Computing with private information: Techniques from Cryptology

Léonard Assouline

PSL
UNIVERSITÉ PARIS

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée à l'École Normale Supérieure de Paris

Computing with private information: Techniques from Cryptology

Soutenue par
Léonard Assouline
Le 1er Décembre 2023

École doctorale n°386
Sciences Mathématiques de Paris Centre

Spécialité
Informatique

Composition du jury :

Matthieu Rivain CryptoExperts	*Rapporteur*
Damien Vergnaud Sorbonne Université	*Rapporteur*
Cécile Delerablée Leanear	*Examinatrice*
Louis Goubin Université Versailles Saint-Quentin-en-Yvelines	*Examinateur*
Pierre-Alain Fouque Université Rennes 1	*Examinateur*
David Naccache ENS	*Examinateur*
Michel Abdalla ENS	*Directeur de thèse*
Brice Minaud ENS	*Directeur de thèse*

1794
ENS | PSL

Résumé

Dans cette thèse, nous étudions différentes méthodes permettant de faire des calculs sur de l'information cachée. Nous présentons de nouvelles constructions de protocoles, allant de l'amélioration des complexités de l'état de l'art à l'ajout de fonctionnalités. Nous analysons également la sécurité d'autres protocoles, et présentons des attaques dessus.

En premier lieu, nous nous intéressons au Private Simultaneous Messages, où un groupe d'individus souhaite faire apprendre à une entité externe le résultat d'un calcul sur leurs données privées. En utilisant un nouveau cadre, nous montrons de nouvelles constructions avec un coût en communication plus bas que celles déjà existantes. Ensuite, nous étudions des protocoles d'Oblivious RAM, qui dissimulent le motif d'accès qu'un client fait lorsqu'il accède à des données privées stockées sur un serveur non fiable. Nous présentons les premières constructions permettant au client d'enregistrer des objets de tailles arbitrairement différentes. Nous montrons également comment ces constructions permettent de construire des protocoles de Chiffrement Recherchable. Enfin, nous nous intéressons à la cryptographie en boîte-blanche, qui a pour objectif de donner à un appareil non fiable un accès sans restrictions à un algorithme de chiffrement, tout en empêchant la fuite de la clé secrète. Nous présentons une cryptanalyse de la construction de Ranea et al. de CRYPTO 2022, où des cryptosystèmes en boîte blanche sont donnés à partir de chiffrements symétriques basés sur des SPN et ARX. Nous prouvons que ce système n'est pas sûr en présentant une attaque structurelle de récupération de clé.

Mots-clés ⋆ PSM, MPC, Boîte-Blanche, ORAM, SSE, Cryptanalyse

Abstract

In this thesis, we study different methods of computing on hidden information. We present new constructions of protocols, which either improve on their state-of-the-art complexities or enable them to do more. We also analyse the security of other protocols and give attacks on them.

The first subject we consider is that of Private Simultaneous Messages, where a group of individuals wish to have an external entity learn the output of a computation on their private data. Using a new framework, we show new constructions with a smaller communication cost than existing ones. We then study Oblivious RAM protocols, that hide the access pattern that a client makes when accessing private data stored on an untrusted server. We give the first constructions that allow the client to store data items of arbitrarily different sizes. We also show how using these constructions yield new protocols for Searchable Encryption. We finish by studying white-box cryptography, where the goal is to give an untrusted device unrestricted access to an encrytpion algorithm, while preventing the secret key from leaking. We present a cryptanalysis of Ranea et al's construction from CRYPTO 2022, where they give white-box ciphers from SPN and ARX based symmetric cryptosystems. We show that this scheme is insecure by providing a structural key-recovery attack.

Keywords ⋆ PSM, MPC, White-Box, ORAM, SSE, Cryptanalysis

Remerciements

Je souhaite commencer en remerciant mes directeurs de thèse : merci à Brice pour ses conseils, son savoir et sa gentillesse. Je remercie également Michel pour son guidage, son expérience et pour avoir mis cette thèse sur les rails. Un grand merci à Damien et Matthieu pour avoir accepté d'être les rapporteurs de ce manuscrit, et pour leurs commentaires.

L'équipe Cascade du DI de l'ENS est le laboratoire révé pour faire une thèse, je tiens à remercier tous ses membres, en commençant par Lise-Marie, Valérie, Linda, Fanny, Mohamed, Ludovic et Jacques. Merci à Céline, Phong et David pour les discussions. Merci à Michael pour son avoir été un bro, à Lénaïck pour son enthousiasme, thank you Huy for the ping-pong games. Je remmercie Antoine pour les ragots industriels, Henry pour les démonstrations, Guirec pour chaque débat, merci Baptiste, Nicolas, merci à Ky pour bzzzz. Merci Hugo pour les rigolades, Paola pour son esprit, thank you Robert for your didactism, thank you Jianwei, merci Florette, Wissam, Yann.

Merci à Pierre, compère cryptologue depuis la L3, à mes camarades lyonnais, Ulysse et ceux du gslq, Ouassim, Gauthier, Morgane, Martin, Baptiste, Fënril, Solal, Côme, Thomas, Pierre et Jacquelin. Merci à Arnault pour nos reflexions sur le doctorat et le milieu académique, Antoine, Valentin, Emma, Baptiste, Pascal, Madeleine, Billy, Eve. Je remercie également Elie, Vladimir, Araxe, Louise, Roméo, Baptiste.

Merci à Igor, Françoise et Louise. Un grand merci également à ma grand-mère pour les histoires de Paris à travers les époques. Merci à mes parents, qui m'ont toujours aiguillé vers la bonne direction, et Gabriel de remarquer les choses. Merci à Ginza de me tenir compagnie pendant que je termine de rédiger ce manuscrit.

Enfin, merci Jeanne de me faire avancer depuis bien avant le début de cette thèse.

Contents

Chapter 1

Introduction en Français

L'Informatique est la science de l'information : les différentes sous-disciplines de cette science étudient les manières de créer l'information, de la stocker, de la transformer... La cryptologie est l'étude de l'information secrète. Nous faisons la distinction entre deux sujets en cryptologie : la cryptographie cherche à cacher de l'information, la cryptanalyse vise à récupérer de l'information cachée.

Dans cette thèse, nous présentons des résultats s'intégrant dans chacun de ces deux sujets. Un fil conducteur de notre travail concerne des défis contemporains, qui existent car nous vivons dans une époque avec internet, des accès à distance et du calcul délégué. Cette ère est parfois appelée *l'Âge de l'Information*.

Avant le début de cet âge de l'information (qui commence à se former après la Seconde Guerre Mondiale), la cryptologie était utilisée pour cacher un message (par chiffrement), devant rester masqué jusqu'à ce que le destinataire retire le voile servant à dissimuler le message (le déchiffrement), et puisse le lire. Les usages les plus courants étaient alors de transmettre des ordres en temps de guerre, de comploter entre alliés politiques ou d'aider des actes d'espionnage. Le fait que les messages puissent rester masqués était crucial : les États-Unis ont rejoint la Triple-Entente pendant la Première Guerre Mondiale en grande partie grâce à la décryption par des cryptanalystes britanniques du *télégramme Zimmerman*, où l'Allemagne demandait au Mexique de déclarer la guerre à leur voisin du Nord. Marie Stuart, reine d'Écosse, a été exécutée à cause de son implication dans un complot visant à assassiner la reine Elizabeth Ière, découverte lorsque que le chiffre qu'elle utilisait a été décrypté. Ethel et Julius Rosenberg ont transféré les plans d'armes atomiques à l'Union Sovétique. Leur responsabilité a été découverte par le programme secret Venona, où des cryptanalystes américains ont brisé le chiffrement utilisé par les soviétiques.

Une cryptographie robuste était une question de vie ou de mort.

Aujourd'hui, bien qu'il y ait toujours des guerres, complots d'assassinat et cercles d'espionnage, l'utilisation de la cryptographie est bien plus répandue. Le fil d'Ariane de données que nous laissons derrière nous dans la vie de tous les jours au XXIème siècle est immense, et fait fuiter beaucoup d'information sur nous. C'est en partie fait intentionnellement lorsque l'on publie quelque chose sur un réseau social. Cependant, l'écrasante majorité de cette fuite de données trouve son origine en coulisses : des métadonnées fuitent dès que l'on fait le moindre mouvement dans le cyberespace.

Imaginons un arbitre au milieu d'un terrain de football, tourné vers la moitié Est du stade. Soudainement, un ballon – frappé dans la moitié Ouest – entre dans son champ de vision et termine sa course sur le gazon. Si cet arbitre possède quelques connaissances en mécanique Newtonienne, il peut – étant donné deux positions distinctes occupées par le ballon à deux instants différents – calculer la position exacte à partir de laquelle le ballon a été lancé.

N'importe qui peut faire la même chose en regardant les effets de bord d'une action en ligne, et déduire de ces métadonnées de l'information qui aurait potentiellement dû rester secrète. Une des questions qu'un cryptographe peut se poser est "comment puis-je tirer un ballon de football sans que quiconque ne puisse retracer cet événement à moi ?"

1.1 Cryptographie symétrique et asymétrique

Nous présentons maintenant deux personnages bien connus dans le folklore cryptologique : Alice souhaite envoyer de l'information sensible à son ami Bob, en passant par un canal non-sécurisé (i.e. quelqu'un est susceptible d'écouter le trafic). Pendant longtemps la cryptographie était limitée au *chiffrement symétrique* : Alice et Bob partagent une *clé secrète* K, Alice *chiffre* son message m : elle utilise la clé pour transformer m en un *chiffré* c initelligible. Elle envoie c à Bob, qui quant à lui *déchiffre* c à l'aide de K, et obtient le message originel m.

Ce procédé – aussi appelé chiffrement à *clé secrète* ou à *clé privée* – nécessite un accord préalable entre Alice et Bob sur la clé K qu'ils se partagent. C'est une limite, en particulier si Alice et Bob ne se connaîssent ni d'Ève ni d'Adam et doivent établir un premier contact de manière sécurisée.

La solution à ce problème a été trouvée dans les années soixante-dix : en utilisant de la *cryptographie asymétrique*, Bob peut publier sa clé publique PK, celle-ci est utilisée par Alice pour chiffrer m. Elle envoie ensuite le chiffré c à Bob, qui le déchiffre avec sa clé secrète SK.

Également appelé *chiffrement à clé publique*, ce nouveau genre de protocole ouvre la voie vers la cryptographie moderne : Alice et Bob peuvent utiliser des protocoles à clé publique, ainsi que d'autres outils, pour faire une "poignée de main" numérique et se mettre d'accord sur une clé symétrique, Alice peut aussi signer ses messages afin de prouver qu'elle en est bien l'auteure, elle peut assurer leur intégrité afin d'empêcher quiconque d'en altérer le contenu...

Dans cette thèse, nous étudions trois différents types de protocoles en lien avec le chiffrement symétrique et asymétrique, et nous apportons des contributions dans leurs disciplines respectives. Un fil conducteur entre ces sujets est de faciliter les calculs délégués sécurisés.

1.2 Laisser le calcul à un tiers

La découverte de la cryptographie à clé publique va de pair avec l'arrivée de l'Âge de l'Information. Avant l'utilisation répandue d'ordinateurs connectés les uns aux autres dans un réseau, la cryptographie symétrique était suffisante pour ses applications militaires et diplomatiques. Il est logique que les membres d'une même nation (par exemple deux généraux discutant de stratégies) partagent la même clé secrète. Des précautions sont en place pour éviter de révéler cette clé à l'ennemi, mais il n'apparaît pas comme indispensable de créer de toutes nouvelles bases théoriques pout ça. Une anecdote illustrant cette attitude est le fait que des cryptologues du GCHQ – une agence de renseignements britannique – ont inventé un algorithme similaire au cryptosystème RSA (l'algorithme à clé publique le plus utilisé aujourd'hui) entre cinq et dix ans avant la publication de celui-ci. L'idée a été mise de côté à cause de son manque d'applications militaires.

Actuellement, nous avons accès à une grande puissance de calcul et de la cryptographie disponible publiquement. Alice peut choisir parmi des dizaines voire des centaines de cryptosystèmes pour discuter en sécurité avec Bob. Mais les besoins d'Alice ont aussi évolué : elle peut souhaiter vouloir faire plus que simplement bavarder avec ses amis. Elle pourrait avoir un grand jeu de données sensibles mais pas la place pour les stocker. Elle pourrait aussi vouloir faire des calculs sur ces données mais sans posséder une puissance de calcul adéquate. Heureusement, l'offre s'est adaptée à sa demande : elle peut externaliser le stockage et le calcul dont elle a besoin. Si Alice ne fait confiance à personne, comment ses données peuvent-elles rester sécurisées ?

Laisser une entité externe s'occuper de calculs sur des données confidentielles peut se faire de plusieurs façons. Le chiffrement totalement homomorphe est une solution généraliste pour permettre de faire des calculs sur des données chiffrées : n'importe qui peut calculer une fonction arbitraire sur un ensemble de chiffrés, resultant en un chiffré de la sortie de la fonction appliquée sur les messages en clair d'origine. Le chiffrement totalement homomorphe nécessite une grande capacité de calcul et de stockage mais demeure une bonne solution prête à l'emploi. Alice pourraît aussi *obfusquer* un programme avec de l'obfuscation indistinguable, c'est-à-dire (en grossissant le trait) le transformé de telle sorte que quelqu'un faisant tourner le programme obfusqué obtient les mêmes paires entrée-sortie qu'avec le programme d'origine, mais sans savoir comment les rouages de ce programme fonctionnent.

Nous allons maintenant nous intéresser à différentes propositions qui permettent de partager de l'information secrète entre plusieurs parties. Nous commencerons avec un cas particulier de calcul multi-partie appelé *Private Simultaneous Messages*[1] (PSM), où un arbitre calcule une fonction publique prenant en entrée des valeurs secrètes possédées par les parties avec lesquelles il intéragit. Puis nous verrons comment une personne peut stocker et accèder à ses données de manière sécurisée sur un serveur non-fiable en utilisant des protocoles d'*Oblivious RAM* (ORAM). Enfin, nous étudierons la *Cryptographie Boîte-Blanche*, où un algorithme à clé secrète est fait pour être évalué publiquement, et nous montrerons une attaque sur un tel protocole.

1.3 Différentes saveurs de calcul délégué

Durant ma thèse, j'ai travaillé sur des sujets divers dans différents domaines de la cryptologie. Je présente dans ce manuscrit des résultats dans trois de ces sujets, qui font partie du contenu de publications. J'introduis et présente une motivation derrière ces sujets dans cette section. Dans la suivante, je donnerai un résumé des contributions qui ont été apportées.

1.3.1 Calculer avec d'autres

Mettre les données de différentes parties en commun afin d'augmenter sa connaissance est souvent nécessaire. Un exemple typique de notre domaine est le problème des millionaires

[1] L'Informatique est une science jeune, par conséquent beaucoup de termes sont introduits pour nommer des concepts récents. Il est préférable d'employer la terminologie en français quand nous parlons de cryptologie dans cette langue, cependant certains domaines sont peu connus et le vocabulaire associé n'existe pour l'instant qu'en anglais. Plutôt que de proposer une traduction qui ne ferait pas consensus, ce manuscrit fait le choix de garder les termes en anglais dans ces cas là. D'une façon analogue, on parle de "travelling" au cinéma.

de Yao : Alice et Bob veulent savoir lequel parmi eux est le plus riche mais ils ne veulent pas révéler précisément le montant de leurs fortunes [Yao82]. Ce problème peut sembler superficiel, mais l'existence d'un protocole qui le résout est en réalité significatif. Nous pouvons imaginer une autre situation : une entreprise pharmaceutique cherche à collecter des données l'assistant dans le développement d'un nouveau traitement qui pourrait aider les patients souffrant d'une maladie. Les hôpitaux ont des grandes quantités de données potentiellement utiles sur leurs patients (par exemple l'âge, le sexe, le poids, le taux de mortalité ...), néanmoins laisser carte blanche à une entreprise pour utiliser les données des patients est une catastrophe de sécurité en devenir, et c'est à juste titre illégal dans la plupart des pays. La solution naïve serait de simplement anonymiser les données (i.e. en ôtant les noms d'une base de données sans la modifier davantage, un procédé aussi appelé "pseudonymiser"), cela ne fonctionne pas [NS06, Swe00]. Il peut rester assez de bits d'information dans une telle base de données pour identifier personnellement les personnes qui en font partie. Nous pouvons nous tourner vers la cryptographie pour notre salut : dans ce cas toute l'information est cachée, pas seulement les noms et lieux de naissance, qui sont des identifiants suffisants mais pas nécessaires.

Si nous chiffrons toutes ces données, comment pouvons-nous faire des calculs dessus ? Une façon de le faire est par le calcul multipartie sécurisé (*Multi-Party Computation* en anglais, ou MPC).

Dans un protocole MPC, des parties $P_1, \ldots, P_k$ détiennent chacune une entrée secrète $x_1, \ldots, x_k$, et elles souhaitent calculer le résultat d'une fonction (publique) f, de façon à ce que chaque partie apprenne $f(x_1, \ldots, x_k)$ et aucune autre information sur les autres entrées. Le MPC recouvre un vaste domaine de protocoles, avec différentes exigences de sécurité. Pour aller plus loin, cette excellente étude par Yehuda Lindell plonge dans les détails du MPC [Lin20]. Nous nous intéresserons dans cette thèse à une instance minimaliste du MPC appelée Private Simultaneous Messages (PSM).

Là, en plus des k parties, nous ajoutons un *arbitre* qui sera celui qui exécutera le calcul. Le protocole est *non-interactif* : chaque partie envoie un unique message à l'arbitre et les parties ne communiquent pas entre elles. Elles partagent uniquement une *chaîne de caractères aléatoire en commun* qu'elles utilisent pour cacher leurs entrées à l'arbitre. Ce protocole suppose que seul l'arbitre puisse être corrompu, les autres parties demeurent honnêtes. Les entrées des parties seront cachées au sens de la *théorie de l'information* (aussi appelé *sécurité parfaite*) : peu importe la puissance de l'adversaire, aucune information ne peut être obtenue sur les entrées en étudiant les messages reçus par l'arbitre.

1.3.2 Stocker ses données ailleurs

La quantité de données produites augmente indéfiniment et la capacité à faire des calculs sur ces données (par exemple pour entraîner un model de Machine Learning) a une importance cruciale. Bien que la capacité de stockage ait également augmenté, l'espace requis pour entreposer la quantité de données nécessaires à la réalisation de telles tâches est hors de la portée de la plupart des particuliers ou des petites entrepises. Pour externaliser le stockage de ces données de manière sécurisée, la solution naïve serait de chiffrer les données et de télécharger les chiffrés vers le cloud. Cette méthode est aussi sûre que l'est le système symétrique utilisé (i.e. aussi sécurisé que faire se peut si l'on choisit judicieusement), ce qui est une bonne nouvelle si on n'est intéressé que par la sécurité de nos données, quelqu'en soit le coût. Mais c'est comme dire que la meilleure façon d'éviter un accident de voiture est de ne jamais mettre le contact : c'est techniquement correct, mais

ça ne nous fait pas avancer. Si on chiffre un téraoctet de données avec AES et que nous le stockons, nous devrons télécharger un téraoctet de chiffré même si on désire ne lire qu'un octet du message clair. Cela devient rédhibitoire dès que l'on veut mettre cette solution en pratique dans une application de la vie réelle.

Pour avoir le beurre et l'argent du beurre, pourquoi ne pouvons-nous pas juste chiffrer de "petits" morceaux de nos données, stocker ces chiffrés en ligne et ne télécharger que certains morceaux lorsque nous en avons besoin ? C'est possible, mais cette solution apporte de nouveaux problèmes. En premier lieu, nous devons indexer les morceaux que nous stockons et cela a un prix, mais nous pouvons gérer ceci de plusieurs manières. En second lieu, le fait que nous accédons à un morceau en particulier révèle *en soi* de l'information sur les données auxquelles on accède et la façon dont on les utilise. Imaginons un enfant allant se servir un biscuit dans la cuisine, et cachant celui-ci derrière son dos lorsque sa mère vient le voir. Bien entendu, elle ne voit pas ce qu'il tient dans la main, cependant le fait qu'il cache quelque chose, dans la cuisine, alors qu'elle lui avait formellement interdit de manger un autre biscuit, lui permet de deviner ce que son fils lui cache.

Pour empêcher qui que ce soit de deviner ce que l'on fait avec nos données, les algorithmes d'Oblivious RAM (ORAM) ont été inventés dans les années quatre-vingt-dix : ils cachent le *motif d'accès* que l'on fait lorsque l'on envoie une requête pour des données. Originellement, le model ORAM concernait la vraie RAM : si un processeur envoie des requêtes vers des données (chiffrées) dans sa *mémoire vive*, quelqu'un analysant le trafic pourraît étudier le motif d'accès et apprendre de l'information confidentielle sur le calcul en train d'être fait. Récemment, ce cadre a été appliqué dans le cas de communications entre client et serveur dans le cloud. Néanmoins, il existe toujours des applications dans le modèle initial de la RAM, ou bien dans des mélanges en ligne et hors ligne, par exemple lorsque des calculs ont lieu dans des enclaves sécurisées sur un serveur à distance [Con22].

Dans cette thèse, nous donnons de nouvelles constructions de protocoles d'ORAM et les utilisons pour construire du chiffrement symétrique recherchable (Searchable Symmetric Encryption ou SSE en anglais). Avec le SSE, il est possible de faire des recherches par mot-clé sur une base de données chiffrée.

1.3.3 Laisser quelqu'un d'autre chiffrer

Dupliquer et partager du contenu numérique est devenu un jeu d'enfant avec la démocratisation de l'accès aux ortinateurs et à internet. Le piratage numérique a fait ses début à l'époque des cassettes audio, mais avec la disponibilité de lecteurs CD bon marché dans les années quatre-vingt-dix et l'arrivée des torrents pair à pair, les détenteurs de propriétés intellectuelles (en particulier les labels de musique) se sont mis à prendre note : comment une entreprise peut-elle rester dans profitable si un consommateur peut acheter un produit et des millions d'acheteurs potentiels décident de le télecharger à la place ? Aujourd'hui, le téléchargement de contenu sur internet nous paraît évident cependant ce n'a pas toujours été le cas, en raison de la crainte que le télechargement de musique saborde toute l'industrie musicale. Il a fallu convaincre les industriels de permettre le téléchargement de morceaux sur une plate-forme numérique sécurisée.

Parmi les solutions proposées, une nouvelle technologie était présentée pour la gestion des droits numériques (DRM), où un utilisateur pourraît *consommer* le média (par exemple écouter une chanson), mais sans réellement posséder le MP3. Cette technologie s'appelle la Cryptographie Boîte-Blanche, et a participé à ce pourquoi nous avons iTunes, Spotify et Netflix aujour'hui [CFV19].

L'objectif de la cryptographie boîte-blanche est de donner accès à un agent non-fiable à un algorithme de chiffrement opérant avec une clé secrète : la clé est cachée dans l'implémentation de l'algorithme. Avec ce système, l'agent ne doit pas être en mesure de retrouver la clé, même avec un accès en boîte blanche (i.e. l'agent a le contrôle total des rouages de l'algorithme). Idéalement, un morceau de musique serait chiffré de telle manière que la seule façon de l'écouter serait d'utiliser l'algorithme boîte-blanche : l'agent ne peut pas obtenir le fichier MP3 déchiffré.

La promesse d'une contre-mesure permettant de sauvegarder du contenu numérique contre le piratage a donné la confiance nécessaire aux parties prenantes pour permettre à leurs produits d'être vendus en ligne et téléchargés ou streamés. Cela n'a pas freiné les tentatives de piratage : il reste possible d'utiliser l'astuce datant de l'époque des cassettes de simplement enregistrer une chanson pendant qu'elle est jouée. Des attaques en tout genre sur les DRM sont apparues en réponse, il y a même une course à l'armement dans les communautés en ligne de piratage (par exemple la Scène Warez), pour retirer les protections anti-piratage des jeux vidéos [Wik23].

Les attaques qui nous intéressent en particulier sont les cryptanalyses de systèmes en boîte blanche. Nous présentons une telle attaque dans ce manuscrit.

1.4 Résumé des résultats

Nous avons désormais une idée générale de certains domaines de la cryptographie. Dans cette section, je présente les avancées que nous avons produites dans ces sujets.

1.4.1 De nouvelles bornes supérieurs sur la complexité des PSM

Dans un protocole PSM, il y a k parties $P_1, \ldots, P_k$ et un arbitre Ref. Soit $N \in \mathbb{N}$. Pour tout $i \in [k]$, chaque partie P_i détient une entrée secrète $x_i \in [N]$, et chacune envoie un message m_i à l'arbitre, de façon à ce que ce dernier puisse faire un calcul sur les messages qui donne $f(x_1, \ldots, x_k)$ où f est une fonction publique. Les parties sont supposées honnêtes. Nous nous intéressons en particulier à la complexité de communication du protocole, i.e. le nombre de bits que chaque partie doit envoyer afin que le calcul se déroule correctement. Nous regardons aussi la complexité d'aléas : combien de bits aléatoires les parties doivent partager.

Le modèle PSM a été présenté pour la première fois en 1994 par Feige, Kilian et Naor [FKN94] pour le cas à deux parties (bien que celui à k parties ait été évoqué dans l'annexe), puis a été étendu au cas multi-parties dans le travail d'Ishai et Kushilevitz [IK97]. Depuis, de nouvelles bornes inférieures et supérieures ont été découvertes : dans le cas à deux parties, [FKN94] présente un protocole avec une complexité de communication en $O(N)$, puis [BIKK14] montre qu'une complexité en $O(\sqrt{N})$ est possible. Dans [BKN18], un coût de $O(\mathsf{poly}(k) \cdot N^{k/2})$ a été trouvé pour tout $k \geq 2$, améliorant le coût en $O(\mathsf{poly}(k) \cdot N^{k-1})$ de [FKN94]. D'autres coûts encore plus bas sont trouvés pour $k \leq 5$. Cette publication a aussi proposé des protocoles avec une *complexité déséquilibrée* : certaines parties envoient plus d'information que d'autres.

S'agissant des bornes inférieures, [FKN94] montre une limite de $3k - O(1)$ bits, qui a été étendue en une borne inférieure de $3k - O(\log(k))$ par [AHMS20]. Pour $k = \omega(\log(n))$, cette borne inférieure de $\Omega(n)$ est améliorée en une borne de $\Omega(k^2 n / \log(kn))$ par [BR22].

À l'issue d'une collaboration avec Tianren Liu [AL21], nous montrons de nouvelles

bornes supérieures et présentons des protocoles PSM avec communication déséquilibrée pour deux parties. Notre résultat principal est un protocole PSM pour une infinité de k (tous les $k \leq 20$ et toutes les puissances premières) avec un coût de $O(\mathsf{poly}(k) \cdot N^{\frac{k-1}{2}})$.

Nous présentons un nouveau cadre dans lequel nous construisons ces PSM au Chapitre 3. Plus précisément, nous décrivons une manière de créer des *codages aléatoires*, les PSM en étant des instanciations particulières. En bref, un codage aléatoire $\hat{f} : \mathcal{X} \times \mathcal{R} \to \hat{\mathcal{Y}}$ d'une fonction $f : \begin{cases} \mathcal{X} \to \mathcal{Y} \\ x \mapsto f(x) \end{cases}$ est tel que pour $r \in \mathcal{R}$ une valeur aléatoire, $\hat{f}(x, r)$ contient suffisamment d'informations pour récupérer $f(x)$ et rien de plus concernant x. Les PSM en sont un cas particulier : en considérant x comme $x = (x_1, \ldots, x_k)$, les messages $m_1, \ldots, m_k$ reçus par l'arbitre constituent un codage aléatoire (où r est l'aléa partagé) de $f(x_1, \ldots, x_k)$.

L'idée derrière ce nouveau cadre vient de [BIKK14] : chaque entrée x_i est représentée par son vecteur identité $\mathbf{z}_i$, où la i-ème coordonnée de $\mathbf{z}i$ est 1 et les autres sont 0. Ensuite, nous *scindons* ce vecteur en deux : $\mathbf{z}_i = \mathbf{y}_{2i} \otimes \mathbf{y}_{2i+1}$. Nous donnons des exemples simples de constructions de PSM pour 3 et 5 parties en utilisant cette technique au Chapitre 3.1.2. Comme amuse-bouche, voici un exemple très simple de PSM à 2 parties en utilisant ce produit tensoriel des entrées.

Alice et Bob souhaitent s'engager dans un protocole PSM avec l'arbitre. Comme expliqué précédemment, nous scindons les entrées en deux, de façon à ce que les parties envoient deux entrées de taille $N^{1/2}$.

- Alice et Bob partageront l'aléa suivant : $\mathbf{a_1}, \mathbf{a_2}, \mathbf{b_1}, \mathbf{b_2} \xleftarrow{\$} \{0,1\}^{\sqrt{N}}$.
- L'entrée d'Alice $\mathbf{x}$ est telle que $\mathbf{x} = \mathbf{x_1} \otimes \mathbf{x_2}$, avec $\mathbf{x_1}, \mathbf{x_2} \in \{0,1\}^{\sqrt{N}}$.
- De façon similaire, l'entrée de Bob est $\mathbf{y} = \mathbf{y_1} \otimes \mathbf{y_2}$.

Maintenant, avant de voir comment le protocole fonctionne, Alice et Bob vont construire individuellement certains polynômes privés :

- Alice calcule $\mathbf{q_1}, \mathbf{q_2}$ de taille $O(\sqrt{N})$ tels que
  $\langle \mathbf{q_1}, \mathbf{z_1} \rangle = \langle \mathbf{p}, (\mathbf{x_1} + \mathbf{a_1}) \otimes (\mathbf{x_2} + \mathbf{a_2}) \otimes (\mathbf{z_1} \otimes \mathbf{b_2} + \mathbf{b_1} \otimes \mathbf{b_2}) \rangle$ et
  $\langle \mathbf{q_2}, \mathbf{z_2} \rangle = \langle \mathbf{p}, (\mathbf{x_1} + \mathbf{a_1}) \otimes (\mathbf{x_2} + \mathbf{a_2}) \otimes \mathbf{b_1} \otimes \mathbf{z_2} \rangle$.
- Bob calcule $\mathbf{h_1}, \mathbf{h_2}$ de taille $O(\sqrt{N})$ tels que
  $\langle \mathbf{h_1}, \mathbf{z_1} \rangle = \langle \mathbf{p}, (\mathbf{z_1} \otimes \mathbf{a_2} + \mathbf{a_1} \otimes \mathbf{a_2}) \otimes \mathbf{y_1} \otimes \mathbf{y_2} \rangle$ et
  $\langle \mathbf{h_2}, \mathbf{z_2} \rangle = \langle \mathbf{p}, \mathbf{a_1} \otimes \mathbf{z_2} \otimes \mathbf{y_1} \otimes \mathbf{y_2} \rangle$.

Le protocole fonctionne de la façon suivante :

- Alice envoie $(\mathbf{x_1} + \mathbf{a_1}, \mathbf{x_2} + \mathbf{a_2})$ à Ref.
- Bob envoie $(\mathbf{y_1} + \mathbf{b_1}, \mathbf{y_2} + \mathbf{b_2})$ à Ref.
- Alice et Bob s'engagent dans 4 instances d'un protocole PSM pour l'évaluation de polynômes comme dans la section précédente, de façon à ce que Ref puisse apprendre $\gamma = \langle \mathbf{q_1}, \mathbf{y_1} \rangle + \langle \mathbf{q_2}, \mathbf{y_2} \rangle + \langle \mathbf{h_1}, \mathbf{x_1} \rangle + \langle \mathbf{h_2}, \mathbf{x_2} \rangle$.
- Ref calcule $\langle \mathbf{F}, (\mathbf{x_1} + \mathbf{a_1}) \otimes (\mathbf{x_2} + \mathbf{a_2}) \otimes (\mathbf{y_1} + \mathbf{b_1}) \otimes (\mathbf{y_2} + \mathbf{b_2}) \rangle - \gamma$.

Le coût de la communication de ce protocole est de l'ordre de $O(\sqrt{N})$ pour Alice et Bob. Au Chapitre 3.4, nous présentons des protocoles PSM avec une communication *déséquilibrée* : Alice envoie $O(N^\eta)$ bits et Bob envoie $O(N^{1-\eta})$ bits, avec $\eta \in [0, 1]$.

1.4.2 Protocoles d'Oblivious RAM pour des objets de tailles variables

Le modèle d'Oblivious RAM (ORAM) sert à dissimuler le motif d'accès qu'un client produit lorsqu'il émet des requêtes à un serveur pour obtenir les données chiffrées qui y sont stockées. En supposant que le client a stocké N objets, chacun de taille B bits, nous souhaitons pouvoir effectuer des requêtes de lecture/écriture avec une complexité de communication inférieure à celle de l'ORAM trivial, où le client télécharge l'intégralité de la base de données. Les protocoles ORAM ont été présentés pour la première fois par Goldreich et Ostrovsky dans [GO96]. Ils y présentent le premier ORAM non trivial, appelé square-root ORAM (nommé d'après son coût de bande passante de $O(\sqrt{N} \cdot B)$). Dans le square-root ORAM, le serveur détient un tableau de taille $N + 2 \cdot \sqrt{N}$, composé (initialement) des N éléments que l'ORAM doit stocker, suivis de $\sqrt{N}$ objets *factices* (par exemple, des chiffrements de mots aléatoires) et $\sqrt{N}$ objets *protégés* (dans le refuge). Le refuge servira de stockage temporaire, utile lors des requêtes. L'idée générale derrière ce système est la suivante :

En premier lieu, le client permute de manière sécurisée (oblivious permutation dans la langue de Turing) les premiers $N + \sqrt{N}$ objets et garde une représentation de la permutation secrète π. Le client effectue ensuite plusieurs accès. Le j-ème *accès* à un objet (par exemple, le i-ème objet) consiste en :

- La lecture de tout le refuge de taille $\sqrt{N}$ pour trouver l'objet.
- Si trouvé : le client accède à l'objet factice suivant. Dans les premières $N + \sqrt{N}$ cellules, il est à l'index $\pi(N + j)$.
- Si non trouvé : le client accède à l'objet à son véritable emplacement à l'index $\pi(i)$.
- Le client met finalement à jour la valeur de l'objet i, le rechiffre et le stocke dans le refuge.

Après au plus $\sqrt{N}$ accès, le client met à jour les valeurs des objets réels dans les premières $N + \sqrt{N}$ cellules à partir de celles du refuge en utilisant un oblivious sort.

Cette méthode a ensuite été généralisée dans le paradigme ORAM *hiérarchique* dans le même article. Ici, il y a plusieurs niveaux, avec des tableaux doublant leur taille à chaque niveau. Les plus petits tableaux sont réorganisés plus fréquemment que les plus grands. Cela a un coût polylogarithmique. Depuis, des améliorations ont été apportées à la technique hiérarchique dans [AKL+18], conduisant au protocole "OptORAMa" qui atteint la borne inférieure de $\log(n)$ en complexité de communication. Cependant, cette méthode n'est pas réalisable en pratique. Les implémentations des protocoles ORAM sont généralement basées sur l'algorithme à base d'arbre, appelé tree-ORAM.

Le tree-ORAM le plus connu est Path-ORAM [SvS+13a], que nous présentons en détail au Chapitre 4.5.2. Un tree-ORAM encore plus simple, qui ne nécessite pas d'avoir un client avec un stockage de taille $\omega(\log(N))$, s'appelle "Simple ORAM" [CP13]. Ici, le stockage du serveur est structuré sous forme d'arbre binaire complet de profondeur $L = \lceil \log(N) \rceil$, où chaque nœud, appelé *panier*, peut stocker jusqu'à $\log(N)$ objets. Le client ne connaît pas la position exacte de chaque objet, mais connaît la feuille de l'arbre qui identifie le chemin où se trouve cet objet, allant de la racine à cette feuille. Ainsi, pour accéder à l'objet i-ème, le client commence par calculer la feuille l associée à cet objet (à partir d'une *fonction de position*), puis lit (et rechiffre) le contenu de chaque panier

présent sur le chemin de la racine à la feuille l. Lorsque l'objet recherché est rencontré, le client le télécharge, effectue une opération de lecture ou d'écriture, et sélectionne de façon uniformément aléatoire une nouvelle feuille à associer à cet objet. Cet objet est enfin écrit là où on trouve de la place, sur le sous-chemin à l'intersection de l'ancien chemin et du nouveau (ce dernier correspond à la feuille nouvellement sélectionnée).

Dans un travail réalisé avec mon directeur de thèse Brice Minaud [AM23] au Chapitre 4, nous avons étudié comment permettre aux ORAM d'accueillir des objets *pondérés*, i.e. de tailles variables (weighted-ORAM). Nous avons une transformation universelle réalisant cette fonctionnalité, permettant ainsi à tous les ORAM de fonctionner avec des objets pondérés. Néanmoins, elle nécessite un coût asymptotiquement plus élevé. Notre résultat principal est une transformation générique asymptotiquement équivalente à l'ORAM de départ, s'appliquant à la plupart des tree-ORAM utilisés en pratique. L'idée générale est de considérer le poids total des objets : chaque objet a une taille $\leq B$ bits, supposons que le serveur héberge un poids total d'au plus $N \cdot B$. Dans le cas standard (non pondéré), les paniers ne pouvaient stocker qu'un nombre borné Z d'objets. Maintenant, nous permettons aux paniers de stocker un poids total de $Z \cdot B$ bits. Nous pouvons placer un objet dans un panier tant que la limite de $Z \cdot B$ n'est pas atteinte. Le panier est considéré comme plein si le poids total dépasse $Z \cdot B$ (il sera toujours en dessous de $(Z+1) \cdot B$). Cette modification est naturelle et efficace. La difficulté principale est de montrer qu'elle conserve l'exactitude (le fait que l'ORAM se comporte sans échecs) : en effet, les preuves d'exactitude de tree-ORAM sont notoirement complexes, même sans des objets pondérés. Pour pallier ceci nous utilisons un nouvel argument de Schur-convexité qui correspond parfaitement au cadre.

Nous construisons également du chiffrement recherchable en utilisant ces ORAM pondérés.

1.4.3 Cryptanalyse structurelle de systèmes en boîte blanche basés sur des protocoles ARX

Étudions la façon dont un algorithme de chiffrement symétrique Enc, pour une clé secrète K, fonctionne lorsqu'un adversaire y a accès. L'adversaire fait une requête pour l'entrée x et obtient la sortie $y = \mathsf{Enc}_K(x)$. Le modèle de cryptographie en *boîte noire* consiste à se représenter un algorithme comme une fonction opaque, transformant les entrées en sorties sans rien expliquer sur le fonctionnement interne. On dit d'un adversaire essayant d'attaquer un système cryptographique en boîte noire qu'il a un *accès oracle* à ce système.

Évidemment, ce n'est pas ainsi que cela fonctionne dans le monde réel : certaines informations fuitent. Les calculs ne sont pas instantanés, donc le temps nécessaire pour produire une sortie peut révéler quelque chose sur la clé secrète [Koc96]. Un attaquant plus raffiné peut étudier les perturbations électromagnétiques autour de l'appareil effectuant les calculs, ou peut faire des mesures sur les lignes de courant pour étudier la consommation électrique utilisée pour calculer Enc_K [GMO01]. Dans ce cas, on dit que l'attaquant a des informations par un *canal auxiliaire*. Ce modèle est également appelé *boîte grise*.

Dans le modèle en boîte blanche, l'attaquant dispose d'une implémentation de Enc_K. Il lui est possible d'en étudier le fonctionnement interne et de la modifier. La cryptographie en boîte blanche est l'étude de ce modèle, c'est-à-dire des moyens de produire une implémentation de Enc_K de façon à ce que l'adversaire ne puisse pas obtenir d'informations sur K. La première implémentation en boîte blanche d'un cryptosystème (AES) a été présentée dans [CEJv03], on parle de cadre *CEJO*. Ici, un chiffrement est décomposé en une série de tables de correspondance composées. Elles sont chacune

obfusquées en composant leurs entrées et sorties avec des permutations aléatoires. La permutation de sortie d'une table est annulée par celle à l'entrée de la table suivante (en prenant la permutation inverse). L'idée derrière ce modèle est la suivante : un réseau de substitution-permutation (SPN) comme AES est représenté par une composition de couches non linéaires : $\mathsf{Enc}_K = E_1 \circ \ldots \circ E_n$. Le système en boîte blanche est composé d'une suite de couches $(E'_i)_{i\in[n]} = (O_i \circ E_i \circ I_i)_{i\in[n]}$ où $O_i = I_{i+1}^{-1}$ pour tout $i \in [n-1]$. La paire (I_1, O_n) est appelée les *encodages externes*. Les autres encodages sont appelés *encodages internes*. Le chiffrement en boîte blanche est évalué par $Enc'_K = E'_1 \circ \ldots \circ E'_n = I_1 \circ \mathsf{Enc}_K \circ O_n$. Plusieurs implémentations de cette méthode ont été cryptanalaysées dans [RW19, MDFK15, ABMT18, AMR19].

Une nouvelle proposition pour les chiffrements en boîte blanche a été présentée à CRYPTO 2022 par Ranea et al. [RVP22], en utilisant des *implémentations implicites*. Ici, la fonction de tour F d'un chiffrement par bloc est représentée par une fonction P telle que $\forall x, y,\ P(x, y) = 0 \iff F(x) = y$. P doit être telle que pour tous les x, la fonction $y \mapsto P(x, y)$ soit affine. Ce nouveau cadre permet une plus grande souplesse dans le choix de la fonction de tour, ce qui fait de ce cryptosystème en boîte blanche le premier capable de traiter les chiffres ARX (Addition-Rotation-XOR), tels que Speck [BSS+13].

Avec Brice Minaud et Antoine Houssais, nous montrons au Chapitre 5 une attaque structurelle sur ce modèle : il peut être attaqué dans un contexte en boîte noire. Nous utilisons une technique similaire à celle de [MDFK18] lors de l'attaque des implémentations implicites des cryptosystèmes SPN : bien que la première couche soit désormais non-linéaire, alors qu'elle n'était qu'affine auparavant, il est possible de reconstruire cette couche avec un nombre calculatoirement faisable d'échantillons. Nous montrons également une attaque sur les implémentations de cryptosystèmes ARX. Nous suivons la même trame de décomposition du programme couche par couche, reconstruisant ainsi chaque encodage.

En substance, l'idée est la suivante : considérons un cryptosystème ARX simple qui prend en entrée $\mathbf{t} = (\mathbf{x}||\mathbf{y}) \in \{0,1\}^n$, où $\mathbf{x}, \mathbf{y} \in \{0,1\}^{n/2}$, et donne en sortie $(\mathbf{x}||\mathbf{x} \boxplus \mathbf{y})$ où $\boxplus$ est l'addition modulaire dans $\mathbb{Z}_{2^n}$ (la clé secrète apparaîtra dans les couches externes). Nous avons trois couches : I la couche non-linéaire de faible degré au niveau de l'entrée, M est l'addition modulaire telle que décrite plus haut et A est la couche affine au niveau de la sortie. L'attaquant fournit en entrée $\mathbf{a}$ et obtient en sortie $\mathbf{b} = A \circ M \circ I(\mathbf{a})$. Notons $\mathbf{t} = (\mathbf{x}||\mathbf{y})$ par $N(\mathbf{a})$ et $\mathbf{z} = \mathbf{x} \boxplus \mathbf{y}$. On remarque que, pour tout $i \in [n/2]$, $z_i = x_i + y_i + c_i$, où c_i est la retenue de l'addition qui donne z_{i-1} (et $c_{n/2} = 0$). En utilisant cette relation de récurrence, nous sommes en mesure de reconstruire l'intégralité de l'addition modulaire. Cela nous permet ensuite de décomposer la couche A. Puis, nous pouvons décomposer les couches restantes M et I en résolvant certaines équations polynomiales.

Chapter

2 Introduction

Computer Science ("Informatique" in French) is the science of information: different subdisciplines of Computer Science study how to create information, how to store it, how to transform it ... Cryptology is the study of secret information. We distinguish two subjects in cryptology: cryptography regards how we hide information, cryptanalysis deals with recovering hidden information.

In this thesis, we present results pertaining to both of these subjects. A common thread is that our work concerns itself with contemporary challenges, that exist because we live in the age of the Internet, remote accesses and delegated computing. This era is sometimes called the *Information Age*.

Before this age of information (which begins to hatch after WWII), cryptology was used to hide a message (encryption), designed to stay hidden until the intended recipient would undo the hiding process (decryption) and read the original message. The most common uses were to give orders during wartime, to plot among political allies or to assist in acts of espionage. The fact that messages indeed stay hidden was crucial: the United States joined the Entente in WWI in no small part thanks to the decryption by British cryptanalists of the *Zimmerman Telegram*, in which Germany asked Mexico to declare war on their Northern neighbours. Mary Stuart, Queen of Scots, was executed because of her involvement in a plot to assassinate Queen Elizabeth I, which was uncovered when her cipher was broken. Ethel and Julius Rosenberg were also executed after being found guilty of treason and espionage against the United States, because Julius Rosenberg gave the plans to nuclear weapons to the Soviet Union. Their responsiblity was uncovered because of the secret Venona program, where American cryptanalists broke the cipher used by the Soviets.

Robust cryptography was a matter of life and death.

Today, while wars, assassination plots and espionage rings are still around, the use of cryptography is much more widespread. The breadcrumb trail of data that we leave behind us as a result of everyday life in the twenty-first century is massive and leaks much information on us. Some of it is done intentionally when one makes a post on social media. But the vast majority of this dataspill happens behind the scenes: metadata leaks whenever we make any movement in the cyberspace.

Consider a referee standing in the middle of a football field, looking at the eastern half of the stadium. At some point, a football – kicked from the western half – enters his field of view and finishes its course after bouncing a few times on the grass. If this referee knows enough about Physics, he can – from two distinct positions the ball took at two separate points in time – compute the exact position from where that ball was kicked.

Any observer can do the same when looking at the side effects of an online action, and infer from this metadata some information, that may have been intended to remain hidden. One of the questions a cryptographer can ask themselves is "how can I kick a

football accross the field, without anyone tracing that back to me?"

2.1 Symmetric and asymmetric cryptography

Let us introduce two well-known characters from crypto lore: Alice, who wishes to send some critical information to her friend Bob, over an unsecured channel (i.e. there may be an eavesdropper). For a long time, the practice of cryptography was limited to *symmetric encryption*: Alice and Bob share a *secret key* K, Alice *encrypts* her message m: using that key, she turns m into an unintelligible *ciphertext* c. She sends c to Bob, who then *decrypts* c using K, and gets the original message m.

This process – also called *secret-key* or *private-key* encryption – requires Alice and Bob to have agreed on a shared key K beforehand. This is a limitation, especially if Alice and Bob are complete strangers from one another and need to establish first contact securely.

The solution to that problem was invented in the seventies: using *asymmetric cryptography*, Bob can publish his public key PK, which Alice uses to encrypt m. She then sends the ciphertext c to Bob, who decrypts it using his secret key SK.

Also called *public-key* encryption (PKE), this new kind of protocol opens the way for modern cryptography: Alice and Bob can use PKE, along with other tools, to do a digital "handshake" and agree on a symmetric key, Alice can sign her messages to prove that she is the true sender, she can add integrity to prevent anyone from tampering with it...

In this thesis, we study three different types of protocols related to both symmetric and asymmetric encryption, and make contributions to their respective fields. A common thread between these subdisciplines of cryptography is the goal of facilitating secure delegated computation.

2.2 Having someone else do the computing

The discovery of public-key cryptography goes hand in hand with the ushering in of the Age of Information. Before the widespread use of computers, linked to each other in a network, symmetric cryptography was enough for its military and diplomatic uses. It is logical that members of the same nation (e.g. two generals discussing strategies) share the same secret key. Considerations about preventing the leakage of that key to the enemy are taken, but they seemingly do not warrant the need for a completely novel framework. An anecdote that illustrates this is the fact that cryptologists from GCHQ – a British intelligence agency – invented an algorithm similar to the RSA cryptosystem (the most used public-key scheme today) five to ten years before the publication of the scheme. The idea was shelved for lack of military uses.

Now, we have access to good computing power and publically available cryptography. Alice can choose among dozens if not hundreds of cryptosystems to securely communicate with Bob. But Alice's needs have also evolved: she might want to do more than just chat with friends. She could have a lot of sensitive data but no place to store it. Or she might want to perform some computations on this data, but does not have the processing power needed for such a task. Thankfully, supply has adapted to her demand: she can outsource the storage and computation she needs. If Alice trusts no one, how can her data remain secure?

Letting an exterior entity handle computation on sensitive data can be done securely in a number of ways. Fully Homomorphic Encryption (FHE) is a general way of enabling

computation over encrypted data: anyone can compute an arbitrary function on a tuple of ciphertexts, which results in a new ciphertext which is an encryption of the output of that function on the original plaintexts. FHE requires high computational and spatial overhead, but is a great ready-to-use solution. Alice could also *obfuscate* a program using indistinguishability obfuscation (iO): roughly speaking, someone could run the obfuscated program and have the same input-output couples that the original program would produce, but without any knowledge on the inner workings of said program.

In this thesis, we will look at three different proposals that deal with sharing hidden information among various parties. We will first look at a special case of Multi-Party Computation (MPC) called *Private Simultaneous Messages* (PSM) where a referee computes a public function on the private inputs of the parties it interacts with. Then we will see how a person can securely store and access data on an untrusted server using *Oblivious RAM* (ORAM) protocols. Finally, we will study *White-Box Cryptography*, where a secret-key algorithm is made to be publically computed, and show an attack on such a protocol.

2.3 Flavors of delegating computation

In the course of my thesis, I have worked on various subjects within different areas of cryptology. I present in this manuscript results in three of these subjects, which were made into articles. In this section, I introduce and motivate these subjects. In the next section, I will give a summary of the contributions that were made.

2.3.1 Computing with others

Pooling data from different parties to aquire new knowledge is often a necessity. A typical example from our field is Yao's millionaire problem: Alice and Bob want to know who among them is the wealthiest, but they do not want to reveal what their net worth actually is [Yao82]. The subject matter might seem vain, but the existence of a protocol that solves this problem is in fact consequential. We can imagine another situation: a pharmaceutical company wants to collect data to help in developping a new drug that could help patients suffering from a particular disease. Hospitals have plenty of potentially relevant data on their patients (e.g. age, gender, weight, mortality rate ...), however giving *carte blanche* to a company to use patient data is a recipe for confidentiality disaster, and is justly illegal in most countries. The naive solution of simply anonymizing the data (i.e. removing the names in an otherwise unchanged database, sometimes called "pseudonymization") does not work [NS06, Swe00]. Enough bits of information can remain in such a database to personally identify persons present in it. We can turn to cryptography for salvation: here all of the information is hidden, not only names or places of birth, which are sufficient identifiers, but not necessary.

If we encrypt this data, how will we be able to perform computations on it? A way to do so is through secure Multi-Party Computation (MPC).

In an MPC protocol, several parties $P_1, \ldots, P_k$ each hold a secret input $x_1, \ldots, x_k$, and they wish to compute the output of some (public) function f, such that each party learns $f(x_1, \ldots, x_k)$ and no additional information on the other inputs. MPC covers a vast field of protocols, with many different security requirements. This great survey by Yehuda Lindell delves in more depth about MPC in general [Lin20]. In this thesis, we

will focus on a minimalist instance of MPC called Private Simultaneous Messages (PSM).

Here, in addition to the k parties, we add a *Referee* who will be the one actually doing the computation. The protocol is *non-interactive*: each party sends a single message to the referee, and the parties do not discuss among each other. They only share a *common random string* which they will use to hide their input from the referee. This protocol assumes that only the referee can be corrupted, the other parties must remain honest. The parties' inputs will be *information-theoretically* hidden (also called *perfect secrecy*): no matter how powerful an adversary's computing powers are, no information can be obtained on the inputs by studying the messages received by the referee.

2.3.2 Storing data elsewhere

The amount of data produced keeps growing, and doing computation on this data (e.g. to train a Machine Learning model) is valuable. Although storage capacity has grown as well, the space needed to store enough data to perform such tasks is out of reach for most private citizens or smaller companies. To outsource this storage in a secure manner, the naive solution is to encrypt the data and store the ciphertexts in the cloud. This method is as secure as the symmetric encryption algorithm used (i.e. as secure as it gets if one chooses wisely), which is good news if one is only concerned with the security of the data, no matter the cost. But this is as saying that the best way to ensure not getting in an accident in a car is to never start it: technically not wrong, but we are not going anywhere. If we encrypt a terabyte of data using AES and store that, we must download a terabyte of ciphertext even if we only want to read one byte of cleartext whose index is unknown in it. This becomes bad news when we are concerned with any kind of practical application.

To have our cake and eat it too, why could we not just encrypt "small" blocks of our data, store them remotely and only download the small blocks we require as they are needed? We can actually, but this solution raises new problems. First of all, we need to index the blocks we store, this comes at some cost, but we can deal with it in several ways. Secondly, the fact that we access some block *in itself* reveals information on the data we query and the way we use it. Think of a boy grabbing a cookie from the kitchen, and hiding it behind his back when his mother comes to check on him. Sure, she does not see what he is holding, but the fact that he is hiding something, in the kitchen, after she explicitely forbade him to eat another cookie, should enable her to make an educated guess regarding what is being hidden from her.

To keep anyone from guessing what we are doing with our data, Oblivious RAM (ORAM) algorithms were introduced in the 1990s: they hide the *access pattern* one makes when querying data. Initially, the ORAM model was concerned with actual RAM: if a CPU makes queries to (encrypted) values in its *random access memory*, an eavesdropper could analyze the access pattern and learn confidential information on the computation that is being performed. More recently, this framework has been applied in client-server communications that happen when computing in the cloud. However, there are still applications in the originally intended RAM model, or as a mix of both online and offline computation when considering such actions in a secure enclave on a remote server [Con22].

In this thesis, we give new constructions of ORAM protocols and use them to build new Searchable Symmetric Encryption (SSE) schemes. In SSE, one is able to perform keyword searches on an encrypted database.

2.3.3 Letting someone else encrypt

Duplicating and sharing digital content has become orders of magnitude easier with the ubiquitous presence of computers and internet access. Digital piracy began in the era of cassette tapes, but with the inexpensive access to CD readers in the nineties and the advent of peer-to-peer torrents, intellectual property holders (especially music labels) began to take notice: how can a company protect its bottom line if a customer can purchase one product and millions of would-be buyers can just opt to dowload it instead? Today, we find it obvious that downloading digital content on the internet is the way to go, however this was not always the case: because of the aforementionned fear that letting people download music means scuttling the business model, allowing pieces to be sold on a digital marketplace required some convincing.

Among the proposals was a new technology used for Digital Rights Management (DRM), where a user could *consume* the media (i.e. listen to a song), but not actually own the MP3 file. This technology is called White-Box Cryptography, and contributed to the reason why we have iTunes, Netflix and Spotify today [CFV19].

The goal of White-Box Cryptography is to allow an untrusted agent to have access to an encryption algorithm under some secret key: the key is hidden in the implementation of the algorithm. With this *obfuscated* cipher, the agent should not be able to reverse-engineer the key even with *white-box* access (i.e. it is possible to study the inner workings of the algorithm). Ideally, a music piece could be encrypted such that the only way for someone to listen to it is to use the white-box algorithm: the agent cannot obtain the fully decrypted MP3 file.

The promise of a counter-measure to safeguard digital content against piracy gave stakeholders some confidence in allowing their products to be sold online and downloaded or streamed. This has not halted piracy attempts: it remains entirely possible to use the cassette-tape era ruse of simply recording the song as it is being played. In fact, all sorts of attacks on DRMs surfaced as a response. There even is an arms race in online piracy communities (e.g. the Warez Scene) to remove anti-piracy protections from video-games [Wik23].

Attacks of a more compelling nature to us are cryptanalyses of white-box cryptosystems. We give one such attack in this work.

2.4 Summary of results

We now have a bird's-eye view of certain subfields of cryptography. In this section, I summarize the advances that we made in these fields.

2.4.1 New upper bounds on the complexity of PSM protocols

In a PSM protocol, there are k parties $P_1, \ldots, P_k$ and a referee Ref. Let $N \in \mathbb{N}$. For all $i \in [k]$, each party P_i holds a secret input $x_i \in [N]$, and each sends a message m_i to the referee, such that the referee can perform a computation on the messages that outputs $f(x_1, \ldots, x_k)$ where f is a public function. The parties are assumed to be honest. We are mainly interested in the communication complexity of the protocol, i.e. the number of bits each party has to send to have the computation performed correctly. We are also concerned with the randomness complexity: how many random bits the parties must share.

The PSM model was presented first in 1994 by Feige, Kilian and Naor [FKN94] for the 2-party case (although the k-party case was touched on in the appendix), then expanded to multiple parties in a work by Ishai and Kushilevitz [IK97]. Since then, new upper and lower bounds have been found: in the 2-party case, [FKN94] showed a protocol with communication complexity $O(N)$, then it was shown that one can achieve a communication complexity of $O(\sqrt{N})$ [BIKK14]. In [BKN18], a cost of $O(\mathsf{poly}(k) \cdot N^{k/2})$ was shown for all $k \geq 2$, improving on the $O(\mathsf{poly}(k) \cdot N^{k-1})$ cost from [FKN94], along with better results for $k \leq 5$. This work also proposed protocols with *unbalanced complexity*: some parties send more information than others.

Regarding lower bounds, [FKN94] showed a limit of $3k - O(1)$ bits, which was expanded into a $3k - O(\log(k))$ lower bound by [AHMS20]. For $k = \omega(\log(n))$, this $\Omega(n)$ lower bound is improved into a $\Omega(k^2 n / \log(kn))$ one by [BR22].

In a work done in collaboration with Tianren Liu [AL21], we show new upper bounds and introduce 2-party unbalanced communication PSM protocols. Our main result is a PSM protocol for infinitely many k (namely, all $k \leq 20$ and all prime powers) with a cost of $O(\mathsf{poly}(k) \cdot N^{\frac{k-1}{2}})$.

We give a new framework in which we can build PSM protocols in Chapter 3. In fact, we describe a way to build *randomized encodings*, which PSM can be seen as a special instantiation of. In short, a randomized encoding $\hat{f} : \mathcal{X} \times \mathcal{R} \to \hat{\mathcal{Y}}$ of a function $f : \begin{cases} \mathcal{X} \to \mathcal{Y} \\ x \mapsto f(x) \end{cases}$ is such that for $r \in \mathcal{R}$ some random value, $\hat{f}(x, r)$ has enough information to recover $f(x)$ and nothing else about x. PSM are a special case of randomized encodings: seing x as $x = (x_1 \ldots, x_k)$, the messages $\{m_1, \ldots, m_k\}$ received by the referee are a randomized encoding (with r being the shared randomness) of $f(x_1, \ldots, x_k)$.

The idea behind this new framework comes from [BIKK14]: we represent each input x_i with its *one-hot* encoding: x_i is seen as an identity vector $\mathbf{z}_i$, where the x_ith coordinate of $\mathbf{z}_i$ is 1 and the others are 0. We then *split* this vector in two: $\mathbf{z}_i = \mathbf{y}_{2i} \otimes \mathbf{y}_{2i+1}$. We give simple examples of PSM constructions for 3 and 5 parties using this technique in Chapter 3.1.2. To serve as an appetizer, here is a very simple 2-party PSM using this tensoring of inputs.

Alice and Bob wish to engage in a PSM protocol with the referee. As explained above, we cut the inputs in half, so that the parties send two inputs of size $N^{1/2}$. The function f to be evaluated is represented as a polynomial $\mathbf{p} \in \{0,1\}^N$.

- Alice and Bob will share the following randomness: $\mathbf{a_1}, \mathbf{a_2}, \mathbf{b_1}, \mathbf{b_2} \xleftarrow{\$} \{0,1\}^{\sqrt{N}}$.
- Alice's input $\mathbf{x}$ is s.t. $\mathbf{x} = \mathbf{x_1} \otimes \mathbf{x_2}$, with $\mathbf{x_1}, \mathbf{x_2} \in \{0,1\}^{\sqrt{N}}$.
- In the same way, Bob's input is $\mathbf{y} = \mathbf{y_1} \otimes \mathbf{y_2}$.

Now, before we see how the protocol works, Alice and Bob will construct privately some polynomials:

- Alice computes $\mathbf{q_1}, \mathbf{q_2}$ of size $O(\sqrt{N})$ such that
  $\langle \mathbf{q_1}, \mathbf{z_1} \rangle = \langle \mathbf{p}, (\mathbf{x_1} + \mathbf{a_1}) \otimes (\mathbf{x_2} + \mathbf{a_2}) \otimes (\mathbf{z_1} \otimes \mathbf{b_2} + \mathbf{b_1} \otimes \mathbf{b_2}) \rangle$ and
  $\langle \mathbf{q_2}, \mathbf{z_2} \rangle = \langle \mathbf{p}, (\mathbf{x_1} + \mathbf{a_1}) \otimes (\mathbf{x_2} + \mathbf{a_2}) \otimes \mathbf{b_1} \otimes \mathbf{z_2} \rangle$.
- Bob computes $\mathbf{h_1}, \mathbf{h_2}$ of size $O(\sqrt{N})$ such that

$\langle \mathbf{h_1}, \mathbf{z_1} \rangle = \langle \mathbf{p}, (\mathbf{z_1} \otimes \mathbf{a_2} + \mathbf{a_1} \otimes \mathbf{a_2}) \otimes \mathbf{y_1} \otimes \mathbf{y_2} \rangle$ and
$\langle \mathbf{h_2}, \mathbf{z_2} \rangle = \langle \mathbf{p}, \mathbf{a_1} \otimes \mathbf{z_2} \otimes \mathbf{y_1} \otimes \mathbf{y_2} \rangle$.

The protocol works as follows:

- Alice sends $(\mathbf{x_1} + \mathbf{a_1}, \mathbf{x_2} + \mathbf{a_2})$ to Ref.
- Bob sends $(\mathbf{y_1} + \mathbf{b_1}, \mathbf{y_2} + \mathbf{b_2})$ to Ref.
- Alice and Bob engage in 4 instances of a PSM protocol for polynomial evaluation as in the previous section, so that Ref can learn $\gamma = \langle \mathbf{q_1}, \mathbf{y_1} \rangle + \langle \mathbf{q_2}, \mathbf{y_2} \rangle + \langle \mathbf{h_1}, \mathbf{x_1} \rangle + \langle \mathbf{h_2}, \mathbf{x_2} \rangle$.
- Ref computes $\langle \mathbf{F}, (\mathbf{x_1} + \mathbf{a_1}) \otimes (\mathbf{x_2} + \mathbf{a_2}) \otimes (\mathbf{y_1} + \mathbf{b_1}) \otimes (\mathbf{y_2} + \mathbf{b_2}) \rangle - \gamma$.

The communication cost of that protocol is $O(\sqrt{N})$ for both Alice and Bob. In Chapter 3.4, we present PSM protocols with *unbalanced* communication: Alice sends $O(N^\eta)$ bits and Bob sends $O(N^{1-\eta})$ bits, with $\eta \in [0, 1]$.

2.4.2 Oblivious RAM protocols for objects of variable sizes

The Oblivious RAM framework is concerned with hiding the access pattern that a client makes when querying a server for the encrypted data it has stored there. If we assume that the client has stored N objects, each of size B bits, we want to be able to perform read/write queries with a communication complexity lower than that of the trivial ORAM, where the client simply downloads the entire database. ORAM protocols were first introduced by Goldreich and Ostrovsky in [GO96]. They present the first non-trivial ORAM which is now called the square-root ORAM (named after its bandwidth cost of $O(\sqrt{N} \cdot B)$). In the square-root ORAM, the server holds an array of size $N + 2 \cdot \sqrt{N}$, composed (Initially) of the N items the ORAM must store, followed by $\sqrt{N}$ *dummy* items (e.g. encryptions of random words) and $\sqrt{N}$ *sheltered* items. The shelter will serve as a temporary storage, useful during queries. The high-level idea behind this system is as follows:

First, the client obliviously permutes the first $N+\sqrt{N}$ items and keeps a representation of the secret permutation π. The client then makes several accesses. The jth *access* to an object (e.g. the ith object) consists of:

- Reading the entire shelter of size $\sqrt{N}$ to find the object.
- If found: the client accesses the next dummy object. In the first $N + \sqrt{N}$ memory cells, it is at index $\pi(N + j)$.
- If not found: the client accesses the object at its true location at index $\pi(i)$.
- The client eventually updates the value of the ith item, reencrypts it and stores it in the shelter.

After at most $\sqrt{N}$ accesses, the client updates the values of the real objects in the first $N + \sqrt{N}$ cells from the ones in the shelter using oblivious sorting.

This method was then generalized into the *hierarchical* ORAM paradigm in the same paper. Here, there are several levels, with arrays doubling in size at each one. The

smaller arrays are reshuffled more frequently than the larger ones. This yields a poly-logarithmic overhead. In fact, improvements were made on the hierarchical technique in [AKL+18], yielding the "OptORAMa" protocol that achieves the lower bound of $\log(n)$ in communication overhead. However this method is impractical. Implementations of ORAM protocols are usually of algorithm following the tree-based paradigm.

The most well known tree-ORAM is Path-ORAM [SvS+13a], which we present in detail in Chapter 4.5.2. An even simpler tree-ORAM, that removes Path-ORAM's need for a $\omega(\log(N))$ client storage is aptly named "Simple ORAM" [CP13]. Here, the server storage is structured as a complete binary tree of depth $L = \lceil \log(N) \rceil$, where each node, called *bucket*, can store up to $\log(N)$ items. The client does not know the exact position of each item, but knows the tree leaf that identifies the path from root to leaf on which that object is. Thus, to access the ith object, the client first compute the leaf l associated with that object (from a *position map*), then reads (and reencrypts) the content of each bucket on the path from the root node to the leaf l. When the desired object is at some point encountered, the client downloads it, performs a read or write operation, and selects uniformly at random a new leaf to be associated with it. That object is finally written wherever there is space for it, at the intersection of the old path and the new one (corresponding to the freshly selected leaf).

In a joint work with my advisor Brice Minaud [AM23] in Chapter 4, we explored ways to have ORAM schemes handle *weighted* objects (i.e. of variable sizes). We have a universal transformation achieving this, thus having all ORAMs work with weighted objects, however it requires an asymptotically higher bandwidth. Our main result is a generic transformation with no added asymptotic cost that works on most tree-ORAMs used in practice. The general idea is to consider the total weight of the objects: every object has a size $\leq B$ bits, we will assume that the server will host a total weight of at most $N \cdot B$. In the standard (non-weighted) case, buckets could only store a bounded number Z of items. Now, we allow the buckets to store a total weight of $Z \cdot B$ bits. We can put an object in a bucket as long as that bound of $Z \cdot B$ is not reached. The bucket is considered full if the total weight exceeds $Z \cdot B$ (it will always be less than $(Z+1) \cdot B$). This modification is natural and efficient. The main difficulty is to show that it preserves correctness: indeed, tree-ORAM correctness proofs are notoriously intricate, even without variable-size objects. To resolve that issue, we introduce a new Schur-convexity argument, which turns out to fit the setting perfectly.

We also build Searchable Symmetric Encryption using the aforementioned weighted-ORAMs.

2.4.3 Structural cryptanalysis of white-box schemes of ARX ciphers

Let us think about how a symmetric encryption algorithm Enc, for some secret key K, works when an attacker is given access to it. Here, the attacker makes a query for the input x and is given the output $y = \mathsf{Enc}_K(x)$. The *black-box* model of cryptography consists of looking at an algorithm as an opaque functionality, turning inputs into outputs without giving any explaination as to how. An adversary trying to attack a cryptosystem in a black-box way is said to have been given *oracle access* to that system.

Of course, that is not how it works in the real world: some information leaks. Computations are not done instantaneously, thus the time it takes for an output to be produced can reveal something on the secret key [Koc96]. A more elaborate attacker can study elec-

tromagnetic perturbations around the device performing the computing, or make measure on the power lines to study the electrical consumption used to compute Enc_K [GMO01]. Here, the attacker is said to be given *side-channel* information. We also see this model refered to as the *grey-box* model.

In the white-box model, the attacker has at its disposal an implementation of Enc_K. It is possible to study the inner workings of the program and to modify them. White-box cryptography is the study of cryptography in the white-box model, i.e. ways of producing an implementation of Enc_K such that the adversary cannot learn any information on K. The first white-box implementation of a cipher (AES) was introduced in [CEJv03], this is refered to as the *CEJO* framework. Here, a cipher is decomposed into a series of composed look-up tables. They are individually obfuscated by composing their inputs and outputs with random permutations. The permutation at the output of one table is cancelled by the one at the input of the next table (by taking the inverse permutation). Here is a description of the design at a high-level: a subsitution-permutation network such as AES is seen as a composition of non-linear layers: $\mathsf{Enc}_K = E_1 \circ \ldots \circ E_n$. The white-box cipher is made up of a sequence of layers $(E'_i)_{i\in[n]} = (O_i \circ E_i \circ I_i)_{i\in[n]}$ where $O_i = I_{i+1}^{-1}$ for all $i \in [n-1]$. The pair (I_1, O_n) is called the *external encodings*. The other encodings are called *internal* encodings. The white-box cipher is evaluated as $Enc'_K = E'_1 \circ \ldots \circ E'_n = I_1 \circ \mathsf{Enc}_K \circ O_n$. Several implementations of this method have been cryptanalyzed in [RW19, MDFK15, ABMT18, AMR19].

A new proposal for white-box ciphers was presented at CRYPTO 2022 by Ranea et al [RVP22], using *implicit implementations*. Here, the round function F of a block cipher is represented by a function P such that $\forall x, y,\ P(x, y) = 0 \iff F(x) = y$. P must be such that for all x, the function $y \mapsto P(x, y)$ is affine. This new framework allows a greater flexibility in the choice of round function, thus this white-box cryptosystem is the first one able to handle ARX (Addition-Rotation-XOR) ciphers, such as Speck [BSS+13].

With Brice Minaud and Anoine Houssais, we show in Chapter 5 a structural attack on this framework: it can be attacked in a black-box setting. We use a technique similar to that of [MDFK18] to attack implicit implementations of SPN ciphers: even though the first layer is now non-linear whereas it was only affine before, it is possible to reconstruct this layer with a computationally feasible number of samples. We also show an attack on implementation of ARX ciphers. We follow the same pattern of peeling off the program layer by layer, thus reconstructing every encoding.

Here is, roughly, how it works: let us consider a simple ARX cipher that takes as input $\mathbf{t} = (\mathbf{x}||\mathbf{y}) \in \{0,1\}^n$, where $\mathbf{x}, \mathbf{y} \in \{0,1\}^{n/2}$, and outputs $(\mathbf{x}||\mathbf{x} \boxplus \mathbf{y})$ where $\boxplus$ is the modular addition in $\mathbb{Z}_{2^n}$ (the secret key will appear in the external layers). At a high-level, we have three layers: I the low-degree non-linear layer at the input level, M the modular addition as described above and A the affine layer at the output level. The attacker thus feeds an input $\mathbf{a}$ and gets an output $\mathbf{b} = A \circ M \circ I(\mathbf{a})$. Let us write $\mathbf{t} = (\mathbf{x}||\mathbf{y})$ as $N(\mathbf{a})$ and $\mathbf{z} = \mathbf{x} \boxplus \mathbf{y}$. We can notice that, for all $i \in [n/2]$, $z_i = x_i + y_i + c_i$, where c_i is the carry of the addition that gives z_{i-1} (and $c_{n/2} = 0$). Using this recurrence relation, we are able to reconstruct the entire modular addition. This, in turn, allows us to peel off the A layer. We can then peel off the remaining M and I layers by solving some polynomial equations.

Chapter

3 New techniques for Multi-party PSM

3.1 Introduction

Private Simultaneous Messages (PSM) is a minimal model of secure multi-party computation. It was introduced by Feige, Kilian and Naor [FKN94], and was generalized to the multi-party setting by Ishai and Kushilevitz [IK97].

In a PSM protocol for evaluating a k-ary functionality $f : [N]^k \to \{0,1\}$, there are k parties. They all share a common random string. For all $i \in [k]$, the i-th party holds a private input x_i. There is additionally a special party, called the *referee*. The referee receives one message from each party and is able to compute $f(x_1, \ldots, x_k)$, and should learn no other information about $x_1, \ldots, x_k$.

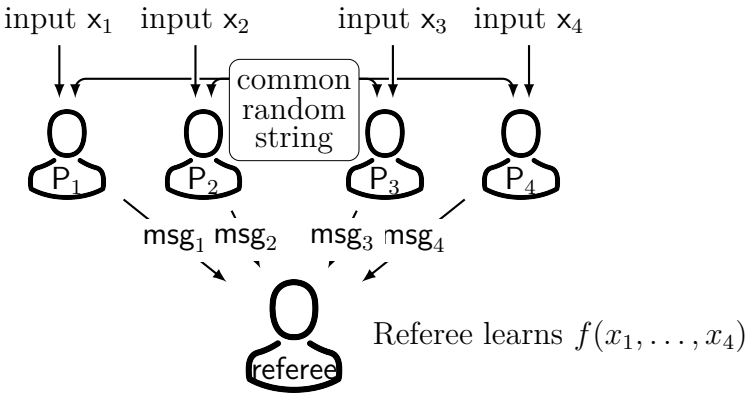

Figure 3.1: Illustration of a multi-party PSM protocol

PSM is studied as an information-theoretic primitive. The key complexity measure is the communication complexity. The common random string is crucial for the model as the common random string is the only mean to protect the privacy against an unbounded adversarial referee, when the k parties cannot communicate with each other.

In the PSM model, there are relatively efficient PSM protocols for computing non-deterministic branching programs [FKN94] and modular branching programs [IK97]. But for general functionalities, little is known regarding their communication complexity in the PSM model. Assuming every party holds an input in $[N]$, the best known lower bound of 2-party PSM is $3 \log N - O(\log \log N)$ [AHMS20]. In k-party PSM where each party holds a 1-bit input, Ball et al. showed an $\Omega(k^2/\log k)$ lower bound [BHI+20]. This lower bound was then improved to be $\Omega(k^2 n/log(kn))$ in [BR22]. Though the lower bounds are at most polynomial in the total input length, all known upper bounds are exponential, leaving an exponential gap between upper and lower bounds. For any functionality f :

Number of parties	[BIKK14]	[BKN18]	This work
2	$O(N^{1/2})$	$O(N^{1/2})$	$O(N^{1/2})$
3		$O(N)$	$O(N)$
4		$O(N^{5/3})$	$O(N^{3/2})$
5		$O(N^{7/3})$	$O(N^2)$
$k \geq 6$		$O(\text{poly}(k) \cdot N^{k/2})$	$2^{O(k)} \cdot N^{\frac{k-1}{2}}$ for infinitely many k including all $k \leq 20$

Table 3.1: The communication complexity of computing general $f : [N]^k \to \{0,1\}$ in multi-party PSM model

	Communication complexity of one party	Communication complexity of the other party
[FKN94]	$O(\log N)$	N
[BIKK14]	$O(N^{1/2})$	$O(N^{1/2})$
This work	$O(N^{\eta})$	$O(N^{1-\eta})$

Table 3.2: The unbalanced communication complexity of general $f : [N] \times [N] \to \{0,1\}$ in 2-party PSM model

$[N]^k \to \{0,1\}$, a "naïve" k-party PSM requires $O(N^{k-1})$ communication (the 2-party version was presented in [FKN94]). The first novel upper bound is $O(\sqrt{N})$ for 2-party PSM [BIKK14], and it was recently generalized to an $O_k(N^{k/2})$ upper bound for k-party PSM [BKN18]. In the same paper, Beimel, Kushilevitz and Nissim also further optimize the communication complexity for small $k = 3, 4, 5$. In particular, they obtain an $O(N)$ upper bound for 3-party PSM. For $k = 4$ or 5, they improve the protocol by letting parties jointly emulate their 3-party PSM. Their results are summarized in Table 3.1.

3.1.1 Our Contributions

In this chapter, we present two classes of results: We present new k-party PSM protocols that improve the communication complexity for infinitely many k. We introduce the notion of *unbalanced* 2-party PSM protocols, which allows a flexible repartition of the communication complexity among the two parties, and we present such protocols.

k-party PSM protocols. We present a framework for constructing multi-party PSM. The new framework improves the communication complexity upper bounds for infinitely many k. To compute any k-ary functionality $f : [N]^k \to \{0,1\}$,

- For all $k \leq 20$, our framework yields a k-party PSM protocol of communication complexity $O(N^{\frac{k-1}{2}})$.
- For all k such that $k+1$ is a prime or a prime power, our framework yields a k-party PSM protocol of communication complexity $O_k(N^{\frac{k-1}{2}})$.
- For all k, we *conjecture* that our framework will yield a k-party PSM protocol of communication complexity $O_k(N^{\frac{k-1}{2}})$.

2-party unbalanced PSM protocols. We also present a framework for constructing 2-party PSM protocols with unbalanced communication complexity. The new framework allows us to reduce the message length of one party at the cost of increasing the communication of the other party. We offer an almost smooth trade-off between the communication complexity of the two parties. To compute any functionality $f : [N] \times [N] \to \{0, 1\}$,

- For every rational $\eta \in (0, 1)$ whose denominator is no more than 20, our framework yields a 2-party PSM protocol, where one party sends $O(N^\eta)$ bits and the other sends $O(N^{1-\eta})$ bits.
- For every rational $\eta \in (0, 1)$, we *conjecture* that our framework will yield a 2-party PSM protocol, where one party sends $O_\eta(N^\eta)$ bits and the other sends $O_\eta(N^{1-\eta})$ bits.

To some extent, such a trade-off was known in the literature when $\eta = 0$. The first 2-party PSM protocol is of communication complexity $O(N)$ but is strongly unbalanced: one of the two parties only sends $O(\log N)$ bits [FKN94].

3.1.2 Proof Overview

This section presents the main ideas behind our new multi-party PSM protocols. We start with a warm-up example of a 3-party PSM, which is originally constructed by [BKN18]. We present it in a way that matches the framework we will later introduce. Then we present a new 5-party PSM to demonstrate the power of our framework. The 5-party PSM example relies on new technique such as "hard terms cancelling". It can be easily generalized into a framework for constructing k-party PSM protocols for any odd k. But we do not formally present this framework in this chapter.

Instead, in section 3.3, we develop a modified framework that supports odd as well as even values of k. The modified framework evenly divides every party's input into two halves, this idea was first introduced in [BIKK14]. When we formally present the modified framework in Section 3.3.1, we use a 4-party PSM as an example.

In section 3.4, we develop another framework for constructing unbalanced 2-party PSM protocols. Most terminologies and techniques are shared between the framework for k-party and the framework for unbalanced 2-party. Informally, the unbalanced 2-party PSM framework is the "tensor product" of two copies of the k-party framework. When we present the new framework in Section 3.4.1, we use as an exmaple a 2-party PSM with unbalanced communication $O(N^{1/3}), O(N^{2/3})$.

Background: 3-Party PSM [BKN18]. In this 3-party PSM protocol, three parties hold $x_1, x_2, x_3 \in [N]$ respectively. The protocol takes $O(N)$ communication and allows the referee to learn $f(x_1, x_2, x_3)$.

Fix a finite field $\mathbb{F}$. Let the i-th party locally computes a unit vector $\mathbf{x}_i \in \mathbb{F}^N$. That is, all entries in $\mathbf{x}_i$ are zero except for $\mathbf{x}_i[x_i] = 1$. Let $\mathbf{F}$ be the truth table of f represented as an $N \times N \times N$ array, we have $f(x_1, x_2, x_3) = \langle \mathbf{F}, \mathbf{x}_1 \otimes \mathbf{x}_2 \otimes \mathbf{x}_3 \rangle$, where $\otimes$ denotes the tensor product and $\langle \cdot, \cdot \rangle$ denotes the inner product.

Therefore, it is sufficient to construct a 3-party PSM protocol, where the i-th party has input $\mathbf{x}_i \in \mathbb{F}^N$ (not necessarily being an unit vector) and the referee learns $\langle \mathbf{F}, \mathbf{x}_1 \otimes \mathbf{x}_2 \otimes \mathbf{x}_3 \rangle$ for some public $\mathbf{F} \in \mathbb{F}^{N \times N \times N}$.

We start by letting the i-th party sample random $\mathbf{r}_i \in \mathbb{F}^N$ from the common random string and send the one-time padded $\bar{\mathbf{x}}_i := \mathbf{x}_i + \mathbf{r}_i$ to the referee. Then the referee can compute $\langle \mathbf{F}, \bar{\mathbf{x}}_1 \otimes \bar{\mathbf{x}}_2 \otimes \bar{\mathbf{x}}_3 \rangle$. We call this term a "*masked term*", because it is computed from the masked inputs $\bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2, \bar{\mathbf{x}}_3$. This masked term can be decomposed as the sum of several "*pure terms*"

$$\begin{aligned}\langle \mathbf{F}, \bar{\mathbf{x}}_1 \otimes \bar{\mathbf{x}}_2 \otimes \bar{\mathbf{x}}_3 \rangle = \langle \mathbf{F}, \mathbf{x}_1 \otimes \mathbf{x}_2 \otimes \mathbf{x}_3 \rangle + \\ \langle \mathbf{F}, \mathbf{x}_1 \otimes \mathbf{x}_2 \otimes \mathbf{r}_3 \rangle + \langle \mathbf{F}, \mathbf{x}_1 \otimes \mathbf{r}_2 \otimes \mathbf{x}_3 \rangle + \langle \mathbf{F}, \mathbf{r}_1 \otimes \mathbf{x}_2 \otimes \mathbf{x}_3 \rangle + \\ \langle \mathbf{F}, \mathbf{x}_1 \otimes \mathbf{r}_2 \otimes \mathbf{r}_3 \rangle + \langle \mathbf{F}, \mathbf{r}_1 \otimes \mathbf{x}_2 \otimes \mathbf{r}_3 \rangle + \langle \mathbf{F}, \mathbf{r}_1 \otimes \mathbf{r}_2 \otimes \mathbf{x}_3 \rangle + \\ \langle \mathbf{F}, \mathbf{r}_1 \otimes \mathbf{r}_2 \otimes \mathbf{r}_3 \rangle.\end{aligned} \tag{3.1}$$

We classify the pure terms into two categories:

Target Term The term $\langle \mathbf{F}, \mathbf{x}_1 \otimes \mathbf{x}_2 \otimes \mathbf{x}_3 \rangle$. It is the term that the referee should learn as a consequence of the 3-party PSM protocol.

Easy Term All the other terms fall into this category. As the name suggested, there also exist "hard terms", which will be introduced in the next example of 5-party PSM.

The easy terms are called "easy" because each of them can be securely revealed to the referee using only $O(N)$ communication. More formally, let the parties additionally sample random $r_1, \ldots, r_7 \in \mathbb{F}$ from their common random string such that $r_1 + \cdots + r_7 = 0$. There exist sub-protocols revealing each of

$$\begin{gathered}r_1 + \langle \mathbf{F}, \mathbf{x}_1 \otimes \mathbf{x}_2 \otimes \mathbf{r}_3 \rangle,\ r_2 + \langle \mathbf{F}, \mathbf{x}_1 \otimes \mathbf{r}_2 \otimes \mathbf{x}_3 \rangle,\ r_3 + \langle \mathbf{F}, \mathbf{r}_1 \otimes \mathbf{x}_2 \otimes \mathbf{x}_3 \rangle, \\ r_4 + \langle \mathbf{F}, \mathbf{x}_1 \otimes \mathbf{r}_2 \otimes \mathbf{r}_3 \rangle,\ r_5 + \langle \mathbf{F}, \mathbf{r}_1 \otimes \mathbf{x}_2 \otimes \mathbf{r}_3 \rangle,\ r_6 + \langle \mathbf{F}, \mathbf{r}_1 \otimes \mathbf{r}_2 \otimes \mathbf{x}_3 \rangle, \\ r_7 + \langle \mathbf{F}, \mathbf{r}_1 \otimes \mathbf{r}_2 \otimes \mathbf{r}_3 \rangle\end{gathered} \tag{3.2}$$

to the referee without leaking any other information, taking at most $O(N)$ communication.

Assume that such sub-protocols exist, we can easily finish the 3-party PSM: The i-th party sends $\bar{\mathbf{x}}_i := \mathbf{x}_i + \mathbf{r}_i$, they use the the aforementioned sub-protocols to reveal (3.2). The correctness follows almost directly from (3.1).

The only missing piece is to contruct sub-protocols for computing the terms in (3.2). Let us discuss them individually:

- For the last term $r_7 + \langle \mathbf{F}, \mathbf{r}_1 \otimes \mathbf{r}_2 \otimes \mathbf{r}_3 \rangle$, any party (e.g. the first party) can compute it and send it to the referee.
- For the term $r_4 + \langle \mathbf{F}, \mathbf{x}_1 \otimes \mathbf{r}_2 \otimes \mathbf{r}_3 \rangle$, the first party computes it and sends it to the referee. Similarly for $r_5 + \langle \mathbf{F}, \mathbf{r}_1 \otimes \mathbf{x}_2 \otimes \mathbf{r}_3 \rangle$ and $r_6 + \langle \mathbf{F}, \mathbf{r}_1 \otimes \mathbf{r}_2 \otimes \mathbf{x}_3 \rangle$.
- For the term $r_1 + \langle \mathbf{F}, \mathbf{x}_1 \otimes \mathbf{x}_2 \otimes \mathbf{r}_3 \rangle$, both first and second party need to participate. Since the first party knows $\mathbf{F}, \mathbf{x}_1, \mathbf{r}_3$, it can locally compute a vector $\mathbf{g} \in \mathbb{F}^N$ such that
$$r_1 + \langle \mathbf{F}, \mathbf{x}_1 \otimes \mathbf{x}_2 \otimes \mathbf{r}_3 \rangle = r_1 + \langle \mathbf{g}, \mathbf{x}_2 \rangle.$$
Then they can jointly reveal it to the referee using the PSM for inner product (more details are provided in Section A.2.1). Similarly for $r_2 + \langle \mathbf{F}, \mathbf{x}_1 \otimes \mathbf{r}_2 \otimes \mathbf{x}_3 \rangle$ and $r_3 + \langle \mathbf{F}, \mathbf{r}_1 \otimes \mathbf{x}_2 \otimes \mathbf{x}_3 \rangle$.

Example: 5-Party PSM. We will sketch a 5-party PSM protocol for any $f : [N]^5 \to \{0,1\}$ with communication complexity $O(N^2)$.

Let $\mathbb{F}$ be a finite field. Following the same observation we made in the 3-party PSM example, it is sufficient to construct a PSM protocol for any function of the form $(\mathbf{x}_1, \ldots, \mathbf{x}_5) \mapsto \langle \mathbf{F}, \mathbf{x}_1 \otimes \cdots \otimes \mathbf{x}_5 \rangle$, where $\otimes$ denotes the *tensor product*, the i-th party having input $\mathbf{x}_i \in \mathbb{F}^N$, $\mathbf{F}$ is public and fixed being the truth table of f.

For each $\Omega \subseteq \{1,2,3,4,5\}$, parties sample a dimension-$|\Omega|$ tensor $\mathbf{R}_\Omega \in \mathbb{F}^{N^{|\Omega|}}$ from the common random string. Define $\bar{\mathbf{X}}_\Omega := \mathbf{R}_\Omega + \bigotimes_{i \in \Omega} \mathbf{x}_i$. For example, $\bar{\mathbf{X}}_{\{2\}} := \mathbf{R}_{\{2\}} + \mathbf{x}_2$ and $\bar{\mathbf{X}}_{\{3,4\}} := \mathbf{R}_{\{3,4\}} + \mathbf{x}_3 \otimes \mathbf{x}_4$. Since the communication budget is $O(N^2)$, they can perform a PSM sub-protocol so that the referee learns $\bar{\mathbf{X}}_\Omega$ for all Ω such that $|\Omega| \leq 2$.

Learning those tensors allows the referee to compute many terms, including $\langle \mathbf{F}, \bar{\mathbf{X}}_{\{1,2\}} \otimes \bar{\mathbf{X}}_{\{3,4\}} \otimes \bar{\mathbf{X}}_{\{5\}} \rangle$. This term can be decomposed into the sum of the following 8 terms:

$$\begin{aligned}
&\langle \mathbf{F}, \bar{\mathbf{X}}_{\{1,2\}} \otimes \bar{\mathbf{X}}_{\{3,4\}} \otimes \bar{\mathbf{X}}_{\{5\}} \rangle \\
&= \langle \mathbf{F}, \mathbf{x}_1 \otimes \mathbf{x}_2 \otimes \mathbf{x}_3 \otimes \mathbf{x}_4 \otimes \mathbf{x}_5 \rangle + \langle \mathbf{F}, \mathbf{x}_1 \otimes \mathbf{x}_2 \otimes \mathbf{x}_3 \otimes \mathbf{x}_4 \otimes \mathbf{R}_{\{5\}} \rangle \\
&\quad + \langle \mathbf{F}, \mathbf{x}_1 \otimes \mathbf{x}_2 \otimes \mathbf{R}_{\{3,4\}} \otimes \mathbf{x}_5 \rangle + \langle \mathbf{F}, \mathbf{x}_1 \otimes \mathbf{x}_2 \otimes \mathbf{R}_{\{3,4\}} \otimes \mathbf{R}_{\{5\}} \rangle \\
&\quad + \langle \mathbf{F}, \mathbf{R}_{\{1,2\}} \otimes \mathbf{x}_3 \otimes \mathbf{x}_4 \otimes \mathbf{x}_5 \rangle + \langle \mathbf{F}, \mathbf{R}_{\{1,2\}} \otimes \mathbf{x}_3 \otimes \mathbf{x}_4 \otimes \mathbf{R}_{\{5\}} \rangle \\
&\quad + \langle \mathbf{F}, \mathbf{R}_{\{1,2\}} \otimes \mathbf{R}_{\{3,4\}} \otimes \mathbf{x}_5 \rangle + \langle \mathbf{F}, \mathbf{R}_{\{1,2\}} \otimes \mathbf{R}_{\{3,4\}} \otimes \mathbf{R}_{\{5\}} \rangle.
\end{aligned} \tag{3.3}$$

Any term that is formed in the same way as the left-hand side of (3.3), i.e. $\langle \mathbf{F}, \bar{\mathbf{X}}_{S_1} \otimes \cdots \otimes \bar{\mathbf{X}}_{S_t} \rangle$ for some $S_1 + \cdots + S_t = \{1,2,3,4,5\}$, is called a *masked term*. It can be computed by the referee if $|S_i| \leq 2$ for all i.

Any term that is formed in the same way as the right-hand side of (3.3), i.e. $\langle \mathbf{F}, \mathbf{R}_{S_1} \otimes \cdots \otimes \mathbf{R}_{S_t} \otimes \mathbf{x}_{i_1} \otimes \cdots \otimes \mathbf{x}_{i_w} \rangle$ for some $S_1 + \cdots + S_t + \{i_1, \ldots, i_w\} = \{1,2,3,4,5\}$, is called a *pure term*. As hinted by equation (3.3), every masked term is equal to the sum of 2^t pure terms.

The pure terms fall naturally into three categories. In particular, we introduce a new category called *hard terms*.

Target term The term $\langle \mathbf{F}, \mathbf{x}_1 \otimes \mathbf{x}_2 \otimes \mathbf{x}_3 \otimes \mathbf{x}_4 \otimes \mathbf{x}_5 \rangle$ is called the target term.

Easy term A pure term $\langle \mathbf{F}, \mathbf{R}_{S_1} \otimes \cdots \otimes \mathbf{R}_{S_t} \otimes \mathbf{x}_{i_1} \otimes \cdots \otimes \mathbf{x}_{i_w} \rangle$ is easy if $w \leq 3$. Every easy term can be computed using a PSM protocol with communication complexity $O(N^2)$. For example, $\langle \mathbf{F}, \mathbf{R}_{\{1,2\}} \otimes \mathbf{x}_3 \otimes \mathbf{x}_4 \otimes \mathbf{x}_5 \rangle$ is an easy term. The 5^{th} party, based on its view, can compute a tensor $\mathbf{G} \in \mathbb{F}^{N^2}$ such that $\langle \mathbf{F}, \mathbf{R}_{\{1,2\}} \otimes \mathbf{x}_3 \otimes \mathbf{x}_4 \otimes \mathbf{x}_5 \rangle = \langle \mathbf{G}, \mathbf{x}_3 \otimes \mathbf{x}_4 \rangle$. And $\langle \mathbf{G}, \mathbf{x}_3 \otimes \mathbf{x}_4 \rangle$ can be computed using a PSM protocol (Section A.2.1) with communication complexity $O(N^2)$.

Hard term Any pure term that is neither the target term nor an easy term.

Let us ignore the easy terms for now. Then equation (3.3) can be rewritten as

$$\begin{aligned}
&\langle \mathbf{F}, \bar{\mathbf{X}}_{\{1,2\}} \otimes \bar{\mathbf{X}}_{\{3,4\}} \otimes \bar{\mathbf{X}}_{\{5\}} \rangle \\
&= \langle \mathbf{F}, \mathbf{x}_1 \otimes \cdots \otimes \mathbf{x}_5 \rangle + \langle \mathbf{F}, \mathbf{x}_1 \otimes \mathbf{x}_2 \otimes \mathbf{x}_3 \otimes \mathbf{x}_4 \otimes \mathbf{R}_{\{5\}} \rangle + \text{easy terms}.
\end{aligned}$$

There is only one hard term left. We would like to cancel out the hard term by combining a few masked terms. Let us consider the following masked terms: $\langle \mathbf{F}, \bar{\mathbf{X}}_{\{1,2\}} \otimes \bar{\mathbf{X}}_{\{3\}} \otimes \bar{\mathbf{X}}_{\{4,5\}} \rangle$, $\langle \mathbf{F}, \bar{\mathbf{X}}_{\{1,2\}} \otimes \bar{\mathbf{X}}_{\{3,5\}} \otimes \bar{\mathbf{X}}_{\{4\}} \rangle$ and $\langle \mathbf{F}, \bar{\mathbf{X}}_{\{1,2\}} \otimes \bar{\mathbf{X}}_{\{3\}} \otimes \bar{\mathbf{X}}_{\{4\}} \otimes \bar{\mathbf{X}}_{\{5\}} \rangle$.

$$\begin{aligned}
&\langle \mathbf{F}, \bar{\mathbf{X}}_{\{1,2\}} \otimes \bar{\mathbf{X}}_{\{3\}} \otimes \bar{\mathbf{X}}_{\{4,5\}} \rangle \\
&\quad = \langle \mathbf{F}, \mathbf{x}_1 \otimes \cdots \otimes \mathbf{x}_5 \rangle + \langle \mathbf{F}, \mathbf{x}_1 \otimes \mathbf{x}_2 \otimes \mathbf{R}_{\{3\}} \otimes \mathbf{x}_4 \otimes \mathbf{x}_5 \rangle + \text{easy terms}, \\
&\langle \mathbf{F}, \bar{\mathbf{X}}_{\{1,2\}} \otimes \bar{\mathbf{X}}_{\{3,5\}} \otimes \bar{\mathbf{X}}_{\{4\}} \rangle \\
&\quad = \langle \mathbf{F}, \mathbf{x}_1 \otimes \cdots \otimes \mathbf{x}_5 \rangle + \langle \mathbf{F}, \mathbf{x}_1 \otimes \mathbf{x}_2 \otimes \mathbf{x}_3 \otimes \mathbf{R}_{\{4\}} \otimes \mathbf{x}_5 \rangle + \text{easy terms}, \\
&\langle \mathbf{F}, \bar{\mathbf{X}}_{\{1,2\}} \otimes \bar{\mathbf{X}}_{\{3\}} \otimes \bar{\mathbf{X}}_{\{4\}} \otimes \bar{\mathbf{X}}_{\{5\}} \rangle \\
&\quad = \langle \mathbf{F}, \mathbf{x}_1 \otimes \cdots \otimes \mathbf{x}_5 \rangle + \langle \mathbf{F}, \mathbf{x}_1 \otimes \mathbf{x}_2 \otimes \mathbf{R}_{\{3\}} \otimes \mathbf{x}_4 \otimes \mathbf{x}_5 \rangle \\
&\qquad + \langle \mathbf{F}, \mathbf{x}_1 \otimes \mathbf{x}_2 \otimes \mathbf{x}_3 \otimes \mathbf{R}_{\{4\}} \otimes \mathbf{x}_5 \rangle + \langle \mathbf{F}, \mathbf{x}_1 \otimes \mathbf{x}_2 \otimes \mathbf{x}_3 \otimes \mathbf{x}_4 \otimes \mathbf{R}_{\{5\}} \rangle \\
&\qquad + \text{easy terms}.
\end{aligned}$$

By carefully combining these masked tensors, we have

$$\begin{aligned}
&\langle \mathbf{F}, \bar{\mathbf{X}}_{\{1,2\}} \otimes \bar{\mathbf{X}}_{\{3,4\}} \otimes \bar{\mathbf{X}}_{\{5\}} \rangle + \langle \mathbf{F}, \bar{\mathbf{X}}_{\{1,2\}} \otimes \bar{\mathbf{X}}_{\{3\}} \otimes \bar{\mathbf{X}}_{\{4,5\}} \rangle \\
&+ \langle \mathbf{F}, \bar{\mathbf{X}}_{\{1,2\}} \otimes \bar{\mathbf{X}}_{\{3,5\}} \otimes \bar{\mathbf{X}}_{\{4\}} \rangle - \langle \mathbf{F}, \bar{\mathbf{X}}_{\{1,2\}} \otimes \bar{\mathbf{X}}_{\{3\}} \otimes \bar{\mathbf{X}}_{\{4\}} \otimes \bar{\mathbf{X}}_{\{5\}} \rangle \\
&\qquad = 2 \cdot \langle \mathbf{F}, \mathbf{x}_1 \otimes \cdots \otimes \mathbf{x}_5 \rangle + \text{easy terms}.
\end{aligned} \tag{3.4}$$

Equation (3.4) shows us how to construct the desired PSM protocol. All of the masked tensors on the left-hand side of (3.4) can be computed by the referee. The parties perform a PSM sub-protocol so that the referee learns the sum of these easy terms. (The details are demonstrated in the last example of 3-party PSM, and are explained in Section 3.3.2.) Then from equation (3.4), the referee learns $2 \cdot \langle \mathbf{F}, \mathbf{x}_1 \otimes \cdots \otimes \mathbf{x}_5 \rangle$.

As long as $\mathbb{F}$ is a finite field in which $2 \neq 0$, the referee has learned the target term. The protocol takes a communication cost of $O(N^2)$ field elements.

3.1.3 Related Works

Besides [BIKK14, BKN18], our construction of PSM protocols is also inspired by the progress in Conditional Disclosure of Secrets (CDS). Until recently, CDS had a similar exponential gap between known upper and lower bounds. CDS can be viewed as a variant of PSM where the referee knows all but 1 bit of the input: Consider the 2-party case and let $[N]$ be the input domain for both parties. The upper bounds of $O(\sqrt{N})$ is conserved [BIKK14, GKW15]. A similar lower bound of $\Omega(\log N)$ is known [GKW15, AARV17]. Recently, Liu, Vaikuntanathan and Wee improved the CDS upper bound for arbitrary function to $2^{\tilde{O}\sqrt{\log N}}$ [LVW17]. In a slightly different setting, the amortized CDS upper bound per party is improved to $\Theta(1)$ [AARV17, AA18].

Gay, Kerenidis and Wee constructed 2-party CDS with smooth communication complexity trade-off between the two parties [GKW15]. In particular, for any $\eta \in [0, 1]$, they constructed a 2-party CDS protocol where one party sends $O(N^\eta)$ bits and the other sends $O(N^{1-\eta})$ bits.

In [ABF+19, CGO21], constructions of *ad hoc* PSM are presented. In this framework, there are k parties, but only a subset of them will perform the computation. This notion, expanded in [BIK17], was shown to imply obfuscation.

3.2 Preliminaries

Let $\mathbb{N} := \{0, 1, \ldots\}$ denote the set of all natural numbers, and let $[n] := \{1, \ldots, n\}$. In this chapter, $\mathbb{F}$ denotes a field, $\mathcal{R}$ denotes a finite commutative ring. For some prime power

p, let $\mathbb{F}_p$ denote the unique finite field of order p. A vector will be denoted by a bold face lowercase letter. For a vector $\mathbf{v}$, let $\mathbf{v}[i]$ denote its i-th entry.

3.2.1 Tensor

A *tensor* refers to the generalization of vectors and matrices which have multiple indices. Roughly speaking, a tensor is a multi-dimensional array. In the chapter, a tensor will be denoted by a bold face capital letter. A k-dimensional tensor $\mathbf{T} \in \mathbb{F}^{n_1 \times n_2 \times \ldots \times n_k}$ is essentially an array of size $n_1 \times n_2 \times \cdots \times n_k$. The entries in $\mathbf{T}$ are indexed by $(i_1, \ldots, i_k) \in [n_1] \times \cdots \times [n_k]$, and denoted by $\mathbf{T}[i_1, \ldots, i_k]$. A tensor can also be viewed as a representation of a multi-linear function: any k-linear function $f : \mathbb{F}^{n_1} \times \mathbb{F}^{n_2} \times \ldots \times \mathbb{F}^{n_k} \to \mathbb{F}$ can be uniquely determined by its coefficient tensor $\mathbf{F} \in \mathbb{F}^{n_1 \times \ldots \times n_k}$, such that

$$f(\mathbf{v}_1, \ldots, \mathbf{v}_k) = \sum_{i_1 \in [n_1], \cdots, i_k \in [n_k]} \mathbf{F}[i_1, \ldots, i_k] \cdot \mathbf{v}_1[i_1] \cdot \ldots \cdot \mathbf{v}_k[i_k]. \tag{3.5}$$

The inner product of two tensors $\mathbf{S}, \mathbf{T} \in \mathbb{F}^{n_1 \times n_2 \times \ldots \times n_k}$ is defined as

$$\langle \mathbf{S}, \mathbf{T} \rangle := \sum_{i_1 \in [n_1], \cdots, i_k \in [n_k]} \mathbf{S}[i_1, \ldots, i_k] \cdot \mathbf{T}[i_1, \ldots, i_k].$$

Given two tensors $\mathbf{S} \in \mathbb{F}^{n_1 \times \ldots \times n_k}$ and $\mathbf{T} \in \mathbb{F}^{m_1 \times \ldots \times m_\ell}$, we define $\mathbf{S} \otimes \mathbf{T}$, their tensor product. It is a tensor in $\mathbb{F}^{n_1 \times \ldots \times n_k \times m_1 \times \ldots \times m_\ell}$ such that

$$(\mathbf{S} \otimes \mathbf{T})[i_1, \ldots, i_k, j_1, \ldots, j_\ell] = \mathbf{S}[i_1, \ldots, i_k] \cdot \mathbf{T}[j_1, \ldots, j_\ell].$$

Using the notation of inner product and tensor product, Equation (3.5) can also be written as $f(\mathbf{v}_1, \ldots, \mathbf{v}_k) = \langle \mathbf{F}, \mathbf{v}_1 \otimes \ldots \otimes \mathbf{v}_k \rangle$.

3.2.2 Private Simultaneous Messages

Definition 1 (private simultaneous message)**.** A k-party functionality is a mapping $f : \mathcal{X}_1 \times \ldots \times \mathcal{X}_k \to \mathcal{Y}$, where $\mathcal{X}_1, \ldots, \mathcal{X}_k$ are its input spaces and $\mathcal{Y}$ is its output space.

A private simultaneous message (PSM) protocol for a functionality f consists of a randomness space $\mathcal{W}$ and a tuple of deterministic functions $(\mathsf{M}_1, \ldots, \mathsf{M}_k, \mathsf{R})$

$$\begin{aligned} \mathsf{M}_i &: \mathcal{X}_i \times \mathcal{W} \to \{0,1\}^{\mathsf{cc}_i}, \quad \text{for all } i \in [k], \\ \mathsf{R} &: \{0,1\}^{\mathsf{cc}_1} \times \ldots \{0,1\}^{\mathsf{cc}_k} \to \{0,1\}, \end{aligned}$$

where cc_i is the communication complexity of the i-th party, $\mathsf{cc} := \mathsf{cc}_1 + \ldots + \mathsf{cc}_k$ is the total communication complexity.

A perfectly secure PSM protocol for f satisfies the following properties:

(correctness.) For all input tuple $(x_1, \ldots, x_k) \in \mathcal{X}_1 \times \ldots \times \mathcal{X}_k$ and randomness $w \in \mathcal{W}$,

$$\mathsf{R}(\mathsf{M}_1(x_1, w), \ldots, \mathsf{M}_k(x_k, w)) = f(x_1, \ldots, x_k)$$

(privacy.) There exists a randomized simulator S, such that for any input $(x_1, \ldots, x_k) \in \mathcal{X}_1 \times \ldots \times \mathcal{X}_k$, the joint distribution of $\mathsf{M}_1(x_1, w), \ldots, \mathsf{M}_k(x_k, w)$ is the same as the distributions of $\mathsf{S}(f(x_1, \ldots, x_k))$, where the distributions are taken over $w \leftarrow \mathcal{W}$ and the coin tosses of S.

3.2.3 Randomized Encoding

Randomized encoding is a primitive closely related to PSM. The randomized encoding of a function f is a randomized function $\hat{f}$. The output $\hat{f}(x, w)$, where w denotes the randomness, contains sufficient information to recover $f(x)$ and no other information about x.

Definition 2 (randomized encoding)**.** A randomized encoding for a function $f : \mathcal{X} \to \mathcal{Y}$ consists of a randomized encoding function $\hat{f} : \mathcal{X} \times \mathcal{W} \to \hat{\mathcal{Y}}$ and a deterministic decoding function $\mathsf{R} : \hat{\mathcal{Y}} \to \mathcal{Y}$, where $\mathcal{W}$ denotes the randomness space and $\hat{\mathcal{Y}}$ denotes the coding space.

A perfectly secure randomized encoding satisfies the following properties:

(correctness.) For all $x \in \mathcal{X}$ and randomness $w \in \mathcal{W}$,

$$\mathsf{R}(\hat{f}(x, w)) = f(x)$$

(privacy.) There exists a randomized simulator S, such that for any input $x \in \mathcal{X}$, the distribution of $\hat{f}(x, w)$ is the same as the distributions of $\mathsf{S}(f(x))$, where the distributions are taken over $w \leftarrow \mathcal{W}$ and the coin tosses of S.

Follows directly from the definitions, $(\mathsf{M}_1, \ldots, \mathsf{M}_k, \mathsf{R})$ is a PSM protocol for f if and only if $(\hat{f}, \mathsf{R})$ is a randomized encoding for f, where $\hat{f}(x_1, \ldots, x_k, w) := (\mathsf{M}_1(x_1, w), \ldots, \mathsf{M}_k(x_k, w))$.

In other words, PSM is a special form of randomized encoding, where the input is divided into a few portions, and each bit of the encoding only depends on the randomness and one portion of the input.

3.3 New Multi-party PSM Protocols

In this section, we present one of our main results: for many k, every functionality $f : [N]^k \to \{0, 1\}$ admits a PSM protocol of communication complexity $O_k(N^{\frac{k-1}{2}})$.

Theorem 3.3.1. *Let $f : [N]^k \to \{0, 1\}$ be an arbitrary k-party functionality.*

- *There is a k-party PSM protocol for f with communication and randomness complexity $O(N^{\frac{k-1}{2}})$, if $k \leq 20$.*
- *There is a k-party PSM protocol for f with communication and randomness complexity $O_k(N^{\frac{k-1}{2}})$, if $k + 1$ is a prime or a prime power.*

In this section, we prove a stronger statement. Let $\mathbb{F}$ be a finite field, consider the following auxiliary k-party functionality Aux^k_N:

k-party functionality Aux^k_N
• The i-th party has input $\mathbf{x}_{2i-1}, \mathbf{x}_{2i} \in \mathbb{F}^{\sqrt{N}}$
• The output is $\langle \mathbf{F}, \mathbf{x}_1 \otimes \cdots \otimes \mathbf{x}_{2k} \rangle$, where $\mathbf{F}$ is public and fixed

As shown in the beginning of Section 3.3.1, a PSM protocol for Aux_N^k implies a PSM for $f : [N]^k \to \{0,1\}$ with the same communication complexity. The reduction consists of having $\mathbf{F}$ be the truth table of f.

We will present a framework of constructing k-party PSM for Aux_N^k, whose communication complexity is $O_k(N^{\frac{k-1}{2}})$. Roughly speaking, the framework reduces the problem to a system of linear equations. A solution of the system implies a PSM protocol with the desired communication complexity. Therefore, we should rule out the possibility that the induced system has no solution. We partially achieve such a goal. We solve the induced system for infinitely many k:

- For all $k \leq 20$, we checked that the induced system of linear equations is solvable. For small k we solve the system by hand, and for larger k we verified it with a computer program.
- For all k such that $k+1$ is a prime power, we prove that the system is solvable.

Backed by the above observations, we strongly believe the induced system is solvable for all k.

Conjecture 1. Let $f : [N]^k \to \{0,1\}$ be an arbitrary k-party functionality. There is a k-party PSM protocol for f with communication and randomness complexity $O_k(N^{\frac{k-1}{2}})$.

Organization Section 3.3.1 presents our framework for constructing multi-party PSM, introduces new notations, and gives a 4-party PSM as a concrete example. The following Sections 3.3.2, 3.3.3, 3.3.4 are independent. Section 3.3.2 provides more technical detail of the PSM protocols yielded by our framework. Section 3.3.3 shows how the framework works for small k, and Section 3.3.4 shows how the framework works for any integer k such that $k+1$ is a prime power.

3.3.1 A Framework for Multi-party PSM

As mentioned in the beginning of Section 3.3, the functionality $f : [N]^k \to \{0,1\}$ can be reduced to functionality Aux_N^k. The reduction works as follows: Let $x_1, \ldots, x_k$ be the input, the j-th party has input $x_j \in [N]$. We evenly divide x_j into $x'_{2j-1}, x'_{2j} \in [\sqrt{N}]$. For each $i \in [2k]$, let $\mathbf{x}_i := \mathbf{e}_{x'_i} \in \mathbb{F}^{\sqrt{N}}$ be the x'_i-th standard unit vector. We reduce f to Aux_N^k:

$$f(x_1, \ldots, x_{2k}) = \langle \mathbf{F}, \mathbf{x}_1 \otimes \ldots \otimes \mathbf{x}_{2k} \rangle$$

where $\mathbf{F}$ is the truth-table of f. For the remainder of the section, it is thus sufficient to construct a PSM protocol for Aux_N^k.

For each non-empty $\Omega \subseteq [2k]$, our protocol will sample a random dimension-$|\Omega|$ tensor $\mathbf{R}_\Omega \in \mathcal{R}^{(\sqrt{N})^{|\Omega|}}$ from the common random string[1]. Define $\bar{\mathbf{X}}_\Omega := \mathbf{R}_\Omega + \bigotimes_{i \in \Omega} \mathbf{x}_i$. E.g., $\bar{\mathbf{X}}_{\{2\}} := \mathbf{R}_{\{2\}} + \mathbf{x}_2$, $\bar{\mathbf{X}}_{\{3,4\}} := \mathbf{R}_{\{3,4\}} + \mathbf{x}_3 \otimes \mathbf{x}_4$.

Within the communication complexity budget $O(N^{\frac{k-1}{2}})$, we can let the referee learn $\bar{\mathbf{X}}_\Omega$ for all Ω such that $|\Omega| \leq k-1$ (more details in Section 3.3.2). The referee does not learn extra information as $\bar{\mathbf{X}}_\Omega$ is one-time padded by $\mathbf{R}_\Omega$. For example when $k = 4$, we can let the referee learn tensors $\bar{\mathbf{X}}_{\{1\}}, \bar{\mathbf{X}}_{\{2\}}, \ldots, \bar{\mathbf{X}}_{\{8\}}, \bar{\mathbf{X}}_{\{1,2\}}, \bar{\mathbf{X}}_{\{1,3\}}, \ldots, \bar{\mathbf{X}}_{\{7,8\}}, \bar{\mathbf{X}}_{\{1,2,3\}}, \bar{\mathbf{X}}_{\{1,2,4\}}, \ldots, \bar{\mathbf{X}}_{\{6,7,8\}}$. The referee learns those tensors by having *subsets of the parties* recursively perform PSM

[1] A note on the randomness complexity: The final protocol uses $\mathbf{R}_\Omega$ only if $|\Omega| \leq k-1$.

protocols with a smaller number of parties, so that the referee learns the required information. Learning those tensors allows the referee to compute many terms including $\langle \mathbf{F}, \bar{\mathbf{X}}_{\{1,2,3\}} \otimes \bar{\mathbf{X}}_{\{4,5,6\}} \otimes \bar{\mathbf{X}}_{\{7,8\}} \rangle$, which equals to the sum of the following 8 terms,

$$\begin{aligned}
&\langle \mathbf{F}, \bar{\mathbf{X}}_{\{1,2,3\}} \otimes \bar{\mathbf{X}}_{\{4,5,6\}} \otimes \bar{\mathbf{X}}_{\{7,8\}} \rangle \\
&= \langle \mathbf{F}, \mathbf{x}_1 \otimes \mathbf{x}_2 \otimes \mathbf{x}_3 \otimes \mathbf{x}_4 \otimes \mathbf{x}_5 \otimes \mathbf{x}_6 \otimes \mathbf{x}_7 \otimes \mathbf{x}_8 \rangle \\
&\quad + \langle \mathbf{F}, \mathbf{x}_1 \otimes \mathbf{x}_2 \otimes \mathbf{x}_3 \otimes \mathbf{x}_4 \otimes \mathbf{x}_5 \otimes \mathbf{x}_6 \otimes \mathbf{R}_{\{7,8\}} \rangle \\
&\quad + \langle \mathbf{F}, \mathbf{x}_1 \otimes \mathbf{x}_2 \otimes \mathbf{x}_3 \otimes \mathbf{R}_{\{4,5,6\}} \otimes \mathbf{x}_7 \otimes \mathbf{x}_8 \rangle \\
&\quad + \langle \mathbf{F}, \mathbf{x}_1 \otimes \mathbf{x}_2 \otimes \mathbf{x}_3 \otimes \mathbf{R}_{\{4,5,6\}} \otimes \mathbf{R}_{\{7,8\}} \rangle \\
&\quad + \langle \mathbf{F}, \mathbf{R}_{\{1,2,3\}} \otimes \mathbf{x}_4 \otimes \mathbf{x}_5 \otimes \mathbf{x}_6 \otimes \mathbf{x}_7 \otimes \mathbf{x}_8 \rangle \\
&\quad + \langle \mathbf{F}, \mathbf{R}_{\{1,2,3\}} \otimes \mathbf{x}_4 \otimes \mathbf{x}_5 \otimes \mathbf{x}_6 \otimes \mathbf{R}_{\{7,8\}} \rangle \\
&\quad + \langle \mathbf{F}, \mathbf{R}_{\{1,2,3\}} \otimes \mathbf{R}_{\{4,5,6\}} \otimes \mathbf{x}_7 \otimes \mathbf{x}_8 \rangle \\
&\quad + \langle \mathbf{F}, \mathbf{R}_{\{1,2,3\}} \otimes \mathbf{R}_{\{4,5,6\}} \otimes \mathbf{R}_{\{7,8\}} \rangle.
\end{aligned} \tag{3.6}$$

Before we continue, let us introduce a few notations to describe the terms appearing in (3.6). The term (tensor) on the left hand side of the equation will be called a *masked term* (masked tensor). The terms (tensors) on the right hand side of the equation will be called *pure terms* (pure tensors).

Definition (masked tensor & masked term)**.** A masked tensor is a tensor product $\bar{\mathbf{X}}_{\Omega_1} \otimes \ldots \otimes \bar{\mathbf{X}}_{\Omega_t}$[2] such that $\Omega_1, \ldots, \Omega_t$ are disjoint and their union equals $[2k]$. The *shape* of a masked tensor $\bar{\mathbf{X}}_{\Omega_1} \otimes \ldots \otimes \bar{\mathbf{X}}_{\Omega_t}$ is the multiset $\{|\Omega_1|, \ldots, |\Omega_t|\}$. The inner product of a masked tensor and $\mathbf{F}$ is called a masked term.

For any multiset P such that $\mathrm{sum}(P) = 2k$, let $\sum \bar{\mathbf{X}}(P)$ denote the sum of all masked tensors of shape P, let $\sum \langle \mathbf{F}, \bar{\mathbf{X}}(P) \rangle$ denote the sum of all masked terms of shape P. We thus have $\sum \langle \mathbf{F}, \bar{\mathbf{X}}(P) \rangle = \langle \mathbf{F}, \sum \bar{\mathbf{X}}(P) \rangle$.

Definition (pure tensor & pure term)**.** A pure tensor is a tensor product $\mathbf{R}_{\Omega_1} \otimes \ldots \otimes \mathbf{R}_{\Omega_t} \otimes \mathbf{x}_{i_1} \otimes \ldots \otimes \mathbf{x}_{i_w}$ such that $\{i_1, \ldots, i_w\}, \Omega_1, \ldots, \Omega_t$ are disjoint and their union equals $[2k]$. The *shape* of a pure tensor $\mathbf{R}_{\Omega_1} \otimes \ldots \otimes \mathbf{R}_{\Omega_t} \otimes \mathbf{x}_{i_1} \otimes \ldots \otimes \mathbf{x}_{i_w}$ is the multiset $\{|\Omega_1|, \ldots, |\Omega_t|\}$. The inner product of a pure tensor and $\mathbf{F}$ is called a pure term.

For any multiset P such that $\mathrm{sum}(P) \leq 2k$, let $\sum \mathbf{R}(P)$ denote the sum of all pure tensors of shape P, let $\sum \langle \mathbf{F}, \mathbf{R}(P) \rangle$ denote the sum of all pure terms of shape P. We thus have $\sum \langle \mathbf{F}, \mathbf{R}(P) \rangle = \langle \mathbf{F}, \sum \mathbf{R}(P) \rangle$.

The pure terms (pure tensors) can be grouped into 3 natural categories:

Target term (target tensor) $\langle \mathbf{F}, \mathbf{x}_1 \otimes \ldots \otimes \mathbf{x}_{2k} \rangle$ is called the target term as it is desired functionality output. The corresponding tensor $\mathbf{x}_1 \otimes \ldots \otimes \mathbf{x}_{2k}$ is called the target tensor.

Easy terms (easy tensors) A pure tensor $\mathbf{R}_{\Omega_1} \otimes \ldots \otimes \mathbf{R}_{\Omega_t} \otimes \mathbf{x}_{i_1} \otimes \ldots \otimes \mathbf{x}_{i_w}$ is called an easy tensor if at most $k+1$ out of the $2k$ dimensions are contributed by vector $\mathbf{x}_i$'s (i.e., $w \leq k+1$). The corresponding term is called an easy term. Every easy term admits a PSM protocol with communication complexity no more than $O(\mathrm{poly}(k) \cdot N^{\frac{k-1}{2}})$ field elements (more details in Section 3.3.2).

[2]We implicitly exchange the order of indices in tensor product. E.g. when $k = 2$, the masked tensor $\bar{\mathbf{X}}_{\{1,4\}} \otimes \bar{\mathbf{X}}_{\{2,3\}}$ is defined by $(\bar{\mathbf{X}}_{\{1,4\}} \otimes \bar{\mathbf{X}}_{\{2,3\}})[j_1, j_2, j_3, j_4] = \bar{\mathbf{X}}_{\{1,4\}}[j_1, j_4] \cdot \bar{\mathbf{X}}_{\{2,3\}}[j_2, j_3]$.

Hard terms (hard tensors) The rest.

With this terminology, we can give *an overview of our PSM protocol*. As the referee can learn $\bar{\mathbf{X}}_\Omega$ for all Ω such that $|\Omega| \le k-1$, the referee can compute any masked term of shape P if $\max(P) \le k-1$. As suggested by Equation (3.6), every masked term is the linear combination of a few pure terms. Ideally, the referee only has to combine some computable masked terms, so that all the hard terms cancel out, resulting a linear combination of the target term and easy terms:

$$\textit{a linear combination of masked terms} = \textit{target term} + \textit{some easy terms}. \tag{3.7}$$

Once we are in this ideal case, the easy terms can be easily removed by standard techniques, resulting the desired k-party PSM protocol for Aux_N^k. (More details are presented in Section 3.3.2.) Therefore, the task is reduced to a linear algebra problem: *is the target term (resp. tensor) spanned by the referee-computable masked terms (resp. tensors) and easy terms (resp. tensors)?*

When solving such linear algebra problem, it is fair to assume that the solution is symmetric. (Otherwise, assume that a solution that looks like (3.7) is asymmetric, it can be symmetrized by applying the symmetric sum on both sides.)

We have defined the (symmetric) sum of terms or tensors of the same shape. For example when $k=4$, $\sum\bar{\mathbf{X}}(3,3,2)$ is defined as the sum of all masked tensors $\bar{\mathbf{X}}_{\Omega_1} \otimes \bar{\mathbf{X}}_{\Omega_2} \otimes \bar{\mathbf{X}}_{\Omega_3}$ such that the multiset $\{|\Omega_1|, |\Omega_2|, |\Omega_3|\}$ equals $\{3,3,2\}$, i.e.

$$\begin{aligned}\sum\bar{\mathbf{X}}(3,3,2) := \bar{\mathbf{X}}_{\{1,2,3\}} \otimes \bar{\mathbf{X}}_{\{4,5,6\}} \otimes \bar{\mathbf{X}}_{\{7,8\}} + \bar{\mathbf{X}}_{\{1,2,3\}} \otimes \bar{\mathbf{X}}_{\{4,5,7\}} \otimes \bar{\mathbf{X}}_{\{6,8\}} \\ + \bar{\mathbf{X}}_{\{1,2,3\}} \otimes \bar{\mathbf{X}}_{\{4,5,8\}} \otimes \bar{\mathbf{X}}_{\{5,6\}} + \bar{\mathbf{X}}_{\{1,2,3\}} \otimes \bar{\mathbf{X}}_{\{4,6,7\}} \otimes \bar{\mathbf{X}}_{\{5,8\}} \\ + \ldots + \bar{\mathbf{X}}_{\{3,4,5\}} \otimes \bar{\mathbf{X}}_{\{6,7,8\}} \otimes \bar{\mathbf{X}}_{\{1,2\}}.\end{aligned}$$

Let's revisit Equation (3.6),

$$\begin{aligned}&\underbrace{\langle \mathbf{F}, \bar{\mathbf{X}}_{\{1,2,3\}} \otimes \bar{\mathbf{X}}_{\{4,5,6\}} \otimes \bar{\mathbf{X}}_{\{7,8\}} \rangle}_{\text{a masked term of shape } \{3,3,2\}} \\ &= \underbrace{\langle \mathbf{F}, \mathbf{x}_1 \otimes \cdots \otimes \mathbf{x}_8 \rangle}_{\text{a pure term of shape } \{\}} + \underbrace{\langle \mathbf{F}, \mathbf{x}_1 \otimes \cdots \otimes \mathbf{x}_6 \otimes \mathbf{R}_{\{7,8\}} \rangle}_{\text{a pure term of shape } \{2\}} \\ &+ \underbrace{\langle \mathbf{F}, \mathbf{x}_1 \otimes \mathbf{x}_2 \otimes \mathbf{x}_3 \otimes \mathbf{R}_{\{4,5,6\}} \otimes \mathbf{x}_7 \otimes \mathbf{x}_8 \rangle + \langle \mathbf{F}, \mathbf{R}_{\{1,2,3\}} \otimes \mathbf{x}_4 \otimes \cdots \otimes \mathbf{x}_8 \rangle}_{\text{pure terms of shape } \{3\}} \\ &+ \underbrace{\langle \mathbf{F}, \mathbf{x}_1 \otimes \mathbf{x}_2 \otimes \mathbf{x}_3 \otimes \mathbf{R}_{\{4,5,6\}} \otimes \mathbf{R}_{\{7,8\}} \rangle + \langle \mathbf{F}, \mathbf{R}_{\{1,2,3\}} \otimes \mathbf{x}_4 \otimes \mathbf{x}_5 \otimes \mathbf{x}_6 \otimes \mathbf{R}_{\{7,8\}} \rangle}_{\text{pure terms of shape } \{3,2\}} \\ &+ \underbrace{\langle \mathbf{F}, \mathbf{R}_{\{1,2,3\}} \otimes \mathbf{R}_{\{4,5,6\}} \otimes \mathbf{x}_7 \otimes \mathbf{x}_8 \rangle}_{\text{a pure term of shape } \{3,3\}} + \underbrace{\langle \mathbf{F}, \mathbf{R}_{\{1,2,3\}} \otimes \mathbf{R}_{\{4,5,6\}} \otimes \mathbf{R}_{\{7,8\}} \rangle}_{\text{a pure term of shape } \{3,3,2\}}.\end{aligned}$$

By applying a symmetric sum on both sides, we get

$$\begin{aligned}\sum\langle \mathbf{F}, \bar{\mathbf{X}}(3,3,2) \rangle = \underbrace{280 \cdot \sum\langle \mathbf{F}, \mathbf{R}() \rangle}_{\text{the target term}} + \underbrace{10 \cdot \sum\langle \mathbf{F}, \mathbf{R}(2) \rangle}_{\text{hard pure terms}} \\ + \underbrace{10 \cdot \sum\langle \mathbf{F}, \mathbf{R}(3) \rangle + \sum\langle \mathbf{F}, \mathbf{R}(3,2) \rangle + \sum\langle \mathbf{F}, \mathbf{R}(3,3) \rangle + \sum\langle \mathbf{F}, \mathbf{R}(3,3,2) \rangle}_{\text{easy pure terms}}.\end{aligned}$$

As another example of the symmetric sum of masked term that the referee can compute,

$$\sum\langle \mathbf{F}, \bar{\mathbf{X}}(2,2,2,2)\rangle = \underbrace{105 \cdot \sum\langle \mathbf{F}, \mathbf{R}()\rangle}_{\text{target term}} + \underbrace{15 \cdot \sum\langle \mathbf{F}, \mathbf{R}(2)\rangle}_{\text{hard pure terms}} + \underbrace{3 \cdot \sum\langle \mathbf{F}, \mathbf{R}(2,2)\rangle + \sum\langle \mathbf{F}, \mathbf{R}(2,2,2)\rangle + \sum\langle \mathbf{F}, \mathbf{R}(2,2,2,2)\rangle}_{\text{easy pure terms}}.$$

By carefully combining the above two equations, we get

$$3 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(3,3,2)\rangle - 2 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(2,2,2,2)\rangle = 630 \cdot \sum\langle \mathbf{F}, \mathbf{R}()\rangle + \text{easy terms}, \tag{3.8}$$

which induces a 4-party PSM whose communication complexity is $O(N^{3/2})$, if we let $\mathbb{F}$ to be any field in which $630 \neq 0$. (Section 3.3.2 explains how Equation (3.8) implies a 4-party PSM with desired communication complexity.)

In the general k-party case, for each legit shape P of masked term (i.e., P is a multiset consisting of positive integers and $\text{sum}(P) = 2k$),

$$\sum\langle \mathbf{F}, \bar{\mathbf{X}}(P)\rangle = \sum_{Q \subseteq P} \alpha(Q) \cdot \sum\langle \mathbf{F}, \mathbf{R}(P \setminus Q)\rangle, \tag{3.9}$$

where $P \setminus Q$ is the multiset subtraction and

$$\alpha(Q) := \frac{(\text{sum}(Q))!}{\prod_{i \in Q} i! \cdot \prod_{m \in \mathbb{Z}^+} (\text{number of } m\text{'s in } Q)!} \tag{3.10}$$

is the following combinatoric number: $\alpha(Q)$ is the number of ways to partition $\text{sum}(Q)$ distinct elements into some unordered subsets $S_1, \ldots, S_t$ such that $Q = \{|S_1|, \ldots, |S_t|\}$. Equations (3.9), (3.10) are proved in Appendix A.1.

3.3.2 The Induced PSM Protocol

In order to develop the previous section smoothly, we skipped some technique details in Section 3.3.1. In this section, we will show how to construct a k-party PSM protocol assuming that the target term is spanned by referee-computable masked terms and easy pure terms.

By our assumption, there are referee-computable masked terms $\bar{\mathbf{X}}^{(1)}, \ldots, \bar{\mathbf{X}}^{(t)}$, easy pure terms $\mathbf{R}^{(1)}, \ldots, \mathbf{R}^{(s)}$, and coefficients $a_1, \ldots a_t, b_1, \ldots, b_s \in \mathbb{F}$ such that

$$\langle \mathbf{F}, \mathbf{x}_1 \otimes \cdots \otimes \mathbf{x}_{2k}\rangle = \sum_{j=1}^{t} a_j \bar{\mathbf{X}}^{(j)} + \sum_{j=1}^{s} b_j \mathbf{R}^{(j)}. \tag{3.11}$$

A k-party PSM for f, together with its correctness and security, is yielded by the following facts:

- Fact I: $\sum_{j=1}^{s} b_j \mathbf{R}^{(j)}$ and $\bar{\mathbf{X}}_\Omega$ for all $0 < |\Omega| \leq k-1$ form a randomized encoding of $\langle \mathbf{F}, \mathbf{x}_1 \otimes \cdots \otimes \mathbf{x}_{2k}\rangle$. That is, they contain the sufficient information to recover $\langle \mathbf{F}, \mathbf{x}_1 \otimes \cdots \otimes \mathbf{x}_{2k}\rangle$, and they are garbled with additional randomness so that no other information can be recovered.
- Fact II: For every $\Omega \subseteq [2k]$ such that $0 < |\Omega| \leq k-1$, there is a PSM protocol for $\bar{\mathbf{X}}_\Omega$ with communication complexity $\text{poly}(k) \cdot N^{\frac{k-1}{2}}$ field elements.

- Fact III: There is a PSM protocol for $\sum_{j=1}^{s} b_j \mathbf{R}^{(j)}$ with communication complexity $\text{poly}(k) \cdot s \cdot N^{\frac{k-1}{2}}$ field elements.

The k-party PSM for f works as the follows: For each $\Omega \subseteq [2k]$ such that $0 < |\Omega| \leq k-1$, use the PSM guaranteed by Fact II to reveal $\bar{\mathbf{X}}_\Omega$ to the referee. Use the PSM guaranteed by Fact III to reveal $\sum_{j=1}^{s} b_j \mathbf{R}^{(j)}$ to the referee. Then Fact I allows the referee to compute the output from Equation (3.11).

Proof of Fact I. Equation (3.11) shows that $\langle \mathbf{F}, \mathbf{x}_1 \otimes \cdots \otimes \mathbf{x}_{2k} \rangle$ can be computed from the encoding. Moreover, the distribution of the encoding is perfectly simulatable: The joint distribution of tensors $\bar{\mathbf{X}}_\Omega$ for $0 < |\Omega| \leq k-1$ is uniform distribution, as they are independently one-time padded. Then the value of $\sum_{j=1}^{s} b_j \mathbf{R}^{(j)}$ is uniquely determined by equation (3.11).

Proof of Fact II. Let $t = |\Omega|$. Each coordinate of $\bar{\mathbf{X}}_\Omega$ is defined as

$$\bar{\mathbf{X}}_{\{j_1,\ldots,j_t\}}[i_1, \ldots, i_t] = \mathbf{R}_{\{j_1,\ldots,j_t\}}[i_1, \ldots, i_t] + \mathbf{x}_{j_1}[i_1] \cdot \ldots \cdot \mathbf{x}_{j_t}[i_t],$$

which is an arithmetic formula of size $O(k)$. Thus each coordinate has a PSM protocol with communication complexity $\text{poly}(k)$ field elements [IK00].

Proof of Fact III. Sample random $c_1, \ldots, c_s \in \mathbb{F}$ from the common random string such that $c_1 + \ldots + c_s = 0$. Then it's sufficient to construct a PSM protocol for computing $b_j \mathbf{R}^{(j)} + c_j$ for each j. Say this easy pure term $\mathbf{R}^{(j)}$ is $\langle \mathbf{F}, \mathbf{R}_{\Omega_1} \otimes \ldots \otimes \mathbf{x}_{i_1} \otimes \ldots \otimes \mathbf{x}_{i_w} \rangle$. By our definition of an easy term, $w \leq k+1$. There exists a special party, such that the other parties hold at most $k-1$ of $\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_w}$. When $w = k+1$, the special party is the one who holds two of $\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_w}$ (the existence is guaranteed by the pigeonhole principle). W.l.o.g. assume that the other parties hold $\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_{w'}}$ such that $w' \leq k-1$. Then the special party knows a dimension-w' tensor $\mathbf{G}$ (which is determined by its input and $b_j, \mathbf{R}_{\Omega_1}, \mathbf{R}_{\Omega_2}, \ldots$) such that

$$b_j \mathbf{R}^{(j)} + c_j = \langle \mathbf{G}, \mathbf{x}_{i_1} \otimes \ldots \otimes \mathbf{x}_{i_{w'}} \rangle + c_j,$$

which admits a PSM protocol (presented in Section A.2.1) with communication complexity $O(\text{poly}(k) \cdot N^{w'/2})$ field elements.

3.3.3 When k is Small

As shown in Section 3.3.1, to construct PSM protocol for Aux_N^k with communication complexity $O_k(N^{\frac{k-1}{2}})$, it is sufficient to prove the target term is spanned by the referee-computable masked terms and easy pure terms. In this section, we verify that the condition holds for all $k \leq 20$, which proves the first bullet of Theorem 3.3.1. However, we do not have a general construction of such linenar systems of equations for an arbitrary k.

The case when $k = 2$ was solved by [BIKK14]. Our framework yields the same protocol from

$$\sum \langle \mathbf{F}, \bar{\mathbf{X}}(1,1,1,1) \rangle = \sum \langle \mathbf{F}, \mathbf{R}() \rangle + \text{easy terms}.$$

The case when $k = 3$ was solved by [BKN18]. Our framework yields a similar protocol from

$$\sum \langle \mathbf{F}, \bar{\mathbf{X}}(2,2,2) \rangle = \sum \langle \mathbf{F}, \mathbf{R}() \rangle + \text{easy terms}.$$

The case when $k = 4$ is solved in section 3.3.1.

For $k = 5$, consider the following two masked terms,

$$\sum\langle \mathbf{F}, \bar{\mathbf{X}}(4,4,2)\rangle = 1575 \cdot \sum\langle \mathbf{F}, \mathbf{R}()\rangle + 35 \cdot \sum\langle \mathbf{F}, \mathbf{R}(2)\rangle + \text{easy terms},$$
$$\sum\langle \mathbf{F}, \bar{\mathbf{X}}(4,2,2,2)\rangle = 3150 \cdot \sum\langle \mathbf{F}, \mathbf{R}()\rangle + 210 \cdot \sum\langle \mathbf{F}, \mathbf{R}(2)\rangle + \text{easy terms}.$$

We have $6 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(4,4,2)\rangle - \sum\langle \mathbf{F}, \bar{\mathbf{X}}(4,2,2,2)\rangle = 6300 \cdot \sum\langle \mathbf{F}, \mathbf{R}()\rangle + \text{easy terms}$, which induces a 5-party PSM with communication complexity $O(N^2)$.

For $k = 6$, consider the following masked terms

$$\begin{bmatrix} \sum\langle \mathbf{F}, \bar{\mathbf{X}}(5,4,3)\rangle \\ \sum\langle \mathbf{F}, \bar{\mathbf{X}}(4,4,4)\rangle \\ \sum\langle \mathbf{F}, \bar{\mathbf{X}}(3,3,3,3)\rangle \end{bmatrix} = \begin{bmatrix} 27720 & 126 & 56 \\ 5775 & & 35 \\ 15400 & 280 & \end{bmatrix} \begin{bmatrix} \sum\langle \mathbf{F}, \mathbf{R}()\rangle \\ \sum\langle \mathbf{F}, \mathbf{R}(3)\rangle \\ \sum\langle \mathbf{F}, \mathbf{R}(4)\rangle \end{bmatrix} + \text{easy terms}$$

Therefore, $100 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(5,4,3)\rangle - 160 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(4,4,4)\rangle - 45 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(3,3,3,3)\rangle = 1155000 \cdot \sum\langle \mathbf{F}, \mathbf{R}()\rangle + \text{easy terms}$, which induces a 6-party PSM with communication complexity $O(N^{2.5})$.

For $k = 7$, consider the following masked terms

$$\begin{bmatrix} \sum\langle \mathbf{F}, \bar{\mathbf{X}}(4,4,4,2)\rangle \\ \sum\langle \mathbf{F}, \bar{\mathbf{X}}(6,6,2)\rangle \\ \sum\langle \mathbf{F}, \bar{\mathbf{X}}(6,4,4)\rangle \end{bmatrix} = \begin{bmatrix} 525525 & 5775 & 1575 \\ 42042 & 462 & \\ 105105 & & 210 \end{bmatrix} \begin{bmatrix} \sum\langle \mathbf{F}, \mathbf{R}()\rangle \\ \sum\langle \mathbf{F}, \mathbf{R}(2)\rangle \\ \sum\langle \mathbf{F}, \mathbf{R}(4)\rangle \end{bmatrix} + \text{easy terms}$$

Therefore, $14 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(4,4,4,2)\rangle - 175 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(6,6,2)\rangle - 105 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(6,4,4)\rangle = -11036025 \cdot \sum\langle \mathbf{F}, \mathbf{R}()\rangle + \text{easy terms}$, which induces a 7-party PSM with communication complexity $O(N^3)$.

For larger k, we wrote a simple program[3] to check if the target term can be spanned by referee-computable masked terms and easy terms. For simplicity, our program requires specifying the finite field in advance. Our program verifies that the framework yields a PSM protocol with c.c. $O(N^{\frac{k-1}{2}})$ for every $k \leq 20$. For example when $k = 20$, our program found:

$$\sum\langle \mathbf{F}, \mathbf{R}()\rangle = 2895 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(19,19,2)\rangle + 1902 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(19,17,4)\rangle + 2843 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(19,16,5)\rangle + 1025 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(19,16,3,2)\rangle + 691 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(19,15,6)\rangle + 2507 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(19,15,4,2)\rangle + 1923 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(19,14,7)\rangle + 1836 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(19,14,5,2)\rangle + 2385 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(19,13,8)\rangle + 2073 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(19,13,6,2)\rangle + 1312 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(19,12,9)\rangle + 2963 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(19,12,7,2)\rangle + 568 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(19,11,10)\rangle + 975 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(19,11,8,2)\rangle + 2445 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(19,10,9,2)\rangle + 2047 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(19,9,8,4)\rangle + 318 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(19,9,8,2,2)\rangle + 2118 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(19,9,6,6)\rangle + 2189 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(19,9,6,4,2)\rangle + 1271 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(19,9,6,2,2,2)\rangle + 1557 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(19,9,4,4,4)\rangle + 2482 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(19,9,4,4,2,2)\rangle + 173 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(19,9,4,2,2,2,2)\rangle + 1943 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(19,9,2,2,2,2,2,2)\rangle + 29 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(18,18,4)\rangle + 1247 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(18,17,5)\rangle + 1768 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(18,17,3,2)\rangle + 2735 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(18,16,6)\rangle + 416 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(18,16,4,2)\rangle + 1009 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(18,15,7)\rangle + 130 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(18,15,5,2)\rangle + 138 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(18,14,8)\rangle + 52 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(18,14,6,2)\rangle + 2661 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(18,13,9)\rangle + 26 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(18,13,7,2)\rangle + 731 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(18,12,10)\rangle + 16 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(18,12,8,2)\rangle + 145 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(18,11,11)\rangle + 12 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(18,11,9,2)\rangle + 818 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(18,10,8,4)\rangle + 1728 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(18,10,8,2,2)\rangle + 2676 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(18,10,6,6)\rangle + 1533 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(18,10,6,4,2)\rangle + 2490 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(18,10,6,2,2,2)\rangle + 760 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(18,10,4,4,4)\rangle + 747 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(18,10,4,4,2,2)\rangle + 2752 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(18,10,4,2,2,2,2)\rangle + 83 \cdot \sum\langle \mathbf{F}, \bar{\mathbf{X}}(18,10,2,2,2,2,2,2)\rangle + \text{easy terms} \mod 3001$$

which induces a PSM protocol with c.c. $O(N^{9.5})$.

3.3.4 When $k+1$ is a Prime Power

As shown in Section 3.3.1, to construct PSM protocol for Aux_N^k with communication complexity $O_k(N^{\frac{k-1}{2}})$, it is sufficient to prove the target term is spanned by the referee-computable masked terms and easy pure terms. In this section, we prove that the condition holds for all k such that $k+1$ is a prime power, which proves the second bullet of Theorem 3.3.1.

When $k+1$ is a prime p or a prime power p^e, we obtain a simple k-party PSM, by doing computations in the finite field $\mathbb{F}_p$.

[3] The source code can be downloaded from https://github.com/tianren/psm.

Proof. Consider the symmetric sum of all masked terms of shape $\{k-1,1,\ldots,1\}$

$$\begin{aligned}
&\sum\langle \mathbf{F}, \bar{\mathbf{X}}(k-1,\underbrace{1,\ldots,1}_{k+1\text{ 1's}})\rangle \\
&= \sum_{i=0}^{k+1} \alpha(k-1,\underbrace{1,\ldots,1}_{k+1-i\text{ 1's}}) \cdot \sum\langle \mathbf{F}, \mathbf{R}(\underbrace{1,\ldots,1}_{i\text{ 1's}})\rangle \\
&\quad + \sum_{i=0}^{k+1} \alpha(\underbrace{1,\ldots,1}_{k+1-i\text{ 1's}}) \cdot \sum\langle \mathbf{F}, \mathbf{R}(k-1,\underbrace{1,\ldots,1}_{i\text{ 1's}})\rangle \qquad (3.12)\\
&= \alpha(k-1,\underbrace{1,\ldots,1}_{k+1\text{ 1's}}) \cdot \sum\langle \mathbf{F}, \mathbf{R}()\rangle \\
&\quad + \sum_{i=1}^{k-2} \alpha(k-1,\underbrace{1,\ldots,1}_{k+1-i\text{ 1's}}) \cdot \sum\langle \mathbf{F}, \mathbf{R}(\underbrace{1,\ldots,1}_{i\text{ 1's}})\rangle + \text{easy terms}.
\end{aligned}$$

(Recall that a pure term of shape P is easy iff $\text{sum}(P) \geq k-1$.)

W.l.o.g. assume $k > 2$. By definition, $\alpha(k-1,\underbrace{1,\ldots,1}_{t\text{ 1's}}) = \binom{k-1+t}{k-1}$. Lemma 3.3.3 shows that $\alpha(k-1,\underbrace{1,\ldots,1}_{k+1\text{ 1's}}) = \binom{2k}{k-1} \equiv 1 \mod p$, while $\alpha(k-1,\underbrace{1,\ldots,1}_{k+1-i\text{ 1's}}) = \binom{2k-i}{k-1}$ is a multiple of p for all $1 \leq i \leq k-2$. Therefore,

$$\sum\langle \mathbf{F}, \bar{\mathbf{X}}(k-1,\underbrace{1,\ldots,1}_{k+1\text{ 1's}})\rangle = \sum\langle \mathbf{F}, \mathbf{R}()\rangle + \text{easy terms} \mod p,$$

which induces a k-party PSM protocol with c.c. $O_k(N^{\frac{k-1}{2}})$.

□

Lemma 3.3.2. *For any prime p and positive integer e, $\binom{p^e}{t}$ is a multiple of p for all $0 < t < p^e$.*

Proof.

$$\binom{p^e}{t} = \frac{p^e}{t} \cdot \binom{p^e-1}{t-1}.$$

□

Lemma 3.3.3. *For any prime p and positive integer e, binomial coefficient $\binom{p^e+t}{p^e-2}$ is a multiple of p for all $0 \leq t \leq p^e-3$, while binomial coefficient $\binom{2p^e-2}{p^e-2} \equiv 1 \mod p$.*

Proof. For every $0 \leq t \leq p^e-3$,

$$\binom{p^e+t}{p^e-2} = \sum_{j=0}^{t} \binom{t}{j} \underbrace{\binom{p^e}{p^e-2-j}}_{\text{multiple of } p}$$

is a multiple of p. While

$$\binom{2p^e-2}{p^e-2} = \sum_{j=0}^{p^e-3} \binom{p^e-2}{j} \underbrace{\binom{p^e}{p^e-2-j}}_{\text{multiple of } p} + \binom{p^e-2}{p^e-2}\binom{p^e}{0} \equiv 1 \mod p.$$

□

3.4 Unbalanced 2-party PSM Protocols

The two parties in 2-party PSM are conventionally called Alice and Bob. Let $x \in [N]$ denote Alice's and $y \in [N]$ denote Bob's input. In this section, we show that every functionality $f : [N] \times [N] \to \{0,1\}$ admits a 2-party PSM protocol, where Alice sends $O(N^\eta)$ bits and Bob sends $O(N^{1-\eta})$ bits.

Theorem 3.4.1. *For any functionality $f : [N] \times [N] \to \{0,1\}$, and any $\eta = d/k$ such that d, k are integers and $0 < d < k \leq 20$, there is a 2-party PSM protocol for f with unbalanced communication complexity $O(N^\eta), O(N^{1-\eta})$.*

In this section, we prove a stronger statement. Let $\mathbb{F}$ be a finite field, consider the following auxiliary 2-party functionality $\mathsf{Aux}^2_{k,N}$:

2-party functionality $\mathsf{Aux}^2_{k,N}$

- Alice has input $\mathbf{x}_1, \ldots, \mathbf{x}_k \in \mathbb{F}^{\sqrt[k]{N}}$
- Bob has input $\mathbf{y}_1, \ldots, \mathbf{y}_k \in \mathbb{F}^{\sqrt[k]{N}}$
- The output is $\langle \mathbf{F}, \mathbf{x}_1 \otimes \ldots \otimes \mathbf{x}_k \otimes \mathbf{y}_1 \otimes \ldots \otimes \mathbf{y}_k \rangle$, where $\mathbf{F}$ is public and fixed

A PSM protocol for $\mathsf{Aux}^2_{k,N}$ implies a PSM for $f : [N] \times [N] \to \{0,1\}$ with the same communication complexity of each party. The reduction consists of having $\mathbf{F}$ be the truth table of f.

We present a framework for the construction of 2-party PSM protocols for $\mathsf{Aux}^2_{k,N}$, where Alice sends $O_\eta(N^\eta)$ bits and Bob sends $O_\eta(N^{1-\eta})$ bits, for all $\eta \in \{\frac{1}{k}, \ldots, \frac{k-1}{k}\}$. Similar to the framework in Section 3.3, the framework in this section also reduces the problem to a system of linear equations. A solution of the system implies a 2-party PSM protocol with the desired communication complexity. By verifying with a computer, we find that our framework works well for all η whose denominator is no larger than 20. Backed by those observations, we believe that our framework allows for a smooth trade-off between the communication complexity of Alice and Bob:

Conjecture 2. For any functionality $f : [N] \times [N] \to \{0,1\}$, and any $0 < \eta < 1$, there is a 2-party PSM protocol for f with unbalanced communication complexity $O_\eta(N^\eta), O_\eta(N^{1-\eta})$.

Organization Section 3.4.1 presents our framework for constructing multi-party PSM, introduces new notations, and gives as a concrete example a 2-party PSM with communication $O(N^{1/3}), O(N^{2/3})$. The following Sections 3.4.2, 3.4.3 are independent. Section 3.4.2 provides more technical detail of the PSM protocols yielded by our framework. Section 3.4.3 shows how the framework works for small k.

3.4.1 A Framework for 2-party PSM

Consider a rational $\eta = \frac{d}{k} \in (0,1)$. Let $\mathbb{F}$ be a finite commutative ring that we will fix later. All the operations are within ring $\mathbb{F}$ unless otherwise specified.

As mentioned in the beginning of Section 3.4, there is an non-interactive reduction from the functionality $f : [N] \times [N] \to \{0,1\}$ to functionality $\mathsf{Aux}^2_{k,N}$. The reduction works as follows: Let $x, y \in [N]$ be the input of Alice and Bob respectively. Evenly divide x into $x_1, \ldots, x_k \in [\sqrt[k]{N}]$, similarly divide y into $y_1, \ldots, y_k \in [\sqrt[k]{N}]$. For each $j \in [k]$, let $\mathbf{x}_j := \mathbf{e}_{x_j} \in \mathbb{F}^{\sqrt{N}}$ be the x_j-th standard unit vector. Similarly let $\mathbf{y}_i := \mathbf{e}_{y_i} \in \mathbb{F}^{\sqrt[k]{N}}$ for every $i \in [k]$. The functionality f can be reduced to $\mathsf{Aux}^2_{k,N}$ by doing:

$$f(x_1, \ldots, x_k, y_1, \ldots, y_k) = \langle \mathbf{F}, \mathbf{x}_1 \otimes \ldots \otimes \mathbf{x}_k \otimes \mathbf{y}_1 \otimes \ldots \otimes \mathbf{y}_k \rangle.$$

where $\mathbf{F}$ is the truth-table of f. For the remainder of the section, it is thus sufficient to construct a PSM protocol for $\mathsf{Aux}^2_{k,N}$.

For every $\Omega \subseteq [k]$, our protocol will sample random $\mathbf{R}_\Omega, \mathbf{S}_\Omega \in \mathbb{F}^{(\sqrt[k]{N})^{|\Omega|}}$ from the common random string. Let $\bar{\mathbf{X}}_\Omega := \mathbf{R}_\Omega + \bigotimes_{i \in \Omega} \mathbf{x}_i$ and $\bar{\mathbf{Y}}_\Omega := \mathbf{S}_\Omega + \bigotimes_{i \in \Omega} \mathbf{y}_i$.

As the communication complexity of Alice is $O_\eta(N^{\frac{d}{k}})$, she can send $\bar{\mathbf{X}}_\Omega$ to the referee for every Ω that $|\Omega| \leq d$. So far no information is leaked as $\bar{\mathbf{X}}_\Omega$ is one-time padded by $\mathbf{R}_\Omega$. Similarly, Bob can send $\bar{\mathbf{Y}}_\Omega$ for every Ω that $|\Omega| \leq k - d$.

There are many meaningful terms that the referee can compute once he receives $(\bar{\mathbf{X}}_\Omega)_{|\Omega| \leq d}$ and $(\bar{\mathbf{Y}}_\Omega)_{|\Omega| \leq k-d}$. For example, when $\eta = d/k = 1/3$, the referee can compute:

$$\begin{aligned}
&\langle \mathbf{F}, \bar{\mathbf{X}}_{\{1\}} \otimes \bar{\mathbf{X}}_{\{2\}} \otimes \bar{\mathbf{X}}_{\{3\}} \otimes \bar{\mathbf{Y}}_{\{1,2\}} \otimes \bar{\mathbf{Y}}_{\{3\}} \rangle \\
&= \langle \mathbf{F}, \mathbf{x}_1 \otimes \mathbf{x}_2 \otimes \mathbf{x}_3 \otimes \mathbf{y}_1 \otimes \mathbf{y}_2 \otimes \mathbf{y}_3 \rangle \\
&\quad + \langle \mathbf{F}, \mathbf{x}_1 \otimes \mathbf{x}_2 \otimes \mathbf{x}_3 \otimes \mathbf{y}_1 \otimes \mathbf{y}_2 \otimes \mathbf{S}_{\{3\}} \rangle \\
&\quad + \langle \mathbf{F}, \mathbf{x}_1 \otimes \mathbf{x}_2 \otimes \mathbf{x}_3 \otimes \mathbf{S}_{\{1,2\}} \otimes \mathbf{S}_{\{3\}} \rangle \\
&\quad + \ldots \quad \text{(28 other terms)} \\
&\quad + \langle \mathbf{F}, \mathbf{R}_{\{1\}} \otimes \mathbf{R}_{\{2\}} \otimes \mathbf{R}_{\{3\}} \otimes \mathbf{S}_{\{1,2\}} \otimes \mathbf{S}_{\{3\}} \rangle.
\end{aligned} \tag{3.13}$$

Before we continue, we have to introduce a few notations. We will define shape, masked tensor, pure tensor, easy & hard tensor, etc., in the same way as in Section 3.3.1.

Definition (masked tensor & masked term)**.** An Alice-side masked tensor is a tensor product $\bar{\mathbf{X}}_{\Omega_1} \otimes \ldots \otimes \bar{\mathbf{X}}_{\Omega_t}$ such that $\Omega_1, \ldots, \Omega_t$ are disjoint and their union equals $[k]$. The *shape* of an Alice-side masked tensor $\bar{\mathbf{X}}_{\Omega_1} \otimes \ldots \otimes \bar{\mathbf{X}}_{\Omega_t}$ is the multiset $\{|\Omega_1|, \ldots, |\Omega_t|\}$. Bob-side masked tensors are defined symmetrically.

The tensor product of an Alice-side masked tensor and a Bob-side masked tensor is called a *masked tensor*. The inner product of $\mathbf{F}$ and a masked tensor is called a *masked term*.

An Alice-side masked tensor of shape P is referee-computable if $\max(P) \leq d$. A Bob-side masked tensor of shape Q is referee-computable if $\max(Q) \leq k - d$. A masked tensor (and its corresponding masked term) is called *referee-computable* if it's the tensor product of a referee-computable Alice-side masked tensor and a referee-computable Bob-side masked tensor.

Definition (pure tensor & pure term)**.** An Alice-side pure tensor is a tensor product $\mathbf{R}_{\Omega_1} \otimes \ldots \otimes \mathbf{R}_{\Omega_t} \otimes \mathbf{x}_{i_1} \otimes \ldots \otimes \mathbf{x}_{i_w}$ such that $\{i_1, \ldots, i_w\}, \Omega_1, \ldots, \Omega_t$ are disjoint and their union equals $[k]$. The *shape* of an Alice-side masked tensor $\mathbf{R}_{\Omega_1} \otimes \ldots \otimes \mathbf{R}_{\Omega_t} \otimes \mathbf{x}_{i_1} \otimes \ldots \otimes \mathbf{x}_{i_w}$ is the multiset $\{|\Omega_1|, \ldots, |\Omega_t|\}$. Bob-side pure tensors are defined symmetrically.

The tensor product of an Alice-side pure tensor and a Bob-side pure tensor is called a *pure tensor*. The inner product of a pure tensor and $\mathbf{F}$ is called a *pure term*.

For any legit shape, let $\sum\mathbf{R}(P)$ denote the sum of all Alice-side pure tensor whose shape is P. Similarly, define Bob-side pure tensor sum $\sum\mathbf{S}(P)$.

Let's go back to the example when $\eta = 1/3$: examine the pure terms on the right side of equation (3.13), and check which of them has a 2-party PSM with communication complexity $O(N^{\frac{1}{3}}), O(N^{\frac{2}{3}})$.

- The term $\langle \mathbf{F}, \mathbf{x}_1 \otimes \mathbf{x}_2 \otimes \mathbf{x}_3 \otimes \mathbf{y}_1 \otimes \mathbf{y}_2 \otimes \mathbf{y}_3 \rangle$ is the desired functionality.
- The term $\langle \mathbf{F}, \mathbf{x}_1 \otimes \mathbf{x}_2 \otimes \mathbf{x}_3 \otimes \mathbf{S}_{\{1,2\}} \otimes \mathbf{y}_3 \rangle$ has a PSM protocol with communication complexity $O(N^{\frac{1}{3}})$. Because Alice knows a vector $\mathbf{g}$ (which is determined by $\mathbf{F}$, Alice's input and randomness $(\mathbf{R}_\Omega)_\Omega, (\mathbf{S}_\Omega)_\Omega$) such that $\langle \mathbf{F}, \mathbf{x}_1 \otimes \mathbf{x}_2 \otimes \mathbf{x}_3 \otimes \mathbf{S}_{\{1,2\}} \otimes \mathbf{y}_3 \rangle = \langle \mathbf{g}, \mathbf{y}_3 \rangle$.
- The term $\langle \mathbf{F}, \mathbf{S}_{\{1\}} \otimes \mathbf{x}_2 \otimes \mathbf{x}_3 \otimes \mathbf{y}_1 \otimes \mathbf{y}_2 \otimes \mathbf{y}_3 \rangle$ admits a PSM protocol with unbalanced communication complexity $O(N^{\frac{1}{3}})$, $O(N^{\frac{2}{3}})$. Because Bob knows a dimension-2 tensor $\mathbf{G}$ such that $\langle \mathbf{F}, \mathbf{S}_{\{1\}} \otimes \mathbf{x}_2 \otimes \mathbf{x}_3 \otimes \mathbf{y}_1 \otimes \mathbf{y}_2 \otimes \mathbf{y}_3 \rangle = \langle \mathbf{x}_2 \otimes \mathbf{x}_3, \mathbf{G} \rangle$. (This PSM is presented in Section A.2.2.)

The discussion above hints at the right classification of pure tensors.

target tensor The only Alice-side target tensor is $\mathbf{x}_1 \otimes \cdots \otimes \mathbf{x}_k$. The only Bob-side target tensor is $\mathbf{y}_1 \otimes \cdots \otimes \mathbf{y}_k$. The only target tensor is $\mathbf{x}_1 \otimes \cdots \otimes \mathbf{x}_k \otimes \mathbf{y}_1 \otimes \cdots \otimes \mathbf{y}_k$.

easy tensor An Alice-side pure tensor of shape P is called easy if $\mathrm{sum}(P) \geq d$. A Bob-side pure tensor of shape Q is called easy if $\mathrm{sum}(Q) \geq k - d$. A pure tensor $\mathbf{R} \otimes \mathbf{S}$ is called easy if either $\mathbf{R}$ or $\mathbf{S}$ is easy.

hard tensor The rest.

The inner product of $\mathbf{F}$ and a target/easy/hard tensor is called a target/easy/hard term.

Then, equation (3.13) can be rewritten by grouping and ignoring the easy terms:

$$\begin{aligned} &\langle \mathbf{F}, \bar{\mathbf{X}}_{\{1\}} \otimes \bar{\mathbf{X}}_{\{2\}} \otimes \bar{\mathbf{X}}_{\{3\}} \otimes \bar{\mathbf{Y}}_{\{1,2\}} \otimes \bar{\mathbf{Y}}_{\{3\}} \rangle \\ &= \langle \mathbf{F}, \mathbf{x}_1 \otimes \mathbf{x}_2 \otimes \mathbf{x}_3 \otimes \mathbf{y}_1 \otimes \mathbf{y}_2 \otimes \mathbf{y}_3 \rangle \\ &\quad + \langle \mathbf{F}, \mathbf{x}_1 \otimes \mathbf{x}_2 \otimes \mathbf{x}_3 \otimes \mathbf{y}_1 \otimes \mathbf{y}_2 \otimes \mathbf{S}_{\{3\}} \rangle + \text{easy terms} \end{aligned}.$$

By a symmetric sum, we get

$$\begin{aligned} &\langle \mathbf{F}, \textstyle\sum \bar{\mathbf{X}}(1,1,1) \otimes \sum \bar{\mathbf{Y}}(2,1) \rangle \\ &= 3 \cdot \underbrace{\langle \mathbf{F}, \textstyle\sum \mathbf{R}() \otimes \sum \mathbf{S}() \rangle}_{\text{target}} + \langle \mathbf{F}, \textstyle\sum \mathbf{R}() \otimes \sum \mathbf{S}(1) \rangle + \text{easy terms}. \end{aligned}$$

Similarly, we have decomposed another referee-computable term

$$\begin{aligned} &\langle \mathbf{F}, \textstyle\sum \bar{\mathbf{X}}(1,1,1) \otimes \sum \bar{\mathbf{Y}}(1,1,1) \rangle \\ &= \underbrace{\langle \mathbf{F}, \textstyle\sum \mathbf{R}() \otimes \sum \mathbf{S}() \rangle}_{\text{target}} + \langle \mathbf{F}, \textstyle\sum \mathbf{R}() \otimes \sum \mathbf{S}(1) \rangle + \text{easy terms}. \end{aligned}$$

Combine them to cancel out the hard terms:

$$\begin{aligned} &\langle \mathbf{F}, \textstyle\sum \bar{\mathbf{X}}(1,1,1) \otimes \sum \bar{\mathbf{Y}}(2,1) \rangle - \langle \mathbf{F}, \textstyle\sum \bar{\mathbf{X}}(1,1,1) \otimes \sum \bar{\mathbf{Y}}(1,1,1) \rangle \\ &= 2 \cdot \langle \mathbf{F}, \textstyle\sum \mathbf{R}() \otimes \sum \mathbf{S}() \rangle + \text{easy terms}. \end{aligned}$$

Thus, by setting $\mathbb{F}$ to be any finite field where $2 \neq 0$, the above equation induces a 2-party PSM protocol with unbalanced communication complexity $O(N^{\frac{1}{3}}), O(N^{\frac{2}{3}})$.

In general, a masked term $\langle \mathbf{F}, \sum\bar{\mathbf{X}}(P) \otimes \sum\bar{\mathbf{Y}}(Q)\rangle$ can be decomposed into pure terms by

$$\begin{aligned}
\sum\bar{\mathbf{X}}(P) &= \sum_{P' \subseteq P} \alpha(P') \sum\bar{\mathbf{X}}(P \setminus P'), \\
\sum\bar{\mathbf{Y}}(Q) &= \sum_{Q' \subseteq Q} \alpha(Q') \sum\bar{\mathbf{Y}}(Q \setminus Q'), \\
\langle \mathbf{F}, \sum\bar{\mathbf{X}}(P) \otimes \sum\bar{\mathbf{Y}}(Q)\rangle &= \sum_{\substack{P' \subseteq P \\ Q' \subseteq Q}} \alpha(P')\alpha(Q') \Big\langle \mathbf{F}, \sum\bar{\mathbf{X}}(P \setminus P') \otimes \sum\bar{\mathbf{Y}}(Q \setminus Q') \Big\rangle.
\end{aligned}$$

with the combinatoric number α defined as in Section 3.3.1. The first two equations are essentially the same as equation (3.9) and they imply the third equation.

To construct a PSM protocol of the desired unbalanced communication complexity, it is sufficient to show the target term is spanned by the referee-computable masked terms and the easy terms. Namely,

$$\begin{aligned}
&\textit{the target term} = \textit{a linear combination of referee-computable masked terms} + \\
&\quad \textit{a linear combination of easy terms.}
\end{aligned} \tag{3.14}$$

The details of how this sufficient condition implies a PSM with desired communication complexity is presented in Section 3.4.2.

This sufficient condition of form (3.14) is unfortunately too combinatorically hard to use in practice, especially since we are going to use a program to search for the proof for different values of η. There are too many distinct masked terms and pure terms – their number is equal to the number of pairs of legit shapes (P, Q).

Fortunately, we come up with a simpler sufficient condition. A PSM protocol of the desired unbalanced communication complexity exists if both of the following hold:

- The Alice-side target tensor is spanned by referee-computable Alice-side masked tensors and Alice-side easy tensors;
- The Bob-side target tensor is spanned by referee-computable Bob-side masked tensors and Bob-side easy tensors.

The proof is quite straight forward: Assume the new sufficient condition,

$$\begin{aligned}
&\textit{a linear combination of referee-computable Alice-side masked tensors} \\
&\quad = \sum\mathbf{R}() + \textit{Alice-side easy tensors}, \\
&\textit{a linear combination of referee-computable Bob-side masked tensors} \\
&\quad = \sum\mathbf{S}() + \textit{Bob-side easy tensors.}
\end{aligned}$$

The tensor product of the above two equations is

$$\begin{aligned}
&\textit{a linear combination of referee-computable masked tensors} \\
&\quad = \sum\mathbf{R}() \otimes \sum\mathbf{S}() + \textit{a linear combination of easy tensors.}
\end{aligned}$$

Multiplying both sides of the above equation with $\mathbf{F}$ yields the desired sufficient condition of form (3.14).

3.4.2 The Induced PSM Protocol

In order to develop the previous section smoothly, we skipped the technique details on how the condition (3.14) implies a 2-party PSM of the desired communication complexity. In this section, we will show how to construct such a 2-party PSM protocol assuming that the target term is spanned by referee-computable masked terms and easy pure terms.

By the condition (3.14), there are referee-computable masked terms $\bar{\mathbf{Z}}^{(1)}, \ldots, \bar{\mathbf{Z}}^{(t)}$, easy pure terms $\mathbf{T}^{(1)}, \ldots, \mathbf{T}^{(s)}$, and coefficients $a_1, \ldots a_t, b_1, \ldots, b_s \in \mathbb{F}$ such that

$$\langle \mathbf{F}, \mathbf{x}_1 \otimes \cdots \otimes \mathbf{x}_k \otimes \mathbf{y}_1 \otimes \cdots \otimes \mathbf{y}_k \rangle = \sum_{j=1}^{t} a_j \bar{\mathbf{Z}}^{(j)} + \sum_{j=1}^{s} b_j \mathbf{T}^{(j)}. \tag{3.15}$$

A 2-party PSM for f, together with its correctness and security, is yielded by the following facts:

- Fact I: $\sum_{j=1}^{s} b_j \mathbf{T}^{(j)}$ together with $\bar{\mathbf{X}}_\Omega$ for all $0 < |\Omega| \leq d$ and $\bar{\mathbf{Y}}_\Omega$ for all $0 < |\Omega| \leq k - d$ form a randomized encoding of the functionality output.

- Fact II: There is a PSM protocol for $\sum_{j=1}^{s} b_j \mathbf{T}^{(j)}$, in which Alice sends $k \cdot s \cdot N^{\frac{d}{k}}$ field elements, Bob sends $k \cdot s \cdot N^{1-\frac{d}{k}}$ field elements.

The 2-party PSM for f works as the follows: For each $\Omega \subseteq [k]$ such that $0 < |\Omega| \leq d$, Alice sends $\bar{\mathbf{X}}_\Omega$ to the referee. Symmetrically, for $\Omega \subseteq [k]$ such that $0 < |\Omega| \leq k - d$, Bob sends $\bar{\mathbf{Y}}_\Omega$ to the referee. Use the PSM guaranteed by Fact II to reveal $\sum_{j=1}^{s} b_j \mathbf{T}^{(j)}$ to the referee. Then Fact I allows the referee to compute the output from Equation (3.15).

Proof of Fact I. (Similar to the proof of Fact I in Section 3.3.2.) Equation (3.15) shows that $\langle \mathbf{F}, \mathbf{x}_1 \otimes \cdots \otimes \mathbf{x}_k \otimes \mathbf{y}_1 \otimes \cdots \otimes \mathbf{y}_k \rangle$ can be computed from the encoding. Moreover, the distribution of the encoding is perfectly simulatable: The joint distribution of tensors $\bar{\mathbf{X}}_\Omega$ for $0 < |\Omega| \leq d$ and $\bar{\mathbf{Y}}_\Omega$ for $0 < |\Omega| \leq k - d$ is uniform, as they are independently one-time padded. Then the value of $\sum_{j=1}^{s} b_j \mathbf{T}^{(j)}$ is uniquely determined by Equation (3.15).

Proof of Fact II. Sample random $c_1, \ldots, c_s \in \mathbb{F}$ from the common random string such that $c_1 + \ldots + c_s = 0$. Then it's sufficient to construct a PSM protocol for computing $b_j \mathbf{T}^{(j)} + c_j$ for each j.

Because $\mathbf{T}^{(j)}$ is an easy term, we have $\mathbf{T}^{(j)} = \langle \mathbf{F}, \mathbf{R}^{(j)} \otimes \mathbf{S}^{(j)} \rangle$, where $\mathbf{R}^{(j)}$ is an Alice-side pure tensor, $\mathbf{S}^{(j)}$ is a Bob-side pure tensor, and either $\mathbf{R}^{(j)}$ is an Alice-side easy tensor, $\mathbf{S}^{(j)}$ is a Bob-side easy tensor. W.l.o.g., assume $\mathbf{R}^{(j)}$ is an Alice-side easy tensor.

Say this Alice-side easy pure term $\mathbf{R}^{(j)}$ is $\mathbf{R}_{\Omega_1} \otimes \ldots \otimes \mathbf{x}_{i_1} \otimes \ldots \otimes \mathbf{x}_{i_w}$. By the definition of an Alice-side easy term, $w \leq k - d$. Then Bob knows a dimension-w tensor $\mathbf{G}$ (which is determined by $\mathbf{S}^{(j)}, b_j, \mathbf{R}_{\Omega_1}, \mathbf{R}_{\Omega_2}, \ldots$) such that

$$b_j \mathbf{T}^{(j)} + c_j = \langle \mathbf{G}, \mathbf{x}_{i_1} \otimes \ldots \otimes \mathbf{x}_{i_w} \rangle + c_j,$$

which admits a PSM protocol (presented in Section A.2.2) in which Alice sends $O(w \cdot N^{1/k})$ field elements, Bob sends $N^{w/k}$ field elements.

3.4.3 When η has a Small Denominator

Section 3.4.1 proves a sufficient condition that implies 2-party PSM protocols with the desired unbalanced communication complexity. In this section, we will verify that the sufficient condition holds for all rational $\eta \in (0,1)$ whose denominator is no larger than 20. Theorem 3.4.1 follows as a consequence.

For $\eta = 1/3$, the 2-party PSM protocol in Section 3.4.1 is also induced by

$$\begin{aligned} \sum \bar{\mathbf{X}}(1,1,1) &= \sum \mathbf{R}() + \text{Alice-side easy tensors}, \\ \sum \bar{\mathbf{Y}}(2,1) - \sum \bar{\mathbf{Y}}(1,1,1) &= 2 \cdot \sum \mathbf{S}() + \text{Bob-side easy tensors}. \end{aligned}$$

For $\eta = 1/4$, a 2-party PSM protocol with c.c. $O(N^{1/4})$, $O(N^{3/4})$ is induced by

$$\begin{aligned} \sum \bar{\mathbf{X}}(1,1,1,1) &= \sum \mathbf{R}() + \text{Alice-side easy tensors}, \\ \sum \bar{\mathbf{Y}}(1,1,1,1) + 2 \cdot \sum \bar{\mathbf{Y}}(3,1) & \\ + \sum \bar{\mathbf{Y}}(2,2) - \sum \bar{\mathbf{Y}}(2,1,1) &= 6 \cdot \sum \mathbf{S}() + \text{Bob-side easy tensors}. \end{aligned}$$

For $\eta = 1/5$, a 2-party PSM protocol with desired c.c. is induced by

$$\begin{aligned} \sum \bar{\mathbf{X}}(1,1,1,1,1) &= \sum \mathbf{R}() + \text{Alice-side easy tensors}, \\ 6 \cdot \sum \bar{\mathbf{Y}}(4,1) + 2 \cdot \sum \bar{\mathbf{Y}}(3,2) & \\ - 2 \cdot \sum \bar{\mathbf{Y}}(3,1,1) - \sum \bar{\mathbf{Y}}(2,2,1) & \\ + \sum \bar{\mathbf{Y}}(2,1,1,1) - \sum \bar{\mathbf{Y}}(1,1,1,1,1) &= 24 \cdot \sum \mathbf{S}() + \text{Bob-side easy tensors}. \end{aligned}$$

For $\eta = 2/5$, a 2-party PSM protocol with desired c.c. is induced by

$$\begin{aligned} 2 \cdot \sum \bar{\mathbf{X}}(2,2,1) - \sum \bar{\mathbf{X}}(2,1,1,1) &= 20 \cdot \sum \mathbf{R}() + \text{Alice-side easy tensors}, \\ 3 \cdot \sum \bar{\mathbf{Y}}(3,2) + \sum \bar{\mathbf{Y}}(3,1,1) & \\ - \sum \bar{\mathbf{Y}}(2,2,1) - \sum \bar{\mathbf{Y}}(1,1,1,1,1) &= 24 \cdot \sum \mathbf{S}() + \text{Bob-side easy tensors}. \end{aligned}$$

For larger denominators, we wrote a computer program[4] to assist us in the proof. For example, for $\eta = 7/20$, a 2-party PSM with desired c.c. is induced by

$$\begin{aligned} &\Sigma\mathbf{R}() = \text{Alice-side easy tensors} + 18 \cdot \Sigma\bar{\mathbf{X}}(7,7,6) + 10 \cdot \Sigma\bar{\mathbf{X}}(7,7,5,1) + 14 \cdot \Sigma\bar{\mathbf{X}}(7,7,4,2) + 14 \cdot \Sigma\bar{\mathbf{X}}(7,7,4,1,1) + 17 \cdot \Sigma\bar{\mathbf{X}}(7,7,3,3) + 20 \cdot \Sigma\bar{\mathbf{X}}(7,7,3,2,1) + \\ &20 \cdot \Sigma\bar{\mathbf{X}}(7,7,3,1,1,1) + 10 \cdot \Sigma\bar{\mathbf{X}}(7,7,2,2,2) + 10 \cdot \Sigma\bar{\mathbf{X}}(7,7,2,2,1,1) + 10 \cdot \Sigma\bar{\mathbf{X}}(7,7,2,1,1,1,1) + 10 \cdot \Sigma\bar{\mathbf{X}}(7,7,1,1,1,1,1,1) + 6 \cdot \Sigma\bar{\mathbf{X}}(7,6,6,1) + 19 \cdot \\ &\Sigma\bar{\mathbf{X}}(7,6,5,2) + 19 \cdot \Sigma\bar{\mathbf{X}}(7,6,5,1,1) + 21 \cdot \Sigma\bar{\mathbf{X}}(7,6,4,3) + 22 \cdot \Sigma\bar{\mathbf{X}}(7,6,4,2,1) + 22 \cdot \Sigma\bar{\mathbf{X}}(7,6,4,1,1,1) + 7 \cdot \Sigma\bar{\mathbf{X}}(7,6,3,3,1) + 15 \cdot \Sigma\bar{\mathbf{X}}(7,6,3,2,2) + 15 \cdot \\ &\Sigma\bar{\mathbf{X}}(7,6,3,2,1,1) + 15 \cdot \Sigma\bar{\mathbf{X}}(7,6,3,1,1,1,1) + 19 \cdot \Sigma\bar{\mathbf{X}}(7,6,2,2,2,1) + 19 \cdot \Sigma\bar{\mathbf{X}}(7,6,2,2,1,1,1) + 19 \cdot \Sigma\bar{\mathbf{X}}(7,6,2,1,1,1,1,1) + 19 \cdot \Sigma\bar{\mathbf{X}}(7,6,1,1,1,1,1,1,1) \\ &\mod 23 \end{aligned}$$

$$\begin{aligned} &\Sigma\mathbf{S}() = \text{Bob-side easy tensors} + 13 \cdot \Sigma\bar{\mathbf{Y}}(13,7) + 20 \cdot \Sigma\bar{\mathbf{Y}}(13,6,1) + 2 \cdot \Sigma\bar{\mathbf{Y}}(13,5,2) + 22 \cdot \Sigma\bar{\mathbf{Y}}(13,5,1,1) + 1 \cdot \Sigma\bar{\mathbf{Y}}(13,4,3) + 17 \cdot \Sigma\bar{\mathbf{Y}}(13,4,2,1) + 3 \cdot \\ &\Sigma\bar{\mathbf{Y}}(13,4,1,1,1) + 19 \cdot \Sigma\bar{\mathbf{Y}}(13,3,3,1) + 21 \cdot \Sigma\bar{\mathbf{Y}}(13,3,2,2) + 1 \cdot \Sigma\bar{\mathbf{Y}}(13,3,2,1,1) + 11 \cdot \Sigma\bar{\mathbf{Y}}(13,3,1,1,1,1) + 12 \cdot \Sigma\bar{\mathbf{Y}}(13,2,2,2,1) + 17 \cdot \Sigma\bar{\mathbf{Y}}(13,2,2,1,1,1) + \\ &3 \cdot \Sigma\bar{\mathbf{Y}}(13,2,1,1,1,1,1) + 10 \cdot \Sigma\bar{\mathbf{Y}}(13,1,1,1,1,1,1,1) + 11 \cdot \Sigma\bar{\mathbf{Y}}(12,8) + 17 \cdot \Sigma\bar{\mathbf{Y}}(12,7,1) + 1 \cdot \Sigma\bar{\mathbf{Y}}(12,6,2) + 11 \cdot \Sigma\bar{\mathbf{Y}}(12,6,1,1) + 17 \cdot \Sigma\bar{\mathbf{Y}}(11,9) + 14 \cdot \\ &\Sigma\bar{\mathbf{Y}}(11,8,1) + 6 \cdot \Sigma\bar{\mathbf{Y}}(11,7,2) + 20 \cdot \Sigma\bar{\mathbf{Y}}(11,7,1,1) + 7 \cdot \Sigma\bar{\mathbf{Y}}(11,6,3) + 4 \cdot \Sigma\bar{\mathbf{Y}}(11,6,2,1) + 21 \cdot \Sigma\bar{\mathbf{Y}}(11,6,1,1,1) + 2 \cdot \Sigma\bar{\mathbf{Y}}(10,10) + 4 \cdot \Sigma\bar{\mathbf{Y}}(10,9,1) + 15 \cdot \\ &\Sigma\bar{\mathbf{Y}}(10,8,2) + 4 \cdot \Sigma\bar{\mathbf{Y}}(10,8,1,1) + 1 \cdot \Sigma\bar{\mathbf{Y}}(10,7,3) + 17 \cdot \Sigma\bar{\mathbf{Y}}(10,7,2,1) + 3 \cdot \Sigma\bar{\mathbf{Y}}(10,7,1,1,1) + 21 \cdot \Sigma\bar{\mathbf{Y}}(10,6,4) + 8 \cdot \Sigma\bar{\mathbf{Y}}(10,6,3,1) + 4 \cdot \Sigma\bar{\mathbf{Y}}(10,6,2,2) + \\ &21 \cdot \Sigma\bar{\mathbf{Y}}(10,6,2,1,1) + 1 \cdot \Sigma\bar{\mathbf{Y}}(10,6,1,1,1,1) + 20 \cdot \Sigma\bar{\mathbf{Y}}(9,9,2) + 13 \cdot \Sigma\bar{\mathbf{Y}}(9,9,1,1) + 4 \cdot \Sigma\bar{\mathbf{Y}}(9,8,3) + 22 \cdot \Sigma\bar{\mathbf{Y}}(9,8,2,1) + 12 \cdot \Sigma\bar{\mathbf{Y}}(9,8,1,1,1) + 14 \cdot \\ &\Sigma\bar{\mathbf{Y}}(9,7,4) + 13 \cdot \Sigma\bar{\mathbf{Y}}(9,7,3,1) + 18 \cdot \Sigma\bar{\mathbf{Y}}(9,7,2,2) + 14 \cdot \Sigma\bar{\mathbf{Y}}(9,7,2,1,1) + 16 \cdot \Sigma\bar{\mathbf{Y}}(9,7,1,1,1,1) + 11 \cdot \Sigma\bar{\mathbf{Y}}(9,6,5) + 13 \cdot \Sigma\bar{\mathbf{Y}}(9,6,4,1) + 12 \cdot \Sigma\bar{\mathbf{Y}}(9,6,3,2) + \\ &17 \cdot \Sigma\bar{\mathbf{Y}}(9,6,3,1,1) + 20 \cdot \Sigma\bar{\mathbf{Y}}(9,6,2,2,1) + 13 \cdot \Sigma\bar{\mathbf{Y}}(9,6,2,1,1,1) + 5 \cdot \Sigma\bar{\mathbf{Y}}(9,6,1,1,1,1,1) + 19 \cdot \Sigma\bar{\mathbf{Y}}(8,8,4) + 16 \cdot \Sigma\bar{\mathbf{Y}}(8,8,3,1) + 8 \cdot \Sigma\bar{\mathbf{Y}}(8,8,2,2) + \\ &19 \cdot \Sigma\bar{\mathbf{Y}}(8,8,2,1,1) + 2 \cdot \Sigma\bar{\mathbf{Y}}(8,8,1,1,1,1) + 17 \cdot \Sigma\bar{\mathbf{Y}}(8,7,5) + 18 \cdot \Sigma\bar{\mathbf{Y}}(8,7,4,1) + 6 \cdot \Sigma\bar{\mathbf{Y}}(8,7,3,2) + 20 \cdot \Sigma\bar{\mathbf{Y}}(8,7,3,1,1) + 10 \cdot \Sigma\bar{\mathbf{Y}}(8,7,2,2,1) + \\ &18 \cdot \Sigma\bar{\mathbf{Y}}(8,7,2,1,1,1) + 14 \cdot \Sigma\bar{\mathbf{Y}}(8,7,1,1,1,1,1) + 18 \cdot \Sigma\bar{\mathbf{Y}}(8,6,6) + 6 \cdot \Sigma\bar{\mathbf{Y}}(8,6,5,1) + 13 \cdot \Sigma\bar{\mathbf{Y}}(8,6,4,2) + 5 \cdot \Sigma\bar{\mathbf{Y}}(8,6,4,1,1) + 1 \cdot \Sigma\bar{\mathbf{Y}}(8,6,3,3) + \\ &17 \cdot \Sigma\bar{\mathbf{Y}}(8,6,3,2,1) + 3 \cdot \Sigma\bar{\mathbf{Y}}(8,6,3,1,1,1) + 20 \cdot \Sigma\bar{\mathbf{Y}}(8,6,2,2,2) + 13 \cdot \Sigma\bar{\mathbf{Y}}(8,6,2,2,1,1) + 5 \cdot \Sigma\bar{\mathbf{Y}}(8,6,2,1,1,1,1) + 9 \cdot \Sigma\bar{\mathbf{Y}}(8,6,1,1,1,1,1,1) + \\ &5 \cdot \Sigma\bar{\mathbf{Y}}(7,7,6) + 1 \cdot \Sigma\bar{\mathbf{Y}}(7,7,5,1) + 6 \cdot \Sigma\bar{\mathbf{Y}}(7,7,4,2) + 20 \cdot \Sigma\bar{\mathbf{Y}}(7,7,4,1,1) + 4 \cdot \Sigma\bar{\mathbf{Y}}(7,7,3,3) + 22 \cdot \Sigma\bar{\mathbf{Y}}(7,7,3,2,1) + 12 \cdot \Sigma\bar{\mathbf{Y}}(7,7,3,1,1,1) + 11 \cdot \\ &\Sigma\bar{\mathbf{Y}}(7,7,2,2,2) + 6 \cdot \Sigma\bar{\mathbf{Y}}(7,7,2,2,1,1) + 20 \cdot \Sigma\bar{\mathbf{Y}}(7,7,2,1,1,1,1) + 13 \cdot \Sigma\bar{\mathbf{Y}}(7,7,1,1,1,1,1,1) + 15 \cdot \Sigma\bar{\mathbf{Y}}(7,6,6,1) + 13 \cdot \Sigma\bar{\mathbf{Y}}(7,6,5,2) + 5 \cdot \Sigma\bar{\mathbf{Y}}(7,6,5,1,1) + \\ &18 \cdot \Sigma\bar{\mathbf{Y}}(7,6,4,3) + 7 \cdot \Sigma\bar{\mathbf{Y}}(7,6,4,2,1) + 8 \cdot \Sigma\bar{\mathbf{Y}}(7,6,4,1,1,1) + 20 \cdot \Sigma\bar{\mathbf{Y}}(7,6,3,3,1) + 10 \cdot \Sigma\bar{\mathbf{Y}}(7,6,3,2,2) + 18 \cdot \Sigma\bar{\mathbf{Y}}(7,6,3,2,1,1) + 14 \cdot \Sigma\bar{\mathbf{Y}}(7,6,3,1,1,1,1) + \\ &9 \cdot \Sigma\bar{\mathbf{Y}}(7,6,2,2,2,1) + 7 \cdot \Sigma\bar{\mathbf{Y}}(7,6,2,2,1,1,1) + 8 \cdot \Sigma\bar{\mathbf{Y}}(7,6,2,1,1,1,1,1) + 19 \cdot \Sigma\bar{\mathbf{Y}}(7,6,1,1,1,1,1,1,1) \mod 23 \end{aligned}$$

We checked every rational $\eta = d/k$ such that $k \leq 20$, and verified that our framework does in fact yield a 2-party PSM protocol with unbalanced communication complexity $O(N^{\eta})$, $O(N^{1-\eta})$.

[4] The source code can be downloaded from https://github.com/tianren/psm.

3.5 Open Problems

This chapter presents two frameworks: a framework for constructing k-party PSM protocols for general $f : [N]^k \to \{0,1\}$ with c.c. $O_k(N^{\frac{k-1}{2}})$, and a framework for constructing 2-party PSM protocols for general $f : [N] \times [N] \to \{0,1\}$ where one party sends $O_\eta(N^\eta)$ bits and the other party sends $O_\eta(N^{1-\eta})$ bits. An immediate open problem is to prove our frameworks work for all integer k and all rational η. Currently, we can only prove it works for some k and η.

For simplicity, our analysis only considers the symmetric sum of terms. The symmetric sum incurs a blow-up exponential on k. Thus the communication complexity of our k-party PSM protocols is $\exp(k) \cdot N^{\frac{k-1}{2}}$. While [BKN18] achieves communication complexity $\text{poly}(k) \cdot N^{\frac{k}{2}}$. Our protocols are less efficient in the domain where $\log N < k$. A possible approach of getting rid of the exponential dependency in k is to break the symmetry. The potential of such an approach is evidenced by the 5-party PSM protocol in Section 3.1.2, which is asymmetric.

There is no clear reason why our framework will not yield more efficient PSM protocols. Can our multi-party framework yield PSM protocols with communication complexity $O_k(N^{\frac{k}{2}-1})$, when k is sufficiently large? Can our 2-party framework yield PSM protocol with communication $O_\eta(N^\eta)$ for some rational $\eta < \frac{1}{2}$? Our technique transfers such questions into some linear systems. Each question has an affirmative answer (for a given k or η) if and only if the corresponding linear system is solvable. We have modified our program to generate and solve these linear systems, but all the systems we have tried are unsolvable. This failure suggests that our new upper bounds might be tight, or are tight for a natural class of PSM protocols.

The question of the communication complexity trade-off for multi-party PSM remains widely open. In our k-party PSM protocol, every party sends $O_k(N^{\frac{k-1}{2}})$ bits. A variant of [FKN94] provides a k-party PSM protocol where the i-th party sends $\tilde{O}_k(N^{i-1})$ bits, whose geometric average is $\tilde{O}_k(N^{\frac{k-1}{2}})$. Should a future work achieves the smooth trade-off between the two, there is little doubt that it will bring us a deep insight into PSM.

Finally, this work belongs to a not-fully-successful attempt at constructing PSM with sub-exponential communication complexity, which is probably the moonshot open problem in the PSM literature.

Publication

This work was in an article published at the Theory of Cryptography Conference 2021, under the following title [AL21]:

Multi-party PSM, revisited: Improved communication and unbalanced communication.

by Léonard Assouline and Tianren Liu.

In Kobbi Nissim and Brent Waters, editors, Theory of Cryptography, pages 194–223

Chapter

4 Weighted ORAM with Applications to SSE

4.1 Introduction

When sensitive data is stored in an untrusted environment, encryption is not enough. The pattern of memory accesses to encrypted data can reveal a great deal about its contents. In some settings, observing the pattern of memory accesses can allow a honest-but-curious host server to fully reconstruct the contents of an encrypted database [GLMP19]; in others, measuring cache misses can enable an attacker to recover secret key material [WHS12]. Untrusted environments where an adversary may be able to observe memory accesses, partially or completely, arise in many common scenarios. These include private information stored in an external cloud service, trusted enclaves running on an untrusted computer, or even public clouds where memory caches are shared across multiple tenants. In all these settings, security requires to hide not only the contents of each data item, but also which item is accessed.

Oblivious RAM (ORAM) protocols provide a powerful tool to fully hide memory access patterns. The notion of ORAM was introduced by Goldreich and Ostrovsky [GO96], motivated by a scenario where a processor accesses untrusted memory. The processor operates in a RAM model of computation: it wishes to access memory words at arbitrary addresses. Naturally, memory words have a fixed size. In line with its historical motivation, ORAM is normally viewed as storing items of fixed size.

However, in many potential applications of ORAM, it is natural to consider items of variable size. Suppose for instance that a client wishes to store private files on an external cloud storage service. Different files may have different sizes; it may also be the case that the size of a file varies with time. This motivates the idea of ORAM for variable-size items.

The Trivial Approach. Of course, it is possible to generically emulate the storage of variable-size files using a memory allocation scheme for fixed-size items. In our case, using an ORAM for items of fixed size B, the most natural approach is to split each file into chunks of size B. Each chunk is then stored as a separate data item (called a block) within the ORAM, on the server side. To retrieve a file, the client simply queries all chunks corresponding to the desired file.

This simple variable-size-to-fixed-size reduction is not always satisfactory. A first issue relates to padding. If $B > 1$, before files can be split into chunks of size B, they must be padded to a multiple of the block size B. If many files are much smaller than the block size, padding becomes expensive. Both motivating applications given below show examples where padding can be prohibitive.

To reduce the cost of padding, it is tempting to reduce the block size B. However,

this has several drawbacks, asymptotically and in practice. Let us start with theoretical ones. For a fixed total amount of data, reducing B increases the ORAM overhead (i.e. the ratio between the communication cost of the ORAM scheme, and the cost of an insecure exchange), since the overhead scales with the number of blocks. For example, an ORAM storing N blocks of size 1 typically has an overhead in polylog N; whereas with N/B blocks of size B, the overhead becomes polylog(N/B). In later applications such as length-hiding ORAM, or ZeroSSE (Section 4.6.2), B can be very large, which makes the difference significant. In theory, larger block sizes are also preferable: for instance, Path ORAM achieves optimal $\mathcal{O}(\log n)$ overhead if the block size is $\Omega(\lambda^2)$ bits, but its overhead is $\mathcal{O}(\log^2 n)$ if the block size is $\Theta(\lambda)$ bits (where λ is the security parameter).

In practice, setting the block size to be very small, say a single memory word of 128 bits, has a deeper impact that is easy to overlook, but much more impactful in practice that the asymptotic difference above. Modern computers can only fetch memory from disk at the granularity level of a page, typically 4kB. This is enforced at all levels: by the operating system, in caches, and at the physical disk layer (for both HDDs and SSDs). When fetching many 128-bit words at random locations in the ORAM scheme, the server actually fetches the entire page for each. In each of those pages, only a fraction 1/256 of the data in the page is actually useful (128 bits out of 4kB). This results in very poor I/O efficiency, which correlates directly with disk throughput [BBF+21]. The issue is easy to overlook because it is not reflected in the simple Random-Access Machine model of computation that is used to compute asymptotics, where all memory accesses have unit cost. But it has a very large impact in practice. This is well-known in Searchable Symmetric Encryption (SSE) literature, where an entire branch of the area studies memory efficiency [ANSS16, MM17, BBF+21, MR22]. In ORAM literature, the PHANTOM implementation of Path ORAM uses blocks of size one page, likely for the same reason [MLS+13]. Reading many tiny items at random memory locations is extremely inefficient, losing a factor up to B in throughput (when the bottleneck in throughput does not come from communication bandwidth).

If one thinks of storing entire documents in the context of a private online storage service, having many documents much smaller than the page size is rather unlikely. But in other applications, it is quite realistic. A case in point is the use of ORAM for Searchable Symmetric Encryption (SSE).

Motivating Application 1: Searchable Symmetric Encryption (SSE). The goal of SSE is to enable a client to outsource the storage of an encrypted database to an untrusted server, while being able to securely search the data. At minimum, the client is able to issue search queries asking for all entries that match a given keyword. To realize this functionality, for efficiency reasons, virtually all modern SSE constructions rely on a *reverse index*. The reverse index records, for each keyword, the list of identifiers of entries that match the keyword.

The majority of SSE solutions accept to leak the *search pattern* and *access pattern* of the client: that is, they leak to the server the repetition of queries, and the identifiers of documents matched by each query. This allows those constructions to trade off privacy for efficiency and scalability. Nevertheless, revealing access patterns to the server can be quite damaging, and has led to a number of attacks [CGPR15, GLMP19]. Those attacks have in turn motivated SSE approaches that rely on ORAM [GMP16, KMO18].

The most natural way to avoid leaking the search pattern is to store the reverse index in an ORAM. In that scenario, the "files" to be stored on an ORAM are actually the list of matches for a given keyword in an SSE scheme. For some databases, there may

be many keywords that uniquely identify a file, or that match only a few files. In other words, there may be many lists much smaller than the block size B of the ORAM.

In practice, this is actually a major roadblock. As argued earlier, it is desirable to have a relatively large ORAM block size, at least one memory page. On the other hand, the identifier of an entry can be set to 64 bits, or even less. This means that a single ORAM block is 512 times larger than the minimal list size. If, say, many keywords match less than 10 entries in the database, padding those lists to the block size blows up their storage by a factor more than 50. More generally, if we set p to be the page size, measured in number of identifiers per page, then padding means that server storage grows at least in $\mathcal{O}(pN)$, where N is the size of the plaintext reverse index; whereas we would like to achieve linear storage $\mathcal{O}(N)$. Of course, with $p = 512$ as earlier, the practical difference is quite large.

Addressing that problem is not an easy task. In SSE literature, avoiding the cost of padding to the page size has been the focus of several recent works [BBF+21, MR22]. Those works have motivated the creation of *weighted* memory allocation schemes, that can accommodate items of variable size, including weighted cuckoo hashing [BBF+21], and weighted two-choice allocation [MR22]. However, there is no weighted ORAM. This means that in order to use ORAM with SSE, current options are either to choose a block size much smaller than the page size, or to suffer a prohibitive padding overhead for some data distributions—both of which are undesirable.

Motivating Application 2: Length-Hiding ORAM. Let us go back to the scenario where a client wishes to store private files of various sizes on a honest-but-curious cloud server. As noted earlier, the simplest way to hide access patterns is to store the files in an ORAM. Each file is split into chunks of size B, and each chunk is stored in a separate ORAM block. In order to fetch a file, the client queries all chunks of the file to the ORAM. When a file is queried, the only information leaked to the server is the number of chunks of the file.

In some settings, even that much information may be too much information. For instance, the number of chunks of a file might be enough information to uniquely identify the file [CGPR15]. In that case, repeated accesses to the file are leaked to the server. This reveals the access pattern of the client to the files, defeating the purpose of ORAM. More subtly, the length of answers to certain types of database queries can be enough to infer the entire contents of encrypted database [GLMP18]. Traffic analysis attacks are another example of using length information to infer sensitive data [DCRS12]. Attacks that exploit length information can be particularly insidious, because traditional encryption does not attempt to hide length.

If leaking the lengths of the files is judged to be too damaging, the client may wish to use additional mechanisms to protect their privacy. Going back to our running example about private file storage, the simplest and most secure protection is to mandate that, whenever a file is accessed, the client should query as many chunks as the size of the *largest* file. In that case, only the number of chunks of the largest file is leaked to the server—or an upper bound on that number.

Let N be the total size of the files to be stored on the remote server. Let B be the ORAM block size, and let U be an upper bound on the size of the largest file (all quantities are counted in number of memory words). The overhead of ORAM constructions typically scales in $\mathsf{Polylog}(n)$, where n is the number of blocks stored in the ORAM. Setting aside padding issues for a moment, with block size B, we have $n = N/B$. In order to minimize the overhead, it would be attractive to simply set $B = U$. But here again, we would run

into padding issues: most files might be much smaller than the largest file. The optimal solution would be a *weighted* ORAM able to accommodate files of arbitrary size up to U, with an overhead $\mathsf{Polylog}(N/U)$, or optimally, $\log(N/U)$.

4.1.1 Our Contributions

The discussion so far leads to the following question: can we devise a *weighted* ORAM—that is, an ORAM that natively accommodates items of variable size? Beside the motivating applications given in the introduction, the existence of weighted ORAM may be viewed as a natural question: it fits within a long line of work on weighted allocation mechanisms, both within and outside cryptography, such as [TW07, BFHM08, ANSS16, BBF+21, MR22].

We answer the previous question in the affirmative, and build a weighted ORAM. Our construction naturally handles not only items of different sizes, but items whose size varies with time, without the need for padding. To state the result precisely, let us introduce some notation. In the remainder, an atomic item stored within the ORAM is called a *block*. Let B denote an upper bound on the block size. Unlike traditional ORAMs, blocks can take any size in $[1, B]$. We will sometimes call the size of a block its *weight*. Let $w_i \in [1, B]$ denote the weight of the i-th block. Let m be the total number of blocks. Let N be an upper bound on the *total weight* $\sum_{i \leq m} w_i$. We want to build an ORAM that can accommodate *any* vector $\mathbf{w} = (w_i)_{i \leq m}$ of weights, as long as the following two conditions are fulfilled.

Condition 1. Every block w_i has weight at most B;

Condition 2. The total weight $\sum w_i$ is at most N.

For ease of exposition, we will assume that the number of blocks m is fixed, but our constructions can be easily adapted to a variable number of blocks, so long as the previous two conditions continue to hold. The parameters of our ORAM constructions will depend *only* on B and N; crucially, they do not depend on the distribution of the weight vector $\mathbf{w}$.

The interface of our weighted ORAMs is identical to standard ORAM: to retrieve a block, the client queries an identifier of the block (e.g. a virtual memory address). When writing a block, the client also inputs new data for the block. This data need not be of the same size as the data originally associated to the block identifier. The client can freely change the size of a block with every access, as long as Conditions 1 and 2 remain true.

As our main contribution, we build a weighted ORAM in the sense given above. In fact, we show a significantly stronger result. Many standard Tree-based ORAM algorithms admit a natural extension to handle blocks of variable size: at setup, the ORAM is dimensioned as if to accommodate N/B blocks of size B, but instead receives an arbitrary number of blocks of variable size bounded by B, with total size N. These blocks are read and written through the ORAM in essentially the same way as in the original, fixed-block size Tree ORAM, except for minor alterations to reflect the fact that blocks do not have the same size.

The main obstacle with that approach is technical. While Path ORAM is one of the most attractive solutions for practical Tree ORAM [SvS+13b], its correctness proof is notoriously difficult—prompting the introduction of Simple ORAM as a less efficient variant that allows for a simpler correctness proof [CP13]. Our main result is to show

that the natural weighted extensions of several existing Tree ORAM schemes, including Path ORAM and Simple ORAM, are in fact correct. For that purpose, we introduce a general framework: we prove that as long as a Tree ORAM fulfills a certain structural property, its weighted extension preserves correctness. The centerpiece of the proof is a Schur-convexity argument, which ultimately reduces the correctness of the weighted extension to that of the original ORAM. (An overview of the proof argument is given in Section 4.4.3, before the formal proof.) Practical experiments show that our weighted ORAM construction behaves in line with the analysis.

As an application of weighted ORAM, we build two SSE schemes, ZeroSSE and BlockSSE. Unlike most of SSE literature, both constructions completely hide access patterns. To our knowledge, ZeroSSE is the only construction that leaks neither the access pattern nor the size of retrieved objects, with full correctness. (The only other construction that we are aware of, in [KMO18], pays the price of having a non-negligible correctness failure probability.) BlockSSE hides access pattern, but not the size of retrieved objects. To our knowledge, it is the only ORAM-based SSE with worst-case server storage $\mathcal{O}(N)$, rather than $\mathcal{O}(BN)$, where B is the ORAM block size.

Our main result builds on Tree ORAMs, because of their higher practical efficiency compared to hierarchical ORAMs. This makes tree-based construction currently more attractive for applications such as SSE. Nevertheless, it is worth remarking that the position map of a weighted Tree-based ORAM, as we have built, has blocks of fixed size. Hence, it can be stored using any standard ORAM scheme, not necessarily tree-based. In particular, from a more theoretical perspective, the position map can be stored using an optimal ORAM with logarithmic overhead, following the groundbreaking result of Asharov *et al.* [AKL+20]. This results in a weighted ORAM with logarithmic overhead. The question of building weighted hierarchical ORAM schemes is briefly discussed in Appendix B.4.

As another direct application of our construction, setting the block size B of our weighted ORAM to be equal to an upper-bound bound U on the size of the largest item to be stored in the ORAM, we immediately deduce from our construction weighted Path ORAM a length-hiding ORAM with communication overhead $\mathcal{O}(\log^2(N/U))$. If we use an optimal (standard) ORAM for the position map, as indicated above, we obtain a length-hiding ORAM with communication overhead $\log(N/U)$. This overhead is optimal, since such a goal includes as a special case the setting where all blocks have size U, and is thus subject to known ORAM lower bounds for an ORAM storing N/U blocks [GO96, LN18].

4.1.2 Related work

While there is a rich literature on ORAM, surprisingly little of it deals with objects of variable size. To the best of our knowledge, only two articles mention this subdomain of ORAM.

In [RAC16], Roche *et al.* present the first ORAM that stores objects of variable size. Their goal is to build a remote data structure that satisfies the security requirements of ORAM, and in addition allows for secure deletion of items and history independence. In other words, in the case of a *total leakage of the structure* (such an event is referred to as a *catastrophic attack*):

- Items that have been deleted by the client can never be recovered through leaked data.

- The internal structure does not reveal information about which elements were last accessed.

The data structure is built on top of a weighted ORAM. However, their construction for such an ORAM is limited: obliviousness and correctness (i.e. the client-side stash overflows with negligible probability) can be proven only if the size of the blocks follow a geometric probability distribution. In comparison, although we assume that block sizes are bounded by B, we do not need to assume anything on the distribution of block sizes. In more detail, there are two limitations to the assumptions of [RAC16]. First, many common distributions are not upper-bounded by a geometric distribution, for instance Zipf distributions. Second and more fundamentally, the ORAM user has no reason in general to pick item sizes independently, or to pick them from the same distribution. The construction of [RAC16] was designed with a specific use case in mind; its applicability beyond that use case is limited.

Another construction of ORAM for objects of variable size may be found in [LHL+18]. Their construction is also based on Tree ORAMs. The idea is to allow block size to be equal to a multiple of some value s (padding up to a multiple if needed), and to store all "splinters" of size s of a block along the same path from root to leaf. This construction has the strong requirement of a trusted proxy that shuffles blocks during certain operations. Moreover, the construction is flawed: cf. Appendix B.3.

4.1.3 Organization of the chapter

In Section 4.3 we recall the definitions of ORAM, SSE, and Schur convexity, a tool we will use in our proof. Section 4.4 is where we state our generic criterion for converting a standard ORAM into one that supports objects of variable size, and prove our main result. Concrete examples of known ORAM schemes that we can turn into weighted ORAM are presented in Section 4.5. An alternative generic construction that converts any ORAM into a weighted ORAM at the cost of a small blowup is presented in Appendix B.2. We discuss applications to the field of SSE in Section 4.6.

4.2 General Preliminaries

Throughout this work, memory size will be counted as a number of *memory words*. It is assumed that a memory word is large enough to store any address in memory. In practical applications, one may think of 64-bit or 128-bit words. Algorithms will be considered in the RAM model, where accessing an arbitrary memory word costs $O(1)$ operations.

The security parameter is denoted by λ. A quantity is said to be *negligible*, denoted $\mathrm{negl}(\lambda)$, if it is $O(\lambda^{-c})$ for every constant c. A probability is said to be *overwhelming* if it is $1 - \mathrm{negl}(\lambda)$. It is always assumed that the number of blocks n stored in the ORAM satisfies $n \geq \lambda$, so that any quantity that is $\mathrm{negl}(n)$ is also $\mathrm{negl}(\lambda)$.

When an algorithm A with input x is probabilistic, we may sometimes explicitly write the random coins used by A as an input of A, separated by a semicolon, as in $A(x; r)$.

4.2.1 Majorization and Schur Convexity

Given a vector $\mathbf{v}$ in $\mathbb{R}^m$, we denote by $\mathbf{v}^{\downarrow} \in \mathbb{R}^m$ the vector with the same components, sorted in decreasing order.

Definition 3 (Majorization order)**.** Let $\mathbf{v}$, $\mathbf{w}$ be two vectors in $\mathbb{R}^m$ such that $\sum_{i=1}^m v_i = \sum_{i=1}^m w_i$. The vector $\mathbf{w}$ is said to *majorize* $\mathbf{v}$, written $\mathbf{v} \prec \mathbf{w}$, if:

$$\forall k \in [1, m], \quad \sum_{i=1}^{k} v_i^{\downarrow} \leq \sum_{i=1}^{k} w_i^{\downarrow}.$$

Definition 4 (Schur convexity)**.** Let $f : \mathbb{R}^m \mapsto \mathbb{R}$. The map f is said to be *Schur-convex* if it is non-decreasing for the majorization order. That is, for any two vectors $\mathbf{v}$, $\mathbf{w}$ with $\sum_{i=1}^m v_i = \sum_{i=1}^m w_i$,

$$\mathbf{v} \prec \mathbf{w} \Rightarrow f(\mathbf{v}) \leq f(\mathbf{w}).$$

Definition 5 (Convexity)**.** Let $f : \mathbb{R}^m \mapsto \mathbb{R}$. The map f is said to be *convex* if for any two vectors $\mathbf{v}$, $\mathbf{w}$ in $\mathbb{R}^m$, and any α in $[0, 1] \subset \mathbb{R}$, it holds that:

$$f(\alpha\mathbf{v} + (1 - \alpha)\mathbf{w}) \leq \alpha f(\mathbf{v}) + (1 - \alpha) f(\mathbf{w}).$$

Definition 6 (Symmetry)**.** Let $f : \mathbb{R}^m \mapsto \mathbb{R}$. The map f is said to be *symmetric* if for any vector $\mathbf{v} \in \mathbb{R}^m$, and any permutation matrix P over m elements, $f(\mathbf{v}) = f(P\mathbf{v})$.

The link between convexity and Schur convexity is visible in the next lemma.

Lemma 4.2.1. *Let $f : \mathbb{R}^m \mapsto \mathbb{R}$. If f is symmetric and convex, then it is Schur-convex.*

We refer the reader to [MOA79] for a detailed presentation of the theory of majorization, including a proof of Lemma 4.2.1.

4.3 ORAM Preliminaries

4.3.1 Weighted Oblivious RAM

A weighted ORAM, also written wORAM, is a pair of client-server protocols $(\texttt{Setup}, \texttt{Access})$, defined as follows.

- $\texttt{Setup}(N, B, \mathsf{D})$ takes as input an upper bound N on the total amount of data, a block size B, and a set D of pairs of the form $(\mathsf{a}_i, \mathsf{data}_i)$, where the a_i's are pairwise distinct *addresses*, and data_i is arbitrary data of size $|\mathsf{data}_i| \in [1, B]$. `Setup` outputs an initial client state and initial server state.

- $\texttt{Access}(\mathsf{op}, \mathsf{a}, \mathsf{data})$ takes as input an operation $\mathsf{op} \in \{\text{read}, \text{write}\}$, an address a, and some data data of size $|\mathsf{data}| \in [1, B]$. If $\mathsf{op} = \text{read}$, `Access` outputs to the client the data last written to address a. If $\mathsf{op} = \text{write}$, `Access` replaces the data written at address a by data. `Access` may also update the client and server states.

We say that $\texttt{Setup}(N, B, \mathsf{D})$ is *legal* if the total amount of data in D (*i.e.* the sum of the sizes of the data_i's) is at most N. Likewise, we say that $\texttt{Access}(\mathsf{op}, \mathsf{a}, \mathsf{data})$ is *legal* if address a was defined during setup, and in the case that $\mathsf{op} = \text{write}$, if $|\mathsf{data}| \in [1, B]$, and the total amount of data contained in the database after replacing the data at address a by data remains of size at most N. On the other hand, it is *not* required that the size of data matches the size of the data previously written at a.

Definition 7 (Correctness)**.** A wORAM scheme is said to be *correct* if, given a legal setup and any sequence of legal access operations, a read access at address a outputs the data last written to address a, except with negligible probability.

Definition 8 (Security)**.** A wORAM scheme is said to be *secure* if, given any two legal sequences of operations of the same length $(\mathtt{Setup}(N, B, \mathsf{D}), \mathtt{Access}(\mathsf{op}_1, \mathsf{a}_1, \mathsf{data}_1), \ldots, \mathtt{Access}(\mathsf{op}_k, \mathsf{a}_k, \mathsf{data}_k))$ and $(\mathtt{Setup}(N, B, \mathsf{D}'), \mathtt{Access}(\mathsf{op}'_1, \mathsf{a}'_1, \mathsf{data}'_1), \ldots, \mathtt{Access}(\mathsf{op}'_k, \mathsf{a}'_k, \mathsf{data}'_k))$, the views of the server arising from each sequence are computationally indistinguishable.

A few remarks are in order. First, although we have defined `Setup` and `Access` as general client-server protocols, it is common in ORAM to require that the server behave like a passive memory, allowing only read and write accesses. That is, the client only ever asks the server to read or write specific data at a specific address. The server performs no computation of its own. Although this is not required in the previous definition, the wORAM schemes in this work are in that model.

Second, it is assumed that the contents of all memory locations on the server are encrypted using IND-CCA encryption, with a key known only to the client. Whenever the client accesses a memory location, they can reencrypt the data at that location, so that the server cannot learn the contents of any memory location, or whether it was changed during the access. As a result, the only way the server can infer information is by observing which locations the client queries in server memory. That is why the security definition of wORAM (following that of ORAM) focuses only on memory locations.

Finally, note that standard ORAM is the special case of wORAM where all addresses store data of the same size B. If the total amount of data is N, then such a scheme contains $n = N/B$ blocks. Throughout this chapter, the letter N is reserved for the total amount of data. In the case of standard ORAM, the letter n is used for the number of blocks, with $n = N/B$.

4.3.2 Tree ORAM

We build wORAM by altering standard ORAM schemes following the *Tree ORAM* paradigm. In this section, we provide a high-level algorithmic view of that paradigm. That view is purposefully designed to accomodate several existing Tree ORAM schemes. It will also lay the groundwork for the construction of wORAM in the next section.

Existing Tree ORAM schemes are standard ORAMs, designed to store items of fixed size. In a Tree ORAM, to store n items of size B, the server creates a full binary tree with n leaves. (From now on, we assume n is a power of 2, increasing to the next power of 2 if necessary.) Throughout the chapter, the root of the tree is viewed as being at the top, and leaves as being at the bottom of the tree. Given a leaf l of the tree, the path from the root to the leaf l is denoted by $\mathcal{P}(l)$.

Each node of the tree, also called a *bucket*, can store up to Z data blocks of size B. Nodes are always padded to be of size ZB before being stored (encrypted) on the server.

In addition to the tree, the server may also store a *stash*, which may contain additional data blocks that could not fit in the tree. In the remainder, we view the stash as a special node directly above the root. This is relevant in two situations. First, there may be cases where a node is full (*i.e* it contains Z items), and where additional items need to be pushed to the parent node; if this happens at the root level, overflowing items are pushed to the stash. Second, whenever we consider the path $\mathcal{P}(l)$ from some leaf l to the root in the tree, we implicitly (and slightly abusively) also consider the stash to be part

of the path. The stash is always padded to some upper bound R, before being stored (encrypted) on the server.

To each item with address a is associated a leaf of the tree $\texttt{pos}(a)$. The array mapping each address a to the corresponding leaf $\texttt{pos}(a)$ is called a *position map*. For now, we will assume the position map is stored by the client. By design, Tree ORAMs maintain the following invariant at all times: the item at address a is stored in one of the nodes on the path $\mathcal{P}(\texttt{pos}(a))$ from the root to the leaf $\texttt{pos}(a)$ (including the stash, as noted earlier).

During setup, each item with address a is stored in the leaf $\texttt{pos}(a)$; or if it is full, in the lowest parent of $\texttt{pos}(a)$ that is not yet full. To access item a, the client retrieves $\texttt{pos}(a)$ from the position map, then reads the path $\mathcal{P}(\texttt{pos}(a))$ on the server. Thanks to the invariant, that path contains the item a. Item a is then assigned a new uniformly random leaf. Finally, a special *eviction* procedure is called, which re-inserts item a somewhere on the path to its newly assigned leaf, and may also move other items.

Pseudo-code for the `Access` procedure is given in Algorithm 1, with additional parameters Z (the number of blocks per bucket, specifided by the Tree ORAM scheme; to reflect the fact that Z is an internal parameter of the ORAM construction, and not part of its interface, it is written between brackets), and random coins r. It makes use of the following subroutines:

- `ReadBucket(`*bucket*`)` retrieves a set of pairs $(a_i, \texttt{data}_i)$ from the tree node *bucket*.
- `RemoveBlock(`*bucket*, *a*`)` removes the item with address a from the tree node *bucket*.
- `ChooseEvictionPath` outputs a path for eviction, which differs depending on the specific Tree ORAM scheme.

Algorithm 1 Access algorithm of a Tree-ORAM.

Access$[Z; r]$(op, a, *newdata*):

1: *leaf* $\leftarrow$ $\texttt{pos}[a]$
2: $\texttt{pos}[a] \leftarrow$ uniformly random leaf
3: **for** *bucket* in $\mathcal{P}(leaf)$ **do**
4: **if** $(a, \texttt{data}) \in$ `ReadBucket`$(bucket)$ **then**
5: `RemoveBlock`$(bucket, a)$
6: **if** op = write **then**
7: data = newdata
8: $stash \leftarrow stash \cup \{(a, data)\}$
9: $path \leftarrow$ `ChooseEvictionPath`$(leaf)$
10: Evict$[Z; r](path)$
11: **return** data

Pseudo-code for the `Evict` procedure is given in Algorithm 2, with additional parameters Z (the number of blocks per bucket, specifided by the Tree ORAM scheme), and random coins r. It makes use of the following subroutines:

- `Size`(X) returns the number of items $|X|$ in X.
- `ChooseNextBlock`$(stash, bucket, path)$ pops an item from the stash, to be stored in the bucket, or outputs $\perp$.

Algorithm 2 Generic eviction algorithm.

Evict$[Z; r](path)$:

```
1:  Move all blocks in path to the stash
2:  for bucket in path do
3:      X ← ∅
4:      while Size(X) < Z do
5:          block ← ChooseNextBlock(stash, bucket, path)
6:          if block = ⊥ then
7:              break
8:          else
9:              X ← X ∪ {block}
10:     WriteBucket(bucket, X, Z)
11: return
```

- `WriteBucket`(*bucket*,X,Z) writes the items in X to the node *bucket*, padding the node to size Z if needed.

We will discuss in Section 4.5 how several existing Tree ORAM schemes are captured by the above paradigm.

Correctness of Tree ORAM

Since Tree ORAM is a special case of ORAM, the correctness definition remains the same (Definition 7). However, because of the specificities of Tree ORAM, it can be reformulated in a more convenient manner. That is, the only correctness failure that can occur in a Tree ORAM scheme is that the stash overflows. (The reader familiar with Tree ORAM may object that some Tree ORAM schemes do not use a stash; that case will be handled in Section 4.5.)

Recall that the stash is always padded to size RB, *i.e.* it can store up to R items. Hence, correctness amounts to the following statement: at the outcome of any sequence of legal accesses $(\texttt{Setup}, \texttt{Access}_1, \ldots, \texttt{Access}_k)$, it holds that

$$\Pr[\texttt{Size}(stash) > R] = \mathrm{negl}(\lambda).$$

4.3.3 The ∞-ORAM Model

Consider a Tree ORAM instantiation $ORAM^Z \leftarrow \texttt{Setup}[Z](N, B, \mathsf{D})$, with bucket capacity Z. If $\mathbf{s}$ is a sequence of accesses, we call $st(ORAM^Z[\mathbf{s}])$ the stash usage, that is, the number of items in the stash at the outcome of the accesses.

In Path ORAM and many Tree ORAM schemes derived from it, the proof of correctness follows similar steps:

- Consider an *infinite* ORAM structure $ORAM^\infty$, which is the same protocol, except buckets have infinite capacity (that is, $Z = \infty$).

- Define a post-processing algorithm G_Z that moves items in the tree produced by running $ORAM^\infty$ (arranging in particular that each tree node contains at most Z items). Denote the stash usage of the post-processed ∞-ORAM by $st^Z(ORAM^\infty[\mathbf{s}])$.

- Prove that $st(ORAM^Z[\mathbf{s}]) = st^Z(ORAM^\infty[\mathbf{s}])$ when using the same random coins on both sides.
- Prove that $\Pr[st^Z(ORAM^\infty[\mathbf{s}]) > R] = \text{negl}(N)$.

The last two points imply that $\Pr[st(ORAM^Z[\mathbf{s}]) > R] = \text{negl}(N)$, *i.e.* the original ORAM scheme is correct. We say that such a protocol admits a *proof via infinite ORAM*.

4.4 Generic Construction of wORAM from Tree ORAM

Our goal is to give a generic way to transform an existing standard Tree ORAM design into one that supports blocks of variable size. The transformation presented in this section is perhaps the most natural transformation one could imagine for that purpose. The main challenge is not the transformation itself, but rather proving that it preserves correctness.

Still, there is one aspect of the transformation whose necessity may not be immediately apparent. Namely, if the original ORAM has bucket capacity Z, then its weighted version will have bucket capacity $Z + 1$. This change is not necessary for the main theorem of this section to hold. However, it will be necessary when applying the main theorem to specific ORAMs, notably Path ORAM: see the discussion at the end of Section 4.5.2. In practice, experiments in Section 4.4.4 suggest that this increase in bucket capacity can be heuristically dispensed with.

4.4.1 The `Weighted` Transform

We define a general transform `Weighted` that takes as input a standard Tree ORAM scheme $\mathsf{ORAM}^Z = (\texttt{Setup}, \texttt{Access})$ following the framework of Section 4.3.2, and outputs a weighted ORAM scheme $\texttt{Weighted}(\mathsf{ORAM}^Z) = \mathsf{ORAM}^{*Z} = (\texttt{Setup}^*, \texttt{Access}^*)$.

Let us first consider the setup. We say that the starting scheme ORAM^Z has a *regular* setup if its setup procedure is equivalent to creating an empty tree with all items in the stash, then doing repeated evictions towards every leaf in the tree from left to right. Here, by "equivalent" we mean that the output of this process and the output of the normal setup process are identically distributed. In our main theorem, we will require that the starting Tree ORAM ORAM^Z has a regular setup. Although that notion of regularity is non-standard, it has the benefit that the behavior of the setup process can be deduced from that of the eviction process. For our purpose, this means it will be enough to explain how to transform the eviction process to handle blocks of variable size.

ORAM^{*Z} is defined in the following way, making only minimal modifications to ORAM^Z to handle items of variable size.

- $\texttt{Setup}^*(N, B, \mathsf{D})$ initializes a tree with N leaves, whose nodes can each hold data of size $(Z + 1)B$, and a stash of the same size RB as the standard instance ORAM^Z. It initializes a position map where each address a in D is mapped to a uniformly random leaf. Finally, it performs a *regular* setup: that is, all items in D are placed in the stash, and the $\texttt{Evict}^*$ procedure is called on the path from the root to each leaf, from left to right.
- $\texttt{Access}^*$ is identical to `Access`, except that it calls the modified subroutine $\texttt{Evict}^*$.
- $\texttt{Evict}^*$ is identical to `Evict`, except that it calls the modified subroutines $\texttt{Size}^*$ and $\texttt{WriteBucket}^*$.

- $\texttt{Size}^*(X)$ returns the sum of the sizes of all items in X divided by B, instead of the number of items in X.

- $\texttt{WriteBucket}^*(\mathit{bucket}, X, Z)$ still writes the items in X to node bucket, the only difference is that it pads the bucket to size $Z+1$ instead of Z.

4.4.2 Suitable Tree ORAM Schemes

For the `Weighted` transform to preserve correctness, the original standard ORAM scheme must satisfy certain conditions. This section serves to define those conditions.

Given a sequence of accesses $\mathbf{s}$, some fixed random coins r used during those accesses, and a subset S of nodes in an ∞-ORAM scheme ORAM^*, define the *usage* of S, written $u^S(\mathsf{ORAM}^{*\infty}[\mathbf{s}; r])$, to be the total number of items assigned to the nodes in S. For a wORAM scheme, the usage of S is defined to be the total size of the items assigned to nodes in S, divided by the block size B.

As discussed in Section 4.3.2, a correctness failure for a Tree ORAM scheme ORAM occurs if and only if, at the outcome of a series of accesses $\mathbf{s}$ with random coins r, the stash receives strictly more than R elements. Using the notation from Section 4.3.3, this translates to $st(ORAM^Z[\mathbf{s}; r]) > R$. We say that a subset S of nodes *witnesses* the failure if, in the corresponding ∞-ORAM scheme ORAM^* when performing the same sequence of accesses using the same random coins (*viz.* the choices of fresh uniformly random leaves for the position of each accessed item remain the same), $u^S(\mathsf{ORAM}^{*\infty}[\mathbf{s}; r]) > |S| \cdot Z + R$. Intuitively, since the nodes in S can store at most $|S| \cdot Z$ items, it is clear that more than R items must be reassigned to the stash in the original ORAM, which breaks correctness: that is why we say that S witnesses the failure.

Definition 9 (F $\Rightarrow$ W, W $\Rightarrow$ F)**.** We say that ORAM satisfies the F $\Rightarrow$ W property (read: "failure implies witness") with respect to a set $\mathscr{S}$ of subset of nodes, iff for all access sequences $\mathbf{s}$ and all choices of random coins r, $st(ORAM^Z[\mathbf{s}; r]) > R$ implies $\exists S \in \mathscr{S}, u^S(\mathsf{ORAM}^{*\infty}[\mathbf{s}; r]) > |S| \cdot Z + R$. We say that ORAM satisfies the W $\Rightarrow$ F (read: "witness implies failure") property if the converse is true.

Moreover, we say that ORAM satisfies the F $\Rightarrow$ W (resp. W $\Rightarrow$ F) property *via union bound* if the scheme also satisfies that $\sum_{S \in \mathscr{S}} \Pr[u^S(\mathsf{ORAM}^{*\infty}_L[\mathbf{s}]) > |S| \cdot Z + R] = \mathrm{negl}(\lambda)$. Informally, this means the statement "the probability that a failure witness exists is negligible" can be proved via a union bound over all possible witnesses $S \in \mathscr{S}$.

The definitions remain the same for a wORAM scheme. In particular, for a wORAM scheme ORAM^*, a subset S witnesses a failure if $u^S(\mathsf{ORAM}^{*\infty}_L[\mathbf{s}]) > |S| \cdot Z + R$ (and not $|S| \cdot (Z+1) + R$, even though our construction of wORAM uses buckets of size $(Z+1)B$).

Definition 10 (Suitable Tree ORAM)**.** We say that a Tree ORAM scheme is *suitable* if it satisfies the following conditions.

1. It admits a proof via infinite ORAM. That is, for all access sequence $\mathbf{s}$ and random coins r, $st(ORAM^Z[\mathbf{s}; r]) > R$ iff $st^Z(ORAM^\infty[\mathbf{s}; r]) > R$.

2. ORAM satisfies the W $\Rightarrow$ F property with respect to some set $\mathscr{S}$, via union bound.

3. $\texttt{Weighted}(\mathsf{ORAM})$ satisfies the F $\Rightarrow$ W property with respect to the same $\mathscr{S}$.

4. ORAM allows *free evictions*. That is, if the client is allowed to trigger evictions on uniformly random leaves at will during a sequence of accesses, correctness still holds.

Requiring all those properties may seem demanding, but they naturally hold for several existing Tree ORAM schemes, including Path ORAM and Simple ORAM. This will be shown in more detail in Section 4.5. Intuitively, this is because many schemes admit a proof via infinite ORAM, either explicitly (in the case of Path ORAM), or trivially (in the case of Simple ORAM, where the ORAM and its infinite variant are identical up to correctness failures). Similarly, the F $\Rightarrow$ W property is either already known, or trivial; and the *free eviction* property is immediate. The only property that requires some care is to show that `Weighted`(ORAM) satisfies the F $\Rightarrow$ W property. However, it is much more tractable than trying to analyze the correctness of a wORAM scheme directly.

4.4.3 Main Result

Theorem 4.4.1 (Main Theorem)**.** *Let* ORAM *be any suitable Tree ORAM scheme. If* ORAM *is a correct ORAM scheme, then* `Weighted`(ORAM) *is a correct wORAM scheme.*

Before diving into the proof proper, we sketch the underlying approach. Because of the F $\Rightarrow$ W and W $\Rightarrow$ F properties required by the suitability assumption, showing the wORAM scheme is correct essentially amounts to showing that no set $S \in \mathscr{S}$ witnesses a failure (except with negligible probability). We wish to analyze the function that maps the sizes of items to the usage of S (*i.e.* the sum of sizes of all items in S). Ultimately, we want to show that the probability that the usage of S exceeds $|S| \cdot Z + R$ is negligible, regardless of item sizes.

The proof strategy is to upper-bound the previous probability by a Schur-convex function (Section 4.2.1), and show that this function is negligible. The idea behind this strategy is that if a function of item sizes is Schur-convex, then in order to upper bound the function for *all* possible vectors of item sizes, it is enough to upper-bound it for a set of maximal vectors for the majorization order. Luckily, due to the requirement that item are of size at most B, and that the sum of items sizes is at most NB, a single weight vector majorizes all others, namely the vector $(B, \ldots, B, 0, \ldots, 0)$. Hence, it is enough to upper-bound the function for that specific vector. This is quite convenient, because that weight vector essentially amounts to having all items be of the same size, which reduces to the correctness of the original (unweighted) ORAM instance.

To instantiate the proof idea outlined above, the core argument is to show that there exists a suitable Schur-convex function. This is done via a first-moment argument (Lemma 4.4.2), which allows us to work with expectations instead of probabilities. Expectations are much better behaved with respect to convexity (due to the linearity of expectation). Eventually, we get the upper bound to metamorphosize into a suitable Schur-convex function (in the proof, this is the map $\mathbf{w} \mapsto \mathbb{E}[X_{\mathbf{s},L,S}(\mathbf{w})]$), and show it is convex essentially by showing that it is structured as a composition of convex maps. Using Lemma 4.2.1, we deduce that it is Schur-convex.

Proof. We start with a small self-contained lemma.

Lemma 4.4.2. *Let X be an integral random variable defined over $[0, t] \subset \mathbb{N}$, with $t \in$ $\mathsf{Poly}(\lambda)$. Then $\Pr[X > R] = \mathrm{negl}(\lambda)$ if and only if $\mathbb{E}(\max(0, X - R)) = \mathrm{negl}(\lambda)$.*

Proof. Recall that the expectation of a positive integral variable Y can be written as:

$$\mathbb{E}(Y) = \sum_{i \geq 0} \Pr[X > i].$$

As a corollary, for any integral variable Y satisfying $0 \leq Y \leq t$:

$$\Pr[Y > 0] \leq \mathbb{E}(Y) \leq t \Pr[Y > 0]. \tag{4.1}$$

Observe that the event $X > R$ is equivalent to $\max(0, X - R) > 0$. Using that observation, and applying (4.1) to the variable $Y = \max(0, X - R)$, we get:

$$\Pr[X > R] \leq \mathbb{E}(\max(0, X - R)) \leq t \Pr[X > R].$$

Since $t \in \mathsf{Poly}(\lambda)$, we are done. ■

Let ORAM be a suitable and correct Tree ORAM scheme. Let $\mathsf{ORAM}^* \leftarrow \mathtt{Weighted}(\mathsf{ORAM})$. Let $\mathbf{s}$ be a legal sequence of accesses for ORAM^*. We need to show that $\Pr[st(\mathsf{ORAM}^*[\mathbf{s}]) > R] = \mathrm{negl}(\lambda)$.

Since ORAM satisfies the F $\Rightarrow$ W property with respect to some set $\mathscr{S}$, it suffices to show that the probability that there exists $S \in \mathscr{S}$ witnessing the failure is negligible, *i.e.* $\Pr[\exists S \in \mathscr{S}, u^S(\mathsf{ORAM}^{*\infty}_L[\mathbf{s}]) > |S| \cdot Z + R]$ is negligible.

Let us fix $S \in \mathscr{S}$. We want to show that $\Pr[u^S(\mathsf{ORAM}^{*\infty}_L[\mathbf{s}]) > |S| \cdot Z + R]$ is negligible. (This is not enough to imply that the probability that there *exists* such an S is negligible, since $\mathscr{S}$ may have superpolynomial cardinality; we will come back to this point later.) A crucial observation is that in $\mathsf{ORAM}^{*\infty}_L$, the sizes of data items plays no role. In particular, given an access sequence $\mathbf{s}$ and associated random coins r, the location of each item in the tree is entirely determined *independently of the size of the data items.*

Given an access sequence $\mathbf{s}$ with m items in total, and a *size allocation vector* $\mathbf{w} = (w_i)_{i \leq m} \in [0, 1]^m$, define $\mathbf{s}(\mathbf{w})$ to be the access sequence $\mathbf{s}$, modified such that at the outcome of the sequence the i-th item has size w_i. Let Π be the set of permutation matrices of size m. Let $X_{\mathbf{s},L,S}(\mathbf{w}) = \max_{\mathbf{P} \in \Pi}(\max(0, u^S(ORAM^{*\infty}_L[\mathbf{s}(P\mathbf{w})]) - (|S| \cdot Z + R)))$. By Lemma 4.4.2, $\mathbb{E}[X_{\mathbf{s},L,S}(\mathbf{w})]$ is an upper bound on $\Pr[u^S(\mathsf{ORAM}^{*\infty}_L[\mathbf{s}]) > |S| \cdot Z + R]$, so it is enough to show that $\mathbb{E}[X_{\mathbf{s},L,S}(\mathbf{w})]$ is negligible. This will follow from the next lemma.

Lemma 4.4.3. *Let $\mathbf{s}$ be a legal sequence of accesses, and let $S \in \mathscr{S}$. Then the map $f : \mathbf{w} \mapsto \mathbb{E}[X_{\mathbf{s},L,S}(\mathbf{w})]$ is Schur-convex.*

Proof. First, we show that $X_{\mathbf{s},L,S}$ is convex when the random coins used in the ORAM construction are fixed. Until further notice, we assume that all random coins are fixed. Only $\mathbf{w}$ varies. Let $\lambda \in [0, 1]$, and let $\mathbf{v}$, $\mathbf{w}$ be two size allocation vectors. Observe that the map $g : \mathbf{w} \mapsto u^S(ORAM^{*\infty}_L[\mathbf{s}(P\mathbf{w})])$ is linear. This is because, as already noted, whether an item is stored in a node from S or not is independent of the weight of the items. As a consequence, $g(\mathbf{w})$ is equal to the sum of the weights of items stored in S, *i.e.* it is a fixed linear combination of w_i's (with binary coefficients). Since g is linear, it is trivially convex.

Also observe that for any constant C, the map $h : x \mapsto \max(0, x - C)$ is increasing and convex. Since the composition of an increasing convex function with a convex function is convex, we deduce that the map $h \circ g$ is convex. Since $X_{\mathbf{s},L,S}(\mathbf{w}) = \max_{\mathbf{P} \in \Pi} h \circ g(\mathbf{P}\mathbf{w})$, it is a maximum of convex maps, so it is also convex.

On the other hand, $X_{\mathbf{s},L,S}(\mathbf{w})$ is symmetric by construction, since it takes the maximum over all permutations of $\mathbf{w}$. By Lemma 4.2.1, since $X_{\mathbf{s},L,S}(\mathbf{w})$ is both symmetric and convex, it is Schur-convex.

It remains to show that Schur convexity still holds when considering the expectation of $X_{\mathbf{s},L,S}(\mathbf{w})$. (From now on, we no longer assume that random coins are fixed.) However, it is straightforward to show that if a probabilistic map is Schur-convex for every fixed choice of random coins (sometimes called stochastical Schur-convexity), then its expectation is also Schur-convex [MOA79]. We conclude that $\mathbf{w} \mapsto \mathbb{E}[X_{\mathbf{s},L,S}(\mathbf{w})]$ is Schur-convex. ■

Corollary 4.4.4. *Let* $\mathbf{s}$ *be a legal sequence of accesses with weight vector* $\mathbf{w}$*, and let* $S \in \mathscr{S}$*. Then* $\mathbb{E}[X_{\mathbf{s},L,S}(\mathbf{w})]$ *is negligible.*

Proof. For an access sequence to be legal, its weight vector $\mathbf{w}$ must satisfy that $w_i \leq B$ for all i, and $\sum w_i \leq NB$. Observe that all such vectors are majorized by the vector $\mathbf{v} = (B, \ldots, B, 0, \ldots, 0)$ containing N initial B's. Since $\mathbb{E}[X_{\mathbf{s},L,S}(\mathbf{w})]$ is Schur-convex, it follows that $\mathbb{E}[X_{\mathbf{s},L,S}(\mathbf{w})] \leq \mathbb{E}[X_{\mathbf{s},L,S}(\mathbf{v})]$: in order to upper bound $\mathbb{E}[X_{\mathbf{s},L,S}(\mathbf{w})]$, it suffices to focus on the weight vector $\mathbf{v}$. (This is the point of using a Schur-convexity argument.)

But in the case of the vector $\mathbf{v}$, all items are of the same size B, or of size 0.[1] In that case, ORAM^* behaves exactly like ORAM, except that accesses to items of size 0 translate to evictions without any prior item access. In particular, the usage of S is the same for ORAM^* and ORAM. Since we assume that ORAM has the free eviction property, it remains correct when allowing eviction queries by the client. Since it is also assumed to be correct and to satisfy W $\Rightarrow$ F, it follows that the usage of S cannot exceed $|S| \cdot Z + R$ except with negligible probability, hence the same holds for ORAM^*, and we are done. ■

So far, we have shown that the probability that any given S witnesses a failure in ORAM^* is negligible. To conclude the proof, it remains to show that the probability that there *exists* an $S \in \mathscr{S}$ witnessing a failure is negligible. This does not follow immediately from the previous statement, because $|\mathscr{S}|$ may be superpolynomial. However, looking at the proof of Lemma 4.4.2, we see that when switching from expectation to probability and back, we only lose a factor t. In our case, the stash size is a random variable bounded by NB, so we have that for every S,

$$\Pr[u^S(\mathsf{ORAM}^{*\infty}_L[\mathbf{s}(\mathbf{v})]) > |S| \cdot Z + R] \leq NB \Pr[u^S(\mathsf{ORAM}^{\infty}_L[\mathbf{s}]) > |S| \cdot Z + R].$$

Since ORAM is assumed to satisfy W $\Rightarrow$ F *via union bound*, and $NB \in \mathrm{poly}(\lambda)$, we know that the sum of the latter quantity over all $S \in \mathscr{S}$ is negligible, hence ORAM^* inherits the same union bound property. It follows that the probability that there exists a failure witness S for ORAM^* is negligible. Since ORAM^* satisfies the F $\Rightarrow$ W property, we conclude that ORAM^* is correct. □

4.4.4 Experimental Results

To test empirically the correctness of our weighted ORAM, we implemented a Path ORAM structure and performed simulated accesses. We did two experiments: one with N objects

[1] The reader may observe that items of size 0 are not technically legal per the earlier definition of wORAM, which asks that items are of size at least 1; however, $\mathtt{Weighted}(\mathsf{ORAM})$ remains well-defined even for items of size 0, so nothing stops us from using them within the proof —the reason we forbade items of size 0 is that they would allow for an unbounded number of items, which would require a position map of unbounded size, but this is irrelevant for the current line of reasoning.

of the same size (which simulates the standard case) and one with objects of variable sizes (the sizes are uniformly random, but sum to N). Our results are presented as graphs in Figure 4.1.

We took inspiration from the experiment in Section 7 of [SvS+13b]. The experiment went as follows:

- We generated ORAM structures for N objects, with $N = 2^L$ and $L \in \{10, 11, \ldots, 22\}$. The bucket size is $Z \in \{3, 4\}$
- We chose the maximum block size to be $B = 512$.
- For the standard ORAM simulation, all blocks were of size B. For the wrighted ORAM simulation, blocks were taken uniformly at random in $[B]$, with the total sum of the sizes being $N \cdot B$. The number m of blocks generated is roughly $2 \cdot N$.
- We start with the Path ORAM loaded randomly with the objects at its leaves, and perform between $10 \cdot m$ and $50 \cdot m$ accesses in the order $\{1, 2, \ldots, m, 1, 2, \ldots\}$.

Figure 4.1 suggests that objects of variable size are even less prone to stash overflows than the standard case. Regarding bucket size, we make an observation similar to that of [SvS+13b]: even though the correctness of the ORAM was proved for $Z + 1 = 6$, the construction appears resilient enough to work correctly even when $Z + 1 = 4$. In [SvS+13b], the empirical results suggest that Path ORAM can be used with Z as low as 4.

4.5 Application to Existing Tree ORAMs

In this section we present several concrete constructions: a weighted Simple ORAM, based on Chung and Pass's Simple ORAM [CP13], and a weighted Path ORAM, based on the seminal work by Shi *et al.* [SvS+13b]. The construction for Path ORAM can be easily adapted to build a weighted *Random-Index ORAM* from the one presented in [HK22], as the block-holding structure is virtually the same. We also sketch the application to Circuit ORAM [WCS14] and OPRAM [CCS17]. By Theorem 4.4.1, in each case, it suffices to show that the scheme is suitable. The weighted variant is then obtained by applying the `Weighted` transformation.

4.5.1 Weighted Simple ORAM [CP13]

In `SimpleOram`, each bucket has a capacity of $Z = \mathcal{O}(\log N)$, and the ORAM overflows if and only if there is a bucket with more than Z items: there is no stash. From the perspective of the Tree ORAM framework from Section 4.3.2, a stash does exist, however, it is required that it is empty at the outcome of any (legal) sequence of accesses. That is, we set the stash bound R to 0.

In `SimpleOram`:

- The `ChooseEvictionPath`(*leaf*) method is implemented by choosing a path uniformly at random (the *leaf* argument is ignored).

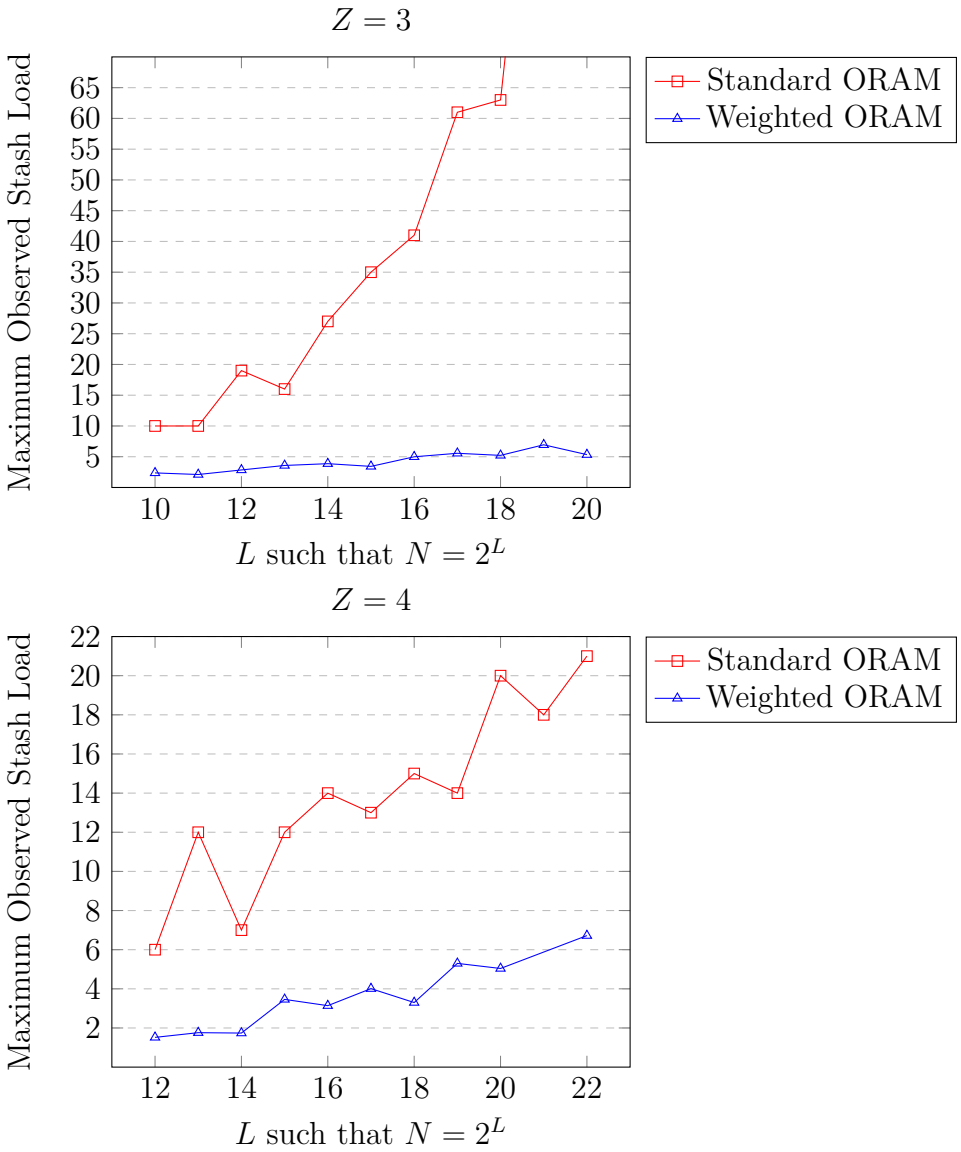

Figure 4.1: Experimental results for $Z \in \{3, 4\}$

- The `ChooseNextBlock`(*stash*, *bucket*, *path*) method is implemented by returning the first item among items whose position is such that its meet with the current path is exactly *bucket*. (In other words, all items are stored as low as possible along the eviction path, while preserving the invariant that they are stored above their associated leaf.) The correctness of `SimpleOram` relies on the fact that all such items will fit in the current bucket; items are never pushed somewhere else in case a bucket is full.

We want to show that `SimpleOram` is suitable. Define $\mathscr{S}$ to be the set containing the singleton $\{bucket\}$ for each tree node *bucket*. The fact that the correctness of `SimpleOram` is equivalent to the fact that no element of $\mathscr{S}$ witnesses a failure is immediate, since the correctness of `SimpleOram` requires precisely that no node overflows. Hence, `SimpleOram` satisfies W $\Rightarrow$ F (and F $\Rightarrow$ W) with respect to $\mathscr{S}$. The fact that `Weighted`(`SimpleOram`) satisfies F $\Rightarrow$ W is immediate for the same reason. `SimpleOram` also satisfies the union bound requirement, because its analysis in [CP13] relies on just such a union bound. The fact that it supports free evictions also follows directly the analysis in [CP13] (additional evictions translate to more success chances in the dart game argument at the center of the analysis). We conclude that `SimpleOram` is suitable.

Theorem 4.5.1. `Weighted`(`SimpleOram`) *is a correct wORAM scheme.*

4.5.2 Weighted Path ORAM [SvS+13b]

Let $ORAM^Z \leftarrow$ `PathOram.Setup`(N, Z) be an instance of Path ORAM. The bucket capacity Z for Path ORAM is a small constant; the scheme is proven correct for $Z = 5$, we shall use this value. The stash capacity is $R = \mathcal{O}(\log N)$.

In `PathOram`:

- `ChooseEvictionPath`(*leaf*) simply returns $\mathcal{P}(leaf)$.
- `ChooseNextBlock`(*stash*, *bucket*, *path*) returns an item from the buffer such that its associated leaf is below *bucket*, and the meet between the position of the item and *path* is lowest among such items.

Theorem 4.5.2. `Weighted`(`PathOram`) *is a correct wORAM scheme.*

Proof. Define $\mathscr{S}$ to be the set of all subtrees of the ORAM tree, where a *subtree* is defined to be a subset of nodes closed for the *parent* relation (*i.e.* if the set contains a node, it also contains its parent). The analysis of [SvS+13b] proves that `PathOram` satisfies both F $\Rightarrow$ W and W $\Rightarrow$ F with respect to $\mathscr{S}$, via a union bound. The fact that `PathOram` supports free evictions also follows from the analysis. In the remainder, we focus on showing that `PathOram`$^* \leftarrow$ `Weighted`(`PathOram`) satisfies F $\Rightarrow$ W.

We follow a similar approach to the initial part of the analysis in [SvS+13b]. Let us define a post-processing algorithm G_Z, which is applied to $ORAM^{*\infty}$ after a sequence of accesses. Like in the original analysis of Path ORAM, the G_Z algorithm is a "thought experiment": that is, the algorithm is used within the proof, but never actually run by any party in the ORAM protocol. As a consequence, we can allow G_Z to perform operations that are not normally allowed within the wORAM framework. In particular, we let G_Z "split" any object of size w in two objects of sizes w_1 and w_2 such that $w_1 + w_2 = w$, storing the two chunks at distinct locations. G_Z repeats the following process, as long as there are overfull buckets (*i.e.* whose size is strictly more than Z—to avoid cluttering the notation, all sizes are implicitly divided by the block size B):

1. Select a bucket b that has load strictly more than Z. Remove blocks from the bucket (splitting if needed) so that it ends up having load exactly Z. Removed blocks are stored in a temporary buffer.

2. Find the nearest ancestor of b that has load strictly less than Z. Store items from the buffer into this ancestor, until its load is exactly Z (or the buffer is empty), splitting if needed.

3. As long as the buffer is not empty, repeat the previous step, working our way up the path $\mathcal{P}(b)$, until reaching the stash. At that point, any remaining items in the buffer is stored in the stash.

First, let us prove that the stash usage (i.e. the cumulated size of all items in the stash) of the post-processed ∞-ORAM is greater than the stash usage of $ORAM^{*Z}$:

$$st^Z(ORAM^{*\infty}[\mathbf{s}]) \geq st(ORAM^{*Z}[\mathbf{s}]). \tag{4.2}$$

First, notice that the order in which overfull blocks are processed by G_Z has no bearing on the final occupancy of the nodes and stash. In particular, $st^Z(ORAM^{*\infty}[\mathbf{s}])$ is well-defined, even though we put no restriction on the order in which G_Z processes overfull buckets. To show this, we can generalize the argument from [SvS+13b]: assume that G_Z processes blocks from the bucket β_1 at level l_1 on path p_1, then blocks from the bucket β_2 at level l_2 on path p_2. We want to show that the loads in the buckets in $p_1 \cup p_2$ do not change if we let G_Z process β_2 before β_1. Without loss of generality, we can assume that those buckets are siblings (i.e. $l_1 = l_2 = l$), since only $p_1 \cap p_2$ will be affected by a change in the order. Assume that the post-processed blocks from β_1 (resp. β_2) are of total size W_1 (resp. W_2). G_Z first distributes a "mass" of size W_1 in the buckets from level $l-1$ to -1 in $p_1 \cap p_2$, and then a mass of size W_2 in those same buckets. Before the distribution, let us call V_i the available space in the bucket at level $i \in \{-1, 0, \ldots, l-1\}$ on path $p_1 \cap p_2$. When distributing a mass W, G_Z performs Algorithm 3 (we assume that $V_{-1} = \infty$): The $\{V_i\}$ are the same after a successive application of Algorithm 3 on W_1 then W_2 or after its application on W_2 then W_1.

Algorithm 3 Distribution of mass of blocks

Distribution(W):

```
1: i ← l
2: while W > 0 do
3:     i ← i − 1
4:     if V_i ≥ W then
5:         V_i ← V_i − W
6:         W ← 0
7:     else
8:         V_i ← 0
9:         W ← W − V_i
```

Remark. We wish to draw the reader's attention to one point: in what precedes, we consider for simplicity that blocks are taken in bulk from the buckets, whereas in what follows it is more convenient to assume that G_Z processes them individually. It doesn't make a difference for the same reason that the order doesn't matter.

We can finally prove (4.2). Informally, we can see that during the accesses, $ORAM^{*Z}$ stores blocks in buckets and in the stash in a more lenient way than G_Z, since it allows blocks to "stick out" of the buckets. More precisely, after the accesses of $\mathbf{s}$ in $ORAM^{*\infty}[\mathbf{s}]$, there exists a way to move blocks from the buckets they reside in to their final destination from $ORAM^{*Z}[\mathbf{s}]$ (in another bucket or the stash). Since the order in which we post-process blocks from the buckets does not matter, we can assume that this particular order is accessed by G_Z. If that is the case, after the processing of each block, G_Z puts that block in the bucket where it belongs according to $ORAM^{*Z}[\mathbf{s}]$. However, should a part of the block (or its entirety) stick out of the bucket (i.e. causes the load to become $\geq Z$), this part will be moved to a higher block or the stash. Thus the processing of each block by G_Z causes the stash size to either stay the same or to increase. Thus at the end of the processing, $st^Z(ORAM^{*\infty}[\mathbf{s}]) \geq st(ORAM^{*Z}[\mathbf{s}])$.

We are now ready to show that `Weighted(PathOram)` satisfies F $\Rightarrow$ W. By (4.2), if correctness fails for `Weighted(PathOram)`, then it also fails (with the same random coins) for the post-processed ∞-ORAM variant. That is, $st^Z(ORAM^{*\infty}[\mathbf{s}]) > R$. Let us define T to be the maximal subtree that only contains buckets of size at least Z after the post-processing. If a bucket b is not in T, it has an ancestor b' that has usage strictly less than Z, so the blocks of b cannot go to the stash. Thus all blocks in the stash came from buckets in T, and thus $u^T(ORAM^{*\infty}[\mathbf{s}]) > n(T) \cdot Z + R$. This shows that T is a failure witness. □

Discussion. In the proof above, a "splitting" variant of wORAM was introduced. In that splitting variant, items can be split into pieces of arbitrary sizes, and each piece can be stored at a different tree node (still maintaining the invariant that all pieces belonging to the same item are stored somewhere along the path from the root to the leaf associated with the item). Splitting is a convenient intermediate abstraction, because of its "continuous" behavior, which avoids rounding issues.

It is worth noting that splitting is not just a proof abstraction, but could also be used in actual tree-based wORAM algorithms. In fact, we could have defined our weighted transformation to allow splitting. In that case, the proof above shows that the weighted variant of Path ORAM would have the same bucket capacity Z as the original Path ORAM.

By contrast, our actual construction, which does not use splitting, needs to increase the bucket capacity to $Z + 1$ to avoid "rounding" issues. Intuitively, without splitting, if a bucket of capacity ZB words contains data of total size $(Z - 1)B + 1$, and the client wants to insert a block of maximum size B, it cannot fit in that bucket. This means each bucket can have up to $B - 1$ words of "wasted" space. Note that this cannot happen in standard ORAM, where all items have size B: it is an issue specific to weighted ORAM (and weighted allocation mechanisms in general). Splitting solves the issue without the need to increase bucket capacity: the ability to split avoids any wasted space.

Our construction solves the issue differently, without splitting, by increasing bucket capacity to $(Z+1)B$ words (or $Z+1$ full blocks). This guarantees that if a bucket cannot receive a new block, then it already contains at least ZB words. In short, the reason we increase bucket capacity to $Z + 1$ in our constructions is to resolve rounding issues inherent to fitting weighted items into buckets of fixed capacity.

As noted, we could have chosen to define our `Weighted` transformation to allow splitting. We avoided that option, because it complicates implementation in practice, and may require significantly more metadata (each piece needs an identifier tag). Nevertheless, it

is a valid option, and may be preferable if a slightly lower Z is desirable.

4.5.3 Weighted Oblivious Parallel RAM [BCP16]

Boyle, Chung, and Pass's protocol [BCP16] is based on Simple ORAM. Their framework present ORAM protocols for parallel algorithms, i.e. with multiple processors (clients). The `Weighted` function is not impacted by the fact that there are several clients: The modifications to the subroutines still capture this case, and the correctness analysis holds. The only new component is the broadcast routine, where one of the CPUs broadcasts information about a certain block to the other CPUs. These messages are bounded by the size of the block. That could lead to a leakage however: we need to index every block, in the weighted case there are m objects and $\log m \geq \log N$. Thus, we can lower bound the size of the messages by $\mathcal{O}(\log N)$: their size will not leak information on the block. This yields a correct weighted OPRAM.

4.5.4 Weighted Circuit ORAM [WCS14]

Circuit ORAM is a variant of Path ORAM, where the client only needs local space to hold one block. To achieve this, the eviction algorithm is slightly different and the stash is stored by the server (as the parent of the root node) instead of locally.

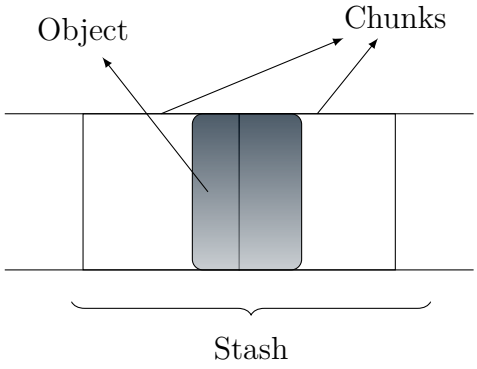

Figure 4.2: A block of size $< B$ in 2 chunks

The correctness analysis of this scheme is based on the same principles as the one for Path ORAM. The function `Weighted` yields a correct ORAM here too. To prove this, we only need to show how we can adapt for the fact that the stash is stored on top of the tree. Figuring out which way to do this is not obvious since, because of the varying block size, the client cannot simply stream the content of the stash block by block: it would leak block size information. We propose a simple fix, which we also use when dealing with Trivial ORAM:

- We allow the client to store an additional space of size B (the maximal size of a block). This gives the client a local space of at least $2 \cdot B$.
- Whenever the client needs to access the stash, the client streams the content of the stash *chunk by chunk*, where each chunk is of size B. That way, since a block must always reside inside at most 2 chunks (see Figure 4.2), the client will read every object at the end of the stash stream.

- When the client wishes to write back a block, it is done locally among two chunks.

This way, the scheme stays correct and secure even with objects of variable size. The application of the suitability criterion shows that Circuit ORAM is compatible with blocks of variable size.

4.6 Searchable Encryption from Weighted ORAM

With Searchable Symmetric Encryption (SSE), a client can delegate the storage of a database to a honest-but-curious server. The client is then able to perform searches on the database by issuing `Search` queries to the server. In the case of *dynamic* SSE, the client may also update the database by issuing `Update` queries to the server. The security goal is that the information leaked to the server during these operations should be limited, in a sense that will be defined soon.

Here, we focus on the case of single-keyword search. In that setting, the client's database consists of a collection of documents, and `Search` queries ask to retrieve all documents that contain a given keyword. In modern SSE schemes, this functionality is realized efficiently by building a reverse index: For each keyword w, a list of the identifiers of documents matching the keyword, written $\mathsf{DB}(w)$, is maintained on the server side in some encrypted form. *Response-revealing* SSE allows the server to learn the list of document identifiers, while response-hiding SSE does not: they are sent back to the client in encrypted form. Once having retrieved the desired document identifiers, the client may perform some additional computation, such as intersecting the results with other queries, or may fetch the documents on the same or a different server. In the case of response-revealing SSE, if the same server stores the reverse index and the documents, the server can immediately send back the documents without the need of an additional roundtrip, at the cost of possibly leaking additional information to the server. We note that the documents could be stored in an ORAM to avoid additional leakage, and that a weighted ORAM would reduce the performance cost of this approach. However, as in most SSE literature, we focus on the reverse index.

For efficiency reasons, SSE typically does not seek to have minimum leakage, but rather to strike a compromise between security and performance by allowing a controlled amount of leakage. In the security model, the leakage allowed by the scheme is expressed by a *leakage function* $\mathcal{L} = \{\mathcal{L}_{\texttt{Setup}}, \mathcal{L}_{\texttt{Search}}, \mathcal{L}_{\texttt{Update}}\}$. The security model asks that during a `Setup` operation (resp. `Search`, `Update`) with input x, the information leaked to the server is included in $\mathcal{L}_{\texttt{Setup}}(x)$ (resp. $\mathcal{L}_{\texttt{Search}}(x)$, $\mathcal{L}_{\texttt{Update}}(x)$). More formally, it is required that there must exist a simulator $\mathcal{S}$ such that the view of the server during $\texttt{Setup}(x)$ (resp. $\texttt{Search}(x)$, $\texttt{Update}(x)$) should be indistinguishable from $\mathcal{S}(\mathcal{L}_{\texttt{Setup}}(x))$ (resp. $\mathcal{S}(\mathcal{L}_{\texttt{Search}}(x))$, $\mathcal{S}(\mathcal{L}_{\texttt{Update}}(x))$).

Response-revealing SSE schemes leak the *access pattern*: That is, the server learns the identifiers of all documents matched by a query. In some use cases, access pattern leakage can be quite damaging, and allow the server to infer a sizable amount of information about the database [CGPR15, GLMP19]. Even in the case of response-hiding SSE, the server can typically learn the *query pattern*: that is, the server can learn whenever the client repeats the same query. In many cases, the server can learn the *volume* of the answer: that is, the number of documents matched by the query.

In some use cases, these different types of information leakage can be quite damaging, as shown by so-called *leakage-abuse* attacks [CGPR15, GSB+17]. To thwart those

Scheme	client storage	bandwidth overhead
TWORAM [GMP16]	$\mathcal{O}(1)$	$\mathcal{O}(\lambda \log^2 N)$
ZeroSSE	$\mathcal{O}(W)$	$\mathcal{O}(\log(N/U))$
ZeroSSE′	$\mathcal{O}(1)$	$\mathcal{O}(\log^2 W + \log(N/U))$
BlockSSE	$\mathcal{O}(W)$	$\mathcal{O}(\log(N/B))$
BlockSSE′	$\mathcal{O}(1)$	$\mathcal{O}(\log^2 W + \log(N/B))$

Figure 4.3: Overhead of ORAM-based SSE constructions. U is an upper bound on the longest list size, $W \leq N$ is the number of keywords, B is the ORAM block size.

attacks, recent works have developed various protections, such as volume-hiding SSE [KM19, PPYY19], and the line of work on *leakage suppression* [KMO18]. The strongest form of protection, considered for instance in [GMP16, MM17, KMO18], involves the use of ORAM, or specialized variants of ORAM. This raises some questions about how to optimize the use of ORAM, in order to preserve the high efficiency goal of SSE. In particular, as discussed *e.g.* in [GMP16], since reverse indexes contain lists that can greatly vary in size, it is not obvious how to fit them into a (fixed-block size) ORAM. Our main point in this section is that the weighted construction introduced here fits this setting perfectly. Concretely, we propose two SSE constructions based on weighted ORAM: ZeroSSE and BlockSSE. A brief overview is given Figure 4.3. We note that the main point of TWORAM, not reflected in the table, is to reduce the number of roundtrips in the iterative version of Path ORAM, thanks to a clever use of garbled circuits. However garbled circuits add a considerable overhead in practice.

4.6.1 Preliminaries

We follow the standard definition of SSE. A dynamic SSE scheme Σ consists of four protocols, defined as follows.

- $\Sigma.\mathtt{KeyGen}(1^\lambda)$: Takes as input the security parameter λ. Outputs the master secret key K.
- $\Sigma.\mathtt{Setup}(\mathsf{K}, N, \mathsf{DB})$: Takes as input the client secret key K, an upper bound on the database size N, and a database DB. Outputs an encrypted database EDB.
- $\Sigma.\mathtt{Search}(\mathsf{K}, w, \mathsf{st}; \mathsf{EDB})$: The client receives as input the secret key K, and keyword w. The server receives as input the encrypted database EDB. Outputs the list of document identifiers associated with the keyword w
- $\Sigma.\mathtt{Update}(\mathsf{K}, (w, e); \mathsf{EDB})$: The client receives as input the secret key K, and a pair (w, e) of keyword w and document identifier e. The server receives as input the encrypted database EDB. Outputs updated encrypted database EDB' for the server, where the list of documents identifiers associeted with w has e appended to it.

The security model expresses that the view of the server can be simulated by an efficient simulator, receiving as input only the output of the leakage function. In more detail, we define two games, $\mathsf{SSEReal}$ and $\mathsf{SSEIdeal}$. First, the adversary chooses a database DB. In $\mathsf{SSEReal}$, the encrypted database EDB is generated by $\mathtt{Setup}(\mathsf{K}, N, \mathsf{DB})$, whereas in $\mathsf{SSEIdeal}$, the encrypted database is simulated by a (stateful) simulator $\mathcal{S}$ on input

$\mathcal{L}_{\texttt{Setup}}(\mathsf{DB}, N)$. After receiving EDB, the adversary can issue search and update queries. In $\mathsf{SSEReal}$, queries are answered using the real-world protocol. In $\mathsf{SSEIdeal}$, the `Search` queries (resp. `Update`, `Setup`) on input x are simulated by $\mathcal{S}$ on input $\mathcal{L}_{\texttt{Search}}(x)$ (resp. $\mathcal{L}_{\texttt{Update}}(x)$, $\mathcal{L}_{\texttt{Setup}}(x)$). Finally, the adversary outputs a bit b.

The scheme is said to be $\mathcal{L}$-*secure* (*i.e.* secure with respect to the leakage function $\mathcal{L}$) if for all PPT adversaries, there exists a PPT simulator such that the transcripts in the real and ideal world are computationally indistinguishable.

4.6.2 ZeroSSE

A line of recent works has aimed to hide volume leakage: that is, to hide the number of identifiers matching a given query [KM19, PPYY19]. Hiding volume leakage seems sensible when using ORAM techniques to hide the query pattern, since volume leakage reveals information about the repetition of queries. This leads to the question of building an ORAM that also hides volume. For that purpose, an upper bound U is assumed on the volume of the longest list (that is, the longest query answer). As discussed in [GMP16], the first approach one may think of is to use an ORAM with block size U; however, this would require padding all lists to U, which would be prohibitive in many use cases, since the longest list may be several orders of magnitude larger than the average list size. In the worst case, the blowup in storage is $\Omega(U)$, even before considering ORAM overheads. Another approach would be to use a smaller block size, at the cost of a larger ORAM overhead.

The idea of $\mathsf{ZeroSSE}$ is simply to use the weighted variant of Path-ORAM, `Weighted`(`PathOram`) with $B = U$ as the block size upper bound. Relative to the previous two approaches, this minimizes both the overhead due to padding, which is nonexistent since no padding is necessary, and the overhead due to ORAM, since we use the largest block size possible. In fact, since our main result is that Path ORAM can handle items of variable size at essentially no overhead, we contend that this is both the most natural and most efficient solution to building SSE with minimum leakage.

We define $\mathsf{ZeroSSE}$ in more details as follows. We note that `Setup` takes as input additional parameters U, which is an upper bound on the longest list size, and W, an upper bound on the number of keywords.

- $\mathsf{ZeroSSE}.\texttt{Setup}(\mathsf{K}, N, \mathsf{DB}, U)$: Initializes `Weighted`(`PathOram`) with block size U and number of leaves $\lceil N/U \rceil$, containing as (variable-size) blocks $\mathsf{DB}(w)$ for each keyword w. The position map, of size $\mathcal{O}(W)$ memory words, is stored on the client side.

- $\mathsf{ZeroSSE}.\texttt{Search}(\mathsf{K}, w; \mathsf{EDB})$: The client queries the ORAM for keyword w to retrieve $\mathsf{DB}(w)$.

- $\mathsf{ZeroSSE}.\texttt{Update}(\mathsf{K}, (w, e); \mathsf{EDB})$: The client queries the ORAM for keyword w to retrieve $\mathsf{DB}(w)$, and simply writes back $\mathsf{DB}(w) \cup \{e\}$. (Recall that our weighted construction allows modifying the size of blocks on the fly.)

$\mathsf{ZeroSSE}$ uses the non-iterative variant of Path-ORAM. This is because a client storage of $\mathcal{O}(W)$, while undesirable in general, is often accepted in forward-secure SSE [BF19]. Alternatively, we define $\mathsf{ZeroSSE}'$ to use the fully iterative version of Path-ORAM, which reduces the client storage to $\mathcal{O}(1)$ memory words, at the cost of additional roundtrips, and an additional $\mathcal{O}(\log^2 W)$ bandwidth overhead. (In theory, this could be reduced to

$\mathcal{O}(\log W)$ using an optimal ORAM for the position map, but current constructions of optimal ORAM are not practical.)

Theorem 4.6.1 (Security of ZeroSSE)**.** *Assuming Path-ORAM is a correct and secure ORAM scheme,* ZeroSSE *is $\mathcal{L}$-secure with respect to the leakage function $\mathcal{L} = \{\mathcal{L}_{\texttt{Setup}}, \mathcal{L}_{\texttt{Search}}, \mathcal{L}_{\texttt{Update}}\}$, with $\mathcal{L}_{\texttt{Setup}} = \{N, U\}$ and $\mathcal{L}_{\texttt{Search}} = \mathcal{L}_{\texttt{Update}} = \varnothing$.*

A proof of Theorem 4.6.1 is given in Appendix B.1.

4.6.3 BlockSSE

An interesting peroperty of ZeroSSE is that updates are indistinguishable from searches. In fact, addition and deletion of an arbitrary number of documents in a list can be performed in a single interaction at no additional cost. However, this also means that adding a single document to a keyword incurs an $\mathcal{O}(U \log^2 U)$ bandwidth cost. If cheaper updates for single documents are desirable, an alternative solution is to use a smaller block size. A smaller block size (linearly) reduces the cost of updates, while (logarithmically) increasing the cost of searches.

BlockSSE offers that trade-off, by storing the reverse index in a wORAM with parametrizable maximum block size B. An interesting feature of BlockSSE is that we can choose the block size B such that the size of a Path-ORAM tree node ZB is one memory page (or an integral number of memory pages). This optimizes the IO-efficiency of the resulting SSE, as discussed *e.g.* in [BBF+21]. This results in the first SSE with both high memory efficiency and no access pattern leakage.

While BlockSSE may perform significantly better than ZeroSSE in update-heavy workloads, the fact that searches and updates are indistinguishable, regardless of the number of documents added or deleted during an update, is lost. BlockSSE also does not support deletions by default, although they can be added generically at some additional cost, as in [Bos16].

To reduce the size of the position map, we borrow the pointer idea introduced in [WNL+14]. Namely, each block belonging to the same list $\mathsf{DB}(w)$ contains the position of the previous block. This allows the position map, stored on the client, to only store the position of the last block, resulting in $\mathcal{O}(W)$ storage.

We define BlockSSE in more details as follows. Note that `Setup` takes as input additional parameters B, which is the desired block size, and W, and upper bound on the number of keywords. `Update` takes as additional parameter U, which is an upper bound on the longest list size.

- $\mathsf{BlockSSE}.\texttt{Setup}(\mathsf{K}, N, \mathsf{DB}, U)$: Initializes $\texttt{Weighted}(\texttt{PathOram})$ with block size B and number of leaves $n = \lceil N/B \rceil$. For each keyword w, the list $\mathsf{DB}(w)$ is split into $\lceil \mathsf{DB}(w)/B \rceil$ chunks of size at most $B - \lceil \log n \rceil$, with no padding. The i-th chunk for keyword w is inserted into the ORAM at a random position, together with the position of the $(i-1)$-th chunk. The position l_w of the last chunk is stored on the client side.

- $\mathsf{BlockSSE}.\texttt{Search}(\mathsf{K}, w, U; \mathsf{EDB})$: The client queries the ORAM at position l_w, retrieves the last chunk $\mathsf{DB}(w)$, together with the position of the penultimate chunk. The client iteratively retrieves the position of each previous chunk in the same manner. Each chunk i is assigned a new position uniformly at random, updating the position stored within the next chunk accordingly.

- $\mathsf{BlockSSE}.\mathtt{Update}(\mathsf{K}, (w, e); \mathsf{EDB})$: If l_w is not a multiple of $B - \lceil \log n \rceil$, the client accesses the ORAM at position p_w, adds e to the data, replaces p_w by a new uniformly random position, and updates the ORAM according to this new data and position. If l_w is a multiple of $B - \lceil \log n \rceil$, a new block is inserted at a new uniformly random position, containing as data $\{e\}$ together with the position p_w of the previous last block. On the client side, p_w is updated to the position of the newly inserted block.

Theorem 4.6.2 (Security of $\mathsf{BlockSSE}$)**.** $\mathsf{BlockSSE}$ *is* $\mathcal{L}$*-secure with respect to the leakage function* $\mathcal{L} = \{\mathcal{L}_{\mathtt{Setup}}, \mathcal{L}_{\mathtt{Search}}, \mathcal{L}_{\mathtt{Update}}\}$*, with* $\mathcal{L}_{\mathtt{Setup}} = \{N, B\}$*,* $\mathcal{L}_{\mathtt{Search}}(w) = \lceil |\mathsf{DB}(w)|/B \rceil$*, and* $\mathcal{L}_{\mathtt{Update}} = \varnothing$*.*

A proof of Theorem 4.6.2 is given in Appendix B.1. $\mathsf{BlockSSE}'$ is the same as $\mathsf{BlockSSE}$, except the iterative variant of Path-ORAM is used.

Publication

This work was in an article published at the EUROCRYPT 2023 conference, under the following title [AM23]:

Weighted Oblivious RAM, with Applications to Searchable Symmetric Encryption.
by Léonard Assouline and Brice Minaud.

Advances in Cryptology – EUROCRYPT 2023: 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part I Apr 2023 Pages 426–455

Chapter

5 Structural Cryptanalysis of Implicit White-Box Implementations

5.1 Introduction

The promise of white-box cryptography is to hide key material from an adversary who has full access to the implementation of a cipher. Using the implementation, the adversary can compute the cipher on any given input, she may observe all intermediate variables, and she may even modify them at will to inject faults. Still, the implementation should be obfuscated in such a way that the adversary cannot infer the encryption key. In this manner, a white-box implementation of a cipher offers a functionality similar to running the encryption algorithm within a trusted enclave in hardware: it allows the user to execute the algorithm, without leaking sensitive key material contained in the code. The difference is that white-box cryptography aims to hide information in software.

This goal is quite attractive within industry. Historically, the concept was introduced by Chow *et al.* in two seminal articles, proposing white-box implementations of AES and DES [CEJvO02a, CEJvO02b]. Since then, white-box cryptography has been instrumental in Digital Rights Management solutions (e.g. for iTunes, or pay TV services), and payment applications. It also offers an alternative solution for third parties who often cannot access proprietary trusted computing units (e.g. in iOS).

While the demand for secure white-box cryptography is high within industry, all public white-box constructions have so far been proven insecure (see Section 5.1.2). In some cases, short-term security may be enough, as the goal of practical deployments may be to dissuade or slow down attackers, in conjunction with frequent rekeying of sensitive white-box implementations. Nevertheless, a secure white-box implementation of a block cipher would be a major breakthrough.

Towards that end, an innovative approach was proposed at Crypto 2022 by Ranea, Vandersmissen, and Preneel [RVP22]. The authors of [RVP22] show that by using so-called *implicit* representations of round functions, it is possible to realize white-box implementations of SPN and ARX ciphers with rich *non-linear* encodings. The use of non-linear encodings in white-box cryptography is not new, but previous uses of non-linear encodings were very restricted, while the approach of Ranea *et al.* allows for a much wider range of encodings. As an illustration of that point, Ranea *et al.* introduce the first white-box implementation technique suitable for ARX ciphers. To our knowledge, no prior technique applied to that setting.

To understand why non-linear encodings matter, it is helpful to look back to existing attacks on white-box cryptography. Roughly, published attacks fit into three families:

ad-hoc attacks, structural attacks, and gray-box attacks.

Ad-hoc attacks. Ad-hoc attacks target a specific white-box construction. The earliest attacks on white-box cryptography were of that type [BGE04, GMQ07, WMGP07]. While they show that the targeted design is insecure, they are restricted to that design.

Structural attacks. Structural cryptanalysis was introduced by Biryukov and Shamir, before the invention of white-box cryptography [BS01]. In brief, given black-box access to a function with some hidden structure, the goal of structural cryptanalysis is to recover that structure. In their original work, Biryukov and Shamir consider the so-called SASAS structure, where the function of composed of three S-box layers, interleaved with affine layers A. Because existing white-box approaches for white-box cryptography rely on hiding intermediate components of a block ciphers (typically, its round functions) behind low-degree (typically affine) encodings, it is naturally amenable to structural cryptanalysis. Indeed, structural attacks allow the attacker to peel off the encodings, and recover the original round function.

Structural cryptanalysis provides considerable insight: for instance the structural cryptanalysis of ASA contained in [BS01] implies that it is essentially impossible to hide the round function of an SPN cipher behind affine encodings. In fact, the authors of [DFLM18] argue that many ad-hoc attacks on white-box constructions can be recast as special cases of the structural attack on ASA. It is worth noting that structural attacks are also interesting outside of white-box cryptography—the notion even predates white-box cryptography. The work of [BS01] was meant to provide insight on SPN ciphers, and the structural attack of [MDFK18] applies to a multivariate public-key encryption scheme. (Some other attacks on multivariate cryptography are also naturally viewed as structural attacks, even if they do not use the terminology, such as [DFKYZD99].)

Gray-box attacks. Gray-box attacks use techniques inspired by side-channel cryptanalysis to break the security of white-box encryption [SMG16, GRW20]. After all, white-box security is intended to be a stronger requirement than resisting side channels: the attacker has full access to the implementation. This approach has been quite effective in automatically breaking constructions such as in the WhiBox challenge [BDG+22, BBD+22]. Nevertheless, it does not suffice to break the most secure variants, and is experimental in nature. Compared to structural attacks, it has the advantage of being more easily automatable for real-world deployment, but it does not provide the same kind of insight as structural attacks, which can be understood as general impossibility results.

5.1.1 Our Contributions

At Crypto 2022, Ranea, Vandersmissen, and Preneel have proposed a new technique for white-box cryptography: implicit encodings [RVP22]. A compelling aspect of their proposal is that it enables rich non-linear input encodings. This circumvents the structural cryptanalysis results of [BS01, DFLM18], which show that it is generically impossible to hide the round function of an SPN cipher behind affine or linear encodings. A limitation of [RVP22] is that output encodings must still be affine, and input encodings must be of low degree. This raises the question of a structural cryptanalysis of the ISA structure, where a layer of S-boxes is composed with a low-degree layer I at its input, and an affine

layer A at its output. That is, given black-box access to a function F of the form ISA, is it generically possible to recover the three inner layers I, S, A, such that $F = A \circ S \circ I$? Since the work of [RVP22] also applies to ARX ciphers, the same question applies to the IMA structure, where the S-box layer S is replaced with modular addition.

As our main result, we give a *structural* cryptanalysis of both ISA and IMA. In particular, this breaks the white-box proposal of [RVP22]. Beyond that, it shows that it is generically impossible to hide an S-box layer (or modular addition) behind encodings, if only the input encoding is non-linear. Because the implicit function technique of [RVP22] inherently requires affine output encodings, our work implies that it does not suffice to build secure white-box cryptography.

In terms of techniques, our structural cryptanalysis of ISA is similar in essence to the structural cryptanalysis of *ASASA* in [MDFK18]. Our structural cryptanalysis of IMA introduces interesting new techniques, which can be viewed as a generalization of linearization attacks. A practical validation of the attack is presented in Section 5.6.

5.1.2 Related Work

In 2002, Chow, Eisen, Johnson and van Oorschot introduced a white-box implementation of the Advanced Encryption Standard (AES) [CEJvO02a], and the Data Encryption Standard (DES) [CEJvO02b]. Those seminal contributions introduced the idea of white-box cryptography, but quickly proved insecure. In 2004, Billet *et al.* [BGE04] proposed an algebraic attack against the AES white-box implementation of [CEJvO02a]. Further attacks were proposed in [MGH09, LRdM+13]. Using differential cryptanalysis, [WMGP07] and [GMQ07] broke the DES S-box of the white-box implementation from [CEJvO02b] and [LN04]. These attacks led to a new proposal of white-box design in [XL09], but their attempt was thwarted by the attack from [MRP12]. In 2014, [BBK14] presented a new AES white-box implementation based on an Affine – S-box – Affine (ASASA) structure, which was broken by [MDFK18], using structural cryptanalysis. In [VRP22], the authors break a white-box implementation of the cipher Speck based on self-equivalence encodings, using a Gröbner basis approach. So-called gray-box attacks have also been proposed, including Differential Fault Analysis (DFA), and Differential Computer Analysis (DCA). These attacks offered new ways of attacking white-boxes, especially suited for automatic tools. Those techniques were repurposed from Side-Channel Attacks [BHMT16]. Enhancements of the DCA attacks were proposed in [BRVW19].

Beyond symmetric cryptography, the ability to hide a secret symmetric key from the users of a program is related to general obfuscation [DLPR13]. The most demanding form of program obfuscation, called virtual black-box obfuscation, has been proven to be impossible in [BGI+01]. This does not preclude the possibility that secure white-box cryptography may be possible, because white-box cryptography attemps to obfuscate a specific class of functions, namely symmetric ciphers. The diagonalization argument from [BGI+01] does not apply to that class of function. Nevertheless, obfuscating specific programs remains a hard task: all publicly available white-box schemes have been broken so far.

In a work published in TCHES 2023, Biryukov, Lambin and Udovenko present a cryptanalysis of the same white-box system we are studying in this chapter [BLU23]. Their attack is of a more practical nature than ours: their paper targets key recovery on the specific white-box construction of Ranea *et al.*, only in the case of ARX ciphers. To our understanding, it does not contain a general structural cryptanalysis of ISA or IMA

(Section 5 of their work comes the closest, but it leaves many details to later sections, and those later sections use specificities of the construction of Ranea *et al.*). By contrast, our work is a general structural cryptanalysis of ISA and IMA. It applies to the framework of Ranea *et al.*, by showing that the non-linear input encodings are not enough for security. It is not limited to their specific construction.

5.2 Technical Overview

At a high level, structural attacks in the literature follow the same general steps (including those in [BS01, BBK14, DFLM18]). We begin by outlining those steps. Then we will explain how our own technique proceeds. For now, we purposefully remain at a high level of abstraction.

Recall that the attacker has black-box access to a function with a known structure (such as ASA or SASAS). The goal is to recover the inner components.

Step 1: Model some part of the target structure as polynomial unknowns in a judicious manner. Then find a way to use the black-box oracle to derive low-degree equations over those unknowns.

This is typically the "clever" part of the attack. The cryptanalyst needs to find a way to exploit the properties of the target structure to derive equations.

Examples: In the first stage of the SASAS attack [BS01], unknowns are entries in the table of the inverse S-box (from the last S-box layer). Equations are derived using ideas from integral cryptanalysis. In the first stage of the ASASA attack [DFLM18], unknowns are entries in the matrix representing the last linear layer. Equations are derived via a modified cube attack.

Step 2: Solve the equation system by linearization.

In general, the equations from Step 1 may not be linear. On the other hand, in existing attacks, it is easy to derive new equations by repeating Step 1 as needed. The point is that, while solving even quadratic systems is NP-hard when the number of equations is limited, the problem becomes easy if the attacker has access to as many (random enough) equations as they want. Indeed, in that case, the linearization technique can be applied [BS01, AG11]. We refer the reader to Section 5.3.5 for more information on linearization, and do not discuss it here to continue focusing on the overall structure of the attack.

Step 3: Repeat the same general process to recover each one of the other components of the target structure. (Each time a component is recovered, the process of recovering the other components typically becomes easier, as the attacker has access to more and more information.)

In a typical attack, Step 1 and 2 recover the first or last layer of the structure. For example, Step 1 and 2 of the ASASA attack in [DFLM18] recover the last A layer. By composing the inverse of that layer with the black-box oracle, the attacker now has oracle access to an ASAS structure. Since it is a "simpler" structure, recovering the next layer is expected to be easier. (This is indeed the case in all structural attacks to our knowledge: peeling layers becomes progressively easier.)

This chapter contains two structural attacks: one on each of the new structures proposed for white-box protection in [RVP22]. Namely, the ISA structure ("low degree–S box–affine"), and the IMA structure ("low degree–modular addition–affine").

ISA attack.

For the ISA structure (unlike the IMA structure later on), our attack follows the blueprint presented earlier. In fact, our ISA attack is a direct generalization of the structural cryptanalysis of ASASA in [MDFK18], and shares much of the same main ideas. For that reason, we only sketch it here, and refer to the full description in Section 5.4 for more information.

In an IS structure, where the first layer has degree d, and the S-box is over s bits, a bit of the output has degree (at most) $d(s-1)$ over the input variables (see [MDFK18, Lemma 2]). A key observation is that the product of two output bits arising from the *same* S-box still has degree $d(s-1)$, whereas if the two bits arise from distinct S-boxes, the degree rises to $2d(s-1)$. The other key observation is that such a disparity in the behavior of the degree can be exploited by a cube attack. In more details, notice that in the ISA structure, the value of one bit at the output of the first two IS layers is a linear combination of output bits of the ISA function. Defining the coefficients of that linear combination as unknowns, it is possible to obtain (via a cube attack) a quadratic equation over those unknowns. From there, the general strategy outlined at the start of the section can be implemented.

IMA attack.

In prior work that followed the general steps outlined at the start of the section [BS01, BBK14, DFLM18], as well as in our own ISA attack, a crucial fact the attacks implicitly rely on is that the overall function has low algebraic degree[1]. Indeed, they rely on the key lemma of cube attacks, namely that the sum of a Boolean function of algebraic degree d over an affine space of dimension $d+1$ is zero (cf. Lemma 5.3.1)[2]. The number of summands is exponential in the degree d of the function, so this approach is only viable for low-degree functions.

An immediate problem when tackling the IMA structure is that modular addition over n bits has algebraic degree n. In particular, the IMA structure does not have low algebraic degree. For that reason, we do not directly use the cube attack approach of previous work. Instead, we rely on the iterative structure of modular addition: if modular addition is computed in the "elementary school" way by using carries, each output of the addition can be expressed as a quadratic polynomial in the input and in the value of the carry. That is, if we compute the modular addition $z = x \boxplus y$ in $\mathbb{Z}_{2^n}$, then denoting by z_i the i-th bit of z (likewise for x and y), we have:

$$z_i = x_i + y_i + c_i,$$

where c_i denotes the carry coming from the addition of the first $i-1$ bits, and "+" is addition in $\mathbb{Z}_2$. From there, a straightforward computation yields:

$$z_i = x_i + y_i + x_{i-1}y_{i-1} + c_{i-1}(x_{i-1} + y_{i-1}).$$

[1] More exactly, the attack of [BS01] relies on the SASAS function minus the last layer having low degree, while the attacks in [BBK14, DFLM18]rely on the whole function having low degree.

[2] This analysis is done differently in [BS01], but in retrospect, it could also have been done via a cube attack—except they did not exist yet [DS09].

What is interesting about this equation is that if the attacker has recovered the expression of the first $i-1$ bits of x, y, then the attacker can compute c_{i-1}. The attacker can then look for a linear relation between z_i (linear combination of output bits of the black-box function with unknown coefficients), $c_{i-1}(x_{i-1}+y_{i-1})$ (a known quantity times a linear combination of input bits with unknown coefficients), and the vector of pairwise products of the bits of x_{i-1} and y_{i-1} (again with unknown coefficients). To do so, the attacker creates a matrix whose rows are the concatenation of : input bits, input bits times c_{i-1}, output bits, and finally pairwise products of input bits—for random input bits. Because the previous relation, the kernel of the matrix must be non-trivial, and essentially reveals the unknown coefficients of x_{i-1} and y_{i-1} (we refer to Section 5.5 for more details).

To sum up, the attacker finds the coefficients of x_{i-1} and y_{i-1} iteratively, by using the recursive structure of modular addition. At each step, we note that there exist several possible solutions, forcing the attacker to guess. However, our experiments show that when the attacker makes a wrong guess, the very next step of the iteration always fails, meaning that wrong guesses are corrected immediately. We have validated the attack with a full implementation in the Sage programming language, discussed in Section 5.6. The implementation is available as supplementary material.

5.2.1 Structure of the chapter

We present our notation, the white-box framework, and structural cryptanalysis in Section 5.3. In Section 5.4, we show an attack on protocols white-boxing ciphers based on S-boxes using implicit implementations. Our main result is in Section 5.5, where we show an attack on implicit white-box implementations of ciphers based on modular additions. The techniques of polynomial factorization used as a subroutine of our attacks are detailed in Section 5.7.

5.3 Preliminaries

5.3.1 Notation

Boolean vectors. $\mathbb{F}_2$ denotes the field with two elements. Given a vector $v \in \mathbb{F}_2^n$, its i-th coordinate is denoted by $v[i]$, for $1 \leq i \leq n$. We may write $v = (v[1], \ldots, v[n])$. The dot product between two vectors $v, w \in \mathbb{F}_2^n$ is denoted by $\langle v|w\rangle = \sum_{i=1}^n v[i]w[i]$. The concatentation of two vectors $x, y \in \mathbb{F}_2^n$ is written $x \parallel y \in \mathbb{F}_2^{2n}$. Let $e_1, \ldots, e_n$ denote the canonical basis of $\mathbb{F}_2^n$; that is, $e_i[i] = 1$ and $e_i[j] = 0$ for $j \neq i$. The vector space spanned by a set of vectors $v_1, \ldots, v_k \in \mathbb{F}_2^n$ is called the *span* of those vectors, written $\mathsf{span}(v_1, \ldots, v_k)$. Addition over $\mathbb{F}_2^n$ will simply denoted by $+$. Modular addition over $\mathbb{Z}/k\mathbb{Z}$ will be written as $\boxplus$.

Affine subspaces. An *affine subspace* of $\mathbb{F}_2^n$ is a subset of $\mathbb{F}_2^n$ of the form $v+S = \{v+s : s \in S\}$, where $v \in \mathbb{F}_2^n$ is an arbitrary vector, and $S \subseteq \mathbb{F}_2^n$ is a linear subspace. An *affine map* $\mathbb{F}^m \to \mathbb{F}^n$ is a map of the form $x \mapsto f(x) + v$, where $v \in \mathbb{F}_2^n$ is an arbitrary vector, and $f : \mathbb{F}^m \to \mathbb{F}^n$ is linear. We will sometimes say that certain objects are affine: for example, affine layers, affine encodings. This means that the relevant object is an affine map.

Boolean functions. A Boolean function is a map $F : \mathbb{F}_2^n \to \mathbb{F}_2$. Recall that any Boolean function is functionally equivalent to a polynomial P in $\mathbb{F}_2[X_1, \ldots, X_n]$. That is, for all $x \in \mathbb{F}_2^n$, $F(x)$ is equal to the evaluation of P where X_i is substituted with $x[i]$. The polynomial P can be chosen to be square-free: that is, it is a linear combination of monomials of the form $X_1^{\varepsilon_1} \cdot \cdots \cdot X_n^{\varepsilon_n}$ with $\varepsilon_i \in \{0, 1\}$. In the remainder, in accordance with the literature on algebraic attacks, we only consider square-free polynomials, which are identified with their class in $\mathbb{F}_2[X_1, \ldots, X_n]/\langle X_i^2 - X_i \rangle$. Note that the square-free polynomial associated with F is unique, *i.e.* the space of Boolean functions $\mathbb{F}_2^n \to \mathbb{F}_2$ is isomorphic to the space of square-free polynomials $\mathbb{F}_2[X_1, \ldots, X_n]/\langle X_i^2 - X_i \rangle$.

Algebraic degree. The *algebraic degree* of a function F is the degree of the associated (square-free) polynomial. For a k-bit function $F : \mathbb{F}_2^n \to \mathbb{F}_2^k$, the algebraic degree of F is by convention the maximum of the algebraic degrees of the bit functions $F[i] : \mathbb{F}_2^n \to \mathbb{F}_2$, $1 \leq i \leq k$. Throughout this chapter, the term *degree* exclusively refers to the algebraic degree. The degree of F will be denoted by $\deg(F)$.

5.3.2 Round encodings

Consider a block cipher E composed of r rounds:

$$E = E_r \circ \cdots \circ E_1.$$

In most block ciphers, the round key is added to the inner state. In that case, given black-box access to a round function E_i, it is trivial to recover the round key. Indeed, all round operations are public and invertible, save for key addition. For that reason, white-box compilers typically attempt to obfuscate each round function by composing E_i with a so-called *input encoding* I_i, and *output encoding* O_i. The *encoded round function* can be expressed as:

$$\overline{E_i} = O_i \circ E_i \circ I_i.$$

In a typical white-box scheme, consecutive encodings cancel out, that is: $I_{i+1} \circ O_i = \mathsf{Id}$.

The white-box implementation of the block cipher E only gives access to the encoded round functions $\overline{E_1}, \ldots, \overline{E_r}$. Taking advantage of the cancellation property $I_{i+1} \circ O_i = \mathsf{Id}$, a honest user can compute:

$$\begin{aligned} \overline{E_r} \circ \cdots \circ \overline{E_1}(x) &= O_r \circ E_r \circ E_{r-1} \circ \cdots \circ E_1 \circ I_1(x) \\ &= O_r \circ E \circ I_1(x). \end{aligned}$$

If O_r and I_r are not the identity, the previous computation is not equivalent to an encryption of x. In most white-box schemes, it is accepted that the input x is encoded as $x = I_1^{-1}(p)$, where p is the plaintext the user wishes to encrypt. Likewise, the output of the encryption is encoded with O_r. Non-trivial I_1 and O_r are called *external encodings*.

Implicit representations. In the actual implementation of the white-box block cipher, the round function may be represented in various ways. In the original CEJO framework, the white-box implementation of $\overline{E_i}$ consists of a series of lookup tables. Given an input x to the encoded round function, the user can compute its output $\overline{E_i}(x)$ directly using those tables (combining them in a predefined way). The recent work of Ranea *et al.* introduces a different representation of the round function, which they call *implicit* representations.

Implicit representations of the round function provide a series of equations that the user can solve to compute the round function in the forward direction. That is, given an input x, the user computes its output $y = \overline{E_i}(x)$ by solving an equation system $f(x, y)$ in y. The description of the round function consists of a description of the equation system f. The point of implicit representations is that it allows Ranea et al. to build, for the first time: (1) white-box implementations with complex non-linear input encodings; and (2) white-box implementations of ARX ciphers.

Nevertheless, all attacks in this chapter are oblivious to how the encoded round function is represented—whether it is an implicit representation or otherwise. This is because our attacks will only require *black-box* access to the encoded round function. That is, given x and some arbitrary representation of $\overline{E_i}$, the only thing we need is that it is possible to efficiently compute $\overline{E_i}(x)$. Note that it is necessary for the honest user to be able to perform those computations, so this requirement is essentially free. All white-box implementations that we know of satisfy it. To stress this point once more: attacks in this chapter do not depend on the type of representation. For that reason, we omit a full description of implicit representations, and refer the curious reader to [RVP22].

5.3.3 Structural Cryptanalysis

Structural cryptanalysis was introduced by Biryukov and Shamir [BS01]. In the setting of structural cryptanalysis, an attacker is given oracle access to a function F. The attacker is able to query the oracle on input x, and gets as output $F(x)$.

Assume the function F has some known structure, but the components of that structure are unknown. For example, $F : \mathbb{F}_2^n \to \mathbb{F}_2^n$ is known to be composed of an n-bit affine layer A_1, followed by a layer S of k parallel s-bit S-boxes with $n = ks$, followed again by an n-bit affine layer A_2. We can write $F = A_2 \circ S \circ A_1$: this is the so-called ASA structure.

Given only *black-box* access to $F(x)$ via the oracle, the goal of the attacker is to find the hidden components A_1, S, A_2. More precisely, since F may not uniquely define A_1, S, A_2, the goal of the attacker is to find k s-bit S-boxes forming a layer S', and two affine layers A'_1, A'_2, such that $F = A'_2 \circ S' \circ A'_1$.

The previous example corresponds to the *ASA* structure, which stands for "Affine $\to$ S-box layer $\to$ Affine". The original article by Biryukov and Shamir presents the structural cryptanalysis of the SASAS structure, including as a sub-component, the structural cryptanalysis of SAS [BS01]. Later on, the ASASA structure was introduced in [BBK14], and a structural cryptanalysis of ASASA was given in [MDFK18]. Intuitively, the structural cryptanalysis of SAS, SASAS and ASASA show that it is exceedingly difficult to "hide" a layer of S-boxes using linear or affine encodings. In fact, even though many of the earlier attempts to build white-box schemes were originally broken in an ad-hoc fashion, it was noted in [DFLM18] that they can be broken generically via the structural cryptanalysis of ASA.

In this work, we extend structural cryptanalysis to non-linear encodings. Non-linear encodings will be represented by the letter I, for higher-order layer. (We have favored a vowel, for ease of pronunciation.) If arbitrary non-linear encodings are allowed, structural cryptanalysis becomes vacuous. To illustrate that point, consider any specific structure Σ followed by an arbitrary non-linear encoding I. Then the mapping $F = I \circ \Sigma$ contains no information about Σ. Indeed, I can be equal to $F' \circ \Sigma^{-1}$ for any arbitrary F'. In practice, existing white-box proposals only use low-degree non-linear encodings. Accordingly, in this work, the letter I should be understood as standing for a low-degree nonlinear layer.

In practice, we will often assume that any non-linear layer I is of lower degree than its surrounding layers.

Another subtlety worth noting is that we only assume that I is low-degree in the *forward* direction. That is, $x \mapsto I(x)$ is of low algebraic degree. The inverse $x \mapsto I^{-1}(x)$ may be of high degree.

Since our attacks also target ARX ciphers, we introduce the letter M to stand for **m**odular addition. The map $M : (\mathbb{F}_{2^n})^2 \to (\mathbb{F}_{2^n})^2$ is defined by $M : (x, y) \mapsto (x, x \boxplus y)$, where $\boxplus$ denotes modular addition. Since surrounding layers are defined over $\mathbb{F}_2^{2n}$, a standard little-endian conversion between $\mathbb{F}_{2^n}$ and $\mathbb{F}_2^n$ will implicitly be assumed throughout this work: we will treat quantities in $\mathbb{F}_{2^n}$ and $\mathbb{F}_2^n$ as if they belong to the same space. The authors of [RVP22] use the "permuted addition" mapping $M' : (x, y) \mapsto (x \boxplus y, y)$ instead of M. This makes no difference, because the two mappings M and M' are equivalent up to permutations of their inputs and outputs, and permutations are linear maps: the structural cryptanalysis of IMA is equivalent to that of IM′A.

Remark. In this work, all mappings within a structure are implicitly assumed to be invertible. This is because we target block ciphers, which are invertible by definition. Nevertheless, the invertibility assumption is mainly for convenience: most attacks can be adapted to dispense with it.

5.3.4 Cube attacks

Cube attacks were introduced by Dinur and Shamir [DS09]. We briefly recall the key lemma behind cube attacks, which will be useful in this work.

A starting observation is that for any non-zero $F : \mathbb{F}_2^n \to \mathbb{F}_2$ and any non-zero constant $\delta \in \mathbb{F}_2^n$:

$$\deg(x \mapsto F(x) + F(x + \delta)) = \deg(F) - 1,$$

with the convention $\deg(0) = -1$. Intuitively, the operation $x \mapsto F(x) + F(x + \delta)$ may be thought of as a differential of F along δ, and affects the degree similarly.

Differentiating F along multiple linearly independent vectors $\delta_1, \ldots, \delta_k$ amounts to summing F over the space spanned by the δ_i's. Since differentiating along each new δ_i decreases the degree by one, we obtain the key lemma of cube attacks (see [DS09] for a more detailed proof).

Lemma 5.3.1. *Let $F : \mathbb{F}_2^n \to \mathbb{F}_2$, and let E be an affine subspace of $\mathbb{F}_2^n$ of dimension $\deg(F) + 1$. Then:*

$$\sum_{x \in E} F(x) = 0.$$

In line with the literature on cube attacks, an affine subspace of $\mathbb{F}_2^n$ will sometimes be called a *cube*. Lemma 5.3.1 states that any function of degree d sums to zero over a cube of dimension $d + 1$.

5.3.5 Linearization

In the course of this chapter, we will be repeatedly confronted with the problem of solving non-linear equation systems over $\mathbb{F}_2$. This is an NP-complete problem even for quadratic systems, and average-case instances are believed to be intractable. Generic algorithms

(such as Gröbner bases [FP06], or approaches based on the polynomial method [Din21]) yield exponential-time algorithms for the average case.

Nevertheless, when the equations are of degree bounded by d over n unknowns, and the number of available equations is $\Omega(n^d)$, the system can be solved in polynomial time $\mathcal{O}(n^{3d})$ using the *linearization* technique. In effect, solving non-linear systems is only difficult when the number of available equations is limited. Throughout this work, we will use the strategy of reducing algorithmic problems to solving non-linear systems, but always in settings where we can freely sample a number of equations exponential in n; in particular, linearization will be applicable.

For the sake of completeness, we briefly sketch the linearization process. A more detailed treatment may be found in [AG11, MDFK18]. Assume we have a system of m independent equations of degree (at most) d over n unknowns $x_1, \ldots, x_n$. As discussed in Section 5.3.1, without loss of generality, we can restrict our attention to equations with square-free monomials, *i.e.* all monomials are of the form $x_1^{\varepsilon_1} \cdot \cdots \cdot x_n^{\varepsilon_n}$ with $\varepsilon \in \mathbb{F}_2^n$. The fact that the monomials are of degree at most d translates to the fact that at most d of the ε_i's are non-zero. Hence, the number of possible monomials is bounded by:

$$\sum_{i=0}^{d} \binom{n}{i} \leq \left(\frac{ne}{d}\right)^d = \mathcal{O}(n^d).$$

For each ε, replace all occurrences of the monomial $x_1^{\varepsilon_1} \cdot \cdots \cdot x_n^{\varepsilon_n}$ in the equation system by a new unknown t_ε. There are $\mathcal{O}(n^d)$ such unknowns t_ε, and the equation system is now linear in the t_ε's. Hence, we can solve the system using any linear system solving algorithm. If we use Gaussian elimination, the time complexity of this step is $\mathcal{O}(n^{3d})$. We obtain the values of all t_ε's.

It remains to deduce the values of the original variables x_i's from the t_ε's. Recall that $e_i \in \mathbb{F}_2^n$ is defined by $e_i[i] = 1$ and $e_i[j] = 0$ for $j \neq i$. If all t_{e_i} appear in the equation system, recovering the x_i's is trivial, because $x_i = t_{e_i}$. If the equation system has specific properties such that the t_{e_i}'s do not appear in the equations (for example, if the equations are homogeneous of degree d), some extra work is needed. Nonetheless, in practice, this step incurs a negligible cost in time complexity compared to linear system solving in the previous step. The reader is referred to [AG11, MDFK18] for detailed algorithms.

5.4 Structural cryptanalysis of ISA

In [RVP22], the authors introduce implicit representations of encoded round functions. This allows the authors the freedom to use non-linear input encodings, but for technical reasons, output encodings must be affine (see [RVP22] for the reasoning). In this section, we show that this approach cannot hide an inner S-box layer. To that end, we give a structural cryptanalysis of the ISA structure.

5.4.1 Problem statement

Let $I : \mathbb{F}_2^n \to \mathbb{F}_2^n$ be a non-linear map of degree d. Let S be a layer of k (not necessarily identical) s-bit S-boxes, with $n = ks$. For simplicity, S-boxes are assumed to be of maximal degree $s - 1$. (The attack can easily be adapted otherwise, but the cost of the attack grows with s, so $s - 1$ is the worst case.) Let $A : \mathbb{F}_2^n \to \mathbb{F}_2^n$ be an affine mapping. Let $F = A \circ S \circ I$. The goal of the attacker is to find a non-linear layer I', an affine layer A', and a layer S' of k parallel s-bit S-boxes such that $F = A' \circ S' \circ I'$.

5.4.2 Equivalent keys

Since F does not uniquely define the triple (I, S, A), the goal of the attacker is to find a triple (I', S', A') such that $F = A' \circ S' \circ I'$, as stated above. Such a triple is called an *equivalent key*. A few remarks about equivalent keys will prove helpful for the cryptanalysis of ISA.

First, we can freely assume that A' is a linear layer, rather than an affine layer. To see this, take an arbitrary equivalent key (I, S, A) with an affine layer $A(x) = L(x) + c$, for some linear map $L : \mathbb{F}_2^n \to \mathbb{F}_2^n$ and constant $c \in \mathbb{F}_2^n$. Then the triple (I, S', L) with $S'(x) = S(x) + L^{-1}(c)$ is also an equivalent key. This follows from the fact that both triple compute the same function, and that S' can still be represented as k parallel s-bit S-boxes. In the remainder, we assume without loss of generality that A is linear.

In fact, the last linear layer A is only defined as far as determining the output space of each S-box. More precisely, let $V_i = \mathsf{span}(e_{s(i-1)+1}, \ldots, e_{s(i-1)+s})$ be the vector space spanned by the bits of the i-th S-box. Let L_i be an arbitrary invertible linear map $V_i \to V_i$. Given an equivalent key (I, S, A), we can freely compose the output of the i-th S-box with L_i, and compose A with L_i^{-1}, to obtain another equivalent key. Thus, we cannot hope to recover more information from A than the output space $A(V_i)$ of each S-box through A.

Conversely, if we know the spaces $A(V_i)$, then for each $A(V_i)$, we can arbitrarily pick a basis of the space, then assign the k-th bit of the i-th S-box to the k-th element of that basis. Letting L' be the map computed in that manner, $L^{-1} \circ F = (L^{-1}A) \circ S \circ I$ has the IS structure, so we have effectively peeled off the linear layer A. Indeed, $L^{-1}A$ operates independently on each V_i, and hence can be folded into the S-box layer.

In conclusion, in order to peel off the A layer, it is necessary and sufficient to recover the output spaces $A(V_i)$ of each S-box through A. Note that the spaces $A(V_i)$ can be recovered in any order. Indeed, given an equivalent key (I, S, A), we are free to permute the S-boxes, up to applying the inverse permutation to I and A (which does not affect their degree).

5.4.3 Peeling off the A layer: from ISA to IS

Our approach is similar to [MDFK18]. Consider two bits b_1 and b_2 at the output of two *distinct* S-boxes in the mapping $S \circ I$. When viewed as a function of the input x of $S \circ I$, each of those bits is of degree at most $d(s-1)$. Hence their product $b_1 b_2$ is of degree at most $2d(s-1)$. On the other hand, let $b_{1'}$ denote a bit at the output of the *same* S-box as b_1. Then by [MDFK18, Lemma 2], the product $b_1 b_{1'}$ is of degree only at most $d(s-1)$.

The idea of the attack is to detect this difference in degree through the last affine layer A. For that purpose, for $i = 1, 2$, define $\lambda_i \in \mathbb{F}_2^n$ such that:

$$b_i(x) = \sum_{j=1}^{n} \lambda_i[j] F(x)[j] = \langle \lambda_i | F(x) \rangle.$$

Such λ_i's necessarily exist, because b_1 and b_2 are bits at the output of $S \circ I = A^{-1} \circ F$, and A^{-1} is linear (because we have assumed that A is linear).

By Lemma 5.3.1, the fact that $b_1 b_2$ is of degree at most $d(s-1)$ implies that $b_1 b_2$ sums to zero over any cube of dimension $d(s-1)+1$. Let C be such a cube. We obtain the equation:

$$\sum_{x \in C} \langle \lambda_1 | F(x) \rangle \cdot \langle \lambda_2 | F(x) \rangle = 0. \tag{5.1}$$

Note that C can be freely chosen by the attacker. Hence, all quantities $F(x)$'s in (5.1) can be computed by querying the F oracle. The only unknowns in the equation are the $\lambda_i[j]$'s.

Observe that (5.1) is an equation of degree 2 over the $\lambda_i[j]$'s. We can sample one new equation for each choice of cube C of dimension $d(s-1)$. Provided $d(s-1) < n$, there are many more choices of cubes than quadratic monomials, hence the system can be solved by linearization (cf. Section 5.3.5).

Solving the system yields pairs of linear masks λ'_1, λ'_2 such that $\langle \lambda'_1 | F(x) \rangle$ and $\langle \lambda'_2 | F(x) \rangle$ are equal to two bits (or linear combination of bits) at the output of the *same* S-box. Taking one such solution λ'_1, λ'_2, fixing $\lambda_1 = \lambda'_1$ in (5.1), and solving for λ_2, we can recover the linear space spanned by all other bits at the output of the same S-box. In the notation of Section 5.4.2, we recover the output space $A(V_i)$ of one S-box.

By repeating the same process, we eventually recover the outputs spaces $A(V_i)$ for $1 \leq i \leq k$, *i.e.* we obtain the output space of all S-boxes. Following the discussion in Section 5.4.2, this is enough to peel off the last linear A of F. We are left with a new structural cryptanalysis problem of the form $F' = S \circ I$.

5.4.4 Recovering the remaining layers: structural cryptanalysis of IS

In this section, we are given $F = S \circ I$, with the promise that S is a layer of k parallel s-bit S-boxes, and $I : \mathbb{F}_2^n \to \mathbb{F}_2^n$ is of degree at most d. The goal is to find S', I' with the same features as S, I, such that $F = S' \circ I'$.

Let S_i denote the i-th S-box in the layer S. Since we are able to observe separately the outputs of each S-box S_i, let F_i denote the restriction of F to the bits at the output of the i-th S-box. That is, $F_i = (F[(s-1)i+1], \ldots, F[(s-1)i+s])$. Likewise, let I_i denote restriction of I to the i-th S-box. Then $F_i = S_i \circ I_i$. Observe that recovering an equivalent key S', I' for F is exactly equivalent to recovering $S'_i : \mathbb{F}_2^s \to \mathbb{F}_2^s$ and $I'_i : \mathbb{F}_2^n \to \mathbb{F}_2^s$ of degree at most d such that $F_i = S'_i \circ I'_i$, for each S-box separately. As a consequence, from now on, we restrict our attention to finding S'_1, I'_1 given black-box access to F_1. To do so, we borrow an idea from [BS01].

For each $x \in \mathbb{F}_2^s$, let $t_x = S^{-1}(x)$. We view the t_x's as unknown. Since I is of degree at most d, by Lemma 5.3.1, for any cube C of dimension $d+1$, we get an equation:

$$\sum_{x \in C} t_{F(x)} = 0.$$

Thus, by sampling sufficiently many cubes, we obtain a linear system of equations on the t_x's, which we can solve to obtain S'_1 [3]. We can then let $I'_1 = (S'_1)^{-1} \circ F_1$.

5.4.5 Attack complexity

The complexity of the attack is determined by the number of bits n of the cipher's internal state, the number of bits s of the S-boxes, and the degree d of the input encoding. In parameter regimes that correspond to typical real-world scenarios (e.g. AES parameters), the complexity bottleneck is to solve Equation (5.1) (Section 5.4.3). This amounts to

[3] Note that any S'_1 of the form $S_1 \circ B$ for affine B is a solution. One may wonder if the attack might recover other solutions, but in practical experiments, this does not appear to be the case [BS01].

solving a linear system in $\mathcal{O}(n^2)$ unknowns over $\mathbb{F}_2$. This costs roughly $\mathcal{O}(n^6)$ operations (using Gaussian elimination for simplicity; the exponent can be reduced using more advanced matrix inversion algorithms). In practice, note that this quantity counts the number of *bit* operations, whereas linear algebra over $\mathbb{F}_2$ can naturally take advantage of word-level operations, resulting in a lower complexity. A secondary bottleneck is to collect $\mathcal{O}(n^2)$ cubes to derive the equations of the previous system. Each cube is of dimension $d(s-1)$, resulting in a total of $2^{d(s-1)}$ calls to the encoded round function. Both computation must be performed for each bit. In the end, the total complexity is $\mathcal{O}(n^7 + n^2 2^{d(s-1)})$. Observe that for AES parameters and reasonable d (say, $d < 5$), the first term $\mathcal{O}(n^7)$ dominates: linear system solving is the bottleneck.

5.4.6 Application to white-box key recovery

The attacks presented in previous sections belong to the family of structural attacks. The output of the attack is an equivalent key, in the sense of structural attacks: that is, for an ISA structure, suitable non-linear layer I', S-box layer S', and affine layer A', such that the function F under attack satisfies $F = A' \circ S' \circ I'$. When applying this technique to attack a white-box encoded round function with the appropriate structure (*e.g.* ISA), a *white-box* attacker (as opposed to a structural attacker) would like to go one step further, and recover the encryption key.

That goal is made possible by the fact that the white-box attacker knows more information than the structural attacker. Contrary to the structural attacker, the white-box attacker knows the full description of the round function, including specific descriptions of the S-boxes and linear layer. The only information the white-box attacker does not have are the round encodings, and the round keys. Compared to the structural attacker, the white-box attacker also has a more specific goal, which is to recover the round keys (and from there the overall encryption key).

This section is to bridge the gap between the two models, and show how a key-recovery attack in the white-box model can be deduced from a structural attack.

Reduction to a simple problem

At the outcome of the structural attacks from Section 5.4.4, the attacker has access to the input and output of every S-box, up to affine equivalence. More precisely, for each S-box S appearing in a round function of the underlying block cipher, the attacker can read the input and output of the S-box up to fixed affine encodings (fixed in the sense that they are always the same for a given S-box; they need not be the same between different S-boxes). Note that S-boxes belonging to the same round may also be permuted, in the sense that the structural attack may not present the S-boxes in the same order that they would be presented in a traditional description of the underlying cipher. In addition to reading the input and output of S-boxes up to affine equivalence, the attacker may also choose to modify those values at will, and observe the effect on later encryption rounds.

Thus, building a white-box attacker from a structural attacker can be neatly reduced to a stand-alone problem: given full read and write access to the input and output of every S-box in a particular SPN cipher, up to affine equivalence for each S-box, and up to permutation of the S-boxes in a given round, recover the round keys. (If the key schedule is invertible, recovering the round keys is equivalent to recovering the encryption key.)

Solving the problem

We now explain how to solve the problem outlined just above. We note that it is not particularly difficult. Intuitively, this is because the "heavy lifting" was done by the structural attack: deducing a white-box key-recovery attack from the structural attack is relatively straightforward.

Let S_i be a particular S-box recovered by the structural attack. The white-box attacker knows the actual S-box S_{orig} in the original SPN block cipher (assuming for simplicity that all S-boxes are the same). Hence, S_i and S_{orig} are affine-equivalent: there exist affine maps A_i, B_i such that $S_i = B_i \circ S_{\text{orig}} \circ A_i$. Given as input S_i and S_{orig}, recovering suitable A_i, B_i is the so-called *affine equivalence problem*. Efficient algorithms for typical S-box sizes are given in [BCBP03, Din18].

The affine equivalence problem may have more than one solution: there may exist $(A_i, B_i) \neq (A'_i, B'_i)$ such that $S_i = B_i \circ S_{\text{orig}} \circ A_i = B'_i \circ S_{\text{orig}} \circ A'_i$. In that case, we have:

$$S_{\text{orig}} = (B_i^{-1} \circ B'_i) \circ S_{\text{orig}} \circ (A'_i \circ A_i^{-1}).$$

The pair $(B_i^{-1} \circ B'_i, A'_i \circ A_i^{-1})$ is said to be an *affine self-equivalence* of S_{orig}.

If S_{orig} admits only the trivial self-equivalence $(\mathsf{Id}, \mathsf{Id})$, the affine equivalence problem has a single solution. After computing that solution for each S-box using the efficient algorithms from [BCBP03, Din18], the attacker has (read and write) access to the *plaintext* input and output of every S-box. The only remaining unknown is the permutation of the S-boxes in a given round. At this stage, recovering the round key for any real-world cipher is trivial.[4]

For an S-box sampled uniformly from permutations on a given number of bits, with high probability, there is no self-equivalence beside the trivial one, and we are done. However, we note that specific ciphers may use S-boxes with special non-random features, whose self-equivalence group may be non-trivial. This is in particular the case of the AES S-box, which is affine-equivalent to modular inversion in $\mathbb{F}_{2^8}$, hence has as many self-equivalences as modular inversion. Concretely, the AES S-box has $2040 \approx 2^{11}$ self-equivalences [BCBP03]. Nevertheless, recovering the encryption key from read-write access to all S-boxes up to self-equivalence is still an easy problem. An algorithm for that purpose is given in [DFLM18, Section 4.2].

5.5 Structural cryptanalysis of IMA

As we have seen in the previous section, the introduction of implicit representations of round functions in [RVP22] makes it possible to create non-linear input encodings for SPN cipher: the ISA structure. Another benefit of implicit representations, and their most novel application, is that they make it possible to apply white-box techniques to ARX ciphers, which was not possible before. More precisely, the authors show how to hide one layer of modular addition, which is the non-linear building brick of ARX ciphers. (For clarity, "linearity" here should be understood as linearity over $\mathbb{F}_2^n$; modular addition in $\mathbb{Z}/k\mathbb{Z}$ is of course $\mathbb{Z}/k\mathbb{Z}$-linear.)

[4] We do not propose a way to do so generically here. This is because the permutation symmetry between S-boxes might be broken only by the key schedule. Since the key schedule of a block cipher can take an arbitrary form, it does not lend itself to a generic structural analysis. On the other hand, for any concrete block cipher, if the cipher leaks plaintext values at the outcome of every S-boxes layer in every round, up to permutation of S-boxes, we contend that breaking the cipher is quite easy.

As earlier, the main limitation remains that only input encodings can be non-linear; the techniques of [RVP22] do not support non-linear output encodings. The construction of [RVP22] thus amount to an IMA structure. In this section, we give a structural cryptanalysis of IMA. The attack we present also applies to AMA. This shows that the use of a low-degree input encoding is not enough to hide an inner modular addition. At the end of the section, we discuss how the structural cryptanalysis result translates to a white-box key-recovery attack on white-box ARX ciphers.

5.5.1 Problem statement

Let $I : \mathbb{F}_2^n \to \mathbb{F}_2^n$ be a non-linear map of degree d. We allow the case $d = 1$ in the attack, *i.e.* the attack also applies to affine input layers. Let $M : (x, y) \mapsto (x, x \boxplus y)$ be the modular addition addition, as discussed in Section 5.3.3.

Let $A : \mathbb{F}_2^n \to \mathbb{F}_2^n$ be an affine mapping. Let $F = A \circ M \circ I$. The goal of the attacker is to find a non-linear layer I' of degree at most d, and an affine layer A', such that $F = A' \circ M \circ I'$.

5.5.2 Equivalent keys

Our algorithm for the structural cryptanalysis of IMA will sometimes make guesses. For instance, at the end of Section 5.5.3, the algorithm will identify a few possible candidates for the map $I[2n-1]$. In those cases, the algorithm will simply guess that one of the candidates is the correct one. The attack continues as if the guess was correct. At some later points in the attack, the algorithm will test that certain vector spaces have the right dimension. If that is not the case, it reveals that one of the previous guesses was wrong. The algorithm then simply backtracks to the latest guess for which not all possibilities have been exhausted, and takes a different guess.

It is important to note that whenever the algorithm reaches a branching point (*i.e.* a guess), it is often the case that multiple choices will be correct. This is because the solution to the structural analysis problem is not unique: modular addition admits many affine self-equivalences [RVP22]. Thus, our algorithm has two types of branching points. Some branching points will arise from the fact that there are multiple solutions: all choice are equally correct, and will not lead to any backtracking. Other branching points will only have one correct choice, and will lead to backtracking if the guess was incorrect. Our experiments show that the same branching point in the algorithm can be of either type, depending on the problem instance and previous choices. This is because equivalent keys for modular addition have a rather complex structure: see [RVP22, Section 5.2]. Interestingly however, our algorithm does not care about the exact structure of equivalent keys, or about which branching point is of which type: it simply makes guesses when necessary, and backtracks when it encounters an inconsistency. In practical experiments, incorrect choices lead to inconsistencies quickly (typically at the next consistency check), so the performance cost of wrong choices is low.

5.5.3 Peeling off the I layer: from IMA to MA

Step 1: recovering affine bits

A starting point is that most bits at the output of the modular addition $M : (x, y) \mapsto (x, x \boxplus y)$ are affine in the input of M: namely, the bits of x, and the last bit $(x \boxplus y)[n]$ of

$x \boxplus y$, which is equal to $x[n] + y[n]$. (Recall that modular addition is denoted by $\boxplus$, while $+$ denotes addition over $\mathbb{F}_2$.) The image of the space of affine bits through the outer affine layer A can be detected using the fact that those bits have degree d in $F = A \circ M \circ I$, whereas other bits have higher degree.

To do so, take any cube C of degree $d+1$ over the input of F. Suppose $\lambda_0 \in \mathbb{F}_2^{2n}$ is such that $\langle \lambda_0 | F \rangle$ is equal to a linear combination of affine bits before the last affine layer (formally: $\langle \lambda_0 | F \rangle = \langle \mu_0 | M \circ I \rangle + \gamma_0$ where γ_0 is a constant and $\mu_0[i] = 0$ for $i \in [n+1, \ldots, 2n-1]$). Then the following equation holds:

$$\sum_{c \in C} \langle \lambda_0 | F(c) \rangle = 0 \quad \text{whenever } \dim(C) = 2. \tag{5.2}$$

Equation (5.2) yields a linear constraint on λ_0, viewed as a vector of binary unknowns $\lambda_0[1], \ldots, \lambda_0[2n]$. Each choice of C yields an equation. Sampling sufficiently many C's allows us to solve the system and recover λ_0. More exactly, since λ_0 is not unique, we recover the space S_0 of dimension $n+1$ of all λ_0's that correspond to affine bits of M, in the sense given earlier.

Step 2: recovering $(x \boxplus y)[n-1]$

The only bit of degree 2 in M is $M[2n-1]$, *i.e.* the $n-1$-th bit of $x \boxplus y$. Hence the only bit of degree $2d$ in $M \circ I$ is $(M \circ I)[2n-1]$. If λ_1 now denotes the linear mask corresponding to $M[2n-1]$ through the last affine layer A (that is: $\langle \lambda_1 | F \rangle = (M \circ I)[2n-1] + \gamma_1$ where γ_1 is a constant), then λ_1 satisfies the following equation:

$$\sum_{c \in C} \langle \lambda_1 | F(c) \rangle = 0 \quad \text{whenever } \dim(C) = 2d+1. \tag{5.3}$$

As before, each C yields an equation. By sampling sufficiently many C's, we can solve the system. However, the system is not satisfied only by λ_1: all masks λ_0 recovered in Section 5.5.3 also satisfy it. Solving (5.3) yields a solution space S_1 of dimension $n+2$, spanned by S_0 and λ_1. (This, and other steps of the attacks, has been confirmed experimentally). For now, to recover λ_1, we simply sample an element of $S_1 \backslash S_0$. This can be done by taking the non-zero element of the quotient $S_1/S_0 \sim \mathbb{F}_2$, and lifting it back to S_1. (Alternatively, because half of the elements of S_1 are in S_0, sampling a uniformly random element of S_1, and testing that it does not belong to S_0, succeeds within two tries in expectation.)

Note that the previous computation does not "truly" recover λ_1; rather, it recovers $\lambda_1' = \lambda_1 + \lambda_0$ for some $\lambda_0 \in S_0$. This is good enough for our next computation. We have:

$$\langle \lambda_1' | F \rangle = \langle \lambda_1 | F \rangle + \langle \lambda_0 | F \rangle = (M \circ I)[2n-1] + \langle \mu_0 | M \circ I \rangle + \gamma_1$$

where $\langle \mu_0 | M \rangle$ is linear, while $M[2n-1]$ has degree two. On the other hand, observe that:

$$M(x,y)[2n-1] = (x \boxplus y)[n-1] = x[n]y[n] + x[n-1] + y[n-1]$$

where the first term is the carry of the modular addition. Reinjecting into the previous computation, we get:

$$\langle \lambda_1' | F \rangle = I[n]I[2n] + I[n-1] + I[2n-1] + \langle \mu_0 | M \circ I \rangle + \gamma_1. \tag{5.4}$$

Let P denote the polynomial of degree $2d$ in $\mathbb{F}_2[X_1, \ldots, X_{2n}]/\langle X_i^2 - X_i \rangle$ coinciding with $\langle \lambda_1' | F \rangle$. Because λ_1' is known to the attacker, and d is small (typically, in [RVP22], $d = 2$), P can be computed explicitly by the attacker. Looking at the right-hand side of Equation (5.4), the only term of degree $2d$ is $I[n]I[2n]$: other terms are of degree at most d. Let B and C denote the polynomials of degree d corresponding to $I[n]$ and $I[2n]$ respectively. Let A denote the polynomial of degree d corresponding to the other terms of the expression. Then Equation (5.4) can be rewritten as:

$$P = A + B \cdot C \tag{5.5}$$

where A, B, C are of degree d. This leads to the following problem.

Factorization Problem. Given $P \in \mathbb{F}_2[X_1, \ldots, X_{2n}]/\langle X_i^2 - X_i \rangle$ such that $P = A + B \cdot C$, where $\deg(A) = \deg(B) = \deg(C) = d$, find A', B', C' with $\deg(A') = \deg(B') = \deg(C') = d$ such that $P = A' + B' \cdot C'$.

In other words, we want to solve a particular kind of structural cryptanalysis problem, for the structure $P = A + B \cdot C$. As is often the case with structural cryptanalysis, the solution is not unique: if (A, B, C) is a solution, then so is $(A + B, B, B + C)$, for instance. In general, any two linearly independent vectors in the linear span of B, C, 1 can be chosen for B' and C'. Unless P has a special structure, there are no other solutions (confirmed by experiments).

In fact, the factorization problem above was already considered in [MDFK18], in the special case $d = 2$. The authors of [MDFK18] give an efficient algorithm for that case. A polynomial-time algorithm for the case of arbitrary constant d can be found in [Bha14]. In fact, the work of [Bha14] applies to a much more general class of problems, and is quite complex as a result, and less efficient in practice. In Section 5.7, we generalize the algorithm from [MDFK18] for $d > 2$, which suffices for our purpose.

At the outcome of running either one of the previous algorithms on $\langle \lambda_1' | F \rangle$, we recover the linear span of $I[n]$, $I[2n]$, 1, viewed as polynomials in their inputs. Up to the contribution of 1 in that span, which only affects the constant in the polynomial, $I[n]$ can be recovered by observing that it also lies in the span of polynomials generated by elements of S_0. For $I[2n]$, we simply take one of the other two non-zero elements in the span of $I[n]$, $I[2n]$, which can be either $I[2n]$ or $I[2n] + I[n]$ (again, up to the constant 1). If we happened to pick $I[2n] + I[n]$, our guess was wrong. Nevertheless, we continue with the rest of the attack. As discussed in Section 5.5.2, the algorithm will backtrack to this point if it detects an inconsistency later on.

Step 3: recovering the $(x \boxplus y)[i]$'s

In the previous step, we isolated the penultimate bit of the modular addition $(x \boxplus y)[n-1]$, by using the fact that it is the only bit of degree two in M. This required using cubes of dimension $2d + 1$.

We then factorized the polynomial expression of that bit to obtain the polynomial expressions of $I[n]$ and $I[2n]$.

A simple approach would be to repeat this process: starting from the observation that $(x \boxplus y)[n - i]$ is the only bit of degree $i + 1$ in M, for $i > 1$, we could isolate each bit and somehow factorize it, as before. This runs into two issues. First, to isolate a bit of degree $i + 1$, we would need to sum over cubes of dimension $i + 2$. The number of summands grows exponentially with i, and i goes up to n, so this is not feasible. Second,

the polynomial expression of $(x \boxplus y)[n-1]$ is simple, which led to a simple factorization problem; but for $(x \boxplus y)[n-i]$, the polynomial expression becomes increasingly complex as i grows, and so would the associated factorization problem.

We use a different approach, based on the following observation. Consider for a moment the expression $x \boxplus y = z$ over $\mathbb{F}_2^n$. Let c_i denote the value of the carry coming into the $(n-i)$-th bit of $x \boxplus y$, that is:

$$(x \boxplus y)[n-i] = x[n-i] + y[n-i] + c_i.$$

The a straightforward computation shows that the following recursive relation holds:

$$c_i = x[n-i-1]y[n-i-1] + c_{i-1}(x[n-i-1] + y[n-i-1]).$$

Hence, if we know c_{i-1}, then we have:

$$(x \boxplus y)[n-i] = x[n-i-1]y[n-i-1] + c_{i-1}(x[n-i-1] + y[n-i-1]) + x[n-i] + y[n-i]. \quad (5.6)$$

Going back to our problem, we proceed by induction. For the i-th inductive stage, assume we have recovered $I[n-j]$ and $I[2n-j]$ for $j \leq i$. Note that Step 2 in Section 5.5.3 initialized that induction for $i = 0$. Then using our knowledge of $I[n-j]$ and $I[2n-j]$ for $j \leq i$, we know the inputs of the last $i+1$ inputs of the modular addition M in F. This means that for any given input v of F, we can compute the value of the carry $c_{i-1}(v)$ coming into $(M \circ I(v))[2n-i-1]$. Using (5.6), we can thus write:

$$(M \circ I(v))[2n-i] = B(v)C(v) + c_{i-1}(v)A(v) + D(v) \quad (5.7)$$

where A, B, C, D have algebraic degree d.

Equation (5.7) may seem difficult to exploit, since it depends on c_{i-1}, which may be of high degree. For that reason, we no longer rely on cubes, and propose a different technique. For $v \in \mathbb{F}_2^n$ and $S = \{s_1, \ldots, s_\ell\} \subseteq [1, n]$, let $v_S = v[s_1] \cdot \ldots v[s_\ell]$. For $k \leq n$, let $v^{\leq k}$ denote the vector of length $\sum_{i=0}^{k} \binom{n}{i}$ whose coordinates are equal to v_S for each $S \subseteq [1, n]$ of cardinality at most ℓ (with the convention $v_\varnothing = 1$). Observe that a map $f(v)$ is of algebraic degree at most k if and only if $f(v)$ is a linear combination of elements of $v^{\leq k}$. (Algebraically, we are moving into the tensor space $T^{\leq k}(\mathbb{F}_2^n)$, but we prefer to describe our algorithm in more concrete terms.) Applying this observation to A in (5.7), there exists λ_A such that $A(v) = \langle \lambda | v^{\leq d} \rangle$, and likewise for D, and for the product $B \cdot C$. Reinjecting into (5.7), we have:

$$(M \circ I(v))[2n-i] = \langle \lambda_{B \cdot C} | v^{\leq 2d} \rangle + \langle \lambda_A | c_{i-1}(v) \cdot v^{\leq d} \rangle + \langle \lambda_D | v^{\leq d} \rangle. \quad (5.8)$$

Given v, the attacker can compute all elements $v^{\leq 2d}$, $v^{\leq d}$, and $c_{i-1}(v) \cdot v^{\leq d}$ on the right-hand side of (5.8). Here, we use the fact that $c_{i-1}(v)$ can be computed for any v by induction hypothesis, discussed earlier. Letting λ_i be such that $\langle \lambda_i | F \rangle = (M \circ I(v))[2n-i]$, we have

$$\langle \lambda_i | F \rangle = \langle \lambda_{B \cdot C} | v^{\leq 2d} \rangle + \langle \lambda_A | c_{i-1}(v) \cdot v^{\leq d} \rangle + \langle \lambda_D | v^{\leq d} \rangle. \quad (5.9)$$

Each choice of v yields a new equation of this form. We obtain a linear system in the λ_*'s, viewed as vectors of binary unknowns. Without loss of generality, we set $\lambda_D = 0$, since its coefficients can be folded in $\lambda_{B \cdot C}$. Solving the system, and discarding the subspaces where either λ_A is zero, or the coefficients of degree higher than d in $\lambda_{B \cdot C}$ are zero, yields a solution. In fact, experiments show that the solution space is sometimes of dimension

2, even for valid solution, which requires a guess. On the other hand, if the solution space has dimension 0, it reveals that one of our previous guesses was wrong, and we backtrack to a previous guess, as discussed in Section 5.5.2.

At the outcome of this computation, we recover $\lambda_{B \cdot C}$, which reveals the coefficients of $B \cdot C + D$. Recovering the coefficients of B and C from there is a new instance of the factorization problem from Section 5.5.3, for the same degree d. Solving the problem yields the coefficients of B and C, *i.e* it reveals the polynomial expressions of $I[n-i]$ and $I[2n-i]$. This sets us up for the next stage of the induction. Eventually, all $I[i]$ are recovered: we have peeled off the initial I layer.

Remark. Taking a step back, the use of the inductive definition of modular addition via carries allows the attack to only have to consider equations of degree at most $2d = \mathcal{O}(d)$, as opposed to $\mathcal{O}(nd)$ for a naive approach. The technique we use involving the vectors $v^{\leq k}$ can be viewed as a variant of the linearization technique from Section 5.3.5. In essence, what we show is that if we have equations that are of low degree with respect to some input v, except for some term in the equation (here, $c_{i-1}(v)$) that may be of high degree, but whose value can be computed by the attacker, then it is still possible to solve the equation system with a linearization-like approach. Generalizing this idea, while normal linearization techniques allow an attacker to solve a system of low degree, given enough equations, our variant allows to solve a system of equations that are low degree in v *and* in other quantities $f_i(v)$, as long as the f_i's are known—even if the equations may be of arbitrarily high degree when expressed using only v.

Step 4: wrapping up the attack

At the outcome of the previous step, we have removed the initial non-linear layer I, and are left with the structural cryptanalysis of MA. Contrary to the case of SA, the cryptanalysis of MA is trivial, since M is a fixed function. To recover A, it suffices to compose the function with M^{-1}.

5.5.4 Attack complexity

The complexity of the attack is determined by the number of bits n of the cipher's internal state, and the degree d of the input encoding. Similar to Section 5.4.5, the complexity bottleneck comes from solving Equation (5.9). This amounts to solving a linear system in $\binom{n}{2d} \leq n^{2d}$ unknowns over $\mathbb{F}_2$. Using Gaussian elimination for simplicity, this costs $\mathcal{O}(n^{6d})$ bit operations. Since this must be repeated for each of the $n/2$ bits targeted by the attack, the total complexity is $\mathcal{O}(n^{6d+1})$. To provide some perspective, note that simply representing an arbitrary input layer of degree d requires $\mathcal{O}(n^d)$ bits. (Implicit representations in [RVP22] are also of size $\mathcal{O}(n^d)$.)

5.5.5 Application to white-box key recovery

In a white-box key-recovery scenario, two aspects of the attack differ.

First, it is usually the case that the round function of an ARX cipher involves several modular additions. Our technique targets the case of a single modular addition. The Speck cipher contains a single modular addition, and our structural cryptanalysis directly applies. More generally, it is not clear how the implicit encoding technique of [RVP22] would perform for more complex round functions. It would likely be implemented

separately for each modular addition, which would again enable our attack. Even if it did not, we suspect that very similar techniques would apply. In any case, our structural cryptanalysis problem focuses on the simplest and cleanest case of a single modular addition. This also avoids becoming dependent on a specific round function.

A second point where the structural cryptanalysis approach differs from white-box key-recovery is that, as in Section 5.5.5, a white-box attack aims to recover a secret key, not just equivalent descriptions of inner encryption layers. In the case of modular addition, at the outcome of structural cryptanalysis, we recover an equivalent key. In practice, our attack recover the input and output of the modular addition layer *up to self-equivalences of the modular addition*. An (affine) self-equivalence of modular addition is a pair of affine layers (A, B) such that $M = B \circ M \circ A$. By executing the cipher on a given input, and using the output of our structurcal cryptanalysis, we can observe the values of bits after the modular addition layer, for every round, up to self-equivalence. If it was not up to self-equivalence, recovering round keys would be trivial, since all operations other than key addition are public and invertible.

Similar to the treatment in Section 5.5.5, applying the structural cryptanalysis result to white-box key recovery comes down to solving the following problem. Assume we have access to the full inner state of the cipher after every modular addition layer, but those inner states may be modified by an affine map arising from a self-equivalence (always the same for a given round). The goal is to recover the encryption key. The problem may appear difficult, because modular addition has many self-equivalences. But in fact, those self-equivalences mainly differ in the values of the first few and last few bits of the modular addition output $(x \boxplus y)$. For the vast majority of bits, namely $(x \boxplus y)[i]$ for $2 < i < n - 2$, the self-equivalent encodings behave like the identity matrix, up to the addition of a vector confined to a space of dimension 4 (see [RVP22, Section 5.2]). Recall that this information is available at every round. Recovering the secret key from there is not amenable to a general analysis, since it depends on the specific key schedule of a cipher. Nevertheless, we believe that for any concrete block cipher, it is not difficult to recover the round keys given that much leakage.

5.6 Practical validation

As noted in Section 5.4.3, our attacks on ISA are similar in spirit to the attack on ASASA in [MDFK18]. On the other hand, the attack on IMA is quite new, and relies on a backtracking mechanism whose practical behavior is not obvious. For that reason, we have decided to fully implement the IMA attack in the case $d = 1$, aiming especially to observe the behavior of the backtracking process. Our implementation is available in the appendix.

The implementation is in the Sage language (release 9.8), an open-source Python variant suited to formal computation. The implementation can be run simply with the command `sage full.sage`. It generates a random instance of the problem for n bits, where n can be specified in the `full.sage` file. We have run the program up to $n = 64$ bits. The attack always succeeds, and does so on a laptop in 95 minutes on inputs of size $n = 64$ bits.

Our main observation is that (to our surprise), the attack never needs to backtrack more than one step. To elaborate on that point, recall that the algorithm from Section 5.5 needs to make some choices when recovering each bit of the input layer, due to the fact that

the Factorization problem (in Section 5.5.3) has multiple solutions. What our experiments observe is that any time a choice is incorrect (i.e., the choice will not lead to a correct solution for the overall problem), an inconsistency is detected at the very next step of the computation. That is, the computation of the next bit immediately fails. As a consequence, the algorithm essentially follows a linear path, recovering the expression of one bit after the other. It never needs to revisit choices made for bits before the current one.

Because affine self-equivalences of the modular addition have a complex structure (see [RVP22, Section 5]), one might have expected that a non-trivial amount of backtracking may be necessary. It turns out that this is not the case. As a consequence, recovering the input (and output) encodings is much simpler than one might have expected. The main computational cost is to solve the linear system in each step, as discussed in Section 5.5.4.

5.7 Polynomial factorization

In this section, we focus on the factorization problem identified in Section 5.5.3. To solve the problem efficiently, we sketch a generalization of the polynomial factorization algorithms from [MDFK18, Section 5.3].

The problem is: given a polynomial $P = A + B \cdot C$, where P is known and A, B, C are of degree d, find A', B', C' of degree d such that $P = A' + B' \cdot C'$.

In [MDFK18], the problem is solved when $d = 2$. The solution can be generalized as long as d remains small, as we discuss in what follows. For simplicity, we first present the algorithm in the case $d = 3$, and discuss its generalization to $d > 3$ later on.

5.7.1 Notation

Let D be a degree-3 polynomial in $\mathcal{Q} = \mathbb{F}_2[X_0, \ldots, X_{n-1}]/\langle X_i^2 - X_i \rangle$. For (i, j, k) three pairwise distinct elements of $[n]^3$, let $D_{i,j,k}$ denote the coefficient of $X_i X_j X_k$ in D.

Let $D_{i,j,*}$ denote the vector $(D_{i,j,0}, D_{i,j,1}, \ldots, D_{i,j,n-1}) \in \mathbb{F}_2^n$, and let $D_{i,*,*}$ denote the $n \times n$ matrix whose j-th row is $D_{i,j,*}$, for $j \in [n]$.

5.7.2 Algorithms to recover the polynomials

We focus on homogeneous monomials of P degree $2 * d = 6$. Lower-degree monomials can be recovered in a similar way, as in [MDFK18]. The value of A can be deduced from the values of P, B, C using the equation $A = P - B \cdot C$. Thus, we focus on recovering B, C. More precisely, following the approach of [MDFK18], we recover $\mathsf{span}(B, C)$. For polynomials of degree $d = 2$, $\mathsf{span}(B, C)$ is computed from the knowledge of $\mathsf{span}(B_{i,*}, C_{i,*})$ for all $i \in [n]$. In our case, it is likewise possible to compute $\mathsf{span}(B, C)$ from the knowledge of $\mathsf{span}(B_{i,*,*}, C_{i,*,*})$.

We first need an algorithm that will decide if, given (i, j, k), $B_{i,j,k} = C_{i,j,k} = 0$. This can be done in instant time with a low probability of failure: we use Algorithm 4, then recursively use variants that take a smaller number of indices as inputs.

Algorithm 4 may output an incorrect answer, with (very) low probability. If this happens, the factorization algorithm will fail at a later stage. Since this happens with low probability, in the event of a failure, the algorithm can simply be run again. Using Algorithm 4, we can build an algorithm that takes as input $i \in [n]$ and returns (j, k, l)

Algorithm 4 ZeroOracle

Input: distinct $i, j, k, l, g \in [n]$
Output: True if $B_x = C_x = 0$ for all possible triples x in $\{i, j, k, l, g\}$.
for $h \in [n] \backslash \{i, j, k, l, g\}$ **do**
    if $P_{i,j,k,l,g,h} = 1$ **then**
        return False
return True

such that $(B_{j,k,l}, C_{j,k,l}) \neq (0, 0)$ but $B_x = C_x = 0$ for all other triples x in $\{i, j, k, l\}$. This is analogous to the algorithm "FindGood" in [MDFK18]:

Algorithm 5 FindGood

Input: $i \in [n]$
Output: $j, k, l \in [n]$ such that $(B_{j,k,l}, C_{j,k,l}) \neq (0, 0)$ and $B_x = C_x = 0$ for all other triples x in $\{i, j, k, l\}$.
while True **do**
    $j \xleftarrow{\$} [n] \backslash \{i\}$
    $k \xleftarrow{\$} [n] \backslash \{i, j\}$
    $l \xleftarrow{\$} [n] \backslash \{i, j, k\}$
    if not ZeroOracle(j, k, l) and ZeroOracle(x) for all other triples in $\{i, j, k, l\}$ **then**
        return (j, k, l)

With such a triple (j, k, l) for some $i \in [n]$, let $(\mu, \lambda) = (B_{j,k,l}, C_{j,k,l})$. Then, $\forall g, h$, we have $P_{i,j,k,l,g,h} = \lambda B_{i,g,h} + \mu C_{i,g,h}$. Without loss of generality, we can assume that $(\mu, \lambda) = (0, 1)$, since we are looking for $\mathsf{span}(B, C)$.

We gain knowledge of $B_{i,g,h}$ by evaluating $P_{i,j,k,l,g,h}$, thus for every $i \in [n]$, we can recover $\mathsf{span}(B_{i,*,*}, C_{i,*,*})$ by evaluating $P_{i,j,k,l,g,h}$ for all g, h. This step is analogous to the algorithm "GetSpace" in [MDFK18].

Algorithm 6 GetSpace

Input: $i \in [n]$
Output: $\mathsf{span}(B_{i,*,*}, C_{i,*,*})$
Let $v \in \mathbb{F}_2^{n^2}$
Let $E = \{0\} \subseteq \mathbb{F}_2^{n^2}$
while $dim(E) < 2$ **do**
    $(j, k, l) \leftarrow$ FindGood(i)
    for $g, h \in [n]$ **do**
        $v_{g,h} \leftarrow P_{i,j,k,l,g,h}$
    $E \leftarrow E + \mathsf{span}(v)$
return E

We can now build $\mathsf{span}(B, C)$ step by step: we first build $\mathsf{span}(B_{i,*,*}, C_{i,*,*})$, and we create a basis by selecting a random pair of vectors from this space. For $i > 1$, we check that the i-th vector space is consistent with the spaces from steps $i - 1$ and $i - 2$. If there is an inconsistency at some point, we backtrack and select a fresh random pair of vectors.

In the end, we obtain $\mathsf{span}(B, C)$. Let B', C' be two linearly independent vectors in

$\mathsf{span}(B, C)$, and let $A' = P - B'C'$. Then A', B', C' is a solution to the factorization problem.

5.8 Conclusion

Our results show that any function of the form "low-degree layer, S-box layer, affine layer" can be efficiently decomposed into its inner components (up to equivalence) using only *black-box* access to the function. Moreover, the same is true of functions of the form "low-degree layer, modular addition, affine layer". An immediate consequence is that the white-box encryption framework of [RVP22] is insecure, both for SPN and ARX ciphers. A broader consequence is that low-degree input encodings cannot hide a structured inner round function. If the round-encoding-based approach to white-box cryptography initiated by [CEJvO02a, CEJvO02b] is to ever succeed in producing *cryptographically* secure schemes (as opposed to schemes meant to heuristically slow down real-world attackers), it must use non-trivial non-linear encodings for both input and output encodings.

Unfortunately, we are not aware of a general framework to embed non-linear encodings in both the input and output of the round. Simple non-linear encodings have been used in the past on specific schemes, but were severely restricted (operating over a few bits only, in the manner of S-boxes) [CEJvO02a, CEJvO02b]. A more flexible way to embed non-linear encodings is a compelling goal for white-box cryptography. It might lead to cryptographically secure designs, which would be a major breakthrough. Indeed, having to structurally recover non-linear layers at both the input and output of a cipher relates to polynomial decomposition problems, which is NP-hard in the multivariate case [Dic93, FP09] (polynomial-time algorithms are known in the univariate case [Bla14]).

Publication

This work has been submitted for publication at the time of writing this thesis.

Chapter

6 Conclusion

In this thesis, we have studied different problems related to computing in association with untrusted agents. Cryptography is a great tool because it spares us the faith and time needed to build trust: we build protocols and prove that they are secure: this is enough to ensure that our information is protected. This is of course in theory: in practice the implementation of a cryptosystem is a potential liability, a small oversight can prove catastrophic. Nevertheless, even when dealing with theoretical designs of cryptosystems, security might not be proven to the highest degree of certainty, save for perfect secrecy, also called information theoretical security. Cryptanalysis can be devastating to a cipher with no solid security ground to stand on.

In Chapter 3, we present new constructions of PSM protocols, that achieve information theoretical security. In our model however, no party (except the referee) can become corrupted. We can use such protocols to achieve a variant where parties collude with the referee, called non-interactive MPC [BKN18]. By expanding the cube approach of [BKN18], we create a framework where we see functions and inputs as tensors. With it, we get a better view the information that the parties can securely send to the referee. Some of this information contains the output we want the referee to learn. When the right conditions are met, the garbling around this output can be cancelled, and the referee learns the cleartext value. This framework yields PSM protocols for all functions, however not for any number k of parties: only prime powers for now. An interesting question is: can we expand this construction to handle k as a composite of two primes, then eventually as any composite number (and thus all of $\mathbb{N}$)? Other techniques exist: in [SESN23], Shinagawa et al build PSM procotols based on *quadratic residues*. Not all functions fit this framework, but for k parties and any symmetric function $f : \{0,1\}^k \to \{0,1\}$, they achieve a communication cost of $O(k^2)$, improving on the $O(k^2 \cdot log(k))$ upper bound of [BIKK14].

In Chapter 4, we show how to build weighted-ORAM protocols: ORAM for objects of variable sizes. This is done by modifying existing ORAM protocols to make them handle weighted items. In our construction, no added cost in bandwidth is needed when done on a large subset of tree-ORAM protocols, including Path-ORAM which is used in practical applications [SvS+13a]. In other ORAMs however, a small blowup is needed. An interesting direction to explore would be to search for a natural way to expand the hierarchical method to handle weighted items. Then no blowup would be needed. We also present new single-keyword SSE protocols based on weighted tree-ORAMs. This method is more efficient than previous attempts at constructing SSE using ORAM. This work is part of a broader wish for secure encrypted databases, although improvements are being made, it remains an open problem to achieve functionalities on par with plaintext databases while retaining some notion of security.

In Chapter 5, we cryptanalyze white-box constructions based on implicit implementa-

tions from [RVP22]. We show that the attack on the ASASA structure from [MDFK18] still applies when the first affine layer is replaced by a low-degree nonlinear layer. We also attack the structure behind the white-box based on an ARX cipher. [BLU23] shows a decomposition-based key recovery attack on the same white-box, however without the structural aspect. Because of a flaw in the structure, we do not have suggestions to repair the system. The search for provably secure white-box cryptography continues. An open question is to analyze an ISI or an IMI construction: we prove that having a low-degree non-linear input layer is not enough, but while it is not clear how one could build a white-box where that layer is combined with a low-degree non-linear output layer, it is not clear either if one could find a (structural) attack. If we can keep building white-box schemes based on polynomial decomposition, we can hope to base them off of hard problems [Dic93, FP09].

Appendix

A Supplementary material on PSM

A.1 Proof of Equation (3.9) and (3.10)

Proof of Equation (3.9). By definition:

$$\sum\langle \mathbf{F}, \bar{\mathbf{X}}(P)\rangle = \sum_{(*)} \langle \mathbf{F}, \bar{\mathbf{X}}_{S_1} \otimes \ldots \otimes \bar{\mathbf{X}}_{S_t}\rangle$$

where $(*)$ denotes "for all unordered $E = \{S_1, \ldots, S_t\}$ being a partition of $[2k]$ such that $\{|S_1|, \ldots, |S_t|\} = P$". Thus,

$$\begin{aligned}
\sum\langle \mathbf{F}, \bar{\mathbf{X}}(P)\rangle &= \sum_{(*)} \left\langle \mathbf{F}, \bigotimes_{i\in[t]} (\mathbf{R}_{S_i} + \bigotimes_{j\in S_i} \mathbf{x}_j) \right\rangle \\
&= \sum_{(*)} \sum_{G\subseteq E} \left\langle \mathbf{F}, \bigotimes_{S\in G} \mathbf{R}_S \otimes \bigotimes_{j\notin \bigcup_{S\in G} S} \mathbf{x}_j \right\rangle \\
&= \sum_{Q\subseteq P} \sum_{\substack{G=\{S_1,\ldots,S_t\} \text{ s.t.} \\ \{|S_1|,\ldots,|S_t|\}=P\setminus Q}} \beta(P,G) \cdot \left\langle \mathbf{F}, \bigotimes_{i\in[t]} R_{S_i} \otimes \bigotimes_{j\notin \bigcup_{i\in[t]} S_i} \mathbf{x}_j \right\rangle,
\end{aligned}$$

where $\beta(P, G)$ accounts for the redundancy: define $\beta(P, G)$ as the number of unordered partitions E of $[2k]$ such that $G \subseteq E$ and P is the shape of E. It is equivalent to count the number of $F := E \setminus G$. That is, $\beta(P, G)$ also equals the number of unordered partitions F of $[2k] \setminus \bigcup_{S\in G} S$ such that Q is the shape of F. Thus by definition, $\beta(P, G) = \alpha(Q)$. The proof is concluded by

$$\begin{aligned}
\sum\langle \mathbf{F}, \bar{\mathbf{X}}(P)\rangle &= \sum_{Q\subseteq P} \sum_{\substack{G=\{S_1,\ldots,S_t\} \text{ s.t.} \\ \{|S_1|,\ldots,|S_t|\}=P\setminus Q}} \alpha(Q) \cdot \left\langle \mathbf{F}, \bigotimes_{i\in[t]} R_{S_i} \otimes \bigotimes_{j\notin \bigcup_{i\in[t]} S_i} \mathbf{x}_j \right\rangle \\
&= \sum_{Q\subseteq P} \alpha(Q) \cdot \sum\langle \mathbf{F}, \mathbf{R}(P \setminus Q)\rangle. \qquad \square
\end{aligned}$$

Proof of Equation (3.10). Let $n = \mathrm{sum}(Q)$. By definition, $\alpha(Q)$ is the number of unsorted partitions $E = \{S_1, \ldots, S_t\}$ of $[n]$ such that the multiset $\{|S_1|, \ldots, |S_t|\}$ (i.e. the shape of E) equals Q.

To compute $\alpha(Q)$, we count the number of ways to arranging $1, \ldots, n$ into a sequence.

- First, pick an unsorted partitions E of $[n]$ s.t. the shape of E equals Q. The number of choices is $\alpha(Q)$.

- Then, sort the sets in the partion $E = \{S_1, \ldots, S_t\}$. Sort them by their sizes, i.e. $|S_1| \leq |S_2| \leq \cdots \leq |S_t|$. For any m, if several sets are of the size m, their order has to be specified, the number of such choices is (number of m's in Q)!.

- Finally, arrange the elements in each S_i into a sub-sequence, the number of possible sequences is $|S_i|!$. Concatenate these sub-sequences in order.

$$\alpha(Q) \cdot \prod_{m \in \mathbb{Z}^+} (\text{number of } m\text{'s in } Q)! \cdot \prod_{i \in Q} i! = n! \qquad \square$$

A.2 Auxiliary PSM Protocols for $\langle \mathbf{x}_1 \otimes \ldots \otimes \mathbf{x}_k, \mathbf{Y} \rangle + s$

A.2.1 The Multi-party Variant

In this section, we present an auxiliary PSM protocol that is used as a subroutine by our multi-party PSM in Section 3.3.

The functionality is $\langle \mathbf{x}_1 \otimes \ldots \otimes \mathbf{x}_k, \mathbf{Y} \rangle + s$. It is a $(k+1)$-party functionality where the i-th party has as input $\mathbf{x}_i \in \mathbb{F}^N$ for $i \in [k]$, and the $(k+1)$-th party has as inputs $\mathbf{Y} \in \mathbb{F}^{N \times \cdots \times N}_{k \text{ times}}$ and $s \in \mathbb{F}$. We will present a PSM protocol for this functionality with a communication complexity of $O(\text{poly}(k) \cdot N^k)$ field elements. This protocol is implicitly used in [BKN18].

First, we consider the special case when $k = 1$. That is, there are only two parties. Say we call them Alice and Bob. Alice has $\mathbf{x} \in \mathbb{F}^N$, Bob has $\mathbf{y} \in \mathbb{F}^N, s \in \mathbb{F}$. The functionality output is $\langle \mathbf{x}, \mathbf{y} \rangle + s$. The PSM protocol works as follows:

- Random $\mathbf{a}, \mathbf{b} \in \mathbb{F}^N, c \in \mathbb{F}$ are sampled from the common random string, which is known by both Alice and Bob.

- Alice sends $\bar{\mathbf{x}} := \mathbf{x} + \mathbf{a}, z := c - \langle \mathbf{b}, \mathbf{x} \rangle$ to the referee.

- Bob sends $\bar{\mathbf{y}} := \mathbf{y} + \mathbf{b}, w := s - c - \langle \mathbf{a}, \mathbf{y} \rangle - \langle \mathbf{a}, \mathbf{b} \rangle$ to the referee.

- The referee outputs $\langle \bar{\mathbf{x}}, \bar{\mathbf{y}} \rangle + z + w$.

For the case $k \geq 2$, the first k parties need to jointly emulate Alice. The protocol works as follows:

- Random $\mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbb{F}^{N \times \cdots \times N}$ are sampled from the common random string. Define $c \in \mathbb{F}$ as the sum of entries in $\mathbf{C}$.

- The $(k+1)$-th party sends $\bar{\mathbf{Y}} := \mathbf{Y} + \mathbf{B}, z := s - c - \langle \mathbf{A}, \mathbf{Y} \rangle - \langle \mathbf{A}, \mathbf{B} \rangle$ to the referee.

- The first k parties jointly reveal $\bar{\mathbf{X}} := \mathbf{x}_1 \otimes \ldots \otimes \mathbf{x}_k + \mathbf{A}$, $w := c - \langle \mathbf{B}, \mathbf{x}_1 \otimes \ldots \otimes \mathbf{x}_k \rangle$ to the referee.

  Since every coordinate of $\bar{\mathbf{X}}$ can be computed by an arithmetic formula of size $O(k)$, each of these coordinates can be computed by the referee by using a PSM protocol with communication complexity of $O(\text{poly}(k))$ field elements [IK00]. The referee learns $\bar{\mathbf{X}}$ after receiving $O(\text{poly}(k) \cdot N^k)$ field elements.

The term $w := c - \langle \mathbf{B}, \mathbf{x}_1 \otimes \ldots \otimes \mathbf{x}_k \rangle$ equals the sum of all entries in $\mathbf{W} := \mathbf{C} - \mathbf{B} \circ_{\text{p.w.}} (\mathbf{x}_1 \otimes \ldots \otimes \mathbf{x}_k)$, where $\circ_{\text{p.w.}}$ denotes the point-wise product. In other words, we defines $\mathbf{W} \in \mathbb{F}^{N \times \cdots \times N}$ as

$$\mathbf{W}[i_1, \ldots, i_k] = \mathbf{C}[i_1, \ldots, i_k] - \mathbf{B}[i_1, \ldots, i_k]\mathbf{x}_1[i_1] \ldots \mathbf{x}_k[i_k].$$

Due to the randomness of $\mathbf{C}$, we know $\mathbf{W}$ is a randomized encoding of w. Thus, it is equivalent for the first k parties to jointly reveal $\mathbf{W}$ to the referee. Since every coordinate of $\mathbf{W}$ can be computed by an arithmetic formula of size $O(k)$, each of them can be revealed by using the Ishai-Kushilevitz PSM protocol [IK00], which has a communication complexity of $O(\text{poly}(k))$ field elements. The referee learns w after receiving $O(\text{poly}(k) \cdot N^k)$ field elements.

- The referee outputs $\langle \bar{\mathbf{X}}, \bar{\mathbf{Y}} \rangle + z + w$.

The correctness of the protocol can be verified in the following equation:

$$\begin{aligned} & \langle \bar{\mathbf{X}}, \bar{\mathbf{Y}} \rangle + z + w \\ = & \langle \mathbf{x}_1 \otimes \ldots \otimes \mathbf{x}_k + \mathbf{A}, \mathbf{Y} + \mathbf{B} \rangle + s - c - \langle \mathbf{A}, \mathbf{Y} \rangle - \langle \mathbf{A}, \mathbf{B} \rangle + \\ & c - \langle \mathbf{B}, \mathbf{x}_1 \otimes \ldots \otimes \mathbf{x}_k \rangle \\ = & \langle \mathbf{x}_1 \otimes \ldots \otimes \mathbf{x}_k, \mathbf{Y} \rangle + s. \end{aligned}$$

The privacy is guaranteed by the following simulator:

- Simulate $\bar{\mathbf{X}}, \bar{\mathbf{Y}}, \mathbf{W}$ as uniform random, since they are one-time-padded by $\mathbf{A}, \mathbf{B}, \mathbf{C}$.
- Given $\bar{\mathbf{X}}, \bar{\mathbf{Y}}, \mathbf{W}$ and the function output, w, z are uniquely determined since $w = \sum(\mathbf{W})$ and $\langle \bar{\mathbf{X}}, \bar{\mathbf{Y}} \rangle + z + w = \text{output}$.
- Simulate the transcripts of the inner Ishai-Kushilevitz PSM protocols using its own simulator, which takes $\bar{\mathbf{X}}, \mathbf{W}$ as input.

A.2.2 The 2-party Variant

In this section, we present an auxiliary PSM protocol that is used as a subroutine by our unbalanced 2-party PSM in Section 3.4.

The functionality is $\langle \mathbf{x}_1 \otimes \ldots \otimes \mathbf{x}_k, \mathbf{Y} \rangle + s$. It is a 2-party functionality where the first party, namely Alice, has as inputs $\mathbf{x}_1, \ldots, \mathbf{x}_k \in \mathbb{F}^N$ and the second party, namely Bob, has as inputs $\mathbf{Y} \in \mathbb{F}^{\underbrace{N \times \cdots \times N}_{k \text{ times}}}$ and $s \in \mathbb{F}$. We will present a PSM protocol for this functionality with unbalanced communication complexity, where Alice sends $O(kN)$ field elements and Bob sends $(N+1)^k$ field elements.

As the first step, we consider a harder problem instead. Bob's input is replaced by a multi-affine function $f : \mathbb{F}^N \times \cdots \times \mathbb{F}^N \to \mathbb{F}$. Corresponding, the functionality is replaced by $f(\mathbf{x}_1, \ldots, \mathbf{x}_k)$. Every multi-affine function f can be uniquely represented by its coefficient tensor $\mathbf{F} \in \mathbb{F}^{(N+1) \times \cdots \times (N+1)}$ such that for any $\mathbf{z}_1, \ldots, \mathbf{z}_k \in \mathbb{F}^N$,

$$f(\mathbf{z}_1, \ldots, \mathbf{z}_k) = \langle \mathbf{z}_1 \| 1 \otimes \cdots \otimes \mathbf{z}_k \| 1, \mathbf{F} \rangle.$$

Here $\mathbf{z}_i \| 1$ denotes the concatenation of $\mathbf{z}_i$ and 1, which is a dimension-$(N+1)$ vector. Notice that, if we let the "first" $N \times \cdots \times N$ subtensor of $\mathbf{F}$ equal $\mathbf{Y}$, let its "last" entry $\mathbf{F}[N+1, \ldots, N+1] = s$, and let all other entries in $\mathbf{F}$ be 0, we have

$$f(\mathbf{x}_1, \ldots, \mathbf{x}_k) = \langle \mathbf{x}_1 \| 1 \otimes \cdots \otimes \mathbf{x}_k \| 1, \mathbf{F} \rangle = \langle \mathbf{x}_1 \otimes \ldots \otimes \mathbf{x}_k, \mathbf{Y} \rangle + s.$$

The protocol works as follows:

- Random $\mathbf{r}_1, \ldots, \mathbf{r}_k \in \mathbb{F}^N$ and a random multi-affine function g are sampled from the common random string.
- Alice sends $\bar{\mathbf{x}}_i = \mathbf{x}_i + \mathbf{r}_i$ to the referee, for all $i \in [k]$.
- Bob computes the multi-affine function g, such that

$$g(\mathbf{z}_1, \ldots, \mathbf{z}_k) := f(\mathbf{z}_1 - \mathbf{r}_1, \ldots, \mathbf{z}_k - \mathbf{r}_k).$$

  Bob sends $\bar{g} = g + h$ to the referee.
- Alice additionally sends $s = h(\bar{\mathbf{x}}_1, \ldots, \bar{\mathbf{x}}_k)$ to the referee.
- The referee outputs $\bar{g}(\bar{\mathbf{x}}_1, \ldots, \bar{\mathbf{x}}_k) - s$.

The correctness follows directly from the following equation:

$$\begin{aligned}
\bar{g}(\bar{\mathbf{x}}_1, \ldots, \bar{\mathbf{x}}_k) - s &= g(\bar{\mathbf{x}}_1, \ldots, \bar{\mathbf{x}}_k) + h(\bar{\mathbf{x}}_1, \ldots, \bar{\mathbf{x}}_k) - h(\bar{\mathbf{x}}_1, \ldots, \bar{\mathbf{x}}_k) \\
&= g(\bar{\mathbf{x}}_1, \ldots, \bar{\mathbf{x}}_k) \\
&= f(\mathbf{x}_1 - \mathbf{r}_1 + \mathbf{r}_1, \ldots, \mathbf{x}_k - \mathbf{r}_k + \mathbf{r}_k) \\
&= f(\mathbf{x}_1, \ldots, \mathbf{x}_k).
\end{aligned}$$

The privacy is guaranteed by the following simulator:

- Simulate $\bar{\mathbf{x}}_1, \ldots, \bar{\mathbf{x}}_k, \bar{g}$ as uniform random, since they are one-time padded by $\mathbf{r}_1, \ldots, \mathbf{r}_k, h$.
- Given $\bar{\mathbf{x}}_1, \ldots, \bar{\mathbf{x}}_k, \bar{g}$ and the function output, simulate s by computing s from the equation $\bar{g}(\bar{\mathbf{x}}_1, \ldots, \bar{\mathbf{x}}_k) - s = \text{output}$.

Appendix

B Additional proofs for weighted ORAM

B.1 SSE Proofs

B.1.1 Proof of ZeroSSE

The simulator $\mathcal{S}$ for ZeroSSE behaves as follows.

- `Setup`: on input $\mathcal{L}_{\texttt{Setup}} = (N, U)$, $\mathcal{S}$ creates a wORAM instance `Weighted(PathOram)` instance with block size U and number of blocks $\lceil N/U \rceil$, initialized with $\lceil N/U \rceil$ items containing dummy data of size B. Let a denote the address of one of these items.

- `Search` and `Update`: in both cases, $\mathcal{S}$ queries the wORAM instance on item a with $\mathsf{op} = \text{read}$.

We claim that the resulting transcript is indistinguishable from the real transcript. In fact, this follows immediately from the security of `Weighted(PathOram)`, since the guarantees of a wORAM scheme is that any two sequence of accesses of the same length are indistinguishable, and the simulator makes the same number of accesses as the client in the real game. Moreover, Theorem 4.4.1 implies that `Weighted(PathOram)` is a correct wORAM scheme.

Note that the analysis relies on the fact that Path ORAM is correct for the given parameters, which in turn implies that we require $N/U = \Omega(\lambda)$. If $N/U < \lambda$, it may not be worth using an elaborate ORAM scheme, since the overhead of trivial ORAM is less than λ.

B.1.2 Proof of BlockSSE

The simulator $\mathcal{S}$ for BlockSSE behaves as follows.

- `Setup`: on input $\mathcal{L}_{\texttt{Setup}} = (N, B)$, $\mathcal{S}$ creates a wORAM instance `Weighted(PathOram)` instance with block size B and number of blocks $\lceil N/B \rceil$, initialized with $\lceil N/B \rceil$ items containing dummy data of size B. Let a denote the address of one of these items.

- `Update`: $\mathcal{S}$ queries the wORAM instance on item a with $\mathsf{op} = \text{read}$.

- `Search`: on input $\mathcal{L}_{\texttt{Search}} = x$, $\mathcal{S}$ queries the wORAM instance x times on item a with $\mathsf{op} = \text{read}$.

We claim that the resulting transcript is indistinguishable from the real transcript. Again, this follows immediately from the security of `Weighted(PathOram)`, since the guarantees of a wORAM scheme is that any two sequence of accesses of the same length are indistinguishable, and the simulator makes the same number of accesses as the client in the real game. Moreover, Theorem 4.4.1 implies that `Weighted(PathOram)` is a correct wORAM scheme. As with ZeroSSE, that last point assumes that $N/B = \Omega(\lambda)$.

B.2 A General Transformation

In this section we present a way to transform any standard ORAM into a variable ORAM, at the cost of some blowup in the overhead. We have m objects, of individual size $w_i \in [0, B]$ for $i \in [k]$ such that the sum of the sizes is N. The idea is:

- We create a standard ORAM structure for N objects, each of the same size higher than B (we discuss the size we need later). Let us call these N objects in the standard ORAM *super objects*.
- We want to store our m objects of varying sizes *inside* of the super objects. To do so, we map every object $i \in [m]$ to a super object $H(i) \in [N]$.
- When we wish to access the object i, we make a standard ORAM access to the super object $H(i)$, and read the object from the super object.

This technique yields the same computational, storage and bandwidth metrics as the ORAM we are modifying, with the caveat that the object size must be bigger.

The crucial part here is the function H. We must choose a way of hashing the m objects into the N super objects while having the following properties:

- A super object cannot overflow, except with negligible probability.
- The hash table should be modified as little as possible, so as to save bandwidth.
- The super objects should be as small as possible.

We present two hash algorithms based on the balls-in-bins problem that can be used to achieve this type of weighted ORAM.

B.2.1 One-Choice Process: $\mathcal{O}(\log N)$ blowup

We let each object be associated with a super object uniformly at random. In other words, $H : [m] \to [N]$ is a random function. At each access, to get object i we call the access algorithm of the ORAM on super object $H(i)$, modify object i in it, re-encrypt the super-object with fresh randomness, and write back the super object. It suffices to have each super object be of size $B \cdot \log N$.

Proof. Let m balls, each of weight $w_i \in [0, 1]$ such that $\sum_{i=1}^{m} w_i = N$, be stored in N bins, which we associate uniformly at random. Let us call S the bin with the highest load. Let us call $\mathbf{w}$ the current weight distribution of the balls, and $\mathbf{u}$ be the weight distribution when we have only N balls of size 1, that is:

$$\mathbf{u} = (\underbrace{1, \ldots, 1}_{N}, \underbrace{0, \ldots, 0}_{m-N}).$$

From the majorization argument in [BFHM05], we know that for any $\mathbf{w}$, $\mathbb{E}(\sum_{i\in S} w_i) \leq \mathbb{E}(\sum_{i\in S} u_i)$. Since the max function is convex and the expectancy is linear, we have $\mathbb{E}(\max(0, (\sum_{i\in S} w_i) - \log N)) \leq \mathbb{E}(\max(0, (\sum_{i\in S} u_i) - \log N))$. From now on, we will let the random variable $X(\mathbf{v})$ denote $\max(0, (\sum_{i\in S} u_i) - \log(N))$.

We have $\mathbb{P}(X(\mathbf{w}) > 0) \leq \mathbb{E}(X(\mathbf{w}))$. The event $X(\mathbf{w}) > 0$ is exactly the event of having at least one overflow in a given bin, so to prove that it happens with negligeable probability, it suffices to prove that $\mathbb{E}(X(\mathbf{u}))$ is negligeable. We have:

$$\begin{aligned}
\mathbb{E}(X(\mathbf{u})) &= \sum_{k=0}^{\infty} \mathbb{P}(X(\mathbf{u}) > k) \\
&\leq N \cdot \mathbb{P}(X(\mathbf{u}) > 0) \\
\mathbb{P}(X(\mathbf{u}) > 0) &= \mathbb{P}[\text{a given bin has more than } k \text{ balls}] \text{ with } k = \log N \\
&\leq \binom{N}{k} \cdot \frac{1}{N^k} \\
&\leq \frac{N^k}{k!} \cdot \frac{1}{N^k} \\
&= \frac{1}{k!} \\
&= \text{negl}(N) \text{ via Stirling's formula.}
\end{aligned}$$

□

B.2.2 Layered Two-Choice Process: $\mathcal{O}(\log \log N)$ blowup

The Layered 2-choice method of hashing m balls into N bins was introduced in [MR22, Section 4]. The idea is to split the m objects into "layers" according to their sizes. Then we can store them in the super objects by running standard (unweighted) 2-choice balls-in-bins allocation separately for each layer. Such an allocation is achievable with overhead $\mathcal{O}(\log \log N)$.

However this algorithm does not support offline modification of the bins. This means that after an access for object i, we must update the ORAM by selecting 2 new super objects uniformly at random and write i in one of those super objects. The first consequence is that we double the bandwidth. Another problem is that the hash table has to be modified. To prevent client side spatial blowup, we must store the hash table (effectively a position map) on the server. This may be realized using a second ORAM (which need not use the same ORAM protocol).

Overall, while this approach is generic, it is significantly less efficient in practice than our main construction, which incurs almost no overhead relative to standard Tree ORAM.

B.3 A Flaw in DivORAM

DivORAM was introduced in [LHL+18]. The DivORAM construction is (essentially) a weighted ORAM following a Tree ORAM structure: blocks are stored in buckets, and each bucket is located at the node of a binary tree. The blocks are decomposed into so-called splinters, which are stored in the buckets. A bucket at level $i \in [\log N]$ can store up to 2^i splinters.

Stash analysis is given in [LHL+18, Section 5.3]. Whenever a splinter cannot be stored in a tree bucket, it ends up in the stash. The worst case is when an entire block, composed

of $\log N$ splinters, cannot be stored in the tree: all splinters are then moved to the stash. The stash capacity is $2\log(N) - 1$. To avoid overflows, the content of the stash is written back into the tree whenever its size exceeds $\log N$.

Such a behavior reveals to the server when the stash size passes the $\log N$ threshold. Unfortunately, leaking when the stash size passes a threshold breaks Path ORAM security, because certain access patterns fill the stash faster than others. For more information, we refer the reader to [RYF+13, Section 3.1.1], where this exact point is discussed. Because of this leakage, the DivORAM construction of [LHL+18] is insecure.

B.4 Weighted Hierarchical ORAM

Our work focuses on Tree ORAMs, mainly because of their higher practical efficiency. It is natural to ask whether similar results can be obtained for hierarchical ORAM. We briefly discuss this question here.

As shown in chapter 4, there is a natural way to adapt Tree ORAM algorithms to handle weighted items. The problem of building weighted Tree ORAM is not so much an algorithmic problem as a combinatorial one: we need to prove that the "natural" weighted variant of Tree ORAM creates no overflow.

With hierarchical ORAMs, this first step is already a challenge. There is no obvious natural algorithmic modification of hierarchical ORAMs that handles weighted items. Indeed, we would first need weighted oblivious hash tables. That is, we need oblivious hash tables that can handle items of variable size. We also need those tables to support oblivious sorting, or at least oblivious compaction, in a way that is compatible with variable-size items. Building such a data structure looks quite challenging, and we do not know of a natural candidate.

In more details, we know of two constructions that come close to that goal in the literature, but neither quite reaches it. The first is two-choice hashing for weighted items [TW07, MR22]. Two-choice hashing is easy to realize obliviously. The problem is that we would have to pay (at least) a $\log\log$ overhead, and efficient oblivious compaction seems difficult. An alternative approach is to use the "Data-Independent Packing" primitive from [BBF+21]. But the only known instantiation requires computing a maximum flow algorithm. It is not clear how to compute that algorithm in quasilinear time, let alone in an oblivious manner.

Bibliography

[AA18] Benny Applebaum and Barak Arkis. On the power of amortization in secret sharing: d-uniform secret sharing and CDS with constant information rate. In Amos Beimel and Stefan Dziembowski, editors, *Theory of Cryptography - 16th International Conference, TCC 2018, Panaji, India, November 11-14, 2018, Proceedings, Part I*, volume 11239 of *Lecture Notes in Computer Science*, pages 317–344. Springer, 2018. 26

[AARV17] Benny Applebaum, Barak Arkis, Pavel Raykov, and Prashant Nalini Vasudevan. Conditional disclosure of secrets: Amplification, closure, amortization, lower-bounds, and separations. *Electronic Colloquium on Computational Complexity (ECCC)*, 24:38, 2017. 26

[ABF+19] Benny Applebaum, Amos Beimel, Oriol Farràs, Oded Nir, and Naty Peter. Secret-sharing schemes for general and uniform access structures. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part III*, volume 11478 of *Lecture Notes in Computer Science*, pages 441–471. Springer, 2019. 26

[ABMT18] Estuardo Alpirez Bock, Chris Brzuska, Wil Michiels, and Alexander Treff. On the ineffectiveness of internal encodings - revisiting the DCA attack on white-box cryptography. In Bart Preneel and Frederik Vercauteren, editors, *ACNS 18*, volume 10892 of *LNCS*, pages 103–120. Springer, Heidelberg, July 2018. 10, 19

[AG11] Sanjeev Arora and Rong Ge. New algorithms for learning in presence of errors. In Luca Aceto, Monika Henzinger, and Jirí Sgall, editors, *Automata, Languages and Programming - 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part I*, volume 6755 of *Lecture Notes in Computer Science*, pages 403–415. Springer, 2011. 76, 82

[AHMS20] Benny Applebaum, Thomas Holenstein, Manoj Mishra, and Ofer Shayevitz. The communication complexity of private simultaneous messages, revisited. *J. Cryptol.*, 33(3):917–953, 2020. 6, 16, 21

[AKL+18] Gilad Asharov, Ilan Komargodski, Wei-Kai Lin, Kartik Nayak, Enoch Peserico, and Elaine Shi. OptORAMa: Optimal oblivious RAM. Cryptology ePrint Archive, Report 2018/892, 2018. https://eprint.iacr.org/2018/892. 8, 18

[AKL+20] Gilad Asharov, Ilan Komargodski, Wei-Kai Lin, Kartik Nayak, Enoch Peserico, and Elaine Shi. OptORAMa: Optimal oblivious RAM. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 403–432. Springer, Heidelberg, May 2020. 49

[AL21] Leonard Assouline and Tianren Liu. Multi-party psm, revisited: Improved communication and unbalanced communication. In Kobbi Nissim and Brent Waters, editors, *Theory of Cryptography*, pages 194–223, Cham, 2021. Springer International Publishing. 6, 16, 43

[AM23] Léonard Assouline and Brice Minaud. Weighted oblivious ram, with applications to searchable symmetric encryption. In *Advances in Cryptology – EUROCRYPT 2023: 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part I*, pages 426–455, Berlin, Heidelberg, 2023. Springer-Verlag. 9, 18, 71

[AMR19] Alessandro Amadori, Wil Michiels, and Peter Roelse. A DFA attack on white-box implementations of AES with external encodings. In Kenneth G. Paterson and Douglas Stebila, editors, *SAC 2019*, volume 11959 of *LNCS*, pages 591–617. Springer, Heidelberg, August 2019. 10, 19

[ANSS16] Gilad Asharov, Moni Naor, Gil Segev, and Ido Shahaf. Searchable symmetric encryption: optimal locality in linear space via two-dimensional balanced allocations. In Daniel Wichs and Yishay Mansour, editors, *48th ACM STOC*, pages 1101–1114. ACM Press, June 2016. 46, 48

[BBD+22] Guillaume Barbu, Ward Beullens, Emmanuelle Dottax, Christophe Giraud, Agathe Houzelot, Chaoyun Li, Mohammad Mahzoun, Adrián Ranea, and Jianrui Xie. ECDSA white-box implementations: Attacks and designs from WhibOx 2021 contest. *Cryptology ePrint Archive*, 2022. 74

[BBF+21] Angèle Bossuat, Raphael Bost, Pierre-Alain Fouque, Brice Minaud, and Michael Reichle. SSE and SSD: Page-efficient searchable symmetric encryption. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part III*, volume 12827 of *LNCS*, pages 157–184, Virtual Event, August 2021. Springer, Heidelberg. 46, 47, 48, 69, 106

[BBK14] Alex Biryukov, Charles Bouillaguet, and Dmitry Khovratovich. Cryptographic schemes based on the ASASA structure: Black-box, white-box, and public-key (extended abstract). In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 63–84. Springer, 2014. 75, 76, 77, 80

[BCBP03] Alex Biryukov, Christophe De Cannière, An Braeken, and Bart Preneel. A toolbox for cryptanalysis: Linear and affine equivalence algorithms. In Eli Biham, editor, *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques,*

Warsaw, Poland, May 4-8, 2003, Proceedings, volume 2656 of *Lecture Notes in Computer Science*, pages 33–50. Springer, 2003. 86

[BCP16] Elette Boyle, Kai-Min Chung, and Rafael Pass. Oblivious parallel RAM and applications. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part II*, volume 9563 of *LNCS*, pages 175–204. Springer, Heidelberg, January 2016. viii, 65

[BDG+22] Sven Bauer, Hermann Drexler, Maximilian Gebhardt, Dominik Klein, Friederike Laus, and Johannes Mittmann. Attacks against white-box ECDSA and discussion of countermeasures-a report on the WhibOx contest 2021. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022. 74

[BF19] Raphael Bost and Pierre-Alain Fouque. Security-efficiency tradeoffs in searchable encryption. *PoPETs*, 2019(4):132–151, October 2019. 68

[BFHM05] Petra Berenbrink, Tom Friedetzky, Zengjian Hu, and Russell Martin. On weighted balls-into-bins games. In Volker Diekert and Bruno Durand, editors, *STACS 2005*, pages 231–243, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. 105

[BFHM08] Petra Berenbrink, Tom Friedetzky, Zengjian Hu, and Russell Martin. On weighted balls-into-bins games. *Theoretical Computer Science*, 409(3):511–520, 2008. 48

[BGE04] Olivier Billet, Henri Gilbert, and Charaf Ech-Chatbi. Cryptanalysis of a white box AES implementation. In Helena Handschuh and M. Anwar Hasan, editors, *Selected Areas in Cryptography, 11th International Workshop, SAC 2004, Waterloo, Canada, August 9-10, 2004, Revised Selected Papers*, volume 3357 of *Lecture Notes in Computer Science*, pages 227–240. Springer, 2004. 74, 75

[BGI+01] Boaz Barak, Oded Goldreich, Rusell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *Advances in Cryptology — CRYPTO 2001*, pages 1–18, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. 75

[Bha14] Arnab Bhattacharyya. Polynomial decompositions in polynomial time. In *European Symposium on Algorithms*, pages 125–136. Springer, 2014. 89

[BHI+20] Marshall Ball, Justin Holmgren, Yuval Ishai, Tianren Liu, and Tal Malkin. On the complexity of decomposable randomized encodings, or: How friendly can a garbling-friendly PRF be? In Thomas Vidick, editor, *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, volume 151 of *LIPIcs*, pages 86:1–86:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. 21

[BHMT16] Joppe W Bos, Charles Hubain, Wil Michiels, and Philippe Teuwen. Differential computation analysis: Hiding your white-box designs is not enough. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 215–236. Springer, 2016. 75

[BIK17] Amos Beimel, Yuval Ishai, and Eyal Kushilevitz. Ad hoc PSM protocols: Secure computation without coordination. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part III*, volume 10212 of *Lecture Notes in Computer Science*, pages 580–608, 2017. 26

[BIKK14] Amos Beimel, Yuval Ishai, Ranjit Kumaresan, and Eyal Kushilevitz. On the cryptographic complexity of the worst functions. In *TCC*, pages 317–342, 2014. 6, 7, 16, 22, 23, 26, 33, 97

[BKN18] Amos Beimel, Eyal Kushilevitz, and Pnina Nissim. The complexity of multiparty PSM protocols and related models. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 287–318. Springer, 2018. 6, 16, 22, 23, 26, 33, 42, 97, 100

[Bla14] Raoul Blankertz. A polynomial time algorithm for computing all minimal decompositions of a polynomial. *ACM Communications in Computer Algebra*, 48(1/2):13–23, 2014. 95

[BLU23] Alex Biryukov, Baptiste Lambin, and Aleksei Udovenko. Cryptanalysis of arx-based white-box implementations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(3):97–135, Jun. 2023. 75, 98

[Bos16] Raphael Bost. $\Sigma o \phi o \varsigma$: Forward secure searchable encryption. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1143–1154. ACM Press, October 2016. 69

[BR22] Marshall Ball and Tim Randolph. A note on the complexity of private simultaneous messages with many parties. In Dana Dachman-Soled, editor, *3rd Conference on Information-Theoretic Cryptography (ITC 2022)*, volume 230 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:12, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. 6, 16, 21

[BRVW19] Andrey Bogdanov, Matthieu Rivain, Philip S. Vejre, and Junwei Wang. Higher-order DCA against standard side-channel countermeasures. In Ilia Polian and Marc Stöttinger, editors, *Constructive Side-Channel Analysis and Secure Design - 10th International Workshop, COSADE 2019, Darmstadt, Germany, April 3-5, 2019, Proceedings*, volume 11421 of *Lecture Notes in Computer Science*, pages 118–141. Springer, 2019. 75

[BS01] Alex Biryukov and Adi Shamir. Structural cryptanalysis of SASAS. In Birgit Pfitzmann, editor, *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic*

Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding, volume 2045 of *Lecture Notes in Computer Science*, pages 394–405. Springer, 2001. 74, 76, 77, 80, 84

[BSS+13] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK families of lightweight block ciphers. Cryptology ePrint Archive, Report 2013/404, 2013. https://eprint.iacr.org/2013/404. 10, 19

[CCS17] T.-H. Hubert Chan, Kai-Min Chung, and Elaine Shi. On the depth of oblivious parallel RAM. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 567–597. Springer, Heidelberg, December 2017. 60

[CEJv03] Stanley Chow, Philip A. Eisen, Harold Johnson, and Paul C. van Oorschot. White-box cryptography and an AES implementation. In Kaisa Nyberg and Howard M. Heys, editors, *SAC 2002*, volume 2595 of *LNCS*, pages 250–270. Springer, Heidelberg, August 2003. 9, 19

[CEJvO02a] Stanley Chow, Philip A. Eisen, Harold Johnson, and Paul C. van Oorschot. White-box cryptography and an AES implementation. In Kaisa Nyberg and Howard M. Heys, editors, *Selected Areas in Cryptography, 9th Annual International Workshop, SAC 2002, St. John's, Newfoundland, Canada, August 15-16, 2002. Revised Papers*, volume 2595 of *Lecture Notes in Computer Science*, pages 250–270. Springer, 2002. 73, 75, 95

[CEJvO02b] Stanley Chow, Philip A. Eisen, Harold Johnson, and Paul C. van Oorschot. A white-box DES implementation for DRM applications. In Joan Feigenbaum, editor, *Security and Privacy in Digital Rights Management, ACM CCS-9 Workshop, DRM 2002, Washington, DC, USA, November 18, 2002, Revised Papers*, volume 2696 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2002. 73, 75, 95

[CFV19] Federico Cioschi, Nicolò Fornari, and Andrea Visconti. *White-Box Cryptography: A Time-Security Trade-Off for the SPNbox Family*, pages 153–166. 01 2019. 5, 15

[CGO21] Michele Ciampi, Vipul Goyal, and Rafail Ostrovsky. Threshold garbled circuits and ad hoc secure computation. Cryptology ePrint Archive, Report 2021/308, 2021. https://eprint.iacr.org/2021/308. 26

[CGPR15] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-abuse attacks against searchable encryption. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 668–679. ACM Press, October 2015. 46, 47, 66

[Con22] Graeme Connell. Technology deep dive: Building a faster oram layer for enclaves. https://signal.org/blog/building-faster-oram/, 2022. 5, 14

[CP13] Kai-Min Chung and Rafael Pass. A simple ORAM. Cryptology ePrint Archive, Report 2013/243, 2013. https://eprint.iacr.org/2013/243. viii, 8, 18, 48, 60, 62

[DCRS12] Kevin P. Dyer, Scott E. Coull, Thomas Ristenpart, and Thomas Shrimpton. Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail. In *2012 IEEE Symposium on Security and Privacy*, pages 332–346. IEEE Computer Society Press, May 2012. 47

[DFKYZD99] Ye Ding-Feng, Lam Kwok-Yan, and Dai Zong-Duo. Cryptanalysis of "2R" schemes. In *Annual International Cryptology Conference*, pages 315–325. Springer, 1999. 74

[DFLM18] Patrick Derbez, Pierre-Alain Fouque, Baptiste Lambin, and Brice Minaud. On recovering affine encodings in white-box implementations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):121–149, 2018. 74, 76, 77, 80, 86

[Dic93] Matthew T Dickerson. General polynomial decomposition and the s-1-decomposition are np-hard. *International Journal of Foundations of Computer Science*, 4(02):147–156, 1993. 95, 98

[Din18] Itai Dinur. An improved affine equivalence algorithm for random permutations. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part I*, volume 10820 of *Lecture Notes in Computer Science*, pages 413–442. Springer, 2018. 86

[Din21] Itai Dinur. Improved algorithms for solving polynomial systems over GF(2) by multiple parity-counting. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2550–2564. SIAM, 2021. 82

[DLPR13] Cécile Delerablée, Tancrède Lepoint, Pascal Paillier, and Matthieu Rivain. White-box security notions for symmetric encryption schemes. Cryptology ePrint Archive, Paper 2013/523, 2013. https://eprint.iacr.org/2013/523. 75

[DS09] Itai Dinur and Adi Shamir. Cube attacks on tweakable black box polynomials. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, volume 5479 of *Lecture Notes in Computer Science*, pages 278–299. Springer, 2009. 77, 81

[FKN94] Uriel Feige, Joe Kilian, and Moni Naor. A minimal model for secure computation (extended abstract). In Frank Thomson Leighton and Michael T. Goodrich, editors, *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, pages 554–563. ACM, 1994. 6, 16, 21, 22, 23, 42

[FP06] Jean-Charles Faugère and Ludovic Perret. Polynomial equivalence problems: Algorithmic and theoretical aspects. In Serge Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, volume 4004 of *Lecture Notes in Computer Science*, pages 30–47. Springer, 2006. 82

[FP09] Jean-Charles Faugère and Ludovic Perret. An efficient algorithm for decomposing multivariate polynomials and its applications to cryptography. *Journal of Symbolic Computation*, 44(12):1676–1689, 2009. 95, 98

[GKW15] Romain Gay, Iordanis Kerenidis, and Hoeteck Wee. Communication complexity of conditional disclosure of secrets and attribute-based encryption. In *CRYPTO (II)*, pages 485–502, 2015. 26

[GLMP18] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G. Paterson. Pump up the volume: Practical database reconstruction from volume leakage on range queries. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 315–331. ACM Press, October 2018. 47

[GLMP19] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G. Paterson. Learning to reconstruct: Statistical learning theory and encrypted database attacks. In *2019 IEEE Symposium on Security and Privacy*, pages 1067–1083. IEEE Computer Society Press, May 2019. 45, 46, 66

[GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *CHES 2001*, volume 2162 of *LNCS*, pages 251–261. Springer, Heidelberg, May 2001. 9, 19

[GMP16] Sanjam Garg, Payman Mohassel, and Charalampos Papamanthou. TWORAM: Efficient oblivious RAM in two rounds with applications to searchable encryption. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 563–592. Springer, Heidelberg, August 2016. 46, 67, 68

[GMQ07] Louis Goubin, Jean-Michel Masereel, and Michaël Quisquater. Cryptanalysis of white box DES implementations. In Carlisle M. Adams, Ali Miri, and Michael J. Wiener, editors, *Selected Areas in Cryptography, 14th International Workshop, SAC 2007, Ottawa, Canada, August 16-17, 2007, Revised Selected Papers*, volume 4876 of *Lecture Notes in Computer Science*, pages 278–295. Springer, 2007. 74, 75

[GO96] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious RAMs. *J. ACM*, 43(3):431–473, May 1996. 8, 17, 45, 49

[GRW20] Louis Goubin, Matthieu Rivain, and Junwei Wang. Defeating state-of-the-art white-box countermeasures with advanced gray-box attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(3):454–482, 2020. 74

[GSB+17] Paul Grubbs, Kevin Sekniqi, Vincent Bindschaedler, Muhammad Naveed, and Thomas Ristenpart. Leakage-abuse attacks against order-revealing encryption. In *2017 IEEE Symposium on Security and Privacy*, pages 655–672. IEEE Computer Society Press, May 2017. 66

[HK22] Shai Halevi and Eyal Kushilevitz. Random-index oblivious ram. Cryptology ePrint Archive, Paper 2022/982, 2022. 60

[IK97] Yuval Ishai and Eyal Kushilevitz. Private simultaneous messages protocols with applications. In *Fifth Israel Symposium on Theory of Computing and Systems, ISTCS 1997, Ramat-Gan, Israel, June 17-19, 1997, Proceedings*, pages 174–184. IEEE Computer Society, 1997. 6, 16, 21

[IK00] Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*, pages 294–304. IEEE Computer Society, 2000. 33, 100, 101

[KM19] Seny Kamara and Tarik Moataz. Computationally volume-hiding structured encryption. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 183–213. Springer, Heidelberg, May 2019. 67, 68

[KMO18] Seny Kamara, Tarik Moataz, and Olga Ohrimenko. Structured encryption and leakage suppression. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 339–370. Springer, Heidelberg, August 2018. 46, 49, 67

[Koc96] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 104–113. Springer, Heidelberg, August 1996. 9, 18

[LHL+18] Zheli Liu, Yanyu Huang, Jin Li, Xiaochun Cheng, and Chao Shen. DivORAM: Towards a practical oblivious RAM with variable block size. *Information Sciences*, 447:1–11, 2018. 50, 105, 106

[Lin20] Yehuda Lindell. Secure multiparty computation (mpc). Cryptology ePrint Archive, Paper 2020/300, 2020. https://eprint.iacr.org/2020/300. 4, 13

[LN04] Hamilton E. Link and William D. Neumann. Clarifying obfuscation: Improving the security of white-box encoding. Cryptology ePrint Archive, Paper 2004/025, 2004. https://eprint.iacr.org/2004/025. 75

[LN18] Kasper Green Larsen and Jesper Buus Nielsen. Yes, there is an oblivious RAM lower bound! In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 523–542. Springer, Heidelberg, August 2018. 49

[LRdM+13] Tancrède Lepoint, Matthieu Rivain, Yoni de Mulder, Bart Preneel, and Peter Roelse. Two Attacks on a White-Box AES Implementation. In *SAC 2013 - 20th International Conference Selected Areas in Cryptography*, volume 8282 of *Selected Areas in Cryptography – SAC 2013*, pages 265–285, Burnaby, British Columbia, Canada, August 2013. Springer. 75

[LVW17] Tianren Liu, Vinod Vaikuntanathan, and Hoeteck Wee. Conditional disclosure of secrets via non-linear reconstruction. In *CRYPTO Part I*, pages 758–790, 2017. 26

[MDFK15] Brice Minaud, Patrick Derbez, Pierre-Alain Fouque, and Pierre Karpman. Key-recovery attacks on ASASA. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part II*, volume 9453 of *LNCS*, pages 3–27. Springer, Heidelberg, November / December 2015. 10, 19

[MDFK18] Brice Minaud, Patrick Derbez, Pierre-Alain Fouque, and Pierre Karpman. Key-recovery attacks on ASASA. *Journal of Cryptology*, 31(3):845–884, 2018. 10, 19, 74, 75, 77, 80, 82, 83, 89, 92, 93, 94, 98

[MGH09] Wil Michiels, Paul Gorissen, and Henk D. L. Hollmann. Cryptanalysis of a generic class of white-box implementations. In Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors, *Selected Areas in Cryptography*, pages 414–428, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. 75

[MLS+13] Martin Maas, Eric Love, Emil Stefanov, Mohit Tiwari, Elaine Shi, Krste Asanovic, John Kubiatowicz, and Dawn Song. PHANTOM: practical oblivious computation in a secure processor. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 311–324. ACM Press, November 2013. 46

[MM17] Ian Miers and Payman Mohassel. IO-DSSE: Scaling dynamic searchable encryption to millions of indexes by improving locality. In *NDSS 2017*. The Internet Society, February / March 2017. 46, 67

[MOA79] Albert W Marshall, Ingram Olkin, and Barry C Arnold. *Inequalities: theory of majorization and its applications*, volume 143. Springer, 1979. 51, 59

[MR22] Brice Minaud and Michael Reichle. Dynamic local searchable symmetric encryption. In Y. Dodis and T. Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022*, Lecture Notes in Computer Science. Springer, 2022. 46, 47, 48, 105, 106

[MRP12] Yoni De Mulder, Peter Roelse, and Bart Preneel. Cryptanalysis of the Xiao - Lai white-box AES implementation. In Lars R. Knudsen and Huapeng Wu, editors, *Selected Areas in Cryptography, 19th International Conference, SAC 2012, Windsor, ON, Canada, August 15-16, 2012, Revised Selected Papers*, volume 7707 of *Lecture Notes in Computer Science*, pages 34–49. Springer, 2012. 75

[NS06] Arvind Narayanan and Vitaly Shmatikov. How to break anonymity of the netflix prize dataset. *CoRR*, abs/cs/0610105, 2006. 4, 13

[PPYY19] Sarvar Patel, Giuseppe Persiano, Kevin Yeo, and Moti Yung. Mitigating leakage in secure cloud-hosted data structures: Volume-hiding for multi-maps via hashing. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 79–93. ACM Press, November 2019. 67, 68

[RAC16] Daniel S. Roche, Adam J. Aviv, and Seung Geol Choi. A practical oblivious map data structure with secure deletion and history independence. In *2016 IEEE Symposium on Security and Privacy*, pages 178–197. IEEE Computer Society Press, May 2016. 49, 50

[RVP22] Adrián Ranea, Joachim Vandersmissen, and Bart Preneel. Implicit white-box implementations: White-boxing ARX ciphers. In *Crypto*, 2022. 10, 19, 73, 74, 75, 77, 80, 81, 82, 86, 87, 89, 91, 92, 93, 95, 98

[RW19] Matthieu Rivain and Junwei Wang. Analysis and improvement of differential computation attacks against internally-encoded white-box implementations. Cryptology ePrint Archive, Report 2019/076, 2019. https://eprint.iacr.org/2019/076. 10, 19

[RYF+13] Ling Ren, Xiangyao Yu, Christopher W Fletcher, Marten Van Dijk, and Srinivas Devadas. Design space exploration and optimization of Path Oblivious RAM in secure processors. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, pages 571–582, 2013. 106

[SESN23] Kazumasa Shinagawa, Reo Eriguchi, Shohei Satake, and Koji Nuida. Private simultaneous messages based on quadratic residues. *Designs, Codes and Cryptography*, pages 1–18, 2023. 97

[SMG16] Pascal Sasdrich, Amir Moradi, and Tim Güneysu. White-box cryptography in the gray box. In *International Conference on Fast Software Encryption*, pages 185–203. Springer, 2016. 74

[SvS+13a] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path ORAM: An extremely simple oblivious RAM protocol. Cryptology ePrint Archive, Report 2013/280, 2013. https://eprint.iacr.org/2013/280. 8, 18, 97

[SvS+13b] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher W. Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path ORAM: an extremely simple oblivious RAM protocol. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 299–310. ACM Press, November 2013. viii, 48, 60, 62, 63

[Swe00] Latanya Sweeney. Simple Demographics Often Identify People Uniquely. 1 2000. 4, 13

[TW07] Kunal Talwar and Udi Wieder. Balanced allocations: the weighted case. In David S. Johnson and Uriel Feige, editors, *39th ACM STOC*, pages 256–265. ACM Press, June 2007. 48, 106

[VRP22] Joachim Vandersmissen, Adrián Ranea, and Bart Preneel. A white-box Speck implementation using self-equivalence encodings. In Giuseppe Ateniese and Daniele Venturi, editors, *Applied Cryptography and Network Security - 20th International Conference, ACNS 2022, Rome, Italy, June 20-23, 2022, Proceedings*, volume 13269 of *Lecture Notes in Computer Science*, pages 771–791. Springer, 2022. 75

[WCS14] Xiao Wang, Hubert Chan, and Elaine Shi. Circuit ORAM: On tightness of the Goldreich-Ostrovsky lower bound. Cryptology ePrint Archive, Report 2014/672, 2014. https://ia.cr/2014/672. viii, 60, 65

[WHS12] Michael Weiß, Benedikt Heinz, and Frederic Stumpf. A cache timing attack on AES in virtualization environments. In Angelos D. Keromytis, editor, *FC 2012*, volume 7397 of *LNCS*, pages 314–328. Springer, Heidelberg, February / March 2012. 45

[Wik23] Wikipedia contributors. Empress (cracker) — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Empress_(cracker)&oldid=1170884992, 2023. [Online; accessed 28-August-2023]. 6, 15

[WMGP07] Brecht Wyseur, Wil Michiels, Paul Gorissen, and Bart Preneel. Cryptanalysis of white-box des implementations with arbitrary external encodings. In *Selected Areas in Cryptography*, pages 264–277. Springer, 2007. 74, 75

[WNL+14] Xiao Shaun Wang, Kartik Nayak, Chang Liu, T.-H. Hubert Chan, Elaine Shi, Emil Stefanov, and Yan Huang. Oblivious data structures. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014*, pages 215–226. ACM Press, November 2014. 69

[XL09] Yaying Xiao and Xuejia Lai. A secure implementation of white-box aes. *2009 2nd International Conference on Computer Science and its Applications*, pages 1–6, 2009. 75

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, November 1982. 4, 13

RÉSUMÉ

Dans cette thèse, nous étudions différentes méthodes permettant de faire des calculs sur de l'information cachée. Nous présentons de nouvelles constructions de protocoles, allant de l'amélioration des complexités de l'état de l'art à l'ajout de fonctionnalités. Nous analysons également la sécurité d'autres protocoles, et présentons des attaques dessus.
En premier lieu, nous nous intéressons au Private Simultaneous Messages, où un groupe d'individus souhaite faire apprendre à une entité externe le résultat d'un calcul sur leurs données privées. En utilisant un nouveau cadre, nous montrons de nouvelles constructions avec un coût en communication plus bas que celles déjà existantes. Ensuite, nous étudions des protocoles d'Oblivious RAM, qui dissimulent le motif d'accès qu'un client fait lorsqu'il accède à des données privées stockées sur un serveur non fiable. Nous présentons les premières constructions permettant au client d'enregistrer des objets de tailles arbitrairement différentes. Nous montrons également comment ces constructions permettent de construire des protocoles de Chiffrement Recherchable. Enfin, nous nous intéressons à la cryptographie en boîte-blanche, qui a pour objectif de donner à un appareil non fiable un accès sans restrictions à un algorithme de chiffrement, tout en empêchant la fuite de la clé secrète. Nous présentons une cryptanalyse de la construction de Ranea et al. de CRYPTO 2022, où des cryptosystèmes en boîte blanche sont donnés à partir de chiffrements symétriques basés sur des SPN et ARX. Nous prouvons que ce système n'est pas sûr en présentant une attaque structurelle de récupération de clé.

MOTS CLÉS

PSM, MPC, Boîte-Blanche, ORAM, SSE, Cryptanalyse

ABSTRACT

In this thesis, we study different methods of computing on hidden information. We present new constructions of protocols, which either improve on their state-of-the-art complexities or enable them to do more. We also analyse the security of other protocols and give attacks on them.
The first subject we consider is that of Private Simultaneous Messages, where a group of individuals wish to have an external entity learn the output of a computation on their private data. Using a new framework, we show new constructions with a smaller communication cost than existing ones. We then study Oblivious RAM protocols, that hide the access pattern that a client makes when accessing private data stored on an untrusted server. We give the first constructions that allow the client to store data items of arbitrarily different sizes. We also show how using these constructions yield new protocols for Searchable Encryption. We finish by studying white-box cryptography, where the goal is to give an untrusted device unrestricted access to an encrytpion algorithm, while preventing the secret key from leaking. We present a cryptanalysis of Ranea et al's construction from CRYPTO 2022, where they give white-box ciphers from SPN and ARX based symmetric cryptosystems. We show that this scheme is insecure by providing a structural key-recovery attack.

KEYWORDS

PSM, MPC, White-Box, ORAM, SSE, Cryptanalysis