Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και
Υπολογιστών

Αποδείξεις Μηδενικής Γνώσης για Προβλήματα Τετραγωνικού Χρόνου

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΠΑΝΑΓΙΩΤΗΣ ΜΑΡΑΒΙΤΣΑΣ

Επιβλέπων : Αριστείδης Παγουρτζής
Καθηγητής Ε.Μ.Π.

Αθήνα, Νοέμβριος 2022

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και
Υπολογιστών

Αποδείξεις Μηδενικής Γνώσης για Προβλήματα Τετραγωνικού Χρόνου

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΠΑΝΑΓΙΩΤΗΣ ΜΑΡΑΒΙΤΣΑΣ

Επιβλέπων : Αριστείδης Παγουρτζής
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 22η Νοεμβρίου 2022.

.....	.....	.....
Αριστείδης Παγουρτζής Καθηγητής Ε.Μ.Π.	Δημήτριος Φωτάκης Καθηγητής Ε.Μ.Π.	Βασίλης Ζήκας Αναπληρωτής Καθηγητής Purdue University

Αθήνα, Νοέμβριος 2022

Παναγιώτης Μαραβίτσας

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Σε αυτή τη διπλωματική εργασία, κατασκευάζουμε μία νέα οικογένεια πολυωνύμων, την οποία ονομάζουμε $\mathcal{F}EIP$ τα οποία δεν μπορούν να υπολογιστούν σε πραγματικά υποτετραγωνικό χρόνο στη μέση περίπτωση εφόσον η Ισχυρή Υπόθεση Εκθετικού Χρόνου ($SETH$) ισχύει. Η δυσκολία μέσης περίπτωσης των πολυωνύμων μας βασίζεται στην ισχυρότερη υπόθεση πως το πρόβλημα ΑΚΡΙΒΕΣ ΕΣΩΤΕΡΙΚΟ ΓΙΝΟΜΕΝΟ (EIP_t) δεν μπορεί να λυθεί σε πραγματικά υποτετραγωνικό χρόνο στη χειρότερη περίπτωση. Το πρόβλημα EIP_t δέχεται σαν είσοδο δύο σύνολα διανυσμάτων U, V και ζητά την εύρεση δύο διανυσμάτων $u \in U$ και $v \in V$ τέτοια ώστε $\langle u, v \rangle = t$ για κάποιο ακέραιο $t > 0$. Το EIP_t είναι τουλάχιστον τόσο δύσκολο όσο το Πρόβλημα των Ορθογώνιων Διανυσμάτων (OV), αλλά δεν γνωρίζουμε αν είναι ισοδύναμα στην περίπτωση που για τη διάσταση d των διανυσμάτων ισχύει $d = \omega(logn)$, που είναι μία συνηθισμένη υπόθεση. Αυτό καθιστά την $\mathcal{F}EIP$ μία πιο υποσχόμενη οικογένεια δύσκολων συναρτήσεων στη μέση περίπτωση από την $\mathcal{F}OV$ η οποία ορίζεται με παρόμοιο τρόπο στο [Ball17a]. Παρουσιάζουμε μία λεπτομερή αναγωγή από το OV στο EIP_t, μία λεπτομερή αναγωγή από το EIP_t στη $\mathcal{F}EIP$ και μία λεπτομερή αναγωγή χειρότερης-προς-μέσης περίπτωσης από τη $\mathcal{F}EIP$ στον εαυτό της. Στη συνέχεια κατασκευάζουμε ένα MA πρωτόκολλο για την $\mathcal{F}EIP$ με έναν πραγματικά υποτετραγωνικό επαληθευτή και το χρησιμοποιούμε μαζί με την απόδειξη δυσκολίας υπολογισμού της $\mathcal{F}EIP$ για να κατασκευάσουμε μία λεπτομερή Απόδειξη Εργασίας. Για το υπόλοιπο της διπλωματικής αυτής εργασίας, μελετάμε την έννοια της μηδενικής γνώσης στο περιβάλλον της λεπτομερούς κρυπτογραφίας. Αρχικά ορίζουμε τις αποδείξεις μηδενικής γνώσης με τέτοιο τρόπο ώστε να να μπορούν να χρησιμοποιηθούν για προβλήματα επιλύσιμα σε πολυωνυμικό χρόνο με κοινώς αποδεκτά κάτω φράγματα. Στη συνέχεια κατασκευάζουμε αποδείξεις μηδενικής γνώσης για τα κυριότερα προβλήματα της λεπτομερούς πολυπλοκότητας, το OV και το $3SUM$ τα οποία είναι εύκολα επιλύσιμα σε τετραγωνικό χρόνο. Για τις αποδείξεις αυτές θεωρούμε ότι ο τετραγωνικός χρόνος είναι υπολογιστικά απρόσιτος, οπότε απαιτούμε τόσο ο αποδείκτης όσο και ο επαληθευτής να είναι αλγόριθμοι πραγματικά υποτετραγωνικού χρόνου. Τέλος δείχνουμε πώς τα πρωτόκολλά μας μπορούν να χρησιμοποιηθούν για να κατασκευάσουμε νέα πρωτόκολλα μηδενικής γνώσης για περισσότερα προβλήματα τετραγωνικού χρόνου μέσω λεπτομερών αναγωγών. Σαν παράδειγμα, χρησιμοποιούμε το πρόβλημα $GeomBase$ από την υπολογιστική γεωμετρία.

Λέξεις κλειδιά

Λεπτομερής Κρυπτογραφία, Λεπτομερής Πολυπλοκότητα, Δυσκολία Μέσης Περίπτωσης, Αποδείξεις Μηδενικής Γνώσης, Ορθογώνια Διανύσματα, 3SUM

Abstract

In this diploma thesis, we construct a new family of polynomials, which we call $\mathcal{F}EIP$, that is hard to compute in truly subquadratic time on average assuming $SETH$ holds. In fact, the average case hardness of our polynomials is based on a stronger assumption, which is the worst case hardness of the problem EXACT INNER PRODUCT (EIP_t). EIP_t asks, given two sets of vectors U, V, to find vectors $u \in U$ and $v \in V$ such that $\langle u, v \rangle = t$ for some target integer $t > 0$. EIP_t is at least as hard as OV, but it is not known whether they are equivalent in the dense setting, where $d = \omega(logn)$, making $\mathcal{F}EIP$ a more promising candidate for an average case hard family of functions than the similarly constructed $\mathcal{F}OV$, which is presented in [Ball17a]. We provide a fine-grained reduction from OV to EIP_t, a fine-grained reduction from EIP_t to $\mathcal{F}EIP$ and a worst-to-average case fine-grained reduction from $\mathcal{F}EIP$ to itself. Then, we construct an MA protocol for $\mathcal{F}EIP$ with a truly subquadratic time verifier and use it along with our hardness results for $\mathcal{F}EIP$ to construct a fine-grained Proof of Work scheme. For the rest of the thesis, we study the notion of zero knowledge in the setting of fine-grained cryptography. We give our definition of fine-grained zero knowledge arguments for problems which have conjectured polynomial lower bounds and then construct zero knowledge protocols for the problems OV and $3SUM$, both of which are easily solvable in quadratic time. In these protocols, we consider running times of $\Omega(n^2)$ to be intractable, thus we require the prover and the verifier to be truly subquadratic time algorithms. Finally we demonstrate how we can use these protocols to efficiently obtain new generic zero knowledge arguments for a wider variety of quadratic time problems through fine-grained reductions. As such an example, we use the problem $GeomBase$ from computational geometry.

Key words

Fine-Grained Complexity, Fine-Grained Cryptography, Average Case Hardness, Zero Knowledge Arguments, Orthogonal Vectors, 3-SUM

Ευχαριστίες

Με την ολοκλήρωση της παρούσας διπλωματικής εργασίας, ο κύκλος των προπτυχιακών μου σπουδών φτάνει στο τέλος του. Στο σημείο αυτό θα ήθελα να αποδώσω ευχαριστίες στα άτομα τα οποία με βοήθησαν και με στήριξαν κατά τη διάρκεια αυτής της πορείας. Αρχικά, θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή αυτής της διπλωματικής εργασίας, κ. Παγουρτζή, για την καθοδήγησή του και για όλες τις επικοδομητικές συζητήσεις που είχαμε καθ' όλη τη διάρκεια της συγγραφής. Ακόμα, θα ήθελα να ευχαριστήσω τον κ. Φωτάκη για όλη την αλγοριθμική γνώση που μου προσέφερε απλόχερα μέσω των μαθημάτων του, καθώς και τον κ. Ζήκα για τη συμμετοχή του στην τριμελή επιτροπή. Επιπλέον, θα ήθελα να ευχαριστήσω όλους τους φίλους μου, τόσο τους παλιότερους, όσο και αυτούς που γνώρισα στη σχολή, χωρίς τους οποίος τα φοιτητικά μου χρόνια δεν θα ήταν τα ίδια. Τέλος, οφείλω ένα μεγάλο ευχαριστώ στους γονείς μου για την ανεξάντλητη αγάπη τους και για όλη τη στήριξη που μου προσέφεραν, και εξακολουθούν να μου προσφέρουν, όλα αυτά τα χρόνια.

Παναγιώτης Μαραβίτσας,

Αθήνα, 22η Νοεμβρίου 2022

Η εργασία αυτή είναι επίσης διαθέσιμη ως Τεχνική Αναφορά CSD-SW-TR-42-17, Εθνικό Μετσόβιο Πολυτεχνείο, Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών, Εργαστήριο Τεχνολογίας Λογισμικού, Νοέμβριος 2022.

URL: `http://www.softlab.ntua.gr/techrep/`
FTP: `ftp://ftp.softlab.ntua.gr/pub/techrep/`

Contents

Κεφάλαιο 0

Εκτεταμένη Ελληνική Περίληψη

0.1 Εισαγωγή

Η λεπτομερής κρυπτογραφία είναι ένας κλάδος της κρυπτογραφίας που έχει τα εξής χαρακτηριστικά [Degw16]:

- Η ασφάλεια επιτυγχάνεται μόνο ενάντια σε μία κλάση αντιπάλων οι οποίοι διαθέτουν πολυωνυμικά περιορισμένους πόρους.
- Οι τίμιοι αλγόριθμοι χρησιμοποιούν λιγότερους πόρους από τους αντιπάλους που προσπαθούν να ξεγελάσουν.

Η χρησιμότητα μιας τέτοιας μορφής κρυπτογραφίας έγκειται στο γεγονός ότι είναι ευκολότερο να στηρίξουμε την ασφάλεια σε υποθέσεις δυσκολίας χειρότερης περίπτωση, καθώς και στο γεγονός ότι μπορεί να χρησιμοποιηθεί ακόμα και κάτω από την υπόθεση ότι όλα τα προβλήματα του NP επιδέχονται πιθανοτικούς πολυωνυμικούς αλγορίθμους.

Όπως είναι φυσικό, προκειμένου να πετύχουμε λεπτομερή κρυπτογραφία, χρειαζόμαστε κάποιες υποθέσεις δυσκολίας εντός της κλάσης P. Ο κλάδος της λεπτομερούς πολυπλοκότητας, ο οποίος ασχολείται με τη δυσκολία των προβλημάτων εντός της κλάσης P, μας έχει εφοδιάσει με πολλές κοινώς αποδεκτές εικασίες που χρησιμοποιούνται για τον σκοπό αυτό. Μερικές από τις σημαντικότερες εικασίες είναι η Ισχυρή Υπόθεση Εκθετικού Χρόνου ($SETH$) και η Υπόθεση των Ορθογώνιων Διανυσμάτων (OVC).

Τα τελευταία χρόνια, όλο και περισσότερες ιδέες της συμβατικής κρυπτογραφίας μεταφέρονται στο περιβάλλον της λεπτομερούς κρυπτογραφίας. Μερικά χαρακτηριστικά παραδείγματα είναι η λεπτομερής δυσκολία μέσης περίπτωσης, για την οποία υπάρχει πλήθος ενδιαφέροντων αποτελεσμάτων [Ball17a, Dali20, Gold20], οι λεπτομερείς αποδείξεις εργασίας [Ball17b, Ball18] και η ύπαρξη λεπτομερών μονόδρομων συναρτήσεων [Brzu22].

0.2 Συνεισφορά

Στην παρούσα διπλωματική εργασία κάνουμε κάποια πρόοδο στον τομέα της λεπτομερούς κρυπτογραφίας, συγκεκριμένα στη λεπτομερή δυσκολία μέσης περίπτωσης και στη μηδενική γνώση.

Αρχικά, κατασκευάζουμε μία οικογένεια συναρτήσεων, που ονομάζουμε $\mathcal{FEIP}$, οι οποίες είναι δύσκολο να υπολογιστούν σε πραγματικά υποτετραγωνικό χρόνο στη μέση περίπτωση. Η δυσκολία υπολογισμού των συναρτήσεών μας, στηρίζεται στη δυσκολία επίλυσης του προβλήματος ΑΚΡΙΒΕΣ ΕΣΩΤΕΡΙΚΟ ΓΙΝΟΜΕΝΟ (EIP_t) στη χειρότερη περίπτωση. Το πρόβλημα αυτό δέχεται σαν είσοδο δύο σύνολα U, V τα οποία περιέχουν από n δυαδικά διανύσματα το καθένα και ζητά να βρεθούν δύο διανύσματα (ένα από το κάθε σύνολο) που έχουν εσωτερικό γινόμενο ίσο με κάποιο δεδομένο ακέραιο t. Χρησιμοποιούμε μία λεπτομερή αναγωγή από το πρόβλημα των Ορθογώνιων Διανυσμάτων για να δείξουμε ότι το πρόβλημα αυτό δεν λύνεται σε πραγματικά υποτετραγωνικό χρόνο στη χειρότερη περίπτωση. Στη συνέχεια, κατασκευάζουμε μία λεπτομερή απόδειξη εργασίας που στηρίζεται στην οικογένεια συναρτήσεων $\mathcal{FEIP}$. Στόχος

αυτής της απόδειξης εργασίας είναι ένας αποδείκτης να πείσει έναν επαληθευτή υποτετραγωνικού χρόνου ότι έχει εκτελέσει κάποια εργασία που απαιτεί τετραγωνικό χρόνο.

Στη συνέχεια ασχολούμαστε με την υπάρξη αποδείξεων μηδενικής γνώσης στο πλαίσιο της λεπτομερούς κρυπτογραφίας. Συμβατικά, οι αποδείξεις μηδενικής γνώσης ορίζονται για προβλήματα για τα οποία δεν υπάρχει αλγόριθμος πολυωνυμικού χρόνου. Εμείς ορίζουμε τις λεπτομερείς αποδείξεις μηδενικής γνώσης οι οποίες μπορούν να δοθούν για προβλήματα που είναι επιλύσιμα σε πολυωνυμικό χρόνο. Στη συνέχεια κατασκευάζουμε λεπτομερείς αποδείξεις μηδενικής γνώσης για τα κυριότερα προβλήματα της λεπτομερούς πολυπλοκότητας: το πρόβλημα Ορθογώνιων Διανυσμάτων και το $3SUM$. Τα προβλήματα αυτά επιλύονται εύκολα σε τετραγωνικό χρόνο, όμως θεωρούμε πως δεν επιδέχονται αλγορίθμους πραγματικά υποτετραγωνικού χρόνου στη χειρότερη περίπτωση. Τόσο ο αποδείκτης, όσο και ο επαληθευτής στις αποδείξεις μας χρειάζονται χρόνο $\tilde{O}(n)$. Τέλος, δείχνουμε ότι μπορούμε να χρησιμοποιήσουμε τις αποδείξεις αυτές για να πάρουμε εύκολα αποδείξεις μηδενικής γνώσης και για πολλά άλλα προβλήματα μέσω λεπτομερών αναγωγών. Σαν παράδειγμα χρησιμοποιούμε το πρόβλημα της υπολογιστικής γεωμετρίας $GeomBase$ που δέχεται σαν είσοδο σημεία του επιπέδου που βρίσκονται πάνω σε 3 ευθείες και ζητα την εύρεση τριών σημείων (ένα από κάθε ευθεία) τα οποία είναι συνευθειακά.

0.3 Προαπαιτούμενα

0.3.1 Αποδείξεις Μηδενικής Γνώσης

Οι αποδείξεις μηδενικής γνώσης είναι ένα θεμελιώδες εργαλείο της κρυπτογραφίας που παρουσιάστηκαν για πρώτη φορά στο [Gold89]. Επιτρέπουν σε έναν αποδείκτη να πείσει έναν επαληθευτή πως μία μαθηματική πρόταση είναι αληθής χωρίς να αποκαλύψει επιπλέον πληροφορίες πέρα από την ορθότητά της. Πριν ορίσουμε τις αποδείξεις μηδενικής γνώσης, χρειάζεται να περιγράψουμε τι είναι ένα διαδραστικό πρωτόκολλο.

Έστω $\mathbb{F}$ ένα πεπερασμένο σώμα και $\mathcal{R} : \mathbb{F}^n \times \mathbb{F}^m \to \{0,1\}$ μία δυαδική σχέση, αποκρίσιμη σε πολυωνυμικό χρόνο. Ένα διαδραστικό πρωτόκολλο IP k μηνυμάτων για τη σχέση $\mathcal{R}$ αποτελείται από έναν πιθανοτικό αλγόριθμο επαληθευτή $\mathcal{V}$ πολυωνυμικού χρόνου και έναν αλγόριθμο αποδεικτη $\mathcal{P}$. Τόσο ο $\mathcal{V}$ όσο και ο $\mathcal{P}$ δέχονται κοινή είσοδο $x \in \mathbb{F}^n$ και επιπλέον ο $\mathcal{P}$ δέχεται είσοδο $w \in \mathbb{F}^m$, τέτοια ώστε $\mathcal{R}(x,w) = 1$. Λέμε ότι το x είναι η προς απόδειξη πρόταση και w είναι ο αντίστοιχος μάρτυρας. Στη συνέχεια ο $\mathcal{P}$ και ο $\mathcal{V}$ ανταλάσσουν εναλλάξ μία σειρά μηνυμάτων $m_1, \ldots, m_k$. Τόσο για τον $\mathcal{P}$ όσο και για τον $\mathcal{V}$, το επόμενο μήνυμα υπολογίζεται ως μία συνάρτηση του x και όλων των προηγούμενων μηνυμάτων. Μετά από k γύρους, ο αλγόριθμος επαληθευτής επιστρέψει ένα bit $b \in \{0,1\}$. Αν $b = 0$, λέμε ότι ο επαληθευτής απορρίπτει αλλιώς αποδέχεται την ορθότητα της πρότασης. Τα μηνύματα $(m_1, \ldots, m_k)$ μαζί με το bit b αποτελούν το αντίγραφο του πρωτοκόλλου. Συμβολίζουμε με $(\mathcal{P}(x,w), \mathcal{V}(x)) \in \{0,1\}$ την έξοδο του επαληθευτή $\mathcal{V}$ με είσοδο x όταν αλληλεπιδρά με μία στρατηγική αποδείκτη $\mathcal{P}$ (που διαθέτει μάρτυρα w σαν είσοδο).

Ορισμός 0.3.1. Μία απόδειξη μηδενικής γνώσης για τη σχέση $\mathcal{R}$ με σφάλμα πληρότητας ϵ_c και σφάλμα ορθότητας ϵ_s είναι ένα διαδραστικό πρωτόκολλο μεταξύ ενός αλγορίθμου αποδείκτη $\mathcal{P}$ και ενός αλγορίθμου επαληθευτή $\mathcal{V}$ που ικανοποιεί τις παρακάτω ιδιότητες:

- Πληρότητα: Αν $(x,w) \in \mathcal{R}$ και ο $\mathcal{P}$ γνωρίζει τον μάρτυρα w για το x, θα πείσει τον επαληθευτή με πιθανότητα $1 - \epsilon_c$.

$$Pr[(\mathcal{P}(x,w), \mathcal{V}(x)) = 1] \geq 1 - \epsilon_c$$

- Ορθότητα: Αν ένας πιθανοτικός αλγόριθμος πολυωνυμικού χρόνου $\tilde{\mathcal{P}}$ δεν διαθέτει έναν μάρτυρα w για το x, δεν μπορεί να πείσει τον $\mathcal{V}$ παρά με πιθανότητα ϵ_s.

$$Pr[(\tilde{\mathcal{P}}(x), \mathcal{V}(x)) = 1] \leq \epsilon_s$$

- Μηδενική Γνώση: Για κάθε πιθανοτικό αλγόριθμο πολυωνυμικού χρόνου $\tilde{\mathcal{V}}$, υπάρχει πιθανοτικός αλγόριθμος προσομοιωτής $\mathcal{S}$ πολυωνυμικού χρόνου που, με είσοδο την πρόταση προς απόδειξη x και με πρόσβαση στον αλγόριθμο $\tilde{\mathcal{V}}$ ως "μαύρο κουτί" με επαναφορά, επιστρέφει ένα προσομοιωμένο αντίγραφο πρωτοκόλλου έτσι ώστε

$$\mathcal{S}(x, \tilde{\mathcal{V}}(\cdot)) \equiv View(\mathcal{P}(x, w), \tilde{\mathcal{V}}(x))$$

Τα πρωτόκολλά μας ικανοποιούν μία πιο αδύναμη μορφή μηδενικής γνώσης που ονομάζεται υπολογιστική μηδενική γνώση. Για την ιδιότητα αυτή απαιτούμε το προσομοιωμένο αντίγραφο πρωτοκόλλου να είναι (t, ζ)-μη διακρίσιμο από ένα πραγματικό αντίγραφο πρωτοκόλλου.

Επιπλέον τα πρωτόκολλά μας ικανοποιούν μία ισχυρότερη μορφή ορθότητας που ονομάζεται ειδική ορθότητα. Αυτό σημαίνει ότι υπάρχει αποδοτικός αλγόριθμος που, με είσοδο μερικά αντίγραφα του πρωτοκόλλου με την ίδια πρώτη δέσμευση από τον αποδείκτη αλλά διαφορετικές προκλήσεις από τον επαληθευτή, δίνει σαν έξοδο έναν έγκυρο μάρτυρα για την πρόταση προς απόδειξη x. Ένας τέτοιος αλγόριθμος ονομάζεται εξαγωγέας γνώσης. Παραθέτουμε τον ορισμό της ειδικής ορθότητας:

Ορισμός 0.3.2. Έστω ϵ_s το σφάλμα ορθότητας της απόδειξης μηδενικής γνώσης. Αν υπάρχει πιθανοτικός αλγόριθμος πολυωνυμικού χρόνου $\tilde{\mathcal{P}}$ που δεν διαθέτει κάποιο μάρτυρα w για το x, ο οποίος πείθει τον επαληθευτή $\mathcal{V}$ με πιθανότητα $\tilde{\epsilon} > \epsilon_s$:

$$\tilde{\epsilon} := Pr[(\tilde{\mathcal{P}}(x), \mathcal{V}(x)) = 1] > \epsilon_s,$$

τότε υπάρχει ένας αλγόριθμος εξαγωγέας $\mathcal{E}$ ο οποίος, διαθέτοντας πρόσβαση "μαύρου κουτιού" με επαναφορά στον $\tilde{\mathcal{P}}$, δίνει σαν έξοδο έναν μάρτυρα w' για το x σε χρόνο $poly(n, (\tilde{\epsilon} - \epsilon)^{-1})$ με πιθανότητα τουλάχιστον $\frac{1}{2}$.

Τέλος, δίνουμε τον (πιο αδύναμο) ορισμό της μηδενικής γνώσης με τίμιο επαληθευτή που χρησιμοποιούμε στα πρωτόκολλά μας.

Ορισμός 0.3.3. Υπάρχει ένας πιθανοτικός πολυωνυμικός αλγόριθμος προσομοιωτής $\mathcal{S}$ που, με είσοδο την πρόταση προς απόδειξη x, δίνει σαν έξοδο ένα προσομοιωμένο αντίγραφο πρωτοκόλλου που είναι (t, ζ)-μη διακρίσιμο από ένα πραγματικό αντίγραφο με τίμιο επαληθευτή $View(\mathcal{P}(x, w), V(x))$.

0.3.2 MPC-in-the-head

Η κύρια τεχνική των πρωτοκόλλων μας ονομάζεται MPC-in-the-head και εισήχθησε για πρώτη φορά στο [Isha09]. Είναι μία τεχνική που μας επιτρέπει να δημιουργούμε αποδείξεις μηδενικής γνώσης από ασφαλή πρωτόκολλα υπολογισμού μεταξύ πολλών συμμετοχόντων (MPC). Θα περιγράψουμε τώρα τα βήματα με τα οποία μπορούμε να μετατρέψουμε ένα ασφαλές πρωτόκολλο υπολογισμού μεταξύ πολλών συμμετοχόντων σε μία απόδειξη μηδενικής γνώσης για μία σχέση $\mathcal{R}(x, w)$ που αναγνωρίζει μία NP γλώσσα $\mathcal{L}$.

Έστω Π ένα ασφαλές πρωτόκολλο υπολογισμού μεταξύ N συμμετοχόντων και P το σύνολο των συμμετοχόντων. Συμβολίζουμε με $View_i$ την εικόνα που έχει το P_i κατά την εκτέλεση του Π. Οι συμμετέχοντες και ο αποδείκτης παίρνουν σαν είσοδο μία πρόταση προς απόδειξη $x \in \mathcal{L}$ και επιπλέον ο αποδείκτης παίρνει σαν είσοδο έναν μάρτυρα w για το x. Τα βήματα του πρωτοκόλλου είναι τα ακόλουθα:

- **Γύρος 1:** Ο αποδείκτης χρησιμοποιεί ένα προσθετικό μοίρασμα για να δημιουργήσει N το πλήθος μερίδια του w και τα μοιράζει μεταξύ N εικονικών συμμετοχόντων. Στη συνέχεια προσομοιώνει μία εκτέλεση του πρωτοκόλλου Π "στο μυαλό του" για τη συνάρτηση $\mathcal{F}([\![w]\!]) = \mathcal{R}(x, \oplus_{i \in [N]} [\![w]\!]_i)$. Ο αποδείκτης κάνει χρήση ενός ασφαλούς σχήματος δέσμευσης Com για να υπολογίσει τη δέσμευση com_i στην εικόνα $View_i$ του κάθε συμμετέχοντος P_i. Στη συνέχεια στέλνει τη δέσμευση $com_i := Com(View_i; r_i)$ στον επαληθευτή για κάθε $i \in [N]$.

- **Γύρος 2:** Υποθέτουμε ότι το πρωτόκολλο Π διαθέτει t-ιδιωτικότητα. Ο επαληθευτής επιλέγει τυχαία ένα σύνολο δεικτών $\mathcal{I} \subset [N]$ τέτοιο ώστε $|\mathcal{I}| \leq t$ και το στέλνει στον αποδείκτη.

- **Γύρος 3:** Ο αποδείκτης φανερώνει όλες τις δεσμεύσεις στις εικόνες των συμμετοχόντων που ανήκουν στο $\mathcal{I}$ και τις στέλνει στον επαληθευτή.

Στη συνέχεια ο επαληθευτής ελέγχει ότι οι φανερωμένες δεσμεύσεις είναι συνεπείς μεταξύ τους και κανένας συμμετέχοντας δεν απορρίπτει. Αν ο έλεγχος είναι επιτυχής, ο επαληθευτής αποδέχεται, αλλιώς απορρίπτει.

0.3.3 Λεπτομερής Πολυπλοκότητα

Η λεπτομερής πολυπλοκότητα [Vass15] είναι κλάδος της θεωρίας πολυπλοκότητας που μελετάει πολυωνυμικά κάτω φράγματα προβλημάτων για τα οποία δεν έχει υπάρξει καμία σημαντική βελτίωση στην πολυπλοκότητα επίλυσής τους. Στη συνέχεια θα ορίσουμε τις λεπτομερείς αναγωγές, οι οποίες σε συνδυασμό με ορισμένες εικασίες αποτελούν ισχυρές και κοινώς αποδεκτές ενδείξεις πως η πολυπλοκότητα μερικών προβλημάτων δεν μπορεί να βελτιωθεί σημαντικά περαιτέρω.

Ορισμός 0.3.4 (Λεπτομερής Αναγωγή). Έστω $a(n), b(n)$ δύο μη φθίνουσες συναρτήσεις του n. Το πρόβλημα A είναι (a, b)-αναγώγιμο στο πρόβλημα B, και συμβολίζεται με $A \leq_{a,b}$, αν για κάθε $\epsilon > 0$, υπάρχει $\delta > 0$, ένας αλγόριθμος F με πρόσβαση σε ένα μαντείο B, μία σταθερά d, και για κάθε $n \geq 1$ ένας ακέραιος $k(n)$, έτσι ώστε για κάθε n ο αλγόριθμος F δέχεται σαν είσοδο ένα στιγμιότυπο του A και:

- Ο αλγόριθμος F χρειάζεται χρόνο το πολύ $d \cdot (a(n))^{1-\delta}$.

- Ο αλγόριθμος F παράγει το πολύ $k(n)$ στιγμιότυπα του B προσαρμοστικά. Συγκεκριμένα, το j-οστό στιγμιότυπο B_j είναι μία συνάρτηση των $\{(B_i, a_i)\}_{1 \leq i < j}$, όπου το B_i είναι το j-οστό στιγμιότυπο που έχει παραχθεί και το a_i είναι η απάντηση του μαντείου για το B στο στιγμιότυπο B_i.

- Τα μεγέθη n_i των στιγμιοτύπων B_i για οποιαδήποτε επιλογή των απαντήσεων του μαντείου a_i ικανοποιούν την ανισότητα $\sum_{i=1}^{k(n)} (b(n))^{1-\epsilon} \leq d \cdot (a(n))^{1-\delta}$.

Άμεση συνέπεια αυτού του ορισμού είναι πως οποιαδήποτε βελτίωση του χρόνου $b(n)$ που απαιτείται για το B συνεπάγεται βελτίωση του χρόνου $a(n)$ για το A.

Θα παρουσιάσουμε τώρα τις τέσσερις κύριες εικασίες της λεπτομερούς πολυπλοκότητας από τις οποίες πηγάζουν όλα τα πολυωνυμικά κάτω φράγματα.

Εικασία 1 (Ισχυρή Υπόθεση Εκθετικού Χρόνου). Για κάθε $\epsilon > 0$, υπάρχει ένας ακέραιος αριθμός k, τέτοιος ώστε το SAT σε προτάσεις n μεταβλητών που βρίσκονται σε k-CNF μορφή δεν μπορεί να λυθεί σε εκτιμώμενο χρόνο $O(2^{(1-\epsilon)n} poly(n))$.

Η επόμενη εικασία βασίζεται στο πρόβλημα των ορθογώνιων διανυσμάτων που είναι το ακόλουθο:

Ορισμός 0.3.5 (Πρόβλημα των Ορθογώνιων Διανυσμάτων). Έστω U, V δύο σύνολα διανυσμάτων στο $\{0,1\}^d$ έτσι ώστε $|U| = |V| = n$ και $d = \omega(logn)$. Το πρόβλημα των ορθογώνιων διανυσμάτων ζητά την εύρεση δύο διανυσμάτων $u \in U$ και $v \in V$ τέτοια ώστε $\langle u, v \rangle = 0$.

Εικασία 2 (Εικασία Ορθογώνιων Διανυσμάτων). Οποιοσδήποτε αλγόριθμος χρειάζεται εκτιμώμενο χρόνο $n^{2-o(1)}$ για να λύσει το πρόβλημα των ορθογώνιων διανυσμάτων.

Η τρίτη εικασία βασίζεται στο πρόβλημα $3SUM$:

Ορισμός 0.3.6 (3SUM). Έστω U, V, W τρία σύνολα ακεραίων τέτοια ώστε $|U| = |V| = |W| = n$ και $U, V, W \subset \{-m, \ldots, m\}$ για κάποιο $m = poly(n)$. Το πρόβλημα $3SUM$ ζητά την εύρεση τριών στοιχείων u, v, w τέτοια ώστε $u \in U$, $v \in V$, $w \in W$ και $u + v + w = 0$.

Εικασία 3 (Αδυναμία επίλυσης του 3SUM σε πραγματικά υποτετραγωνικό χρόνο). Οποιοσδήποτε αλγόριθμος χρειάζεται εκτιμώμενο χρόνο $n^{2-o(1)}$ για να λύσει το πρόβλημα $3SUM$.

Η τελευταία εικασία στηρίζεται στο πρόβλημα $APSP$:

Ορισμός 0.3.7 (APSP). Έστω G ένας κατευθυνόμενος (η μη) γράφος με ακέραια βάρη ακμών. Το πρόβλημα $APSP$ ζητά την εύρεση των αποστάσεων μεταξύ όλων των ζευγαριών κορυφών του G.

Εικασία 4 (Αδυναμία επίλυσης του APSP σε πραγματικά υποκυβικό χρόνο). Οποιοσδήποτε αλγόριθμος χρειάζεται εκτιμώμενο χρόνο $n^{3-o(1)}$ για να λύσει το πρόβλημα $APSP$.

0.4 Λεπτομερής Δυσκολία Μέσης Περίπτωσης

Σε αυτή την ενότητα, παρουσιάζουμε μία οικογένεια πολυωνύμων, που ονομάζουμε $\mathcal{F}EIP$ τα οποία δεν μπορούν να υπολογιστούν σε πραγματικά υποτετραγωνικό χρόνο στη μέση περίπτωση εκτός αν το πρόβλημα ΑΚΡΙΒΕΣ ΕΣΩΤΕΡΙΚΟ ΓΙΝΟΜΕΝΟ (EIP_t) το οποίο θα ορίσουμε τώρα μπορεί να λυθεί σε πραγματικά υποτετραγωνικό χρόνο στη χειρότερη περίπτωση.

Ορισμός 0.4.1 (ΑΚΡΙΒΕΣ ΕΣΩΤΕΡΙΚΟ ΓΙΝΟΜΕΝΟ). Έστω U, V δύο σύνολα διανυσμάτων στο $\{0,1\}^d$ και ένας ακέραιος t έτσι ώστε $|U| = |V| = n$, $d = \omega(logn)$ και $t \in [d]$. Το πρόβλημα EIP_t ζητά την εύρεση δύο διανυσμάτων $u \in U$ και $v \in V$ τέτοια ώστε $\langle u, v \rangle = t$.

Θα ορίσουμε τώρα την οικογένεια πολυωνύμων $\mathcal{F}EIP$. Πρώτα όμως πρέπει να ορίσουμε το εξής πολυώνυμο:

Ορισμός 0.4.2.

$$fEIP_{t,n}(U, V) = \sum_{i,j \in [n]} \prod_{l \in [\lceil logd \rceil]} f(t_l, b_l(\sum_{k \in [d]} u_{ik} v_{jk})),$$

όπου

- ο όρος t_l συμβολίζει το l-οστό bit της δυαδικής αναπαράστασης του t.
- b_l είναι ένα πολυώνυμο με συντελεστές στο $\mathbb{F}_p$ που υπολογίζει το l-οστό bit της δυαδικής αναπαράστασης των πρώτων $logd$ στοιχείων του $\mathbb{F}_p$.
- $f(x_1, x_2) = x_1 x_2 + (1 - x_1)(1 - x_2)$

Ορισμός 0.4.3.

$$\mathcal{F}EIP = \{fEIP_{t,n}\}$$

Για την οικογένεια πολυωνύμων $\mathcal{F}EIP$ αποδεικνύουμε το εξής θεώρημα:

Θεώρημα 0.4.1. Αν η $\mathcal{F}EIP$ μπορεί να υπολογιστεί σε χρόνο $O(n^{1+\alpha})$ στη μέση περίπτωση για κάποιο $\alpha > 0$, τότε το EIP μπορεί να υπολογιστεί σε χρόνο $\tilde{O}(n^{1+\alpha})$ στη χειρότερη περίπτωση.

Άμεσα προκύπτει το ακόλουθο πόρισμα:

Πόρισμα 0.4.1. Αν το EIP_t χρειάζεται χρόνο $n^{2-o(1)}$ στη χειρότερη περίπτωση, τότε η $\mathcal{F}EIP$ χρειάζεται χρόνο $n^{2-o(1)}$ για να υπολογιστεί στη μέση περίπτωση.

Θα παρουσιάσουμε τώρα είναι MA πρωτόκολλο για την οικογένεια $\mathcal{F}EIP$ με επαληθευτή υποτετραγωνικού χρόνου. Ορίζουμε αρχικά 2 βοηθητικά πολυώνυμα:

$$fEIP_V(x_1, \ldots, x_d) = \sum_{j\in[n]} \prod_{l\in[\lceil logd \rceil]} f(t_l, b_l(\sum_{k\in[d]} x_k v_{jk}))$$

$$R_{U,V}(x) = fEIP_V(\phi_1(x), \ldots, \phi_d(x)),$$

όπου για $i \in [n]$, το $\phi_l(i)$ είναι ένα πολυώνυμο τέτοιο ώστε $\phi_l(i) = u_{il}$.
Το MA πρωτόκολλο είναι το εξής:

1. Ο αποδείκτης στέλνει στον επαληθευτή τους συντελεστές ενός πολυωνύμου R^* και υποστηρίζει ότι είναι το $R_{U,V}$.

2. Ο επαληθευτής επιλέγει ένα τυχαίο $x \xleftarrow{\$} \mathbb{F}_p$ και ελέγχει αν $R^*(x) \stackrel{?}{=} R_{U,V}(x)$.

Χρησιμοποιώντας το παραπάνω MA πρωτόκολλο είναι εύκολο πλέον να κατασκευάσουμε μία Απόδειξη Εργασίας που στηρίζεται στη δυσκολία χειρότερης περίπτωσης του EIP_t. Για να δημιουργήσουμε μία πρόκληση αρκεί να επιλέξουμε τυχαία μία έγκυρη είσοδο U, V για το πολυώνυμο $fEIP_{t,n}$. Η απόδειξη εργασίας είναι οι συντελεστές του πολυωνύμου $R_{U,V}$. Ο επαληθευτής μπορεί να ελέγξει την απόδειξη επιλέγοντας ένα τυχαίο $x \xleftarrow{\$} \mathbb{F}_p$ και ελέγχοντας αν οι συντελεστές είναι έγκυροι για την τυχαία είσοδο x.

0.5 Μηδενική Γνώση στη Λεπτομερή Κρυπτογραφία

Όπως φαίνεται από τον ορισμό που δώσαμε για τη μηδενική γνώση, αυτή δεν μπορεί να χρησιμοποιηθεί για προβλήματα επιλύσιμα σε πολυωνυμικό χρόνο, καθώς ένας επαληθευτής πολυωνυμικού χρόνου μπορεί να υπολογίσει μόνος του κάποιον μάρτυρα. Σε αυτή την ενότητα δίνουμε έναν ορισμό για αποδείξεις μηδενικής γνώσης που μπορούν να χρησιμοποιηθούν για προβλήματα εντός του P για τα οποία υπάρχουν κοινώς αποδεκτά κάτω φράγματα.

Ορισμός 0.5.1. Έστω Π ένα πρόβλημα για το οποίο εικάζουμε πως απαιτείται χρόνος $\Omega(n^{c-o(1)})$ για την επίλυσή του για κάποιο $c > 0$ και έστω $\mathcal{R}$ η σχέση που το αναγνωρίζει. Μία λεπτομερής απόδειξη μηδενικής γνώσης για την $\mathcal{R}$ με σφάλμα πληρότητας ϵ_c και σφάλμα ορθότητας ϵ_s είναι ένα διαδραστικό πρωτόκολλο μεταξύ ενός αλγορίθμου αποδείκτη $\mathcal{P}$ και ενός αλγορίθμου επαληθευτή $\mathcal{V}$ που ικανοποιεί τις παρακάτω ιδιότητες:

- Ο $\mathcal{V}$ χρειάζεται χρόνο το πολύ $\tilde{O}(n^{c-\epsilon})$ για κάποιο $\epsilon > 0$.

- Πληρότητα: Αν $(x, w) \in \mathcal{R}$ και ο $\mathcal{P}$ γνωρίζει ένα μάρτυρα w για το x, θα καταφέρει να πείσει τον $\mathcal{V}$ με πιθανότητα το λιγότερο $1 - \epsilon_c$.

- Ορθότητα: Αν ο $\tilde{\mathcal{P}}$ δεν διαθέτει κάποιο μάρτυρα w για το x, δεν μπορεί να πείσει τον $\mathcal{V}$ με πιθανότητα μεγαλύτερη από ϵ_s.

- Μηδενική Γνώση: Για κάθε αλγόριθμο επαληθευτή $\tilde{\mathcal{V}}$ χρόνου $\tilde{O}(n^{c-\epsilon})$, υπάρχει ένας αλγόριθμος προσομοιωτής $\mathcal{S}$ χρόνου $\tilde{O}(n^{c-\epsilon})$, ο οποίος δέχεται σαν είσοδο την πρόταση προς απόδειξη x και πρόσβαση "μαύρου κουτιού" με επαναφορά στον $\tilde{\mathcal{V}}$ και επιστρέφει ένα προσομοιωμένο αντίγραφο πρωτοκόλλου τέτοιο ώστε:

$$\mathcal{S}(x, \tilde{\mathcal{V}}(\cdot)) \equiv View(\mathcal{P}(x, w), \tilde{\mathcal{V}}(x))$$

Ομοίως με τον κλασσικό ορισμό των αποδείξεων μηδενικής γνώσης, μπορούμε να ορίσουμε την πιο αδύναμη υπολογιστική μηδενική γνώση και την ισχυρότερη ειδική ορθότητα.

Με βάση τον παραπάνω ορισμό, μπορούμε να δώσουμε πλέον αποδείξεις μηδενικής γνώσης ακόμα και για προβλήματα που είναι επιλύσιμα σε τετραγωνικό χρόνο. Παραθέτουμε μία απόδειξη μηδενικής γνώσης που ικανοποιεί τον παραπάνω ορισμό για το πρόβλημα των Ορθογώνιων Διανυσμάτων.

Δημόσια Είσοδος: Ένα σύνολο U διανυσμάτων $u_1, \ldots, u_n \in \mathbb{F}_2^d$, Ένα σύνολο V διανυσμάτων $v_1, \ldots, v_n \in \mathbb{F}_2^d$, παράμετροι συσσωρευτή par
Ιδιωτική Είσοδος του Αποδείκτη: Ένας μάρτυρας $w = u_\alpha || v_\beta$ τέτοιος ώστε $\alpha, \beta \in [n]$ και $\langle u_\alpha, v_\beta \rangle = 0$
Παράμετροι Πρωτοκόλλου: Αριθμός προεπεξεργασιών για το cut and choose: M, Αριθμός Συμμετοχόντων: N, Αριθμός Επαναλήψεων: τ, Παράμετρος Ασφαλείας: λ

Γύρος 1

Αποδείκτης

Δημιούργησε την απαραίτητη τυχαιότητα:

1. $mseed \xleftarrow{\$} \{0,1\}^\lambda$
2. $(r^{(mask)}, r^{(\Delta)}, r^{(party)}) \leftarrow PRG(mseed)$
3. $\{r_i^{(mask)}\}_{i\in[M]} \leftarrow TreePRG(r^{(mask)})$
4. $\{r_i^{(\Delta)}\}_{i\in[M]} \leftarrow TreePRG(r^{(\Delta)})$
5. $\{r_i^{(party)}\}_{i\in[\tau]} \leftarrow TreePRG(r^{(party)})$

Για κάθε προεπεξεργασία $i \in [M]$:

1. Διάλεξε 2 τυχαίες μάσκες και δημιούργησε N προσθετικά μοιράσματα:
   (a) $(mask_i^{(U)}, mask_i^{(V)}) \leftarrow PRG(r_i^{(mask)})$, όπου $mask_i^{(U)}, mask_i^{(V)} \in \mathbb{F}_2^d$
   (b) $\forall k \in [n]$: $\Delta_{i,k}^{(U)} = u_k - mask_i^{(U)}$; $\Delta_{i,k}^{(V)} = v_k - mask_i^{(V)}$
   (c) $\{[\![mask_i^{(U)}]\!]_j, [\![mask_i^{(V)}]\!]_j, r_{i,j}^{(mask-com)}\}_{j\in[N]} \leftarrow TreePRG(r_i^{(mask)})$;
   $[\![mask_i^{(U)}]\!]_N \leftarrow mask_i^{(U)} - \sum_{k=1}^{N-1} [\![mask_i^{(U)}]\!]_k$;
   $[\![mask_i^{(V)}]\!]_N \leftarrow mask_i^{(V)} - \sum_{k=1}^{N-1} [\![mask_i^{(V)}]\!]_k$
2. Δεσμεύσου στα μοιράσματα των μασκών και στις διαφορές:
   (a) $\forall j \in [N]$: $com_{i,j}^{(mask)} = Com(([\![mask_i^{(U)}]\!]_j, [\![mask_i^{(V)}]\!]_j); r_{i,j}^{(mask-com)})$
   (b) $\{r_{i,k}^{(\Delta)(U)}, r_{i,k}^{(\Delta)(V)}\}_{k\in[n]} \leftarrow TreePRG(r_i^{(\Delta)})$
   (c) $\forall k \in [n]$: $com_{i,k}^{(\Delta)(U)} = Com(\Delta_{i,k}^{(U)}; r_{i,k}^{(\Delta)(U)})$; $com_{i,k}^{(\Delta)(V)} = Com(\Delta_{i,k}^{(V)}; r_{i,k}^{(\Delta)(V)})$
3. Μετάθεσε και συσσώρευσε τις δεσμεύσεις των διαφορών:
   (a) Διάλεξε 2 τυχαίες μεταθέσεις $\phi_i^{(U)}, \phi_i^{(V)}$ από το χώρο όλων των μεταθέσεων μεγέθους n.

(b) $(aux_i^{(U)}, acc_i^{(U)}) \leftarrow Acc.Eval(par, com^{(\Delta)(U)}_{i,\phi_i^{(U)}(1)}, \ldots, com^{(\Delta)(U)}_{i,\phi_i^{(U)}(n)})$;
$(aux_i^{(V)}, acc_i^{(V)}) \leftarrow Acc.Eval(par, com^{(\Delta)(V)}_{i,\phi_i^{(V)}(1)}, \ldots, com^{(\Delta)(V)}_{i,\phi_i^{(V)}(n)})$

(c) Set $R_i = (com_{i,1}^{(mask)}, \ldots, com_{i,N}^{(mask)}, acc_i^{(U)}, acc_i^{(V)})$

Για κάθε επανάληψη $i \in [\tau]$:

1. Δημιούργησε τις ιδιωτικές εισόδους των συμμετοχόντων:
   (a) $\{r_{i,j}^{(party)}, r_{i,j}^{(party-com)}\}_{j\in[N]} \leftarrow TreePRG(r_i^{(party)})$
   (b) Για κάθε $j \in [N]$: $([\![a_i]\!]_j, [\![u_\alpha]\!]_j^{(i)}, [\![v_\beta]\!]_j^{(i)}, [\![c_i]\!]_j) \leftarrow TreePRG(r_{i,j}^{(party)})$, όπου $[\![a_i]\!]_j, [\![u_\alpha]\!]_j^{(i)}, [\![v_\beta]\!]_j^{(i)} \in \mathbb{F}_2^d$ and $[\![c_i]\!]_j) \in \mathbb{F}_2$
2. Δημιούργησε βοηθητικές τιμες:
   (a) $\Delta u_\alpha^{(i)} = u_\alpha - \sum_{j=1}^N [\![u_\alpha]\!]_j^{(i)}$; $\Delta v_\beta^{(i)} = v_\beta - \sum_{j=1}^N [\![v_\beta]\!]_j^{(i)}$
   (b) $\Delta c_i = \langle a_i, u_\alpha \rangle - \sum_{j=1}^N [\![c_i]\!]_j$, όπου $a_i = \sum_{j=1}^N [\![a_i]\!]_j$
3. Δεσμεύσου στις εισόδους των συμμετοχόντων:
   (a) $\forall j \in [N]$: $com_{i,j}^{(party)} = Com(r_{i,j}^{(party)}; r_{i,j}^{(party-com)})$
   (b) $I_i = (\Delta u_\alpha^{(i)}, \Delta v_\beta^{(i)}, \Delta c_i, com_{i,1}^{(party)}, \ldots, com_{i,N}^{(party)})$.

Στείλε τους συσσωρευτές και όλες τις δεσμεύσεις στον επαληθευτή:

1. $R = (R_1, \ldots, R_M)$.
2. $I = (I_1, \ldots, I_\tau)$.
3. $h_1 = Hash(R, I)$.
4. Στείλε το h_1 στον επαληθευτή.

Γύρος 2

Επαληθευτής

1. Λάβε το h_1' από τον αποδείκτη.
2. Διαμέρισε το $[M]$ σε υποσύνολα J, S επιλέγοντας τυχαία $J \xleftarrow{\$} \{J \subset [M]; |J| = \tau\}$. Στο εξής, θα συμβολίζουμε το i-οστό στοιχείο του J με J_i.
3. $\forall i \in [\tau]$: Διάλεξε τυχαία $\epsilon_i \xleftarrow{\$} \mathbb{F}_2^d$.
4. Στείλε $(J, \{\epsilon_i\}_{i\in[\tau]})$ στον αποδείκτη.

Γϋρος 3

Αποδείκτης

Άνοιξε όλες τις προεπεξεργασίες στο S:

$\forall i \in S$: Στείλε $(r_i^{(mask)}, r_i^{(\Delta)}, \phi_i^{(U)}, \phi_i^{(V)})$ στον επαληθευτή.

Για κάθε επανάληψη $i \in [\tau]$, χρησιμοποίησε ένα πρωτόκολλο MPC-in-the-head για να αποδείξεις ότι $u_\alpha \in U$, $v_\beta \in V$ και $\langle u_\alpha, v_\beta \rangle = 0$:

1. Κάθε συμμετέχων P_j δέχεται ιδιωτικές εισόδους: $[\![mask_{J_i}^{(U)}]\!]_j, [\![mask_{J_i}^{(V)}]\!]_j, [\![u_\alpha]\!]_j^{(i)}, [\![v_\beta]\!]_j^{(i)}, [\![a_i]\!]_j, [\![c_i]\!]_j$. Επίσης ολοι οι συμμετέχοντες και ο επαληθευτής έχουν δημόσια πρόσβαση στα σύνολα U, V και στα $\Delta_{J_i,\alpha}^{(U)}, \Delta_{J_i,\beta}^{(V)}$.

2. Κάθε συμμετέχων P_j κάνει τα ακόλουθα:

   (a) $[\![\Delta_{J_i,\alpha}^{(U)}]\!]_j = [\![u_\alpha]\!]_j^{(i)} - [\![mask_{J_i}^{(U)}]\!]_j$.

   (b) $[\![\Delta_{J_i,\beta}^{(V)}]\!]_j = [\![v_\beta]\!]_j^{(i)} - [\![mask_{J_i}^{(V)}]\!]_j$.

   (c) $[\![e_i]\!]_j = \epsilon_i \circ [\![v_\beta]\!]_j^{(i)} + [\![a_i]\!]_j$.

3. Όλοι οι συμμετέχοντες ανοίγουν τα $[\![\Delta_{J_i,\alpha}^{(U)}]\!], [\![\Delta_{J_i,\beta}^{(V)}]\!], [\![e_i]\!]_j$ για να πάρουν τα $\Delta_{J_i,\alpha}^{(U)}, \Delta_{J_i,\beta}^{(V)}, e_i$ αντίστοιχα.

4. Κάθε συμμετέχων P_j υπολογίζει $[\![g_i]\!]_j = \langle e_i, [\![u_\alpha]\!]_j^{(i)} \rangle - [\![c_i]\!]_j$.

5. Δημιούργησε αποδείξεις μέλους για τα $\Delta_{J_i,\alpha}^{(U)}, \Delta_{J_i,\beta}^{(V)}$:

   (a) $\pi_i^{(U)} \leftarrow Acc.Proof(par, com^{(\Delta)(U)}_{J_i,\phi_{J_i}^{(U)}(\alpha)}, acc_{J_i}^{(U)}, aux_{J_i})$.

   (b) $\pi_i^{(V)} \leftarrow Acc.Proof(par, com^{(\Delta)(V)}_{J_i,\phi_{J_i}^{(V)}(\beta)}, acc_{J_i}^{(V)}, aux_{J_i})$.

$h_2 = Hash(\{[\![\Delta_{J_i,\alpha}^{(U)}]\!], [\![\Delta_{J_i,\beta}^{(V)}]\!], [\![e_i]\!], [\![g_i]\!]\}_{i\in[\tau]})$

Στείλε $(h_2, \{\pi_i^{(U)}, \pi_i^{(V)}, \Delta_{J_i,\alpha}^{(U)}, \Delta_{J_i,\beta}^{(V)}, r_{J_i,\alpha}^{(\Delta)(U)}, r_{J_i,\beta}^{(\Delta)(V)}\}_{i\in[\tau]})$ στον επαληθευτή.

Γύρος 4

Επαληθευτής

1. Λάβε τα $\{(r_i^{(mask)\prime}, r_i^{(\Delta)\prime}, \phi_i^{(U)\prime}, \phi_i^{(V)\prime})\}_{i\in S}$ από τον αποδείκτη.

2. Λάβε τα $(h_2', \{\pi_i^{(U)\prime}, \pi_i^{(V)\prime}, \Delta_{J_i,\alpha}^{(U)\prime}, \Delta_{J_i,\beta}^{(V)\prime}, r_{J_i,\alpha}^{(\Delta)(U)\prime}, r_{J_i,\beta}^{(\Delta)(V)\prime}\}_{i\in[\tau]})$ από τον αποδείκτη.

3. $\forall i \in [\tau]$: Διάλεξε τυχαίες προκλήσεις $j_i^* \leftarrow [N]$.

4. Στείλε τα $\{j_i^*\}_{i\in[\tau]}$ στον αποδείκτη.

Γϋρος 5

Αποδείκτης

Για κάθε $i \in [\tau]$: Άνοιξε όλους τους συμμετέχοντες $\{P_j\}_{j\in[N]\setminus\{j_i^*\}}$ και στείλε τις απαραίτητες βοηθητικές πληροφορίες:

Στείλε τα ακόλουθα στον επαληθευτή:

- $\{r_{J_i,j}^{(mask-com)}\}_{j\in[N]\setminus\{j_i^*\}}$
- $\{[\![mask_{J_i}^{(U)}]\!]_j, [\![mask_{J_i}^{(V)}]\!]_j\}_{j\in[N]\setminus\{j_i^*\}}$
- $\{r_{i,j}^{(party)}, r_{i,j}^{(party-com)}\}_{j\in[N]\setminus\{j_i^*\}}$

- $com_{i,j_i^*}^{(party)}$
- $\Delta u_\alpha^{(i)}, \Delta v_\beta^{(i)}$
- Δc_i
- $[\![e_i]\!]_{j_i^*}$
- $(acc_{J_i}^{(U)}, acc_{J_i}^{(V)})$
- $com_{J_i,j_i^*}^{(mask)}$

Επαληθευτής

Λάβε τα ακόλουθα από τον αποδείκτη:

- $\{r_{J_i,j}^{(mask-com)\prime}\}_{j\in[N]\setminus\{j_i^*\}}$
- $\{[\![mask_{J_i}^{(U)}]\!]'_j, [\![mask_{J_i}^{(V)}]\!]'_j\}_{j\in[N]\setminus\{j_i^*\}}$
- $\{r_{i,j}^{(party)\prime}, r_{i,j}^{(party-com)\prime}\}_{j\in[N]\setminus\{j_i^*\}}$
- $com_{i,j_i^*}^{(party)\prime}$
- $\Delta u_\alpha^{(i)\prime}, \Delta v_\beta^{(i)\prime}$
- $\Delta c_i'$
- $[\![e_i]\!]'_{j_i^*}$
- $acc_{J_i}^{(U)\prime}, acc_{J_i}^{(V)\prime}$
- $com_{J_i,j_i^*}^{(mask)\prime}$

Υπολόγισε τις προεπεξεργασίες στο S:

Για όλα τα $i \in S$:

1. $(mask_i^{(U)\prime}, mask_i^{(V)\prime}) \leftarrow PRG(r_i^{(mask)\prime})$.
2. $\forall k \in [n]$: $\Delta_{i,k}^{(U)\prime} = u_k - mask_i^{(U)\prime}$; $\Delta_{i,k}^{(V)\prime} = v_k - mask_i^{(V)\prime}$
3. $\{[\![mask_i^{(U)}]\!]'_j, [\![mask_i^{(V)}]\!]'_j, r_{i,j}^{(mask-com)\prime}\}_{j\in[N]} \leftarrow TreePRG(r_i^{(mask)\prime})$;
   $[\![mask_i^{(U)}]\!]'_N \leftarrow mask_i^{(U)\prime} - \sum_{k=1}^{N-1} [\![mask_i^{(U)}]\!]'_k$;
   $[\![mask_i^{(V)}]\!]'_N \leftarrow mask_i^{(V)\prime} - \sum_{k=1}^{N-1} [\![mask_i^{(V)}]\!]'_k$
4. $\forall j \in [N]$: $com_{i,j}^{(mask)\prime} = Com(([\![mask_i^{(U)}]\!]'_j, [\![mask_i^{(V)}]\!]'_j); r_{i,j}^{(mask-com)\prime})$
5. $\{r_{i,k}^{(\Delta)(U)\prime}, r_{i,k}^{(\Delta)(V)\prime}\}_{k\in[n]} \leftarrow TreePRG(r_i^{(\Delta)\prime})$
6. $\forall k \in [n]$: $com_{i,k}^{(\Delta)(U)\prime} = Com(\Delta_{i,k}^{(U)\prime}; r_{i,k}^{(\Delta)(U)\prime})$; $com_{i,k}^{(\Delta)(V)\prime} = Com(\Delta_{i,k}^{(V)\prime}; r_{i,k}^{(\Delta)(V)\prime})$
7. $(aux_i^{(U)\prime}, acc_i^{(U)\prime}) \leftarrow Acc.Eval(par, com_{i,\phi_i^{(U)}(1)'}^{(\Delta)(U)\ \prime}, \ldots, com_{i,\phi_i^{(U)}(n)'}^{(\Delta)(U)\ \prime})$;
   $(aux_i^{(V)\prime}, acc_i^{(V)\prime}) \leftarrow Acc.Eval(par, com_{i,\phi_i^{(V)}(1)'}^{(\Delta)(V)\ \prime}, \ldots, com_{i,\phi_i^{(V)}(n)'}^{(\Delta)(V)\ \prime})$

8. $R'_i = (com^{(mask)\prime}_{i,1}, \ldots, com^{(mask)\prime}_{i,N}, acc^{(U)\prime}_i, acc^{(V)\prime}_i)$

Επαλήθευσε τις αποδείξεις μέλους και τις μάσκες:

Για κάθε επανάληψη $i \in [\tau]$:

1. $com^{(\Delta)(U)\prime}_{J_i,\alpha} = Com(\Delta^{(U)\prime}_{J_i,\alpha}; r^{(\Delta)(U)\prime}_{J_i,\alpha})$; $com^{(\Delta)(V)\prime}_{J_i,\beta} = Com(\Delta^{(V)\prime}_{J_i,\beta}; r^{(\Delta)(V)\prime}_{J_i,\beta})$.
2. Check: $Acc.Verify(par, com^{(\Delta)(U)\prime}_{J_i,\alpha}, acc^{(U)\prime}_{J_i}, \pi^{(U)\prime}_i) \stackrel{?}{=} 1$; $Acc.Verify(par, com^{(\Delta)(V)\prime}_{J_i,\beta}, acc^{(V)\prime}_{J_i}, \pi^{(V)\prime}_i) \stackrel{?}{=} 1$.
3. $\forall j \in [N] \setminus \{j^*_i\}$: $com^{(mask)\prime}_{J_i,j} = Com((\llbracket mask^{(U)}_{J_i} \rrbracket'_j, \llbracket mask^{(V)}_{J_i} \rrbracket'_j); r^{(mask-com)\prime}_{J_i,j})$.

Για κάθε $i \in J$: $R'_i = (com^{(mask)\prime}_{i,1}, \ldots, com^{(mask)\prime}_{i,N}, acc^{(U)\prime}_i, acc^{(V)\prime}_i)$.

$R' = (R'_1, \ldots, R'_M)$.

Επαλήθευσε τις εισόδους των συμμετοχόντων:

Για κάθε επανάληψη $i \in [\tau]$:

1. $\forall j \in [N] \setminus \{j^*_i\}$: $com^{(party)\prime}_{i,j} = Com(r^{(party)\prime}_{i,j}; r^{(party-com)\prime}_{i,j})$
2. $I'_i = (\Delta u^{(i)\prime}_\alpha, \Delta v^{(i)\prime}_\beta, \Delta c'_i, com^{(party)\prime}_{i,1}, \ldots, com^{(party)\prime}_{i,N})$.

$I' = (I'_1, \ldots, I'_\tau)$.

Έλεγξε αν $h'_1 \stackrel{?}{=} Hash(R', I')$.

Για κάθε επανάληψη $i \in [\tau]$, επαλήθευσε το πρωτόκολλο MPC-in-the-head:

1. $\forall j \in [N] \setminus \{j^*_i\}$: $(\llbracket a_i \rrbracket'_j, \llbracket u_\alpha \rrbracket^{(i)\prime}_j, \llbracket v_\beta \rrbracket^{(i)\prime}_j, \llbracket c_i \rrbracket'_j) \leftarrow TreePRG(r^{(party)\prime}_{i,j})$.
2. $\Delta e'_i = \epsilon_i \circ \Delta v^{(i)\prime}_\beta$.
3. $\forall j \in [N] \setminus \{j^*_i\}$: $\llbracket \Delta^{(U)}_{J_i,\alpha} \rrbracket'_j = \llbracket u_\alpha \rrbracket^{(i)\prime}_j - \llbracket mask^{(U)}_{J_i} \rrbracket'_j$.
4. $\forall j \in [N] \setminus \{j^*_i\}$: $\llbracket \Delta^{(V)}_{J_i,\beta} \rrbracket'_j = \llbracket v_\beta \rrbracket^{(i)\prime}_j - \llbracket mask^{(V)}_{J_i} \rrbracket'_j$.
5. $\forall j \in [N] \setminus \{j^*_i\}$: $\llbracket e_i \rrbracket'_j = \epsilon_i \circ \llbracket v_\beta \rrbracket^{(i)\prime}_j + \llbracket a_i \rrbracket'_j$.
6. $\forall j \in [N] \setminus \{j^*_i\}$: $\llbracket g_i \rrbracket'_j = \langle \sum_{k=1}^N \llbracket e_i \rrbracket'_k, \llbracket u_\alpha \rrbracket^{(i)\prime}_j \rangle - \llbracket c_i \rrbracket'_j$.
7. $e'_i = \sum_{j=1}^N \llbracket e_i \rrbracket'_j + \Delta e'_i$.
8. $\Delta g'_i = \langle e'_i, \Delta u^{(i)\prime}_\alpha \rangle - \Delta c'_i$.
9. $\llbracket \Delta^{(U)}_{J_i,\alpha} \rrbracket'_{j^*_i} = \Delta^{(U)\prime}_{J_i,\alpha} - \Delta u^{(i)\prime}_\alpha - \sum_{j=1, j\neq j^*_i}^N \llbracket \Delta^{(U)}_{J_i,\alpha} \rrbracket'_j$.
10. $\llbracket \Delta^{(V)}_{J_i,\beta} \rrbracket'_{j^*_i} = \Delta^{(V)\prime}_{J_i,\beta} - \Delta v^{(i)\prime}_\beta - \sum_{j=1, j\neq j^*_i}^N \llbracket \Delta^{(V)}_{J_i,\beta} \rrbracket'_j$.
11. $\llbracket g_i \rrbracket'_{j^*_i} = -\Delta g'_i - \sum_{j=1, j\neq j^*_i} \llbracket g_i \rrbracket'_j$.

Έλεγξε αν $h_2' \stackrel{?}{=} Hash(\{[\![\Delta_{J_i,\alpha}^{(U)}]\!]', [\![\Delta_{J_i,\beta}^{(V)}]\!]', [\![e_i]\!]', [\![g_i]\!]'\}_{i\in[\tau]})$.

Επίστρεψε 1.

Η ασυμπτωτική επικοινωνιακή πολυπλοκότητα του παραπάνω πρωτοκόλλου είναι

$$O(\tau(log\tau + \lambda logN + d))$$

ενώ η ασυμπτωτική υπολογιστική πολυπλοκότητα είναι

$$O(M(n+N)(\lambda + d))$$

Επιπλέον μπορούμε να αποδείξουμε τα παρακάτω θεωρήματα που αφορούν την πληρότητα, την ορθότητα και τη μηδενική γνώση του πρωτοκόλλου:

Θεώρημα 0.5.1. Ένας αποδείκτης $\mathcal{P}$ που γνωρίζει μία λύση $u_\alpha||v_\beta$ για ένα στιγμιότυπο U, V του προβλήματος των Ορθογώνιων Διανυσμάτων και που ακολουθεί τα βήματα του πρωτοκόλλου πείθει τον επαληθευτή $\mathcal{V}$ με πιθανότητα 1.

Θεώρημα 0.5.2. Ένας αποδείκτης $\tilde{\mathcal{P}}$ που δεν γνωρίζει μία λύση $u_\alpha||v_\beta$ σε ένα στιγμιότυπο U, V του προβλήματος των Ορθογώνιων Διανυσμάτων δεν μπορεί να πείσει τον επαληθευτή $\mathcal{V}$ με πιθανότητα μεγαλύτερη από ϵ_s που ικανοποιεί:

$$\epsilon_s \leq \max_A \left\{ \frac{\binom{M-A}{M-\tau}}{\binom{M}{M-\tau}} \cdot \left(\frac{1}{2} + \frac{1}{N}\right)^{\tau - A} \right\}$$

Θεώρημα 0.5.3. Εστω

$$\epsilon_s = \max_A \left\{ \frac{\binom{M-A}{M-\tau}}{\binom{M}{M-\tau}} \cdot \left(\frac{1}{2} + \frac{1}{N}\right)^{\tau - A} \right\}$$

Έστω ότι υπάρχει ένας αποδοτικός αποδείκτης $\tilde{\mathcal{P}}$ τέτοιος ώστε

$$\tilde{\epsilon}_s := Pr[\langle \tilde{\mathcal{P}}(x), \mathcal{V}(x) \rangle (U, V) = 1] > \epsilon_s,$$

όπου τα U, V είναι ένα στιγμιότυπο του προβλήματος των Ορθογώνιων Διανυσμάτων. Τότε, υπάρχει ένας αλγόριθμος εξαγωγέας $\mathcal{E}$ ο οποίος, με πρόσβαση "μαύρου κουτιού" με επαναφορά στον $\tilde{\mathcal{P}}$, παράγει είτε έναν μάρτυρα $u||v$ έτσι ώστε $\langle u, v \rangle = 0$, $u \in U$ και $v \in V$, μία σύγκρουση στη συνάρτηση κατακερματισμού ή μία σύγκρουση στο σχήμα δέσμευσης, κάνοντας κατά μέσο όρο

$$\frac{4}{\tilde{\epsilon}_s - \epsilon_s} \cdot \left(1 + \tilde{\epsilon}_s \cdot \frac{8M}{\tilde{\epsilon}_s - \epsilon_s}\right)$$

κλήσεις του $\tilde{\mathcal{P}}$.

Θεώρημα 0.5.4. Έστω ότι το PRG και το $TreePRG$ που χρησιμοποιήθηκαν στο πρωτόκολλο είναι (t, ϵ_{PRG})-ασφαλή, ο συσσωρευτής ACC είναι (t, ϵ_{ACC})-ασφαλής το σχήμα δέσμευσης Com είναι (t, ϵ_{Com})-ασφαλές. Τότε, υπάρχει ένας αποδοτικός αλγόριθμος προσομοιωτής $\mathcal{S}$ ο οποίος, με είσοδο τυχαίες προκλήσεις $J, \{\epsilon_i\}_{i\in[\tau]}, \{j_i^*\}_{i\in[\tau]}$, επιστρέφει ένα αντίγραφο πρωτοκόλλου που είναι $(t, \tau \cdot \epsilon_{PRG} + \tau \cdot \epsilon_{ACC} + \tau \cdot \epsilon_{Com})$-μη διακρίσιμο από ένα πραγματικό αντίγραφο του πρωτοκόλλου.

Χρησιμοποιώντας τις ίδιες τεχνικές μπορούμε να επαναλάβουμε το παραπάνω πρωτόκολλο για το πρόβλημα $3SUM$ της λεπτομερούς πολυπλοκότητας, κάθως και για άλλα προβλήματα τετραγωνικού χρόνου για τα οποία γνωρίζουμε λεπτομερείς αναγωγές προς αυτά από τα OV, $3SUM$.

0.6 Συμπεράσματα

Η λεπτομερής κρυπτογραφία είναι μία γοητευτική περιοχή έρευνας που έχει τραβήξει μεγάλο ενδιαφέρον τα τελευταία χρόνια, καθώς μπορεί δυνητικά να μας προσφέρει κρυπτογραφικά εργαλεία τα οποία μπορούν να χρησιμοποιηθούν ακόμα και κάτω από καταστροφικές για την κρυπτογραφία υποθέσεις όπως ότι $NP \subseteq BPP$. Στη διπλωματική αυτή εργασία κάναμε κάποια πρόοδο στον τομέα της λεπτομερούς κρυπτογραφίας ορίζοντας μία οικογένεια συναρτήσεων που είναι δύσκολο να υπολογιστεί σε υποτετραγωνικό χρόνο υποθέτοντας τη δυσκολία χειρότερης περίπτωσης του προβλήματος ΑΚΡΙΒΕΣ ΕΣΩΤΕΡΙΚΟ ΓΙΝΟΜΕΝΟ. Στη συνέχεια εκμεταλλευτήκαμε τη δυσκολία υπολογισμού των συναρτήσεών μας προς τη δημιουργία μίας λεπτομερούς Απόδειξης Εργασίας. Τέλος μελετήσαμε πώς η έννοια της μηδενικής γνώσης μπορεί να μεταφερθεί στη λεπτομερή κρυπτογραφία και δώσαμε πρωτόκολλα μηδενικής γνώσης για τα κυριότερα προβλήματα της περιοχής.

Chapter 1

Introduction

In 1995, Russell Impagliazzo [Impa95] classified the ultimate consequences of complexity theory into 5 possible worlds:

- Algorithmica: All problems in NP can be solved in polynomial time, i.e., P=NP. This world also includes the possibility that efficiency within NP is derived from probabilistic polynomial time algorithms, i.e., $NP \subseteq BPP$. $NP \nsubseteq BPP$ is the least requirement (but that is hardly ever enough) [Impa89b] to achieve traditional cryptography, so the construction of any cryptographic tool is impossible in this world.
- Heuristica: NP problems are intractable in the worst case, but tractable on average for any samplable distribution, i.e., $DistNP \subseteq AvgP$.
- Pessiland: $DistNP \nsubseteq AvgP$, however one-way functions do not exist. In this world, creating hard instances of NP problems is easy, however we can't get any cryptographic advantage from them as we can't create them along with their solution.
- Minicrypt: One-way functions exist, but public key cryptography is impossible, i.e., it's impossible to agree on a secret with a stranger via a publicly accessible channel.
- Cryptomania: Public key cryptography is possible.

Even though no world has been excluded yet, most computer scientists believe that we live in Cryptomania or Minicrypt. A fascinating question is the following: Is there a way to get some form of cryptography that can be applied in all five worlds? It turns out this is possible by considering hardness within P and thus, a different notion of security than in traditional cryptography.

This field of research has only been recently formalised in [Degw16] as fine-grained cryptography, however constructions that would positively answer this question (under assumptions) have already been created before. Assuming the existence of a random oracle, accessible to both the honest parties and the adversary, Merkle [Merk78] constructed a public key encryption scheme, where the honest parties run in $O(n)$ time and security is shown against $o(n^2)$ adversaries. Later, Maurer [Maur92] introduced a model in which adversaries have unbounded computation time but only an a priori bounded amount of space. Following this work, Cachin and Maurer [Cach97] constructed unconditionally secure symmetric-key encryption and key-exchange protocols in that model. Finally, Hastard [Hast87] created a one-way permutation that can be computed in NC^0, but is hard to invert for any AC^0 circuit.

As also noted in [Degw16], all fine-grained cryptographic constructions share two common features:

- Security is only achieved against a class of adversaries with bounded resources (e.g. time, space, parallel time etc.)
- The honest algorithms use less resources than the class of adversaries they are trying to fool.

We note two additional facts about fine-grained cryptography:

- Adversaries are either a priori bounded by some fixed polynomial or by a circuit complexity class. The latter case has been extensively studied [Appl06, Degw16, Egas21] and many cryptographic constructions (e.g., one-way functions, pseudo-random generators, collision-resistant hash functions, public-key encryptions schemes) have been created. However, the unique structure of circuit complexity classes makes those constructions difficult to be used in practice. In this thesis, we are interested in the first case.

- We can also get unconditional security. So far, this has only been achieved by basing security on some circuit lower bounds, e.g., the hardness to compute the PARITY function by any AC^0 circuit [Furs84, Ajta83, Hast86].

For the rest of this thesis, fine-grained cryptography only refers to the case where adversaries are bounded by some fixed polynomial.

In order to achieve fine-grained cryptography, we need some hardness assumptions within P. The theory of NP-completeness [Cook71, Karp72, Levi73] has identified many natural problems as NP-complete problems, which are considered to be "intractable". Similarly, fine-grained complexity has conjectured polynomial lower bounds for a variety of important problems like Orthogonal Vectors (OV) [Vass15], 3SUM [Vass15], Longest Common Subsequence (LCS) [Abbo15a, Brin15], Edit Distance [Back15] etc. Some of these lower bounds are based on the Strong Exponential Time Hypothesis (SETH), a stronger version of Exponential Time Hypothesis (ETH) [Impa01b].

Recently, there has been a long line of research towards transferring notions of traditional cryptography into the fine-grained world. Most of this work is centered around fine-grained average case hardness which has been achieved using a variety of methods [Ball17a, Dali20, Gold20]. Other important works construct fine-grained proofs of work [Ball17b, Ball18] or make some progress towards proving the existence of fine-grained one-way functions [Brzu22].

1.1 Contribution

We follow the approach of [Ball17a] to construct a new family of polynomials, which we call $\mathcal{F}EIP$, that are hard to evaluate on average assuming $SETH$ holds. In fact, we base the average case hardness of our polynomials on the worst case hardness of the problem EIP_t, which is a stronger assumption. EIP_t asks, given two sets U, V of n vectors from $\{0,1\}^{d(n)}$, whether there exist $u \in U$ and $v \in V$ such that $\langle u, v \rangle = t$ (over $\mathbb{Z}$). Specifically we provide a fine-grained reduction from OV to EIP_t, a fine grained reduction from EIP_t to $\mathcal{F}EIP$ and a worst-case to average-case reduction from $\mathcal{F}EIP$ to itself. We, then, give an MA protocol for $\mathcal{F}EIP$ and use it to construct a fine-grained Proof of Work scheme that is based on the average-case hardness of $\mathcal{F}EIP$.

We propose a scaled down definition for zero knowledge arguments of knowledge that allows us to create zero knowledge arguments for problems that are conjectured to have polynomial lower bounds. Then, we create zero knowledge protocols that satisfy our definition's restrictions for the problems OV and $3SUM$ in the random oracle model [Cane04].In practice, the random oracle can be substituted for any traditional hash function or a fine-grained hash function. We use MPC-in-the-head [Isha09] as our main technique and also rely on ideas from [Goel22], originally used to create ring signatures. The running time of both the verifier and the prover is $\tilde{O}(n)$. Finally, we demonstrate how to use our zero knowledge protocols to obtain zero knowledge arguments of knowledge for other quadratic time problems.

1.2 Outline

The rest of the thesis is structured as follows:

- Chapter 2: We present some background knowledge on basic cryptographic primitives that are used in our zero knowledge arguments of knowledge.
- Chapter 3: We provide some theoretical background on zero knowledge arguments and MPC-in-the-head. We also describe the MPC-in-the-Head protocol that is used to prove orthogoniality in our protocol for OV. Then, we define fine-grained reductions and present the most popular conjectures of fine-grained complexity.
- Chapter 4: We present some existing results in average-case fine grained hardness. Then we introduce our new family of polynomials that are conjectured to be hard to evaluate on average. We also give an MA protocol for our family of polynomials and present our fine-grained Proof of Work.
- Chapter 5: We define fine-grained zero knowledge arguments for problems with known polynomial lower bounds and present our protocols for OV and $3SUM$. Then, we demonstrate how to obtain more protocols for other problems.
- Chapter 6: We summarise our results and provide some directions for future research.

Chapter 2

Cryptographic Primitives

In this chapter we provide some background on the cryptographic primitives that are used in our zero-knowledge arguments of knowledge. As we have already mentioned, our protocols work in the random oracle model, i.e., both the prover and the verifier have access to a random function. For the rest of this work, we assume that the random oracle is a hash function and it's denoted by $Hash$. We note that this is our only assumption, since all the other cryptographic primitives can be constructed by only using hash functions. We will present the cryptographic primitives in the classical setting, where security is achieved against probabilistic polynomial time (PPT) adversaries.

2.1 Hash Functions

Hash functions are functions that take strings of arbitrary length as input and compress them into short outputs of fixed length. In order to be of practical use, hash functions also need to have other properties like collision resistance, i.e., it is infeasible for any PPT algorithm to find two distinct inputs that give the same output. We now provide a formal definition as given in [Katz20].

Definition 2.1.1. A **hash function** (with output length l) is a pair of probabilistic polynomial-time algorithms (Gen, H) satisfying the following:

- Gen is a probabilistic algorithm which takes as input a security parameter 1^n and outputs a key s. We assume that 1^n is implicit in s.
- H takes as input a key s and a string $x \in \{0,1\}^*$ and outputs a string $H^s(x) \in \{0,1\}^{l(n)}$ (where n is the value of the security parameter implicit in s).

In order to define security, we first need to define an experiment for a hash function $\Pi = (Gen, Hash)$, an adversary $\mathcal{A}$, and a security parameter n:

Definition 2.1.2 (**The Collision-Finding Experiment** $HashColl_{\mathcal{A},\Pi}(n)$)**.** The steps of the experiment are the following:

1. A key s is generated by running $Gen(1^n)$.
2. The adversary $\mathcal{A}$ is given s and outputs x, x'.
3. The output of the experiment is defined to be 1 if and only if $x \neq x'$ and $H^s(x) = H^s(x')$. In such a case we say that $\mathcal{A}$ has found a collision.

We finally give the definition of collision resistant hash functions, as we are only interested in those.

Definition 2.1.3. A hash function $\Pi = (Gen, H)$ is **collision resistant** if for all probabilistic polynomial time adversaries $\mathcal{A}$ there is a negligible function $negl$ such that

$$Pr[HashColl_{\mathcal{A},\Pi}(n) = 1] \leq negl(n)$$

In order to be used for cryptographic applications, hash functions also need to satisfy some additional properties. For example, a good hash function should also not reveal information about any number of input bits, which is not guaranteed by the property of collision resistance. Hash functions that are widely used today [Dwor15] seem to satisfy all security requirements.

2.1.1 Fine-Grained One-Way Functions

The traditional definition of collision resistant hash functions works well our proofs in chapter 5. However, the ultimate goal of fine-grained cryptography is to create cryptographic primitives that rely on completely different hardness assumptions, so they can also be used in obscure settings (e.g., a world where $NP \subseteq BPP$). Research is focused on creating a one-way function whose security is based on the worst case hardness of a core problem in fine-grained complexity theory [Ball17a, Brzu22]. A definition that captures this concept, as well as fine-grained cryptography in general was given in [Ball17a] and is the following:

Definition 2.1.4. We say a function $f\colon \{0,1\}^* \to \{0,1\}^*$ is **(t, ϵ)-one-way** if it can be computed in $O(t(n)^{1-\delta})$ time for some $\delta > 0$, but for any $\delta' > 0$, any $O(t(n)^{1-\delta'})$-time algorithm $\mathcal{A}$, and all sufficiently large n,

$$\Pr_{x \leftarrow \{0,1\}^n}[\mathcal{A}(f(x)) \in f^{-1}(f(x))] \leq \epsilon(n, \delta')$$

If we consider fine-grained one-way functions whose security relies on fine-grained complexity assumptions (e.g., the hardness of solving an Orthogonal Vectors instance in subquadratic time), we can expect $t(n)$ to be some fixed polynomial.

2.2 Pseudorandom Generators

Pseudorandom generators (PRGs))were first introduced by Blum and Micali in [Blum84] and later formalised by Yao in [Yao82]. They are efficient, deterministic algorithms for transforming a short, uniform string, which is called a seed, into a longer output string which looks like a uniformly random one. Pseudorandomness speeds up the computations, as creating truly random bits is a hard and time consuming task. In our zero knowledge proofs, it also achieves a lower communication complexity. In order to be considered secure, a PRG needs to be able to pass every efficient statistical test, i.e., no PPT algorithm should be able to distinguish the output of a PRG from a uniformly random string, unless with a negligible probability. We now present a formal definition of a pseudorandom generator as given in [Katz20].

Definition 2.2.1. Let l be a polynomial and let G be a deterministic polynomial-time algorithm such that for every n and any input $s \in \{0,1\}^n$, the result $G(s)$ is a string of length $l(n)$. We say that G is a **pseudorandom generator** if the following conditions hold:

- Expansion: For every n it holds that $l(n) > n$.
- Pseudorandomness: For any PPT algorithm D, there is a negligible function $negl$ such that
$$|Pr[D(G(s)) = 1] - Pr[D(r) = 1]| \leq negl(n),$$
where the first probability is taken over uniform choice of $s \in \{0,1\}^n$ and the randomness of D, and the the second probability is taken over uniform choice of $r \in \{0,1\}^{l(n)}$ and the randomness of D. We call l the expansion factor of G.

We note that pseudorandom generators can efficiently be constructed from any one-way function. Such results are known for a long time [Impa89a, Hast90, Hast99], however more recent results exist as well [Bold12].

2.2.1 Tree-Based Pseudorandom Generators

We now present the definition of Tree-Based Pseudorandom Generators (Tree PRGs) as given in [Goel22], which we use in our protocols to generate the shares of many parties efficiently and also lower the communication complexity.

Definition 2.2.2 (**Tree-Based Pseudorandom Generator**)**.** A tree based pseudorandom generator corresponding to a pseudorandom generator $PRG : \{0,1\}^{\kappa} \to \{0,1\}^{2\kappa}$ is defined by a tuple of algorithms $(TreePRG.Gen, TreePRG.Punc, TreePRG.GenPunc)$ as follows:

- $TreePRG.Gen(seed, l)$: On input the seed and number of leaves l, this algorithm outputs l seeds, computed as follows:
  - Set $seed_1^0 = seed$.
  - For each $k \in [log(l)]$ and $i \in [2^{k-1}]$, compute $seed_{2i-1}^k || seed_{2i}^k = PRG(seed_i^{k-1})$.
  - Output $seed_1^{log(l)}, \ldots, seed_l^{log(l)}$.
- $TreePRG.Punc(seed, l, q)$: On input the seed, number of leaves l and an index q, this algorithm computes a punctured seed S, that is computed as follows:
  - Set $seed_1^0 = seed$.
  - For each $k \in [log(l)]$ and $i \in [2^{k-1}]$, compute $seed_{2i-1}^k || seed_{2i}^k = PRG(seed_i^{k-1})$.
  - Initialise $S = \{\}$. For each $k \in [log(l)]$, set $S = S \cup \{sibling(seed^k_{\lfloor \frac{q}{2^{log(l)-k}} \rfloor})\}$.
- $TreePRG.GenPunc(S, l, q)$: On input the punctured seed S, number of leaves l and the punctured index q, parse $S = ((1, seed^1), \ldots, (log(l), seed^{log(l)}))$, this algorithm recovers all the leaf seeds except for the punctured leaf seed as follows:
  - For each $k \in [log(l)-1]$, for each $i \in [2^k] \backslash \{\lfloor \frac{q}{2^{log(l)-k}} \rfloor\}$, compute $seed_{2i-1}^{k+1} || seed_{2i}^{k+1} = PRG(seed_i^k)$.
  - Output $\{seed_i^{log(l)}\}_{i \in [l] \backslash \{q\}}$.

2.3 Commitment Schemes

A commitment scheme is a cryptographic primitive that allows one party to "commit" to a message m by sending a commitment com that "contains" the message m and some randomness r. A secure commitment scheme needs to have the following properties:

- Hiding: The commitment com reveals no information to the receiver about the message m.
- Binding: The committer cannot efficiently produce a commitment com that can be opened as two different messages m, m'.

We now present a formal definition of a secure commitment scheme as given in [Katz20]. In order to define the security of a commitment scheme, we also need to define two experiments.

Definition 2.3.1 (**The Commitment Hiding Experiment** $Hiding_{\mathcal{A},Com}(n)$)**.** The steps of the experiment are the following:

1. Parameters $params \leftarrow Gen(1^n)$ are generated.
2. The adversary $\mathcal{A}$ is given input $params$, and outputs a pair of messages $m_0, m_1 \in \{0,1\}^n$.

3. A uniform $b \in \{0,1\}$ and randomness r are chosen and $com \leftarrow Com(params, m_b, r)$ is computed.

4. The adversary $\mathcal{A}$ is given com and outputs a bit b'.

5. The output of the experiement is 1 if and only if $b' = b$.

Definition 2.3.2 (**The Commitment Binding Experiment** $Binding_{\mathcal{A},Com}(n)$)**.** The steps of the experiment are the following:

1. Parameters $params \leftarrow Gen(1^n)$ are generated.

2. $\mathcal{A}$ is given input $params$ and outputs (com, m, r, m', r').

3. The output of the experiment is defined to be 1 if and only if $m \neq m'$ and $Com(params, m; r) = com = Com(params, m'; r')$.

Definition 2.3.3. A **commitment scheme** Com is **secure** if for all PPT adversaries $\mathcal{A}$ there is a negligible function $negl$ such that:

$$Pr[Hiding_{\mathcal{A},Com}(n) = 1] \leq \frac{1}{2} + negl(n)$$

$$Pr[Binding_{\mathcal{A},Com}(n) = 1] \leq negl(n)$$

Constructing a secure commitment scheme, assuming the existence of a "good" collision resistant hash function is fairly easy. Let $H : \{0,1\}^* \to \{0,1\}^n$ be such a function. In order to commit to a message m, the commiter samples a uniform $r \leftarrow \{0,1\}^n$ and outputs $H(m\|r)$. The binding property is directly derived from the collision resistance of H, however this commitment scheme is not necessarily hiding as a collision resistant hash function might still leak information about some input bits. That's why we need to use a secure hash function that resembles a random function, as required by the modern standards of hashing.

2.4 Accumulators

An accumulator is a cryptographic construction that allows us to combine a large set of values into one, short value, which is called the accumulated value. After that, we can efficiently verify the membership of any element in the set by generating a short witness.

We now present a formal definition of an accumulator as given in [Goel22].

Definition 2.4.1. Let n be the security parameter. A **secure accumulator** for inputs in $\mathcal{Y}$ is a tuple of the four PPT algorithms $(Acc.Gen, Acc.Eval, Acc.Proof, Acc.Verify)$ defined as follows:

- $Acc.Gen(1^n, m)$: On input 1^n and the number of values that can be securely accumulated m, this algorithm returns a key key.

- $Acc.Eval(key, Y)$: On input the key, and accumulation set $Y = \{y_1, \ldots, y_{m'}\} \in \mathcal{Y}^{m'}$, where $m' \leq m$, this algorithm returns a an accumulated value z and auxiliary information aux.

- $Acc.Proof(key, y, z, aux)$: On input key, a value y, an accumulated value z of some set Y, and some auxiliary information aux, this algorithm returns a proof π if $y \in Y$, else it returns the special symbol $\perp$.

- $Acc.Verify(key, y, z, \pi)$: This is a deterministic algorithm that takes in a key key, a value y, a proof π and an accumualted value z, and returns a bit $b \in \{0,1\}$.

These algorithms satisfy the following properties:

- Completeness: For any input set $Y = \{y_1, \ldots, y_{m'}\} \in \mathcal{Y}^{m'}$, and any $i \in [m']$, it holds that:

$$Pr[Acc.Verify(key, y_i, z, w) = 1 | key \leftarrow Acc.Gen(1^n, m), m' \leq m, \\ (z, aux) \leftarrow Acc.Eval(key, Y), \\ w \leftarrow Acc.Proof(key, y_i, z, aux)] = 1$$

- Soundness: No PPT adversary $\mathcal{A}$, can win the following game with more than negligible probability (in n):
  1. The challenger samples $key \leftarrow Acc.Gen(1^n)$ and sends it to $\mathcal{A}$.
  2. $\mathcal{A}$ responds with $(\{y_j\}_{j \in [m']})$.
  3. The challenger computes $z, aux \leftarrow Acc.Eval(key, Y = \{y_1, \ldots, y_{m'}\})$ and sends (z, aux) to $\mathcal{A}$.
  4. $\mathcal{A}$ responds with a pair (y', w), and wins if $Acc.Verify(key, y', z, w) = 1$ and $y' \neq y_i$, for every $i \in [m']$.

2.4.1 Merkle Hash Proof System

Here we present the definition of the Merkle Hash Proof system [Merk87]. It is a fairly simple and commonly used accumulator that relies solely on hash functions.

Definition 2.4.2. A **Merkle hash proof system** corresponding to a hash function family $H^s : \{0,1\}^* \to \{0,1\}^\lambda$ is defined by 4 algorithms ($Merkle.Gen$, $Merkle.Hash$, $Merkle.Proof$, $Merkle.Verify$) as follows:

- $Merkle.Gen(1^n)$: On input 1^n, this algorithm returns a key key.
- $Merkle.Hash(key, x_1, \ldots, x_m)$: On input key key, vector $x_1, \ldots, x_m$, the Merkle hash algorithm computes and returns a hash y using a Merkle tree as follows:
  - For each $i \in [m]$, compute $y_i^0 = H(x_i)$.
  - For each $l \in [log(m)]$ and $i \in [\frac{m}{2^l}]$, compute $y_i^l = H^{key}(y_{2i-1}^{l-1} \| y_{2i}^{l-1})$. Set $y = y_1^{log(m)}$.
- $Merkle.Proof(key, x_1, \ldots, x_m, x_i)$: On input key, key, a vector $x_1, \ldots, x_m$, and an element x_i, the Merkle proof algorithm computes and outputs a proof π as follows:
  - For each $k \in [m]$, compute $y_k^0 = H^{key}(x_k)$.
  - For each $l \in [log(m)]$ and $k \in [\frac{n}{2^l}]$, compute $y_k^l = H^{key}(y_{2k-1}^{l-1} \| y_{2k}^{l-1})$.
  - Initialize $\pi = \{(i, sibling(y_i^0))\}$ and for each $l \in [log(m)]$, set $\pi = \pi \cup \{(\lceil \frac{i}{2^l} \rceil, sibling(y^l_{\lceil \frac{i}{2^l} \rceil}))\}$.
- $Merkle.Verify(key, x_i, y, \pi)$: On input key, key,an element x_i, Merkle hash y, and a Merkle proof π, the Merkle verification algorithm parses $\pi = ((i_0, x^0), \ldots, (i_{log(m)}, x^{log(m)}))$ and returns $\{0,1\}$, proceeding as follows:
  - If i_0 is an even number, compute $y^1 = H^{key}(H^{key}(x_i) \| x^0)$, else compute $y^1 = H^{key}(x^0 \| H^{key}(x_i))$.
  - For each $l \in [log(m)]$, if i_l is an even number, compute $y^l = H^{key}(H^{key}(y^{l-1}) \| x^l)$, else compute $y^l = H^{key}((x^l \| H^{key}(y^{l-1}))$.
  - If $y^{log(m)} = y$, output 1, else output 0.

A Merkle hash proof system has the following properties.

Theorem 2.4.1. *The Merkle hash proof system satisfies the following properties:*

- *Completeness: For any input string* $x_1, \ldots, x_m \in \{0,1\}^{m\lambda}$ *and* $i \in [m]$*, it holds that:*

$$Pr[Merkle.Verify(key, x_i, y, \pi) = 1 | key = Merkle.Gen(1^n) \\ y = Merkle.Hash(x_1, \ldots, x_m) \\ \pi = Merkle.Proof(x_1, \ldots, x_m, x_i)] = 1$$

- *Soundness: No PPT adversary* $\mathcal{A}$*, can win the following game with more than negligible probability (in* n*):*

  1. *The challenger samples* $key = Merkle.Gen(1^n)$*.*
  2. $\mathcal{A}$ *responds with* $(i \in [m], \{y_j\}_{j \in [m] \setminus \{i\}})$*.*
  3. *The challenger samples a uniform* $x_i \{0,1\}^\lambda$*, computes* $y = Merkle.Hash(key, x_1, \ldots, x_m)$ *and sends* x_i, y *to* $\mathcal{A}$*.*
  4. $\mathcal{A}$ *responds with a pair* (x', π)*, and wins if* $Merkle.Verify(key, x', y, \pi) = 1$ *and* $x' \neq x_i$ *for every* $i \in [m]$*.*

Chapter 3

Complexity Background

In this chapter we will provide some background on complexity theory that is necessary for the rest of this thesis. Specifically, we will define zero knowledge arguments of knowledge [Gold89] and present the MPC-in-the-head technique [Isha09], which can be used to construct zero knowledge proofs from any secure protocol for multi-party computation (MPC). We will also present the MPC protocol of [Kale22], which we use in our MPC-in-the-head proof to prove nonlinear relations. Finally, we present some background knowledge and important results of fine-grained complexity theory.

3.1 Zero Knowledge Proofs and Arguments

Zero knowledge proofs and arguments have been a fundamental tool of cryptography since they were introduced in [Gold89]. They enable a prover to convince a verifier that a mathematical statement is true without revealing any additional information. In order to define zero knowledge proofs and arguments, we first need to describe interactive proofs.

Let $\mathbb{F}$ be a finite field and $\mathcal{R} : \mathbb{F}^n \times \mathbb{F}^m \to \{0,1\}$ be a binary relation, decidable in polynomial time. A k-message interactive proof system (IP) for $\mathcal{R}$ consists of a probabilistic polynomial time verifier algorithm $\mathcal{V}$ and a prescribed prover algorithm $\mathcal{P}$. Both $\mathcal{V}$ and $\mathcal{P}$ are given a common input $x \in \mathbb{F}^n$ and $\mathcal{P}$ is additionally given input $w \in \mathbb{F}^m$ such that $\mathcal{R}(x,w) = 1$. We call x a statement and w the corresponding witness. Then $\mathcal{P}$ and $\mathcal{V}$ alternately exchange a sequence of messages $m_1, \ldots, m_k$. For both $\mathcal{P}$ and $\mathcal{V}$, the next message of each round is computed as a function of x and all the previously sent messages. That is, in round i: $m_i = f(x, m_1, \ldots, m_{i-1})$. After all k rounds, the verifier outputs a bit $b \in \{0,1\}$. If $b = 0$, the verifier rejects, otherwise it accepts the prover's claim. The messages $(m_1, \ldots, m_k)$ exchanged by $\mathcal{P}$ and $\mathcal{V}$ along with the answer b are called a transcript. We denote $(\mathcal{P}(x,w), \mathcal{V}(x)) \in \{0,1\}$ the output of verifier $\mathcal{V}$ on input x when interacting with prover strategy $\mathcal{P}$ (that has a witness w as private input).

Definition 3.1.1. A **zero knowledge argument of knowledge** for $\mathcal{R}$ with completeness error ϵ_c and soundness error ϵ_s is an interactive protocol between a prover algorithm $\mathcal{P}$ and a verifier algorithm $\mathcal{V}$ that satisfies the following properties:

- Completeness: If $(x,w) \in \mathcal{R}$, and $\mathcal{P}$ knows a witness w for x, they will succeed in convincing $\mathcal{V}$ except with probability ϵ_c, i.e.,

$$Pr[(\mathcal{P}(x,w), \mathcal{V}(x)) = 1] \geq 1 - \epsilon_c$$

- Soundness: If a PPT algorithm $\tilde{\mathcal{P}}$ does not possess a witness w for x, they cannot succeed in convincing $\mathcal{V}$ except with probability ϵ_s, i.e.,

$$Pr[(\tilde{\mathcal{P}}(x), \mathcal{V}(x)) = 1] \leq \epsilon_s$$

- Zero Knowledge: For every PPT algorithm $\tilde{\mathcal{V}}$, there exists a PPT simulator algorithm $\mathcal{S}$ which, given the input statement x and rewindable black-box access to $\tilde{\mathcal{V}}$, outputs a simulated transcript such that

$$\mathcal{S}(x, \tilde{\mathcal{V}}(\cdot)) \equiv View(\mathcal{P}(x, w), \tilde{\mathcal{V}}(x))$$

  We can relax perfect zero knowledge to computational zero knowledge by requiring the output of the simulator $\mathcal{S}$ to be (t, ζ)-indistinguishable from $View(\mathcal{P}(x, w), \tilde{\mathcal{V}}(x))$. In our proofs we achieve computational zero knowledge.

In our protocols, it holds that $\epsilon_c = 0$, thus a prover $\mathcal{P}$ who possesses a witness w for x will always convince the verifier.

Furthermore, our protocols satisfy a stronger notion of soundness, called special soundness. That means, there exists an algorithm which, given some transcripts of the protocol with the same first commitment from the prover but different challenges from the verifier, can output a valid witness for the statement. Such an algorithm is called a knowledge extractor. We now provide a formal definition [Fene22b]:.

Definition 3.1.2 (**Special Soundness**)**.** Let ε_s denote the soundness error of the zero knowledge argument. If there exists a PPT algorithm $\tilde{\mathcal{P}}$ that does not possess a witness w for x, that successfully convinces $\mathcal{V}$ with probability $\tilde{\epsilon} > \epsilon_s$, i.e.,

$$\tilde{\epsilon} := Pr[(\tilde{\mathcal{P}}(x), \mathcal{V}(x)) = 1] > \epsilon_s,$$

then there exists an extractor algorithm $\mathcal{E}$ which, given rewindable black box access to $\tilde{\mathcal{P}}$, outputs a witness w' for x in time $poly(n, (\tilde{\epsilon} - \epsilon)^{-1})$ with probability at least $\frac{1}{2}$.

Finally, our protocols use a cut-and-choose approach to prove set membership, so we can only achieve a weaker notion of zero knowledge, called honest verifier zero knowledge, i.e., we only achieve the zero knowledge property against verifiers who follow the protocol. A formal definition follows:

Definition 3.1.3 (**Honest Verifier Zero Knowledge**)**.** There exists a PPT simulator algorithmm $\mathcal{S}$ which, given the input statement x, outputs a simulated transcript which is (t, ζ)-indistinguishable from $View(\mathcal{P}(x, w), V(x))$.

3.2 MPC-in-the-head

We will now present the MPC-in-the-head technique [Isha09] which allows us to create zero knowledge arguments from secure multi-party computation protocols.

3.2.1 Secure Multi-Party Computation

Secure multi-party computation is a subfield of cryptography with the goal of creating methods for parties to jointly compute a function of their inputs without revealing any information about them to the other parties. They have found numerous applications in cryptography, thus they have been extensively studied since their introduction in [Yao86].

We will now mention some properties that an MPC protocol needs to satisfy so that it can be compiled to a zero knowledge proof [Isha09]. Let $P_1, \ldots, P_N$ denote the N parties of the protocol. We will assume that the parties intend to verify the validity of an NP relation $\mathcal{R}(x, w)$. Each party P_j has access to the public statement x and a secret share of the witness $[\![w]\!]_j$ and all parties use a secure MPC protocol Π to compute $\mathcal{R}(x, [\![w]\!])$. Each party's P_j next messages are computed by a function that takes as input its index j, the statement x, its witness share $[\![w]\!]_j$, its random coins r_j and the messages that it received in the previous

rounds. The function returns a tuple of $N-1$ messages, one for each of the other parties. Finally we denote as $View_j$ to constitute the P_j's private input $[\![w]\!]_j$, its randomness r_j and the set of messages sent and received by it during the execution of the protocol.

Definition 3.2.1. We say that the views $View_i$, $View_j$ of the parties P_i, P_j respectively are **consistent** (with respect to a protocol Π and common input x) if the messages received by P_j from P_i are the same as the messages received by P_i from P_j and vice versa.

We now present an important lemma for secure MPC. It states that the views of every pair of parties is locally consistent (with respect to some protocol Π and input x) if and only if it is globally consistent with respect to the same protocol Π.

Lemma 3.2.1. Let $View_1, \ldots, View_N$ be a set of views. The views $View_i$, $View_j$ for every $i, j \in [N]$ are consistent (with respect to a protocol Π and common input x) if and only if there exists an honest execution of the protocol Π on a common input x and private inputs w_i for party P_i where the view of the party P_i corresponds to $View_i$.

Proof. The If part of the proof follows directly from the definition of protocol Π. We now prove the only-if part. We prove it using the induction on the number of rounds. The base case is when the protocol Π has only 1 round in which case the assertion follows directly. Let us assume that assertion is true for d rounds. Let $[\![w]\!]_i$ and r_i be the private input and randomness in $View_i$. In the $(d+1)$-th round every party P_i generates a message to every other party P_j based on the protocol Π, the input x, private input $[\![w]\!]_i$, randomness r_i and the set of messages $(m_1, \cdots, m_d)$ it has received in the first d rounds (follows from local consistency constraint, i.e, views of parties P_i and P_j are consistent with a protocol Π). Since for the first d rounds the messages are equivalent in the local view and the global view, the $(d+1)$-th message in the local view is equivalent to the $(d+1)$-th message in the global view. □

Definition 3.2.2 (Correctness). We say that the protocol Π perfectly computes the functionality $\mathcal{R}$ if for all common input x and private inputs $[\![w]\!]_1, \ldots, [\![w]\!]_N$ the output of each party P_i, where $i \in [N]$, is equal to $\mathcal{R}(x, [\![w]\!])$ with probability 1, where the probability is over the random inputs of each party.

Definition 3.2.3 (t-privacy). Let $1 \leq t < N$. We say that Π realizes $\mathcal{R}$ with perfect t-privacy if there exists a simulator $\mathcal{S}$ such that for any inputs $x, [\![w]\!]_1, \ldots, [\![w]\!]_N$ and every set of corrupted players $T \subset [N]$ such that $|T| \leq t$, the joint view of players in T is distributed identically to the output of the simulator, i.e.,

$$\mathcal{S}(T, x, \{[\![w]\!]_i\}_{i \in [T]}, \mathcal{R}(x, [\![w]\!])) \equiv View_T(x, [\![w]\!]_1, \ldots, [\![w]\!]_N)$$

We can relax perfect t-privacy to computational t-privacy by requiring the output of the simulator to be (t, ζ)-indistinguishable from $View_T$.

Definition 3.2.4 (t-robustness). We say that Π realizes $\mathcal{R}$ with perfect t-robustness if it is perfectly correct in the presence of a semi-honest adversary as in definition 3.2.2 and furthermore for any computationally unbounded malicious adversary corrupting a set T of at most t players, and for any inputs $(x, [\![w]\!]_1, \ldots, [\![w]\!]_N)$, the following robustness property holds. If there is no $([\![w]\!]'_1, \ldots, [\![w]\!]'_N)$ such that $(\mathcal{R}(x, [\![w]\!]'_1, \ldots, [\![w]\!]'_N) = 1$, then the probability that some uncorrupted player outputs 1 in an execution of Π in which the inputs of the honest are consistent with $(x, [\![w]\!]_1, \ldots, [\![w]\!]_N)$ is 0.

3.2.2 Additive Secret Sharing

In our protocols, we will share the witness $w \in \mathbb{F}^m$ among the N parties by using an additive secret sharing as follows:

$$\begin{cases} [\![w]\!]_i \xleftarrow{\$} \mathbb{F}^m \text{ for all } i \in [N-1] \\ [\![w]\!]_N \leftarrow w - \sum_{i=1}^{N-1} [\![w]\!]_i \end{cases}$$

where the operations are performed over the field $\mathbb{F}$.

We note that the additive secret sharing is $(N-1)$-private, i.e., an adversary who has corrupted at most $N-1$ parties can't get any information about the secret.

3.2.3 MPC-in-the-head Based Protocols

We will now show how to compile a secure MPC protocol into a zero knowledge argument for a relation $\mathcal{R}(x, w)$ that recognises an NP language $\mathcal{L}$.

Let Π be an N-party secure MPC protocol and P be the set of parties. Let $View_i$ denote the view of the party $P_i \in P$ in Π. The parties and the prover get a statement $x \in \mathcal{L}$ as input, also the prover gets a witness w, as an additional input, such that $\mathcal{R}(x, w) = 1$. The protocol proceeds as follows:

- **Round 1:** The prover uses the additive secret sharing of 3.2.2 to create N shares of w and shares them among the N parties. Then, it emulates an execution of Π "in its head" for the function $\mathcal{F}([\![w]\!]) = \mathcal{R}(x, \oplus_{i\in[N]}[\![w]\!]_i)$. The prover makes use of a secure commitment scheme Com to compute a commitment com_i to the view $View_i$ of each party P_i. Then it sends $com_i := Com(View_i; r_i)$ to the verifier for each $i \in [N]$.

- **Round 2:** Assume the underlying MPC protocol is t-private. The verifier samples a uniform set of indexes $\mathcal{I} \subset [N]$ such that $|\mathcal{I}| \leq t$ and sends it to the prover.

- **Round 3:** The prover opens all the commitments to the parties' views that belong in $\mathcal{I}$ and sends them to the prover.

  The verifier verifies that all the opened views are consistent among each other and that no party rejects. If the check succeeds, the verifier accepts and outputs 1, otherwise it outputs 0.

The completeness property of the above protocol directly follows from its construction. We will now argue about its soundness and zero knowledge properties.

Soundness: The protocol's soundness heavily relies on the number of malicious parties that are required to impact the protocol's output. A lower bound of this number is guaranteed by the robustness property of the underlying MPC protocol. In order to achieve a negligible soundness error, we may need to amplify the protocol's soundness by performing multiple iterations. If no iteration results in rejection, the verifier outputs 1.

Zero Knowledge: The zero knowledge property of the protocol can be proved by constructing a PPT simulator which, given input a statement x, outputs a transcript that is computationally indistinguishable from a transcript obtained by running the real protocol. In order to achieve that, it samples a uniform set $\mathcal{I} \subset [N]$ and generates the views $\{View_i\}_{i\in\mathcal{I}}$ by using the simulator of the underlying MPC protocol and uniformly random private inputs $\{[\![w]\!]_i\}_{i\in\mathcal{I}}$. For the parties that don't belong in $\mathcal{I}$, the simulator can commit to any value, e.g., $View_i = 0$ as those commitments are never opened. Thus, the computational zero knowledge property of the compiled protocol follows from the privacy property of the underlying MPC and the hiding property of the commitment scheme.

3.2.4 Batch Product Verification

Verifying linear relations over a field $\mathbb{F}$ using MPC-in-the-head is not challenging. The parties can simply perform all the necessary operations locally using their private additive shares, and then broadcast their share of the final result. The result can easily be computed by adding all the broadcasted shares. This simple case is demonstrated in our zero knowledge protocol for $3SUM$.

Handling non-linear operations (e.g., multiplications in $\mathbb{F}$) is more complicated and several approaches have been proposed to overcome this obstacle. Recently, the authors of [Lind17, Baum20] constructed an MPC protocol to verify the correctness of a product in $\mathbb{F}$ by "sacrificing" another one. Their construction enables that a triple of sharings $([\![x]\!], [\![y]\!], [\![z]\!])$ is such that $x \cdot y = z$, by using a second random triple $([\![a]\!], [\![b]\!], [\![c]\!])$ satisfying $a \cdot b = c$. The second triple can only be used once to preserve the zero knowledge property.

Much more recently [Kale22], this method was optimized to an efficient MPC protocol which checks simultaneously many products by sacrificing a single dot product. Specifically, given n triples $([\![x_j]\!], [\![y_j]\!], [\![z_j]\!])$ and a tuple $(([\![a_j]\!])_{j\in[n]}, [\![c]\!])$, their protocol verifies that $\langle a, y\rangle = -c$ and $z_j = x_z \cdot y_j$ for all $j \in [n]$, without revealing any information on (x, y, z).

We now present the optimized MPC protocol of [Kale22] which we use (in an MPC-in-the-head context) to prove orthogoniality in our zero knowledge protocol for OV:

1. The verifier samples a uniform $\epsilon \in \mathbb{F}^n$ and sends it to the parties.
2. Each party P_i locally sets $[\![e_j]\!]_i = \epsilon[\![x_j]\!]_i + [\![a_j]\!]_i$, for all $j \in [n]$.
3. The parties open e by broadcasting their shares.
4. Each party P_i locally sets $[\![v]\!]_i = \langle \epsilon, [\![z]\!]_i\rangle - \langle e, [\![y]\!]_i\rangle - [\![c]\!]_i$.
5. The parties open v by broadcasting their shares.
6. The parties accept if and only if $v = 0$.

The compilation of this protocol into an MPC-in-the-head based zero knowledge argument can be done easily since the tuple $(([\![a_j]\!])_{j\in[n]}, [\![c]\!])$ can be computed locally by the parties. Specifically, for all $j \in [n]$, a_j needs to be a uniform element of $\mathbb{F}$, thus, each party can locally compute their share $[\![a_j]\!]_i$ by sampling a uniform element $[\![a_j]\!]_i \xleftarrow{\$} \mathbb{F}$. The parties can also compute their share $[\![c_j]\!]_i$ by sampling a uniform element $[\![c_j]\!]_i \xleftarrow{\$} \mathbb{F}$. The prover can then introduce an auxiliary variable Δ_c to correct the resulting error and send it to the verifier.

We will now show that that if $([\![x_j]\!], [\![y_j]\!], [\![z_j]\!])_{j\in[n]}$ contains an incorrect multiplication triple (i.e., there exists a j_0 such that $x_{j_0} \cdot y_{j_0} \neq z_{j_0}$) or if $(([\![a_j]\!])_{j\in[n]}, [\![c]\!])$ does not satisfy the relation $\langle a, y\rangle = -c$, then the parties accept with a probability at most $\frac{1}{|\mathbb{F}|}$ [Kale22]. We will first present the Schwartz-Zippel lemma [Schw80, Zipp79].

Lemma 3.2.2 (General Schwartz-Zippel Lemma). Let $P(x_1, \ldots, x_n)$ be a non-zero polynomial of n variables with total degree d over $\mathbb{F}$. For any finite subset S of $\mathbb{F}$, with at least d elements in it,

$$Pr[(r_1, \ldots, r_n) \xleftarrow{\$} S^n : P(r_1, \ldots, r_n) = 0] \leq \frac{d}{|S|}$$

Proof. We prove the lemma by induction on n. When $n = 1$, the result follows from the fact that any non-zero degree d polynomial in one variable has at most d roots. Thus $P(r) = 0$ if and only if r is a root, which happens with probability at most $\frac{d}{|S|}$.

For the general case, let us write the polynomial in the form

$$P(x_1, \ldots, x_n) = x_n^l \cdot Q(x_1, \ldots, x_{n-1}) + R(x_1, \ldots, x_n),$$

where R is a polynomial in which the degree of x_n is at most $l-1$. Now let E_1 be the event that $P(r_1,\ldots,r_n)=0$, and let E_2 be the event that $Q(r_1,\ldots,r_n)=0$. Then we have:

$$\begin{aligned} Pr[E_1] &= Pr[E_1 \wedge E_2] + Pr[E_1 \wedge \neg E_2] \\ &= Pr[E_2] \cdot Pr[E_2|E_1] + Pr[\neg E_2] \cdot Pr[E_1|\neg E_2] \\ &\leq Pr[E_2] + Pr[E_1|\neg E_2]. \end{aligned}$$

By induction, since Q is a degree $d-l$ polynomial, $Pr[E_2] \leq \frac{d-l}{|S|}$. Since $x_1,\ldots,x_n$ are fixed in $\neg E_2$, we have that $P(r_1,\ldots,r_{n-1},x_n)$ is a non-zero polynomial of degree l, so $Pr[E_1|\neg E_2] \leq \frac{l}{|S|}$. Thus $Pr[E_1] \leq \frac{d}{|S|}$. □

Lemma 3.2.3. If the secret shared input $(\llbracket x_j \rrbracket, \llbracket y_j \rrbracket, \llbracket z_j \rrbracket)_{j\in[n]}$ contains an incorrect multiplication triple, or if $((\llbracket a_j \rrbracket, \llbracket y_j \rrbracket), \llbracket c \rrbracket)_{j\in[n]}$ form an incorrect dot product, then the parties accept with probability at most $\frac{1}{|\mathbb{F}|}$.

Proof. Let $\Delta_{z_j} = z_j - x_j \cdot y_j$ and $\Delta_c = c + \sum_{j=1}^{n} a_j \cdot y_j$. If the parties accept, then $v=0$, leading to:

$$\begin{aligned} v &= -c + \sum_{j=1}^{n} \epsilon_j z_j - e_j y_j \\ &= -c + \sum_{j=1}^{n} \epsilon_j \cdot (\Delta_{z_j} + x_j y_j) - (\epsilon_j x_j + a_j) \cdot y_j \\ &= -c + \sum_{j=1}^{n} \epsilon_j \Delta_{z_j} + \epsilon_j x_j y_j - \epsilon_j x_j y_j + a_j y_j \\ &= -c + \sum_{j=1}^{n} a_j y_j + \epsilon_j \Delta_{z_j} \\ &= \epsilon_1 \Delta_{z_1} + \epsilon_2 \Delta_{z_2} \cdots + \epsilon_n \Delta_{z_n} + \Delta_c = 0 \end{aligned}$$

We now define a multivariate polynomial:

$$Q(X_1,\ldots,X_n) = X_1 \Delta_{z_1} + \cdots + X_n \Delta_{z_n} + \Delta_c$$

in $\mathbb{F}[X_1,\ldots,X_n]$ and note that $v=0$ if and only if $Q(\epsilon_1,\ldots,\epsilon_n)=0$. When Q is the zero polynomial, then $\Delta_{z_j}=0$ for all $j\in[n]$ and $\Delta_c=0$, implying $z_j = x_j \cdot y_j$ and $c = -\sum_{j=1}^{n} a_j \cdot y_j$, so $v=0$ is the correct result.

In the case of a cheating prover, Q is nonzero, and by the multivariate version of the Schwartz-Zippel lemma 3.2.2, the probability that $Q(\epsilon_1,\ldots,\epsilon_n)=0$ is at most $\frac{1}{|\mathbb{F}|}$, since Q has total degree 1 and $(\epsilon_1,\ldots,\epsilon_n)$ is chosen uniformly at random. □

Privacy: Now we show that the above protocol is $(N-1)$-private by constructing a simulator $\mathcal{S}$. The simulator takes as input the shares $(\llbracket z_j \rrbracket_i, \llbracket y_j \rrbracket_i, \llbracket c_j \rrbracket_i)_{j\in[n]}$ for all parties $i\in[N]$ except one. We denote that one party by i^*. $\mathcal{S}$ behaves as follows:

1. $\forall i \in [N].\forall j \in [n]$: Sample a uniform $\llbracket e_j \rrbracket_i \xleftarrow{\$} \mathbb{F}$
2. $\forall i \in [N] \setminus \{i^*\}$: Compute $\llbracket v \rrbracket_i = \langle \epsilon, \llbracket z \rrbracket_i \rangle - \langle e, \llbracket y \rrbracket_i \rangle - \llbracket c \rrbracket_i$.
3. Compute $\llbracket v \rrbracket_{i^*} = -\sum_{i=1, i\neq i^*}^{N} \llbracket v \rrbracket_i$

In a real execution of the protocol, $[\![e_j]\!]_i = \epsilon_j[\![x_j]\!]_i + [\![a_j]\!]_i$ for all $i \in [N], j \in [n]$ and, since $[\![a_j]\!]_i$ is a uniformly random element of $\mathbb{F}$, $[\![e_j]\!]_i$ is also uniformly random. The simulator computes $[\![e_j]\!]_i$ by sampling a uniform element of $\mathbb{F}$, thus, they follow the same distribution.

For all parties $i \in [N] \setminus \{i^*\}$, the shares $[\![v]\!]_i$ are computed exactly as in a real execution of the protocol.

Finally, there is only one possible choice for $[\![v]\!]_{i^*}$ that makes the parties accept, i.e., the one that satisfies $[\![v]\!]_{i^*} = -\sum_{i=1, i\neq i^*}^{N} [\![v]\!]_i$. It follows that $[\![v]\!]$ is distributed exactly as in a real execution of the protocol.

As we mentioned earlier, we use the above protocol to prove orthogoniality in our zero knowledge protocol for OV. We saw that the MPC protocol fails with a probability at most $\frac{1}{|\mathbb{F}|}$. We also saw that the protocol is $(N-1)$-private so, in order to turn this into a zero knowledge argument using the MPC-in-the-head technique, the verifier needs to verify all but one parties during the final verification round. Thus, the soundness error of this step is upper bounded by $\frac{1}{|\mathbb{F}|} + \frac{1}{N}$. We will present more details about the soundness of our protocol for OV in Chapter 5.

3.3 Fine-Grained Complexity

We will now present some background knowledge on fine-grained complexity, as well as some important results and assumptions.

Computational complexity has provided a simple, but still very useful, notion of "intractability" by introducing the theory of NP-completeness [Cook71, Karp72, Levi73]. A polynomial time reduction from any NP-complete problem to a problem Π is enough evidence that Π is unlikely to be solvable in polynomial time. However, there are many problems that lie in P, but still cannot be solved efficiently and despite many years of extensive algorithmic research, there have not been any improvements. The goal of fine-grained complexity is to provide evidence that a problem Π that can be solved in time, let's say, $O(n^k)$ also requires $n^{k-o(1)}$ time. This can be achieved via fine-grained reductions and some widely believed hypotheses and conjectures.

3.3.1 Fine-Grained Reductions

Since fine-grained reductions need to distinguish between different polynomial running times, they must be more right than regular polynomial time reductions. Specifically, for two problems A, B and running times $a(n), b(n)$, a fine-grained reduction should imply that an $O(b(n)^{1-\epsilon})$ algorithm for B would imply an $O(a(n)^{1-\delta})$ algorithm for A for some constant δ. We will now present a formal definition of fine-grained reductions [Vass15].

Definition 3.3.1 (**Fine-Grained Reduction**). Let $a(n)$ and $b(n)$ be nondecreasing functions of n. Problem A is (a, b)-reducible to problem B, denoted as $A \leq_{a,b} B$ if, for every $\epsilon > 0$, there is a $\delta > 0$, an algorithm F with access to an oracle B, a constand d, and for every $n \geq 1$ an integer $k(n)$, such that for every n, the algorithm F takes any instance of A of size n and

- F runs in at most $d \cdot (a(n))^{1-\delta}$ time,
- F produces at most $k(n)$ instances of B adaptively, that is, the j-th instance B_j is a function of $\{(B_i, a_i)\}_{1 \leq i < j}$ where B_i is the i-th instance produced and a_i is the answer of the oracle for B on instance B_i, and
- the sizes n_i of the instances B_i for any choice of oracle answers a_i obey the inequality $\sum_{i=1}^{k(n)} (b(n_i))^{1-\epsilon} \leq d \cdot (a(n))^{1-\delta}$.

A consequence of this definition is that any improvement over $b(n)$ for B implies an improvement over $a(n)$ for A. Specifically, if there is an algorithm for B of running time $c \cdot N^{1-\epsilon}$ on instances of size N, then composing the algorithm with a fine-grained (a, b)-reduction from A to B produces an algorithm for A running in time

$$d \cdot (a(n))^{1-\delta} + \sum_{i=1}^{k(n)} c \cdot (b(n_i))^{1-\epsilon} \leq d \cdot (c+1) \cdot (a(n))^{1-\delta},$$

where the first summand is for running the reduction algorithm F, and the second summand is for running the algorithm for B, on the instances $\{B_i\}$. As c and d are constants, the running time for A is $O(a(n)^{1-\delta})$.

3.3.2 Strong Exponential Time Hypothesis

The Strong Exponential Time Hypothesis (SETH)is one of the most popular conjectures in fine-grained complexity and relies on the hardness of solving k-SAT as k increases. The k-SAT problem asks the following: Given a formula on n boolean variables in k-CNF (i.e., the formula is a conjuction of clauses, where each clause is a disjunction of at most k literals, and each literal is a negated or unnegated variable) decide whether there exists a satisfying assignment (i.e., an assignment of each variable to true or false for which the formula evaluates to true).

A formal definition of SETH follows [Impa01a, Impa01b]:

First we need to define:

$$s_k = inf\{\delta \mid \text{there is an } O^*(2^{\delta n}) \text{ time algorithm for } k\text{-}SAT \text{ with } n \text{ variables}\},$$

where O^* hides any polynomial factors in n. SETH states that

$$\lim_{k \to \infty} s_k = 1$$

Equivalently, the following definition can be given:

Conjecture 1 (SETH). For every $\epsilon > 0$, there exists an integer k, such that SAT on k-CNF formulas on n variables cannot be solved in $O(2^{(1-\epsilon)n} poly(n))$ time in expectation.

3.3.3 Orthogonal Vectors

We will now define the Orthogonal Vectors problem (OV). It is one of the core problems of fine-grained complexity, because the fine-grained reduction from k-SAT to OV allows for many other lower bounds that are based on $SETH$.

Definition 3.3.2 (Orthogonal Vectors). Let U, V be two sets of vectors over $\{0,1\}^d$, such that $|U| = |V| = n$ and $d = \omega(logn)$. The OV problem asks to determine whether there exist $u \in U$ and $v \in V$ such that $\langle u, v \rangle = 0$.

The naive algorithm uses exhaustive search by testing each pair of vectors, thus achieving a complexity of $O(n^2 d) \leq \tilde{O}(n^2)$. The best algorithm for this problem is found in [Abbo14] and achieves a complexity of $O(n^{2-1/O(log(d/logn))})$. The lack of progress to create a truly subquadratic algorithm for OV has led to the following conjecture.

Conjecture 2 (OVC). In the Word RAM model with $O(logn)$ bit words, any algorithm requires $n^{2-o(1)}$ time in expectation to solve the OV problem.

We will now present the fine-grained reduction from k-SAT to OV [Will05].

Theorem 3.3.1 (k-$SAT \leq_{2^n,n^2} OV$)**.** *For each positive integer k, the k-SAT problem on n variables is $(2^n, n^2)$-reducible to OV on n vectors.*

Proof. Let F be a k-SAT formula on n variables. Making use of the sparsification lemma of [Impa01b], we can assume that F has $m = O(n)$ clauses, as this only gives a $2^{\alpha n}$ overhead for arbitrarily small $\alpha > 0$. We now create two sets U_1, U_2 of vectors over $\{0,1\}^m$ as follows:

1. Split the variables into sets V_1, V_2 of size $\frac{n}{2}$ each.
2. For each $i \in \{1,2\}$ and each partial assignment ϕ to the variables of V_i, add a vector $u_{i,\phi}$ to U_i where $u_{i,\phi}[c] = 1$ if ϕ does not satisfy clause c.

It now holds that $\langle u_{1,\phi}, u_{2,\psi} \rangle = 0$ if and only if for every clause c at least one of ϕ and ψ satisfies clause c. Thus, there exists an orthogonal pair if and only if ϕ and ψ together form a satisfying assignment. The number of vectors in the instances created is $N = O(2^{n/2})$ and the number of coordinates is $O(n) = O(logN)$. □

3.3.4 3SUM

$3SUM$ is another fundamental problem of fine-grained complexity, mainly because of its simple structure and its relation to a variety of core problems in computational geometry [Gaje95]. $3SUM$ is defined as follows:

Definition 3.3.3 (**3SUM**)**.** Let U, V, W be sets such that $|U| = |V| = |W| = n$ and $U, V, W \subset \{-m, \ldots, m\}$ for some $m = poly(n)$. $3SUM_m$ asks to determine whether there exist three elements u, v, w such that $u \in U$, $v \in V$, $w \in W$ and $u + v + w = 0$.

$3SUM$ can be solved by a simple $\tilde{O}(n^2)$ time algorithm by sorting the set W and then, for each pair of elements $(u, v) \in U \times V$, use binary search to check whether their negative sum $-(u + v) \in W$. The best known algorithm for $3SUM$ achieves a complexity of $O(n^2(loglogn)^2/log^2 n)$ [Bara08]. Even though truly subquadratic algorithms have been created for some structured instances of $3SUM$ [Chan15], there are still no $O(n^{2-\epsilon})$ time algorithms for the general problem. The lack of progress on that direction has led to the following conjecture [Gaje95, Patr10]:

Conjecture 3 (**No truly subquadratic 3SUM**)**.** In the Word RAM model with $O(logn)$ bit words, any algorithm requires $n^{2-o(1)}$ time in expectation to solve $3SUM$.

3.3.5 All-Pairs Shortest Paths

The All-Pairs Shortest Paths ($APSP$) problem is defined as follows:

Definition 3.3.4 (All-Pairs Shortest Paths)**.** Let G be a directed or undirected graph with integer edge weights. $APSP$ asks to determine the distances between every pair of vertices in G.

The standard approach of solving this problem would be to run a classical algorithm such as Floyd-Warshall's algorithm to achieve a running time of $O(n^3)$ in an n-node graph. The fastest known algorithm for $APSP$ has an exact running time of $n^3/\Theta(\sqrt{logn})$ and is described in [Will14]. The lack of progress towards solving $APSP$ in truly subcubic time (i.e., $O(n^{3-\epsilon})$) has led to the following conjecture [Rodi04, Will10]:

Conjecture 4 (**No truly subcubic APSP**)**.** There is a constant c such that in the Word RAM model with $O(logn)$ bit words, any algorithm requires $n^{3-o(1)}$ time in expectation to compute the distances between every pair of vertices in an n node graph

Chapter 4

Average Case Fine-Grained Hardness

In classical cryptography, there is a straightforward notion of hardness; problems that are solvable in polynomial time are considered to be easy and, thus, unusable for cryptographic constructions. The theory of NP completeness has identified many problems as intractable through polynomial time reductions. However, most of those reductions only prove hardness in the worst case, which is not enough for cryptographic applications. To combat this problem, the theory of average case complexity was proposed [Levi86, Bogd06]. Even though average case complexity perfectly captures the notion of hardness that is needed in cryptography, creating worst case to average case reductions is much more challenging. As a result, modern cryptography heavily relies on unproven assumptions such as the average case hardness of factoring, discrete logarithm etc.

The emergence of fine-grained cryptography has motivated developing an analogue of average case complexity in the fine-grained world. Specifically, we want to create families of functions that can be computed in some fixed polynomial time, but not less than that in the average case over some natural sampling method. It is suspected [Ball17a] that families of functions with the above properties can eventually lead to the construction of fine-grained one-way functions as defined in 2.1.4, which, in turn, could be the building blocks to create many other fine-grained cryptographic primitives.

In this chapter, we present some known average case fine-grained hard families of functions whose average case hardness relies on the core worst case assumptions of fine-grained complexity. Then, we present our own family of functions whose average case fine-grained hardness is based on a stronger assumption and show how we can use it to obtain a fine-grained Proof of Work.

4.1 $\mathcal{F}$OV, $\mathcal{F}$ZWT, $\mathcal{F}$TC

Before presenting some known results on average case fine-grained hardness we need to define the notion of average case complexity that was used by the authors of [Ball17a]. We later use the same definition for our family of functions.

Definition 4.1.1. A family of functions $\mathcal{F} = \{f_n\}$ is computable in time t on average if there is an algorithm that runs in $t(n)$ time on the domain of f_n and, for all large enough n, computes f_n correctly with probability at least $\frac{3}{4}$ over the uniform distribution of inputs in its domain.

The method used by the authors of [Ball17a] to create average case fine-grained hard families of functions was to express the core problems of fine-grained complexity as low degree polynomials and then use the random self-reducibility lemma of [Lipt91, Feig90, Gemm92] to obtain average case hardness.

We will now present their first family of polynomials. For any n, let $p(n)$ be the smallest prime number largest than n^2 and $d(n) = \omega(logn)$. They defined polynomials $fOV_n : \mathbb{F}_p^{2nd} \to \mathbb{F}_p$ over $2nd$ variables. The input of the polynomials represents an OV instance and is denoted by two matrices $U, V \in \mathbb{F}_p^{n \times d}$. fOV_n is defined as follows:

Definition 4.1.2.

$$fOV_n(U,V) = \sum_{i,j\in[n]} \prod_{l\in[d]} (1 - u_{il}v_{jl})$$

We note that, given as input an instance of OV of n vectors, fOV_n counts the number of orthogonal vectors. The family of polynomials $\mathcal{F}OV$ is defined as follows:

Definition 4.1.3.

$$\mathcal{F}OV = \{fOV_n\}$$

Then they proved the following theorem about the average case hardness of $\mathcal{F}OV$:

Theorem 4.1.1. *If $\mathcal{F}OV$ can be computed in $O(n^{1+\alpha})$ time on average for some $\alpha > 0$, then OV can be decided in $\tilde{O}(n^{1+\alpha})$ time in the worst case.*

The average case hardness of their next family of polynomials is based on the worst case hardness of the Zero-Weight Triangle (ZWT) problem which is the following:

Definition 4.1.4 (Zero-Weight Triangle)**.** Let G be an edge-weighted graph on n vertices where edge weights are in $\{-n^c, \ldots, n^c\}$ for some sufficiently large c. ZWT asks to determine whether there exists a triangle with edge weights w_1, w_2, w_3 in G such that $w_1 + w_2 + w_3 = 0$.

For any n, let $p(n)$ denote the smallest prime number larger than n^3 and let $d = \lceil log(2(2n^c + 1)) \rceil + 3$. Also, E denotes the set of $n^2 d$ variables that is the input of ZWT. The corresponding polynomial $fZWT_n : \mathbb{F}_p^{n^2 d} \to \mathbb{F}_p$ is the following:

Definition 4.1.5.

$$fZWT_n(E) = \sum_{i,j,k\in[n]} \prod_{l\in[d]} (1 - (s_l(w_{ij}, w_{jk}) - s_l(\overline{w}_{ik}, 0 \ldots 01))^2)$$

where $s_l : \mathbb{F}_p^{2d} \to \mathbb{F}_p$ is the polynomial such that if $x, y \in \{0,1\}^d$, then $s_l(x,y)$ equals the l-th bit of $(x + y)$.

Definition 4.1.6.

$$\mathcal{F}ZWT = \{fZWT_n\}$$

The following theorem holds:

Theorem 4.1.2. *If $\mathcal{F}ZWT$ can be computed in $O(n^{1.5+\alpha})$ time on average for some $\alpha > 0$, then ZWT can be decided in $\tilde{O}(n^{1.5+\alpha})$ time in the worst case.*

It is important to note that both $3SUM$ and $APSP$ reduce to ZWT [Will13, Patr10, Will10]. As a result, Theorem 4.1.2 holds if either the $3SUM$ or $APSP$ conjecture are true.

For their final family of polynomials, the authors of [Ball17a] reduced from the Triangle-Collection (TC) problem [Abbo15b] which is the following:

Definition 4.1.7 (Triangle-Collection)**.** Let G be an undirected tripartite node-coloured graph with n colours and $m = nlog^2 n + 2nlog^4 n$ nodes and with partitiions A, B, C of the form:

- A contains $nlog^2 n$ nodes $a_{l,i}$, where $i \in [n]$, $l \in [log^2 n]$ and $a_{l,i}$ is coloured with colour i.
- B (respectively C) contains $nlog^4 n$ nodes $b_{l,i,x}$ (respectively $c_{l,i,x}$) where $i \in [n]$, $l \in [log^2 n]$, $x \in [log^2 n]$ and $b_{l,i,x}$ (respectively $c_{l,i,x}$) is coloured with colour i.

- For each node $a_{l,i}$ and colours $j, k \in [n]$, there is exactly one edge from A to B of the form $(a_{l,i}, b_{l,j,x})$ and exactly one edge from A to C of the form $(a_{l,i}, c_{l,k,y})$, for some $x, y \in [log^2 n]$.
- A node $b_{l,j,x}$ can only be connected to nodes of the form $c_{l,k,y}$ in C.

TC asks to determine if, for all triples of distinct colours i, j, k, there is a triangle (u, v, w) in G where u has colour i, v has colour j, and w has colour k.

Let $p(n)$ denote the smallest prime number larger than n^3 and E denote the set of $m = (nlog^2 n + 2nlog^4 n)^2$ variables corresponding to the input of Triangle-Collection. The following polynomial $fTC_n : \mathbb{F}_p^m \to \mathbb{F}_p$ counts the number of triples that are violations to being a YES instance:

Definition 4.1.8.

$$fTC_n(E) = \sum_{1 \le i < j < k \le n} \prod_{\substack{l,x,y \in [log^2 n] \\ \pi \in S_{\{i,j,k\}}}} (1 - e_{a_{l,\pi(i)}, b_{l,\pi(j),x}} e_{a_{l,\pi(i)}, c_{l,\pi(k),y}} e_{b_{l,\pi(j),x}, c_{l,\pi(k),y}})$$

Definition 4.1.9.

$$\mathcal{F}TC = \{fTC_n\}$$

Similarly to the previous examples, the following theorem has been proven:

Theorem 4.1.3. *If $\mathcal{F}TC$ can be computed in $O(n^{1.5+\alpha})$ time on average for some $\alpha > 0$, then TC can be decided in $\tilde{O}(1.5 + \alpha)$ time in the worst case.*

It has been shown in [Abbo15b] that k-SAT, $3SUM$ and $APSP$ reduce to TC, thus Theorem 4.1.3 holds if either of $3SUM$, $APSP$ conjecture or $SETH$ holds.

4.2 $\mathcal{F}$EIP

It is noticeable from the previous section that reducing from problems that are further down in a chain of reductions is more beneficial, because hardness might remain even if one of the strongest assumptions is false. For example, even if $SETH$ doesn't hold, $\mathcal{F}OV$ remains hard to compute on average as long as OVC holds.

In this section, we present a family of polynomials, which we call $\mathcal{F}EIP$ that is hard to compute in truly subquadratic time on average, as long as the problem EIP_t (EIP_t), which will be defined later, cannot be solved in truly subquadratic time in the worst case. As we show below, there is a fine-grained reduction from OV to EIP_t, but, to the best of our knowledge, those problems have not be shown to be equivalent for $t > 0$. In [Chen19], it is shown that EIP and OV are equivalent in the sparse setting, where the dimension of the vectors is $d = O(logn)$. However, when $d = \omega(logn)$, which is the classical setting in fine-grained complexity, it is likely that EIP_t is even harder than OV. This implies that $\mathcal{F}EIP$ potentially remains average case fine-grained hard even if OVC is falsified, which makes it a more promising candidate for an average case fine-grained hard family of functions than $\mathcal{F}OV$.

We will now define EIP:

Definition 4.2.1 (Exact IP). Let U, V be two sets of vectors over $\{0,1\}^d$ and t be an integer, such that $|U| = |V| = n$, $d = \omega(logn)$ and $t \in [d]$. EIP_t asks to determine whether there exist $u \in U$ and $v \in V$ such that $\langle u, v \rangle = t$.

We will now show that there is a fine-grained reduction from OV to EIP_t:

Theorem 4.2.1 ($OV \leq_{n^2,n^2} EIP_t$)**.** *The OV problem on n vectors of dimension d is* (n^2, n^2)*-reducible to* EIP_t *on n vectors of dimension* $d + t$*.*

Proof. Let $U, V \in \{0,1\}^d$ be the input of OV. We define the gadget $g_{d,t} : \mathbb{F}_2^d \to \mathbb{F}_2^{d+t}$ as follows:

$$g_{d,t}(x) = x||1^t$$

We create sets U', V' as follows:

1. $U' = \emptyset$, $V' = \emptyset$
2. For each $u \in U$: $U' \leftarrow U' \cup \{g_{d,t}(u)\}$
3. For each $v \in V$: $V' \leftarrow V' \cup \{g_{d,t}(v)\}$

It is easy to see that OV on input U, V has a solution if and only if EIP_t has a solution on input U', V'. We note that $d = o(n)$ and the total running time of the reduction algorithm is $\tilde{O}(n)$, thus a truly subquadratic algorithm for EIP_t implies a truly subquadratic algorithm for OV. □

For our reduction, we use the following lemma on the random self-reducibility of evaluating low degree polynomials which was defined in [Lipt91, Feig90] and also used in [Ball17a]:

Lemma 4.2.1. Consider positive integers N, D, and p, and an $\epsilon \in (0, 1/3)$ such that $D > 9$, p is prime and $p > 12D$. Suppose that for some polynomial $f : \mathbb{F}_p^N \to \mathbb{F}_p$ of degree D, there is an algorithm A running in time t such that:

$$\Pr_{x \leftarrow \mathbb{F}_p^N}[A(x) = f(x)] \geq 1 - \epsilon$$

Then there is a randomized algorithm B that runs in time $O(ND^2 log^2 p + D^3 + tD)$ such that for any $x \in \mathbb{F}_p^N$:

$$Pr[B(x) = f(x)] \geq \frac{2}{3}$$

Proof. The algorithm B works as follows on input x:

- Sample uniformly random points $y_1, y_2 \in \mathbb{F}_p^N$ and define the curve $c(w) = x + wy_1 + w^2 y_2$ for $w \in \mathbb{F}_p$.
- For a value $m < p$ to be determined later, compute $(A(c(1)), \ldots, A(c(m)))$ to get $(z_1, \ldots, z_m) \in \mathbb{F}_p$.
- Run Berlekamp-Welch on $(z_1, \ldots, z_m)$. If it succeeds and outputs a polynomial $\hat{g}$, output $\hat{g}(0)$. Otherwise output 0.

To see why the above algorithm works, define the polynomial $g(w) = f(c(w))$. g is a polynomial of degree $2D$ over the single variale w, with the property that $g(0) = f(x)$. So if we had $2D + 1$ evaluations of g at different points in $\mathbb{F}_p$, we could retrieve g and compute $f(x)$. While we may not be able to obtain these evaluations directly (since they were involved in computing f), we do have access to A, which promises to be correct about the value of f on a random point with probability 2/3.

So we replace the values $g(w) = f(c(w))$ with $A(c(w))$, which will hopefully be correct at several points. Now our problem is to retrieve g given a set of its alleged evalutions at m points, some of which may be wrong.

We do this by interpreting $(g(1), g(2), \ldots, g(m))$ as a Reed-Solomon encoding of g and running its decoding algorithm on $(A(c(1)), \ldots, A(c(m)))$, which now corresponds to a corrupt codeword. The Berlekamp-Welch algorithm can do this as long as less than $(m - 2D)/2$ of

the m values of $A(c(w))$ are wrong. We now bound the probability of too many of these values being wrong.

Let Q_w be an indicator variable such that $Q_w = 1$ if and only if $A(c(w)) \neq f(c(w))$. Let $Q = \sum_{w=1}^{m} Q_w$. We note at this point the fact that over the randomness of y_1 and y_2, the distributions of $c(w)$ and $c(w')$ for any two distinct $w, w' \in \mathbb{F}_p$ are uniform and independent. This gives us the following statistics:

$$E[Q] = \epsilon m$$

$$Var[Q] = m\epsilon(1-\epsilon)$$

Thus, by the Chebyshev inequality, we have that the probability that more than a $\delta(> \epsilon)$ fraction of $A(c(w))$'s disagree with $f(c(w))$ is:

$$\begin{aligned} Pr[Q > \delta m] &\leq Pr[|Q - \epsilon m| > (\delta - \epsilon)m] \\ &\leq \frac{\epsilon(1-\epsilon)}{(\delta-\epsilon)^2 m} \leq \frac{1}{4(\delta-\epsilon)^2 m} \end{aligned}$$

We are interested in $\delta = \frac{m-2D}{2m} = \frac{1}{2} - \frac{D}{m}$. So if we set, for instance, $m = 12D$, the bound on the probability above is at most $1/3$ if $D > 9$ and $\epsilon < 1/3$.

So except with this small probability, the decoding algorithm correctly recovers g as $\hat{g}$, and consequently B computes $f(x)$ correctly.

Generating each $c(w)$ and running A on it takes $O(NDlog^2 p + t)$ time. The Berlekamp-Welch algorithm then takes $O(m^3)$ time, and the final evaluation of $\hat{g}$ takes $O(Dlog^2 p)$. Hence, given A with the above properties, the running time of B is:

$$O(m(NDlog^2 p + t) + m^3 + Dlog^2 p) = O(ND^2 log^p + D^3 + tD)$$

□

Using the above lemma, we can get get average case hardness by expressing problems as low degree polynomials. This requires tD to be the highest order term in the complexity expression, thus the degree of our polynomial needs to be a polylogarithmic function of the problem's input. We will now show how to construct a polynomial for EIP_t with the aforementioned properties.

For any n, let $p(n)$ (which we denote as p for brevity) be the smallest prime number bigger than n^2. We define polynomials $fEIP_{t,n} : \mathbb{F}_p^{2nd} \to \mathbb{F}_p$ over $2nd$ variables. The input of these polynomials represents an instance of EIP. The polynomial $fEIP_{t,n}$ is defined as follows:

Definition 4.2.2.

$$fEIP_{t,n}(U,V) = \sum_{i,j \in [n]} \prod_{l \in [\lceil logd \rceil]} f(t_l, b_l(\sum_{k \in [d]} u_{ik} v_{jk})),$$

where

- t_l denotes the l-th bit of the binary representation of t.
- b_l is a polynomial over $\mathbb{F}_p$ that computes the l-th bit of the binary representation of the first $logd$ elements of $\mathbb{F}_p$.
- $f(x_1, x_2) = x_1 x_2 + (1 - x_1)(1 - x_2)$

Remark. We note the following:

- b_l is a polynomial of degree $logd$ and can easily be constructed by using Lagrange interpolation over finite fields.
- Suppose $x_1, x_2 \in \{0,1\}$. In this case, $f(x_1, x_2)$ outputs 1 if and only if $x_1 = x_2$.

It is easy to see that the degree of $fEIP_{t,n}$ is polylogarithmic in n, thus it is a suitable low degree polynomial for EIP_t.

The corresponding family of polynomials is the following:

Definition 4.2.3.

$$\mathcal{F}EIP = \{fEIP_{t,n}\}$$

We will now prove that $\mathcal{F}EIP$ is average case fine-grained hard, in the same sense as in 4.1. In our proof, we will use the following lemma [Laga87]:

Lemma 4.2.2. The smallest prime number greater than m can be computed deterministically in $\tilde{O}(m^{1/2+\alpha})$ time for any $\alpha > 0$.

We can now prove the following theorem:

Theorem 4.2.2. *If $\mathcal{F}EIP$ can be computed in $O(n^{1+\alpha})$ time on average for some $\alpha > 0$, then EIP can be decided in $\tilde{O}(n^{1+\alpha})$ time in the worst case.*

Proof. Let A be an average case solver for $fEIP_{t,n}$ with a running time of $O(n^{1+\alpha})$. Here, we use the definition 4.1.1 for average case hardness, which implies that A fails to compute $fEIP_{t,n}$ on at most $\frac{1}{4}$ of the inputs. We will now construct an algorithm B that can decide EIP in the worst case. All operations are performed over $\mathbb{F}_p$, thus the algorithm needs to determine the value of p. $p(n)$ was defined to be the smallest prime number bigger than n^2, so it can be computed directly using Lemma 4.2.2 in time $\tilde{O}(n^{1+\alpha})$.

We now note that EIP_t is trivially reducible to computing $fEIP_{t,n}$. Let $U, V \in \{0,1\}^{2nd}$ be a valid input of EIP_t. It is easy to see that $fEIP_{t,n}(U,V)$ counts the number of pairs of vectors in $U \times V$ whose inner product equals t. It follows that the input (U,V) is a YES instance of EIP_t if and only if $fEIP_{t,n}(U,V) > 0$.

The only thing left to show is that A can be turned into a worst case algorithm. This can be achieved by using Lemma 4.2.1. This gives a probabilistic worst case algorithm with a running time of $\tilde{O}(n^{1+\alpha})$ □

Corollary 4.2.1. If EIP_t requires $n^{2-o(1)}$ time to decide, $\mathcal{F}EIP$ requires $n^{2-o(1)}$ time to compute on average.

We can also use theorems 3.3.1 and 4.2.1 to get the following (weaker) statements based on more popular assumptions:

Corollary 4.2.2. If SETH holds, $\mathcal{F}EIP$ requires $n^{2-o(1)}$ time to compute on average.

Corollary 4.2.3. If OV requires $n^{2-o(1)}$ time to decide, $\mathcal{F}EIP$ requires $n^{2-o(1)}$ time to compute on average.

4.3 An MA Protocol for $fEIP_{t,n}$

We will now use the general framework of [Will16] to get an MA protocol for $fEIP_{t,n}$. This framework has also been used in [Will16, Ball17a] to get an MA protocol for OV. In this protocol, the prover's goal is to convince the verifier that, for given (x,y), it holds that $fEIP_{t,n}(x) = y$. We remind that the input to $fEIP_{t,n}$ is two sets $U, V \in \mathbb{F}_p^{n \times d}$. We now define a supporting polynomial $fEIP_V$ as follows:

$$fEIP_V(x_1, \ldots, x_d) = \sum_{j \in [n]} \prod_{l \in [\lceil logd \rceil]} f(t_l, b_l(\sum_{k \in [d]} x_k v_{jk})),$$

where f, t_l, b_l are defined exactly as in Definition 4.2.2. This polynomial takes as input a vector $x \in \mathbb{F}_p^d$ and computes the number of vectors $v \in V$ such that $\langle x, v \rangle = t$. We can now write $fEIP_{t,n}$ as:

$$fEIP_{t,n}(U, V) = \sum_{i \in [n]} fEIP_V(u_i 1, \ldots, u_i d)$$

We define the polynomial $R_{U,V}$ as:

$$R_{U,V}(x) = fEIP_V(\phi_1(x), \ldots, \phi_d(x)),$$

where for $i \in [n]$, $\phi_l(i)$ is a polynomial such that $\phi_l(i) = u_{il}$. Finally, we write $fEIP_{t,n}$ as:

$$fEIP_{t,n}(U, V) = \sum_{i \in [n]} R_{U,V}(i)$$

The MA protocol is the following:

1. The prover sends the coefficients of a polynomial R^* that claims to be $R_{U,V}$ to the verifier.

2. The verifier samples a random $x \xleftarrow{\$} \mathbb{F}_p$ and computes the values $\{\phi_l(x)\}_{l \in [d]}$ by using fast interpolation techniques for multivariate polynomials [Horo72]. This procedure can be done in $\tilde{O}(n)$ time. Then, it uses those values to evaluate $fEIP_V$ and checks whether $R^*(x) \stackrel{?}{=} R_{U,V}(x)$. If the condition holds, the verifier uses R^* to compute $fEIP_{t,n}(U, V)$.

The field $\mathbb{F}_p$ is considerably larger than the degree of $R_{U,V}$, thus if the polynomials $R^*, R_{U,V}$ are different, the verifier rejects with high probability.

4.4 A Proof of Work Based on $\mathcal{F}EIP$

We will now use our average case hard family of polynomials $\mathcal{F}EIP$ to create a Proof of Work in the same manner as in [Ball17a]. This is also a proof of useful work as defined in [Ball17b], because in order to successfully respond to a challenge, the prover needs to solve an instance of Exact IP.

A proof of work scheme consists of three algorithms:

- $Challenge(1^n)$: It takes as input a difficulty parameter n and produces a challenge c of this difficulty.

- $Solve(c)$: It takes as input a challenge c and produces a solution s for that challenge.

- $Verify(c, s)$: It takes as input a challenge c and a solution s and checks whether s is a valid solution to the challenge c.

We now present the definition of a Proof of Work scheme [Ball17a]:

Definition 4.4.1. The triple of algorithms $(Challenge, Solve, Verify)$ is a Proof of Work scheme if the following properties are satisfied:

- $Challenge$ and $Verify$ are computable in time $s(n)$.

- $Solve$ is computable in time $t(n)$ such that $t(n) > s(n)$ and $Verify(c, Solve(c)) = 1$.

- There are constants $\epsilon, \epsilon' \in [0,1]$ such that for any A and n satisfying:

$$\Pr_{c \leftarrow Challenge(1^n)}[Verify(c, A(c)) = 1] \geq \epsilon$$

  it is the case that A takes time at least $t'(n)$ (such that $t'(n) > t(n)$) on a ϵ' fraction of challenges of difficulty n.

The algorithms of our Proof of Work scheme behave exactly the same as in the Proof of Work scheme of [Ball17a] which is based on the average case hardness of $\mathcal{F}OV$ and are described below:

- $Challenge(1^n)$: Sample a uniformly random input to $fEIP_{t,n}$:

$$(U, V) \xleftarrow{\$} \mathbb{F}_p^d \times \mathbb{F}_p^d$$

- $Solve(U, V)$: Output the coefficients of a polynomial R^* that he claims to be $R_{U,V}$ as defined in 4.3.

- $Verify((U, V), R^*)$: Sample a uniformly random $x \xleftarrow{\$} \mathbb{F}_p$ and check whether $R^*(x) \stackrel{?}{=} R_{U,V}(x)$.

The algorithms $Challenge, Verify$ can be computed in $\tilde{O}(n)$ time, also the output of $Solve$ can be computed in $\tilde{O}(n^2)$ time. As we mentioned in 4.3, the probability of convincing the verifier without computing the polynomial $R_{U,V}$ is very small, thus the average case hardness of $\mathcal{F}EIP$ implies that a prover who doesn't solve the Exact IP instance, cannot convince the verifier in this Proof of Work scheme on more than a $\frac{3}{4}$ fraction of inputs, unless there exists an $\tilde{O}(n^{2-\epsilon})$ time algorithm for worst-case Exact IP for some $\epsilon > 0$. This is an improvement over the Proof of Work scheme of [Ball17a] because, as we have already mentioned, Exact IP is likely to be harder than OV in the setting where $d = \omega(logn)$.

Chapter 5

Zero Knowledge In Fine-Grained Cryptography

Zero knowledge proofs and arguments are a fascinating result of complexity theory that has found numerous applications in cryptography. As presented in 3.1, in the classical setting, zero knowledge arguments have been defined to work with respect to a prover and a verifier, both of which are polynomial time algorithms. That restricts its usage only to problems that do not admit polynomial time algorithms, otherwise the verifier would be able to simply compute the witness on its own.

In fine-grained cryptography, hardness is no longer defined as our inability to create polynomial time algorithms, but as our inability to achieve running times lower than some fixed polynomial. Recent research towards achieving fine-grained cryptography is heavily focused on the hardness conjectures of fine-grained complexity. The core problems of fine-grained complexity (e.g., OV, $3SUM$, $APSP$) all admit quadratic or cubic time algorithms, thus creating zero knowledge arguments for those problems by using the classical definitions would be of no purpose.

In this chapter, we define a fine-grained analogue of zero knowledge arguments in such a way that zero knowledge for problems that lie in P is meaningful. Then, we provide zero knowledge arguments for the problems OV and $3SUM$. To the best of our knowledge, there are no zero knowledge protocols for those problems in the bibliography. Finally, we demonstrate how to use those protocols to get zero knowledge arguments for more problems.

5.1 Fine-Grained Zero Knowledge Arguments

We will now point out the differences between fine-grained zero knowledge arguments and traditional ones. As the name suggests, fine-grained zero knowledge arguments can be used for problems that are conjectured to have polynomial lower bounds, i.e., require time $n^{c-o(1)}$ for some integer $c > 0$. Similarly to the classical setting, the verifier needs to be computationally bounded in such a way that it would be infeasible to solve the problem on its own. If a problem requires $n^{c-o(1)}$ time to decide, we allow the verifier to be an $\tilde{O}(n^{c-\epsilon})$ algorithm for some $\epsilon > 0$. Completeness and soundness are defined in the same way as in standard zero knowledge arguments. For the zero knowledge property, we note that the simulator also needs to be an $\tilde{O}(n^{c-\epsilon})$ algorithm, otherwise it could start the simulation by computing the witness on its own. Finally, in the case of computational zero knowledge (which is the one that we use in our protocols), indistinguishability is only required against $\tilde{O}(n^{c-\epsilon})$ time bounded adversaries.

We are now ready to give a formal definition:

Definition 5.1.1 (**Fine-Grained Zero Knowledge Arguments**)**.** Let Π be a problem that is conjectured to require $n^{c-o(1)}$ time to decide for some integer $c > 0$ and $\mathcal{R}$ be its corresponding relation. A fine-grained zero knowledge argument for $\mathcal{R}$ with completeness error ϵ_c and soundness error ϵ_s is an interactive protocol between a prover algorithm $\mathcal{P}$ and a verifier algorithm $\mathcal{V}$ that satisfies the following properties:

- $\mathcal{V}$ requires at most $\tilde{O}(n^{c-\epsilon})$ time for some $\epsilon > 0$.

- Completeness: If $(x, w) \in \mathcal{R}$, and $\mathcal{P}$ knows a witness w for x, they will succeed in convincing $\mathcal{V}$ except with probability ϵ_c, i.e.,

$$Pr[(\mathcal{P}(x, w), \mathcal{V}(x)) = 1] \geq 1 - \epsilon_c$$

- Soundness: If $\tilde{\mathcal{P}}$ does not possess a witness w for x, they cannot succeed in convincing $\mathcal{V}$ except with probability ϵ_s, i.e.,

$$Pr[(\tilde{\mathcal{P}}(x), \mathcal{V}(x)) = 1] \leq \epsilon_s$$

- Zero Knowledge: For every $\tilde{O}(n^{c-\epsilon})$ algorithm $\tilde{\mathcal{V}}$, there exists an $\tilde{O}(n^{c-\epsilon})$ simulator algorithm $\mathcal{S}$ which, given the input statement x and rewindable black-box access to $\tilde{\mathcal{V}}$, outputs a simulated transcript such that

$$\mathcal{S}(x, \tilde{\mathcal{V}}(\cdot)) \equiv View(\mathcal{P}(x, w), \tilde{\mathcal{V}}(x))$$

Similarly to the classical definition of zero knowledge arguments, we can relax the notion of zero knowledge to computational zero knowledge by requiring $\mathcal{S}(x, \tilde{\mathcal{V}}(\cdot))$ to be $(\tilde{O}(n^{c-\epsilon}), \zeta)$-indistinguishable from $View(\mathcal{P}(x, w), \tilde{\mathcal{V}}(x))$ for every $\epsilon > 0$ and some negligible function ζ.

Finally we can define an analogue of special soundness for our case:

Definition 5.1.2. Let ϵ_s denote the soundness error of the zero knowledge argument. If there exists an algorithm $\tilde{\mathcal{P}}$ that does not possess a witness w for x, that successfully convinces $\mathcal{V}$ with probability $\tilde{\epsilon} > \epsilon_s$, i.e.,

$$\tilde{\epsilon} := Pr[(\tilde{\mathcal{P}}(x), \mathcal{V}(x)) = 1] > \epsilon_s,$$

then there exists an extractor algorithm $\mathcal{E}$ which, given rewindable black box access to $\tilde{\mathcal{P}}$, outputs a witness w' for x in time $\tilde{O}(n^{c-\epsilon} + (\tilde{\epsilon} - \epsilon)^{-1})$ with probability at least $\frac{1}{2}$.

Remark. At this point, we will make two notes about our definition:

- In the fine-grained setting, the verifier can guess the witness with a non-negligible probability. Our definition of the (computational) zero knowledge property ensures that the verifier doesn't gain any additional (non-negligible) advantage through the zero knowledge protocol.

- In our definition, we haven't specified any complexity requirements for the prover. This solely depends on the hash function used in the zero knowledge protocol to make the commitments. If we use a traditional hash function, then we can allow the prover to be any PPT algorithm. In case we use a fine-grained hash function, we need to bound the prover such that it can't "break" its security definition. Unfortunately, such a function has not been created yet, since there are some obstacles towards that direction, the most prominent, being the Nondeterministic Strong Exponential Time Hypothesis ($NSETH$) [Carm16, Ball17a]. In our protocol, we achieve a running time of $\tilde{O}(n)$ for both the verifier and the prover, thus we can use any hash function.

5.2 A Zero Knowledge Argument for OV

In this section we provide a zero knowledge argument for OV which satisfies definition 5.1.1.

5.2.1 Protocol Description

Let U, V be two sets of n d-dimensional vectors each, such that U, V is a YES instance of OV. This implies that there exist two vectors $u_\alpha \in U$, $v_\beta \in V$ such that $\langle u_\alpha, v_\beta \rangle = 0$. Suppose that the prover knows u_α, v_β. We will now present the high level idea behind our protocol and then proceed to explain each round in more detail.

The main technique of the protocol is MPC-in-the-head as presented in 3.2.3. Specifically, the prover splits u_α and v_β into N additive shares each and distributes them among N virtual parties $P_1, \ldots, P_N$. The parties now execute a secure MPC protocol to prove the following functionality:

$$\mathcal{F}([\![u_\alpha]\!], [\![v_\beta]\!]) := \langle u_\alpha, v_\beta \rangle = 0$$

This is done by running the batch product verification MPC protocol, which is presented in 3.2.4, to prove that for each $j \in [d]$: $u_{\alpha_j} \cdot v_{\beta_j} = 0$. This concludes the orthogoniality proof, however this naive approach fails because the prover can easily cheat by secret sharing two vectors u'_α, v'_β that do not belong in the inputs sets U, V respectively. To overcome this obstacle, it is essential to add a set membership proof in our protocol so that the parties can additionally prove that their secret shared values correspond to elements in U, V. We achieve this by using the core ideas of [Goel22]. The prover starts by masking each element of U, V and then accumulating those masked elements. This is done by sampling random masks $mask^{(U)}$, $mask^{(V)}$ and computing $\Delta^{(U)} = u - mask^{(U)}$ (respectively $\Delta^{(V)} = v - mask^{(V)}$) for each $u \in U$ (respectively $v \in V$). The prover then additively secret shares the masks and distributes them among the parties. In addition to the orthogoniality proof, the virtual parties of the MPC-in-the-head protocol need to run a secure MPC protocol to verify that their mask and vector shares correspond to a masked element of the accumulated input set by proving the following functionality:

$$\mathcal{F}([\![u_\alpha]\!], [\![v_\beta]\!], [\![mask^{(U)}]\!], [\![mask^{(V)}]\!]) := \langle u_\alpha, v_\beta \rangle = 0 \wedge \Delta^{(U)} = u_\alpha - mask^{(U)} \wedge \Delta^{(V)} = v_\alpha - mask^{(V)}$$

The prover also sends to the verifier the proof that the values $\Delta^{(U)}$, $\Delta^{(V)}$ have been accumulated. Each "package" that contains the mask shares and the accumulated values is called a preprocessing.

The only obstacle of this approach is that the verifier cannot check whether the mask shares and the accumulator have been computed honestly by the prover. This problem is tackled by using a cut-and-choose approach: Before the actual protocol starts, the prover creates multiple preprocessings, more than the number of preprocessings that will be consumed by the protocol (each parallel execution of the protocol consumes one preprocessing). The verifier then randomly chooses to open some of those preprocessings to check whether they have been honestly computed. After this check, the verifier is convinced that there are no malicious preprocessings and proceeds with the execution of the protocol.

Finally, we note that each repetition of the MPC-in-the-head protocol results in poor soundness, thus we need to perform τ parallel executions to drop to a negligible soundness error.

The structure of the protocol is the following:

1. In the first round, the prover computes the preprocessings and commits to the parties' inputs for each parallel execution.

2. In the second round, the verifier asks the prover to open a subset of the preprocessings and sends a challenge for each execution of the MPC-in-the-head protocol.

3. In the third round, the prover opens the requested preprocessings and performs τ parallel executions of the MPC protocol using the verifier's challenge. It then commits to the parties' views.

4. In the fourth round, the verifier asks the prover to reveal all but one of the parties' views for each parallel execution.

5. In the fifth round, the prover sends all the requested views to the verifier and the verifier proceeds to verify the commitments.

We will now describe each round of the protocol in detail:

Round 1

For succinctness, the prover generates as much randomness as possible using PRGs and small seeds. Specifically, the prover samples a random seed $mseed$ and uses it to generate all the randomness needed for the rest of the protocol. For efficiency, it uses a Tree PRG instead of a normal PRG to create shares. First, the prover computes M preprocessings, some of which will be used to prove membership of the secret shared vectors in the input sets U, V. For each preprocessing i, it samples two random masks $mask_i^{(U)}, mask_i^{(V)}$ to be used on the sets U, V respectively. We use $\Delta_{i,k}^{(U)}$ (respectively $\Delta_{i,k}^{(V)}$) to denote the difference between the k-th vector of set U (respectively V) and the mask $mask_i^{(U)}$ (respectively $mask_i^{(V)}$). It also uses the additive secret sharing scheme 3.2.2 to create N shares of each mask (one for each party) and commits to those shares. Then it commits to the Delta values (the masked vectors), permutes them and accumulates them. Note that each of the sets U, V is accumulated seperately. To summarize, each preprocessing contains the commitments to the mask shares and the accumulated values $acc_i^{(U)}, acc_i^{(V)}$ (one for each set).

The prover now prepares the additional inputs of each of the virtual parties for each parallel execution i of the protocol:

- N additive secret shares of the orthogonal vectors u_α, v_β that are the solution to OV.

- N additive secret shares of the values a_i, c_i that are necessary for the batch product evaluation MPC protocol as presented in 3.2.4.

Then, the prover computes some auxiliary values:

- $\Delta u_\alpha^{(i)}, \Delta v_\beta^{(i)}$ to correct the sum of the vector shares, i.e., $\Delta u_\alpha^{(i)} + \sum_{j=1}^{N} [\![u_\alpha]\!]_j^{(i)} = u_\alpha, \Delta v_\beta^{(i)} + \sum_{j=1}^{N} [\![v_\beta]\!]_j^{(i)} = v_\beta$.

- Δc_i to correct the sum of the c_i shares such that $\Delta c_i + \sum_{j=1}^{N} [\![c_i]\!]_j = \langle a_i, u_\alpha \rangle$

Finally, the prover commits to the parties' inputs, computes the hashes of the preprocessings, the commitments and the auxiliary values and sends them to the verifier.

Round 2

The verifier randomly partitions the preprocessings into two sets S, J, where $|J| = \tau$ and $|S| = M - \tau$. This is the challenge for the cut-and-choose strategy that is used to verify the preprocessings' validity. All preprocessings in S will be opened and the verifier will check whether they have been computed honestly. The unopened preprocessings in J will be used for the τ parallel executions of the MPC protocol. Then the verifier randomly samples τ challenges $\epsilon_1, \ldots, \epsilon_\tau$, one for each of the parallel executions of 3.2.4, and sends them to the prover along with the sets S, J.

Round 3

The prover opens all preprocessings in S by sending all the randomness used to compute them to the verifier. Then, it proceeds with running τ parallel executions of a secure MPC protocol "in its head" where, in each repetition i, the parties get as private inputs: 1) the mask shares, 2) the orthogonal vector shares, and 3) the shares of the values a_i, c_i. Each party also has public access to the masked vectors u_α, v_β of the current preprocessing J_i (denoted by $\Delta^{(U)}_{J_i,\alpha}, \Delta^{(V)}_{J_i,\beta}$).

The parties can now verify that their vector shares correspond to vectors belonging in U, V and that $\langle u_\alpha, v_\beta \rangle = 0$. The former is easy as the parties only need to verify that $\Delta^{(U)}_{J_i,\alpha} = u_\alpha - mask^{(U)}_{J_i}$ and $\Delta^{(V)}_{J_i,\beta} = v_\beta - mask^{(V)}_{J_i}$. Those equalities imply that the vectors u_α, v_β and the corresponding masks are consistent with the public values $\Delta^{(U)}_{J_i,\alpha}$ and $\Delta^{(V)}_{J_i,\beta}$. The only thing left to prove membership is to verify that the commitments of those values are indeed part of the accumulated set. We also note that those equalities are linear and thus trivial to prove through an MPC-in-the-head protocol; The parties can simply use their private shares to compute the values $[\![\Delta^{(U)}_{J_i,\alpha}]\!]_j = [\![u_\alpha]\!]^{(i)}_j - [\![mask^{(U)}_{J_i}]\!]_j$ and $[\![\Delta^{(V)}_{J_i,\beta}]\!]_j = [\![v_\beta]\!]^{(i)}_j - [\![mask^{(V)}_{J_i}]\!]_j$ and broadcast those values to get $\Delta^{(U)}_{J_i,\alpha}, \Delta^{(V)}_{J_i,\beta}$. Those values are perfectly hiding and no information can be extracted about the vector and mask shares. Now that the parties are convinced they possess valid shares of two vectors in U and V, they can proceed with verifying orthogoniality. Verifying that $\langle u_\alpha, v_\beta \rangle = 0$ is more challenging, because we need to deal with a non linear relation. For this purpose, the parties run the batch product verification MPC protocol presented in 3.2.4 using the challenges $\epsilon_1, \ldots, \epsilon_\tau$ received by the verifier.

The prover now computes the membership proofs for the commitmets of $\Delta^{(U)}_{J_i,\alpha}, \Delta^{(V)}_{J_i,\beta}$ using the accumulated values $acc^{(U)}_{J_i}, acc^{(V)}_{J_i}$ respectively.

The prover sends the following to the verifier for each parallel execution i of the MPC protocol:

- A hash of the parties' views during the execution of the MPC protocol.
- The public values $\Delta^{(U)}_{J_i,\alpha}, \Delta^{(V)}_{J_i,\beta}$ and the randomness $r^{(\Delta)(U)}_{J_i,\alpha}, r^{(\Delta)(V)}_{J_i,\beta}$ used to commit to them.
- The proofs $\pi^{(U)}_i, \pi^{(V)}_i$ computed earlier.

Round 4

For each parallel execution i of the MPC protocol, the verifier asks the prover to open the views of all parties except one. The index of the unopened party is denoted by j^*_i. We note that an additive secret sharing scheme among N parties is $(N-1)$-private which implies that the verifier gets no information about the secret shared values.

Round 5

For each parallel execution of the MPC protocol i, the prover opens the views of all parties except the one chosen by the verifier. It also sends all missing information that the verifier needs to check the validity of the commitments and the membership proofs.

The verifier reruns the protocol using the randomness provided by the prover and checks whether the commitments were honestly computed.

5.2.2 Protocol

Public Inputs: Set U of vectors $u_1, \ldots, u_n \in \mathbb{F}_2^d$, Set V of vectors $v_1, \ldots, v_n \in \mathbb{F}_2^d$, Accumulator Parameters *par*

Prover Inputs: A witness $w = u_\alpha || v_\beta$ such that $\alpha, \beta \in [n]$ and $\langle u_\alpha, v_\beta \rangle = 0$
Protocol Parameters: Number of preprocessings for cut and choose: M, Number of Parties: N, Number of repetitions: τ, Security parameter: λ

Round 1

Prover

Create the necessary randomness:

1. $mseed \xleftarrow{\$} \{0,1\}^\lambda$
2. $(r^{(mask)}, r^{(\Delta)}, r^{(party)}) \leftarrow PRG(mseed)$
3. $\{r_i^{(mask)}\}_{i\in[M]} \leftarrow TreePRG(r^{(mask)})$
4. $\{r_i^{(\Delta)}\}_{i\in[M]} \leftarrow TreePRG(r^{(\Delta)})$
5. $\{r_i^{(party)}\}_{i\in[\tau]} \leftarrow TreePRG(r^{(party)})$

For each preprocessing $i \in [M]$:

1. Sample 2 random masks and create N additive shares for each:
   (a) $(mask_i^{(U)}, mask_i^{(V)}) \leftarrow PRG(r_i^{(mask)})$, where $mask_i^{(U)}, mask_i^{(V)} \in \mathbb{F}_2^d$
   (b) $\forall k \in [n]$: Compute $\Delta_{i,k}^{(U)} = u_k - mask_i^{(U)}$; $\Delta_{i,k}^{(V)} = v_k - mask_i^{(V)}$
   (c) $\{[\![mask_i^{(U)}]\!]_j, [\![mask_i^{(V)}]\!]_j, r_{i,j}^{(mask-com)}\}_{j\in[N]} \leftarrow TreePRG(r_i^{(mask)})$;
   $[\![mask_i^{(U)}]\!]_N \leftarrow mask_i^{(U)} - \sum_{k=1}^{N-1} [\![mask_i^{(U)}]\!]_k$;
   $[\![mask_i^{(V)}]\!]_N \leftarrow mask_i^{(V)} - \sum_{k=1}^{N-1} [\![mask_i^{(V)}]\!]_k$
2. Commit to mask shares and Deltas:
   (a) $\forall j \in [N]$: Compute $com_{i,j}^{(mask)} = Com((([\![mask_i^{(U)}]\!]_j, [\![mask_i^{(V)}]\!]_j); r_{i,j}^{(mask-com)})$
   (b) $\{r_{i,k}^{(\Delta)(U)}, r_{i,k}^{(\Delta)(V)}\}_{k\in[n]} \leftarrow TreePRG(r_i^{(\Delta)})$
   (c) $\forall k \in [n]$: Compute $com_{i,k}^{(\Delta)(U)} = Com(\Delta_{i,k}^{(U)}; r_{i,k}^{(\Delta)(U)})$; $com_{i,k}^{(\Delta)(V)} = Com(\Delta_{i,k}^{(V)}; r_{i,k}^{(\Delta)(V)})$
3. Permute and accumulate Delta commitments:
   (a) Sample two uniformly random permutations $\phi_i^{(U)}, \phi_i^{(V)}$ from the space of all permutations of size n.
   (b) $(aux_i^{(U)}, acc_i^{(U)}) \leftarrow Acc.Eval(par, com_{i,\phi_i^{(U)}(1)}^{(\Delta)(U)}, \ldots, com_{i,\phi_i^{(U)}(n)}^{(\Delta)(U)})$;
   $(aux_i^{(V)}, acc_i^{(V)}) \leftarrow Acc.Eval(par, com_{i,\phi_i^{(V)}(1)}^{(\Delta)(V)}, \ldots, com_{i,\phi_i^{(V)}(n)}^{(\Delta)(V)})$
   (c) Set $R_i = (com_{i,1}^{(mask)}, \ldots, com_{i,N}^{(mask)}, acc_i^{(U)}, acc_i^{(V)})$

For each repetition $i \in [\tau]$:

1. Generate the parties' private inputs:
   (a) $\{r_{i,j}^{(party)}, r_{i,j}^{(party-com)}\}_{j\in[N]} \leftarrow TreePRG(r_i^{(party)})$

(b) For each $j \in [N]$: $(\llbracket a_i \rrbracket_j, \llbracket u_\alpha \rrbracket_j^{(i)}, \llbracket v_\beta \rrbracket_j^{(i)}, \llbracket c_i \rrbracket_j) \leftarrow TreePRG(r_{i,j}^{(party)})$, where $\llbracket a_i \rrbracket_j, \llbracket u_\alpha \rrbracket_j^{(i)}, \llbracket v_\beta \rrbracket_j^{(i)} \in \mathbb{F}_2^d$ and $\llbracket c_i \rrbracket_j) \in \mathbb{F}_2$

2. Create auxiliary values:

   (a) Compute $\Delta u_\alpha^{(i)} = u_\alpha - \sum_{j=1}^N \llbracket u_\alpha \rrbracket_j^{(i)}$; $\Delta v_\beta^{(i)} = v_\beta - \sum_{j=1}^N \llbracket v_\beta \rrbracket_j^{(i)}$

   (b) Compute $\Delta c_i = \langle a_i, u_\alpha \rangle - \sum_{j=1}^N \llbracket c_i \rrbracket_j$, where $a_i = \sum_{j=1}^N \llbracket a_i \rrbracket_j$

3. Commit to the parties' inputs:

   (a) $\forall j \in [N]$: Compute $com_{i,j}^{(party)} = Com(r_{i,j}^{(party)}; r_{i,j}^{(party-com)})$

   (b) Set $I_i = (\Delta u_\alpha^{(i)}, \Delta v_\beta^{(i)}, \Delta c_i, com_{i,1}^{(party)}, \ldots, com_{i,N}^{(party)})$.

Send the accumulators and all the commitments to the verifier:

1. Set $R = (R_1, \ldots, R_M)$.
2. Set $I = (I_1, \ldots, I_\tau)$.
3. Compute $h_1 = Hash(R, I)$.
4. Send h_1 to the verifier.

Round 2

Verifier

1. Receive h_1' from the prover.
2. Partition $[M]$ into subsets J, S by uniformly sampling $J \xleftarrow{\$} \{J \subset [M]; |J| = \tau\}$. From now on, we denote the i-th element of J by J_i.
3. $\forall i \in [\tau]$: Sample uniformly random $\epsilon_i \xleftarrow{\$} \mathbb{F}_2^d$.
4. Send $(J, \{\epsilon_i\}_{i \in [\tau]})$ to the prover.

Round 3

Prover

Open all preprocessings in S:

$\forall i \in S$: Send $(r_i^{(mask)}, r_i^{(\Delta)}, \phi_i^{(U)}, \phi_i^{(V)})$ to the verifier.

For each repetition $i \in [\tau]$, use an MPC-in-the-head protocol to prove that $u_\alpha \in U$, $v_\beta \in V$ and $\langle u_\alpha, v_\beta \rangle = 0$:

1. Each party P_j gets private inputs: $\llbracket mask_{J_i}^{(U)} \rrbracket_j, \llbracket mask_{J_i}^{(V)} \rrbracket_j, \llbracket u_\alpha \rrbracket_j^{(i)}, \llbracket v_\beta \rrbracket_j^{(i)}, \llbracket a_i \rrbracket_j, \llbracket c_i \rrbracket_j$. Also, all parties and the verifier have public access to the sets U, V as well as to $\Delta_{J_i,\alpha}^{(U)}, \Delta_{J_i,\beta}^{(V)}$.
2. Each party P_j does the following:

   (a) Compute $\llbracket \Delta_{J_i,\alpha}^{(U)} \rrbracket_j = \llbracket u_\alpha \rrbracket_j^{(i)} - \llbracket mask_{J_i}^{(U)} \rrbracket_j$.

(b) Compute $[\![\Delta^{(V)}_{J_i,\beta}]\!]_j = [\![v_\beta]\!]^{(i)}_j - [\![mask^{(V)}_{J_i}]\!]_j$.

(c) Compute $[\![e_i]\!]_j = \epsilon_i \circ [\![v_\beta]\!]^{(i)}_j + [\![a_i]\!]_j$.

3. All parties open $[\![\Delta^{(U)}_{J_i,\alpha}]\!], [\![\Delta^{(V)}_{J_i,\beta}]\!], [\![e_i]\!]_j$ to get $\Delta^{(U)}_{J_i,\alpha}, \Delta^{(V)}_{J_i,\beta}, e_i$ respectively.

4. Each party P_j computes $[\![g_i]\!]_j = \langle e_i, [\![u_\alpha]\!]^{(i)}_j \rangle - [\![c_i]\!]_j$.

5. Create membership proofs for $\Delta^{(U)}_{J_i,\alpha}, \Delta^{(V)}_{J_i,\beta}$:

(a) Compute $\pi^{(U)}_i \leftarrow Acc.Proof(par, com^{(\Delta)(U)}_{J_i,\phi^{(U)}_{J_i}(\alpha)}, acc^{(U)}_{J_i}, aux_{J_i})$.

(b) Compute $\pi^{(V)}_i \leftarrow Acc.Proof(par, com^{(\Delta)(V)}_{J_i,\phi^{(V)}_{J_i}(\beta)}, acc^{(V)}_{J_i}, aux_{J_i})$.

Compute $h_2 = Hash(\{[\![\Delta^{(U)}_{J_i,\alpha}]\!], [\![\Delta^{(V)}_{J_i,\beta}]\!], [\![e_i]\!], [\![g_i]\!]\}_{i\in[\tau]})$

Send $(h_2, \{\pi^{(U)}_i, \pi^{(V)}_i, \Delta^{(U)}_{J_i,\alpha}, \Delta^{(V)}_{J_i,\beta}, r^{(\Delta)(U)}_{J_i,\alpha}, r^{(\Delta)(V)}_{J_i,\beta}\}_{i\in[\tau]})$ to the verifier.

Round 4

Verifier

1. Receive $\{(r^{(mask)\prime}_i, r^{(\Delta)\prime}_i, \phi^{(U)\prime}_i, \phi^{(V)\prime}_i)\}_{i\in S}$ from the prover.
2. Receive $(h'_2, \{\pi^{(U)\prime}_i, \pi^{(V)\prime}_i, \Delta^{(U)\prime}_{J_i,\alpha}, \Delta^{(V)\prime}_{J_i,\beta}, r^{(\Delta)(U)\prime}_{J_i,\alpha}, r^{(\Delta)(V)\prime}_{J_i,\beta}\}_{i\in[\tau]})$ from the prover.
3. $\forall i \in [\tau]$: Sample a uniformly random $j^*_i \leftarrow [N]$.
4. Send $\{j^*_i\}_{i\in[\tau]}$ to the prover.

Round 5

Prover

For each $i \in [\tau]$: Open all parties $\{P_j\}_{j\in[N]\setminus\{j^*_i\}}$ and send the necessary auxiliary information:

Send the following to the verifier:

- $\{r^{(mask-com)}_{J_i,j}\}_{j\in[N]\setminus\{j^*_i\}}$
- $\{[\![mask^{(U)}_{J_i}]\!]_j, [\![mask^{(V)}_{J_i}]\!]_j\}_{j\in[N]\setminus\{j^*_i\}}$
- $\{r^{(party)}_{i,j}, r^{(party-com)}_{i,j}\}_{j\in[N]\setminus\{j^*_i\}}$
- $com^{(party)}_{i,j^*_i}$
- $\Delta u^{(i)}_\alpha, \Delta v^{(i)}_\beta$
- Δc_i
- $[\![e_i]\!]_{j^*_i}$
- $(acc^{(U)}_{J_i}, acc^{(V)}_{J_i})$

- $com^{(mask)}_{J_i,j_i^*}$

Verifier

Receive the following from the prover:

- $\{r^{(mask-com)\prime}_{J_i,j}\}_{j\in[N]\setminus\{j_i^*\}}$
- $\{[\![mask^{(U)}_{J_i}]\!]'_j, [\![mask^{(V)}_{J_i}]\!]'_j\}_{j\in[N]\setminus\{j_i^*\}}$
- $\{r^{(party)\prime}_{i,j}, r^{(party-com)\prime}_{i,j}\}_{j\in[N]\setminus\{j_i^*\}}$
- $com^{(party)\prime}_{i,j_i^*}$
- $\Delta u^{(i)\prime}_{\alpha}, \Delta v^{(i)\prime}_{\beta}$
- $\Delta c'_i$
- $[\![e_i]\!]'_{j_i^*}$
- $acc^{(U)\prime}_{J_i}, acc^{(V)\prime}_{J_i}$
- $com^{(mask)\prime}_{J_i,j_i^*}$

Compute the preprocessings in S:

For all $i \in S$:

1. $(mask^{(U)\prime}_i, mask^{(V)\prime}_i) \leftarrow PRG(r^{(mask)\prime}_i)$.
2. $\forall k \in [n]$: Compute $\Delta^{(U)\prime}_{i,k} = u_k - mask^{(U)\prime}_i$; $\Delta^{(V)\prime}_{i,k} = v_k - mask^{(V)\prime}_i$
3. $\{[\![mask^{(U)}_i]\!]'_j, [\![mask^{(V)}_i]\!]'_j, r^{(mask-com)\prime}_{i,j}\}_{j\in[N]} \leftarrow TreePRG(r^{(mask)\prime}_i)$;
   $[\![mask^{(U)}_i]\!]'_N \leftarrow mask^{(U)\prime}_i - \sum_{k=1}^{N-1}[\![mask^{(U)}_i]\!]'_k$;
   $[\![mask^{(V)}_i]\!]'_N \leftarrow mask^{(V)\prime}_i - \sum_{k=1}^{N-1}[\![mask^{(V)}_i]\!]'_k$
4. $\forall j \in [N]$: Compute: $com^{(mask)\prime}_{i,j} = Com(([\![mask^{(U)}_i]\!]'_j, [\![mask^{(V)}_i]\!]'_j); r^{(mask-com)\prime}_{i,j})$
5. $\{r^{(\Delta)(U)\prime}_{i,k}, r^{(\Delta)(V)\prime}_{i,k}\}_{k\in[n]} \leftarrow TreePRG(r^{(\Delta)\prime}_i)$
6. $\forall k \in [n]$: Compute: $com^{(\Delta)(U)\prime}_{i,k} = Com(\Delta^{(U)\prime}_{i,k}; r^{(\Delta)(U)\prime}_{i,k})$; $com^{(\Delta)(V)\prime}_{i,k} = Com(\Delta^{(V)\prime}_{i,k}; r^{(\Delta)(V)\prime}_{i,k})$
7. $(aux^{(U)\prime}_i, acc^{(U)\prime}_i) \leftarrow Acc.Eval(par, com^{(\Delta)(U)\ \prime}_{i,\phi_i^{(U)}(1)'}, \ldots, com^{(\Delta)(U)\ \prime}_{i,\phi_i^{(U)}(n)'})$;
   $(aux^{(V)\prime}_i, acc^{(V)\prime}_i) \leftarrow Acc.Eval(par, com^{(\Delta)(V)\ \prime}_{i,\phi_i^{(V)}(1)'}, \ldots, com^{(\Delta)(V)\ \prime}_{i,\phi_i^{(V)}(n)'})$
8. Set $R'_i = (com^{(mask)\prime}_{i,1}, \ldots, com^{(mask)\prime}_{i,N}, acc^{(U)\prime}_i, acc^{(V)\prime}_i)$

Verify the Delta memberships and the masks:

For all repetitions $i \in [\tau]$:

1. Compute: $com^{(\Delta)(U)\prime}_{J_i,\alpha} = Com(\Delta^{(U)\ \prime}_{J_i,\alpha}; r^{(\Delta)(U)\prime}_{J_i,\alpha})$; $com^{(\Delta)(V)\prime}_{J_i,\beta} = Com(\Delta^{(V)\prime}_{J_i,\beta}; r^{(\Delta)(V)\prime}_{J_i,\beta})$.

2. Check: $Acc.Verify(par, com^{(\Delta)(U)\prime}_{J_i,\alpha}, acc^{(U)\prime}_{J_i}, \pi^{(U)\prime}_i) \stackrel{?}{=} 1$; $Acc.Verify(par, com^{(\Delta)(V)\prime}_{J_i,\beta}, acc^{(V)\prime}_{J_i}, \pi^{(V)\prime}_i) \stackrel{?}{=} 1$.
3. $\forall j \in [N] \setminus \{j^*_i\}$: Compute $com^{(mask)\prime}_{J_i,j} = Com((\llbracket mask^{(U)}_{J_i} \rrbracket'_j, \llbracket mask^{(V)}_{J_i} \rrbracket'_j); r^{(mask-com)\prime}_{J_i,j})$.

For all $i \in J$: Set $R'_i = (com^{(mask)\prime}_{i,1}, \ldots, com^{(mask)\prime}_{i,N}, acc^{(U)\prime}_i, acc^{(V)\prime}_i)$.

Set $R' = (R'_1, \ldots, R'_M)$.

Verify the parties' inputs:

For each repetition $i \in [\tau]$:

1. $\forall j \in [N] \setminus \{j^*_i\}$: Compute $com^{(party)\prime}_{i,j} = Com(r^{(party)\prime}_{i,j}; r^{(party-com)\prime}_{i,j})$
2. Set $I'_i = (\Delta u^{(i)\prime}_\alpha, \Delta v^{(i)\prime}_\beta, \Delta c'_i, com^{(party)\prime}_{i,1}, \ldots, com^{(party)\prime}_{i,N})$.

Set $I' = (I'_1, \ldots, I'_\tau)$.

Check if $h'_1 \stackrel{?}{=} Hash(R', I')$.

For each iteration $i \in [\tau]$, verify the MPC-in-the-head protocol:

1. $\forall j \in [N] \setminus \{j^*_i\}$: Compute $(\llbracket a_i \rrbracket'_j, \llbracket u_\alpha \rrbracket^{(i)\prime}_j, \llbracket v_\beta \rrbracket^{(i)\prime}_j, \llbracket c_i \rrbracket'_j) \leftarrow TreePRG(r^{(party)\prime}_{i,j})$.
2. Compute $\Delta e'_i = \epsilon_i \circ \Delta v^{(i)\prime}_\beta$.
3. $\forall j \in [N] \setminus \{j^*_i\}$: Compute $\llbracket \Delta^{(U)}_{J_i,\alpha} \rrbracket'_j = \llbracket u_\alpha \rrbracket^{(i)\prime}_j - \llbracket mask^{(U)}_{J_i} \rrbracket'_j$.
4. $\forall j \in [N] \setminus \{j^*_i\}$: Compute $\llbracket \Delta^{(V)}_{J_i,\beta} \rrbracket'_j = \llbracket v_\beta \rrbracket^{(i)\prime}_j - \llbracket mask^{(V)}_{J_i} \rrbracket'_j$.
5. $\forall j \in [N] \setminus \{j^*_i\}$: Compute $\llbracket e_i \rrbracket'_j = \epsilon_i \circ \llbracket v_\beta \rrbracket^{(i)\prime}_j + \llbracket a_i \rrbracket'_j$.
6. $\forall j \in [N] \setminus \{j^*_i\}$: Compute $\llbracket g_i \rrbracket'_j = \langle \sum_{k=1}^N \llbracket e_i \rrbracket'_k, \llbracket u_\alpha \rrbracket^{(i)\prime}_j \rangle - \llbracket c_i \rrbracket'_j$.
7. Compute $e'_i = \sum_{j=1}^N \llbracket e_i \rrbracket'_j + \Delta e'_i$.
8. Compute $\Delta g'_i = \langle e'_i, \Delta u^{(i)\prime}_\alpha \rangle - \Delta c'_i$.
9. Compute $\llbracket \Delta^{(U)}_{J_i,\alpha} \rrbracket'_{j^*_i} = \Delta^{(U)\prime}_{J_i,\alpha} - \Delta u^{(i)\prime}_\alpha - \sum_{j=1, j\neq j^*_i}^N \llbracket \Delta^{(U)}_{J_i,\alpha} \rrbracket'_j$.
10. Compute $\llbracket \Delta^{(V)}_{J_i,\beta} \rrbracket'_{j^*_i} = \Delta^{(V)\prime}_{J_i,\beta} - \Delta v^{(i)\prime}_\beta - \sum_{j=1, j\neq j^*_i}^N \llbracket \Delta^{(V)}_{J_i,\beta} \rrbracket'_j$.
11. Compute $\llbracket g_i \rrbracket'_{j^*_i} = -\Delta g'_i - \sum_{j=1, j\neq j^*_i} \llbracket g_i \rrbracket'_j$.

Check if $h'_2 \stackrel{?}{=} Hash(\{\llbracket \Delta^{(U)}_{J_i,\alpha} \rrbracket', \llbracket \Delta^{(V)}_{J_i,\beta} \rrbracket', \llbracket e_i \rrbracket', \llbracket g_i \rrbracket'\}_{i\in[\tau]})$.

Return 1.

5.2.3 Communication Complexity

We will now compute the communication complexity of our protocol:

- We take the output of the hash function $Hash$ to be a λ-bit string.
- The set J can be described by using $\tau log\tau$ bits.
- We assume that the prover uses a PRG to sample each permutation, thus it can send it to the verifier by using λ bits.
- Each membership proof has a size of $\lambda logn$ bits.
- The challenges sent by the verifier in Round 4 have a total size of $\tau logN$ bits.
- In order to send $\{[\![mask_{J_i}^{(U)}]\!]_j, [\![mask_{J_i}^{(V)}]\!]_j, r_{J_i,j}^{(mask-com)}\}_{j\in[N]\setminus\{j_i^*\}}$, the prover only needs to send $\lambda logN$ bits for each repetition i by taking advantage of the TreePRG. The same holds for $\{r_{i,j}^{(party)}, r_{i,j}^{(party-com)}\}_{j\in[N]\setminus\{j_i^*\}}$.
- Each commitment, accumulated value and randomness has a size of λ bits.

The exact communication complexity of our protocol is:

$$2\lambda + (M-\tau)4\lambda + \tau(log\tau + 2\lambda logN + 2\lambda \log n + logN + 6d + 6\lambda + 1)$$

We take $M = O(\tau)$ to get an asymptotic communication complexity:

$$O(\tau(log\tau + \lambda logN + \lambda \log n + d))$$

Finally we take $\tau = O(log^2 n)$, $\lambda = log^2 n$ and $N = O(1)$ to achieve a communication complexity:

$$O(\log^2 n(\log^3 n + d))$$

5.2.4 Computation Complexity

The asymptotic computation complexity of our protocol solely comes from the creation and verification of the preprocessings which is:

$$O(M(n+N)(\lambda + d))$$

Using the values of 5.2.3, we achieve a computation complexity:

$$O(nlog^2 n(log^2 n + d))$$

We remind that in the Orthogonal Vectors problem it holds that $d = o(n)$, thus the limitation for a truly subquadratic complexity always holds.

5.2.5 Completeness

Theorem 5.2.1. *A prover $\mathcal{P}$ who knows a solution $u_\alpha||v_\beta$ to the OV instance U, V and who follows the steps of Protocol 5.2.2 convinces the verifier $\mathcal{V}$ with probability* 1.

Proof. For any sampling of the random coins of $\mathcal{P}$ and $\mathcal{V}$, if the computation described in the protocol is honestly performed, then all the checks of $\mathcal{V}$ pass. Thus, the completeness error ϵ_c satisfies

$$\epsilon_c = 0$$

. □

5.2.6 Soundness

Soundness Error

Theorem 5.2.2. *A prover $\tilde{\mathcal{P}}$ who does not know a solution $u_\alpha || v_\beta$ to the OV instance U, V cannot convince the verifier $\mathcal{V}$ except with probability ϵ_s that satisfies:*

$$\epsilon_s \leq \max_A \left\{ \frac{\binom{M-A}{M-\tau}}{\binom{M}{M-\tau}} \cdot \left(\frac{1}{2} + \frac{1}{N}\right)^{\tau - A} \right\}$$

Proof. In each parallel execution i of the MPC-in-the-head protocl, a malicious prover $\mathcal{P}^*$ can cheat in two different ways:

1. Perform the parallel execution of the MPC-in-the-head protocol honestly while using a maliciously crafted preprocessing J_i which successfully passed the opening check (1st challenge).

2. Cheat in the execution of the MPC-in-the-head protocol while using an honestly crafted preprocessing. There are two ways to ways to cheat in this case:

   (a) Perform an honest execution of the MPC protocol while using a "bad" challenge given by the verifier, i.e., an ϵ_i such that the verifier accepts, even though the vectors are not orthogonal.

   (b) Perform a malicious execution of the MPC protocol by corrupting the "right" party, i.e., the party that will not be opened by the verifier.

We will first compute the probability that the cheating prover $\mathcal{P}^*$ successfully passes the cut-and-choose phase. If it fails to do so (i.e., the verifier opens at least one maliciously crafted preprocessing), the verifier aborts with probability 1. Let A denote the number of preprocessings in which $\mathcal{P}^*$ cheats. The number of ways to sample a set of honestly crafted preprocessings to be opened is $\binom{M-A}{M-\tau}$. The number of ways to sample any set of preprocessings to be opened is $\binom{M}{M-\tau}$. It follows that the probability of a cheating prover P^* to successfully pass the cut-and-choose phase is

$$\frac{\binom{M-A}{M-\tau}}{\binom{M}{M-\tau}}$$

.

We now suppose that the verifier successfully verified the cut-and-choose phase and proceeds in verifying the MPC-in-the-head protocol. In order to cheat in this phase, the prover must either use a malicious preprocessing or cheat during the protocol execution. We remind that there are τ preprocessings remaining and there are A malicious preprocessings among those. This implies that the prover does not need to cheat during A repetitions of the MPC-in-the-head protocol. The batch product verification MPC protocol 3.2.4 gives a false positive with probability at most $\frac{1}{2}$. The compilation of that protocol into an MPC-in-the-head protocol adds a soundness error of $\frac{1}{N}$, which corresponds to the case where the verifier doesn't open the corrupted party during the verification stage. Thus, the soundness error of the MPC-in-the-head protocol while using an honestly crafted preprocessing is:

$$1 - (1 - \frac{1}{2})(1 - \frac{1}{N}) < \frac{1}{2} + \frac{1}{N},$$

where N is an integer constant. It follows that the probability of the prover violating the MPC-in-the-head protocol during the rest $\tau - A$ repetitions is upper bounded by $(\frac{1}{2} + \frac{1}{N})^{\tau - A}$. The overall soundness error ϵ_s of our zero knowledge protocol for OV is upper bounded by:

$$\epsilon_s \leq \max_A \left\{ \frac{\binom{M-A}{M-\tau}}{\binom{M}{M-\tau}} \cdot \left(\frac{1}{2} + \frac{1}{N}\right)^{\tau-A} \right\}$$

□

We can make this error negligible by carefully choosing the parameters. We take

$$\tau = \frac{log^2 n}{log(\frac{1}{2} + \frac{1}{N})}, M = \frac{3}{2}\tau$$

It holds that $0 \leq A \leq \tau$. If the prover chooses to create exactly τ malicious preprocessings, then the second term of the soundness error becomes one, also it is easy to show by induction that

$$\frac{\binom{M-A}{M-\tau}}{\binom{M}{M-\tau}} \leq \frac{1}{n^{logn}}$$

. If the prover chooses to create 0 malicious preprocessings, then the first term of the soundness error becomes one, also

$$\left(\frac{1}{2} + \frac{1}{N}\right)^{\tau-A} = \frac{1}{n^{logn}}$$

Special Soundness

Theorem 5.2.3. *Let*

$$\epsilon_s = \max_A \left\{ \frac{\binom{M-A}{M-\tau}}{\binom{M}{M-\tau}} \cdot \left(\frac{1}{2} + \frac{1}{N}\right)^{\tau-A} \right\}$$

Suppose there is an efficient prover $\tilde{\mathcal{P}}$ such that

$$\tilde{\epsilon}_s := Pr[\langle \tilde{\mathcal{P}}(x), \mathcal{V}(x) \rangle (U, V) = 1] > \epsilon_s,$$

where U, V is an OV instance. Then, there exists an extractor algorithm $\mathcal{E}$ which, given rewindable black-box access to $\tilde{\mathcal{P}}$, produces either a witness $u||v$ such that $\langle u, v \rangle = 0$, $u \in U$ and $v \in V$, a hash collision or a commitment collision, by making in average

$$\frac{4}{\tilde{\epsilon}_s - \epsilon_s} \cdot \left(1 + \tilde{\epsilon}_s \cdot \frac{8M}{\tilde{\epsilon}_s - \epsilon_s}\right)$$

calls to $\tilde{\mathcal{P}}$.

Proof. We will first demonstrate how to extract the witness $u_\alpha||v_\beta$ from three transcripts that satisfy specific conditions. For each transcript $i \in \{1, 2, 3\}$, we use the following notation:

$$T_i = (COM^{(i)}, CH_1^{(i)}, RSP_1^{(i)}, CH_2^{(i)}, RSP_2^{(i)}),$$

where

- $COM^{(i)}$ is the prover's first commitment.
- $CH_1^{(i)}$ is the verifier's first challenge that consists of $J^{(i)}$ and $\{\epsilon_j^{(i)}\}_{j \in [\tau]}$
- $RSP_1^{(i)}$ is the prover's first response.
- $CH_2^{(i)}$ is the verifier's second challenge that consists of $\{j_k^{*(i)}\}_{k \in [\tau]}$. For the rest of this proof we use the notation $l_j^{(i)}$ to denote the challenge for the parallel repetition of the MPC protocol that is executed using the j-th preprocessing.

- $RSP_2^{(i)}$ is the prover's second response.

The transcripts that will be used for the witness extraction need to satisfy the following properties:

1. $COM^{(1)} = COM^{(2)} = COM^{(3)} = h_1$
2. $\exists j_0 \in (J^{(1)} \cap J^{(2)}) \setminus J^{(3)}$ such that $l_{j_0}^{(1)} \neq l_{j_0}^{(2)}$
3. Transcripts T_1, T_2 result in the prover accepting.
4. The parties' inputs $([\![a_i]\!]_j, [\![u_\alpha]\!]_j^{(i)}, [\![v_\beta]\!]_j^{(i)}, [\![c_i]\!]_j)$ of the transcripts T_1, T_2 are consistent with $RSP_1^{(3)}$.

If the revealed shares are not consistent between the three transcripts, the extractor can find a hash collision or a commitment collision. We will now assume that all the shares are consistent.

The second requirement for the transcripts implies that the extractor algorithm $\mathcal{E}$ can reconstruct the vectors that were secret shared among the parties in the MPC protocol. Specifically, $\mathcal{E}$ uses $RSP_2^{(1)}, RSP_2^{(2)}$ to obtain the shares $\{[\![u_\alpha]\!]_j^{(j_0)}, [\![v_\beta]\!]_j^{(j_0)}\}_{j\in[N]}$ and $\Delta u_\alpha^{(j_0)}, \Delta v_\beta^{(j_0)}$. $\mathcal{E}$ is now able to compute a candidate solution u'_α, v'_β from u_α, v_β using the additive secret sharing scheme as follows:

$$u'_\alpha = \Delta u_\alpha^{(j_0)} + \sum_{j=1}^{N} [\![u_\alpha]\!]_j^{(j_0)}$$

$$v'_\beta = \Delta v_\beta^{(j_0)} + \sum_{j=1}^{N} [\![v_\beta]\!]_j^{(j_0)}$$

The extractor algorithm now checks whether $\langle u'_\alpha, v'_\beta \rangle = 0$. If this condition holds, then the only thing left to verify is whether $u'_\alpha \in U$ and $v'_\beta \in V$. $\mathcal{E}$ uses the third transcript T_3 to verify membership. Specifically, it uses $RSP_1^{(3)}$ to check whether the vectors u'_α, v'_β have been masked and accumulated. If that also holds, $\mathcal{E}$ returns $u'_\alpha || v'_\beta$ as the witness.

In order to extract three transcripts T_1, T_2, T_3 with the required properties, we can use the extraction prodecure described in the appendix E of [Fene22a] which provides an extraction algorithm that makes in average at most

$$\frac{4}{\tilde{\epsilon}_s - \epsilon_s} \cdot \left(1 + \tilde{\epsilon}_s \cdot \frac{8M}{\tilde{\epsilon}_s - \epsilon_s}\right)$$

calls to $\tilde{\mathcal{P}}$. □

5.2.7 Honest Verifier Zero Knowledge

Theorem 5.2.4. *Let the PRG and the TreePRG used in Protocol 5.2.2 be (t, ϵ_{PRG})-secure, the accumulator ACC be (t, ϵ_{ACC})-secure and the commitment scheme Com be (t, ϵ_{Com})-hiding. Then, there exists an efficient simulator $\mathcal{S}$ which, given random challenges $J, \{\epsilon_i\}_{i\in[\tau]}, \{j_i^*\}_{i\in[\tau]}$, outputs a transcript which is $(t, \tau \cdot \epsilon_{PRG} + \tau \cdot \epsilon_{ACC} + \tau \cdot \epsilon_{Com})$-indistinguishable from a real transcript of Protocol 5.2.2.*

Proof. We will now construct a simulator $\mathcal{S}$ for protocol 5.2.2 which, given access to the input sets U, V, outputs a transcript that is computationally indistinguishable from a real transcript. We remind that, in our case, the simulated transcript is required to be $(\tilde{O}(n^{2-\epsilon}), \zeta)$-indistinguishable (for some negligible function ζ) from a real transcript. However, our simulator satisfies the (stronger) classical notion of indistinguishability against polynomial time adversaries (as long as we use cryptographic primitives that are secure against polynomial time adversaries), thus it is suitable for this purpose. The simulator behaves as follows:

1. Sample the verifier's challenges:
   (a) Randomly partition M into subsets S, J such that $|J| = \tau$ and $|S| = M - \tau$.
   (b) $\forall i \in [\tau]$: Sample uniformly random $\epsilon_i \xleftarrow{\$} \mathbb{F}_2^d$
   (c) $\forall i \in [\tau]$: Sample uniformly random $j_i^* \xleftarrow{\$} N$.
2. Sample randomness:
   (a) $mseed \xleftarrow{\$} \{0,1\}^\lambda$
   (b) $(r^{(mask)}, r^{(\Delta)}, r^{(party)}) \leftarrow PRG(mseed)$
   (c) $\{r_i^{(mask)}\}_{i\in[M]} \leftarrow TreePRG(r^{(mask)})$
   (d) $\{r_i^{(\Delta)}\}_{i\in[M]} \leftarrow TreePRG(r^{(\Delta)})$
   (e) $\{r_i^{(party)}\}_{i\in[\tau]} \leftarrow TreePRG(r^{(party)})$
3. Sample random "witness" vectors: $\alpha, \beta \xleftarrow{\$} [n]$.
4. Simulate the preprocessings that will be opened by the verifier. For each $i \in S$:
   (a) $(mask_i^{(U)}, mask_i^{(V)}) \leftarrow PRG(r_i^{(mask)})$, where $mask_i^{(U)}, mask_i^{(V)} \in \mathbb{F}_2^d$
   (b) $\forall k \in [n]$: Compute $\Delta_{i,k}^{(U)} = u_k - mask_i^{(U)}$; $\Delta_{i,k}^{(V)} = v_k - mask_i^{(V)}$.
   (c) $\{[\![mask_i^{(U)}]\!]_j, [\![mask_i^{(V)}]\!]_j, r_{i,j}^{(mask-com)}\}_{j\in[N]} \leftarrow TreePRG(r_i^{(mask)})$;
   $[\![mask_i^{(U)}]\!]_N \leftarrow mask_i^{(U)} - \sum_{k=1}^{N-1} [\![mask_i^{(U)}]\!]_k$;
   $[\![mask_i^{(V)}]\!]_N \leftarrow mask_i^{(V)} - \sum_{k=1}^{N-1} [\![mask_i^{(V)}]\!]_k$
   (d) $\forall j \in [N]$: Compute $com_{i,j}^{(mask)} = Com(([\![mask_i^{(U)}]\!]_j, [\![mask_i^{(V)}]\!]_j); r_{i,j}^{(mask-com)})$.
   (e) $\{r_{i,k}^{(\Delta)(U)}, r_{i,k}^{(\Delta)(V)}\}_{k\in[n]} \leftarrow TreePRG(r_i^{(\Delta)})$
   (f) $\forall k \in [n]$: Compute $com_{i,k}^{(\Delta)(U)} = Com(\Delta_{i,k}^{(U)}; r_{i,k}^{(\Delta)(U)})$; $com_{i,k}^{(\Delta)(V)} = Com(\Delta_{i,k}^{(V)}; r_{i,k}^{(\Delta)(V)})$.
   (g) Sample two uniformly random permutations $\phi_i^{(U)}, \phi_i^{(V)}$ from the space of all permutations of size n.
   (h) $(aux_i^{(U)}, acc_i^{(U)}) \leftarrow Acc.Eval(par, com_{i,\phi_i^{(U)}(1)}^{(\Delta)(U)}, \ldots, com_{i,\phi_i^{(U)}(n)}^{(\Delta)(U)})$;
   $(aux_i^{(V)}, acc_i^{(V)}) \leftarrow Acc.Eval(par, com_{i,\phi_i^{(V)}(1)}^{(\Delta)(V)}, \ldots, com_{i,\phi_i^{(V)}(n)}^{(\Delta)(V)})$
   (i) Set $R_i = (com_{i,1}^{(mask)}, \ldots, com_{i,N}^{(mask)}, acc_i^{(U)}, acc_i^{(V)})$.
5. Simulate the preprocessings that will be used for the MPC-in-the-head protocol. For each $i \in J$:
   (a) $(mask_i^{(U)}, mask_i^{(V)}) \leftarrow PRG(r_i^{(mask)})$, where $mask_i^{(U)}, mask_i^{(V)} \in \mathbb{F}_2^d$
   (b) Compute $\Delta_{i,\alpha}^{(U)} = u_\alpha - mask_i^{(U)}$; $\Delta_{i,\beta}^{(V)} = v_\beta - mask_i^{(V)}$.
   (c) $\{[\![mask_i^{(U)}]\!]_j, [\![mask_i^{(V)}]\!]_j, r_{i,j}^{(mask-com)}\}_{j\in[N]} \leftarrow TreePRG(r_i^{(mask)})$;
   $[\![mask_i^{(U)}]\!]_N \leftarrow mask_i^{(U)} - \sum_{k=1}^{N-1} [\![mask_i^{(U)}]\!]_k$;
   $[\![mask_i^{(V)}]\!]_N \leftarrow mask_i^{(V)} - \sum_{k=1}^{N-1} [\![mask_i^{(V)}]\!]_k$
   (d) $\forall j \in [N]\backslash\{j_i^*\}$: Compute $com_{i,j}^{(mask)} = Com(([\![mask_i^{(U)}]\!]_j, [\![mask_i^{(V)}]\!]_j); r_{i,j}^{(mask-com)})$
   (e) Compute $com_{i,j_i^*}^{(mask)} = Com(0; r_{i,j_i^*}^{(mask-com)})$.

(f) $\{r_{i,k}^{(\Delta)(U)}, r_{i,k}^{(\Delta)(V)}\}_{k\in[n]} \leftarrow TreePRG(r_i^{(\Delta)})$

(g) $\forall k \in [n] \setminus \{\alpha\}$: Compute $com_{i,k}^{(\Delta)(U)} = Com(0; r_{i,k}^{(\Delta)(U)})$

(h) $\forall k \in [n] \setminus \{\beta\}$: Compute $com_{i,k}^{(\Delta)(V)} = Com(0; r_{i,k}^{(\Delta)(V)})$

(i) Compute $com_{i,\alpha}^{(\Delta)(U)} = Com(\Delta_{i,\alpha}^{(U)}; r_{i,k}^{(\Delta)(U)})$; $com_{i,\beta}^{(\Delta)(V)} = Com(\Delta_{i,\beta}^{(V)}; r_{i,k}^{(\Delta)(V)})$.

(j) Sample two uniformly random permutations $\phi_i^{(U)}, \phi_i^{(V)}$ from the space of all permutations of size n.

(k) $(aux_i^{(U)}, acc_i^{(U)}) \leftarrow Acc.Eval(par, com_{i,\phi_i^{(U)}(1)}^{(\Delta)(U)}, \ldots, com_{i,\phi_i^{(U)}(n)}^{(\Delta)(U)})$;
$(aux_i^{(V)}, acc_i^{(V)}) \leftarrow Acc.Eval(par, com_{i,\phi_i^{(V)}(1)}^{(\Delta)(V)}, \ldots, com_{i,\phi_i^{(V)}(n)}^{(\Delta)(V)})$

(l) Set $R_i = (com_{i,1}^{(mask)}, \ldots, com_{i,N}^{(mask)}, acc_i^{(U)}, acc_i^{(V)})$

6. Set $R = (R_1, \ldots, R_M)$.

7. Simulate the parties' inputs and the auxiliary variables. For each $i \in [\tau]$:

(a) $\{r_{i,j}^{(party)}, r_{i,j}^{(party-com)}\}_{j\in[N]} \leftarrow TreePRG(r_i^{(party)})$

(b) $\forall j \in [N] \setminus \{j_i^*\}$: $([\![a_i]\!]_j, [\![u_\alpha]\!]_j^{(i)}, [\![v_\beta]\!]_j^{(i)}, [\![c_i]\!]_j) \leftarrow TreePRG(r_{i,j}^{(party)})$

(c) $\forall j \in [N] \setminus \{j_i^*\}$: Compute $com_{i,j}^{(party)} = Com(r_{i,j}^{(party)}; r_{i,j}^{(party-com)})$.

(d) Compute $com_{i,j_i^*}^{(party)} = Com(0; r_{i,j_i^*}^{(party-com)})$.

(e) Sample uniformly random values $\Delta u_\alpha^{(i)}, \Delta v_\beta^{(i)} \xleftarrow{\$} \mathbb{F}_2^d$.

(f) Sample a uniformly random value $\Delta c_i \xleftarrow{\$} \mathbb{F}_2$.

(g) Set $I_i = (\Delta u_\alpha^{(i)}, \Delta v_\beta^{(i)}, \Delta c_i, com_{i,1}^{(party)}, \ldots, com_{i,N}^{(party)})$.

8. Set $I = (I_1, \ldots, I_\tau)$.

9. Compute $h_1 = Hash(R, I)$.

10. Simulate the MPC protocol. For each $i \in [\tau]$:

(a) Sample a uniformly random $e_i \xleftarrow{\$} \mathbb{F}_2^d$.

(b) $\forall j \in [N] \setminus \{j_i^*\}$: Compute $[\![\Delta_{J_i,\alpha}^{(U)}]\!]_j = [\![u_\alpha]\!]_j^{(i)} - [\![mask_{J_i}^{(U)}]\!]_j$; $[\![\Delta_{J_i,\beta}^{(V)}]\!]_j = [\![v_\beta]\!]_j^{(i)} - [\![mask_{J_i}^{(V)}]\!]_j$.

(c) Compute $[\![\Delta_{J_i,\alpha}^{(U)}]\!]_{j_i^*} = \Delta_{J_i,\alpha}^{(U)} - \Delta u_\alpha^{(i)} - \sum_{j=1,j\neq j_i^*}^N [\![\Delta_{J_i,\alpha}^{(U)}]\!]_j$.

(d) Compute $[\![\Delta_{J_i,\beta}^{(V)}]\!]_{j_i^*} = \Delta_{J_i,\beta}^{(V)} - \Delta v_\beta^{(i)} - \sum_{j=1,j\neq j_i^*}^N [\![\Delta_{J_i,\beta}^{(V)}]\!]_j$.

(e) $\forall j \in [N] \setminus \{j_i^*\}$: Compute $[\![e_i]\!]_j = \epsilon_i \circ [\![v_\beta]\!]_j^{(i)} + [\![a_i]\!]_j$.

(f) Compute $\Delta e_i = \epsilon_i \circ \Delta v_\beta^{(i)}$.

(g) Compute $[\![e_i]\!]_{j_i^*} = e_i - \Delta e_i - \sum_{j=1,j\neq j_i^*}^N [\![e_i]\!]_j$.

(h) $\forall j \in [N] \setminus \{j_i^*\}$: Compute $[\![g_i]\!]_j = \langle e_i, [\![u_\alpha]\!]_j^{(i)} \rangle - [\![c_i]\!]_j$.

(i) Compute $\Delta g_i = \langle e_i, \Delta u_\alpha^{(i)} \rangle - \Delta c_i$.

(j) Compute $[\![g_i]\!]_{j_i^*} = -\Delta g_i - \sum_{j=1,j\neq j_i^*} [\![g_i]\!]_j$.

11. Create the membership proofs for the Deltas. For each $i \in [\tau]$:

(a) Compute $\pi_i^{(U)} \leftarrow Acc.Proof(par, com^{(\Delta)(U)}_{J_i,\phi^{(U)}_{J_i}(\alpha)}, acc^{(U)}_{J_i}, aux_{J_i})$.

(b) Compute $\pi_i^{(V)} \leftarrow Acc.Proof(par, com^{(\Delta)(V)}_{J_i,\phi^{(V)}_{J_i}(\beta)}, acc^{(V)}_{J_i}, aux_{J_i})$.

12. Compute $h_2 = Hash(\{[\![\Delta^{(U)}_{J_i,\alpha}]\!], [\![\Delta^{(V)}_{J_i,\beta}]\!], [\![e_i]\!], [\![g_i]\!]\}_{i\in[\tau]})$.

After performing these steps, the simulator is able to output a transcript that is computationally indistinguishable (by any PPT algorithm) from a real transcript. We also note that the running time of the simulator is truly subquadratic, thus it satisfies definition 5.1.1. We will now argue that the simulated transcript is computationally indistinguishable from the real transcript:

- The verifier's challenges are sampled exactly as in the real protocol.
- Randomness is sampled exactly as in the real protocol.
- The simulator is bounded to be truly subquadratic, thus it can't find the orthogonal pair of vectors. Instead, it uses two random vectors u_α, v_β, that belong in the sets U, V respectively, for the rest of the simulation, as if they were the solution to the OV instance. However, both sets U and V are permuted before accumulating, resulting in an identical distribution (in respect to the real protocol).
- The preprocessings that have been chosen to be opened as computed exactly as in the real protocol.
- The preprocessings that will be used for the MPC protocol are computed exactly as in the real protocol except that:
  - The mask commitment for the unopened party j_i^* of each round is computed as $com^{(mask)}_{i,j_i^*} = Com(0; r^{(mask-com)}_{i,j_i^*})$.
  - The Delta commitments that will not be opened by the verifier in each round are computed as $com_i^{(\Delta)} = Com(0; r_i^{(\Delta)})$.

  Indistinguishability here follows from the hiding property of the commitment scheme.
- In the simulated transcript, the auxiliary variables are chosen uniformly at random. In the real protocol, they are computed as the XOR operation of random values, thus they follow the uniform distribution.
- The shares of all opened parties are computed exactly as in the real protocol. The broadcasted shares of the unopened party j_i^* are chosen in such a way so all the shares follow the same distribution in both protocols.
- The commitment to the unopened party's inputs com_{i,j_i^*} in round i is computed as $com^{(party)}_{i,j_i^*} = Com(0; r^{(party-com)}_{i,j_i^*})$. Indistinguishability here follows directly from the hiding property of the commitment scheme.

□

5.3 A Zero Knowledge Argument for 3SUM

In this section, we will provide a zero knowledge argument for a closely related variant of $3SUM$, which we call Modular $3SUM$. We define Modular $3SUM$ as follows:

Definition 5.3.1 (**Modular 3SUM**). Let U, V, W be sets such that $|U| = |V| = |W| = n$ and $U, V, W \subset \mathbb{Z}_q$ for some integer modulus $q = poly(n)$. $3SUM_{\mathbb{Z}_q}$ asks to determine whether there exist three elements u, v, w such that $u \in U$, $v \in V$, $w \in W$ and $u + v + w = 0 \ (mod\ q)$.

We will now show that Modular $3SUM$ is at least as hard as $3SUM$.

Theorem 5.3.1 ($3SUM_m \leq_{n^2,n^2} 3SUM_{\mathbb{Z}_{3m+1}}$). *Let m be a positive integer and q be an integer modulus such that $q = 3m + 1$. The $3SUM_m$ problem is (n^2, n^2)-reducible to $3SUM_{\mathbb{Z}_q}$.*

Proof. Let A be an algorithm for $3SUM_{\mathbb{Z}_q}$ and U, V, W be the input to $3SUM_m$. The reduction works as follows:

1. Create sets U', V', W' as follows:
   - $U' = \{u + m \ (mod\ q) | u \in U\}$
   - $V' = \{v + m \ (mod\ q) | v \in V\}$
   - $W' = \{w - 2m \ (mod\ q) | w \in W\}$
2. Call A on input U', V', W'.
3. If A returns a solution (u', v', w') output $(u' - m, v' - m, -u' - v' + 2m)$ otherwise output NO.

We will now show that A outputs a valid solution (u', v', w') to $3SUM_{\mathbb{Z}_q}$ if and only if $3SUM_m$ has a solution (u, v, w).

Let (u', v', w') be the solution that A outputs. From the construction of u', v', w', it holds that

$$(u + m) + (v + m) + (w - 2m) = 0 \ (mod\ q) \iff u + v + w = 0 \ (mod\ q)$$

From assumption, it holds that

$$-q < -3m \leq u + v + w \leq 3m < q,$$

thus we can also get that

$$u + v + w = 0 \ (mod\ q) \iff u + v + w = 0,$$

which implies that (u, v, w) is a valid $3SUM_m$ solution.

It is easy to see that the reduction requires $\tilde{O}(n)$ time which implies that an $\tilde{O}(n^{2-\epsilon})$ time algorithm for $3SUM_{\mathbb{Z}_q}$ can be compiled into an $\tilde{O}(n^{2-\epsilon})$ time algorithm for $3SUM_m$. □

Corollary 5.3.1 (**No truly subquadratic Modular 3SUM**). In the Word RAM model with $O(logn)$ bit words, any algorithm requires $n^{2-o(1)}$ time in expectation to solve Modular $3SUM$, unless Conjecture 3 is false.

5.3.1 Protocol Description

The protocol for $3SUM_{\mathbb{Z}_q}$ works in the exact same way as Protocol 5.2.2. The prover masks and accumulates the sets U, V, W and secret shares the group elements $u_\alpha, v_\beta, w_\gamma$ among N virtual parties. Then, it uses an MPC-in-the-head Protocol to prove that $u_\alpha + v_\beta + w_\gamma = 0$ and that those elemengs are indeed part of the input sets. In order to verify that the sets have been masked and accumulated honestly, we use a cut-and-choose approach by creating many preprocessings. The verifier opens some of them to check their validity and uses the rest during the protocol's execution. In this protocol there is no need to use the batch product verification protocol, because all the relations are linear.

5.3.2 Protocol

In this section, all operations are performed modulo q, thus we omit $(mod\ q)$ for brevity.

Public Inputs: Set U of group elements $u_1, \ldots, u_n \in \mathbb{Z}_q$, Set V of group elements $v_1, \ldots, v_n \in \mathbb{Z}_q$, Set W of group elements $w_1, \ldots, w_n \in \mathbb{Z}_q$, Accumulator Parameters par
Prover Inputs: A witness $w = u_\alpha || v_\beta || w_\gamma$ such that $\alpha, \beta, \gamma \in [n]$ and $u_\alpha + v_\beta + w_\gamma = 0$
Protocol Parameters: Number of preprocessings for cut and choose: M, Number of Parties: N, Number of repetitions: τ, Security parameter: λ

Round 1

Prover

Create the necessary randomness:

1. $mseed \xleftarrow{\$} \{0,1\}^\lambda$
2. $(r^{(mask)}, r^{(\Delta)}, r^{(party)}) \leftarrow PRG(mseed)$
3. $\{r_i^{(mask)}\}_{i\in[M]} \leftarrow TreePRG(r^{(mask)})$
4. $\{r_i^{(\Delta)}\}_{i\in[M]} \leftarrow TreePRG(r^{(\Delta)})$
5. $\{r_i^{(party)}\}_{i\in[\tau]} \leftarrow TreePRG(r^{(party)})$

For each preprocessing $i \in [M]$:

1. Sample 3 random masks and create N additive shares for each:
   (a) $(mask_i^{(U)}, mask_i^{(V)}, mask_i^{(W)}) \leftarrow PRG(r_i^{(mask)})$, where $mask_i^{(U)}, mask_i^{(V)}, mask_i^{(V)} \in \mathbb{Z}_q$
   (b) $\forall k \in [n]$: Compute $\Delta_{i,k}^{(U)} = u_k - mask_i^{(U)}$; $\Delta_{i,k}^{(V)} = v_k - mask_i^{(V)}$; $\Delta_{i,k}^{(W)} = w_k - mask_i^{(W)}$
   (c) $\{[\![mask_i^{(U)}]\!]_j, [\![mask_i^{(V)}]\!]_j, [\![mask_i^{(W)}]\!]_j, r_{i,j}^{(mask-com)}\}_{j\in[N]} \leftarrow TreePRG(r_i^{(mask)})$;
   $[\![mask_i^{(U)}]\!]_N \leftarrow mask_i^{(U)} - \sum_{k=1}^{N-1} [\![mask_i^{(U)}]\!]_k$;
   $[\![mask_i^{(V)}]\!]_N \leftarrow mask_i^{(V)} - \sum_{k=1}^{N-1} [\![mask_i^{(V)}]\!]_k$;
   $[\![mask_i^{(W)}]\!]_N \leftarrow mask_i^{(W)} - \sum_{k=1}^{N-1} [\![mask_i^{(W)}]\!]_k$
2. Commit to mask shares and Deltas:
   (a) $\forall j \in [N]$: Compute $com_{i,j}^{(mask)} = Com(([\![mask_i^{(U)}]\!]_j, [\![mask_i^{(V)}]\!]_j, [\![mask_i^{(W)}]\!]_j); r_{i,j}^{(mask-com)})$
   (b) $\{r_{i,k}^{(\Delta)(U)}, r_{i,k}^{(\Delta)(V)}, r_{i,k}^{(\Delta)(W)}\}_{k\in[n]} \leftarrow TreePRG(r_i^{(\Delta)})$
   (c) $\forall k \in [n]$: Compute $com_{i,k}^{(\Delta)(U)} = Com(\Delta_{i,k}^{(U)}; r_{i,k}^{(\Delta)(U)})$; $com_{i,k}^{(\Delta)(V)} = Com(\Delta_{i,k}^{(V)}; r_{i,k}^{(\Delta)(V)})$; $com_{i,k}^{(\Delta)(W)} = Com(\Delta_{i,k}^{(W)}; r_{i,k}^{(\Delta)(W)})$
3. Permute and accumulate Delta commitments:
   (a) Sample three uniformly random permutations $\phi_i^{(U)}, \phi_i^{(V)}, \phi_i^{(W)}$ from the space of all permutations of size n.

(b) $(aux_i^{(U)}, acc_i^{(U)}) \leftarrow Acc.Eval(par, com^{(\Delta)(U)}_{i,\phi_i^{(U)}(1)}, \ldots, com^{(\Delta)(U)}_{i,\phi_i^{(U)}(n)})$;
$(aux_i^{(V)}, acc_i^{(V)}) \leftarrow Acc.Eval(par, com^{(\Delta)(V)}_{i,\phi_i^{(V)}(1)}, \ldots, com^{(\Delta)(V)}_{i,\phi_i^{(V)}(n)})$;
$(aux_i^{(W)}, acc_i^{(W)}) \leftarrow Acc.Eval(par, com^{(\Delta)(W)}_{i,\phi_i^{(W)}(1)}, \ldots, com^{(\Delta)(W)}_{i,\phi_i^{(W)}(n)})$

(c) Set $R_i = (com_{i,1}^{(mask)}, \ldots, com_{i,N}^{(mask)}, acc_i^{(U)}, acc_i^{(V)}, acc_i^{(W)})$

For each repetition $i \in [\tau]$:

1. Generate the parties' private inputs:
   (a) $\{r_{i,j}^{(party)}, r_{i,j}^{(party-com)}\}_{j\in[N]} \leftarrow TreePRG(r_i^{(party)})$
   (b) For each $j \in [N]$: $([\![u_\alpha]\!]_j^{(i)}, [\![v_\beta]\!]_j^{(i)}, [\![w_\gamma]\!]_j^{(i)}) \leftarrow TreePRG(r_{i,j}^{(party)})$, where $[\![u_\alpha]\!]_j^{(i)}, [\![v_\beta]\!]_j^{(i)}, [\![w_\gamma]\!]_j^{(i)} \in \mathbb{Z}_q$
2. Create auxiliary values:
   (a) Compute $\Delta u_\alpha^{(i)} = u_\alpha - \sum_{j=1}^N [\![u_\alpha]\!]_j^{(i)}$.
   (b) Compute $\Delta v_\beta^{(i)} = v_\beta - \sum_{j=1}^N [\![v_\beta]\!]_j^{(i)}$.
   (c) Compute $\Delta w_\gamma^{(i)} = w_\gamma - \sum_{j=1}^N [\![w_\gamma]\!]_j^{(i)}$.
3. Commit to the parties' inputs:
   (a) $\forall j \in [N]$: Compute $com_{i,j}^{(party)} = Com(r_{i,j}^{(party)}; r_{i,j}^{(party-com)})$
   (b) Set $I_i = (\Delta u_\alpha^{(i)}, \Delta v_\beta^{(i)}, \Delta w_\gamma^{(i)}, com_{i,1}^{(party)}, \ldots, com_{i,N}^{(party)})$.

Send the accumulators and all the commitments to the verifier:

1. Set $R = (R_1, \ldots, R_M)$.
2. Set $I = (I_1, \ldots, I_\tau)$.
3. Compute $h_1 = Hash(R, I)$.
4. Send h_1 to the verifier.

Round 2

Verifier

1. Receive h_1' from the prover.
2. Partition $[M]$ into subsets J, S by uniformly sampling $J \xleftarrow{\$} \{J \subset [M]; |J| = \tau\}$. From now on, we denote the i-th element of J by J_i.
3. Send J to the prover.

Round 3

Prover

Open all preprocessings in S:

$\forall i \in S$: Send $(r_i^{(mask)}, r_i^{(\Delta)}, \phi_i^{(U)}, \phi_i^{(V)}, \phi_i^{(W)})$ to the verifier.

For each repetition $i \in [\tau]$, use an MPC-in-the-head protocol to prove that $u_\alpha \in U$, $v_\beta \in V$, $w_\gamma \in W$ and $u_\alpha + v_\beta + w_\gamma = 0$:

1. Each party P_j gets private inputs: $[\![mask_{J_i}^{(U)}]\!]_j, [\![mask_{J_i}^{(V)}]\!]_j, [\![mask_{J_i}^{(W)}]\!]_j, [\![u_\alpha]\!]_j^{(i)}, [\![v_\beta]\!]_j^{(i)}, [\![w_\gamma]\!]_j^{(i)}$. Also, all parties and the verifier have public access to the sets U, V, W as well as to $\Delta_{J_i,\alpha}^{(U)}, \Delta_{J_i,\beta}^{(V)}, \Delta_{J_i,\gamma}^{(W)}$.
2. Each party P_j does the following:
   (a) Compute $[\![\Delta_{J_i,\alpha}^{(U)}]\!]_j = [\![u_\alpha]\!]_j^{(i)} - [\![mask_{J_i}^{(U)}]\!]_j$.
   (b) Compute $[\![\Delta_{J_i,\beta}^{(V)}]\!]_j = [\![v_\beta]\!]_j^{(i)} - [\![mask_{J_i}^{(V)}]\!]_j$.
   (c) Compute $[\![\Delta_{J_i,\gamma}^{(W)}]\!]_j = [\![w_\gamma]\!]_j^{(i)} - [\![mask_{J_i}^{(W)}]\!]_j$.
   (d) Compute $[\![t_i]\!]_j = [\![u_\alpha]\!]_j^{(i)} + [\![v_\beta]\!]_j^{(i)} + [\![w_\gamma]\!]_j^{(i)}$
3. All parties open $[\![t_i]\!], [\![\Delta_{J_i,\alpha}^{(U)}]\!], [\![\Delta_{J_i,\beta}^{(V)}]\!], [\![\Delta_{J_i,\gamma}^{(W)}]\!]$ to get $t_i, \Delta_{J_i,\alpha}^{(U)}, \Delta_{J_i,\beta}^{(V)}, \Delta_{J_i,\gamma}^{(W)}$ respectively.
4. Create membership proofs for $\Delta_{J_i,\alpha}^{(U)}, \Delta_{J_i,\beta}^{(V)}, \Delta_{J_i,\gamma}^{(W)}$:
   (a) Compute $\pi_i^{(U)} \leftarrow Acc.Proof(par, com^{(\Delta)(U)}_{J_i,\phi_{J_i}^{(U)}(\alpha)}, acc_{J_i}^{(U)}, aux_{J_i})$.
   (b) Compute $\pi_i^{(V)} \leftarrow Acc.Proof(par, com^{(\Delta)(V)}_{J_i,\phi_{J_i}^{(V)}(\beta)}, acc_{J_i}^{(V)}, aux_{J_i})$.
   (c) Compute $\pi_i^{(W)} \leftarrow Acc.Proof(par, com^{(\Delta)(W)}_{J_i,\phi_{J_i}^{(W)}(\gamma)}, acc_{J_i}^{(W)}, aux_{J_i})$.

Compute $h_2 = Hash(\{[\![t_i]\!], [\![\Delta_{J_i,\alpha}^{(U)}]\!], [\![\Delta_{J_i,\beta}^{(V)}]\!], [\![\Delta_{J_i,\gamma}^{(W)}]\!]\}_{i\in[\tau]})$

Send $(h_2, \{\pi_i^{(U)}, \pi_i^{(V)}, \pi_i^{(W)}, \Delta_{J_i,\alpha}^{(U)}, \Delta_{J_i,\beta}^{(V)}, \Delta_{J_i,\gamma}^{(W)}, r_{J_i,\alpha}^{(\Delta)(U)}, r_{J_i,\beta}^{(\Delta)(V)}, r_{J_i,\gamma}^{(\Delta)(W)}\}_{i\in[\tau]})$ to the verifier.

Round 4

Verifier

1. Receive $\{(r_i^{(mask)\prime}, r_i^{(\Delta)\prime}, \phi_i^{(U)\prime}, \phi_i^{(V)\prime}, \phi_i^{(W)\prime})\}_{i\in S}$ from the prover.
2. Receive $(h_2', \{\pi_i^{(U)\prime}, \pi_i^{(V)\prime}, \pi_i^{(W)\prime}, \Delta_{J_i,\alpha}^{(U)\prime}, \Delta_{J_i,\beta}^{(V)\prime}, \Delta_{J_i,\gamma}^{(W)\prime}, r_{J_i,\alpha}^{(\Delta)(U)\prime}, r_{J_i,\beta}^{(\Delta)(V)\prime}, r_{J_i,\gamma}^{(\Delta)(W)\prime}\}_{i\in[\tau]})$ from the prover.
3. $\forall i \in [\tau]$: Sample a uniformly random $j_i^* \leftarrow [N]$.
4. Send $\{j_i^*\}_{i\in[\tau]}$ to the prover.

Round 5

Prover

For each $i \in [\tau]$: Open all parties $\{P_j\}_{j\in[N]\setminus\{j_i^*\}}$ and send the necessary auxiliary information:

Send the following to the verifier:

- $\{r_{J_i,j}^{(mask-com)}\}_{j\in[N]\setminus\{j_i^*\}}$
- $\{[\![mask_{J_i}^{(U)}]\!]_j, [\![mask_{J_i}^{(V)}]\!]_j, [\![mask_{J_i}^{(W)}]\!]_j\}_{j\in[N]\setminus\{j_i^*\}}$
- $\{r_{i,j}^{(party)}, r_{i,j}^{(party-com)}\}_{j\in[N]\setminus\{j_i^*\}}$
- $com_{i,j_i^*}^{(party)}$
- $\Delta u_\alpha^{(i)}, \Delta v_\beta^{(i)}, \Delta w_\gamma^{(i)}$
- $(acc_{J_i}^{(U)}, acc_{J_i}^{(V)}, acc_{J_i}^{(W)})$
- $com_{J_i,j_i^*}^{(mask)}$

Verifier

Receive the following from the prover:

- $\{r_{J_i,j}^{(mask-com)\prime}\}_{j\in[N]\setminus\{j_i^*\}}$
- $\{[\![mask_{J_i}^{(U)}]\!]'_j, [\![mask_{J_i}^{(V)}]\!]'_j, [\![mask_{J_i}^{(W)}]\!]'_j\}_{j\in[N]\setminus\{j_i^*\}}$
- $\{r_{i,j}^{(party)\prime}, r_{i,j}^{(party-com)\prime}\}_{j\in[N]\setminus\{j_i^*\}}$
- $com_{i,j_i^*}^{(party)\prime}$
- $\Delta u_\alpha^{(i)\prime}, \Delta v_\beta^{(i)\prime}, \Delta w_\gamma^{(i)\prime}$
- $acc_{J_i}^{(U)\prime}, acc_{J_i}^{(V)\prime}, acc_{J_i}^{(W)\prime}$
- $com_{J_i,j_i^*}^{(mask)\prime}$

Compute the preprocessings in S:

For all $i \in S$:

1. $(mask_i^{(U)\prime}, mask_i^{(V)\prime}, mask_i^{(W)\prime}) \leftarrow PRG(r_i^{(mask)\prime})$.
2. $\forall k \in [n]$: Compute $\Delta_{i,k}^{(U)\prime} = u_k - mask_i^{(U)\prime}$; $\Delta_{i,k}^{(V)\prime} = v_k - mask_i^{(V)\prime}$; $\Delta_{i,k}^{(W)\prime} = w_k - mask_i^{(W)\prime}$
3. $\{[\![mask_i^{(U)}]\!]'_j, [\![mask_i^{(V)}]\!]'_j, [\![mask_i^{(W)}]\!]'_j, r_{i,j}^{(mask-com)\prime}\}_{j\in[N]} \leftarrow TreePRG(r_i^{(mask)\prime})$;
   $[\![mask_i^{(U)}]\!]'_N \leftarrow mask_i^{(U)\prime} - \sum_{k=1}^{N-1}[\![mask_i^{(U)}]\!]'_k$;
   $[\![mask_i^{(V)}]\!]'_N \leftarrow mask_i^{(V)\prime} - \sum_{k=1}^{N-1}[\![mask_i^{(V)}]\!]'_k$;
   $[\![mask_i^{(W)}]\!]'_N \leftarrow mask_i^{(W)\prime} - \sum_{k=1}^{N-1}[\![mask_i^{(W)}]\!]'_k$
4. $\forall j \in [N]$: Compute: $com_{i,j}^{(mask)\prime} = Com((([\![mask_i^{(U)}]\!]'_j, [\![mask_i^{(V)}]\!]'_j, [\![mask_i^{(W)}]\!]'_j); r_{i,j}^{(mask-com)\prime})$
5. $\{r_{i,k}^{(\Delta)(U)\prime}, r_{i,k}^{(\Delta)(V)\prime}, r_{i,k}^{(\Delta)(W)\prime}\}_{k\in[n]} \leftarrow TreePRG(r_i^{(\Delta)\prime})$
6. $\forall k \in [n]$: Compute: $com_{i,k}^{(\Delta)(U)\prime} = Com(\Delta_{i,k}^{(U)\prime}; r_{i,k}^{(\Delta)(U)\prime})$; $com_{i,k}^{(\Delta)(V)\prime} = Com(\Delta_{i,k}^{(V)\prime}; r_{i,k}^{(\Delta)(V)\prime})$; $com_{i,k}^{(\Delta)(W)\prime} = Com(\Delta_{i,k}^{(W)\prime}; r_{i,k}^{(\Delta)(W)\prime})$

7. $(aux_i^{(U)\prime}, acc_i^{(U)\prime}) \leftarrow Acc.Eval(par, com^{(\Delta)(U)}_{i,\phi_i^{(U)}(1)'}{}', \ldots, com^{(\Delta)(U)}_{i,\phi_i^{(U)}(n)'}{}')$;
   $(aux_i^{(V)\prime}, acc_i^{(V)\prime}) \leftarrow Acc.Eval(par, com^{(\Delta)(V)}_{i,\phi_i^{(V)}(1)'}{}', \ldots, com^{(\Delta)(V)}_{i,\phi_i^{(V)}(n)'}{}')$;
   $(aux_i^{(W)\prime}, acc_i^{(W)\prime}) \leftarrow Acc.Eval(par, com^{(\Delta)(W)}_{i,\phi_i^{(W)}(1)'}{}', \ldots, com^{(\Delta)(W)}_{i,\phi_i^{(W)}(n)'}{}')$
8. Set $R'_i = (com_{i,1}^{(mask)\prime}, \ldots, com_{i,N}^{(mask)\prime}, acc_i^{(U)\prime}, acc_i^{(V)\prime}, acc_i^{(W)\prime})$.

Verify the Delta memberships and the masks:

For all repetitions $i \in [\tau]$:

1. Compute: $com_{J_i,\alpha}^{(\Delta)(U)\prime} = Com(\Delta_{J_i,\alpha}^{(U)}{}'; r_{J_i,\alpha}^{(\Delta)(U)\prime})$; $com_{J_i,\beta}^{(\Delta)(V)\prime} = Com(\Delta_{J_i,\beta}^{(V)}{}'; r_{J_i,\beta}^{(\Delta)(V)\prime})$; $com_{J_i,\gamma}^{(\Delta)(W)\prime} = Com(\Delta_{J_i,\gamma}^{(W)\prime}; r_{J_i,\gamma}^{(\Delta)(W)\prime})$.
2. Check: $Acc.Verify(par, com_{J_i,\alpha}^{(\Delta)(U)\prime}, acc_{J_i}^{(U)\prime}, \pi_i^{(U)\prime}) \stackrel{?}{=} 1$; $Acc.Verify(par, com_{J_i,\beta}^{(\Delta)(V)\prime}, acc_{J_i}^{(V)\prime}, \pi_i^{(V)\prime}) \stackrel{?}{=} 1$; $Acc.Verify(par, com_{J_i,\gamma}^{(\Delta)(W)\prime}, acc_{J_i}^{(W)\prime}, \pi_i^{(W)\prime}) \stackrel{?}{=} 1$.
3. $\forall j \in [N] \backslash \{j_i^*\}$: Compute $com_{J_i,j}^{(mask)\prime} = Com((\llbracket mask_{J_i}^{(U)} \rrbracket'_j, \llbracket mask_{J_i}^{(V)} \rrbracket'_j, \llbracket mask_{J_i}^{(W)} \rrbracket'_j); r_{J_i,j}^{(mask-com)\prime})$.

For all $i \in J$: Set $R'_i = (com_{i,1}^{(mask)\prime}, \ldots, com_{i,N}^{(mask)\prime}, acc_i^{(U)\prime}, acc_i^{(V)\prime}, acc_i^{(W)\prime})$.

Set $R' = (R'_1, \ldots, R'_M)$.

Verify the parties' inputs:

For each repetition $i \in [\tau]$:

1. $\forall j \in [N] \setminus \{j_i^*\}$: Compute $com_{i,j}^{(party)\prime} = Com(r_{i,j}^{(party)\prime}; r_{i,j}^{(party-com)\prime})$
2. Set $I'_i = (\Delta u_\alpha^{(i)\prime}, \Delta v_\beta^{(i)\prime}, \Delta w_\gamma^{(i)\prime}, com_{i,1}^{(party)\prime}, \ldots, com_{i,N}^{(party)\prime})$.

Set $I' = (I'_1, \ldots, I'_\tau)$.

Check if $h'_1 \stackrel{?}{=} Hash(R', I')$.

For each iteration $i \in [\tau]$, verify the MPC-in-the-head protocol:

1. $\forall j \in [N] \setminus \{j_i^*\}$: Compute $(\llbracket u_\alpha \rrbracket_j^{(i)\prime}, \llbracket v_\beta \rrbracket_j^{(i)\prime}, \llbracket w_\gamma \rrbracket_j^{(i)\prime})' \leftarrow TreePRG(r_{i,j}^{(party)\prime})$.
2. $\forall j \in [N] \setminus \{j_i^*\}$: Compute $\llbracket \Delta_{J_i,\alpha}^{(U)} \rrbracket'_j = \llbracket u_\alpha \rrbracket_j^{(i)\prime} - \llbracket mask_{J_i}^{(U)} \rrbracket'_j$.
3. $\forall j \in [N] \setminus \{j_i^*\}$: Compute $\llbracket \Delta_{J_i,\beta}^{(V)} \rrbracket'_j = \llbracket v_\beta \rrbracket_j^{(i)\prime} - \llbracket mask_{J_i}^{(V)} \rrbracket'_j$.
4. $\forall j \in [N] \setminus \{j_i^*\}$: Compute $\llbracket \Delta_{J_i,\gamma}^{(W)} \rrbracket'_j = \llbracket v_\gamma \rrbracket_j^{(i)\prime} - \llbracket mask_{J_i}^{(W)} \rrbracket'_j$.
5. $\forall j \in [N] \setminus \{j_i^*\}$: $\llbracket t_i \rrbracket'_j = \llbracket u_\alpha \rrbracket_j^{(i)\prime} + \llbracket v_\beta \rrbracket_j^{(i)\prime} + \llbracket w_\gamma \rrbracket_j^{(i)\prime}$
6. Compute $\Delta t'_i = \Delta u_\alpha^{(i)\prime} + \Delta v_\beta^{(i)\prime} + \Delta w_\gamma^{(i)\prime}$
7. Compute $\llbracket \Delta_{J_i,\alpha}^{(U)} \rrbracket'_{j_i^*} = \Delta_{J_i,\alpha}^{(U)}{}' - \Delta u_\alpha^{(i)\prime} - \sum_{j=1, j\neq j_i^*}^{N} \llbracket \Delta_{J_i,\alpha}^{(U)} \rrbracket'_j$.

8. Compute $[\![\Delta^{(V)}_{J_i,\beta}]\!]'_{j_i^*} = \Delta^{(V)}_{J_i,\beta}{}' - \Delta v^{(i)}_{\beta}{}' - \sum_{j=1,j\neq j_i^*}^{N} [\![\Delta^{(V)}_{J_i,\beta}]\!]'_j$.

9. Compute $[\![\Delta^{(W)}_{J_i,\gamma}]\!]'_{j_i^*} = \Delta^{(W)}_{J_i,\gamma}{}' - \Delta w^{(i)}_{\gamma}{}' - \sum_{j=1,j\neq j_i^*}^{N} [\![\Delta^{(W)}_{J_i,\gamma}]\!]'_j$.

10. Compute $[\![t_i]\!]'_{j_i^*} = -\Delta t'_i - \sum_{j=1,j\neq j_i^*} [\![t_i]\!]'_j$.

Check if $h'_2 \stackrel{?}{=} Hash(\{[\![t_i]\!]', [\![\Delta^{(U)}_{J_i,\alpha}]\!]', [\![\Delta^{(V)}_{J_i,\beta}]\!]', [\![\Delta^{(W)}_{J_i,\gamma}]\!]'\}_{i\in[\tau]})$.

Return 1.

5.3.3 Communication Complexity

All notes on communication complexity that can be found in 5.2.3 also apply here. The exact communication complexity of our protocol is:

$$2\lambda + (M-\tau)5\lambda + \tau(log\tau + 2\lambda logN + 3\lambda logn + logN + 6logq + 8\lambda)$$

As in 5.2.3, taking $M = O(\tau), \tau = O(log^2 n), \lambda = log^2 n$ and $N = O(1)$, we achieve an asymptotic communication complexity:

$$O(log^5 n)$$

5.3.4 Computation Complexity

The asymptotic computation complexity of Protocol 5.3.2 is:

$$O(M(n+N)(\lambda + logq))$$

Using the values from 5.2.3 and the fact that $q = poly(n)$, we get an asymptotic computation complexity of:

$$O(nlog^4 n)$$

5.3.5 Completeness

Theorem 5.3.2. *A prover $\mathcal{P}$ who knows a solution $u_\alpha||v_\beta||w_\gamma$ to the $3SUM_{\mathbb{Z}_q}$ instance U, V, W and who follows the steps of Protocol 5.3.2 convinces the verifier $\mathcal{V}$ with probability* 1.

Proof. For any sampling of the random coins of $\mathcal{P}$ and $\mathcal{V}$, if the computation described in the protocol is honestly performed, then all the checks of $\mathcal{V}$ pass. Thus, the completeness error ϵ_c satisfies

$$\epsilon_c = 0$$

. □

5.3.6 Soundness

Soundness Error

Theorem 5.3.3. *A prover $\tilde{\mathcal{P}}$ who does not know a solution $u_\alpha||v_\beta||w_\gamma$ to the $3SUM_{\mathbb{Z}_q}$ instance U, V, W cannot convince the verifier $\mathcal{V}$ except with probability ϵ_s that satisfies:*

$$\epsilon_s \leq \max_A \left\{ \frac{\binom{M-A}{M-\tau}}{\binom{M}{M-\tau}} \cdot \left(\frac{1}{N}\right)^{\tau-A} \right\}$$

Proof. The soundness error can be computed in the exact same way as in 5.2.6, however in this protocol we do not use a batch product verification protocol, because all the relations that need to be proven are linear. Thus, the protocol never produces a false positive result assuming the prover honestly follows the protocol using a valid preprocessing and an invalid triple of group elements. This implies that the probability of successfully cheating in the MPC-in-the-head phase of each parallel execution without using a malicious preprocessing is upper bounded by $\frac{1}{N}$. The overall soundness error is:

$$\epsilon_s \leq \max_A \left\{ \frac{\binom{M-A}{M-\tau}}{\binom{M}{M-\tau}} \cdot \left(\frac{1}{N}\right)^{\tau-A} \right\}$$

□

Special Soundness

Theorem 5.3.4. *Let*

$$\epsilon_s = \max_A \left\{ \frac{\binom{M-A}{M-\tau}}{\binom{M}{M-\tau}} \cdot \left(\frac{1}{N}\right)^{\tau-A} \right\}$$

Suppose there is an efficient prover $\tilde{\mathcal{P}}$ such that

$$\tilde{\epsilon}_s := Pr[\langle \tilde{\mathcal{P}}(x), \mathcal{V}(x) \rangle (U, V, W) = 1] > \epsilon_s,$$

where U, V, W is a $3SUM_{\mathbb{Z}_q}$ instance. Then, there exists an extractor algorithm $\mathcal{E}$ which, given rewindable black-box access to $\tilde{\mathcal{P}}$, produces either a witness $u||v||w$ such that $u+v+w = 0 \ (mod\ q)$, $u \in U$, $v \in V$ and $w \in W$, a hash collision or a commitment collision, by making in average

$$\frac{4}{\tilde{\epsilon}_s - \epsilon_s} \cdot \left(1 + \tilde{\epsilon}_s \cdot \frac{8M}{\tilde{\epsilon}_s - \epsilon_s}\right)$$

calls to $\tilde{\mathcal{P}}$.

Proof. The extractor can output a valid witness by using three transcripts T_1, T_2, T_3 that satisfy the properties of 5.2.6. Specifically, $\mathcal{E}$ uses $RSP_2^{(1)}, RSP_2^{(2)}$ to obtain the shares $\{[\![u_\alpha]\!]_j^{(j_0)}, [\![v_\beta]\!]_j^{(j_0)}, [\![w_\gamma]\!]_j^{(j_0)}\}_{j \in [N]}$ and $\Delta u_\alpha^{(j_0)}, \Delta v_\beta^{(j_0)}, \Delta w_\gamma^{(j_0)}$. $\mathcal{E}$ is now able to compute a candidate solution $u'_\alpha, v'_\beta, w'_\gamma$ from $u_\alpha, v_\beta, w_\gamma$ using the additive secret sharing scheme as follows:

$$u'_\alpha = \Delta u_\alpha^{(j_0)} + \sum_{j=1}^{N} [\![u_\alpha]\!]_j^{(j_0)}$$

$$v'_\beta = \Delta v_\beta^{(j_0)} + \sum_{j=1}^{N} [\![v_\beta]\!]_j^{(j_0)}$$

$$w'_\gamma = \Delta w_\gamma^{(j_0)} + \sum_{j=1}^{N} [\![w_\gamma]\!]_j^{(j_0)}$$

The extractor algorithm now checks whether $u'_\alpha + v'_\beta + w'_\gamma = 0 \ (mod\ q)$. Then, it uses $RSP_1^{(3)}$ to check whether the group elements $u'_\alpha, v'_\beta, w'_\gamma$ have been masked and accumulated. If that also holds, $\mathcal{E}$ returns $u'_\alpha||v'_\beta||w'_\gamma$ as the witness.

In order to extract three transcripts T_1, T_2, T_3 with the required properties, we can, again, use the extraction prodecure described in the appendix E of [Fene22a] which provides an extraction algorithm that makes in average at most

$$\frac{4}{\tilde{\epsilon}_s - \epsilon_s} \cdot \left(1 + \tilde{\epsilon}_s \cdot \frac{8M}{\tilde{\epsilon}_s - \epsilon_s}\right)$$

calls to $\tilde{\mathcal{P}}$. □

5.3.7 Honest Verifier Zero Knowledge

Theorem 5.3.5. *Let the PRG and the TreePRG used in Protocol 5.2.2 be (t, ϵ_{PRG})-secure, the accumulator ACC be (t, ϵ_{ACC})-secure and the commitment scheme Com be (t, ϵ_{Com})-hiding. Then, there exists an efficient simulator $\mathcal{S}$ which, given random challenges $J, \{j_i^*\}_{i\in[\tau]}$, outputs a transcript which is $(t, \tau \cdot \epsilon_{PRG} + \tau \cdot \epsilon_{ACC} + \tau \cdot \epsilon_{Com})$-indistinguishable from a real transcript of Protocol 5.3.2.*

Proof. The simulator works almost in the same way as the simulator of Protocol 5.2.2. It behaves as follows:

1. Sample the verifier's challenges:
   (a) Randomly partition M into subsets S, J such that $|J| = \tau$ and $|S| = M - \tau$.
   (b) $\forall i \in [\tau]$: Sample uniformly random $j_i^* \xleftarrow{\$} N$.
2. Sample randomness:
   (a) $mseed \xleftarrow{\$} \{0,1\}^\lambda$
   (b) $(r^{(mask)}, r^{(\Delta)}, r^{(party)}) \leftarrow PRG(mseed)$
   (c) $\{r_i^{(mask)}\}_{i\in[M]} \leftarrow TreePRG(r^{(mask)})$
   (d) $\{r_i^{(\Delta)}\}_{i\in[M]} \leftarrow TreePRG(r^{(\Delta)})$
   (e) $\{r_i^{(party)}\}_{i\in[\tau]} \leftarrow TreePRG(r^{(party)})$
3. Sample random "witness" group elements: $\alpha, \beta, \gamma \xleftarrow{\$} [n]$.
4. Simulate the preprocessings that will be opened by the verifier. For each $i \in S$:
   (a) $(mask_i^{(U)}, mask_i^{(V)}, mask_i^{(W)}) \leftarrow PRG(r_i^{(mask)})$, where $mask_i^{(U)}, mask_i^{(V)}, mask_i^{(W)} \in \mathbb{Z}_q$
   (b) $\forall k \in [n]$: Compute $\Delta_{i,k}^{(U)} = u_k - mask_i^{(U)}$; $\Delta_{i,k}^{(V)} = v_k - mask_i^{(V)}$; $\Delta_{i,k}^{(W)} = w_k - mask_i^{(W)}$.
   (c) $\{[\![mask_i^{(U)}]\!]_j, [\![mask_i^{(V)}]\!]_j, [\![mask_i^{(W)}]\!]_j, r_{i,j}^{(mask-com)}\}_{j\in[N]} \leftarrow TreePRG(r_i^{(mask)})$;
   $[\![mask_i^{(U)}]\!]_N \leftarrow mask_i^{(U)} - \sum_{k=1}^{N-1} [\![mask_i^{(U)}]\!]_k$;
   $[\![mask_i^{(V)}]\!]_N \leftarrow mask_i^{(V)} - \sum_{k=1}^{N-1} [\![mask_i^{(V)}]\!]_k$;
   $[\![mask_i^{(W)}]\!]_N \leftarrow mask_i^{(W)} - \sum_{k=1}^{N-1} [\![mask_i^{(W)}]\!]_k$
   (d) $\forall j \in [N]$: Compute $com_{i,j}^{(mask)} = Com(([\![mask_i^{(U)}]\!]_j, [\![mask_i^{(V)}]\!]_j, [\![mask_i^{(W)}]\!]_j); r_{i,j}^{(mask-com)})$.
   (e) $\{r_{i,k}^{(\Delta)(U)}, r_{i,k}^{(\Delta)(V)}, r_{i,k}^{(\Delta)(W)}\}_{k\in[n]} \leftarrow TreePRG(r_i^{(\Delta)})$
   (f) $\forall k \in [n]$: Compute $com_{i,k}^{(\Delta)(U)} = Com(\Delta_{i,k}^{(U)}; r_{i,k}^{(\Delta)(U)})$; $com_{i,k}^{(\Delta)(V)} = Com(\Delta_{i,k}^{(V)}; r_{i,k}^{(\Delta)(V)})$; $com_{i,k}^{(\Delta)(W)} = Com(\Delta_{i,k}^{(W)}; r_{i,k}^{(\Delta)(W)})$.
   (g) Sample three uniformly random permutations $\phi_i^{(U)}, \phi_i^{(V)}, \phi_i^{(W)}$ from the space of all permutations of size n.

(h) $(aux_i^{(U)}, acc_i^{(U)}) \leftarrow Acc.Eval(par, com^{(\Delta)(U)}_{i,\phi_i^{(U)}(1)}, \ldots, com^{(\Delta)(U)}_{i,\phi_i^{(U)}(n)})$;
$(aux_i^{(V)}, acc_i^{(V)}) \leftarrow Acc.Eval(par, com^{(\Delta)(V)}_{i,\phi_i^{(V)}(1)}, \ldots, com^{(\Delta)(V)}_{i,\phi_i^{(V)}(n)})$;
$(aux_i^{(W)}, acc_i^{(W)}) \leftarrow Acc.Eval(par, com^{(\Delta)(W)}_{i,\phi_i^{(W)}(1)}, \ldots, com^{(\Delta)(W)}_{i,\phi_i^{(W)}(n)})$

(i) Set $R_i = (com_{i,1}^{(mask)}, \ldots, com_{i,N}^{(mask)}, acc_i^{(U)}, acc_i^{(V)}, acc_i^{(W)})$.

5. Simulate the preprocessings that will be used for the MPC-in-the-head protocol. For each $i \in J$:

(a) $(mask_i^{(U)}, mask_i^{(V)}, mask_i^{(W)}) \leftarrow PRG(r_i^{(mask)})$, where $mask_i^{(U)}, mask_i^{(V)}, mask_i^{(W)} \in \mathbb{Z}_q$

(b) Compute $\Delta_{i,\alpha}^{(U)} = u_\alpha - mask_i^{(U)}$; $\Delta_{i,\beta}^{(V)} = v_\beta - mask_i^{(V)}$; $\Delta_{i,\gamma}^{(W)} = w_\gamma - mask_i^{(W)}$.

(c) $\{[\![mask_i^{(U)}]\!]_j, [\![mask_i^{(V)}]\!]_j, [\![mask_i^{(W)}]\!]_j, r_{i,j}^{(mask-com)}\}_{j\in[N]} \leftarrow TreePRG(r_i^{(mask)})$;
$[\![mask_i^{(U)}]\!]_N \leftarrow mask_i^{(U)} - \sum_{k=1}^{N-1} [\![mask_i^{(U)}]\!]_k$;
$[\![mask_i^{(V)}]\!]_N \leftarrow mask_i^{(V)} - \sum_{k=1}^{N-1} [\![mask_i^{(V)}]\!]_k$;
$[\![mask_i^{(W)}]\!]_N \leftarrow mask_i^{(W)} - \sum_{k=1}^{N-1} [\![mask_i^{(W)}]\!]_k$

(d) $\forall j \in [N]\backslash\{j_i^*\}$: Compute $com_{i,j}^{(mask)} = Com(([\![mask_i^{(U)}]\!]_j, [\![mask_i^{(V)}]\!]_j, [\![mask_i^{(W)}]\!]_j); r_{i,j}^{(mask-com)})$

(e) Compute $com_{i,j_i^*}^{(mask)} = Com(0; r_{i,j_i^*}^{(mask-com)})$.

(f) $\{r_{i,k}^{(\Delta)(U)}, r_{i,k}^{(\Delta)(V)}, r_{i,k}^{(\Delta)(W)}\}_{k\in[n]} \leftarrow TreePRG(r_i^{(\Delta)})$

(g) $\forall k \in [n] \setminus \{\alpha\}$: Compute $com_{i,k}^{(\Delta)(U)} = Com(0; r_{i,k}^{(\Delta)(U)})$

(h) $\forall k \in [n] \setminus \{\beta\}$: Compute $com_{i,k}^{(\Delta)(V)} = Com(0; r_{i,k}^{(\Delta)(V)})$

(i) $\forall k \in [n] \setminus \{\gamma\}$: Compute $com_{i,k}^{(\Delta)(W)} = Com(0; r_{i,k}^{(\Delta)(W)})$

(j) Compute $com_{i,\alpha}^{(\Delta)(U)} = Com(\Delta_{i,\alpha}^{(U)}; r_{i,k}^{(\Delta)(U)})$; $com_{i,\beta}^{(\Delta)(V)} = Com(\Delta_{i,\beta}^{(V)}; r_{i,k}^{(\Delta)(V)})$; $com_{i,\gamma}^{(\Delta)(W)} = Com(\Delta_{i,\gamma}^{(W)}; r_{i,k}^{(\Delta)(W)})$.

(k) Sample three uniformly random permutations $\phi_i^{(U)}, \phi_i^{(V)}, \phi_i^{(W)}$ from the space of all permutations of size n.

(l) $(aux_i^{(U)}, acc_i^{(U)}) \leftarrow Acc.Eval(par, com^{(\Delta)(U)}_{i,\phi_i^{(U)}(1)}, \ldots, com^{(\Delta)(U)}_{i,\phi_i^{(U)}(n)})$;
$(aux_i^{(V)}, acc_i^{(V)}) \leftarrow Acc.Eval(par, com^{(\Delta)(V)}_{i,\phi_i^{(V)}(1)}, \ldots, com^{(\Delta)(V)}_{i,\phi_i^{(V)}(n)})$;
$(aux_i^{(W)}, acc_i^{(W)}) \leftarrow Acc.Eval(par, com^{(\Delta)(W)}_{i,\phi_i^{(W)}(1)}, \ldots, com^{(\Delta)(W)}_{i,\phi_i^{(W)}(n)})$

(m) Set $R_i = (com_{i,1}^{(mask)}, \ldots, com_{i,N}^{(mask)}, acc_i^{(U)}, acc_i^{(V)}, acc_i^{(W)})$.

6. Set $R = (R_1, \ldots, R_M)$.

7. Simulate the parties' inputs and the auxiliary variables. For each $i \in [\tau]$:

(a) $\{r_{i,j}^{(party)}, r_{i,j}^{(party-com)}\}_{j\in[N]} \leftarrow TreePRG(r_i^{(party)})$

(b) $\forall j \in [N] \setminus \{j_i^*\}$: $([\![u_\alpha]\!]_j^{(i)}, [\![v_\beta]\!]_j^{(i)}, [\![w_\gamma]\!]_j^{(i)}) \leftarrow TreePRG(r_{i,j}^{(party)})$

(c) $\forall j \in [N] \setminus \{j_i^*\}$: Compute $com_{i,j}^{(party)} = Com(r_{i,j}^{(party)}; r_{i,j}^{(party-com)})$.

(d) Compute $com_{i,j_i^*}^{(party)} = Com(0; r_{i,j_i^*}^{(party-com)})$.

(e) Sample uniformly random values $\Delta u_\alpha^{(i)}, \Delta v_\beta^{(i)}, \Delta w_\gamma^{(i)} \xleftarrow{\$} \mathbb{Z}_q$.

(f) Set $I_i = (\Delta u_\alpha^{(i)}, \Delta v_\beta^{(i)}, \Delta w_\gamma^{(i)}, com_{i,1}^{(party)}, \ldots, com_{i,N}^{(party)})$.

8. Set $I = (I_1, \ldots, I_\tau)$.
9. Compute $h_1 = Hash(R, I)$.
10. Simulate the MPC protocol. For each $i \in [\tau]$:
    (a) $\forall j \in [N] \setminus \{j_i^*\}$: Compute $[\![\Delta_{J_i,\alpha}^{(U)}]\!]_j = [\![u_\alpha]\!]_j^{(i)} - [\![mask_{J_i}^{(U)}]\!]_j$; $[\![\Delta_{J_i,\beta}^{(V)}]\!]_j = [\![v_\beta]\!]_j^{(i)} - [\![mask_{J_i}^{(V)}]\!]_j$; $[\![\Delta_{J_i,\gamma}^{(W)}]\!]_j = [\![w_\gamma]\!]_j^{(i)} - [\![mask_{J_i}^{(W)}]\!]_j$.
    (b) Compute $[\![\Delta_{J_i,\alpha}^{(U)}]\!]_{j_i^*} = \Delta_{J_i,\alpha}^{(U)} - \Delta u_\alpha^{(i)} - \sum_{j=1,j\neq j_i^*}^{N} [\![\Delta_{J_i,\alpha}^{(U)}]\!]_j$.
    (c) Compute $[\![\Delta_{J_i,\beta}^{(V)}]\!]_{j_i^*} = \Delta_{J_i,\beta}^{(V)} - \Delta v_\beta^{(i)} - \sum_{j=1,j\neq j_i^*}^{N} [\![\Delta_{J_i,\beta}^{(V)}]\!]_j$.
    (d) Compute $[\![\Delta_{J_i,\gamma}^{(W)}]\!]_{j_i^*} = \Delta_{J_i,\gamma}^{(W)} - \Delta w_\gamma^{(i)} - \sum_{j=1,j\neq j_i^*}^{N} [\![\Delta_{J_i,\gamma}^{(W)}]\!]_j$.
    (e) $\forall j \in [N] \setminus \{j_i^*\}$: Compute $[\![t_i]\!]_j = [\![u_\alpha]\!]_j^{(i)} + [\![v_\beta]\!]_j^{(i)} + [\![w_\gamma]\!]_j^{(i)}$
    (f) Compute $\Delta t_i = \Delta u_\alpha^{(i)} + \Delta v_\beta^{(i)} + \Delta w_\gamma^{(i)}$.
    (g) Compute $[\![t_i]\!]_{j_i^*} = -\Delta t_i - \sum_{j=1,j\neq j_i^*} [\![t_i]\!]_j$.

We can argue about the computational indistinguishability between the simulated transcript and the real transcript by using the arguments of 5.2.7. □

5.4 More Zero Knowledge Arguments Through Reductions

Being the core problems of fine-grained complexity, OV and $3SUM$ are of significant importance, because many other problems reduce to them. $3SUM$ is also closely related to a huge variety of problems in computational geometry [Gaje95]. In this section we will demonstrate how to get additional generic zero knowledge arguments for problems that reduce to OV or $3SUM$ without designing new protocols from scratch. We will use the problem *GeomBase* from computational geometry as an example. *GeomBase* is defined as follows:

Definition 5.4.1 (GeomBase)**.** Given three sets U, V, W of n points with integer coordinates from $\{-m, \ldots, m\}$ on three horizontal lines $y = 0$, $y = 1$, and $y = 2$ respectively, $GeomBase_m$ asks to determine whether there exists a non-horizontal line containing one point from each set.

It has been shown in [Gaje95] that *GeomBase* is equivalent to $3SUM$. Here, we are only interested in one direction, but for the other direction we can use the same reduction in reverse.

Theorem 5.4.1 ($GeomBase_m \leq_{n^2,n^2} 3SUM_{2m}$)**.** *$GeomBase_m$ on three sets of n points each is (n^2, n^2)-reducible to $3SUM_{2m}$ on three sets of n integers each.*

Proof. Let A be an algorithm that solves $3SUM_{2m}$ and U, V, W be the input to $GeomBase_m$. The reduction works as follows:

1. Create sets U', V', W':
    - $U' = \{u | (u, 0) \in U\}$
    - $V' = \{-2v | (v, 1) \in V\}$
    - $W' = \{w | (w, 2) \in W\}$
2. Call A on input U', V', W'.

3. If A returns a solution (u', v', w') output $((u', 0), (-\frac{v'}{2}, 1), (w', 2))$ otherwise output NO.

It is easy to see that the reduction works correctly. Let (u', v', w') be a solution to $3SUM_{2m}$. It holds that:

$$u' + v' + w' = 0 \iff u - 2v + w = 0 \iff \frac{u+w}{2} = v,$$

which implies that the points $(u, 0), (v, 1), (w, 2)$ that the reduction outputs are colinear. The reduction requires time $\tilde{O}(n)$, thus an $\tilde{O}(n^{2-\epsilon})$ time algorithm for $3SUM$ can be compiled into an $\tilde{O}(n^{2-\epsilon})$ time algorithm for $GeomBase$. □

We can now follow the chain of reductions

$$GeomBase_m \leq_{n^2,n^2} 3SUM_{2m} \leq_{n^2,n^2} 3SUM_{\mathbb{Z}_{6m+1}}$$

to get a zero knowledge argument for $GeomBase$.

Let U, V, W be the input to $GeomoBase_m$ and q be an integer modulus such that $q = 6m + 1$. We create sets U', V', W' as follows:

- $U' = \{u + 2m \ (mod\ q) | (u, 0) \in U\}$
- $V' = \{-2v + 2m \ (mod\ q) | (v, 1) \in V\}$
- $W' = \{w - 4m \ (mod\ q) | (w, 2) \in W\}$

The prover can now run Protocol 5.3.2 on input U', V', W'.

Chapter 6

Conclusion and Research Directions

6.1 Conclusion

Fine-grained cryptography is a fascinating area of research, which has attracted a lot of attention during the last years. This kind of cryptography lies in P which makes it usable even in very obscure settings such that $NP \subseteq BPP$. However, the lack of a fine-grained one way function is still a big obstacle towards achieving fine-grained cryptography. In this thesis, we made some progress on the field by constructing a family of functions ($\mathcal{F}EIP$), which is hard to compute in truly subquadratic time in the average case. Such a family of functions can be potentially turned into a fine-grained one way function in the future. The hardness of our functions is based on the worst case hardness of solving an Exact IP instance in truly subquadratic time. Exact IP is at least as hard as OV in the worst case and, to the best of our knowledge, it has not been proven whether they are equivalent in the dense setting. Basing hardness on this stronger assumption makes our result an improvement over $\mathcal{F}OV$ which has been defined in [Ball17a]. The same holds for our Proof of Work scheme which is based on $\mathcal{F}EIP$. We also studied how the notion of zero knowledge can be used in the fine-grained setting. Being able to prove knowledge of a solution for the core problems of fine-grained complexity is an essential building block for creating fine-grained cryptographic schemes and protocols. In this thesis, we gave a definition for zero knowledge arguments, which can be used for constructing zero knowledge protocols for problems in P. We also presented two zero knowledge arguments for the core problems of fine-grained complexity; OV and $3SUM$. We achieved a running time of $\tilde{O}(n)$ for both the prover and the verifier, which makes our protocols usable in practice under the assumptions that quadratic time problems are intractable. Finally, we showed how to use our protocols to obtain new generic zero knowledge arguments for other problems through fine-grained reductions. We used the problem $GeomBase$ from computational geometry as such an example.

6.2 Research Directions

We will now present some directions for future research:

- Can we turn $\mathcal{F}EIP$ into a fine-grained one way function? The authors of [Ball17a] suggested that constructing an AM protocol with a subquadratic time verifier for the underlying problem (Exact IP in this case) would be a big step towards that direction.
- It seems like memberships proofs, which are computation-heavy, are a core element of our protocols. Can we achieve an improvement in the complexity of Protocols 5.2.2 and 5.3.2?
- Can we create protocols with the same (or better) asymptotic complexity using a different technique other than MPC-in-the-head?
- Can we use our results to create fine-grained signatures and extend those to other types of signatures such as [Ball22] as well?

Bibliography

[Abbo14] Amir Abboud, Ryan Williams and Huacheng Yu, "More applications of the polynomial method to algorithm design", in *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, pp. 218–230, SIAM, 2014.

[Abbo15a] Amir Abboud, Arturs Backurs and Virginia Vassilevska Williams, "Quadratic-time hardness of LCS and other sequence similarity measures", *arXiv preprint arXiv:1501.07053*, 2015.

[Abbo15b] Amir Abboud, Virginia Vassilevska Williams and Huacheng Yu, "Matching triangles and basing hardness on an extremely popular conjecture", in *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pp. 41–50, 2015.

[Ajta83] Miklós Ajtai, "$\boxtimes$ 11-formulae on finite structures", *Annals of pure and applied logic*, vol. 24, no. 1, pp. 1–48, 1983.

[Appl06] Benny Applebaum, Yuval Ishai and Eyal Kushilevitz, "Cryptography in NC^0", *SIAM Journal on Computing*, vol. 36, no. 4, pp. 845–888, 2006.

[Back15] Arturs Backurs and Piotr Indyk, "Edit distance cannot be computed in strongly subquadratic time (unless SETH is false)", in *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pp. 51–58, 2015.

[Ball17a] Marshall Ball, Alon Rosen, Manuel Sabin and Prashant Nalini Vasudevan, "Average-case fine-grained hardness", in *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pp. 483–496, 2017.

[Ball17b] Marshall Ball, Alon Rosen, Manuel Sabin and Prashant Nalini Vasudevan, "Proofs of useful work", *Cryptology ePrint Archive*, 2017.

[Ball18] Marshall Ball, Alon Rosen, Manuel Sabin and Prashant Nalini Vasudevan, "Proofs of work from worst-case assumptions", in *Annual International Cryptology Conference*, pp. 789–819, Springer, 2018.

[Ball22] Danai Balla, Pourandokht Behrouz, Panagiotis Grontas, Aris Pagourtzis, Marianna Spyrakou and Giannis Vrettos, "Designated-Verifier Linkable Ring Signatures with Unconditional Anonymity", in *International Conference on Algebraic Informatics*, pp. 55–68, Springer, 2022.

[Bara08] Ilya Baran, Erik D Demaine and Mihai Pătraşcu, "Subquadratic algorithms for 3SUM", *Algorithmica*, vol. 50, no. 4, pp. 584–596, 2008.

[Baum20] Carsten Baum and Ariel Nof, "Concretely-efficient zero-knowledge arguments for arithmetic circuits and their application to lattice-based cryptography", in *IACR International Conference on Public-Key Cryptography*, pp. 495–526, Springer, 2020.

[Blum84] Manuel Blum and Silvio Micali, "How to generate cryptographically strong sequences of pseudorandom bits", *SIAM journal on Computing*, vol. 13, no. 4, pp. 850–864, 1984.

[Bogd06] Andrej Bogdanov, Luca Trevisan et al., "Average-case complexity", *Foundations and Trends® in Theoretical Computer Science*, vol. 2, no. 1, pp. 1–106, 2006.

[Bold12] Alexandra Boldyreva and Virendra Kumar, "A new pseudorandom generator from collision-resistant hash functions", in *Cryptographers' Track at the RSA Conference*, pp. 187–202, Springer, 2012.

[Brin15] Karl Bringmann and Marvin Künnemann, "Quadratic conditional lower bounds for string problems and dynamic time warping", in *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pp. 79–97, IEEE, 2015.

[Brzu22] Chris Brzuska and Geoffroy Couteau, "On Building Fine-Grained One-Way Functions from Strong Average-Case Hardness", in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 584–613, Springer, 2022.

[Cach97] Christian Cachin and Ueli Maurer, "Unconditional security against memory-bounded adversaries", in *Annual International Cryptology Conference*, pp. 292–306, Springer, 1997.

[Cane04] Ran Canetti, Oded Goldreich and Shai Halevi, "The random oracle methodology, revisited", *Journal of the ACM (JACM)*, vol. 51, no. 4, pp. 557–594, 2004.

[Carm16] Marco L Carmosino, Jiawei Gao, Russell Impagliazzo, Ivan Mihajlin, Ramamohan Paturi and Stefan Schneider, "Nondeterministic extensions of the strong exponential time hypothesis and consequences for non-reducibility", in *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, pp. 261–270, 2016.

[Chan15] Timothy M Chan and Moshe Lewenstein, "Clustered integer 3SUM via additive combinatorics", in *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pp. 31–40, 2015.

[Chen19] Lijie Chen and Ryan Williams, "An equivalence class for orthogonal vectors", in *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 21–40, SIAM, 2019.

[Cook71] Stephen A Cook, "The complexity of theorem-proving procedures", in *Proceedings of the third annual ACM symposium on Theory of computing*, pp. 151–158, 1971.

[Dali20] Mina Dalirrooyfard, Andrea Lincoln and Virginia Vassilevska Williams, "New techniques for proving fine-grained average-case hardness", in *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 774–785, IEEE, 2020.

[Degw16] Akshay Degwekar, Vinod Vaikuntanathan and Prashant Nalini Vasudevan, "Fine-grained cryptography", in *Annual International Cryptology Conference*, pp. 533–562, Springer, 2016.

[Dwor15] Morris J Dworkin et al., "SHA-3 standard: Permutation-based hash and extendable-output functions", 2015.

[Egas21] Shohei Egashira, Yuyu Wang and Keisuke Tanaka, "Fine-grained cryptography revisited", *Journal of Cryptology*, vol. 34, no. 3, pp. 1–43, 2021.

[Feig90] J Feigenbaum and L Fortnow, "On the random-self-reducibility of complete sets, university of chicago technical report 90-22", *Computer Science Department*, 1990.

[Fene22a] Thibauld Feneuil, Antoine Joux and Matthieu Rivain, "Shared permutation for syndrome decoding: New zero-knowledge protocol and code-based signature", *Designs, Codes and Cryptography*, pp. 1–46, 2022.

[Fene22b] Thibauld Feneuil, Jules Maire, Matthieu Rivain and Damien Vergnaud, "Zero-Knowledge Protocols for the Subset Sum Problem from MPC-in-the-Head with Rejection", *Cryptology ePrint Archive*, 2022.

[Furs84] Merrick Furst, James B Saxe and Michael Sipser, "Parity, circuits, and the polynomial-time hierarchy", *Mathematical systems theory*, vol. 17, no. 1, pp. 13–27, 1984.

[Gaje95] Anka Gajentaan and Mark H Overmars, "On a class of O (n2) problems in computational geometry", *Computational geometry*, vol. 5, no. 3, pp. 165–185, 1995.

[Gemm92] Peter Gemmell and Madhu Sudan, "Highly resilient correctors for polynomials", *Information processing letters*, vol. 43, no. 4, pp. 169–174, 1992.

[Goel22] Aarushi Goel, Matthew Green, Mathias Hall-Andersen and Gabriel Kaptchuk, "Efficient Set Membership Proofs using MPC-in-the-Head", *Proceedings on Privacy Enhancing Technologies*, vol. 2022, no. 2, pp. 304–324, 2022.

[Gold89] Shafi Goldwasser, Silvio Micali and Charles Rackoff, "The knowledge complexity of interactive proof systems", *SIAM Journal on computing*, vol. 18, no. 1, pp. 186–208, 1989.

[Gold20] Oded Goldreich and Guy N Rothblum, "Worst-case to Average-case reductions for subclasses of P", in *Computational Complexity and Property Testing*, pp. 249–295, Springer, 2020.

[Hast86] John Hastad, "Almost optimal lower bounds for small depth circuits", in *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pp. 6–20, 1986.

[Hast87] Johan Hastad, "One-way permutations in NC0", *Information Processing Letters*, vol. 26, no. 3, pp. 153–155, 1987.

[Hast90] Johan Håstad, "Pseudo-random generators under uniform assumptions", in *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pp. 395–404, 1990.

[Hast99] Johan Håstad, Russell Impagliazzo, Leonid A Levin and Michael Luby, "A pseudorandom generator from any one-way function", *SIAM Journal on Computing*, vol. 28, no. 4, pp. 1364–1396, 1999.

[Horo72] Ellis Horowitz, "A fast method for interpolation using preconditioning", *Information Processing Letters*, vol. 1, no. 4, pp. 157–163, 1972.

[Impa89a] Russell Impagliazzo, Leonid A Levin and Michael Luby, "Pseudo-random generation from one-way functions", in *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pp. 12–24, 1989.

[Impa89b] Russell Impagliazzo and Michael Luby, "One-way functions are essential for complexity based cryptography", in *30th Annual Symposium on Foundations of Computer Science*, pp. 230–235, IEEE Computer Society, 1989.

[Impa95] Russell Impagliazzo, "A personal view of average-case complexity", in *Proceedings of Structure in Complexity Theory. Tenth Annual IEEE Conference*, pp. 134–147, IEEE, 1995.

[Impa01a] Russell Impagliazzo and Ramamohan Paturi, "On the complexity of k-SAT", *Journal of Computer and System Sciences*, vol. 62, no. 2, pp. 367–375, 2001.

[Impa01b] Russell Impagliazzo, Ramamohan Paturi and Francis Zane, "Which problems have strongly exponential complexity?", *Journal of Computer and System Sciences*, vol. 63, no. 4, pp. 512–530, 2001.

[Isha09] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky and Amit Sahai, "Zero-knowledge proofs from secure multiparty computation", *SIAM Journal on Computing*, vol. 39, no. 3, pp. 1121–1152, 2009.

[Kale22] Daniel Kales and Greg Zaverucha, "Efficient Lifting for Shorter Zero-Knowledge Proofs and Post-Quantum Signatures", *Cryptology ePrint Archive*, 2022.

[Karp72] Richard M Karp, "Reducibility among combinatorial problems", in *Complexity of computer computations*, pp. 85–103, Springer, 1972.

[Katz20] Jonathan Katz and Yehuda Lindell, *Introduction to modern cryptography*, CRC press, 2020.

[Laga87] Jeffrey C Lagarias and Andrew M. Odlyzko, "Computing π (x): An analytic method", *Journal of Algorithms*, vol. 8, no. 2, pp. 173–191, 1987.

[Levi73] Leonid Anatolevich Levin, "Universal sequential search problems", *Problemy peredachi informatsii*, vol. 9, no. 3, pp. 115–116, 1973.

[Levi86] Leonid A Levin, "Average case complete problems", *SIAM Journal on Computing*, vol. 15, no. 1, pp. 285–286, 1986.

[Lind17] Yehuda Lindell and Ariel Nof, "A framework for constructing fast MPC over arithmetic circuits with malicious adversaries and an honest-majority", in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 259–276, 2017.

[Lipt91] Richard Lipton, "New directions in testing", *Distributed computing and cryptography*, vol. 2, pp. 191–202, 1991.

[Maur92] Ueli M Maurer, "Conditionally-perfect secrecy and a provably-secure randomized cipher", *Journal of Cryptology*, vol. 5, no. 1, pp. 53–66, 1992.

[Merk78] Ralph C Merkle, "Secure communications over insecure channels", *Communications of the ACM*, vol. 21, no. 4, pp. 294–299, 1978.

[Merk87] Ralph C Merkle, "A digital signature based on a conventional encryption function", in *Conference on the theory and application of cryptographic techniques*, pp. 369–378, Springer, 1987.

[Patr10] Mihai Patrascu, "Towards polynomial lower bounds for dynamic problems", in *Proceedings of the forty-second ACM symposium on Theory of computing*, pp. 603–610, 2010.

[Rodi04] Liam Roditty and Uri Zwick, "On dynamic shortest paths problems", in *European Symposium on Algorithms*, pp. 580–591, Springer, 2004.

[Schw80] Jacob T Schwartz, "Fast probabilistic algorithms for verification of polynomial identities", *Journal of the ACM (JACM)*, vol. 27, no. 4, pp. 701–717, 1980.

[Vass15] Virginia Vassilevska Williams, "Hardness of easy problems: Basing hardness on popular conjectures such as the strong exponential time hypothesis (invited talk)", in *10th International Symposium on Parameterized and Exact Computation (IPEC 2015)*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

[Will05] Ryan Williams, "A new algorithm for optimal 2-constraint satisfaction and its implications", *Theoretical Computer Science*, vol. 348, no. 2-3, pp. 357–365, 2005.

[Will10] Virginia Vassilevska Williams and Ryan Williams, "Subcubic equivalences between path, matrix and triangle problems", in *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pp. 645–654, IEEE, 2010.

[Will13] Virginia Vassilevska Williams and Ryan Williams, "Finding, minimizing, and counting weighted subgraphs", *SIAM Journal on Computing*, vol. 42, no. 3, pp. 831–854, 2013.

[Will14] Ryan Williams, "Faster all-pairs shortest paths via circuit complexity", in *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pp. 664–673, 2014.

[Will16] Ryan Williams, "Strong ETH breaks with Merlin and Arthur: Short non-interactive proofs of batch evaluation", *arXiv preprint arXiv:1601.04743*, 2016.

[Yao82] Andrew C Yao, "Theory and application of trapdoor functions", in *23rd Annual Symposium on Foundations of Computer Science (SFCS 1982)*, pp. 80–91, IEEE, 1982.

[Yao86] Andrew Chi-Chih Yao, "How to generate and exchange secrets", in *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pp. 162–167, IEEE, 1986.

[Zipp79] Richard Zippel, "Probabilistic algorithms for sparse polynomials", in *International symposium on symbolic and algebraic manipulation*, pp. 216–226, Springer, 1979.