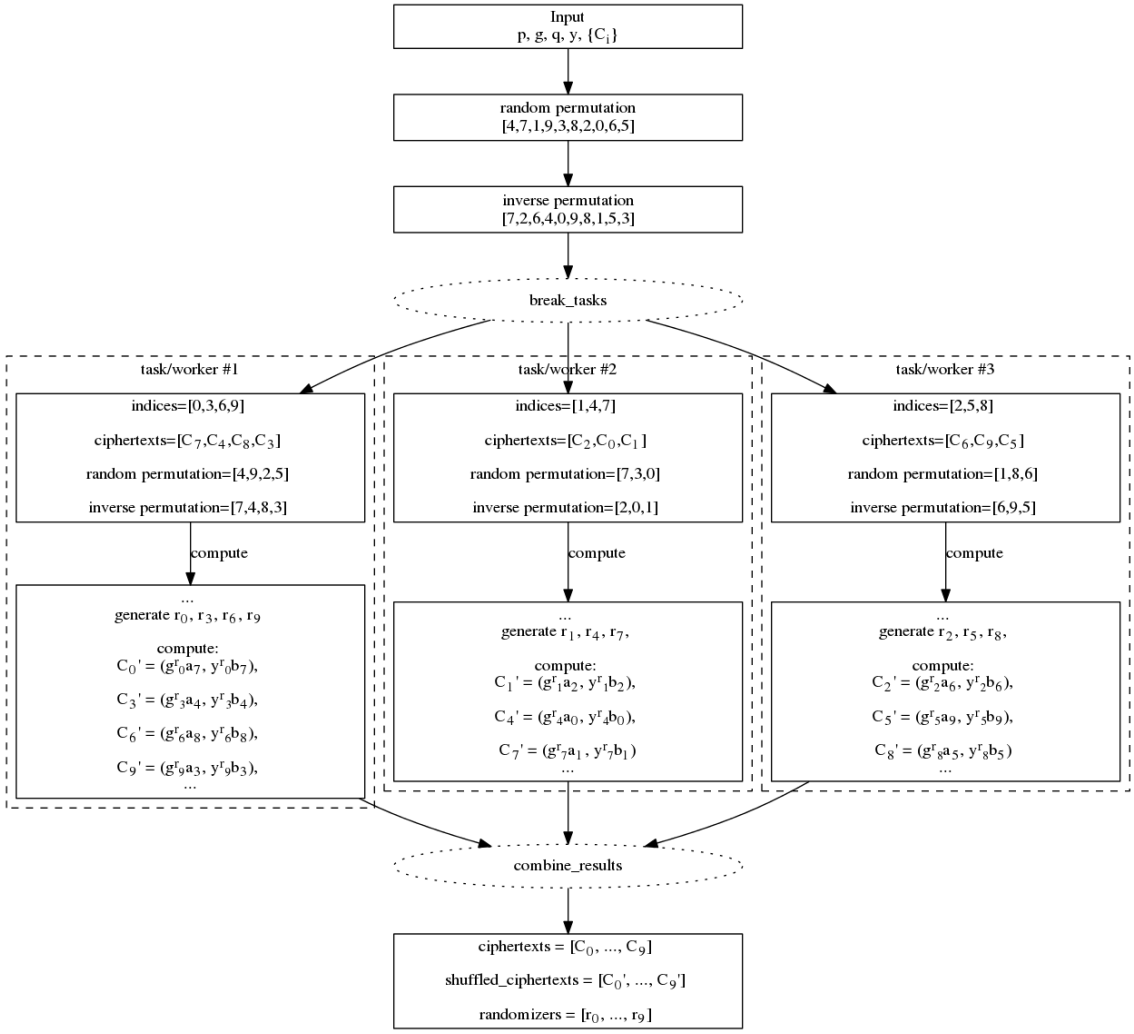

Figure 4.3: Example of permutation and re-encryption in parallel

Here, we can see the cyclic distribution of data we used as basis for the parallelization of the current scheme. All workers executes computations independenly from each other and the only moments of communication occur when they exchange data with a master worker, during the the “break” and “combine” operations. Furthermore, we can see that some tasks can be merged, e.g. the generation of randomizers can be run inside the re-encryption process.

Also, again for simplicity, the inverse permutation is calculated serially. This example, shows exactly the meaning of the term "*parallelized vector calculations*".

In the next section, we quantify our analysis with the execution time prediction and the speedup of the parallelized protocol. Before we do this, though, we must justify the method of parallelization. As it was initially stated, this thesis was prepared in the context of the the "Zeus" e-voting system. Thus, we **chose** not to parallelize all of the operations of the problem, like the generation of random permutations or any hashing operations. This is because of restrictions and security concerns imposed by the system. The architecture of Zeus will judge if how (and if) it can include this implementation in its workflow. However, what we did was to parallelize almost entirely the rest of the algorithm (even the "large" sums and multiplications) and especially all the exponentiations, which are the main focus of attention.

4.2.4 Execution time and speedup prediction

In this part of the analysis, we attempt to create an estimate about the running time and, eventually, the speedup we can achieve with the parallelized protocol. Before that, it is essential that we highlight some fundamental properties of parallel programs.

Amdahl's Law

Parallel algorithms constitute the effort to improve the performance of a solution to a certain problem with the use of more than one cores/processors. The most common metric for this relative performance is the *speedup*, which was was established by Amdahl. It simply states that speedup can be defined as

$$S = \frac{T_s}{T_p} = \frac{1}{f + \frac{1-f}{P}}$$

where T_s denotes the running time of the (best) serial algorithm that solves a problem and T_p denotes the total execution time using P cores for that program. Alternatively, f is the percentage of the serial execution time (it cannot be parallelized) and $1 - f$ is the parallel execution time accordingly, if T_s is normalized to 1. The significance of Amdahl's law is better captured through an example: if we manage to parallelize 90% of a program using 10 cores and the remaining 10% executes serially, then Amdahl's law predicts a maximum speedup of around 5.2. This means that the serial part of a program must be as small as possible, so that we can achieve a high speedup.

Although Amdahl's law is widely used in parallel systems, it does not include other overheads, e.g. communication between processors, synchronization and more. The effects of it can be visualized in the following figure.

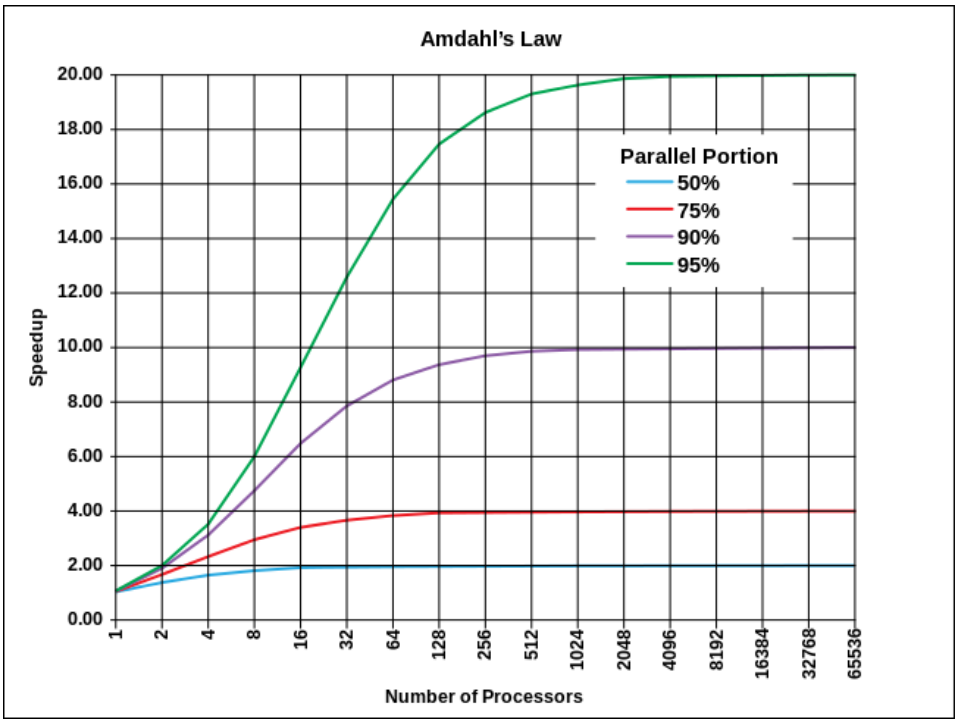

Figure 4.4: Amdahl's Law for speedup

Consequently, the main focus of parallel programming is to parallelize as much of that serial part as possible, even though it is one of the hardest tasks as a key to improvement.

Execution time prediction

To begin with, we assume that the running time for our parallel implementation is divided in three parts: serial, parallel and communication. The serial part comes from the "break" and "combine" phases as described in the previous section, the parallel part represents the code that runs on a certain number of processors and it is mainly the running time due to the exponentiations and, finally, the communication part regards the time consumed on data transfers between the processors/cores. Furukawa and Sako's scheme has a high

degree of (data) parallelism, which we can take advantage of, so as to create isolated sets of computations among the processors, as shown with its parallelization in the beginning of the chapter.

For the case of the parallel part of the execution time, we proceed as follows. If we split N exponentiations into nr_tasks parallel workers then each will get $N/nr\_tasks > 0$ exponentiations, if $N > nr\_tasks$, and remain N mod nr_tasks more, which are given to workers $0 \leq i \leq (N \text{ mod } nr\_tasks - 1)$. If $N < nr\_tasks$, then the maximum workers that are created is N, due to the fact that "batches" of N exponents are distributed each time. If there are P available processors, the maximum degree of data parallelism is reached when $nr\_tasks = P$. From the dependency graphs it is deduced that there are eleven "bathes" of N exponents and nine "remaining" for the shuffle phase and ten "bathes" of N exponents and nine "remaining" for the verification phase. Thus, the load of each worker (here we assume the workers are processors) P_i is:

$$\text{SHUFFLE}(P_i, P) = 11 \cdot \left[\left[\frac{N}{P} \right] + \begin{cases} 1, & \text{if } i < (N \text{ mod } P) \\ 0, & \text{otherwise} \end{cases} \right] + \left[\frac{9}{P} \right] + \begin{cases} 1, & \text{if } i < (9 \text{ mod } P) \\ 0, & \text{otherwise} \end{cases}$$

$$\text{VERIFY}(P_i, P) = 10 \cdot \left[\left[\frac{N}{P} \right] + \begin{cases} 1, & \text{if } i < (N \text{ mod } P) \\ 0, & \text{otherwise} \end{cases} \right] + \left[\frac{9}{P} \right] + \begin{cases} 1 & \text{, if } i < (9 \text{ mod } P) \\ 0, & \text{otherwise} \end{cases}$$

Similar to the above idea, the communication complexity in number of bits per processor can be constructed as follows, where *"send"* stands for the amount of data given to a CPU as input and *"receive"* stands for the amount of data each CPU returns as output. For clearness, the formulas have been replaced with functions, adapted to the parallelization scheme. Also, we assume that $P > 1$ and numbers of B bits:

```
function SHUFFLE_SEND(P_i, P, N)
    /* return the numbers sent to processor P_i */
    if P < 2 then
        return 0
    numbers ← 10 + 7 · [N/P]
    if i < (N mod P) then
        numbers ← numbers + 7
    idxs ← GET_TASK_INPUTS(9, P, i)
    if 2 ∈ idxs then
        numbers ← numbers + 1
    if 3 ∈ idxs then
        numbers ← numbers + 1
    numbers ← numbers + 2 + (N/P)
    return numbers · B

function SHUFFLE_RECEIVE(P_i, P, N)
    /* return the numbers received from processor P_i */
    if P < 2 then
        return 0
    numbers ← 3 + 9 · [N/P]
    if i < (N mod P) then
        numbers ← numbers + 9
    idxs ← GET_TASK_INPUTS(9, P, i)
    for i = 2, ..., 7 do
        if i ∈ idxs then
            numbers ← numbers + 1
    numbers ← numbers + (N/P) + 3
    return numbers · B
```

```
function VERIFICATION_SEND(P_i, P, N)
    /* return the numbers sent to processor P_i */
    if P < 2 then
        return 0
    numbers ← 4 + 12 · [N/P]
    if i < (N mod P) then
        numbers ← numbers + 12
    idxs ← GET_TASK_INPUTS(9, P, i)
    /* do not send the same number more than once */
    sent ← {}
    if 0 ∈ idxs then
        sent ← sent ∪ {s}
        numbers ← numbers + 3
    if 1 ∈ idxs then
        if s ∉ sent then
            sent ← sent ∪ {s}
            numbers ← numbers + 1
        numbers ← numbers + 1
    if 2 ∈ idxs then
        if s ∉ sent then
            sent ← sent ∪ {s}
            numbers ← numbers + 1
        numbers ← numbers + 1
    if 3 ∈ idxs then
        sent ← sent ∪ {l'}
        numbers ← numbers + 2
    if 4 ∈ idxs then
        if l' ∉ sent then
            numbers ← numbers + 1
        numbers ← numbers + 2
    if 5 ∈ idxs then
        if s ∉ sent then
            sent ← sent ∪ {s}
            numbers ← numbers + 1
        numbers ← numbers + 1
    if 6 ∈ idxs then
        numbers ← numbers + 1
    if 7 ∈ idxs then
        if s ∉ sent then
            numbers ← numbers + 1
        numbers ← numbers + 2
```

if $8 \in idxs$ **then**
$numbers \leftarrow numbers + 1$
$numbers \leftarrow numbers + 2(N/P) + 1$
return $numbers \cdot B$

function VERIFY_RECEIVE(P_i, P, N)
/ return the numbers received from processor i */*
if $P < 2$ **then**
return 0
return $14 \cdot B$

The last parameter to complete the estimate for the running time of the parallel protocol is the serial fraction of each phase ("break" and "combine"). The operations involved, as presented earlier, indicate a linear relationship $(O(N))$ between the serial running time and the input ciphertexts. Some of them, do not have a significant impact on the total running time of the protocol when N is quite small, but as N grows large, they determine the overall performance. Those included in this analysis are the generation of random integers and challenges, the creation of a random permutation and its inverse, the initialization and access/traverse of a (large) list and calculation of sums. Thus, the relationship for each phase becomes:

$$\text{SHUFFLE\_SERIAL\_TIME}(N) = \alpha N + \text{E}$$

$$\text{VERIFY\_SERIAL\_TIME}(N) = \beta N$$

where E is the average time per exponentiation.

So, combining all the previously defined relationships, the total execution time for each phase are formed as follows:

$$\begin{aligned}
\text{time(shuffle)} = & \max_{0 \le i \le (P-1)} (\text{SHUFFLE}(P_i, P)) \cdot \text{E} + \\
& \sum_{i=0}^{P-1} \text{SHUFFLE_SEND}(P_i, P, N) \cdot \frac{1}{B} \cdot \text{D} + \\
& \sum_{i=0}^{P-1} \text{SHUFFLE_RECEIVE}(P_i, P, N) \cdot \frac{1}{B} \cdot \text{D} + \\
& \text{SHUFFLE_SERIAL_TIME}(N) \Rightarrow \\
\text{time(shuffle)} = & \left[11 \left(\left[\frac{N}{P} \right] + 1 \right) + \left[\frac{9}{P} \right] + 1 \right] \cdot \text{E} + \\
& (18N + 18P + 8) \cdot \text{D} + \\
& \alpha N + \text{E}, P > 1.
\end{aligned}$$

$$
\begin{aligned}
\text{time(verification)} = & \max_{0 \le i \le (P-1)} (\text{VERIFY}(P_i, P)) \cdot \text{E} + \\
& \sum_{i=0}^{P-1} \text{VERIFY\_SEND}(P_i, P, N) \cdot \frac{1}{B} \cdot \text{D} + \\
& \sum_{i=0}^{P-1} \text{VERIFY\_RECEIVE}(P_i, P, N) \cdot \frac{1}{B} \cdot \text{D} + \\
& \text{VERIFY\_SERIAL\_TIME}(N) \Rightarrow \\
\text{time(verification)} = & \left[10 \left(\left[\frac{N}{P} \right] + 1 \right) + \left[\frac{9}{P} \right] + 1 \right] \cdot \text{E} + \\
& \text{D} \cdot \left[12 + 14N + \begin{cases} 4, & \text{if } P = 2 \\ 5, & \text{if } P = 3 \\ 6, & \text{if } P = 4 \\ 5, & \text{if } P = 5 \\ 6, & \text{if } P = 6 \\ 6, & \text{if } P = 7 \\ 7, & \text{otherwise} \end{cases} + \right. \\
& \left. 19P \right] + \beta N, \; P > 1.
\end{aligned}
$$

where B-bit numbers are assumed, E is the average time per exponentiation as mentioned before and D is the average time for sending/receiving a B-bit number to/from a task(process) using a communication channel.

Furthermore, we can define the speedup at this point for the two main stages of the protocol, where T_s is the execution time of the program when using only one processor ($P = 1$).

$$S_s = \frac{\text{time(shuffle)}|_{P=1}}{\text{time(shuffle)}|_P}$$

$$S_v = \frac{\text{time(verification)}|_{P=1}}{\text{time(verification)}|_P}$$

Having introduced Amdahl's law, we expect that the speedup for both phases will scale up to a certain point, where a maximum value is reached. This is due to both the serial and communication parts of the algorithms. In particular, the serial time is constant, the parallel time decreases and the communication time increases with the addition of more processors for a certain value of input ciphertexts. The number of processors where the maximum speedup is observed is when the total parallel execution time reaches its minimum value, or, alternatively, the rates of change for the parallel and communication times are equal and opposite:

$$\frac{dS_s}{dP} = 0 \Rightarrow P = \sqrt{\frac{(11N+9)E}{18D}}$$
$$\frac{dS_v}{dP} = 0 \Rightarrow P = \sqrt{\frac{(10N+9)E}{19D}}$$

In the next section these two expressions are depicted in separate figures, through through which we can search for the CPU power we need for the mixing in real conditions. More conclusions on the speedup diagrams are explained later on.

4.2.5 Performance

In this part, the performance of the parallel shuffle and verification is presented. The two algorithms were tested on a virtual machine with 8 cores at 2.1 GHz for several cases of input ciphertexts, using 2048-bit numbers.

The first figures presented here concern measurements for the estimation of the serial parts of the algorithms and, shortly after, the results of the actual execution are shown.

Parameter estimation for each execution time fraction

Using the functions defined earlier, we assume an average of $5.6msec$ for computing an exponent and an average of $34\mu sec$ for transferring a number from and to a process. A "simple" power, like square or cube of a number, we assume it takes 10^{-5} and $2 \cdot 10^{-5}$ sec respectively, based on an average for these operations. In the implemented protocol, a queue was used for the inter-process communication and the all the values listed here were measured with a relatively simple benchmarking, using some (Python) scripts on our machine.

For the exponentiation time, a script generates two 2048-bit numbers and then the time elapsed for a *power* call is measured, where one acts as a base and the other as the exponent. This is repeated for NUM times (e.g. 1000) and in the end we can calculate an average for these times.

For the transfer time of a number, a second script creates a queue and a child process so that their communication is tested. For $n = 1, \ldots, L$ the parent process writes/sends n 2048-bit numbers in the queue and measures the time elapsed until the child process reads/receives those numbers. This is repeated for NUM times in order to produce an average for every value of n. We examine the range of $1, \ldots, L$ numbers because there may be a standard overhead when

writing in a queue (it was observed during measurements). Therefore, we consider the time elapsed for $n = 1$ number as a base and after having measured the time elapsed for sending $n = 2, \ldots, L$ numbers, we divide each by $n - 1$ to get a time estimation for transferring a single number in a queue. The average of these times will yield the final transfer time of a number through a queue.

Furthermore, another script was used to test certain operations in the protocol, e.g. the creation of a random permutation, its inverse, the generation of random challenges and numbers (integers) and any arithmetic operation that may have a significant effect on the (parallel) execution time. More precisely, each operation was run NUM times for $L = 1000, \ldots, 10000$, in our case, input ciphertexts and we receive an average for each value. The following figures depict these specific measurements and they confirm the linear relationship between the corresponding times and the length of the input:

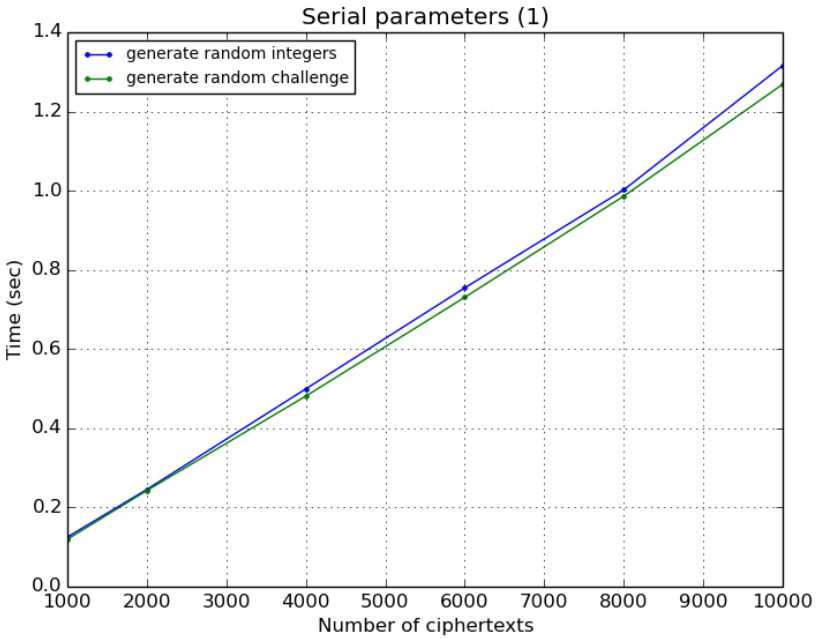

Figure 4.5: Time elapsed for generating random integers and challenges

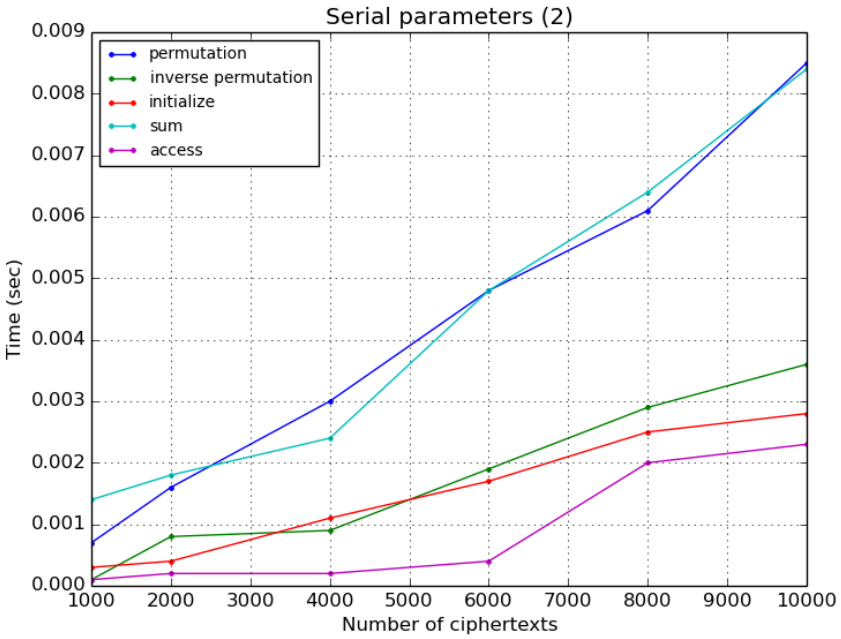

Figure 4.6: Time elapsed for the rest parameters in the serial parts

The flunctuations observed in the above figures are due to the number precision (6 decimal digits) used for the time, where only subtle differences between these operations can be detected, especially for a relative low number of ciphertexts. Eventually, a linear interpolation was applied and the (linear) factor for every operation we are interested in is produced. The following table shows this result:

Operation	**Linear factor** $(\frac{seconds}{ciphertext})$
permutation	$0.8365 \cdot 10^{-6}$
inverse permutation	$0.3814 \cdot 10^{-6}$
initialize list	$0.2997 \cdot 10^{-6}$
access list	$0.2603 \cdot 10^{-6}$
sum	$0.7956 \cdot 10^{-6}$
generate ints	$0.1306 \cdot 10^{-3}$
generate challenge	$0.1267 \cdot 10^{-3}$

Table 4.1: Linear factors for serial parameters

Finally, a general formula for the serial fraction of both shuffle and verification phases, respectively, is given, based on the previous table:

$$\text{SHUFFLE\_SERIAL\_TIME}(N) = (permutation + 22 \cdot access + 17 \cdot init + challenge) \cdot N + 5 \cdot generate\_int + e \Rightarrow$$

$$\boxed{\text{SHUFFLE\_SERIAL\_TIME}(N) = 0.00014 \cdot N + 0.0056 \text{ sec}\,, N > 0, P > 1}$$

$$\text{VERIFY\_SERIAL\_TIME}(N) = (challenge + 12 \cdot access + 12 \cdot init) \cdot N \Rightarrow$$

$$\boxed{\text{VERIFY\_SERIAL\_TIME}(N) = 0.000133 \cdot N \text{ sec}}$$

Results

Initially, the speedup diagrams for the measured parallel shuffle and verification phases are shown alone, where an additional performance metric, the processed ciphertexts/sec, are included. Obviously, the higher the value of this metric, the better the performance is:

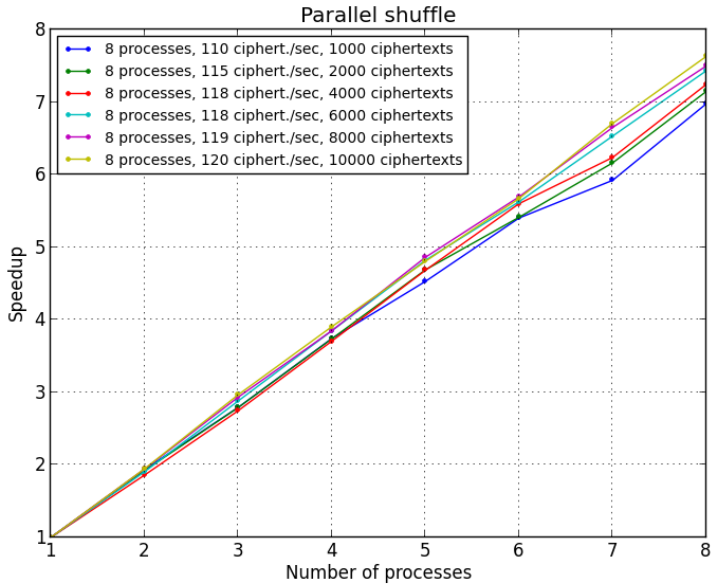

Figure 4.7: Speedup for parallel shuffle

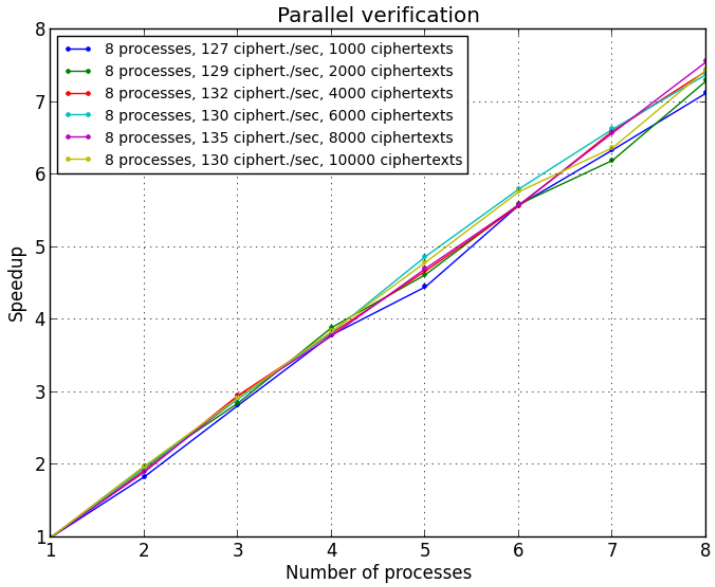

Figure 4.8: Speedup for parallel verification

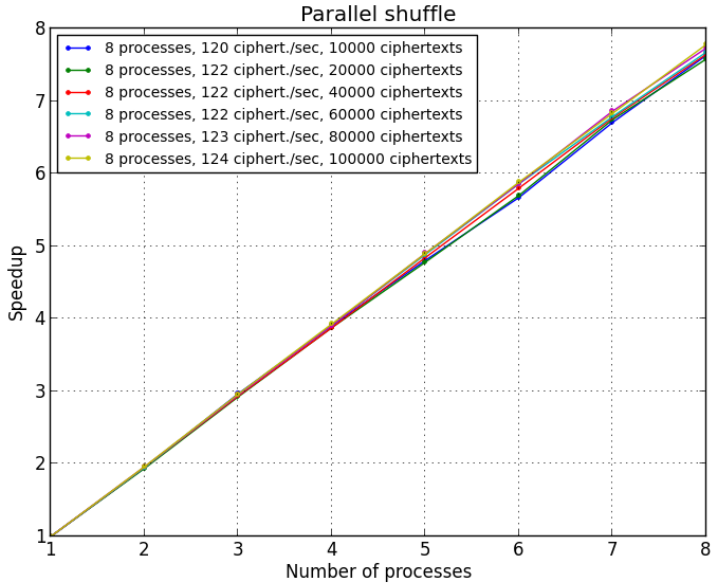

Figure 4.9: Speedup for parallel shuffle - larger input

It can be seen that the speedup of the two parallel algorithms reaches almost its ideal value, $S_{ideal} = P$. Consequently, the scaling is quite "good", because we achieve a speedup of almost 8 with the addition of 8 cores. However, as Amdahl's law indicates, the serial parts and the communication cost set a barrier to the constant increase of speedup. The communication complexity does not have a significant effect on the running time yet, because the number of processors is relatively low. Furthermore, the high degree of parallelism that the Furukawa-Sako scheme offers, contributes to the shown result as well, as the most computationally expensive part (exponentiations) can be run in isolation on the available processors, including a communication overhead with a master process only. In fact, even up to 100000 ciphertexts, our implementation achieves a speedup greater than 7, using 8 cores. But, will see later the speedup curves diverge even more from the ideal curve, when we increase the number of processors.

Also, the spacial complexity of our program is worth mentioning: for 10000 ciphertexts, our implementation produces around 73MB of on-disk text data as a proof and for 100000, around 740MB. More generally, for 2048-bit numbers, we have approximately 617 decimal digits per number. Thus, our proof of shuffle can be translated as follows in terms of disk usage:

$$U(N) = (12N + 14) \cdot 617 \text{ bytes.}$$

where N is the number of input ciphertexts. It is obvious that there is a linear relationship with the input.

In the following figures, the relationship between the total execution time and its components becomes clearer, showing each part separately, i.e. the parallel time, the communication time and the serial time for a variable number of ciphertexts.

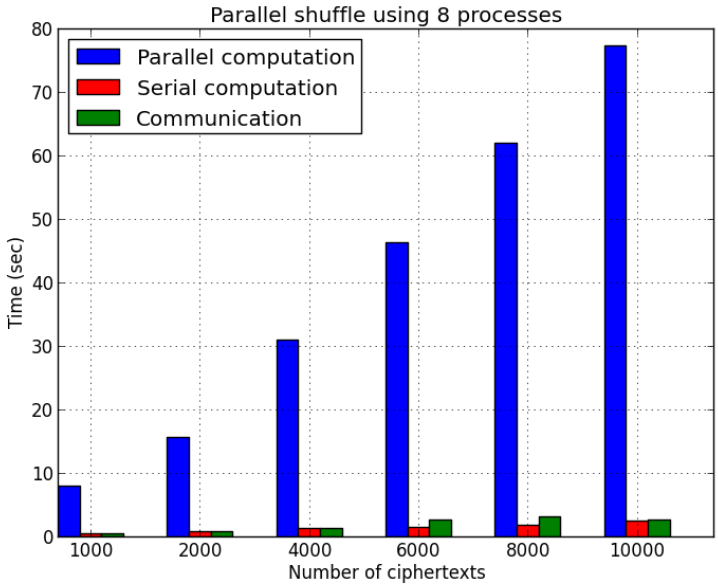

Figure 4.10: Parallel, serial and communication time for shuffle

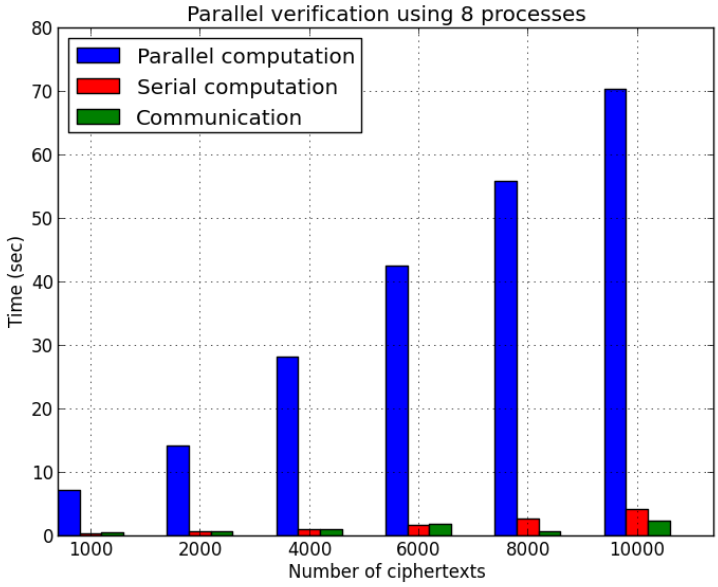

Figure 4.11: Paralllel, serial and communication time for verification

For completeness, we are showing the results for an input up to 100000 ciphertexts. It is noted that in the following analyses, we have the 10000 input as a reference and they would not differ if we used some other value from our measurements.

As expected, the computation of the exponents, dominate the running time. The serial fractions increase as the number of ciphertexts increases and the communication time altough it increases, it has some flunctuations, probably caused by the environment the program runs in. Some parameters that may affect the running time are the synchronization on a shared means of communication (e.g. a queue) and other thead/process management operations (e.g. scheduling). These times are already included in the communication time of the parallel protocol, as it is calculated as the difference between the total execution time and the sum of the parallel and serial fraction.

The next focus of attention, and the most significant one, is the figures that present the comparison between the predicted and the measured speedup of the parallel protocol. As mentioned from the beginning of this thesis, the reason why we need to predict the behavior of the protocol is because we are searching for a maximum number of processors, given its parallelization, in order to perform parallel mixing.Here, an input of 10000 is assumed (it is noted that up to 100000 that we tested the parallel protocol, the diagrams are similar):

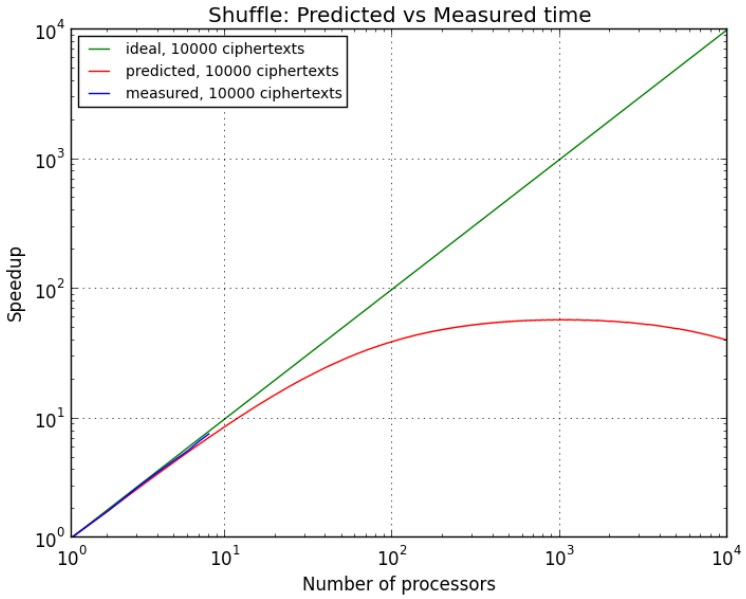

Figure 4.12: Shuffle: theoretical, measured and ideal speedup comparison

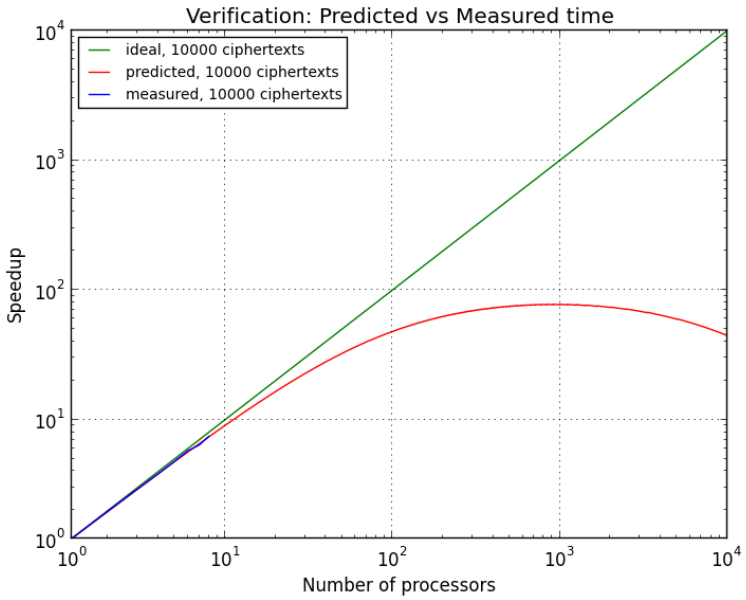

Figure 4.13: Verification: theoretical, measured and ideal speedup comparison

As anticipated, the predicted speedup curves reach a maximum value, around $P = 1000$ processors. The predicted and measured spee-dup are very close for both phases and this is mainly due to the small number of the available processors in our disposal. From the theoretical execution time we can see that the communication complexity affects the performance significantly as we add more and more processors. Later on, we will see the how the serial fraction affects this behavior as well. As mentioned before, until the point where the parallel and communication time rates of change become equal and opposite, the speedup keeps inceasing (with a decreasing rate of change) and beyond that point, communication dominates over parallel computation. As a result, the speedup begins to decrease. This property is better exposed in the following figures, where the curves of each part are shown individually:

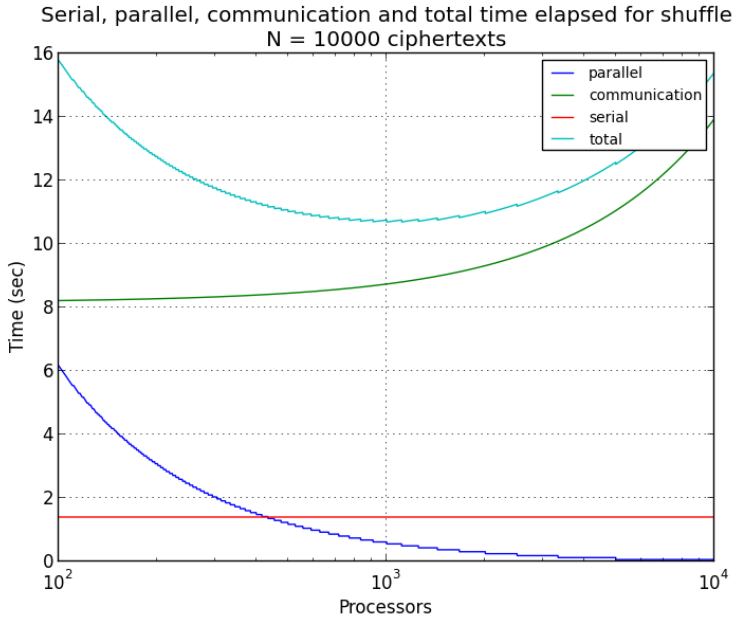

Figure 4.14: Shuffle: serial, parallel and communication times

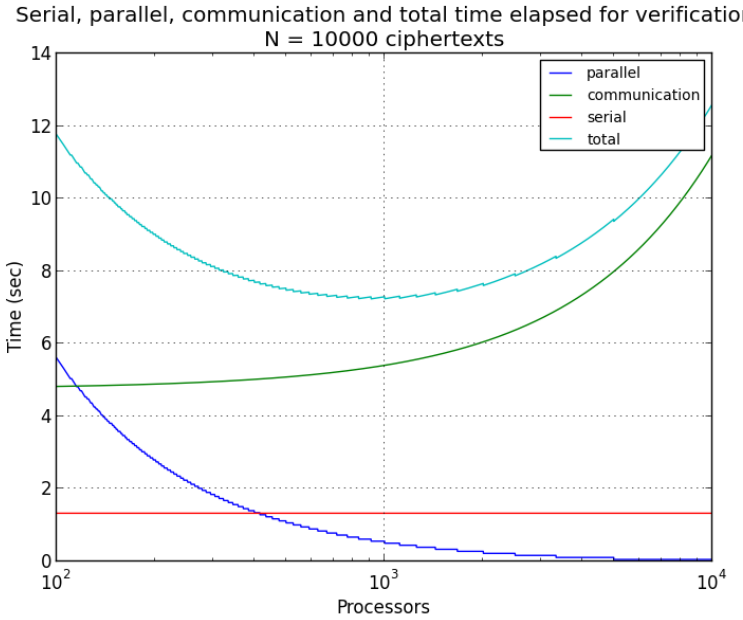

Figure 4.15: Verification: serial, parallel and communication times

Obviously, the serial part is constant, as we have the case of 10000 ciphertexts and the other two components are consistent with the previous explanation about the maximum value of the speedup.

Subsequently, it is very important to present the exact formula we used for the execution time prediction. The previous figures (speedup prediction) are our main guide in order to expose the boundaries of our problem and, thus, decide the maximum number of processors we can use to mix a certain number of ciphertexts. Because we are interested in the performance of the protocol for very large inputs (which we do not test), we use an upper bound to the formulas defining each component of the total execution time and they are proved more valuable for understanding of the problem. This also translates to a condition of $N >> P$, otherwise, there is no point in parallelizing the algorithm. So. we may write approximate expressions which describe the execution times and, therefore, the speedup of the implementation:

$$\text{time(shuffle)} \approx 11\frac{N}{P} \cdot E + N \cdot (18 \cdot D + a), N >> P.$$

$$\text{time(verification)} \approx 10\frac{N}{P} \cdot E + N \cdot (14 \cdot D + b), N >> P.$$

It is now clear that the communication cost and the serial fractions of the scheme will be the decisive factors that will not let the speedup scale for a large number of processors.

The corresponding expressions for the *serial* version of the protocol are:

$$\text{serial(shuffle)} = 11NE + 9 + a'N \approx (11 + a') \cdot N, N \text{sufficiently large.}$$

$$\text{serial(shuffle)} = 10NE + 9 + b'N \approx (10 + b') \cdot N, N \text{sufficiently large.}$$

As we can see, there is linearity between the input ciphertexts and the total (pure serial) time $(O(N))$. Some values calculated values for a, a', b, b' are:

	a	a'	b	b'
Shuffle	0.00014	0.0004	-	-
Verification	-	-	0.000127	0.000133

Table 4.2: Linear factors for the parallel and serial versions of the protocol

Next, the speedup takes the following form:

$$S_s = \frac{\text{serial(shuffle)}_N}{\text{time(shuffle)}_{N,P}} = \frac{(11 + a')N}{11\frac{N}{P} \cdot E + (18D + a) \cdot N}, N >> P.$$

$$S_s = \frac{\text{serial(verification)}_N}{\text{time(verification)}_{N,P}} = \frac{(10 + b')N}{10\frac{N}{P} \cdot E + (14D + b) \cdot N}, N >> P.$$

Now, in order to choose an appropriate number of processors for the parallel execution of the scheme, we have to take a closer look at the speedup diagrams presented before, especially the predicted curves. This choice is not only based on just mathematics, but is has many aspects we need to bear in mind (engineering criteria). For example, it was mentioned that the speedup reaches a maximum point at about 1000 processors. One could say that this is the number we are looking for. However, one factor that could affect our choice is the *bandwidth* that such number of machines need for communication. With respect to the several criteria, either from the scope of resource availability or the financial cost (to which we end up eventually), we may be either more flexible or more bounded as to the number that satisfies us. Simply put, we are as much satisfied as the cost we are willing to pay in order to achieve the corresponding speedup.

Hence, we must define the tradeoffs that exist and evaluate them with a model that fits our problem. From the prediction speedup figures, it can be seen that, gradually, we must add more and more processors to achieve a constant increase of speedup (e.g. 1 unit). Ideally, we would like one more processor to lead to the same amount of speedup. This cannot happen, though, as Amdahl's law indicates, since the implementation cannot be a 100% parallel program and the communication cost keeps rising with the addition of processors.

For this reason, we are dealing with a main tradeoff, whereby we must answer for a practical parallel mixing: the cost in number of processors per speedup unit versus speedup. Intuitively, if we look the (theoretical) figures, we can observe that that as more processors are added, the curve increases "linearly" until a "break" point occurs. So, it is reasonable to be interested in this particular area if we want a a minimal cost per speedup unit. Consequently, for a mix of 10000 ciphertexts, the "break" point seems to occur at around 30 proceesors and for the verification around 40. Thus, we have to choose a number close to these values so that we can obtain the maximum speedup, assuming we want 2 processors at most per speedup unit.

What is more, if bandwidth is a limitation, then we have the following:

$$Bandwidth = 2P \cdot \frac{1}{D} \frac{numbers}{sec} = 2P \cdot \frac{B}{D} \frac{bits}{sec}$$

where we must see when $Bandwith < K$, for a desired value K.

With our data, the desired bandwidth is about $\approx 3.8 \frac{Gbit}{sec}$, for $P = 32$ processors.

$$Bandwidth = 2P \cdot \frac{1}{D} \frac{numbers}{sec} = 2P \cdot \frac{B}{D} \frac{bits}{sec}$$

For our data, the desired bandwidth is between $\approx 1.2 \frac{Gbit}{sec}$ and $\approx 3 \frac{Gbit}{sec}$, for $P = 10$ and $P = 24$ respectively.

4.2.6 Suggesting maximum number of processors

At this point, we can construct a general model that allows us to determine an appropriate number of processors for a parallel execution of this protocol. Practically, the goal of this model is to answer to the following question:

"Assuming that the cost per speedup unit must not be more than M processors, how many processors should we choose to mix N ciphertexts?"

As it was outlined in the previous section, the prediction for the parallel and serial times and, therefore, for the speedup for a certain problem requires knowing some soecific parameters of the target environment. We sum up the parameters presented earlier:

- E sec/exponentiation
- D sec/number (transfer time to and from a processor)
- $A(N) = aN + b$: serial fraction of the parallel program (N ciphertexts)
- $T_s(N)$: (serial) execution time for N ciphertexts
- $T(P)$: parallel time for shuffle/verification accordingly

With the corresponding expressions for speedup:

$$S_{shuf}(N,P) = \frac{T_s}{T_{shuf}(P)} = \frac{T_s}{T_p + T_c + A_{shuf}} \Rightarrow$$

$$\boxed{S_{shuf}(N,P) \approx \frac{(11+a')N}{11\frac{N}{P} \cdot E + (18D+a) \cdot N}, N >> P.}$$

$$S_{ver}(N,P) = \frac{T_s}{T_{ver}(P)} \frac{T_s}{T_p + T_c + A_{ver}} \Rightarrow$$

$$\boxed{S_{ver}(N,P) \approx \frac{(10+b')N}{11\frac{N}{P} \cdot E + (14D+b) \cdot N}, N >> P.}$$

Thus, the answer to the previous question can be thought as follows:

"There exist P', P such that $S(P') - S(P) = 1$ and $\Delta P = P' - P = M$."

Alternatively, the answer is the solution to this equation:

$$\begin{aligned} S(P) &= \frac{T_s(N)}{T(P)} \Rightarrow \\ S(P') - S(P) &= 1 \Rightarrow \\ T(P) - T(P') - \frac{T(P)T(P')}{T_s} &= 0 \Rightarrow \\ T(P) - T(P+M) - \frac{T(P)T(P+M)}{T_s} &= 0 \end{aligned} \tag{4.1}$$

If there is a solution P_a in the above equation, it represents the *maximum* number of processors P, that satisfy the constraint of M processors per speedup unit. The fact that the solution constitutes

the maximum number, can be deduced from the figures in the previous section. The cost per speedup unit increases as we add more processors, beginning with a value greater than 1. Therefore, depending on M we choose, we can find where the speedup increase becomes exactly 1 and choose a number of processors between $2 \leq P \leq \text{FLOOR}(P_a) + M$, based on bandwidth criteria, if any. The FLOOR() operation is used because the root of the equation above may be a real number and any values higher than that will demand more than M processors, which is not acceptable.

As an example, consider a number of 10000 ciphertexts, as seen in the foregoing figures. If we require that no more than 2 processors per speedup unit are acceptable, then equation 4.1 yields 35 processors to perform the mix and 38 for the verification. This means that the cost per speedup unit is 2 up to 37 processors for the shuffle and 40 for the verification. This result confirms our previous estimation, where just by looking the diagrams we deduced that a fair choice would be 30-32 processors (shuffle) and this was an estimate according to the lowest possible value of cost we can ask for (there is no solution if we demand 1 processor per speedup unit).

As for the bandwidth, if we demand that it cannot exceed K bits/sec, then the highest value of P we can get is:

$$R = 2 \cdot P\frac{B}{D} \leq K \Rightarrow P \leq \frac{K \cdot D}{2B}$$

By applying the model for several values of the input ciphertexts and the cost per speedup unit, we can observe the behavior of the protocol eventually when $N >> P$, as well as the boundaries of both the speedup itself and the scaling. The parameters of our system is:

- $D = 0.000034\frac{sec}{number}$
- $E = 0.0056$ sec
- $A_{shuf}(N) = 0.00014N + 0.0056$ sec
- $A_{ver}(N) = 0.000133N$ sec

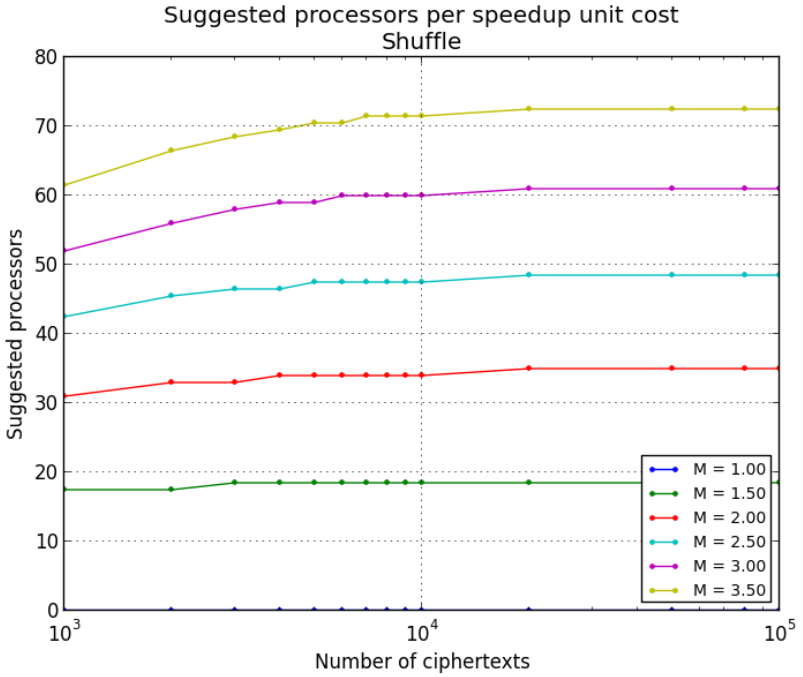

Figure 4.16: Shuffle: suggested number of processors per speedup unit cost

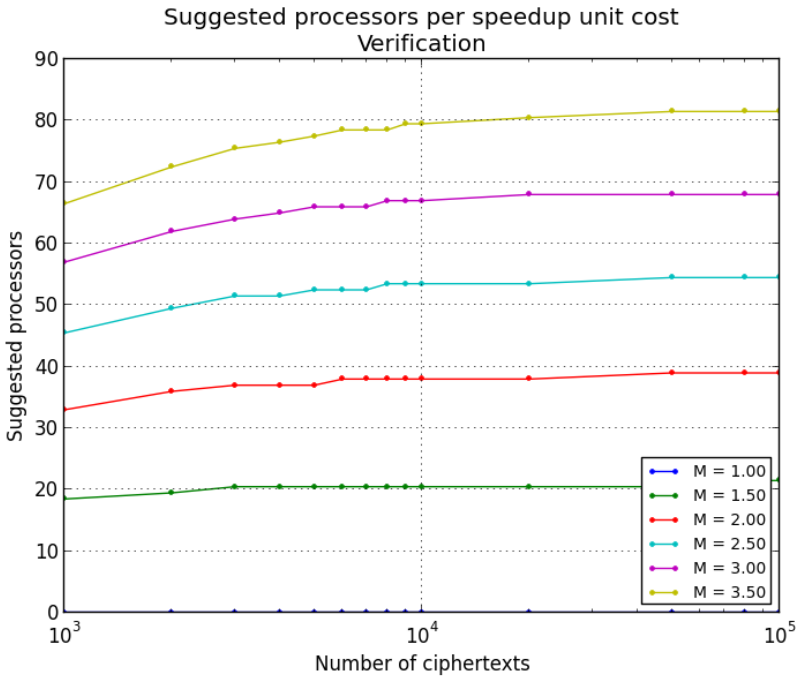

Figure 4.17: Verification: suggested number of processors per speedup unit cost

The figures above assume no bandwidth limitations and, thus, the maximum number of processors that satisfy the equation 4.1 is proposed for each case. Obviously, only the roots that are real and positive are acceptable.

In the end, by using our model and the predictions made about the execution time, our system is expected to mix 1000000 votes with 32 cores in about 45 minutes, producing around 7.5GB of on-disk text data as a proof.

Suggesting processors for distributed mixing scenarios

Since we have the experiment results for our program, which confirm the theoretical predictions, we can apply our model for scenarios where processors are distributed in a network. Until now, our measurements were taken on a *single node*, where the available cores communicate through memory, using a Queue. This yields a bit rate of around 7.8 MB/sec, which is quite slow. So, we consider two cases where, instead of a queue, the processors within a network communicate with 1 GB/sec and 10 GB/sec lines. Thus, a figure is given below,

which presents the processors that our model suggests, along with the corresponding speedup for each case. Also, a number of 1000000 ciphertexts is assumed as input as well as three values for the speedup unit cost ($M = 1.2, 1.5, 2$).

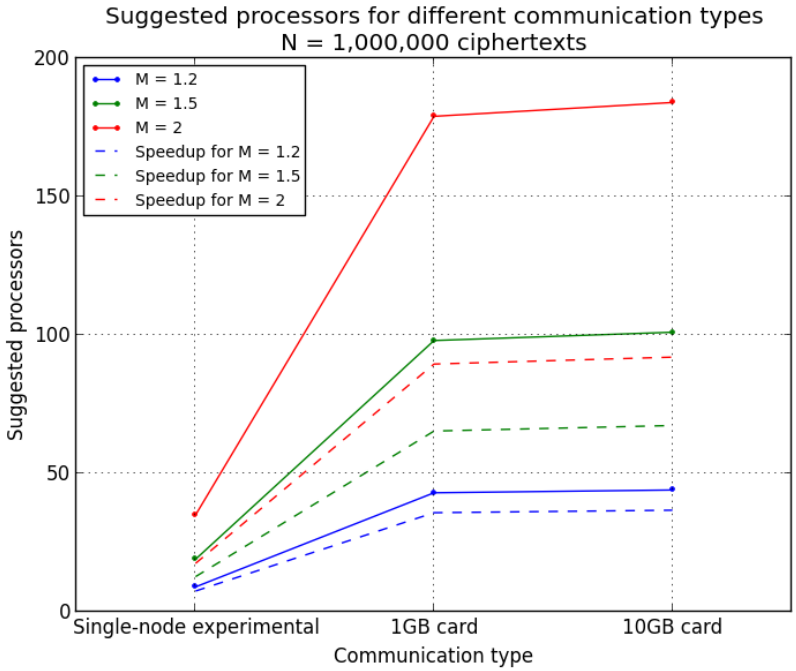

Figure 4.18: Model predictions for two cases of distributed mixing

We can see that as we step away from the case of a single node, the number of suggested processors is significantly increased (e.g. from 35 to around 179 for $M = 2$) for a line of 1 GB/sec. If we make the bit rate ten times higher (the second scenario), then the suggested number of processors is hardly increased (from 179 to 185 for $M = 2$). The reason is that we reach the limit set by the serial fraction of the algorithm (confirmation of Amdahl's law) and, therefore, we cannot achieve a higher speedup.

In addition, with this model, we can propose a combination of parameters for parallel mixing in a distributed network, apart from predicting its performance. For example, if we assume that 185 processors are available, as well as a 10 GB/sec line (eventually, there is no significant difference with the first case - 179 processors and 1 GB/sec line), then we can process up to 10,500,000 ciphertexts in an hour, approximately. On the other hand, if we consider the case of a single node, using 9 cores and a queue, then we can process up to 500,000 ciphertexts in an hour.

In conclusion, using our model for scenarios similar to the above, we can finally see the boundaries of our problem and, in that way, we are able to reach a good tradeoff.

Chapter 5

Conclusions

5.1 Synopsis

In this thesis, there was an extended study of mixnets, in order to find a better alternative to the current mixing stage of the Zeus e-voting system. Zeus has a very slow algorithm for mixing in its workflow, which is the main reason of a long election process. Thus, we analysed and evaluated, both in theory and in practice, Furukawa and Sako's scheme and we predicted its performance for a possible use by Zeus.

This scheme has the prospect of improving the mixing phase significantly and it has "convenient" properties (vector calculations) that we can exploit to achieve as much of speedup as we can. Therefore, we parallelized almost everything of its components and especially the exponentiations, which are the main reason for a slow mixing. However, we did not look for the ideal parallelization, leaving certain tasks out of the scope of parallelization, due to the Zeus architecture. The goal of this thesis is to define the tradeoffs between the speedup and the CPU power we need to achive it, so that we can find a practical number of CPUs for our problem, through a theoretical model. This model exposes the boundaries of our problem and it gives us a general notion about the expected performace of the corresponding parallel program.

Furukawa-Sako mixnet was a significant approach towards the notion of fast mixing and, as we saw, we can achieve a speedup of 37x (single node), which is a very positive outcome for the performance of the algorithm. In that way, we can mix up to 100,000 ciphertexts less than 15 minutes.

5.2 Future work

There are more approaches in implementing the current protocol, which can exploit the data dependencies and parallelism slightly different from one another. For example, we could try and not distribute batches of N exponents to the available processors, but, conceptually, we would consider the exponent-space as a unified sequence which can lead to better load distribution. This can be also applied to any other N-sized vector calculation.

Furthermore, the method of parallelization can be extended. In this thesis, we are dealing with a SIMD logic, where we distribute subsets of the same vectors to different processors. Instead, we could follow a MIMD logic, where entire tasks would be computed by different processors. This might be combined with the idea of a dynamic load balancing among the parallel tasks during runtime, using the corresponding mechanisms.

Finally, the current implementation was not tested on a distributed network of processors. However, we can predict such scenarios using the model proposed in tis thesis. It is interesting to test such approach and it will lead to more conclusions about the scalability and the acceleration of the mixing phase with the specific algorithm.

Bibliography

[1] Masayuki Abe: *Mix-networks on permutation networks*. In *Proceedings of the International Conference on the Theory and Applications of Cryptology and Information Security: Advances in Cryptology*, 1999.

[2] Ben Adida: *Helios: Web-based open-audit voting*. In *In Proceedings of the 17th USENIX Security Symposium (Security '08)*, 2008.

[3] Stephanie Bayer and Jens Groth: *Efficient zero-knowledge argument for correctness of a shuffle*. In *EUROCRYPT*, 2012.

[4] Josh Benaloh: *Rethinking voter coercion: The realities imposed by technology*. In *EVT/WOTE*, 2013.

[5] Dan Boneh and Philippe Golle: *Almost entirely correct mixing with applications to voting*. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, 2002.

[6] David Chaum: *Untraceable electronic mail, return addresses, and digital pseudonyms*. Commun. ACM, 1981.

[7] Yvo Desmedt and Kaoru Kurosawa: *How to break a practical mix and design a new one*, 2000.

[8] Ampos Fiat and Adi Shamir: *How to prove yourself: Practical solutions to identification and signature problems*. In *Advances in Cryptology — CRYPTO' 86*, 1987.

[9] Laure Fouard, Mathilde Duclos, and Pascal Lafourcade: *Survey on electronic voting schemes*. In *Survey on Electronic Voting Schemes*, 2007.

[10] Jun Furukawa: *Efficient and verifiable shuffling and shuffle-decryption*. IEICE Transactions, 2005.

[11] Jun Furukawa and Kazue Sako: *An efficient scheme for proving a shuffle*. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, 2001.

[12] Jens Groth: *A verifiable secret shuffle of homomorphic encryptions*. Journal of Cryptology, 2010.

[13] Markus Jakobsson: *A practical mix*. In *Advances in Cryptology—EUROCRYPT'98*. Springer Berlin Heidelberg, 1998.

[14] Markus Jakobsson and Ari Juels: *Millimix: Mixing in small batches*. Technical report, 1999.

[15] C. Andrew Neff: *Verifiable mixing (shuffling) of elgamal pairs*. http://people.csail.mit.edu/rivest/voting/papers/Neff-2004-04-21-ElGamalShuffles.pdf, 2004.

[16] Choonsik Park, Kazutomo Itoh, and Kaoru Kurosawa: *Efficient anonymous channel and all/nothing election scheme*. In *Advances in Cryptology — EUROCRYPT '93*, 1993.

[17] Jordi Puiggalí Allepuz and Sandra Guasch Castelló: *Universally verifiable efficient re-encryption mixnet*. In *Electronic Voting*, 2010.

[18] Kazue Sako and Joe Kilian: *Receipt-free mix-type voting scheme: A practical solution to the implementation of a voting booth*. In *Proceedings of the 14th Annual International Conference on Theory and Application of Cryptographic Techniques*, 1995.

[19] Björn Terelius Shahram Khazaei and Douglas Wikström: *Cryptanalysis of a universally verifiable efficient re-encryption mixnet*. In *Proceedings of the 2012 International Conference on Electronic Voting Technology/-Workshop on Trustworthy Elections*, 2012.

[20] Wikipedia: *Diffie-helman key exchange – Wikipedia, the free encyclopedia*. en.wikipedia.org/wiki/Diffie-Hellman_key_exchange.

[21] Wikipedia: *Non-interactive zero knowledge proof – Wikipedia, the free encyclopedia*. en.wikipedia.org/wiki/Non-interactive_zero-knowledge_proof.

[22] Wikipedia: *Zero knowledge proof – Wikipedia, the free encyclopedia*. en.wikipedia.org/wiki/Zero-knowledge_proof.