

UNIVERSITÉ PARIS CITÉ

École doctorale Science Informatique de Paris Centre
Institut de Recherche en Informatique Fondamentale

Multiparty Computation from the Hardness of Coding Theory

Par CLÉMENT DUCROS

Thèse de doctorat de SPÉCIALITÉ INFORMATIQUE

Dirigée par

SOPHIE LAPLANTE

et co-dirigée par

GEOFFROY COUTEAU

et ALAIN COUVREUR

Présentée et soutenue publiquement le 12/11/2024

Devant un jury composé de :

NICOLAS SENDRIER	Directeur de recherche	INRIA Paris	Rapporteur
EMMANUELA ORSINI	Tenure-Track Assistant Professor	Bocconi University	Rapportrice
YIXIN SHEN	Chargé de recherche	INRIA Rennes	Examinatrice
PHILIPPE GABORIT	Professeur	CNRS, Université de Limoges	Examinateur
LISA KOHL	Researcher	CWI Amsterdam	Examinatrice
NICO DÖTTLING	Professeur	Helmholtz Center for Information Security (CISPA)	Examinateur
SOPHIE LAPLANTE	Professeure	Université Paris Cité, IRIF	Directrice de thèse
GEOFFROY COUTEAU	Chargé de Recherche	CNRS, IRIF, Université Paris Cité	Co-directeur de thèse
ALAIN COUVREUR	Directeur de Recherche	INRIA Saclay, École polytechnique	Membre invité

Résumé en Français

Depuis l'apparition de la cryptographie moderne, le *calcul sécurisé* (MPC) a longuement été étudié. Le MPC permet à un groupe de parties (ou joueurs) d'effectuer ensemble un ou plusieurs calculs sur des données secrètes, tout en garantissant que les joueurs n'obtiennent rien de plus que le résultat et ce qui peut en être déduit. Le développement massif de notre utilisation quotidienne d'Internet rend le besoin de méthodes de calcul sécurisé plus urgent que jamais : les calculs sur des données privées sont omniprésents, des algorithmes de recommandation aux enchères électroniques. Les algorithmes de MPC modernes bénéficient grandement de l'apport d'aléa corrélé, afin d'obtenir des protocoles rapides. Cet aléa corrélé doit être généré massivement au préalable pour que les protocoles de MPC puissent être efficaces. Toutefois, réaliser cette génération d'aléa corrélé efficacement reste encore un défi.

Dans cette thèse, nous étudions les Générateurs de Corrélations Pseudo-aléatoires (PCGs). Cette primitive cryptographique transforme une petite quantité d'aléa corrélé en une large quantité de pseudo-aléa corrélé, tout en étant efficace en ce qui concerne le temps de calcul et la communication. Nous présentons différentes constructions de PCG dont la sécurité repose sur des variantes de l'hypothèse de Décodage de Syndrome (SD), une hypothèse classique en théorie des codes. L'intuition derrière les constructions proposées est de mélanger l'hypothèse SD avec certaines techniques de MPC qui permettent de partager de manière additive des vecteurs creux. Nos résultats obtenus battent l'état de l'art en ce qui concerne les protocoles de calcul sécurisé nécessitant plus de deux joueurs lorsque le calcul est effectué sur un corps fini de taille > 2. Nous montrons également comment supprimer cette contrainte au prix d'un léger surcoût en communication.

Nous considérons également la construction de Fonctions Pseudo-aléatoires Corrélées (PCFs). Les PCFs produisent des corrélations à la volée : les joueurs détiennent chacun des fonctions corrélées de sorte que les sorties des fonctions, lorsqu'elles sont évaluées sur la même entrée, soient corrélées. Ces objets offrent plus de flexibilité que les PCGs, mais sont aussi plus difficiles à construire. Nous nous appuyons à nouveau sur des variantes de SD pour construire des PCFs. Nous nous basons sur une construction de PCF déjà établie, montrant que la preuve de sécurité associée était incorrecte, et proposons une correction. De plus, nous présentons des optimisations de la construction, provenant d'une meilleure analyse et d'un calcul précis des paramètres nécessaires via des simulations. Nous obtenons des paramètres utilisables en pratique.

Cette thèse repose majoritairement sur l'hypothèse SD. La recherche d'une bonne complexité implique des efforts pour trouver les codes les plus efficaces tout en garantissant que la sécurité ne soit pas compromise. Nous considérons un modèle permettant l'étude de l'efficacité des attaques prometteuses contre SD et ses variantes. Pour chaque construction mentionnée précédemment, nous menons une analyse de sécurité approfondie de la variante associée de SD et tentons d'attaquer notre

propre schéma pour calculer des paramètres concrets.

Mots Clés : Calcul Sécurisé, Correlation Pseudo-aléatoire, Théorie des Codes, Décodage de Syndrome, Generateurs de Correlations Pseudo-aléatoires, Fonction Pseudo-aléatoire Correlées.

Abstract

Since the beginning of modern cryptography, the question of *secure computation* (MPC) has been extensively studied. MPC allows for a set of parties to perform some joint computation while ensuring that their input remains secure up to what is implied by the output. The massive development of our daily Internet usage makes the need for secure computation methods more urgent : computations on private data is everywhere, from recommendation algorithms to electronic auctions. Modern MPC algorithms greatly benefit from correlated randomness to achieve fast online protocols. Correlated randomness has to be massively produced beforehand for the MPC protocols to work best, and this is still a challenge to do efficiently today.

In this thesis, we study pseudorandom correlation generators (PCGs). This construction transforms small amount of correlated randomness into a large amount of correlated pseudorandomness with minimal interaction and local computation. We present different PCG constructions whose security relies on variants of the Syndrome Decoding (SD) assumption, a classical assumption in coding-theory. The intuition behind these constructions is to merge the SD assumption with some MPC techniques that enable additively sharing sparse vectors. We achieve state-of-the-art results regarding secure computation protocols requiring more than two players when the computation is over a finite field of size > 2. We show how to remove this constraint at the cost of a small overhead in communication.

Next, we consider the construction of pseudorandom correlation functions (PCFs). PCFs produce correlations on-the-fly : players each hold correlated functions such that the outputs of the functions, when evaluated on the same entry, are correlated. These objects offer more flexibility than PCG, but are also harder to construct. Again, we rely on variants of SD to construct PCFs. We build on a previous PCF construction, showing that the associated proof of security was incorrect, and propose a correction. Additionally, we present optimizations of the construction, coming from a better analysis and precise optimization of parameters via simulations. We achieve parameters usable in practice.

Finally, this thesis revolves around the use of certain codes, particularly the SD assumption. The search of good complexity entails efforts to find the most efficient codes while ensuring that security is not compromised. We consider a framework tailored to the study of promising attacks on SD and its variants. For each construction previously mentioned, we conduct a thorough security analysis of the associated variant of SD and attempt cryptanalysis efforts to compute concrete parameters.

Keyword : Secure Computation, Pseudorandom correlation, Coding Theory, Syndrome Decoding, Pseudorandom Correlation Generator, Pseudorandom Correlation Function.

Résumé Substantiel en Français

Le Calcul Sécurisé

Garantir qu'un message puisse être envoyé sans qu'aucune tierce personne ne puisse le lire n'est pas chose aisée. Ce problème, très naturel, a donné naissance à la cryptographie, que l'on pourrait informellement surnommer "l'art du message secret". La cryptographie utilise des outils mathématiques pour transformer un message en message chiffré, c'est-à-dire un message incompréhensible. De la même manière qu'il faut que le destinataire et l'expéditeur possèdent tous les deux la même clé pour ouvrir une boîte fermée par un verrou, la cryptographie suppose souvent que les deux joueurs possèdent une clé commune (ou ayant un lien). L'expéditeur peut alors utiliser cette clé pour chiffrer le message, le transformant en message chiffré. Le destinataire, grâce à sa propre clé, peut alors déchiffrer le message chiffré et obtenir le message en clair. La force de la cryptographie réside dans sa capacité à prouver que, sans clé, le message chiffré est bel et bien incompréhensible. La cryptographie, dont l'utilisation est très ancienne, a connu une véritable révolution à partir des années 1980, résultant de l'essor des ordinateurs et du développement d'Internet. Aujourd'hui, la cryptographie fait partie intégrante de notre quotidien. Nous vivons dans un monde plutôt sécurisé en ce qui concerne la communication de messages.

Toutefois, notre utilisation d'Internet va aujourd'hui bien au-delà de la simple communication privée. Les sites utilisent de la publicité ciblée ou des algorithmes de recommandation : on peut par exemple citer Netflix pour ses séries, YouTube pour ses vidéos, Facebook pour des recommandations d'amis. Les applications citées plus haut nécessitent forcément l'utilisation de données personnelles. Cette question des données personnelles est un point sensible. En effet, l'utilisateur ne réalise pas forcément toujours que ses données ne sont plus privées. Il y a donc un paradoxe apparent : si d'une part la population prend graduellement conscience de l'importance de ses données privées, la plupart des utilisateurs acceptent de partager leurs données privées avec des entreprises en échange des services qu'elles ont à proposer.

Mais la cryptographie n'a pas dit son dernier mot ! En effet, le *calcul sécurisé* (MPC) permet justement à différents joueurs d'exécuter un algorithme sur des données privées tout en maintenant cachées ces données durant l'exécution du protocole. Formellement, étant données des entrées $x_1, \ldots, x_N$ avec N le nombre de joueurs impliqués dans le calcul, le MPC permet aux joueurs de calculer une fonction f, de manière à ce qu'à la fin du protocole chaque joueur obtienne $f(x_1, \ldots, x_N)$. Le calcul sécurisé regorge d'applications, de la recommandation d'amis sur les réseaux sociaux aux enchères électroniques en passant par des calculs statistiques sur des données sensibles, comme par exemple des statistiques entre hôpitaux tout en préservant le secret médical. Toutefois, prouver l'existence de protocoles réalisant toutes ces applications n'est pas suffisant : pour qu'un usage massif

de calcul sécurisé sur Internet et sur les réseaux sociaux soit possible, les algorithmes de calcul sécurisé doivent être efficaces. Ainsi, cette thèse a pour but de rendre utilisables en pratique les algorithmes de MPC.

Techniques pour le Calcul Sécurisé

Dans cette thèse, nous nous concentrons sur une technique introduite par Goldreich, Micali et Wigderson en 1987 : le protocole GMW [GMW87]. Supposons le cas du MPC à deux joueurs, disons Alice et Bob. Ce protocole utilise massivement de l'aléa corrélé. Qu'est-ce que l'aléa corrélé ? Nous le définissons comme étant des valeurs additionnelles données à Alice et Bob telles qu'il existe une relation entre leurs valeurs. Ces valeurs sont aléatoires sous condition de satisfaire cette relation (ou corrélation). Un exemple fondateur de corrélation, et qui fut justement celui utilisé dans le cas du protocole GMW, est le transfert inconscient (OT). Un transfert inconscient consiste à donner à Alice un couple (x_0, x_1) de valeurs, et à Bob un bit de sélection b ainsi que la valeur x_b correspondante. Les garanties de sécurité sont les suivantes : Alice ne connaît pas la valeur de b ; Bob ne connaît pas la valeur de x_{1-b}. Le protocole GMW utilise classiquement deux transferts inconscients par porte ET dans le circuit booléen. Il a été montré par Beaver en 1992 [Bea91] que ces transferts inconscients (ou d'autres corrélations) pouvaient en réalité être précalculés, avant même le début du protocole. Ceci a donné naissance à une vision du MPC en deux parties : une première phase de précalcul où les joueurs génèrent l'aléa corrélé dont ils ont besoin. L'utilisation de l'aléa corrélé permet une deuxième phase, dite phase en ligne, où les joueurs exécutent un protocole très rapide grâce à l'aléa corrélé. Toutefois, un problème demeure : la quantité d'aléa corrélé requise pour un circuit donné est énorme, car les fonctions que nous considérons peuvent facilement comporter des milliards de portes ET. Étant donné qu'il n'est pas possible de réutiliser de l'aléa une fois celui-ci utilisé dans le protocole, la production de l'aléa corrélé pose problème. Notons par ailleurs qu'une simple instance d'aléa corrélé nécessite l'utilisation de cryptographie à clé publique, ce qui, étant donné le nombre d'appels que nous devons faire, est simplement impossible.

C'est pour répondre à ce besoin qu'est née l'idée de *Générateur de Corrélation Pseudo-aléatoire* (PCG). À l'origine, les OTs qui doivent être générés pour le protocole en ligne doivent être vraiment aléatoires. Toutefois, nous pouvons nous satisfaire d'OTs (ou plus généralement, de corrélations) *pseudo-aléatoires*[1]. Le but d'un PCG est ainsi de transformer une certaine quantité de véritable aléa corrélé en une large quantité de pseudo-aléa corrélé. Ce faisant, l'objectif est d'avoir un coût en communication faible par rapport à la quantité d'aléa générée, tout en garantissant un temps de calcul faible. Cette thèse porte aussi sur une autre construction proche des PCG, les *Fonctions Pseudo-aléatoires Corrélées* (PCF) qui visent à générer des corrélations à la volée.

PCG et PCF sont construits grâce à deux concepts clés :

— *Fonction Point Distribuée* (DPF) : Cette primitive, introduite par Boyle, Gilboa et Ishai en 2015 [BGI15], permet à deux joueurs d'obtenir respectivement une fonction pseudo-aléatoire f_0, f_1, telles que la somme $f_0 + f_1$ soit égale à une fonction non-nulle en une seule entrée (en un seul point). Cette primitive permet aux deux parties de partager de manière efficace un long vecteur unitaire[2], c'est-à-dire, qui donne un vecteur pseudo-aléatoire $\mathbf{u_0}$ à Alice et un vecteur pseudo-aléatoire $\mathbf{u_1}$ à Bob, tels que $\mathbf{u_0} \oplus \mathbf{u_1}$ soit égal à un vecteur unitaire.

— L'hypothèse de *Décodage de Syndrome* (SD) : Il s'agit d'une hypothèse classique en théorie des codes, qui assure qu'il est difficile de distinguer $(\mathbf{H}, \mathbf{s})$ de $(\mathbf{H}, \mathbf{He})$, avec $\mathbf{H}$ une matrice

1. Signifiant qu'aucun algorithme efficace ne peut distinguer des OTs pseudo-aléatoires de OTs vraiment aléatoires.
2. Si le vecteur unitaire est de taille n, nous voulons que les parts puissent être compressées avec $O(\log(n))$ bits.

aléatoire, $\mathbf{s}$ un vecteur aléatoire et enfin $\mathbf{e}$ un vecteur creux aléatoire, c'est-à-dire avec peu de coordonnées non nulles.

Les constructions ainsi créées nécessitent, de par l'utilisation de SD, des multiplications de matrices. Celles-ci peuvent être très coûteuses, car les matrices considérées sont très grandes : elles sont de l'ordre du nombre de OTs que l'on veut générer (ou de l'ordre de la quantité d'aléa corrélé que l'on désire pour être moins spécifique). Il est donc naturel de considérer des optimisations pour réduire les coûts de calcul. Une piste prometteuse est la suivante : plus haut, il est fait mention du fait que $\mathbf{H}$ est une matrice purement aléatoire, dans l'hypothèse de décodage de syndrome dans sa version classique. Ce n'est toutefois pas forcément nécessaire. Changer de matrice, introduire certaines formes de régularité, peut ainsi permettre de gagner en efficacité. Toutefois, une attention toute particulière est requise pour s'assurer que l'hypothèse de décodage de syndrome tient toujours, même quand on restreint $\mathbf{H}$. Ceci était le point de départ de cette thèse : faire un pont entre le calcul sécurisé d'une part et la théorie des codes d'autre part. Dans cette thèse, nous construisons ainsi toutes nos primitives en utilisant des variantes spécifiques de l'hypothèse de décodage de syndrome, dont nous analysons scrupuleusement la sécurité.

Contenue de la Thèse

Le chapitre 2 introduit formellement les connaissances préalables nécessaires pour comprendre cette thèse. Le chapitre 3 définit précisément les PCGs et PCFs.

Le Modèle des Attaques Linéaires

L'hypothèse de décodage de syndrome (SD) est une hypothèse très classique en cryptographie, introduite il y a plus de quarante ans. Étant donné son ancienneté, elle a fait l'objet de nombreuses études et attaques. Comment, dans ces conditions, s'assurer qu'un adversaire ne réussisse pas à trouver une faille ? Il a été judicieusement observé par [BCGI+20a] que la majorité des attaques sur SD consistent essentiellement en des opérations linéaires sur le syndrome $\mathbf{s}$, opérations qui ne dépendent que de la matrice $\mathbf{H}$. Cela implique que ces attaques peuvent être représentées par un vecteur d'attaque, dont le produit scalaire avec $\mathbf{s}$ peut être biaisé ou non (selon que l'on est dans le cas $(\mathbf{H}, \mathbf{s})$ ou $(\mathbf{H}, \mathbf{He})$). [BCGI+20a] introduisit alors le *modèle des attaques linéaire*, qui peut se résumer ainsi : tout d'abord, la matrice $\mathbf{H}$ est donnée à l'adversaire. L'adversaire possède alors un temps illimité pour déterminer un vecteur d'attaque $\mathbf{v}$. Pendant ce temps, on génère un vecteur creux $\mathbf{e}$ aléatoire. L'adversaire gagne si il réussit à déterminer un $\mathbf{v}$ tel que $\mathbf{v}^\intercal\mathbf{He}$ ait un biais significatif. L'intéret de se modèle est de montrer que si on peut montrer que pour tout vecteurs d'attaque le bias est majoré par une certaine valeur avec forte probabilité, alors cela implique une borne inférieur sur le temps de toutes les attaques dites "linéaire". Cette classe d'attaque, comme dit initiallement est très large, et contient notamment les algorithmes d'*information set-decoding* (ISD), les plus prometteuses contre SD. Le chapitre 4 fait l'état des lieux de ce formalisme récent, prouvant notamment comment les ISD peuvent s'intégrer correctement dans ce modèle, et montrant également comment il est possible de montrer une résistance aux attaques linéaires en via des propriétés de la matrice $\mathbf{H}$.

Générateur de Corrélation Pseudo-aléatoire à partir de l'Hypothèse de Décodage de Syndrome QA-SD

Le travail de cette partie provient de deux articles, l'un publié à Crypto2023, l'autre disponible sur eprint [BCCD23 ; BBCC+24].

Le chapitre 5 traite de la création d'un PCG pour la corrélation OLE[3]. Cette corrélation est définie ainsi : elle donne (u, v) au premier joueur et $(\Delta, w \coloneqq u\Delta + v)$ au second, avec u, v, w, Δ des éléments d'un anneau. Notre construction s'appuie notamment sur le travail de [BCGI+20b], dont la sécurité repose sur une variante de SD pour des anneaux de polynômes : il s'agit de distinguer (a, y) de $(a, a \cdot e_1 + e_2)$ avec a, y des polynômes sur $\mathbb{F}_q[X]$, et e_1, e_2 des polynômes creux, c'est-à-dire avec peu de coefficients non nuls. L'approche de Boyle et al. était de créer une seule corrélation sur un anneau $\mathcal{R} = \mathbb{F}_q[X]/(P(X))$, avec $P(X)$ un polynôme factorisable en produit de facteurs linéaires, puis d'utiliser le Théorème des restes chinois, qui assure l'existence d'un isomorphisme entre $\mathcal{R}$ et $\mathbb{F}_q^n$. En appliquant cet isomorphisme sur l'unique OLE préalablement produite, ils pouvaient obtenir de nombreuses instances de la corrélation sur $\mathbb{F}_q$.

Dans cette thèse, nous présentons une nouvelle variante de SD, avec l'hypothèse de décodage de syndrome quasi-abélien (QA-SD). Notre but est de se débarrasser d'une contrainte pénible de la construction de Boyle et al. : les OLEs produits ne l'étaient que sur des corps suffisamment grands (à savoir plus grands que le nombre d'OLEs que l'on souhaite générer). La construction proposée dans ce manuscrit utilise un anneau différent, en passant au multivarié : formellement, en prenant $\mathcal{R} = \mathbb{F}_q[X_1, \ldots, X_n]/(X_1^{q-1} - 1, \ldots, X_n^{q-1} - 1)$, nous pouvons réduire la contrainte sur la taille du corps à $q \geqslant 3$. Il apparaît que cet anneau est toujours isomorphe à de nombreuses copies de $\mathbb{F}_q$. Comment pouvons-nous créer l'OLE sur cet anneau ? Nous suivons les grandes lignes de [BCGI+20b] : nous pouvons créer u et Δ comme des éléments pseudo-aléatoires de l'anneau résultant de l'hypothèse QA-SD, c'est-à-dire $u = ae_1 + e_2$, pour e_1 et e_2 des polynômes creux. En faisant de même pour Δ, il apparaît que le produit $u\Delta$ fait intervenir a, a^2, mais surtout des produits de polynômes creux. Un point important est alors de remarquer que le produit de polynômes creux reste creux. Nous pouvons alors utiliser la primitive DPF. Cette primitive nous permet de partager additivement des vecteurs unitaires. Mais étant donné qu'un vecteur creux n'est finalement rien d'autre qu'une somme de vecteurs unitaires, il est possible de partager un vecteur creux. Or, il est possible également de voir un polynôme comme un vecteur, en regardant sa liste de coefficients. Il devient alors possible de partager additivement ces produits creux et ainsi de partager $u\Delta$. Il ne reste plus qu'à définir ces parts comme étant v et w.

Il faut maintenant prouver la sécurité de l'hypothèse sous-jacente, c'est-à-dire QA-SD, dans le cadre des polynômes multivariés. Pour ce faire, nous pouvons compter sur l'apport de la théorie des codes : le code ainsi défini sur $\mathcal{R}$ peut être vu comme un code de quasi-groupe. Pour montrer que l'hypothèse est valable, nous utilisons le modèle des attaques linéaires.

La thèse présente également une manière d'obtenir des triplets de Beaver sur $\mathbb{F}_2$, une autre corrélation fondamentale. Ceci est fait en utilisant une astuce, au prix d'une augmentation raisonnable du temps de calcul et de la communication. L'astuce consiste à produire des OLEs sur $\mathbb{F}_4$ en s'appuyant sur notre précédent travail, puis à les transformer en triplets de Beaver sur $\mathbb{F}_2$. Nous montrons comment optimiser la génération d'OLEs sur $\mathbb{F}_4$. Nous arrivons avec cette construction à battre l'état de l'art sur la génération de triplets de Beaver avec environ 10 millions de triplets générés par seconde. Le chapitre 6 concerne l'analyse de sécurité de l'hypothèse QA-SD. Nous tentons d'attaquer notre propre schéma pour en extraire des paramètres résistants à nos tentatives, avec une marge de sécurité standard à 128 bits.

Fonctions Pseudo-aléatoires Corrélées par une Hypothèse de SD avec Densité Variable

Le travail de cette partie provient d'un article publié à PKC2023 [CD23].

3. Évaluation Linéaire Inconsciente

Dans le chapitre 7, cette thèse aborde la construction de Fonctions Pseudo-aléatoires Corrélées (PCF). Le but est de générer des clés courtes k_0 et k_1 à donner aux joueurs, telles que les clés décrivent les fonctions f_{k_0} et f_{k_1}. Sur n'importe quelle entrée x, $f_{k_0}(0)$ et $f_{k_1}(0)$ doivent être corrélées. L'avantage d'une PCF par rapport à un PCG est que l'aléa corrélé n'est pas généré d'un coup, mais sur demande, ce qui offre une bien meilleure flexibilité. Toutefois, cet avantage entraîne également une plus grande difficulté de mise en place.

Le travail présenté ici s'appuie sur une construction que nous devons à [BCGI+20a]. La construction repose elle aussi sur une nouvelle hypothèse de décodage de syndrome : SD avec densité variable (VDSD). Cette fois-ci $\mathbf{H}$ est construite comme une concaténation de matrices dont la densité (poids de la ligne divisé par la taille de cette ligne) décroît strictement, et $\mathbf{e}$ est généré comme le serait une ligne de la matrice. La forme de la matrice à densité variable permet de faire des multiplications de manière efficace, alors que la matrice est supposément très large (mais donc très creuse). Concernant la sécurité de cette hypothèse, [BCGI+20a] avait pour objectif de montrer que cette construction était sécurisée dans le modèle des attaques linéaires. Nous montrons que la preuve de sécurité de cette construction contenait quelques erreurs que nous corrigeons. Nous proposons également une nouvelle preuve de sécurité sur une version légèrement modifiée de la matrice $\mathbf{H}$. Cette nouvelle analyse, plus fine, accompagnée de cette nouvelle matrice qui réduit des effets de bord indésirables, et une fine optimisation des paramètres permet de rendre la construction presque utilisable en pratique : nous atteignons de l'ordre de 2000 corrélations générées par seconde [4].

4. contre environ 100k pour la PCF état-de-l'art

Acknowledgments

These paragraphs mark the final words I will write for this thesis. I have deliberately postponed this moment, but as the defense approaches, I think it is time to reflect on the past three years. Many thanks are in order, and expressing my gratitude to everyone who made this journey possible will not be easy. I am sure I will inevitably forget a few names—if yours is missing, please know it is due to my imperfect memory, not my lack of appreciation. This section will remain unproofed by anyone other than myself and, of course, the various online tools at my disposal [5]. I apologize for any errors that may appear.

I would not have had the opportunity to complete this thesis without the help of my brilliant and kind advisors, Geoffroy and Alain.

Geoffroy, I consider myself very lucky to have been your student. I will always be impressed by your intelligence, the number of ideas you can have, and your ability to handle multiple projects at once [6]. Thank you for your kindness and patience during my moments of confusion. I have learned so much from you, and I am grateful that you introduced me to the exciting world of MPC—back when I was still a master's student at Telecom. These three and a half years have flown by, and I know I will feel nostalgic for my time as your student for a long time.

I also want to thank you, Alain, for your benevolence toward me and your dedication to help, even when you were supposed to be on holiday! I greatly enjoyed our discussions and admired your approach to research. Thank you for introducing me to the wonderful French coding community. I am also grateful for all your good advices, moral support, and your detailed comments on my manuscript.

I would like to warmly thank all the members of my jury—Nicolas Sendrier, Emmanuela Orsini, Yixin Shen, Philippe Gaborit, Lisa Kohl, Nico Döttling, and Sophie Laplante—for agreeing to dedicate their time to my defense. Coordinating a date with so many people was not easy, and I truly appreciate the flexibility you have shown. Special thanks to Nicolas and Emmanuela for agreeing to proofread my thesis. I hope you enjoy reading it.

I would also like to express my heartfelt gratitude to my co-authors. Maxime, thank you for all the explanations you provided when I was confused; I have learned so much from you. I also want to thank Sacha and Dung, with whom I learned a great deal during our meetings. I hope we can collaborate on more projects together in the future.

I feel fortunate to have enjoyed a pleasant PhD experience, which would not have been possible without all the wonderful people who make life at IRIF so enjoyable. I would like to thank Vincent, Victor, Klara, Lucie, Shamisa, Roman, and all the people with whom I had great times during our

5. A true blessing for dyslexic people like me!
6. Among them, creating life!

(perhaps too long!) lunch breaks. Special thanks to Klara for all the moments we spent commiserating about the challenges of writing the manuscript. Thanks to Daniel and Enrique for their kindness and for enriching my cultural life by inviting me to the opera several times. Thanks to Eliana and Dung for sticking together through all the conferences we attended. And thanks to Esaïe and Bernardo for the fun times playing Go!

I would like to thank all the people who guided me in my research and encouraged me to pursue scientific studies. In particular, I would like to thank Pierrick Baudet, my high school math teacher, who pushed me to do better and strengthened my already present passion for math.

I would also like to thank the secretariat for their continuous help in preparing my missions and for their kindness. Thanks as well to all the professors for whom I taught TD and TP sessions during my PhD. I really enjoyed teaching—it was a refreshing break from constant research. I am very grateful to everyone in the GRACE team at INRIA Saclay, where I was a part-time resident.

Additionally, I would like to thank all the participants and organizers of the events in the code-based cryptography community in France, such as the Retraite Barracuda and the Journée C2, which I greatly enjoyed. Research is better as a group effort, and I will never forget the many presentations and exciting discussions, made even more enjoyable by the games and traditional karaoke evenings.

I would like to thank all my friends who supported me during these three years and before. Special thanks to Thomas for his moral support, the movie nights, and the travels we have shared. Thanks also to Aurélien, Bruno, and Bastien for all the fun we have had since high school, and to Lucas and Arnaud for accompanying me during the challenging years of preparatory school [7]. Thanks to Antoine for making me smile in any circumstances. I would also like to thank my dear friends from Telecom: Florian, Gabriel, Maxime, Louis, Guillaume, Zoe, Clément, and many others. During the thesis writing process, I was also fortunate to find hard-working people on a certain Discord server who gave me some moral support: I would like to thank the L, Z, Z, V, M, P, J, and W [8].

Enfin, j'aimerais remercier ma famille, qui m'a continuellement encouragé pendant toutes ces années. Merci à mes parents, qui ont toujours veillé à ce que j'aille bien. Merci, Maman, d'avoir pris sur toi, même si tu n'étais pas très rassurée à l'idée de me voir faire une thèse. Merci, Papa, pour tes petits mots le matin et ta gentillesse lorsque je rentre à la maison. Merci, Vincent, pour tes invitations chez toi et pour tout ce que tu m'as fait découvrir depuis que je suis petit. Je devrais venir plus souvent. Merci, Delphine, pour tes conseils sur la thèse, ta gentillesse, et ton exemple, et enfin, pour m'avoir fait devenir le tonton des plus mignons des enfants ! Enfin, j'aimerais évidemment remercier le Chat, car dans notre famille, c'est important [9].

7. And bravo for predicting that I would pursue research before I even did!
8. And many others, but there are too many pseudonyms to list
9. Mais ouais !!!

Contents

List of Figures

List of Tables

List of acronyms

BT	*Beaver Triple*	28
FSS	*Function Secret Sharing*	28
LPN	*Learning Parity with Noise assumption*	16
OLE	*Oblivious Linear Evaluation*	26
OT	*Oblivious Transfer*	25
PCF	*Pseudorandom Correlation Function*	43
PCG	*Pseudorandom Correlation Generator*	40
PRF	*Pseudorandom Function*	11
PRG	*Pseudorandom Generator*	11
QA-SD	*Decisional Quasi-Abelian Syndrome Decoding assumption*	20
ROLE	*Random Oblivious Linear Evaluation*	27
ROT	*Random Oblivious Transfer*	26
$\mathcal{R}$-SD	*Decisional Syndrome Decoding over a ring*	18
SD	*Decisional Syndrome Decoding assumption*	16
SPFSS	*Distributed Multipoint Point Functions*	32
VDSD	*Variable Density Syndrome Decoding*	101
VOLE	*Vector Oblivious Linear Evaluation*	26
WPRF	*Weak Pseudorandom Function*	12
rVDSD	*Regular Variable Density Syndrome Decoding*	102
MPC	*Secure Multiparty Computation*	21

List of Symbols

Publications

Parts of the thesis are taken from the following publications:

1. [CD23]: Pseudorandom Correlation Functions from Variable-Density LPN, Revisited.
   - Co-author with: Geoffroy Couteau
   - Status: Published at PKC 2023, Part II. Ed. by Alexandra Boldyreva and Vladimir Kolesnikov. Vol. 13941. LNCS. Springer, Heidelberg, May 2023, pp. 221– 250.
   - eprint: https://eprint.iacr.org/2023/650.pdf
   - Presented in Chapter 6.
2. [BCCD23]: Correlated Pseudorandomness from the Hardness of Quasi- Abelian Decoding.
   - Co-author with: Maxime Bombar, Geoffroy Couteau and Alain Couvreur.
   - Status: CRYPTO 2023, Part IV. LNCS. Springer, Heidelberg, Aug. 2023, pp. 567–601.
   - eprint: https://eprint.iacr.org/2023/845.pdf
   - Presented in Chapter 5.
3. [BBCC+24]:FOLEAGE: F4OLE-Based Multi-Party Computation for Boolean Circuits.
   - Co-author with: Maxime Bombar, Dung Bui, Geoffroy Couteau, Alain Couvreur and Sacha Servan-Schreiber,.
   - Status: pre-print.
   - eprint: https://eprint.iacr.org/2024/429
   - Presented in Chapters 5 and 7.

Chapter 1

Introduction

1.1 Cryptography

1.1.1 Private Communication

Imagine you are trying to send a confidential message to a friend. How can you ensure that no one else intercepts and reads it? A child's solution might be to place the message in a locked box and send it, assuming that the recipient also holds the key to unlock it. While simple, this method is not satisfactory: if someone else finds the box, they could potentially break it open and access the message. Another method is steganography, which consists in hiding the message within another message, such that the presence of the additional message is not evident to a non-aware gaze. For example, who has never heard of the famous lemon juice that makes invisible ink, or of hiding secret data within an image by subtly altering its pixels? However, these methods rely on the secrecy of the technique itself. Once the method is discovered, all hidden messages can be exposed, and the technique cannot be reused.

This is where *cryptography* comes in, offering a more sophisticated and reliable solution. Cryptography uses mathematical algorithms to secure messages, replacing physical locks and keys with digital ones. Both the sender and receiver share a secret key. The sender uses this key to encrypt the message, transforming it into a ciphertext, which can be thought of as putting the message in a digital box. The recipient, who also possesses the key, can decrypt the ciphertext and retrieve the original message. Cryptography's strength lies in its ability to ensure that without the key, the encrypted message remains secure, even if the encryption method is known. Many ciphers have been invented over time, but a lot get broken [1] either by humans or by computers. The emergence of the digital age and of the Internet sped up the transformation of cryptography as an important science, with precise methods to analyze if a cipher is secure or not. Today, we can prove that a cryptosystem is secure by demonstrating that breaking it requires solving problems that are complex even for powerful computers. Cryptography has gradually become part of our daily lives. For instance, an estimated 85% of websites uses the TLS protocol for secure Internet communication, a figure that rises to 95% for web pages recommended by browsers like Google or Bing. Nowadays, we live in a world (relatively) secure regarding communication: anyone can send a message to a friend with confidence, knowing that it cannot be read by unauthorized parties, including governments.

1. meaning that it becomes possible to retrieve the message without having to know the key in the first place.

1.1.2 Secure Multiparty Computation

While secure communication is crucial, our daily use of the Internet involves much more. In the early days of the Internet, activities primarily revolved around communication, such as sending emails and browsing websites. Today, our Internet usage has expanded significantly. People use smartphones to monitor various aspects of their lives, from athletes tracking their performance to parents seeking meal ideas. Modern websites and apps also employ targeted advertising and recommendation algorithms, suggesting products or contents based on users' preferences and browsing history. These recommendation tools are everywhere: Netflix recommending shows, YouTube highlighting videos, Facebook suggesting friends, and even Tinder matching users with potential partners. These recommendation systems require data, often collected without users realizing that their personal data are *no longer private*. This raises a paradox: people value privacy but frequently share personal data with large corporations in exchange for the service they provide.

Fortunately, cryptography offers a solution to this paradox through Secure Multiparty Computation (MPC). MPC allows for the execution of an algorithm on private data while keeping the data itself hidden. For example, a platform like YouTube could receive encrypted user preferences and still use them to suggest relevant videos without ever seeing the actual preferences. Formally, given inputs $x_1, \ldots, x_N$ from N different parties, MPC lets them compute a function f such that each party receives the result $f(x_1, \ldots, x_N)$ and nothing more. The applications of MPC are countless. In addition to the previously mentioned advertising application, it can also be used for:

— *Extracting Intersection of Data Records*: A social network can recommend connecting with your friends who also have accounts. To achieve this, it needs to identify the overlap between its list of users and your list of friends, without disclosing all users to you or learning about your friends who are not on the platform.

— *Government and Public Policy*: Government can analyze data for policy-making without sharing sensitive information. For instance, tax authorities and social services can calculate statistics on income distribution without exposing individual records.

— *Banking Fraud Detection*: Banks can collaboratively identify suspicious transaction patterns without revealing individual transaction details.

Therefore, MPC addresses the challenge of maintaining privacy while performing computations on private data. However, for practical use on social media and the Internet, MPC protocols need to be efficient. While protocols exist, achieving true efficiency remains a major challenge. This thesis aims to improve MPC methods.

But how does secure computation work? MPC emerged in the early days of modern cryptography with the seminal work of Yao in 1982 [Yao82]. In 1987, Goldreich, Micali, and Wigderson introduced the GMW protocol [GMW87], which operates on the Boolean circuit representation of the function to be computed. The parties securely compute the circuit by calling a subroutine for each multiplication gate: a protocol called Oblivious Transfer (OT). An oblivious transfer involves a sender holding two values, x_0 and x_1, and a receiver with a selection bit b. The sender transfers x_b to the receiver without knowing b, and the receiver gets x_b without learning x_{1-b}. Originally, the values x_0, x_1 and b are determined by the inputs to the GMW protocol, and therefore one needs the inputs of GMW before computing the OT protocols.

But in 1992 Beaver [Bea91] showed that the OTs could be first precomputed with random inputs, and later adapted with the actual inputs. This leads to a two-phases MPC protocol: a preprocessing phase where the parties construct their random oblivious transfers, and an online phase where the parties use the oblivious transfers they have constructed. This separation reduces the communication

and computational costs during the actual computation (the online phase). But there is a caveat: the computation of each multiplication gates *consumes* 2 OTs, meaning that we cannot reuse them a second time. This is a problem because Oblivious Transfer is not cheap to produce, and above all, a huge quantity of them must be generated. The cost in both computation and communication becomes tremendously big considering that the number of multiplication gates in the boolean circuit of a function that we could consider is likely to be in the billions. Therefore, something has to be done to make the preprocessing phase practical.

Today's solution is to transform efficiently a small amount of OTs into a large amount of OTs, by only dropping one of their characteristics. Originally the oblivious transfers produced during the online phase were supposed to be generated using true randomness and being independent from one another. By dropping this property, we settle for pseudorandomness [2]: if the oblivious transfers produced are now not totally independent from one another, they act in every way as if they would. With this in mind, modern MPC protocols can be of the following shape: first create a small amount of random correlations, transform it into a large amount of OTs, and use this large amount of OTs in a very fast online phase.

This transformation is achieved using Pseudorandom Correlation Generators (PCG) and their counterparts, Pseudorandom Correlation Functions (PCF). This manuscript does not focus solely on OT but also on other *correlations* that can be held by the players, which are values that satisfy specific relationships. Various types of correlations will be studied throughout the manuscript. PCGs and PCFs rely on two key components:

— *Distributed Point Function* (DPF): Introduced by Boyle, Gilboa and Ishai in 2015 [BGI15], a DPF enables two parties to obtain pseudorandom functions f_0, f_1 such that their sum is a point function, a function nonzero at exactly one point. This primitive enables parties to efficiently share between Alice and Bob a large unit vector [3], that is, giving a pseudorandom vector $\mathbf{u_0}$ to Alice, and a pseudorandom vector $\mathbf{u_1}$ to Bob such that $\mathbf{u_0} \oplus \mathbf{u_1}$ is equal a unit vector.

— *Syndrome Decoding assumption* (SD): A classical assumption in coding theory, it captures the difficulty of distinguishing between $(\mathbf{H}, \mathbf{s})$ and $(\mathbf{H}, \mathbf{He})$ for $\mathbf{H}$ a random matrix, $\mathbf{s}$ a random vector and $\mathbf{e}$ a random sparse vector, that is a vector with few nonzero entries.

We illustrate how DPF and SD can combine together via an example, in the case of the OT correlation. Say that our goal is to produce n different OTs. Alice will hold therefore $(x_0^i, x_1^i)_{0\leqslant i\leqslant n-1}$ and Bob $(b^i, y^i := x_{b^i}^i)_{0\leqslant i\leqslant n-1}$. We define the vectors $\mathbf{x_\sigma} = (x_\sigma^i)_{0\leqslant i\leqslant n-1}$ for $\sigma \in \{0,1\}$, $\mathbf{b} = (b^i)_{0\leqslant i\leqslant n-1}$ and $\mathbf{y} = (y^i)_{0\leqslant i\leqslant n-1}$. Define $\mathbf{x_2} = \mathbf{x_1} - \mathbf{x_0}$. Then we have that $\mathbf{b} \odot \mathbf{x_2} = \mathbf{y} + \mathbf{x_0}$, where $\odot$ denote the component-wise product. We show how to parties can construct such vectors. Let say we construct

$$\mathbf{b} = \mathbf{He_0}$$

and

$$\mathbf{x_2} = \mathbf{He_1},$$

where $\mathbf{e_0}, \mathbf{e_1}$ are sparse vectors, and $\mathbf{H}$ is a parity-check matrix known by both Alice and Bob. Alice holds $\mathbf{He_0}$ and Bob holds $\mathbf{He_1}$. Syndrome Decoding ensures that $\mathbf{b}$ and $\mathbf{x_2}$ are pseudorandom. Now, remark that $\mathbf{b} \odot \mathbf{x_2} = \mathbf{He_0} \odot \mathbf{He_1}$. Consider one entry of $\mathbf{b} \odot \mathbf{x_2}$. One can check that it is equal to $\langle \mathbf{h} \otimes \mathbf{h}, \mathbf{e_0} \otimes \mathbf{e_1} \rangle$ where $\mathbf{h}$ is the row of $\mathbf{H}$ corresponding to that entry and $\otimes$ denotes the tensor product. Remark that because $\mathbf{e_0}$ and $\mathbf{e_1}$ are sparse their tensor product remains sparse. Viewing a sparse vector as a sum of unit vector, and observing that each unit vector is the truth table of a point

2. Pseudorandomness means that no efficient algorithm can distinguish the string from a truly random string. This notion will be formally defined later in the manuscript.

3. If the large unit vector is of size n, the shares can be compressed with only $O(\log(n))$ bits

function (that the players can get shares of using a DPF), yields a simple method to distribute shares of sparse vectors with a communication that grows only with the numeber of nonzero entries. Let be $\mathbf{u_0}$ and $\mathbf{u_1}$ the different shares. Then Alice defines $\mathbf{x_0}$ to be equal to $\mathbf{Ku_0}$ where $\mathbf{K}$ is the matrix whose rows are the $\mathbf{h} \otimes \mathbf{h}$. Alice set her value of $\mathbf{x_1} = \mathbf{x_2} + \mathbf{x_0}$. Similarly, Bob sets $\mathbf{y}$ to be equal to $\mathbf{Ku_1}$. Thanks to the property of DPF, $\mathbf{u_0}$ and $\mathbf{u_1}$ appear to be pseudorandom, and this property therefore hold for $\mathbf{x_0}$ and $\mathbf{x_1}$ and $\mathbf{y}$. We managed successfully produce numerous OT.

The code-based PCG approach involves multiple matrix multiplications, which can be costly due to the large matrix sizes: their size is approximately the number of OTs we have to produce, which can easily be in the billions. Strategies are needed to minimize computational costs. To improve efficiency, one natural consideration is to explore alternative matrix structures for $\mathbf{H}$. In the original Syndrome Decoding assumption, $\mathbf{H}$ is assumed to be a purely random matrix. However, introducing structured matrices could potentially accelerate the protocol. Care must be taken, however, to ensure that any structural modification to $\mathbf{H}$ does not compromise the underlying security assumptions. This was the starting point of our thesis: to build a bridge between MPC and coding theory, two communities that do not know each other very well. The contributions we present in the next section are therefore always based on a particular interpretation of the Syndrome Decoding assumption, for which a security analysis is meticulously conducted.

1.2 Our Contributions

Syndrome Decoding is an old assumption and the number of attacks is extensive. How can we ensure that a particular adversary will not compromise our system? It was cleverly observed in [BCGI+20a] that the majority of attacks on the Syndrome Decoding assumption are essentially performing linear computations on the syndrome $\mathbf{s}$, computations which further depend solely on $\mathbf{H}$. This implies that such attacks can be described by the existence of an attack vector, which, when doing the scalar product with $\mathbf{s}$ may be biased in scenario 2 ($(\mathbf{H}, \mathbf{s} = \mathbf{He})$, with $\mathbf{e}$ sparse) but not in scenario 1 ($(\mathbf{H}, \mathbf{s})$, with $\mathbf{s}$ random). This observation motivates the introduction of the *linear test framework*. The framework operates as follows: we provide the matrix $\mathbf{H}$ used in our Syndrome Decoding assumption to the adversary before sampling the noise vector $\mathbf{e}$. The adversary then has unlimited time to identify the optimal attack vector $\mathbf{v}$ such that $\mathbf{v}^\intercal\mathbf{He}$ offers a significant bias. This model serves as a lower bound for all possible linear attacks. If we can demonstrate that a matrix is secure within the linear test framework, we effectively prove its security against all attacks in this category. This is significant because it includes the best-known algorithm against Syndrome Decoding, namely Information Set Decoding (ISD) algorithm. In this thesis, we present a unified treatment of this framework. We demonstrate how ISD fits into this model and discuss the framework, highlighting its advantages and limitations.

1.2.1 A PCG from the Quasi-Abelian Ring Syndrome Decoding Assumption

Our first contribution in this thesis builds upon the work of [BCGI+20b], in which they developed a PCG using Ring Syndrome Decoding assumption. This special case of the syndrome decoding assumption is standard, capturing the difficulty of distinguishing (a, y) from $(a, a \cdot e_1 + e_2)$ where a, y are polynomials over $\mathbb{F}_q[X]$, and e_1, e_2 are sparse polynomials—those with few non-zero coefficients in $\mathbb{F}_q[X]$. Essentially, this generalizes the Syndrome Decoding assumption to the context of polynomial rings. The original approach by Boyle et al. aimed to create a single correlation over a ring $\mathcal{R} = \mathbb{F}_q[X]/(P(X))$, with $P(X)$ being a polynomial that splits completely into linear

factors. Under this condition, invoking the Chinese Remainder Theorem establishes an isomorphism between $\mathcal{R}$ and $\mathbb{F}_q^n$. Applying this isomorphism on our single correlation enables the derivation of multiple correlations. In [BCCD23], we revisit this construction and introduce a new assumption, the *Quasi Abelian Syndrome Decoding assumption*, tailored to our specific requirements. Our goal was to overcome a significant constraint in Boyle et al.'s initial approach: the construction could be achieved only over extremely large fields (larger than the number of pseudorandom correlations obtained at the end, which is large by assumption). We observe that using a multivariate approach eliminates this issue. Specifically, by taking $\mathcal{R} = \mathbb{F}_q[X_1, \ldots, X_n]/(X_1^{q-1} - 1, \ldots, X_n^{q-1} - 1)$, the limitation is reduced to $q \geqslant 3$, which is much more manageable. A new analysis of the security of Ring Syndrome Decoding in the multivariate case is therefore necessary. To achieve this, we use two specific tools: the linear attack framework and Quasi-group codes. They are special class of codes built over the set of all formal sums of elements of the group. Analyzing the security of the construction can be reduced to studying the underlying Syndrome Decoding assumption in the context of Quasi Abelian Syndrome Decoding (QA-SD). We performed this analysis and found that standard results state that $\mathcal{R} = \mathbb{F}_q[X_1, \ldots, X_n]/(X_1^{q-1} - 1, \ldots, X_n^{q-1} - 1) \simeq \mathbb{F}_q^{(q-1)^n}$ for an appropriate choice of group $\mathbb{G}$. To show that the construction is secure, we attempt to show it is secure inside of the linear attack framework. We used a standard technique to relate resistance against linear attacks to some property of the underlying code we consider, in this case the minimal distance of the dual code, and we analyze it.

In a follow-up work [BBCC+24], we push forward different aspects that were overlooked or deferred in [BCCD23]. First, we developed a method to obtain correlations over $\mathbb{F}_2$, rather than $\mathbb{F}_3$. The approach involves generating correlations over $\mathbb{F}_4$ and then converting them to $\mathbb{F}_2$. This method introduces some computational and communication overhead, but it is necessary in order to circumvent the limitation on the field size inherent to the QA-SD construction ($q \geqslant 3$). We therefore focused particularly on the case of $\mathbb{F}_4$ in order to obtain correlations over $\mathbb{F}_2$ in the end. We decided to develop a fully optimized version of the algorithm, incorporating all aspects of the previous construction but tailored specifically to the $\mathbb{F}_4$ case. Additionally, we conducted a new thorough security analysis of the scheme. This analysis uncovered that some of our original parameter selections were too optimistic, and identified new optimized attacks against the quasi-abelian syndrome decoding assumption.

1.2.2 A PCF from the Variable Density Syndrome Decoding Assumption

We also analyze the construction of a Pseudorandom Correlation Function (PCF). A PCF is a more complex cryptographic primitive than a PCG and aims to resolve the main issue that PCGs face: it generates all pseudorandom correlated randomness simultaneously. Instead, PCFs generate the correlations on-the-fly. The players perform a single short protocol at the beginning to obtain some *correlated keys*. Using these keys, they can construct a function indexed by the keys, f_{k_0}, f_{k_1}. The players can agree on the inputs to call the function on and can precisely generate the required amount of correlated randomness without having to redo the protocol when the number of correlations runs out. The advantage of PCFs is that the parties do not need to store a long chain of pseudorandom correlations, as they only need to evaluate their function each time they want a new correlation. We will detail our work [CD23], which is a refinement of [BCGI+20a]. The latter introduced a new assumption, the *Variable Density Learning Parity with Noise* (LPN) assumption, which we will call *Variable Density Syndrome Decoding* in this manuscript, as we focus more on the coding theory perspective.

To prove the construction secure, [BCGI+20a] provides a proof that it resists all linear tests through a precise analysis of the bias. In [CD23], we produce a correction of their analysis, which we found to contain some errors, and propose a slightly modified version of the construction (and thus of the assumption), enhanced with a better analysis and some precise parameter optimizations to make the scheme usable in practice. To be more concrete, the VDSD assumption is a particular variant of SD where both the error vector $\mathbf{e}$ and the matrix $\mathbf{H}$ follow a specific structure: they are composed of different blocks, each block being larger than the previous one but with the same number of non-zero coordinates per block, resulting in the density of each block decreasing (exponentially in the original construction). This particular construction allows the computation of matrix-vector products in linear time while ensuring that the output is not a sparse vector and, at the same time, having an exponentially large matrix compared to its description. We retain the same construction but replace the first few blocks with purely random blocks. With the different optimizations listed above, we were able to reduce the security parameters by approximately four orders of magnitude in [CD23].

1.3 Organization of the Manuscript

Chapter 2 introduces general background on coding theory and the syndrome decoding assumption, as well as general MPC techniques that interest us, such as the GMW technique. The objects that we precisely study in this manuscript, PCG and PCF, are formally introduced and discussed in Chapter 3. Chapter 4 focuses on the linear attacks framework, covering the standard definition and showing its pros and cons. Chapter 5 and Chapter 6 cover the work we did on PCGs ([BCCD23; BBCC+24]): the former covers the original QA-SD construction and the subsequent optimizations, and the latter covers the concrete security analysis of QA-SD. Finally, Chapter 7 presents our work from [CD23] and discusses the VDSD constructions and their improvements.

Chapter 2

Technical Background

We provide the essential background needed to comprehend the rest of the manuscript. Initially, we review key concepts in probability and major constructions in cryptography. Next, we focus on coding theory and the various hardness assumptions that will be referenced throughout the manuscript. Lastly, we offer a brief definition of secure multiparty computation and elaborate on the existing techniques in this domain.

Outline of the current chapter

2.1 Notations

Throughout the manuscript, $\mathbb{F}$ shall denote a finite field, and $\mathbb{F}_q$ is the finite field with q elements. $\mathbb{G}$ shall denote a finite abelian group and $\mathcal{R}$ a ring. $\mathbb{N}$ shall denote the set of natural numbers. We denote by $[a, b]$ the set of integers between a and b (included). $[n]$ denotes $[0, n-1]$. We use $A \simeq B$ to denote that two sets are isomorphic. Matrices and vectors will be denoted with bold letters, with matrices always capitalized. We assume that vectors are always considered as columns, and we denote their row counterparts with $\mathbf{v}^\intercal$. For a vector $\mathbf{v}$, we denote its i-th entry by $\mathbf{v}_i$. For two vectors $\mathbf{u} = (u_1, \ldots, u_t)^\intercal, \mathbf{v} = (v_1, \ldots, v_t)^\intercal \in \mathcal{R}^t$ for some ring $\mathcal{R}$, by $\mathbf{u} \otimes \mathbf{v}$ the tensor product, defined by $\mathbf{u} \otimes \mathbf{v} = (u_i \cdot v_j)_{i,j\leqslant t}^\intercal = (v_1 \cdot \mathbf{u}^\intercal, \ldots, v_t\mathbf{u}^\intercal)^\intercal$ and we denote by $\langle \mathbf{u}, \mathbf{v} \rangle = \mathbf{u}^\intercal \cdot \mathbf{v}$ their inner product. Similarly, we define $\mathbf{u} \boxplus \mathbf{v}$ to denote $\mathbf{u} \boxplus \mathbf{v} = (u_i + v_j)_{i,j\leqslant t}^\intercal$. Finally λ shall design a security parameter.

2.2 Probability Toolbox

In this section, we list different concentration bounds and probability results that will be useful in the thesis.

Definition 2.2.1 (Bias of a Distribution)**.** *Given a distribution $\mathcal{D}$ over $\mathbb{F}^n$ and a vector $\mathbf{v} \in \mathbb{F}^n$, the bias of $\mathcal{D}$ with respect to $\mathbf{v}$, denoted $\mathsf{bias}_{\mathbf{v}}(\mathcal{D})$, is equal to*

$$\mathsf{bias}_{\mathbf{v}}(\mathcal{D}) = |\mathbb{P}_{\mathbf{x}\sim\mathcal{D}}[\mathbf{v}^\intercal \cdot \mathbf{x} = 0] - \mathbb{P}_{\mathbf{x}\sim\mathcal{U}_n}[\mathbf{v}^\intercal \cdot \mathbf{x} = 0]| = \left|\mathbb{P}_{\mathbf{x}\sim\mathcal{D}}[\mathbf{v}^\intercal \cdot \mathbf{x} = 0] - \frac{1}{|\mathbb{F}|}\right|,$$

where $\mathcal{U}_n$ denotes the uniform distribution over $\mathbb{F}^n$. The bias of $\mathcal{D}$, denoted $\mathsf{bias}(\mathcal{D})$, is the maximum bias of $\mathcal{D}$ with respect to any nonzero vector $\mathbf{v}$.

2.2.1 Standard Probability Lemmas

Given t distributions $(\mathcal{D}_1, \ldots, \mathcal{D}_t)$ over $\mathbb{F}_2^n$, we denote by $\bigoplus_{i\leqslant t} \mathcal{D}_i$ the distribution obtained by *independently* sampling $\mathbf{v_i} \xleftarrow{\$} \mathcal{D}_i$ for $i = 1$ to t and outputting $\mathbf{v} \leftarrow \mathbf{v_1} \oplus \cdots \oplus \mathbf{v_t}$. We will use the following bias of the exclusive-or (cf. [Shp09]).

Lemma 2.2.1. *Let $t \in \mathbb{N}$ be an integer, and let $(\mathcal{D}_1, \ldots, \mathcal{D}_t)$ be t independent distributions over $\mathbb{F}_2^n$. Then $\mathsf{bias}(\bigoplus_{i\leqslant t} \mathcal{D}_i) \leqslant 2^{t-1} \cdot \prod_{i=1}^t \mathsf{bias}(\mathcal{D}_i) \leqslant \min_{i\leqslant t} \mathsf{bias}(\mathcal{D}_i)$.*

Let $\mathrm{Ber}_r(\mathbb{F}_2)$ denote the Bernoulli distribution that outputs 1 with probability r, and 0 otherwise. More generally, we denote by $\mathrm{Ber}_r(\mathbb{F})$ the distribution that outputs a uniformly random nonzero element of $\mathbb{F}$ with probability r, and 0 otherwise. We will use a standard simple lemma for computing the bias of a XOR of Bernoulli samples:

Lemma 2.2.2 (Piling-up lemma)**.** *For any $0 < r < 1/2$ and any integer n, given n random variables $X_1, \ldots, X_n$ i.i.d. to $\mathrm{Ber}_r(\mathbb{F}_2)$, it holds that $\Pr[\bigoplus_{i=1}^n X_i = 0] = 1/2 + (1-2r)^n/2$.*

We will also need two concentration bounds. The bounded difference inequality [McD89] is an application of the more general Azuma inequality [Azu67]. Let $(n, m) \in \mathbb{N}^2$ be two integers. We say that a function $\Phi : [n]^m \mapsto R$ satisfies the *Lipschitz property with constant d* if for every $\mathbf{x}, \mathbf{x}' \in [n]^m$ which differ in a single coordinate, it holds that $|\Phi(\mathbf{x}) - \Phi(\mathbf{x}')| \leqslant d$.

Lemma 2.2.3 (Bounded Difference Inequality or McDiarmid inequality). *Let $\Phi : [n]^m \mapsto R$ be a function satisfying the Lipschitz property with constant d, and let $(X_1, \ldots, X_m)$ be independent random variables over $[n]$. Then*

$$\Pr[\Phi(X_1, \ldots, X_m) < \mathbb{E}[\Phi(X_1, \ldots, X_m)] - t] \leqslant \exp\left(-\frac{2t^2}{m \cdot d^2}\right).$$

Eventually, we will rely on the Occupancy Bound from [KMPS94], which provides tight bounds for the balls and bins problem.

Lemma 2.2.4 (Occupancy Bound). *Let E be the number of empty bins when m balls are placed randomly into n bins, and define $r = m/n$. The expectation of E is given by $\mu = \mathbb{E}[E] = (1 - \frac{1}{n})^m \approx ne^{-r}$. For any $\theta > 0$,*

$$\Pr[|E - \mu| \geqslant \theta\mu] \leqslant 2\exp\left(-\frac{\theta^2\mu^2(n - \frac{1}{2})}{n^2 - \mu^2}\right) = \mathcal{B}$$

Remark 2.2.1. Note that we can derive the following two equations : $\Pr[E \geqslant \mu(\theta + 1)] < \mathcal{B}$ and $\Pr[E \leqslant \mu(1 - \theta)] < \mathcal{B}$

2.3 Cryptographic Building Blocks

We present in this section some well-known cryptographic models, constructions, and primitives that will be used or serve as a basis in this manuscript.

2.3.1 Hardness in Cryptography

Cryptography has been constructed around the notion of *hardness of a problem.* Hard problems are problems for which no efficient algorithms [1] can find a solution with high probability. Broadly speaking, a cryptographic construction is said to be secure if breaking the construction can be reduced to solving a hard problem. This hardness is encapsulated in basic cryptographic primitives with which more involved protocols are built. This is the case for the very fundamental *one-way functions.*

Definition 2.3.1 (One-way functions). *A function $f : \{0,1\}^\lambda \to \{0,1\}^{n(\lambda)}$ is said to be* one-way *when*

- *The computation of $f(x)$, given the knowledge of f and x is computable in polynomial time in the security parameter λ.*
- *For all probabilistic polynomial time adversary $\mathcal{A}$*

$$\Pr_{x \xleftarrow{\$} \{0,1\}^\lambda,\ y = f(x)}\left[\, f(\mathcal{A}(y, 1^\lambda)) = y \,\right] \leqslant \mathsf{negl}(\lambda)$$

with $\mathsf{negl}(\lambda)$ a negligible function.

In short, one-way functions are functions that are easy to evaluate, but very hard to invert. But how to build one-way functions? As said previously, it relies on the assumption that hard

1. That we can execute in a polynomial time in size of the input

problems exist. For example, the *discrete logarithm* states that for some groups $\mathbb{G}(\cdot, \epsilon)$, it is hard to recover k from the knowledge of $(\alpha, \alpha^k \coloneqq \underbrace{\alpha \cdot \alpha \cdot \cdots \cdot \alpha}_{k \text{ times}})$, where α is a primitive element of the group. Therefore, $f(x) = \alpha^x$ defines a one-way function: computing the result is easy because it consists in doing $\log(k)$ multiplications using exponentiation by squaring, but coming back is hard. One-way functions imply many concepts in cryptography, among which are pseudorandom generators and pseudorandom functions (see Section 2.3.3), commitments, and digital signature schemes.

2.3.2 Indistinguishability

A classical tool in cryptography that will be used in this thesis is the notion of *indistinguishability*. It is a way to precisely formalize the limitations of the adversary or players.

Definition 2.3.2. *There are three distinct notions of indistinguishability.*

— *Two distributions* $(\mathcal{D}_n^0, \mathcal{D}_n^1)$ *over* $\{0,1\}^n$ *are said to be* perfectly *indistinguishable if*

$$\forall \alpha \in \{0,1\}^n, \Pr_{x \xleftarrow{\$} \mathcal{D}_n^0}[\, x = \alpha \,] = \Pr_{x \xleftarrow{\$} \mathcal{D}_n^1}[\, x = \alpha \,],$$

for every $n \in \mathbb{N}$.

— *Two distributions* $(\mathcal{D}_n^0, \mathcal{D}_n^1)$ *over* $\{0,1\}^n$ *are said to be* statistically *indistinguishable if the statistical distance between* $\mathcal{D}_n^0$ *and* $\mathcal{D}_n^1$ *is bounded by a negligible function of* n.

$$\frac{1}{2} \sum_{\alpha \in \{0,1\}^n} \left| \Pr_{x \xleftarrow{\$} \mathcal{D}_n^0}[\, x = \alpha \,] - \Pr_{x \xleftarrow{\$} \mathcal{D}_n^1}[\, x = \alpha \,] \right| \leqslant \mathsf{negl}(\mathsf{n}).$$

— *Two distributions* $(\mathcal{D}_n^0, \mathcal{D}_n^1)$ *over* $\{0,1\}^n$ *are said to be* computationally *indistinguishable if for every probabilistic polynomial-time adversary* $\mathcal{A}$*, for all* n *large enough, it holds that*

$$\left| \Pr_{x \xleftarrow{\$} \mathcal{D}_n^0}[\, \mathcal{A}(x) = 0 \,] - \Pr_{x \xleftarrow{\$} \mathcal{D}_n^1}[\, \mathcal{A}(x) = 0 \,] \right| \leqslant \mathsf{negl}(\mathsf{n}).$$

Remark 2.3.1. In the manuscript, when not specified otherwise, we will by default consider that indistinguishable means computationally indistinguishable. We write $\mathcal{D}_n^0 \approx \mathcal{D}_n^1$ to indicate that two distributions are computationally indistinguishable, and $\mathcal{D}_n^0 \approx_s \mathcal{D}_n^1$ to indicate that two distributions are statistically indistinguishable. Moreover, we say we have information-theoretic security when even an adversary with unlimited computing resources and time is not able to distinguish between two distributions.

2.3.3 Pseudorandom Generators and Pseudorandom Functions

Randomness is predominant in cryptography. Since its use in the fundamental *one-time pad*, most cryptographic applications require random numbers to generate secret keys, nonces, or salts. This raises a very important question: *how do we find significant amounts of randomness?* This can be achieved by using True Random Generators (TRGs), which are systems that output the result of a physical experiment considered to be random (unpredictable). A simple example is the roll of a dice, which is considered to be unpredictable and thus serves as a source of randomness in many board games. More involved and practical sources can be found, such as measuring the electronic

noise inside an electrical conductor. Nevertheless, the use of TRGs has important limitations. First, one has to be careful that the result is truly unbiased and unpredictable. This kind of bias can be balanced after sampling, but this post-processing would reduce the number of useful bits as well as the efficiency of the generator. In addition, TRGs are often too slow and expensive to implement. This is why *Pseudorandom Generators* (PRG) were designed, as a method to produce a large amount of *pseudorandomness* from a small amount of randomness - that is, something that will act as a large amount of random elements even if it is not truly random. This does not mean that TRGs are useless, as they produce the small amount of *real* randomness required by the PRG.

Definition 2.3.3 (Pseudorandom Generators)**.** *A* Pseudorandom Generator (PRG) *is a polynomial-time algorithm $G : \{0,1\}^\lambda \to \{0,1\}^n$ - with $n \gg \lambda$, such that for any polynomial-time distinguisher $\mathcal{A}$, it holds that*

$$\left| \Pr_{x \xleftarrow{\$} \{0,1\}^\lambda,\, y \leftarrow G(x)} \left[\mathcal{A}(y, 1^\lambda) = 0 \right] - \Pr_{y \xleftarrow{\$} \{0,1\}^n} \left[\mathcal{A}(y, 1^\lambda) = 0 \right] \right| \leqslant \mathsf{negl}(\lambda).$$

Therefore, given a PRG, no polynomial time adversary can distinguish a purely random number from a number outputted by the PRG. PRGs can be constructed from believed true cryptographic assumptions. One of the more important result states that the existence of PRGs is in fact equivalent (in both directions) to the existence of one-way functions [BM82; HILL99].

Remark 2.3.2. The remarkable statement about PRGs is that the entropy stays low: it is not a method to generate entropy since we compute y deterministically from a short input x. Nevertheless, it is hard to distinguish the result of the PRG from a random element of the same size.

Example 2.3.1 (A usage of PRG)**.** The classical *one-time pad* is a protocol that allows two parties, Alice and Bob to exchange messages with perfect security (information theoretic). Here is how it goes: Alice and Bob have already agreed on a key $k \in \mathbb{F}_q^n$. They can send any message $m \in \mathbb{F}_q^n$ to one another by sending $m \oplus k$. Because the key is secret and chosen randomly, the security is perfect. The problem with the one-time pad protocol is that parties have to agree beforehand on a key k, which size is the same as the size of the message they wish to send. This is surely problematic. Therefore, if one-time pad offers perfect security, it is not usable in practice. Nevertheless, if we suppose that parties know a common secret key $k_0 \in \mathbb{F}_q^\lambda$ and a PRG $G : \mathbb{F}_q^\lambda \to \mathbb{F}_q^n$, then parties can send each other a message using the PRG: the encrypted message is then $c = G(k_0) + m$. If the PRG is secure, then c cannot be decrypted without the key. Note that in this example we still assume that the parties both know the same key k_0, but in this case the size of k_0 is $\lambda \ll n$, and therefore, it is easier for the parties to share such a key.

One of the downsides of the PRG is that one needs to generate all the randomness in one block, which can be more or less cumbersome. This raises the question of whether it would be possible to generate *on-the-fly* pseudorandom elements. It is equivalent to asking whether a pseudorandom function is possible to be created, that is a function that mimics a real random function.

Definition 2.3.4 (Pseudorandom function)**.** *Let $k \in \{0,1\}^\lambda$, with λ as the security parameter.*

$$f_k : \mathbb{F}_2^{a(\lambda)} \to \mathbb{F}_2^{b(\lambda)}$$

defines a family of functions called pseudorandom functions (PRF) *if the following holds:*

— *Given k and x, $f_k(x)$ is computable in polynomial time.*

— *For any probabilistic polynomial-time adversary $\mathcal{A}$ and given $(x_i)_{0\leqslant i\leqslant n(\lambda)-1}$*

$$\left| \Pr_{k \xleftarrow{\$} \{0,1\}^\lambda} \left[\mathcal{A}((x_i, f_k(x_i))_{0\leqslant i\leqslant n(\lambda)-1}) = 0 \right] - \Pr_{R \xleftarrow{\$} \mathfrak{R}(a,b)} \left[\mathcal{A}((x_i, R(x_i))_{0\leqslant i\leqslant n(\lambda)-1}) = 0 \right] \right| \leqslant \mathsf{negl}(\lambda).$$

where $\mathfrak{R}(a,b)$ denotes the set of functions $\mathbb{F}_2^a \to \mathbb{F}_2^b$.

Remark 2.3.3. It means that it is difficult to distinguish, in polynomial time, a function chosen from this family from a purely random function by looking at $n(\lambda)$ evaluations of the function. In the standard definition, the x_i (values on which the evaluations are made) can be adaptively chosen by the adversary.

A *Weak Pseudorandom function* (WPRF) is a PRF where the x_i's are not chosen by the adversary but sampled uniformly at random from $\mathbb{F}_2^a$. It is therefore *weak* because it restricts the power of the adversary.

Example 2.3.2 (Usage of PRF)**.** Coming back to our example using one-time pad as before, the notion of PRF offers us more flexibility. More exactly, the parties agree on a value x, and then set the encryption of each bit b_i of the message they want to send to be $f_k(x+i) + b_i$. That is, they can send bits of messages, but not everything just at once.

2.3.3.1 Constructing PRF from PRG: The GGM Construction

We explain the so-called GGM tree construction to create a pseudorandom function. It was named after Goldreich, Goldwasser and Micali for their original construction of PRF in [GGM84]. The protocol uses a length doubling PRG $G : \mathbb{F}_q^\lambda \to \mathbb{F}_q^{2\lambda}$, where λ is a security parameter. Starting from a seed $\mathbf{s} \in \mathbb{F}_q^\lambda$, one constructs the GGM tree as follows:

— Label the root of the tree with $\mathbf{s}$.

— Let x be an internal node, with label $\mathbf{s}(x)$. Extract the two values $\mathbf{s}_0(x), \mathbf{s}_1(x)$ of length λ, using the length-doubling PRG, with $G(\mathbf{s}(x)) = (G_0(\mathbf{s}(x))||G_1(\mathbf{s}(x))) = (\mathbf{s}_0(x)||\mathbf{s}_1(x))$, where $||$ denote the concatenation. Here G_0 and G_1 denote respectively the first half and the second half of G's output . Then, x has two children x_0 and x_1 with respective labels $\mathbf{s}_0(x)$ and $\mathbf{s}_1(x)$.

For an evaluation domain of size 2^m, we have therefore to create a complete binary tree with m levels, and 2^m leaves that we can number canonically in binary. Therefore, for each value of $x \in [2^m]$, there exists a leaf numbered by x. The tree defines a pseudorandom function. Why is that? On entry $x \in [2^m]$, we define the pseudorandom function $f_k(x)$ to be equal to the label of the leaf associated with x, for the tree constructed from seed $s = k$. Note that there is no need to compute all the tree to obtain the $f_k(x)$: one can just compute all the intermediate internal nodes from the root to the concerned leaf. These nodes form the *evaluation path* of leaf x. Therefore, for a leaf x, $\mathsf{Eval}(\mathbf{s}, x)$ compute the labels of all the nodes on its evaluation path until the x, using the seed and the PRG G. This is represented in Figure 2.1.

2.4 Coding Theory

As stated in the introduction, coding theory plays a crucial role in the constructions presented in this manuscript. The following overview will not be exhaustive by any means, but it intends to

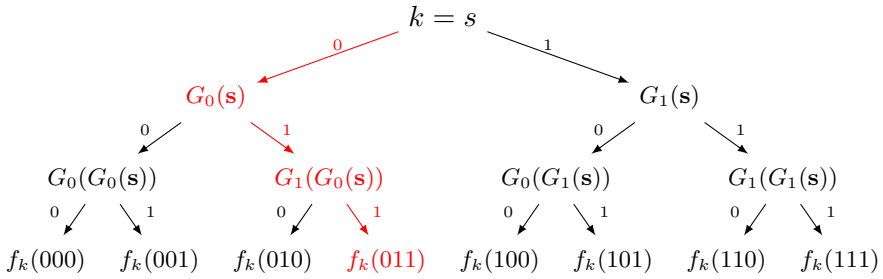

Figure 2.1 – The GGM construction. The red-path is the evaluation path corresponding to the output $f_k(x) = f_k(3) = f_k(011_2)$.

provide all the different notions required to understand the constructions and their analysis.

2.4.1 What Is a Code?

Coding theory is one of the oldest topics in computer science. It was first extensively studied as a way to correct errors that may occur in a noisy channel. Such a channel alters messages, making them noisy, meaning some bits of the message could be flipped or erased. A natural idea to counter these problems is to introduce redundancy: if the same message is sent multiple times, it will be easier to recover it even in the presence of noise. This gave birth to the notion of *error-correcting codes*, which consists in pairing each message with a *codeword*. The codeword can be seen as a longer message, obtained by adding redundancy. The codeword is then sent and may be damaged by the noise of the channel during transmission, but the hope is that it will be possible to recover the original message, using a properly designed *decoding algorithm*. Many different codes exist, but we focus exclusively on linear codes, which means that the pairing associating a message to its codeword is a linear map. Namely:

Definition 2.4.1. *Let $q > 1$ and n, k be positive integers with $n > k$. A linear code* $\mathcal{C}$ of length n is the vector subspace of $\mathbb{F}_q^n$ of dimension k defined as follows

$$\mathcal{C} = \left\{ \mathbf{Gm} \mid \mathbf{m} \in \mathbb{F}_q^k \right\},$$

where $\mathbf{G} \in \mathbb{F}_q^{n \times k}$ of rank k represents a linear map, and is called a *generator matrix* of $\mathcal{C}$. It is therefore a subspace of $\mathbb{F}_q^n$ of dimension k. We say that $\mathcal{C}$ is a $[n, k]_q$-code. The integer n is called the *length of the code* $\mathcal{C}$ and k the *dimension of the code*. The *rate of the code* $\mathcal{C}$ is defined to be $R = k/n$. The elements of $\mathcal{C}$ are called *codewords*.

Remark 2.4.1. Every $\mathbf{G} \in \mathbb{F}_q^{n \times k}$ describes a code, but a generator matrix of a code is not unique. In fact, given a non-singular matrix $\mathbf{U} \in \mathbb{F}_q^{k \times k}$, a standard result states that $\mathbf{G}$ and $\mathbf{GU}$ both produce the same code.

A generator matrix $\mathbf{G}$ is said to be in *systematic form* when

$$\mathbf{G} = \begin{bmatrix} \mathbf{I}_{n-k} \\ \mathbf{G}' \end{bmatrix}.$$

Given $\mathbf{G}$ a generator matrix of rank k such that the first k rows are independent, one can obtain its systematic form by multiplying by the adequate non-singular matrix (representing the column

operations performed). The idea behind coding theory is fundamentally to introduce redundancy, and this redundancy is clearly visible when considering the description of a code by $\mathbf{G}$ in systematic form:

$$\mathcal{C} = \left\{ \mathbf{c} \mid \mathbf{c} = \mathbf{G}\mathbf{m}, \mathbf{m} \in \mathbb{F}_q^k \right\} = \left\{ \begin{bmatrix} \mathbf{m} \\ \mathbf{G}'\mathbf{m} \end{bmatrix} \mid \mathbf{m} \in \mathbb{F}_q^k \right\}.$$

$\mathbf{G}'\mathbf{m}$ corresponds to the redundancy added to every message.

Definition 2.4.2 (Hamming weight, Hamming distance and minimum distance)**.** *The* Hamming weight *of a vector $\mathbf{x} \in \mathbb{F}_q^n$ is defined as the number of non-zero entries in $\mathbf{x}$:*

$$wt(\mathbf{x}) \coloneqq \left| \left\{ i \mid \mathbf{x}_i \neq 0 \right\} \right|.$$

Similarly, Hamming distance *between two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{F}_q^n$ is defined as*

$$d_H \coloneqq wt(\mathbf{x} - \mathbf{y}).$$

Let $\mathcal{C}$ be a linear code of length n. The minimum distance *of $\mathcal{C}$ is defined as the smallest Hamming distance $d_{\mathcal{C}}$ between two distinct codewords of $\mathcal{C}$.*

$$d_{\mathcal{C}} \coloneqq \min_{\mathbf{x},\mathbf{y} \in \mathcal{C}, \mathbf{x} \neq \mathbf{y}} \{ d_H(\mathbf{x}, \mathbf{y}) \}$$

By linearity, it is also equal to the smallest Hamming weight among the vectors of $\mathcal{C} \backslash \{0\}$.

$$d_{\mathcal{C}} \coloneqq \min_{\mathbf{x} \in \mathcal{C}, \mathbf{x} \neq \mathbf{0}} wt(\mathbf{x}).$$

We call $\delta \coloneqq d/n$ the relative distance *of a code. If $\mathcal{C}$ is a $[n,k]_q$-code of minimum distance d, we classically say it is a $[n,k,d]_q$-code.*

Definition 2.4.3 (Parity-check matrix)**.** *Let $\mathcal{C}$ be a $[n,k,d]_q$ code. A* parity-check matrix *$\mathcal{C}$ is a matrix $\mathbf{H} \in \mathbb{F}_q^{n-k \times n}$ such that*

$$\mathcal{C} = \left\{ \mathbf{c} \in \mathbb{F}_q^n \mid \mathbf{H}\mathbf{c} = 0 \right\}.$$

In other words, this means that the code is defined as the right kernel of the matrix $\mathbf{H}$. For the same reasons as for a generator matrix, a parity-check matrix is not unique.

2.4.1.1 Some Important Bounds in Coding Theory

We present some important bounds imported from coding theory.

Proposition 2.4.1 (Singleton Bound)**.** *Let $\mathcal{C}$ be a $[n,k,d]_q$ code. Then we have*

$$k + d \leqslant n + 1.$$

This is asymptotically equivalent to

$$R + \delta \leqslant 1.$$

Definition 2.4.4 (entropy function)**.** *Let $q \geqslant 2$. The entropy function is defined as follows*

$$H_q(p) = \begin{cases} p \log_q(q-1) - p \log_q(p) - (1-p) \log_q(1-p) & \text{if} \quad 0 \leqslant p \leqslant 1; \\ 0 & \text{if} \quad p = 0, p = 1. \end{cases}$$

Proposition 2.4.2 (Gilbert-Varshamov bound theorem). *Let $q \geqslant 2$. For every $0 \geqslant \delta < 1 - \frac{1}{q}$ and $0 < \epsilon \leqslant 1 - H_q(\delta)$, there exists a code with rate $R \geqslant 1 - H_q(\delta) - \epsilon$ and relative distance δ. Moreover, for (δ, ϵ) as defined above, a random code $\mathcal{C}$ of dimension $k \geqslant (1 - H_q(\delta) - \epsilon)n$ satisfies:*

$$\Pr(d(\mathcal{C}) > \delta n) \geqslant 1 - q^{-\epsilon n}.$$

This means that, with high probability, the parameters of random codes are close to the Gilbert-Varshamov bound.

2.4.2 The Syndrome Decoding Problem

As previously stated, the whole idea behind coding theory is to be able to correct errors produced by the noise in the channel. In this thesis, the noise will be modeled by *sparse vectors*.

Definition 2.4.5 (t-sparse vectors). *A vector $\mathbf{v} \in \mathbb{F}_q^n$ is said to be a* t-sparse vector *if it has a Hamming weight inferior or equal to $t \ll n$. We denote by $\mathcal{S}(\mathbb{F}_q^n, t)$ the set of all t-sparse vectors over $\mathbb{F}_q^n$. We say that we have a notion of* sparsity *over $\mathbb{F}_q$.*

The noise can also be modelled via a Gaussian distribution but in this thesis, we will work only with an error defined uniquely by its small Hamming weight. Given a corrupted codeword $\mathbf{c} + \mathbf{e}$, where $\mathbf{e}$ is a t-sparse vector, it should be possible to recover the original message associated with $\mathbf{c}$, assuming that t is not too large. Here are some insights on why coding theory offers correction capability. Consider an $[n, k, d]_q$ code. Whenever $t < d/2$, there exists a decoding method that, although simple to explain, is challenging to implement in practice: given a noisy codeword $\mathbf{c'} = \mathbf{c}+\mathbf{e}$, return the message $\mathbf{m}$ corresponding to the closest codeword to $\mathbf{c'}$. Using this technique there is no ambiguity, and we can recover the codeword. This illustrates the importance of the minimum distance in a code and serves as proof of feasibility.

However, practical decoding is not straightforward. The previously mentioned method shifts the difficulty of error correction to finding the closest codeword. But finding the closest codeword is computationally demanding due to the vast search space. Employing a brute-force approach would require up to $(q-1)^t\binom{n}{t}$ operations, which is impractical for large values of t. Fortunately, more efficient strategies are available. One standard technique is *syndrome decoding*. This approach involves computing the *syndrome*, defined as $\mathbf{s} = \mathbf{Hc'}$, where $\mathbf{H}$ is the parity-check matrix of the code. Notably, $\mathbf{Hc'} = \mathbf{H}(\mathbf{Gm}+\mathbf{e}) = \mathbf{He}$, implying that recovering the noise vector $\mathbf{e}$ would resolve the decoding problem. Hence, the task becomes finding a sparse solution to the equation $\mathbf{s} = \mathbf{He}$. Although techniques for identifying this sparse solution are more efficient than brute-force methods, no polynomial-time algorithms in t are currently known. Some of the most effective techniques will be examined in Appendix B.

In the late 70s', during the early age of modern cryptography, researchers were actively seeking hard problems that could be used as a foundation for encryption schemes. Naturally, they decided to examine decoding, which led to the formulation of an important assumption.

Definition 2.4.6 ((Search) Syndrome Decoding assumption). *Let $(n, k, t) = (n(\lambda), k(\lambda), t(\lambda))$ be parameters polynomial in the security parameter λ. The* Search Syndrome Decoding *problem is defined as follows: Given $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$ a parity-check matrix, and $\mathbf{s} \in \mathbb{F}_q^{n-k}$ as a syndrome, the goal is to recover (if it exists) an error $\mathbf{e} \in \mathcal{S}(\mathbb{F}_q^n, t)$ such that $\mathbf{He} = \mathbf{s}$.*

The search Syndrome Decoding *assumption states that the syndrome decoding problem is hard.*

That is, for every probabilistic polynomial-time algorithm $\mathcal{A}$, it holds that:

$$\Pr_{\mathbf{H} \xleftarrow{\$} \mathbb{F}_q^{(n-k)\times n},\, \mathbf{e} \xleftarrow{\$} \mathcal{S}(\mathbb{F}_q^n, t)} [\, \mathcal{A}(\mathbf{H}, \mathbf{s} = \mathbf{He}) = \mathbf{e} \,] \leqslant \mathsf{negl}(\lambda),$$

where negl *denotes a negligible function.*

This assumption can also be stated in a decisional variant, which will be the one used and studied in this thesis.

Definition 2.4.7 ((Decisional) Syndrome Decoding assumption)**.** *Let $(n, k, t) = (n(\lambda), k(\lambda), t(\lambda)$ be parameters polynomial in the security parameter λ. The goal of the* decisional syndrome decoding problem *is to distinguish, with a non-negligible advantage, between the distributions*

$$\begin{array}{lll} \mathcal{D}_0 : & (\mathbf{H}, \mathbf{s}) & \text{where } \mathbf{H} \xleftarrow{\$} \mathbb{F}_q^{(n-k)\times n}, \mathbf{s} \xleftarrow{\$} \mathbb{F}_q^n \qquad \text{and} \\ \mathcal{D}_1 : & (\mathbf{H}, \mathbf{H}\cdot\mathbf{e}) & \text{where } \mathbf{H} \xleftarrow{\$} \mathbb{F}_q^{(n-k)\times n}, \mathbf{e} \xleftarrow{\$} \mathcal{S}(\mathbb{F}_q^n, t). \end{array}$$

Remark 2.4.2. The *decisional syndrome decoding* assumption, that we will shorten in syndrome decoding, and denote by $(n, k, t)_q - \mathsf{SD}$ or SD when the context is clear, states that this problem is hard. That is, for every probabilistic polynomial-time algorithm $\mathcal{A}$, it holds that

$$\left| \Pr_{\mathbf{H} \xleftarrow{\$} \mathbb{F}_q^{(n-k)\times n},\, \mathbf{e} \xleftarrow{\$} \mathcal{S}(\mathbb{F}_q^n, t)} [\, \mathcal{A}(\mathbf{H}, \mathbf{He}) = 0 \,] - \Pr_{\mathbf{H} \xleftarrow{\$} \mathbb{F}_q^{(n-k)\times n},\, \mathbf{s} \xleftarrow{\$} \mathbb{F}_q^{n-k}} [\, \mathcal{A}(\mathbf{H}, \mathbf{s}) = 0 \,] \right| \leqslant \mathsf{negl}(\lambda),$$

where negl denotes a negligible function.

The two assumptions are, in fact, equivalent: there exists search-to-decision reductions (starting with [FS96]).

Remark 2.4.3 (Distribution of the sparse vectors)**.** Unless stated otherwise, we assume that we always sample sparse vectors uniformly at random in $\mathcal{S}(\mathbb{F}_q^n, t)$. Another distribution that we will consider is the *regular* distribution, which consists of sampling a vector of weight t, the t-non zero coordinates being equally distributed in blocks of size n/t.

Remark 2.4.4 (Link with Learning Parity with Noise assumption (LPN))**.** In the MPC community, the first attempts to solve noisy equations and their associated hardness were mostly done using the *Learning Parity with Noise* assumption ([BCGI18; BCGI+19b]). Imagine that you have access to an oracle that returns either (1) always $(\mathbf{a}, \langle \mathbf{a}, \mathbf{s}\rangle + e)$, with fixed random secret $\mathbf{s}$, (2) $(\mathbf{a}, b)$, for $\mathbf{a}$ random vector, and b a random bit. This *Learning Parity with Noise* assumption (LPN) tackles the hardness of distinguishing if the oracle is in case (1) or (2). The assumption can be described in different flavors, including the dependence on the number of samples one can request from the oracle. Suppose that the number of calls to the oracle is bounded by n. We can construct the matrix $\mathbf{G}$ formed with the $\mathbf{a}^\intercal$ as rows, and the vector $\mathbf{e} = (e_0, \ldots, e_{n-1})^\intercal$. Then it boils down to distinguishing $(\mathbf{G}, \mathbf{Gs} + \mathbf{e})$ from $(\mathbf{G}, \mathbf{y})$ where $\mathbf{y} \coloneqq (b_0, \ldots, b_{n-1})^\intercal$ is a random vector. We recognize the standard decoding problem, with the matrix $\mathbf{G}$ being a generator matrix. Let $\mathbf{H}$ be a parity check-matrix associated with $\mathbf{G}$. Multiplying by $\mathbf{H}$ on the left, we obtain that it is equivalent to distinguishing between $(\mathbf{H}, \mathbf{He})$ and $(\mathbf{H}, \mathbf{s})$, where $\mathbf{s}$ is a random vector. This is nothing else than the syndrome decoding assumption. As we are fundamentally using codes and their properties, we decided to change the standard used

terminology and replace it with Syndrome Decoding (SD) instead of Learning Parity with Noise (LPN), but no difference other than semantic is to be found behind this change.

2.4.2.1 Other Flavors of Syndrome Decoding

Research in coding theory quickly found ways to circumvent the hardness of syndrome decoding by designing specific classes of codes $\mathfrak{C} = \{\mathcal{C}_1, \mathcal{C}_2, \dots\}$ that come with easy decoding algorithms and many other good properties. However, this is precisely what we want to avoid, as we aim to use the hardness of decoding as a basis for constructing cryptographic schemes. Still, restricting ourselves to specific classes of codes can be interesting for many reasons, such as reducing the cost of multiplication by $\mathbf{H}$, or reducing the memory cost. The syndrome decoding assumption would then ask to distinguish between $(\mathbf{H}, \mathbf{He})$ and $(\mathbf{H}, \mathbf{s})$, with $\mathbf{H} \xleftarrow{\$} \mathfrak{C}, \mathbf{s} \xleftarrow{\$} \mathbb{F}_q^n, \mathbf{e} \xleftarrow{\$} \mathcal{S}(\mathbb{F}_q^n, t)$. This new assumption has to be proven secure, as the structure of the matrices in $\mathfrak{C}$ leaks information to the adversary. The choice of a proper class of codes, and the analysis of syndrome decoding restricted to this class, is at the core of this thesis.

If we can consider variants where we restrict ourselves to specific classes of matrices, it is also possible to sample the error vectors from a specific class. For now we consider the error vector to be sampled uniformly at random in $\mathcal{S}(\mathbb{F}_q^n, t)$, but we will also consider the *regular* distribution at some point in the thesis. Other exotic noises distributions and their associated SD-like assumption will be considered in Chapter 6.

Definition 2.4.8 (t-regular sparse vector)**.** *A t-regular sparse vector is defined as follows: Sample $(\mathbf{u_i})_{1 \leqslant i \leqslant t}$, t unit vectors of size n/t. Return $\mathbf{e} = (\mathbf{u_1} \,/\!/ \cdots /\!/\, \mathbf{u_t})$ where $/\!/$ denotes vertical concatenation. Let call r-$\mathcal{S}(\mathbb{F}_q^n, t)$ the set of such t-regular sparse vectors. The associated syndrome decoding assumption (which involves distinguishing $(\mathbf{H}, \mathbf{s})$ from $(\mathbf{H}, \mathbf{He} : \mathbf{e} \xleftarrow{\$} r\text{-}\mathcal{S}(\mathbb{F}_q^n, t))$) is called the* Regular Syndrome Decoding *assumption and denoted r-SD. We said a noise is (c, t)-block-wise regular when it is the vertical concatenation of c different t-sparse vectors.*

2.4.2.2 Syndrome Decoding over a Ring

Let $\mathcal{R}$ be a ring equipped with a structure of $\mathbb{F}_q$-vector space. Let $\mathbf{B} = (b_i)_i : b_i \in \mathcal{R}$ be a basis of the $\mathbb{F}_q$-vector space. In what follows, we will denote $|\mathbf{B}|$ by B. For all $x \in \mathcal{R}$, $x = \sum_{b \in \mathbf{B}} x_b b$. Therefore, the vector space structure gives a natural isomorphism between $\mathcal{R}$ and $\mathbb{F}_q^B$. Let $\phi_{\mathbf{B}}$ be this isomorphism. Given the Hamming weight defined over vectors of $\mathbb{F}_q^B$ (Definition 2.4.2), we can define the Hamming weight over $\mathcal{R}$, denoted $wt_{\mathcal{R}}(\cdot)$. For $x \in \mathcal{R}$, $wt_{\mathcal{R}}(x) = wt\left(\phi_{\mathbf{B}}(x)\right)$. Note that the Hamming weight does not depend on the order of coordinate we consider, and therefore the ordering made on the basis does not matter. As the Hamming weight does not depend on the order of the coordinates, the Hamming weight is then well defined. Consequently, this Hamming weight over $\mathcal{R}$ allows us to similarly define the subset $\mathcal{S}(\mathcal{R}, t) \subset \mathcal{R}$ of elements that are of Hamming weight at most t. With this, we can already state the equivalent of SD but over this general $\mathcal{R}$.

Definition 2.4.9 ((Decisional) Syndrome Decoding Assumption over a Ring $\mathcal{R}$)**.** *Let $(n, k, t) = (n(\lambda), k(\lambda), t(\lambda))$ be parameters polynomial in the security parameter λ, and let $\mathcal{R}$ equipped with a basis. The goal of the* decisional ring syndrome decoding problem *is to distinguish, with a non-negligible advantage, between the distributions*

$$\begin{array}{lll} \mathcal{D}_0 : & (a, s) & \text{where } a, s \xleftarrow{\$} \mathcal{R} \qquad \text{and} \\ \mathcal{D}_1 : & (a, e_0 + a \cdot e_1) & \text{where } a \xleftarrow{\$} \mathcal{R}, e_i \xleftarrow{\$} \mathcal{S}(\mathcal{R}, t). \end{array}$$

The decisional syndrome decoding over a ring *assumption, denoted by* $(n, k, t)_q$-$\mathcal{R}$-SD *or* $\mathcal{R}$-SD *when the context is clear, states that this problem is hard. That is, for every polynomial-time algorithm* $\mathcal{A}$*, it holds that*

$$\left| \Pr_{a \xleftarrow{\$} \mathcal{R},\ (e_0,e_1) \xleftarrow{\$} \mathcal{S}(\mathcal{R},t)} [\ \mathcal{A}(a, e_0 + ae_1) = 0\ ] - \Pr_{(a,s) \xleftarrow{\$} \mathcal{R}} [\ \mathcal{A}(a, s) = 0\ ] \right| \leqslant \mathsf{negl}(\lambda),$$

where negl *denote a negligible function.*

The module *version is the following generalization: it asks to distinguish, with a non-negligible advantage, between the distributions:*

$$\begin{array}{lll} \mathcal{D}_0 : & (\mathbf{a}, s) & \text{where } \mathbf{a} \xleftarrow{\$} (\mathcal{R})^{c-1}, s \xleftarrow{\$} \mathcal{R} \\ \mathcal{D}_1 : & (\mathbf{a}, e_0 + a_1 e_1 + \cdots + a_{c-1} e_{c-1}) & \text{where } \mathbf{a} \xleftarrow{\$} (\mathcal{R})^{c-1}, \mathbf{e} = (e_0, \ldots, e_{c-1}) \xleftarrow{\$} (\mathcal{S}(\mathcal{R}, t))^c), \end{array}$$

where c *is called the* compression factor.

Remark 2.4.5. In the above definition we assumed a regularity in the error: each e_i is sampled as a random sparse element of the ring ($e_i \xleftarrow{\$} \mathcal{S}(\mathcal{R}, t)$). In general this has not to be the case, and we could choose $\mathbf{e}$ to be sampled from c element of $\mathcal{R}$ such that the sum of the weight of all the e_i is equal to ct, instead of having the same weight for all e_i. This would be a valid general SD assumption. Nevertheless in this manuscript we will always consider this regular variant where all the e_i are sampled from $\mathcal{S}(\mathcal{R}, t)$.

Remark 2.4.6 (From $\mathcal{R}$-SD to SD)**.** For $a \in \mathcal{R}$, let $f_a : \mathcal{R} \to \mathcal{R}$, $f_a(x) = ax$ denote the multiplication by a. Remind that $\phi_{\mathbf{B}}(x)$ denotes the vector in $\mathbb{F}_q^B$ which corresponds to the entries of x when written in the basis $\mathbf{B}$. For a fixed basis, there exists a unique matrix $\mathbf{M}_a$ known as the *matrix of multiplication by* a, such that $\phi_{\mathbf{B}}(f_a(x)) = \mathbf{M}_a \phi_{\mathbf{B}}(x)$. The matrix is defined as follows:

$$\mathbf{M}_a = \begin{bmatrix} \phi_{\mathbf{B}}(ab_0) \\ \vdots \\ \phi_{\mathbf{B}}(ab_i) \\ \vdots \\ \phi_{\mathbf{B}}(ab_{B-1}) \end{bmatrix}.$$

Next, the module $\mathcal{R}$-SD can be reformulated as a SD problem with

$$\mathbf{H} = \begin{bmatrix} \mathbf{I}_B & \mathbf{M}_{a_1} & \cdots & \mathbf{M}_{a_{c-1}} \end{bmatrix}$$

and $\mathbf{e} = (\phi_{\mathbf{B}}(e_0) \mathbin{/\!/} \cdots \mathbin{/\!/} \phi_{\mathbf{B}}(e_{c-1}))$, where $\mathbin{/\!/}$ denotes the vertical concatenation, a (c, t)- blockwise-regular noise version of size cB, and weight ct. The coefficient c is referred to as the *compression factor*, as $\mathbf{H}$ compress the error vector by a factor c when it multiplies it (from cB to B). $\mathbf{H}$ corresponds to a code of rate $(c-1)/c$.

2.4.3 The Quasi-Abelian Syndrome Decoding Problem

In this section, we recall the *Quasi-Abelian Syndrome Decoding* assumption (QA-SD), which was introduced in [BCCD23]. It is a restricted version of the syndrome decoding assumption in the case

of *Quasi-Abelian Codes* which generalize quasi-cyclic codes. Let $\mathbb{G}$ be a finite abelian group. We define the group algebra associated to $\mathbb{G}$ as follows:

$$\mathbb{F}_q[\mathbb{G}] \coloneqq \left\{ \sum_{g\in\mathbb{G}} a_g g \mid a_g \in \mathbb{F}_q \right\},$$

which is the set of all the formal linear combinations of elements of $\mathbb{G}$, with coefficients in $\mathbb{F}_q$. One can endow $\mathbb{F}_q[\mathbb{G}]$ with a commutative convolution product to make it an algebra of dimension $|\mathbb{G}|$.

$$\left(\sum_{g\in\mathbb{G}} a_g g\right)\left(\sum_{g\in\mathbb{G}} b_g g\right) \coloneqq \sum_{g\in G}\left(\sum_{h\in\mathbb{G}} a_h b_{h^{-1}g}\right) g.$$

There exists in $\mathbb{F}_q[\mathbb{G}]$ a canonical notion of Hamming weight: we define the Hamming weight $wt_{\mathbb{G}}(\cdot)$ of an element as the number of non-zero coordinates when written in the basis $(g)_{g\in\mathbb{G}}$ (independent of the ordering chosen). This allows us to define the set of t-sparse elements of $\mathbb{F}_q[\mathbb{G}]$, denoted $\mathcal{S}(\mathbb{F}_q[\mathbb{G}], t) = \{x \mid x \in \mathbb{F}_q[\mathbb{G}], wt_{\mathbb{G}}(x) = t\}$.

2.4.3.1 Quasi-Abelian Codes

We can define codes over $\mathbb{F}_q[\mathbb{G}]$. Given a matrix

$$\mathbf{\Gamma} = \begin{pmatrix} \gamma_{1,1} & \cdots & \gamma_{1,k} \\ \vdots & \ddots & \vdots \\ \gamma_{\ell,1} & \cdots & \gamma_{\ell,k} \end{pmatrix} \in (\mathbb{F}_q[\mathbb{G}])^{\ell\times k},$$

the quasi-$\mathbb{G}$ code defined by $\mathbf{\Gamma}$ is, in a similar fashion as before,

$$\mathcal{C} \coloneqq \{\mathbf{\Gamma m} \mid \mathbf{m} = (m_1, \ldots, m_k)^\intercal \in (\mathbb{F}_q[\mathbb{G}])^k\}.$$

That is $\mathbf{\Gamma}$ is the generator matrix of the code, written over $\mathbb{F}_q[\mathbb{G}]$. A random matrix $\mathbf{\Gamma}$ would be a matrix whose entries $\gamma_{i,j} \in \mathbb{F}_q[\mathbb{G}]$ are chosen at random. Because $\mathbb{F}_q[\mathbb{G}]$ enjoy a vector space structure, once you fix an order among the element of $\mathbb{G}$, there exists an isomorphism ϕ from $\mathbb{F}_q[\mathbb{G}]$ to $\mathbb{F}_q^n$ which associates $a \in \mathbb{F}_q[\mathbb{G}]$ to $(a_0, \ldots, a_{n-1})$. Further, we can represent the multiplication by an element $a \in \mathbb{F}_q[\mathbb{G}]$ by the following matrix:

$$\mathbf{M}_a = \begin{pmatrix} \varphi(a \cdot g_0) \\ \vdots \\ \varphi(a \cdot g_{n-1}) \end{pmatrix} \in \mathbb{F}_q^{n\times n},$$

where each row is the vector representation of a shift of a by some element $g_i \in \mathbb{G}$.

Remark 2.4.7. In the case $\mathbb{G} = \mathbb{Z}/nZ$, the quasi-$\mathbb{Z}/n\mathbb{Z}$ code is defined by a matrix with circulant blocks of the following shape over $\mathbb{F}_q$:

$$\mathbf{H} = \left(\begin{array}{ccc|c|ccc} a_0^{(0,0)} & a_1^{(0,0)} & \cdots & a_{n-1}^{(0,0)} & & a_0^{(0,k-1)} & a_1^{(0,k-1)} & \cdots & a_{n-1}^{(0,k-1)} \\ a_{n-1}^{(0,0)} & a_0^{(0,0)} & \cdots & a_{n-2}^{(0,0)} & & a_{n-1}^{(0,k-1)} & a_0^{(0,k-1)} & \cdots & a_{n-2}^{(0,k-1)} \\ \vdots & \vdots & \ddots & \vdots & \cdots & \vdots & \vdots & \ddots & \vdots \\ a_1^{(0,0)} & a_2^{(0,0)} & \cdots & a_0^{(0,0)} & & a_1^{(0,k-1)} & a_2^{(0,k-1)} & \cdots & a_0^{(0,k-1)} \\ \hline \multicolumn{3}{c|}{\vdots} & \ddots & \multicolumn{3}{c}{\vdots} \\ \hline a_0^{(\ell-1,0)} & a_1^{(\ell-1,0)} & \cdots & a_{n-1}^{(\ell-1,0)} & & a_0^{(\ell-1,k-1)} & a_1^{(\ell-1,k-1)} & \cdots & a_{n-1}^{(\ell-1,k-1)} \\ a_{n-1}^{(\ell-1,0)} & a_0^{(\ell-1,0)} & \cdots & a_{n-2}^{(\ell-1,0)} & & a_{n-1}^{(\ell-1,k-1)} & a_0^{(\ell-1,k-1)} & \cdots & a_{n-2}^{(\ell-1,k-1)} \\ \vdots & \vdots & \ddots & \vdots & \cdots & \vdots & \vdots & \ddots & \vdots \\ a_1^{(\ell-1,0)} & a_2^{(\ell-1,0)} & \cdots & a_0^{(\ell-1,0)} & & a_1^{(\ell-1,k-1)} & a_2^{(\ell-1,k-1)} & \cdots & a_0^{(\ell-1,k-1)} \end{array}\right).$$

Proposition 2.4.3. *Let $\mathbb{G}$ be a finite abelian group. $\mathbb{G}$ is isomorphic to a direct product of cyclic groups $\mathbb{G} \simeq \mathbb{Z}/d_1\mathbb{Z} \times \cdots \times \mathbb{Z}/d_r\mathbb{Z}$ where the d_i's can be equal. Then,*

$$\mathbb{F}_q[\mathbb{G}] \simeq \mathbb{F}_q[X_1, \ldots, X_r]/(X_1^{d_1} - 1, \ldots, X_r^{d_r} - 1),$$

and the isomorphism is given by $(k_1, \ldots, k_r) \mapsto X_1^{k_1} \cdots X_r^{k_r}$, and extended by linearity.

Definition 2.4.10 ((Decisional) Quasi-Abelian Syndrome Decoding Assumption)**.** *Let $(n, k, t) = (n(\lambda), k(\lambda), t(\lambda))$ be parameters polynomial in the security parameter λ, and let $\mathbb{G}$ be an abelian group. The goal of the* decisional Quasi-Abelian Syndrome Decoding problem *is to distinguish, with a non-negligible advantage, between the distributions*

$$\begin{array}{lll} \mathcal{D}_0: & (a, s) & \text{where } a, s \xleftarrow{\$} \mathbb{F}_q[\mathbb{G}] \qquad \text{and} \\ \mathcal{D}_1: & (a, a \cdot e_1 + e_2) & \text{where } a \xleftarrow{\$} \mathbb{F}_q[\mathbb{G}], e_i \xleftarrow{\$} \mathcal{S}(\mathbb{F}_q[\mathbb{G}], t)). \end{array}$$

The Decisional Quasi-Abelian Syndrome Decoding *assumption, denoted by* QA-SD *, states that this problem is hard. That is, for every probabilistic polynomial-time algorithm $\mathcal{A}$, it holds that*

$$\left| \Pr_{a \xleftarrow{\$} \mathbb{F}_q[\mathbb{G}],\ (e_0,e_1) \xleftarrow{\$} \mathcal{S}(\mathbb{F}_q[\mathbb{G}],t)} [\, \mathcal{A}(a, ae_0 + e_1) = 0 \,] - \Pr_{(a,s) \xleftarrow{\$} \mathbb{F}_q[\mathbb{G}]} [\, \mathcal{A}(a, s) = 0 \,] \right| \leqslant \mathsf{negl}(\lambda).$$

where negl *denotes a negligible function.*

The module version of QA-SD *is defined in the same manner as for $\mathcal{R}$-*SD*. It asks to distinguish, with a non-negligible advantage, between the distributions*

$$\begin{array}{lll} \mathcal{D}_0: & (\mathbf{a}, s) & \text{where } \mathbf{a} \xleftarrow{\$} (\mathbb{F}_q[\mathbb{G}])^{c-1}, s \xleftarrow{\$} \mathbb{F}_q[\mathbb{G}] \qquad \text{and} \\ \mathcal{D}_1: & (a, e_0 + a_1e_1 + \cdots + a_{c-1}e_{c-1}) & \text{where } \mathbf{a} \xleftarrow{\$} (\mathbb{F}_q[\mathbb{G}])^{c-1}, \mathbf{e} = (e_0, \ldots, e_{c-1}) \xleftarrow{\$} (\mathcal{S}(\mathbb{F}_q[\mathbb{G}], t))^c. \end{array}$$

Similarly as before, we can use the matrix representation of each element $a_i \in \mathbf{a} \in (\mathbb{F}_q[\mathbb{G}])^{c-1}$ *to reformulate the problem as a standard syndrome decoding problem with*

$$\mathbf{H} = \begin{bmatrix} \mathbf{I}_{|\mathbb{G}|} & \mathbf{M}_{a_1} & \dots & \mathbf{M}_{a_{c-1}} \end{bmatrix}$$

and $\mathbf{e} = (\phi(e_0) /\!\!/ \cdots /\!\!/ \phi(e_{c-1}))$*, where* $/\!\!/$ *denotes the vertical concatenation.* $\mathbf{e}$ *is a* (c, t)*-blockwise-regular*[2] *noise vector of size* $c|\mathbb{G}|$*, and weight* ct*, with* c *the compression factor. The thus defined matrix* $\mathbf{H}$ *corresponds to a code of rate* $(c-1)/c$*.*

2.5 Secure Multiparty Computation

Secure Computation (also called MPC) is broadly defined as the set of all the techniques that let N parties perform computations on private information. Let f be the function to be computed and let x_i for $1 \leqslant i \leqslant N$ be the private input of party i. The goal is therefore to compute the value of $f(x_1, \dots, x_N)$ while at the same time ensuring that no party, or group of parties, learn anything beyond what can be implied by the result of the function and its inputs.

MPC was introduced in the early days of modern cryptography, first by Andrew Yao [Yao82] for the two-party case, and by Goldreich, Micali, and Wigderson for the multiparty case [GMW87]. Next, Beaver [Bea91; Bea92] and subsequently Micali and Rogaway [MR92] defined precisely the study of privacy in secure computation protocols. The definitions and the models were later refined in 2000 by Canetti [Can01]. If MPC was at its early stages considered mostly as a curiosity with impractical protocols, it has gradually evolved, becoming more and more efficient. This area of cryptography is nowadays really promising, with numerous applications, as already mentioned in the introduction (Chapter 1).

We will first give an overview of the standard context and of the assumptions commonly made when working in MPC (Section 2.5.1), after which we will give a formal definition of MPC (Section 2.5.2). We will present some important functionalities that will be useful in this manuscript (Section 2.5.3) and finally present a major technique enabling us under certain conditions to achieve fast MPC (Section 2.5.4).

2.5.1 Description of the Framework

MPC comes in many flavors. It has been around for over 40 years, and even if it was a niche specialization for a long time, it became increasingly popular and widespread. Next, we describe the common assumptions in MPC, and more specifically the one related to our use case. The presentation below follows some part of [Gol09a; Cou23].

2.5.1.1 Communications Between Parties

MPC is all about different parties securely doing computation. To perform this computation, the parties have no choice but to exchange messages together. The first natural question would be how communication works between the parties. Classically, MPC supposes that the parties can send messages with perfect encryption and that everyone can talk to everyone, via authenticated channel. All attacks that consist in tampering with the messages or applying man-in-the-middle techniques are de facto discarded. It is also widely assumed that all communications proceed synchronously, getting rid of all the difficulties inherent to distributed computing. This choice may seem idealistic - and it is! - but the motivation is to compartmentalize properly the difficulties of the problems we

2. Definition 2.4.8

have to solve. These not-mentioned "real world" difficulties are in fact subjects of study on their own, and therefore this choice was made as a way to simplify an already quite complicated area. Nonetheless, we mention that if this assumption is usually the case in standard MPC, and throughout this manuscript, some areas of MPC try to understand what would happen if the network is unstable (for example incomplete or with faults).

2.5.1.2 Parties and Corruptions

MPC aims to perform secure computation with multiple parties. In the manuscript, we will denote by N the number of parties. The minimal number of parties requested to do MPC is therefore $N = 2$. This particular case defines a subclass of MPC protocol on its own, called 2PC. It serves as a minimal case on which one can build in order to obtain protocols for N parties. In such an N-party protocol, it would be unrealistic to think that an adversary could corrupt just one party. Suppose there are N different parties $P_1, \ldots, P_N$. An adversary could corrupt several parties, the subset P_S, where $S \subset [1, N]$. Therefore, the adversary might be able to access sensitive data by pooling the information of each corrupted party. This is why security is restated as follows: even if a group S of size $|S| = t$ parties combines the information they gathered during the execution of the protocol, they should not be able to get more information than what is implied by the result of the function and their inputs. The value t introduced above is called the *corruption threshold*. Depending on the security model, the number of corruption a scheme can support varies. In the thesis, we will mostly describe protocols for only two players - and specify when more than two players are involved.

2.5.1.3 Functionalities

We abstract the function to be computed as a *functionality*. For a protocol Π with N parties, we call functionality a probabilistic function $f\colon (\{0,1\}^*)^N \to (\{0,1\}^*)^N$ which associates $\mathbf{x} = (x_1, \ldots, x_N)$ to $f(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_m(\mathbf{x}))$. We say that a protocol Π realizes the functionality f when at the end of the protocol the i-th party obtains the random variable $f_i(x_1, \ldots, x_N)$.

For the sake of convenience, we assume that all players' input sizes are equal. While this assumption may not always hold, we can address discrepancies by padding shorter inputs with zeros. This ensures that protocols with unequal input sizes can be transformed into ones with equal-sized inputs.

2.5.1.4 Malicious Security and Semi-Honest Security

Two types of security are classically considered in MPC: *malicious security* tackles the case of an adversary that would make the corrupted parties deviate arbitrarily from the protocol. That is, the corrupted parties may also send wrong messages, or to the wrong recipient. On the contrary, *semi-honest security* encompasses security against adversaries that follow the protocol correctly but want to learn as much as possible from the other player's inputs. Namely, they will remember the different messages received, coin tosses made, and results obtained, and try to derive from it new information. This is why it is also called *honest-but-curious*.

In this thesis, we focus only on semi-honest security. The question of malicious security, more involved, will not be studied apart from when expressly mentioned.

2.5.1.5 Universal composability

In this manuscript we use a result from [Gol09b]: in the context of semi-honest security, it is guarantee that protocols remain secure even when arbitrarily composed with other instances of the same or other protocols: we have universal composability for free. Namely, a high-level protocol

proven secure assuming the existence of a perfect oracle (an ideal functionality) will still be secure if we replace this ideal functionality with a real protocol securely computing the same functionality.

2.5.2 Formal privacy

In order to tackle precisely the security, we should first define precisely the information that the adversary knows when performing the protocol. This is commonly referred to as the *view* of a party.

Definition 2.5.1 (View of a party). *Let $P_1, \dots, P_N$ denote parties that want to compute a function f on joint input $\mathbf{x} = (x_1, \dots, x_N)$. Then the* view *of party P_i is defined as all the messages received by P_i during the execution of the protocol, and all the different random coin tosses it may have made. We denote it as $\mathsf{View}_i(\mathbf{x})$. For $S \subset [1, N]$, we denote by $\mathsf{View}_S(\mathbf{x})$ the joint view of each parties $(P_i)_{i\in S}$. Similarly, $P_S \coloneqq (P_i)_{i\in S}$ and $x_S \coloneqq (x_i)_{i\in S}$.*

Further, we describe the output distribution of the protocol.

Definition 2.5.2 (Output Distribution). *We denote by $\mathcal{O}_i(\mathbf{x})$ the output distribution of the output of P_i after the execution of the protocol Π on joint input $\mathbf{x}$. For $S \subset [1, N]$, we write by $\mathcal{O}_S(\mathbf{x})$ for $(\mathcal{O}_i(\mathbf{x}))_{i\in S}$.*

We now have all the tools in order to formally define security against semi-honest adversaries.

Definition 2.5.3 (t-Privacy for probabilistic functionalities). *Let $f : (\{0,1\}^*)^N \to (\{0,1\}^*)^N$ be a N party probabilistic functionality. We say that a protocol Π satisfies t-*privacy *against semi-honest adversaries if:*

- *f is correctly computed.*
- *There exist a probabilistic polynomial time adversary Sim such that, for every subset $S \subset [N]$ of size $|S| = t$,*

$$\{\mathsf{View}_S(\mathbf{x}), \mathcal{O}_S(\mathbf{x})\} \approx \{(a, b) \mid a \leftarrow \mathsf{Sim}(S, \mathbf{x_S}, f_S(\mathbf{x})), b \leftarrow f(\mathbf{x})\},$$

  where View_S is the joint distribution of the views and $\mathcal{O}_S(\mathbf{x})$ the output of the corrupted parties.

One can convince oneself that the equation exactly encompasses what is required: the first entry of the tuple indicates that on each pair of inputs, it is possible to simulate the views of a group of parties solely from the joint input and output of these parties. While this is sufficient in the case of deterministic functionality, one has to be careful when dealing with random functionalities. We have to be sure that the distribution of the output of the protocol matches the distribution of the output of the random functionality being emulated. Note that this is trivially the case when we have a deterministic function.

2.5.2.1 Equivalence to the Ideal world vs Real World model

Another model exists for defining the security of MPC protocols: the *ideal world vs real world* model. In an *ideal world*, a trusted party $\mathcal{T}$ exists. A protocol Π to compute a functionality f is therefore straightforward. Each party P_i sends its private input x_i to $\mathcal{T}$, who computes $f(x_1, \dots, x_N)$ and outputs the result to all the parties. In an ideal world, an adversary cannot do anything. Indeed, in such a model, the adversary could corrupt any party P_i, or even a group of parties $P_{i_1}, \dots, P_{i_t}, S = \{i_1, \dots, i_t\} \subset [1, N]$; but not $\mathcal{T}$ which is by definition a trusted party. Then, the adversary does

nothing apart from sending its inputs and then receiving the result of the function. It has no more power, and therefore, this achieves the MPC requirements: *the adversary does not learn more than what is implied by the protocol.* This model is called *ideal world* because we suppose the existence of a perfect trusted party $\mathcal{T}$, which is highly unrealistic. However, the interest of the ideal world is that it encompasses the security requirements we want, as they are trivially achieved in this model. In the *real world* however, the situation is less pleasant: there are no trusted parties. In this real world, parties will communicate to compute the result of the function, sending messages to each other. Since the adversary can corrupt an arbitrary set of parties, it could gain information from those messages. In order for the real world to be secure we would require that everything that the adversary could achieve in the real world could also be achieved in the ideal world. This alternative definition of security is actually equivalent to the one presented above using a simulator (see [Gol09a]).

2.5.3 Cornerstone Functionalities in MPC

Next, we present basic but cornerstone functionalities in MPC. We will describe them in terms of ideal functionalities: it encapsulates exactly what the functionality is expected to do, as it would in an ideal world.

2.5.3.1 Additive Secret Sharing

Secret sharing is an essential functionality that encapsulates very important cryptographic properties while being simple. In essence, an additive secret-sharing scheme with N players divides a secret s into N different shares. These shares, when summed up, reconstruct to the original secret s, while ensuring that no information about s or the other shares is revealed as long as only $t \leqslant n-1$ shares are compromised.

Definition 2.5.4 (Additive Secret Sharing)**.** *The functionality is described in Figure 2.2. Let $s \in \mathbb{F}_q$ be a fixed secret value. An additive secret sharing scheme produces for each party an additive share of the secret value, that we denote by $[\![s]\!]_i$* for party i. It has to verify the following properties:

- *Correctness:* $\sum_{i=0}^{n-1} [\![s]\!]_i = s$.
- *Information theoretic security:* Let $S \subset [N], |S| = t < n$. The knowledge of $[\![s]\!]_S := ([\![s]\!]_i)_{i \in S}$ reveals no information about the secret in the information-theoretic sense.

Secret-sharing offers information-theoretic security, which implies that even with unbounded computing power, no adversary can break the security with more advantage than guessing at random. It exemplifies perfect security.

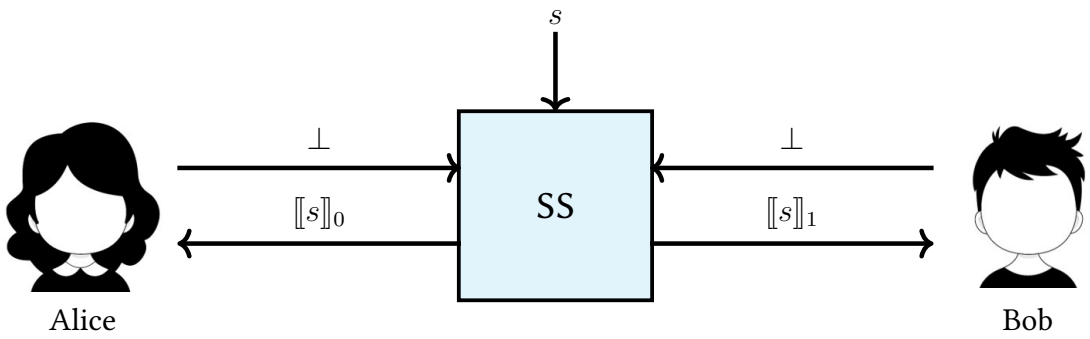

Figure 2.2 – Ideal functionality of Secret Sharing, 2PC case

Lemma 2.5.1 (Linearity of Secret-Sharing). *The map associating $s \in \mathbb{F}_q$ to its share $[\![s]\!]_i$ is linear.*

Proof. We check that for $x \in \mathbb{F}_q$ and $y \in \mathbb{F}_q$, the following holds

$$\sum_{i=0}^{n-1} [\![x]\!]_i + [\![y]\!]_i = \sum_{i=0}^{n-1} [\![x]\!]_i + \sum_{i=0}^{n-1} [\![y]\!]_i = x + y.$$

This proves that the addition of shares is a share of the addition. The same applies to multiplication by a constant. □

Remark 2.5.1. A party P_0 holding an element $x \in \mathbb{F}_q$ can easily split x into shares and send the shares to the other parties in order for the group to have shares of x. To do this, the party can simply send to each other party a random element r_i as its share and set his share to be $[\![x]\!]_0 = x - \sum_{i=1}^{n-1} r_i$. Note that in this case, party P_0 still knows the shared value x.

2.5.3.2 Oblivious Transfer

Now, we describe another fundamental cryptographic primitive: *the Oblivious Transfer.*

Definition 2.5.5 (Oblivious Transfer). *The ideal functionality is displayed in Figure 2.3. An* Oblivious Transfer (OT)*, denotes a two-party protocol with a sender and a receiver. The sender holds two inputs $x_0, x_1 \in \mathbb{F}_q$, while the receiver possesses a selection bit $b \in \mathbb{F}_2$. At the end of the protocol, the receiver should get the value x_b. The security requirements are as follows:*

- Sender Security: *The receiver does not learn any information about the value x_{1-b}*
- Receiver Security: *The sender does not learn any information about the value b.*

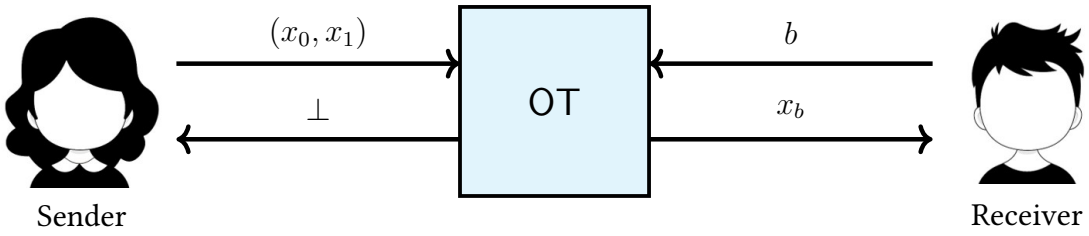

Figure 2.3 – Ideal OT functionality

The concept of Oblivious Transfer is essentially encapsulated by its name: it is exactly as if the sender manages to give to the receiver the value corresponding to the selection bit b, while not knowing which value he is sending. The concept of OT can be generalized: k-out-of-n OT denotes the situation in which the sender has n inputs, among which the receiver can choose k values. OT is one of the fundamental building blocks for secure computation.

Remark 2.5.2 (Construction of an OT). We thereafter provide a standard construction of OT, following the original construction from [BM90]. The construction uses the famous discrete logarithm assumption as a hard problem to prove its security. Discrete logarithm asks to find from $(g, h = g^k)$ the exponent of h, that is k. Let $\mathbb{G}$ be a group of prime order p, and let $g \in \mathbb{G}$ be a generator of $\mathbb{G}$, and $H : \mathbb{G} \to \{0,1\}^n$ be a hash function. Suppose that the sender holds $x_0, x_1 \in \{0,1\}^n$ two messages, and the receiver holds a bit $b \in \mathbb{F}_2$. The protocol is as follows:

— The sender samples a random element $r \xleftarrow{\$} \mathbb{G}$, and sends it to the receiver. The receiver then samples $k \xleftarrow{\$} \mathbb{Z}_p$, and lets $\mathsf{pk}_b = g^k$ and $\mathsf{pk}_{1-b} = r/g^k$. He sends $(\mathsf{pk}_0, \mathsf{pk}_1)$ to the sender. Note that the receiver does not know the discrete logarithm of pk_{1-b}. The sender can verify that $\mathsf{pk}_0\mathsf{pk}_1 = r$.

— The sender constructs $D_0 = (g^{r_0}, H(\mathsf{pk}_0^{r_0}) \oplus x_0), D_1 = (g^{r_1}, H(\mathsf{pk}_1^{r_1}) \oplus x_1)$: that is, he encrypts the messages (x_0, x_1) with keys $(\mathsf{pk}_0, \mathsf{pk}_1)$, and send (D_0, D_1) to the receiver.

— The receiver can decrypt $D_b := (u_1, u_2)$, by computing $x_b = H(u_1^k) \oplus u_2$.

It is straightforward to check the correctness of the protocol. Moreover, it is interesting to see that the sender cannot learn any information on b and that it achieves information-theoretic security. Indeed, the sender can only see two equivalent keys $\mathsf{pk}_0, \mathsf{pk}_1$. On the receiver side, learning x_{1-b} would be equivalent to breaking the decision Diffie-Hellman assumption, a very standard cryptographic assumption which states that given (g, g^x, g^y) it is still hard to distinguish g^{xy} from a random element of $\mathbb{G}$.

Another variant is the OT is the *Random Oblivious Transfer* (ROT). The functionality is displayed in Figure 2.4. It consists in sampling randomly the values x_0, x_1 and b, and outputting (x_0, x_1) to the sender and b, x_b to Bob. This random variant of OT is the one that we will cover the most in the rest of the manuscript, and we will abuse the notation OT to denote it.

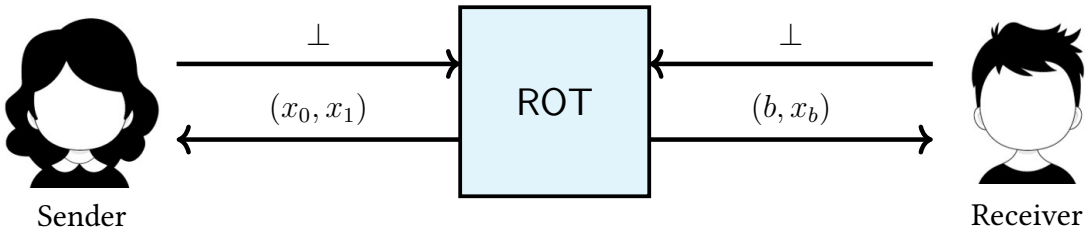

Figure 2.4 – Ideal ROT functionality

2.5.3.3 Oblivious Linear Evaluations

Definition 2.5.6 (Oblivious Linear Evaluations)**.** *The ideal functionality is displayed in Figure 2.5. An* Oblivious Linear Evaluation(OLE)*, denotes a two-party protocol, with a sender and a receiver. The sender holds two inputs $u, v \in \mathbb{F}_q$, and the receiver holds an* evaluation point $\Delta \in \mathbb{F}_q$*. At the end of the protocol, the receiver should get the value $w = u \cdot \Delta + v$. The security requirements are as follows:*

— Sender Security: *The receiver does not learn any additional information on u, v than what is implied by Δ, w.*

— Receiver Security: *The sender does not learn any information about Δ, w.*

A generalization of OLE *is* Vector Oblivious Linear Evaluation*(*VOLE *), where u, v, w are now vectors: $\mathbf{u}, \mathbf{v}, \mathbf{w}$ such that $\mathbf{w} = \mathbf{u} \cdot \Delta + \mathbf{v}$.*

The goal of OLE is again clear from its name: it is an evaluation of a linear function ($\cdot \to u \times \cdot + v$), but with no complete information for both sides. The receiver knows the evaluation point and the result but does not know the function, while the sender knows the function but does not know the evaluation point. OLE can be seen as a generalization of OT. In an OT, the value x_b can always be expressed as follows:

$$x_b = (1 - b)x_0 + bx_1 = (x_1 - x_0)b + x_0$$

Therefore, up to writing $u = (x_1 - x_0)$, $\Delta = b$ and $v = x_0$, this corresponds exactly to the description of an OLE in the case $\mathbb{F}_2$.

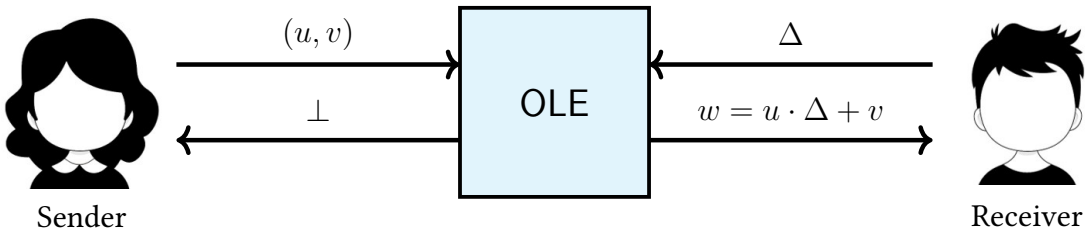

Figure 2.5 – Ideal OLE functionality

Again, another flavor of OLE is *random* OLE (i.e. ROLE) where u, v, and Δ are not known beforehand by the players but are sampled at random during the execution of the protocol. The functionality outputs (u, v) to the Sender and $(\Delta, w \coloneqq u \cdot \Delta + v)$ to the Receiver. Exactly as before, we abuse notation and denote by OLE an instance of *random* OLE, apart when explicitly stated otherwise. Note also that an ROLE can be rewritten using secret-sharing: giving $(u, [\![u \cdot \Delta]\!]_1)$ to the sender and $(\Delta, [\![u \cdot \Delta]\!]_0)$ to the receiver describe an OLE. We will also use this notation throughout the manuscript.

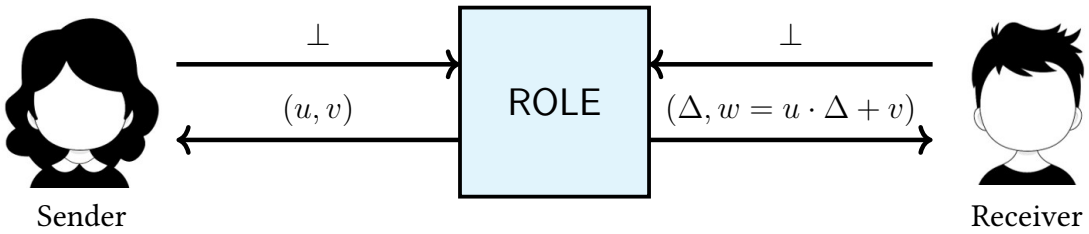

Figure 2.6 – Ideal ROLE functionality

Lemma 2.5.2 (Construction of OLE from OT, from [Gil99])**.** *An* OLE *over* $\mathbb{F}_q$ *can be obtained from* $t = \log(q)$ *invocations to an* OT.

Proof. We assume that q is prime for the proof. The protocol is as follows:

- The sender samples two random elements $u, v \in \mathbb{F}_q$. The receiver samples a random $\Delta \in \mathbb{F}_q$. Let $(\Delta_0, \ldots, \Delta_{t-1})$ be the description of Δ in binary.
- The sender splits v in shares v_i such that $v = \sum_{i=0}^{t-1} v_i$.
- The sender and the receiver perform t distinct OT protocols with sender inputs $(v_i, v_i + 2^i u)$ and with receiver inputs Δ_i.
- The receiver therefore obtains $v_i + \Delta_i 2^i u$, and outputs $\sum_i (v_i + \Delta_i 2^i u)$.

It is easy to check the correctness:

$$\sum_{i=0}^{t-1} v_i + \Delta_i 2^i u = \sum_{i=0}^{t-1} v_i + \sum_{i=0}^{t-1} \Delta_i 2^i u = v + \Delta u.$$

Regarding security, the sender does not learn anything from the t OT protocols, and the security of OT is enough to protect information on Δ. On the other hand, the receiver learns either (v_i) or $v_i \Delta_i 2^i u$ at each step, but because the v_i's are just random shares of v, this does not leak more information on v than it should. □

2.5.3.4 Beaver Triples

We now present a slightly more involved and very useful primitive known as *Beaver Triples*, named after the work of Beaver [Bea91; Bea92]. They are also called *multiplication triples*.

Definition 2.5.7 (Beaver Triples). *The ideal functionality is displayed in Figure 2.7. A* Beaver Triple (BT) *denotes a two party-protocol. The protocol ensures that each player obtains* $[\![a]\!], [\![b]\!], [\![c]\!]$ *with:*

$$a \xleftarrow{\$} \mathbb{F}_q, b \xleftarrow{\$} \mathbb{F}_q, c = a \cdot b.$$

A Beaver Triple is therefore a protocol that gives the players additive shares of values a and b, along with additive shares of their product $c = a \cdot b$. The interest relies on the fact that additive sharing does not commute with multiplication.

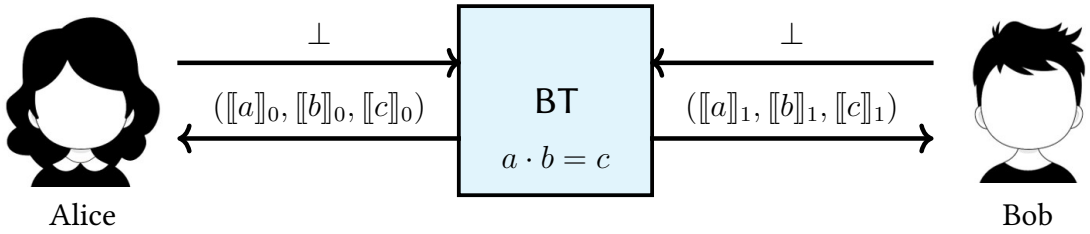

Figure 2.7 – Ideal BT functionality

Lemma 2.5.3 (construction of Beaver Triples). *A random Beaver Triple can be constructed from two instances of* OLE *(or* OT *in the case of* $\mathbb{F}_2$*).*

Proof. Let the parties hold two random OLEs. We want the parties to obtain shares of elements a, b, c where a, b are random elements and $c = a \cdot b$. Therefore, Alice holds $(u_1, [\![u_1\Delta_1]\!]_0)$ and $(u_2, [\![u_2\Delta_2]\!]_0)$ and Bob holds $(\Delta_1, [\![u_1\Delta_1]\!]_1)$ and $(\Delta_2, [\![u_2\Delta_2]\!]_1)$. Let Alice (resp Bob) set the value $[\![c]\!]_0$ (resp. $[\![c]\!]_1$) to be equal to $u_1u_2 + [\![u_1\Delta_1]\!]_0 + [\![u_2\Delta_2]\!]_0$ (resp. $\Delta_1\Delta_2 + [\![u_1\Delta_1]\!]_1 + [\![u_2\Delta_2]\!]_1$). Let $a = u_1 + \Delta_2$, $b = u_2 + \Delta_1$, and $c = a \times b$. Then one can verify that

$$c = a \times b = u_1u_2 + u_1\Delta_1 + u_2\Delta_2 + \Delta_1\Delta_2 = [\![c]\!]_0 + [\![c]\!]_1,$$

and therefore parties have thus constructed shares of $c = a \times b$. Note that because the two OLEs we consider were random instantiations, the values $a = u_1 + \Delta_1$ and $b = u_2 + \Delta_2$ remain random. □

2.5.3.5 Function Secret Sharing

The last primitive we would like to present is way more involved than the previous ones and much more recent. It is an important building block of the cryptographic constructions of this manuscript. Function Secret Sharing (FSS), introduced in [BGI15; BGI16], is the analog of additive secret sharing for functions. The following definitions are mostly taken from [BCGI+20b]. An FSS scheme splits a secret function $f : D \to \mathbb{G}$, where $\mathbb{G}$ is an Abelian group, into two functions f_0, f_1 (the shares of f). The two functions f_0, f_1 are supposed to be pseudorandom functions: they are both represented by a key, respectively k_0, k_1. The requirements are that $f_0(x) + f_1(x) = f(x)$ for every $x \in D$ and that both k_0 and k_1 hide f. The ideal functionality is displayed in Figure 2.8.

Definition 2.5.8 (Function Secret Sharing). *Let* $\mathfrak{C} = \{f : D \to \mathbb{G}\}$ *be a class of function. We denote by* $\mathsf{desc}(f)$ *the description of* f*, which also specifies the input domain* D *and an Abelian group* $(\mathbb{G}, +)$

as the output domain. A (2-party) function secret sharing *(*FSS*) scheme for $\mathfrak{C}$ is a pair of algorithms* $\mathsf{FSS} = (\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$ *with the following syntax:*

- $\mathsf{FSS.Gen}(1^\lambda, desc(f))$ *is a Probabilistic Polynomial Time (PPT) algorithm that given security parameter λ and a description of $f \in \mathfrak{C}$ outputs a pair of keys (k_0, k_1). We assume that the keys specify D and $\mathbb{G}$.*
- $\mathsf{FSS.Eval}(b, k_b, x)$ *is a polynomial-time algorithm that, given a key k_b for party $b \in \{0,1\}$, and an input $x \in D$, outputs a group element $y_b \in \mathbb{G}$.*

The scheme should satisfy the following requirements:

- **Correctness:** *For any $f \in \mathfrak{C}$ and $x \in D$, we have*

$$\Pr_{(k_0,k_1) \xleftarrow{\$} \mathsf{FSS.Gen}(1^\lambda, \mathsf{desc}(f))} \left[\sum_{b \in \{0,1\}} \mathsf{FSS.Eval}(b, k_b, x) = f(x) \right] = 1.$$

- **Security:** *For any $b \in \{0,1\}$, there exists a PPT simulator* Sim *such that for any polynomial-size function sequence $f_\lambda \in \mathfrak{C}$, the distributions $\{k_b \mid (k_0, k_1) \xleftarrow{\$} \mathsf{FSS.Gen}(1^\lambda, f_\lambda)\}$ and $\{k_b \xleftarrow{\$} \mathsf{Sim}(1^\lambda, \mathsf{Leak}(f_\lambda))\}$ are computationally indistinguishable.*

In the constructions we use, the leakage function $\mathsf{Leak} : \{0,1\}^* \to \{0,1\}^*$ *is given by* $\mathsf{Leak}(f_\lambda) = (D, \mathbb{G})$*, namely, it outputs a description of the input and output domains of f.*

We also define a full-domain evaluation algorithm, $\mathsf{FSS.FullEval}(b, k_b)$, which outputs a vector of $|D|$ group elements, corresponding to running Eval on every element x in the domain D. For the type of FSS we consider, FSS.FullEval is significantly faster than the generic solution of running $|D|$ instances of Eval. In the manuscript, we will use FSS for point functions and sums of point functions, as defined below.

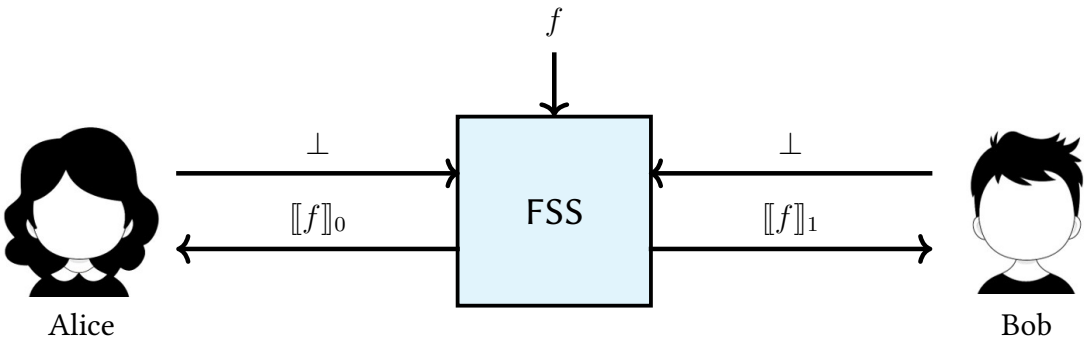

Figure 2.8 – Ideal functionality of FSS in the case of 2PC

FSS is a highly non-trivial primitive. As for now, very few classes of functions can be efficiently shared via FSS schemes. We present thereafter the one at the core of the main protocols of this thesis: the *Distributed Point Functions*.

Definition 2.5.9 (Distributed Point Function (DPF) [GI14; BGI15])**.** *Denote by $[n]$ the set of integers $\{0, \ldots, n-1\}$. For the abelian group $\mathbb{F}_q$, $\alpha \in [n]$, and $\beta \in \mathbb{F}_q$, the* point function $f_{\alpha,\beta}$ *is the function $f_{\alpha,\beta} : [n] \to \mathbb{F}_q$ defined as*

$$f_{\alpha,\beta}(x) := \begin{cases} \beta & \text{if } x = \alpha, \\ 0 & \text{otherwise.} \end{cases}$$

A distributed point function (DPF) *is an* FSS *scheme for the class of point functions* $\{f_{\alpha,\beta} : [n] \to \mathbb{F}_q \mid \alpha \in [n], \beta \in \mathbb{F}_q\}$.

In other words, a point function takes only one non-zero value β, at the specific input $x = \alpha$. An important point to note here is the following: the point function $f_{\alpha,\beta}$ can be represented by a vector of $\mathbb{F}_q^n$ of Hamming weight 1, i.e a vector of the form

$$(0, \ldots, 0, \beta, 0, \ldots, 0),$$

where β is in the α-th position. The DPF can therefore be used to share vectors of weight 1.

The DPF construction. We present thereafter the DPF from [GI14; BGI15]. We assume that the domain of the DPF is a power of 2, that is $n = 2^m$, and assume that $q = 2$ (and therefore $\beta = 1$). This choice is made for simplicity of demonstration while it is possible to construct a DPF for any n and any q. To create a DPF, we want Alice and Bob to hold respectively f_a and f_b, such that both f_a and f_b appear to be random. In Section 2.3.3.1, we have shown that given a seed and a PRG, one can create a pseudorandom function, thanks to the GGM tree-based construction. Nevertheless if we give the parties the same seed $\mathbf{s}$, players will hold additives shares of the all-zero function, and not a point function. The protocol is built upon a tweaked version of the GGM tree, and use a PRG $G : \mathbb{F}_q^\lambda \to \mathbb{F}_q^{2\lambda+2}$, that is just slightly more than length doubling. Let $\mathbf{s}$ be the seed of size λ, and let $t \in \mathbb{F}_q$ be an additional value called a *control element*. We let G_0 and G_1 be respectively the two parts of size $\lambda + 1$ of the output of G, that is $G(\mathbf{s}) = G_0(\mathbf{s})||G_1(\mathbf{s})$. We construct a tree as in the GGM construction:

- Label the root by $(\mathbf{s}, t)$.
- We label a node ν with label $L(\nu) = (\mathbf{s}(\nu), t(\nu))$. Consider its two children ν_0 and ν_1. We label them by respectively $L(\nu_0) = (\mathbf{s}(\nu_0), t(\nu_0))$ and $L(\nu_1) = (\mathbf{s}(\nu_1), t_1(\nu))$, where $G(\mathbf{s}(\nu)) = G_0(\mathbf{s}(\nu))||G_1(\mathbf{s}(\nu)) \coloneqq \mathbf{s}(\nu_0)||t(\nu_0)||\mathbf{s}(\nu_1)||t(\nu_1)$.

We index the leaves with $i \in [2^m]$ where m is the number of levels of the tree, and denote ν^i the i-th leaf. For a leaf ν^i, the *evaluation path* of ν^i denotes all the nodes from the root to ν^i. We define $f_k(i)$ as follows: compute all the labels of the nodes in the evaluation path corresponding to ν^i, that is, compute successively all the labels of the nodes from the root to the leaf, recover its the label of the leaf $(\mathbf{s}(\nu^i), t(\nu^i))$ and return $\mathbf{s}(\nu^i)$. For the same reasons as in the original GGM tree, it defines a pseudorandom function. Now, consider that we want the two pseudorandom functions f_a and f_b to differ on exactly the specific entry α. The leaf associated to α is ν^α. Consider a node ν in the tree, and look at the labels $L_a(\nu)$ and $L_b(\nu)$ associated respectively by Alice and Bob on ν. Using some additional *correction words*, we show how to maintain the following invariant at each step:

- If ν is not in the evaluation path, then the labels are equal: $L_a(\nu) = L_b(\nu)$.
- If ν is in the evaluation path then the $\mathbf{s_a}(\nu)$ and $\mathbf{s_b}(\nu)$ are indistinguishable from two independent and uniformly distributed vectors, and $t_a(\nu) + t_b(\nu) = 1$.

This invariant is motivated by the fact that if the node is on the evaluation path, the associated evaluation is still pseudorandom, whereas if it is not on the evaluation path then the associated evaluation is the same for both players and sums up to 0.

We will show that this property is true by recursion. First, we do not include the correction words to precisely identify where they are needed. Let us suppose that the invariant holds for a node ν. First, assume that ν is outside the evaluation path of ν_α. Note that the children of ν will not be in the evaluation path of ν_α. We apply the recursion hypothesis: the respective labels L_a, L_b of Alice

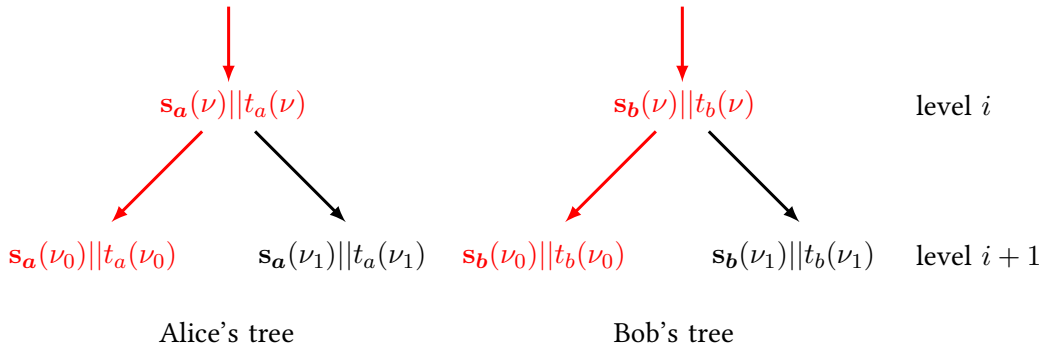

Figure 2.9 – Representation of the labels of a node on the evaluation path of Alice's and Bob's tree, and of its children, before correction. The evaluation path is represented in red.

and Bob are equal. It is straightforward to see that their children will have the same labelled: the labels of the children are uniquely determined by the one of their parents. Therefore, the recursion property is verified in this case. Now suppose that ν is on the evaluation path of ν_α. Then, one of its two children ν_0 and ν_1 will not be in the evaluation path anymore while the other will stay in it. The recursion hypothesis states that $\mathbf{s_a}(\nu)$ and $\mathbf{s_b}(\nu)$ are indistinguishable from being independent and uniformly distributed. Therefore this property propagates to the labels of the children. This is a good property for the child that stays on the evaluation path. The difficulty is therefore to make Alice's and Bob's labels of the node that leaves the evaluation path equal, moreover, because we are hiding the value α to the parties: the parties do not know the evaluation path and therefore which child is leaving the evaluation path.

The method introduced by Boyle et al. was to add to the key m *correction words*, one for each level of the tree. Without a lost of generalities, suppose that we are at level i, and that the child node that leaves the path is the left hand one. Let $\mathbf{w^{i+1}}$ be a *correction words* defined as:

$$\mathbf{w^{i+1}} := G(\mathbf{s_a}) + G(\mathbf{s_b}) - (\mathbf{0}||0||\tilde{\mathbf{s}}||1)$$

where $\tilde{\mathbf{s}}$ is a random seed. We replace the way the labels of the children are defined:

$$(\mathbf{s_a}(\nu_0)||t_a(\nu_0)||\mathbf{s_a}(\nu_1)||t_a(\nu_1)) = G(\mathbf{s_a}) + t_a\mathbf{w^{i+1}} \quad \text{and}$$
$$(\mathbf{s_b}(\nu_0)||t_b(\nu_0)||\mathbf{s_b}(\nu_1)||t_b(\nu_1)) = G(\mathbf{s_b}) + t_b\mathbf{w^{i+1}}$$

Because of the recursion hypothesis, we have that $t_a + t_b = 1$. Suppose that $t_a = 1$. Then, the right node ν_1 is labeled with $(\mathbf{s_a}(\nu_1), t_a(\nu_1)) = G(\mathbf{s_b}) + (\tilde{\mathbf{s}}, 1)$ by Alice, and with $\mathbf{s_b}(\nu_1), t_b(\nu_1) = G(\mathbf{s_b})$ by Bob. The recursion is still verified in this case. As for ν_0, the computation gives $(\mathbf{s_a}(\nu_0)||t_a(\nu_0)) = (\mathbf{s_b}(\nu_0)||t_b(\nu_0))$, and therefore $L_a(\nu_0) = L_b(\nu_0)$. We managed to make the node get out of the path successfully. Note that this new definition changes nothing in the case where the node is not in the evaluation path of ν_α: in this case, we know that both the labels $L_a(\nu)$ and $L_b(\nu)$ are the same, and therefore, again, their children will be labeled exactly in the same way ($G(\mathbf{s_a}) - t_a\mathbf{w}^{i+1} = G(\mathbf{s_b}) - t_b\mathbf{w}^{i+1}$).

Therefore, for each level of the tree, correction words are needed for the invariant to be correct and for the construction to work properly. We set the key k of each party to be:

– A seed $\mathbf{s} \in \mathbb{F}_q^\lambda$, and a random secret sharing of 1.

— A correction word per level of the tree, that is m different correction words for a tree of depth m.

The $\mathbf{w}^i$ can be computed beforehand during the creation of the DPF keys: from the seed, the functionality can compute all the labels on the evaluation path and deduce the correction words. Note that this construction manages to successfully produce a sharing of the DPF into two pseudorandom functions, but at the same time the correction words manage to hide the value α: indeed because we hide the correction words for every node, no information about the path can be learned.

The best known DPF construction [BGI16] use any PRG $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda+2}$ and has the following efficiency features. For $m = \lceil \frac{\log|\mathbb{G}|}{\lambda+2} \rceil$, the key generation algorithm Gen invokes G at most $2(\lceil \log n \rceil + m)$ times, the evaluation algorithm Eval invokes G at most $\lceil \log n \rceil + m$ times, and the full-domain evaluation algorithm FullEval invokes G at most $n \cdot (1+m)$ times. The size of each key is at most $\lceil \log n \rceil \cdot (\lambda+2) + \lambda + \lceil \log_2 |\mathbb{G}| \rceil$ bits. We will use a simple and generic extension of DPF to sum of point functions.

Definition 2.5.10 (FSS for sum of point functions (SPFSS))**.** *For $S = (s_1, \ldots, s_t) \in [n]^t$ and $\mathbf{y} = (y_1, \ldots, y_t) \in \mathbb{G}^t$, define the* sum of point functions $f_{S,\mathbf{y}} : [n] \to \mathbb{G}$ *by*

$$f_{S,\mathbf{y}}(x) = \sum_{i=1}^{t} f_{s_i,y_i}(x),$$

where $f_{s,y}$ is the point function that associates y on entry s and 0 elsewhere. An SPFSS *scheme is an* FSS *scheme for the class of sums of point functions.*

Note that for $S = (s_1, \ldots, s_t)$, the function $f_{S,\mathbf{y}}$ is non-zero on *at most* t points. If the elements of S are distinct, $f_{S,\mathbf{y}}$ coincides with a *multi-point function* for the set of points in S. A simple realization of SPFSS is by summing t independent instances of DPF: by using the linearity of additive sharing we obtain additive shares of sum of points function. This construction leads to a key size in $O(t \cdot (\log(n)\lambda + \log(|\mathbb{G}|)))$ and the number of operations of the full domain evaluation is dominated by tn group operations and evaluations of a PRG.

We will further explain (in Section 5.5.3) an optimization in the case of $\mathbb{G} = \mathbb{F}_q$, and with (c,t)-blockwise-regular vector.

To simplify notation, when generating keys for a scheme $\mathsf{SPFSS} = (\mathsf{SPFSS.Gen}, \mathsf{SPFSS.Eval})$, we write $\mathsf{SPFSS.Gen}(1^\lambda, 1^n, S, \mathbf{y})$, instead of explicitly writing $f_{S,\mathbf{y}}$.

2.5.4 Computing a Function Using Secret Sharing and Random Correlations: the GMW Protocol

We present thereafter a technique central to this manuscript, the so-called GMW protocol, named after its authors Goldreich, Micali, and Wigderson [GMW87]. Let $f : \mathbb{F}_q^{a+b} \to \mathbb{F}_q$, a functionality which maps $(\mathbf{x}, \mathbf{y})$ to $f(\mathbf{x}, \mathbf{y})$ such that $\mathbf{x} = (x_0, \ldots, x_{a-1}) \in \mathbb{F}_q^a$ is the input of Alice, and $\mathbf{y} = (y_0, \ldots, y_{b-1}) \in \mathbb{F}_q^b$ is the input of Bob. We represent the function f as an *arithmetic circuit* C, that is, a directed acyclic graph made up of inputs nodes, multiplications and additions gates. The goal of the protocol is to preserve the following invariant:

From an additive secret sharing of the inputs of a gate in the circuit, it is possible to compute a secret sharing of the output of this gate.

This is motivated for three reasons:

- The players can easily produce an additive secret sharing of their inputs and send the appropriate share to the other party.
- We have shown in Lemma 2.5.1 that additive secret sharing is linear and therefore the invariant is already true for the addition gate.
- The party would obtain at the end a sharing of the result. From there, they can easily reconstruct it by sending their share to each other.

Nevertheless, the multiplication gate causes a problem because $[\![xy]\!] \neq [\![x]\!] \times [\![y]\!]$.

2.5.4.1 Random Correlations as an Efficient Solution

Beaver came with an improvement over this protocol [Bea92; Bea95]: it consists to giving the parties an additional value, called *random correlation*, as a crutch to help.

Definition 2.5.11 (Correlated Randomness and correlation generator)**.** *Consider a 2PC protocol with only two parties, Alice and Bob. Let A, B be sets. Given a binary relation $R(a,b) : A \times B \to \{0,1\}$, we define the* correlation with respect to R *as the following set*

$$\mathtt{C}_R = \{(a,b) \in A \times B \mid R(a,b) = 1\}.$$

Given a binary relation R, we say we give Alice and Bob additional random correlations *when we give to Alice a and to Bob b such that*

$$(a,b) \xleftarrow{\$} \mathtt{C}_R.$$

A PPT algorithm $\mathcal{C}$ is called a correlation generator *for a correlation $\mathtt{C}$, if $\mathcal{C}$ on input 1^λ outputs a pair of elements (a,b) in $\mathtt{C}$.*

Obviously, this can be easily generalized for more than 2 parties. The *correlated randomness model* is a model in which we assume that parties are given random correlations, that they can use during the MPC protocols, as a helper.

Example 2.5.1. The fundamental functionalities presented in Section 2.5.3, when in their random formulation, can be seen as random correlations. For example the ROT corresponds to $A = \mathbb{F}_q \times \mathbb{F}_q$, $B = \mathbb{F}_2 \times \mathbb{F}_q$ and the relation $R : A \times B \to \{0,1\}$ maps $((x_0,x_1),(b,y))$ to $x_b = y$ (seen as a logic assertion).

We assume that Alice and Bob are given a truly random Beaver Triple (see Definition 2.5.7) of the form $([\![a]\!],[\![b]\!],[\![c]\!] \mid a \cdot b = c)$. We will show that with it, the parties can deduce an additive sharing of xy from $[\![x]\!]$ and $[\![y]\!]$. The protocol is as follows:

- The parties start by computing sharing of $d = a + x$ and $e = b + y$ (this is possible because of the linearity of additive secret sharing).
- The parties send their shares of d and e to other parties in order to recover d and e plainly. Note that this does not reveal anything on x and y, because a and b are supposed to be perfectly random. They, therefore, mask completely the values x and y, with information-theoretic security.
- Note that

$$x \cdot y = -ea - db + ed + ab.$$

— It is to be noted that a, b and $c = ab$ are shared among the players and that this expression of $x \cdot y$ is linear in these values. One can therefore easily deduce an additive sharing of $x \cdot y$.

$$[\![x \cdot y]\!] = -e \cdot [\![a]\!] - d \cdot [\![b]\!] + [\![e \cdot d]\!] + [\![c]\!].$$

Both parties know e and d. They can decide arbitrarily who would add $e \cdot d$ to his share and who would not, before the execution of the protocol.

Therefore, thanks to the additional random correlation (the Beaver triple) we managed to preserve the invariant. For each addition gate, Alice and Bob can simply perform the addition of their values locally, without any communication. For the multiplication gates, each party needs to send $2\log(q)$ bits ($\log(q)$ for each d and e). In the end, the parties recover a sharing of the result and after exchanging their shares with each other, they recover the final result.

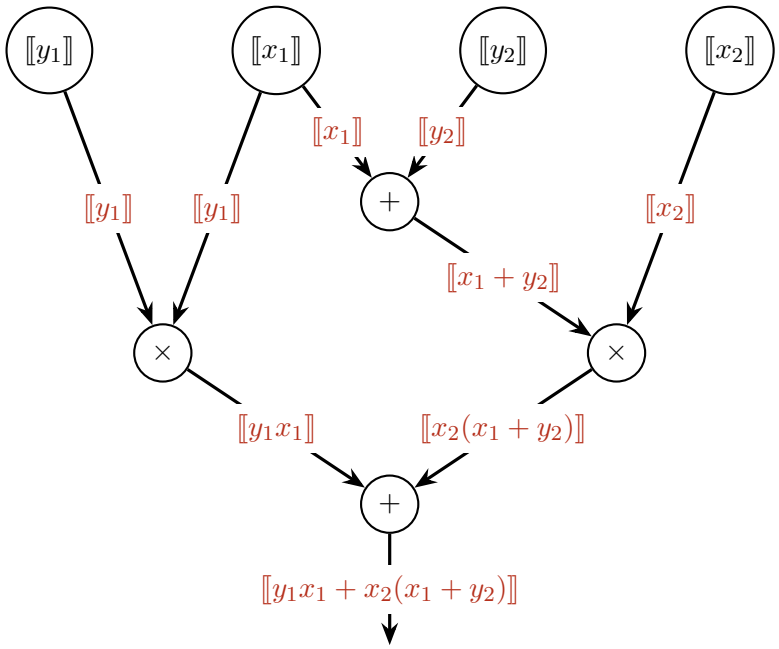

Figure 2.10 – Representation of the GMW algorithm on the arithmetic circuit of $f(x_1, x_2, y_1, y_2) = x_1 y_1 + x_2(x_1 + y_2)$, with (x_1, x_2) being Alice's input, and (y_1, y_2) being the input of Bob. Labels on the edges correspond to what the players compute during the execution.

However, we have not discussed the direct downside of the protocol: for each multiplication gate, *a new fresh Beaver triple has to be used.* If this is not the case, information on the private intermediate computation could be easily obtained. For example, assume that the same beaver triple is used twice, once with inputs (x, y), the second time with inputs (x', y'). Then, players can compute $d_1 - d_0 = (x + a) - (x' + a) = x - x'$ which already reveals information to the players.

2.5.4.2 Efficiency Considerations

This is concerning because, as stated in Chapter 1, the number of multiplication gates in an arithmetic circuit corresponding to classical functions we might wish to compute can easily reach millions, if not billions. For instance, [Cou23, Section 2.4] provides an example: for the edit distance function applied to bit strings of size 4095, the number of AND gates is $\sim 2^{30}$ and the total computa-

tional time is approximately 740 hours on a powerful server [3], and the communication cost is about a terabyte.

Therefore, if the protocol is quite fast in the correlated randomness model, effort needs to be made in order to generate the correlations, as they are natively non practical both in computation complexity and in communication complexity.

2.5.4.3 Solutions?

One can wonder whether we can create a random instance of OT faster than with the protocol presented in Remark 2.5.2. If this is the case to some extent [4], random OT does require public-key cryptography. The problem lies that public-key cryptography, while being impressive and offering beautiful results, is also expensive.

Given the unlikely prospects of significantly reducing the costs of OT, another solution arose: in 2003, Ishai, Kilian, Nissim, and Petrank revived the area of MPC with their groundbreaking result (informally): assuming the existence of an hash function with appropriate security requirements, it is possible to transform λ OTs to an unbounded number n, for a computational cost dominated by only three calls to the hash function per created OT. While the computation remains linear in the number of OTs produced, this reduce massively the computation overhead induced by the computation of exponentiations. The communication costs remains linear in the number of OT produce, and the overhead is only slightly changed.

Looking at the concrete efficiency again, the gain is enormous: looking back at the example of the edit function, the use of hash function entails a reduction of the computation cost from 1000 hours to about 1 minute (see [Cou23, Section 2.5.3.2]). The communication is also slightly reduced, but the gain is very small in comparison to the gain in efficiency: the communication cost is still in the hundreds of gigabytes, remaining highly impractical. This result, if it solved one of the two problems, is not enough.

Pseudorandom Correlations Generator were introduced in 2018 by [BCGI18] as a way to answer to this problem and to get the best of two worlds: efficient computation, with small communication overhead. Their construction is the central subject of this thesis. Chapter 3 will formally introduce the concept and the properties, Chapter 5 offers involved constructions of pseudorandom correlation generators in the special case of the OLE correlation.

2.5.5 Others important techniques in MPC

MPC, since its early start in the 80s, has grown to become a very prolific field, with plenty of techniques and constructions. In this manuscript, we will focus only on the constructions based on the GMW protocol and therefore on secret sharing. Therefore, we will not discuss the following techniques:

2.5.5.1 Garbled-circuit-based Scheme

Protocols based on garbled circuit schemes descend from the seminal work of Yao [Yao82]. The idea behind garbled circuits is as follows: one of the players transforms the protocol to be computed in order to make it *garbled* for the other player. Next, the other player evaluates the garbled circuit without having any information about what is going on, as the circuit is garbled for him. This still requires the parties to exchange a lot of information in comparison to secret-sharing-based-MPC

3. AWS EC2 c5.9xlarge
4. after all, the protocol proposed is quite old, we choose it mostly for its simplicity but better versions exist

schemes but requires only a constant number of rounds. Therefore it can be very useful when the latency is very high and the number of rounds is critical, for example.

2.5.5.2 FHE-based Scheme

FHE stands for *fully homomorphic encryptions*. It was introduced by Gentry in 2009 [Gen09], and is a form of encryption that allows computation to be performed on encrypted data. This is very powerful because the parties just have to send an encrypted version of their secret values to the other party, and perform locally the computation on the encrypted - and therefore unreadable - secret values. This achieves low communication as you only have to send and receive the encryption of your inputs and the result, and a low number of rounds for the same reasons. Nonetheless, the protocol requires advanced and very expensive cryptographic primitives, which results in very large computational costs.

Chapter 3

Efficient Correlated Randomness Generation: PCG and PCF

We have shown, in Section 2.5.4, that using a two-party protocol with additional *random correlation*, the parties can securely compute any function f, represented as an arithmetic circuit. Unfortunately, the amount of random correlations which must be provided to the parties beforehand is as big as the number of multiplication gates in an arithmetic circuit representing f, or equivalently, the number of AND gates in a boolean circuit representing f. Because the number of multiplication gates in such an arithmetic circuit for the functions we consider is easily around $\approx 10^9$, the costs of generating all this amount of random correlation are prohibitively high regarding both computation and communication. A solution was provided by the OT *extension* technique, which greatly reduced the computational costs: players could then obtain the result with only three evaluations of hash function per OLE produced. Nonetheless, the communication complexity remains prohibitive, being in hundreds of gigabytes. Therefore, new solutions have to be designed to tackle the communication problem. The chapter presents the solutions, Pseudorandom Correlation Generator and Pseudorandom Correlation Function. These constructions mainly come from [BCGI18; BCGI+19b; BCGI+20b; BCGI+20a].

Outline of the current chapter

For a general understanding of this chapter, we recommend that the reader review Section 2.5.3 and Section 2.5.4, which serve as motivation.

3.1 Pseudorandom Correlation Generator

3.1.1 Introduction and General Idea

3.1.1.1 The Preprocessing Model

In light of the GMW protocols using Beaver triples, we introduce the *preprocessing model* for computing secure functions. This model consists of two phases:

- The first phase is the *preprocessing phase*, during which the parties create the correlated randomness they will need via multiple calls to OT protocols. This phase does not depend on the function the parties wish to compute, nor on the inputs, and therefore can be conducted ahead of time (hence its name). This phase could be slow.
- The second phase is the *online phase*, during which the parties use the correlated randomness material created during the preprocessing phase, in the GMW protocols. Here, we expect the phase to be fast and non-cryptographic.

The goal of this construction is to reduce the cost of the online phase as much as possible, both in communication and in computation, pushing the majority of the costs into the preprocessing phase. Again, the interest lies in the fact it can be computed ahead of time.

3.1.1.2 About the Importance of *Pseudorandomness*

When dealing with primitives that are difficult to generate randomly, a valuable technique in the cryptographer's toolbox is to use *pseudorandomness*. By relaxing the requirement for real randomness, one can be inspired by some classical constructions in cryptography: the *Pseudorandom Generator* (PRG). We explained in Definition 2.3.3 that a PRG is a powerful primitive that allows a party to generate, from a small amount of real randomness, long strings that appear random. This becomes crucial when pure randomness is hard to obtain. Additionally, get a closer look at the protocol described in Example 2.3.1. The protocol is in the preprocessing model:

- Execute a short protocol to obtain a shared short key k (correlation identity).
- Without further communication, apply the PRG G to the key k to generate a long pseudorandom chain.
- Use this pseudorandom chain $G(k)$ to securely send a message.

This is secure as long as the PRG used is secure. Note that in the end the parties hold the same chain $G(k)$. We can rephrase this fact saying that they manage to obtain a large amount of the identity correlation. Next, we aim to extend this construction to handle general correlations, not just the identity correlation.

3.1.1.3 From PRG to PCG

The core idea behind pseudorandom correlation generator is exactly the one of a PRG: we want to go from *small random correlated seeds* to *long pseudorandom correlated strings*. The requirement for the two strings to remain correlated makes the question hard (otherwise, the parties could just apply a PRG to their seed and obtain a long pseudorandom string, but this would erase the correlation). Contrary to our previous example with the identity correlation, we distinguish the correlation that

exists between the seeds that we give the parties and the correlation that exists between the long pseudorandom correlated strings. The later is the one we will use afterwards in our MPC protocols, and therefore is called the target correlation. More precisely, we search a method to transform λ instances of a correlation C, into $\exp(\lambda)$ instances of the target correlation $\mathsf{C}_{\mathsf{target}}$.

Example 3.1.1. We consider the OLE correlation as target correlation. We want Alice and Bob to hold respectively s_0 and s_1, of size $\mathsf{poly}(\lambda)$, such that a correlation exists between s_0 and s_1, associated to a protocol for them to *expand* locally s_0 and s_1, that is, being an analog of the evaluation by the PRG. It would result in $(u'_i, v'_i)_{1\leqslant i\leqslant n}$ for Alice and $(\Delta'_i, w'_i = \Delta'_i u'_i + v'_i)_{1\leqslant i\leqslant n}$ for Bob, with $\lambda \ll n$. Note that the u'_i, v'_i, Δ'_i are now considered pseudorandom.

Therefore, we want the PCG associated to a target correlation to be a pair of two algorithms (PCG.Gen, PCG.Expand), such that (informally):

- PCG.Gen generates shorts and correlated seeds (k_0, k_1) for the correlation of your choice.
- PCG.Expand takes a short seed k_σ as input and outputs $x_\sigma = \mathsf{PCG.Expand}(k_\sigma)$ this a long pseudorandom string, such that x_0 and x_1 are correlated according to the target correlation

3.1.1.4 How to read this chapter

Next in this chapter, we will formally introduce the PCG primitive as well as PCF. Some necessary high-level concepts and examples for the rests of the thesis are presented in Sections 3.1.3 and 3.2.1. The formal definitions of PCG and PCF are given in Definitions 3.1.3 and 3.2.2 for completeness, but they are not necessary for general understanding.

3.1.2 Pseudorandom Correlation Generators

We recall the notion of pseudorandom correlation generator (PCG) from [BCGI+19b]. At a high level, a PCG for some target ideal correlation takes as input a pair of short, correlated seeds and outputs long correlated pseudorandom strings. The expansion procedure is deterministic and can be applied locally. The definitions below are taken almost verbatim from [BCGI+20b]. Recall the definition of corretaion introduced in Section 2.5.4.1

Definition 3.1.1 (Correlated Randomness and correlation generator)**.** *Consider a 2PC protocol with only two parties, Alice and Bob. Let A, B be sets. Given a binary relation $R(a, b) : A \times B \to \{0, 1\}$, we define the* correlation with respect to R *as the following set*

$$\mathsf{C}_R = \{(a, b) \in A \times B \mid R(a, b) = 1\}.$$

Given a binary relation R, we say we give Alice and Bob additional random correlations *when we give to Alice a and to Bob b such that*

$$(a, b) \xleftarrow{\$} \mathsf{C}_R.$$

A PPT algorithm $\mathcal{C}$ is called a correlation generator *for a correlation C, if $\mathcal{C}$ on input 1^λ outputs a pair of elements a, b in C.*

The security definition of PCGs requires the target correlation to satisfy a technical requirement, which roughly says that it is possible to efficiently sample from the conditional distribution of R_0 given $R_1 = r_1$ and vice versa. It is easy to see that this is true for the correlations considered in this paper.

Definition 3.1.2 (Reverse-sampleable correlation generator)**.** *Let $\mathcal{C}$ be a correlation generator. We say $\mathcal{C}$ is* reverse sampleable *if there exists a PPT algorithm* RSample *such that for $\sigma \in \{0,1\}$ the correlation obtained via:*

$$\{(R'_0, R'_1) \mid (R_0, R_1) \xleftarrow{\$} \mathcal{C}(1^\lambda), R'_\sigma := R_\sigma, R'_{1-\sigma} \xleftarrow{\$} \mathsf{RSample}(\sigma, R_\sigma)\}$$

is computationally indistinguishable from $\mathcal{C}(1^\lambda)$.

Example 3.1.2. For completeness, we give a proof of why OLE is reverse-sampleable. OLE gives the two parties, a sender and a receiver, respectively $R_0 = (u, v) \in \mathbb{F}_q$ and $R_1 = (\Delta, w = u \cdot \Delta + v) \in \mathbb{F}_q$. We show that both in the case of the sender ($\sigma = 0$) or of the receiver ($\sigma = 1$), there exists a PPT algorithm $\mathsf{RSample}(\sigma, R_\sigma)$ that outputs the other part of the correlation $R_{1-\sigma}$ such that (R_0, R_1) looks like a random correlation. Consider the case $\sigma = 0$ (sender). In that case, $\mathsf{RSample}(0, R_0)$ first parses R_0 as (u, v), samples $\Delta \in \mathbb{F}_q$, lets $w = u \cdot \Delta + v$ and outputs (Δ, w). In the case $\sigma = 1$, $\mathsf{RSample}(1, R_1)$ first parses R_1 as (Δ, w), samples $u \in \mathbb{F}_q$, lets $v = w - u \cdot \Delta$ and outputs (u, v).

Definition 3.1.3 (Pseudorandom Correlation Generator)**.** *Let $\mathcal{C}$ be a reverse-sampleable correlation generator. A* pseudorandom correlation generator (PCG) *for $\mathcal{C}$ is a pair of algorithms* (PCG.Gen, PCG.Expand) *with the following syntax:*

- $\mathsf{PCG.Gen}(1^\lambda)$ *is a PPT algorithm that given a security parameter λ, outputs a pair of seeds $(\mathsf{k}_0, \mathsf{k}_1)$;*
- $\mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma)$ *is a polynomial-time algorithm that given a party index $\sigma \in \{0,1\}$ and a seed k_σ, outputs a bit string $R_\sigma \in \{0,1\}^n$.*

The algorithms (PCG.Gen, PCG.Expand) *should satisfy the following:*

- **Correctness.** *The correlation obtained via:*

  $$\{(R_0, R_1) \mid (\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda), R_\sigma \leftarrow \mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma) \text{ for } \sigma \in \{0,1\}\}$$

  is computationally indistinguishable from $\mathcal{C}(1^\lambda)$ (a random correlation of the same size).
- **Security.** *For any $\sigma \in \{0,1\}$, the following two distributions are computationally indistinguishable:*

  $$\{(\mathsf{k}_{1-\sigma}, R_\sigma) \mid (\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda), \quad R_\sigma \leftarrow \mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma)\} \text{ and}$$
  $$\{(\mathsf{k}_{1-\sigma}, R_\sigma) \mid (\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda), \quad R_{1-\sigma} \leftarrow \mathsf{PCG.Expand}(1-\sigma, \mathsf{k}_{1-\sigma}),$$
  $$R_\sigma \xleftarrow{\$} \mathsf{RSample}(1-\sigma, R_{1-\sigma})\}$$

  where RSample *is the reverse sampling algorithm for correlation $\mathcal{C}$ (see Definition 3.1.2).*

Note that PCG.Gen could simply output a sample from $\mathcal{C}$. To avoid this trivial construction, we also require that the seed size is significantly shorter than the output size.

The correctness of the construction is expected: it consists just to be sure that the distribution the correlation induced by the PCG is indistinguishable from the target correlation. As for security, things are a bit more involved: we want that party $1 - \sigma$ does not get more information on R_σ that he could deduce from its own seed and $R_{1-\sigma}$.

Remark 3.1.1 (On reverse-sampleable correlation generator)**.** The notion of reverse-sampleable correlation may seem a bit far-fetched and we might hope for a simpler notion of security. Let Π be a

protocol in the correlated randomness model, we define Π' the protocol which replaces the random correlation in Π with the expanded seeds the parties got from the PCG. A simpler notion of security would be "Π secure in the preprocessing model" imply that "Π' is secure". Unfortunately, this is not sufficient, and there exists secure protocols in the correlated randomness model, that are not secure when the true correlated randomness is replaced by expanded seeds (see [BCGI+19b]).

The authors of [BCGI18; BCGI+19b] introduced a new model called the *corruptible correlated randomness model*. This model is a relaxation of the previous one, as it allows the adversary to decide its part of the correlation, while the additional inputs provided to the honest parties are sampled randomly based on their choices. In this context, the concept of reverse-sampleable correlation becomes more intuitive: we require that no additional information can be inferred from the seed provided by the PCG beyond what it inherently conveys. There are two primary reasons for adopting this model:

— It is easier to achieve.

— The authors demonstrate that if a protocol Π is secure in the *corruptible correlated randomness model*, it can be substituted with a protocol Π' where the correlated pseudorandomness is generated through the expansion of short seeds provided by the PCG, while still maintaining security.

3.1.3 Silent Preprocessing Model

All these constructions motivate the idea of the *silent preprocessing model*. This model can be summarized is three different parts described below:

1. The first phase involves the parties generating the short correlated seeds. This should not be too costly both in computation and communication complexity because it concerns only small seeds.
2. The second phase is the phase of *silent expansion*, or *silent computation*, meaning that the players can expand their correlated seeds locally, without communication.
3. Finally, the players use the thus created correlated randomness in a GMW-like protocol, with no cryptography involved.

Figure 3.1 provides the template for this model.

Remark 3.1.2 (Preprocessing phase: another advantage)**.** It is important to understand that the first step is a preprocessing step, and as such, it can be done beforehand, even before knowing the value of x, or the function f. While this was already the case when designing protocols using additional random correlation given by the parties, this construction using PCG offers us an additional property: parties don't have to evaluate the seeds immediately when they receive them. Therefore, the seeds act as a compressed version of the pseudorandom randomness they will use in the future, and the parties don't have to store billions of elements directly, which would be impractical.

3.1.3.1 On the Importance of Function Secret Sharing and a Taste of PCG Construction

In this section we give a taste of how PCGs are constructed. More information about the construction of PCG can be found in Chapter 5, where it will be deeply studied in the case of the OLE correlation.

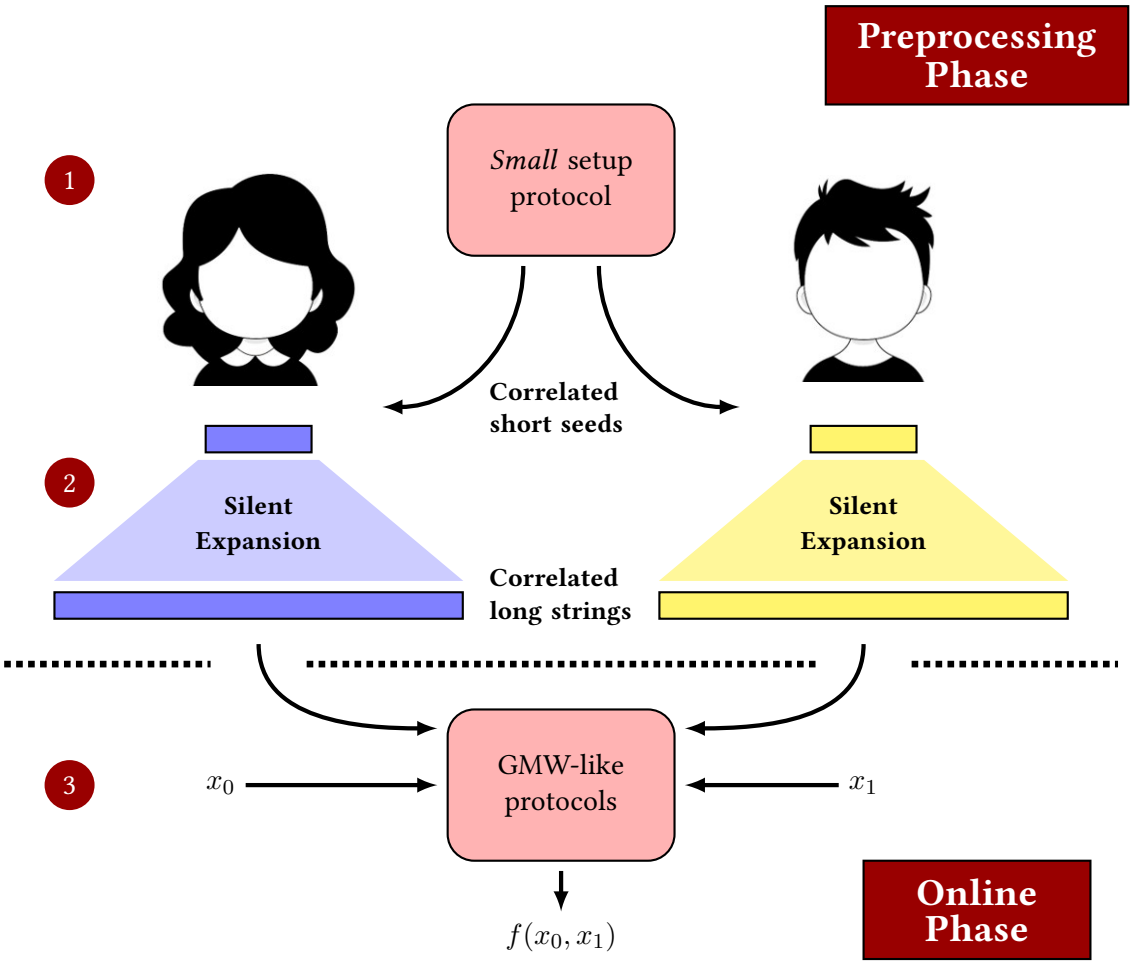

Figure 3.1 – Representation of the silent preprocessing model with: (1) the setup protocol, (2) the silent expansion, (3) use a fast online protocol using correlated pseudorandomness.

PCGs are built on the very important *Function Secret Sharing* (FSS) primitive (see Section 2.5.3.5), and more especially DPF (Definition 2.5.9). The DPF functionality is a very powerful primitive that allows us to share point functions, and more generally multi-point functions for SPFSS (sum of point-function). But obtaining a sharing of a multi-point function is the same as obtaining a sharing of a long but sparse vector: consider the vector of the evaluation of the function. Therefore FSS enables the parties to share long but sparse vectors, from small seeds. That is Alice holds $\mathbf{s_0} \in \mathbb{F}_q^n$ and Bob holds $\mathbf{s_1} \in \mathbb{F}_q^n$ both pseudorandom vectors vectors, such that $\mathbf{e} = \mathbf{s_0} + \mathbf{s_1}$ is a t-sparse vector in $\mathcal{S}(\mathbb{F}_q^n, t)$ (see Definition 2.4.5).

Consider a public parity-check matrix $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$, and two sparse elements $\mathbf{e_0}, \mathbf{e_1} \in \mathcal{S}(\mathbb{F}_q^n, t)$. The Syndrome Decoding assumption guarantees that both $\mathbf{x_0} = \mathbf{He_0}$ and $\mathbf{x_1} = \mathbf{He_1}$ are pseudorandom. $\mathbf{x_0}$ and $\mathbf{x_1}$ have therefore a hidden sparse structure. Let $e : \mathbb{F}_q^{n-k} \times \mathbb{F}_q^{n-k} \to \mathbb{G}$ be a bilinear map, with $\mathbb{G}$ a group. Consider the correlation of the form

$$\mathtt{C}_e = \{((\mathbf{x_0}, s_0)(\mathbf{x_1}, s_1)) \mid \mathbf{x_0}, \mathbf{x_1} \xleftarrow{\$} \mathbb{F}_q^{n-k}, s_0 \xleftarrow{\$} \mathbb{G}, s_1 = e(\mathbf{x_0}, \mathbf{x_1}) - s_0\}.$$

Note for example that OLE is a correlation that satisfy this form, with $\mathbb{G} = \mathbb{F}_q$, and the bilinear map being $e(\mathbf{x_0}, \mathbf{x_1}) = \langle \mathbf{x_0}, \mathbf{x_1} \rangle$.

Let us focus on the $[\![e(\mathbf{x_0}, \mathbf{x_1})]\!]$. Because e is a bilinear function, it exists a function L such that

$$e(\mathbf{x_0}, \mathbf{x_1}) = L(\mathbf{x_0} \cdot \mathbf{x_1}^\intercal),$$

where L is linear in all the entries of the matrix $\mathbf{x_0} \cdot \mathbf{x_1}^\intercal$. Further, we obtain that

$$[\![e(\mathbf{x_0}, \mathbf{x_1})]\!] = [\![L(\mathbf{x_0} \cdot \mathbf{x_1}^\intercal)]\!] = L([\![\mathbf{x_0} \cdot \mathbf{x_1}^\intercal]\!]),$$

where the last equality stand because additive sharing commute with linear map.

$$L([\![\mathbf{x_0} \cdot \mathbf{x_1}^\intercal]\!]) = L([\![\mathbf{H}\mathbf{e_0}\mathbf{e_1}^\intercal\mathbf{H}_1^\intercal]\!]).$$

Using again the linearity of additive sharing:

$$[\![e(\mathbf{x_0}, \mathbf{x_1})]\!] = L(\mathbf{H}[\![\mathbf{e_0}\mathbf{e_1}^\intercal]\!]\mathbf{H}_1^\intercal).$$

And this can be rewritten as:

$$[\![e(\mathbf{x_0}, \mathbf{x_1})]\!] = L^*([\![\mathbf{e_0}\mathbf{e_1}^\intercal]\!]),$$

for L^* the linear function that multiply by $\mathbf{H}_0$ and $\mathbf{H}_1$ respectively on the left and right, and then apply L. What we have shown is that constructing the correlation boils down to obtaining an additive sharing of $\mathbf{e_0}\mathbf{e_1}^\intercal$. Remember that $\mathbf{e_0}, \mathbf{e_1}$ are t-sparse vectors. Therefore, $\mathbf{e_0}\mathbf{e_1}^\intercal$ defines a matrix with at most t^2 non-zero position, that we can share using FSS. Section 5.2 will tackle this general technique in the case of the OLE correlation, underlying its limitations in the case of an unstructured matrix.

3.2 Pseudorandom Correlation Function

Taking a step back, while the construction using PCG is appealing, one downside emerges: Alice and Bob must generate all of the correlated pseudorandomness all at once. They cannot just create a tailored amount of the correlated pseudorandomness for one small computation, and then create more another day, etc. This can be not convenient, in addition to the fact that when the full vector of correlated randomness has been consumed, Alice and Bob have to perform a PCG.Gen protocol again. Like PRF is a natural generalization of PRG, we naturally introduce the notion of *Pseudorandom Correlation Function* (PCF): it allows for generating correlation pseudorandomness incrementally.

Informally, the goal of a pseudorandom correlation function is similar to that of a PCG: after a short interaction for setup, the parties can generate correlation on the fly. Specifically, we want a PCF to be a pair (PCF.Gen,PCF.Eval) of algorithms such that:

- PCG.Gen outputs a pairs of key k_0, k_1;
- PCG.Eval(k_σ, x) is a deterministic polynomial-time algorithm;
- The joint distribution of outputs of PCG.Eval is indistinguishable from the outputs of a random instance realizing the correlation;
- The correlated key k_σ does not help to distinguish PCG.Eval$(k_{1-\sigma}, x)$ from a random instance of the correlation more than what is implied b PCG.Eval(k_σ, x).

We formally define below PCF, in a similar way as we did for PCG. The definitions are taken almost verbatim from [BCGI+20a]. Again the formal definition are very verbose, and it just essentially captures the point written listed above. Reverse-sampleable correlation generator has to be used again, but the definition is a bit different because adapted to the specificities of PCF.

Definition 3.2.1 (Reverse-sampleable correlation, modified)**.** *Let $1 \leqslant \ell_0(\lambda), \ell_1(\lambda) \leqslant \mathsf{poly}(\lambda)$ be two functions. ℓ_0, ℓ_1 denote the length of the output correlation. Let $\mathcal{Y}$ be a probabilistic algorithm that on input 1^λ returns a pair of outputs $(y_0, y_1) \in \{0,1\}^{\ell_0(\lambda)} \times \{0,1\}^{\ell_1(\lambda)}$, defining a correlation on the outputs. We say that $\mathcal{Y}$ defines a* reverse-sampleable correlation*, if there exists a probabilistic polynomial time algorithm* RSample *that takes as input $1^\lambda, \sigma \in \{0,1\}$ and $y_\sigma \in \{0,1\}^{\ell_\sigma(\lambda)}$, and outputs $y_{1-\sigma} \in \{0,1\}^{\ell_{1-\sigma}(\lambda)}$, such that for all $\sigma \in \{0,1\}$ the following distributions are statistically close:*

$$\{(y_0, y_1) \mid (y_0, y_1) \xleftarrow{\$} \mathcal{Y}(1^\lambda)\} \quad \textit{and}$$
$$\{(y_0, y_1) \mid (y'_0, y'_1) \xleftarrow{\$} \mathcal{Y}(1^\lambda), y_\sigma \leftarrow y'_\sigma, y_{1-\sigma} \leftarrow \mathsf{RSample}(1^\lambda, \sigma, y_\sigma)\}.$$

This notion of reverse-sampleable correlation is the same as before, except for the fact that the size of the output correlation can depend on the security parameter λ.

Definition 3.2.2 (Pseudorandom correlation function (PCF))**.** *Let $\mathcal{Y}$ be a reverse-sampleable correlation with output length $\ell_0(\lambda), \ell_1(\lambda)$ and let $\lambda \leqslant n(\lambda) \leqslant \mathsf{poly}(\lambda)$ be an input length function. Let $(\mathsf{PCF.Gen}, \mathsf{PCF.Eval})$ be a pair of algorithms with the following syntax:*

— $\mathsf{PCF.Gen}(1^\lambda)$ *is a probabilistic polynomial time algorithm that on input 1^λ, outputs a pair of keys $(\mathsf{k}_0, \mathsf{k}_1)$; we assume that λ can be inferred from the keys.*

— $\mathsf{PCF.Eval}(\sigma, \mathsf{k}_\sigma, x)$ *is a deterministic polynomial-time algorithm that on input $\sigma \in \{0,1\}$, key k_σ and input value $x \in \{0,1\}^{n(\lambda)}$, outputs a value $y_\sigma \in \{0,1\}^{\ell_\sigma(\lambda)}$* [1].

$\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,0}(\lambda)$:
1: **foreach** $i = 1$ to $N(\lambda)$.
  1.1: $x^{(i)} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$
  1.2: $(y_0^{(i)}, y_1^{(i)}) \leftarrow \mathcal{Y}(1^\lambda)$
2: $b \leftarrow \mathcal{A}(1^\lambda, (x^{(i)}, y_0^{(i)}, y_1^{(i)})_{i \in [N(\lambda)]})$
3: return b

$\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,1}(\lambda)$:
1: $(\mathsf{k}_0, \mathsf{k}_1) \leftarrow \mathsf{PCF.Gen}(1^\lambda)$
2: **foreach** $i = 1$ to $N(\lambda)$.
  2.1: $x^{(i)} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$
  2.2: **foreach** $\sigma \in \{0,1\}$:
    2.2.1: $y_\sigma^{(i)} \leftarrow \mathsf{PCF.Eval}(\sigma, \mathsf{k}_\sigma, x^{(i)})$
3: $b \leftarrow \mathcal{A}(1^\lambda, (x^{(i)}, y_0^{(i)}, y_1^{(i)})_{i \in [N(\lambda)]})$
4: return b

Figure 3.2 – Pseudorandom $\mathcal{Y}$-correlated outputs of a PCF.

1. Note that it would be sufficient for PCF.Eval to take as input k_σ and x by appending σ to the key k_σ. This corresponds to the view of a PCF as a single keyed function.

$\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma,0}(\lambda)$:

1: $(\mathsf{k}_0, \mathsf{k}_1) \leftarrow \mathsf{PCF.Gen}(1^\lambda)$

2: **foreach** $i = 1$ to $N(\lambda)$.

2.1: $x^{(i)} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$

2.2: $y^{(i)}_{1-\sigma} \leftarrow \mathsf{PCF.Eval}(1-\sigma, \mathsf{k}_{1-\sigma}, x^{(i)})$

3: $b \leftarrow \mathcal{A}(1^\lambda, \sigma, k_\sigma, (x^{(i)}, y^{(i)}_{1-\sigma})_{i\in[N(\lambda)]})$

4: return b

$\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma,1}(\lambda)$:

1: $(\mathsf{k}_0, \mathsf{k}_1) \leftarrow \mathsf{PCF.Gen}(1^\lambda)$

2: **foreach** $i = 1$ to $N(\lambda)$.

2.1: $x^{(i)} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$

2.2: $y^{(i)}_{\sigma} \leftarrow \mathsf{PCF.Eval}(\sigma, \mathsf{k}_{\sigma}, x^{(i)})$

2.3: $y^{(i)}_{1-\sigma} \leftarrow \mathsf{RSample}(1^\lambda, \sigma, y^{(i)}_{\sigma})$

3: $b \leftarrow \mathcal{A}(1^\lambda, \sigma, k_\sigma, (x^{(i)}, y^{(i)}_{1-\sigma})_{i\in[N(\lambda)]})$

4: return b

Figure 3.3 – Security of a PCF. Here, RSample is the algorithm for reverse sampling $\mathcal{Y}$ as in Definition 3.2.1.

We say $(\mathsf{PCF.Gen}, \mathsf{PCF.Eval})$ *is a (weak)* (N, B, ε)-secure pseudorandom correlation function (PCF) for $\mathcal{Y}$, *if the following conditions hold:*

— ***Pseudorandom*** $\mathcal{Y}$***-correlated outputs.*** *For every* $\sigma \in \{0,1\}$ *and non-uniform adversary* $\mathcal{A}$ *of size* $B(\lambda)$, *it holds*

$$\left|\Pr\left[\,\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,0}(\lambda) = 1\,\right] - \Pr\left[\,\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,1}(\lambda) = 1\,\right]\right| \leqslant \varepsilon(\lambda)$$

for all sufficiently large λ, *where* $\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,b}(\lambda)$ *for* $b \in \{0,1\}$ *is as defined in Figure 3.2. In particular, the adversary is given access to* $N(\lambda)$ *samples.*

— ***Security.*** *For each* $\sigma \in \{0,1\}$ *and non-uniform adversary* $\mathcal{A}$ *of size* $B(\lambda)$, *it holds*

$$\left|\Pr\left[\,\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma,0}(\lambda) = 1\,\right] - \Pr\left[\,\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma,1}(\lambda) = 1\,\right]\right| \leqslant \varepsilon(\lambda)$$

for all sufficiently large λ, *where* $\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma,b}(\lambda)$ *for* $b \in \{0,1\}$ *is as defined in Figure 3.3 (again, with* $N(\lambda)$ *samples).*

We say that $(\mathsf{PCF.Gen}, \mathsf{PCF.Eval})$ *is a* PCF for $\mathcal{Y}$ *if it is a* $(p, 1/p, p)$*-secure PCF for* $\mathcal{Y}$ *for every polynomial* p. *If* $B = N$, *we will write* (B, ε)*-secure PCF for short.*

Note that PCF are defined above like *weak* pseudorandom function, where security is only required to hold given random adversarial queries. As for PRFs, one can also strengthen the definition to strong PCFs, which allow arbitrary adversarial queries; a formal definition is given in [BCGI+20a]. As shown in [BCGI+20a], any weak PCF can be turned into a strong PCF in the random oracle model, by hashing the input before feeding it to the function.

3.2.1 How to Construct a PCF?

In this section, we explain how the association of FSS and a WPRF can be used to create a PCF. The construction exposed here follows the work of [BCGI+20a, Section 5, 6].

3.2.1.1 Function Secret Sharing for Weak Pseudorandom Function.

The construction will make use of FSS for Weak Pseudorandom Function.

Definition 3.2.3 (FSS with Weak Pseudorandom Outputs)**.** *We say that an* FSS *scheme (*FSS.Gen, FSS.Eval*) for f has weak pseudorandom outputs if for any $\sigma \in \{0,1\}$, $N = \mathsf{poly}(\lambda)$, the distribution*

$$\left\{(x^{(i)}, y^{(i)})_{i=1}^{N} \mid (K_0, K_1) \xleftarrow{\$} \mathsf{FSS.Gen}(1^{\lambda}, f_\lambda), x^{(i)} \xleftarrow{\$} \{0,1\}^n, y^{(i)} \xleftarrow{\$} \mathsf{FSS.Eval}(\sigma, K_\sigma, x^{(i)})\right\}_{\lambda \in \mathbb{N}}$$

is indistinguishable from the uniform distribution on $(\{0,1\}^n \times \mathbb{G})^N$.

We will say that such a WPRF is then FSS-*friendly* if there exists an FSS scheme for the class $\{f_k(\cdot)\}$. The state-of-the-art on FSS is reduce: it tackles the class of multi-point functions, and also step functions. Because the state-of-the-art on FSS provides FSS schemes for multi-point functions, the choice of FSS-friendly WPRF is quite reduced. We will discuss the difficulty of finding such a WPRF and give a construction in Section 7.2.

3.2.2 A Construction Example for the VOLE Correlation

[BCGI+20a] provides different construction - all in the same flavor - constructing PCF with different correlations (OT, VOLE, Beaver Triples, see Figures 2.3, 2.5 and 2.7). In this manuscript, we will present only the construction tackling the VOLE correlations (see Definition 2.5.6). We recall that the VOLE correlation is a vector version of the OLE correlation: it gives $(\mathbf{u}, \mathbf{v})$ to the sender and $(\Delta, \mathbf{w} = \Delta\mathbf{u} + \mathbf{v})$ to the receiver. In the context of a PCF, it can be seen as an OLE correlation with fixed Δ: the sender receives for each evaluation of the PCF (u, v) and the receiver receives $(\Delta, w = \Delta u + v)$, for new u, v, w each time but the same Δ.

Theorem 3.2.1 ([BCGI+20a, Theorem 5.3])**.** *Let $\mathcal{R} = \mathcal{R}(\lambda)$ be a finite commutative ring. Suppose there exists an* FSS *scheme for scalar multiples of a family of weak pseudorandom functions $\mathcal{F} := \{f_k : \{0,1\}^n \to \mathcal{R}\}_{k \in \{0,1\}^\lambda}$. Then, there exists a* PCF *for the* VOLE *correlation over $\mathcal{R}$, given by the construction in Figure 3.4.*

Proof. We will only sketch the proof and invite the interested reader to read the proof in the original paper.

First, check that the protocol is *correct*: the sender obtains (u, v) and the receiver obtains (Δ, w) on input Δ, as required. Therefore, this is indeed an OLE correlation with fixed Δ. As for the security requirements, we can use a sequence of hybrid distributions. The security requirements ask to prove that the result of $\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma,0}(\lambda)$ and $\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma,1}(\lambda)$ (defined in Figure 3.2) are computationally indistinguishable. There are two cases, $\sigma = 0$ and $\sigma = 1$.

When $\sigma = 0$, this is asking to prove that $(k_0, (x^{(i)}, (\Delta^i, w^{(i)})))$ for Δ^i, w^i issued by the PCF, is indistinguishable from $(k_0, (x^{(i)}, (\Delta'^i, w'^{(i)})))$, where Δ' and w' are reverse-sampled using (u^i, v^i). Briefly, this can be done by replacing the key K_0 generated by the FSS scheme by a fresh key $\tilde{K_0}$(using the security of FSS), and then by replacing all the u by new random elements (using this time the security of the WPRF).

For $\sigma = 1$, it is even simpler because it follows directly from the security of the FSS scheme.

Additionally, this defines a proper VOLE correlation on the long fly: N distinct calls to the PCF are indistinguishable from a VOLE correlation with vectors of size N. We first replace v with $w - \Delta u$, which is still indistinguishable because of the FSS correctness property. Next, we replace each w_i with a random element of $\mathcal{R}$, which cannot be distinguished because of the security of the FSS. We can finally replace u with a random element because of the security of WPRF. □

This is again constructed over FSS: The FSS security ensures the pseudorandomness of the w and v elements, whereas the WPRF ensures the pseudorandomness of the element u. Note that an FSS-friendly WPRF is therefore required. This will be the subject of the last chapter of this thesis, Chapter 7.

PCF for Vector Oblivious Linear Evaluation

PARAMETERS:
$\mathcal{F} = \{f_k : \{0,1\}^n \to \mathcal{R}\}_{k\in\{0,1\}^\lambda}$ be a weak PRF, FSS = (FSS.Gen, FSS.Eval) a Function Secret Sharing scheme for $\{cf_k\}_{c\in\mathcal{R},k\in\{0,1\}^\lambda}$, with weak pseudorandom outputs.

$\mathsf{PCF.Gen}(1^\lambda)$:

1: Sample $k \xleftarrow{\$} \{0,1\}^\lambda$ and $\Delta \xleftarrow{\$} \mathcal{R}$.
2: Sample FSS keys $(K_0, K_1) \xleftarrow{\$} \mathsf{FSS.Gen}(1^\lambda, \Delta f_k)$
3: Output the keys $\mathsf{k}_0 = (K_0, \Delta)$ and $\mathsf{k}_1 = (K_1, k)$.

$\mathsf{PCG.Eval}(\sigma, \mathsf{k}_\sigma, x)$:

1: if $\sigma = 0$,
  1.1: Let $w = \mathsf{FSS.Eval}(0, K_0, x)$
  1.2: Output (Δ, w).
2: if $\sigma = 1$,
  2.1: Let $v = -\mathsf{FSS.Eval}(1, K_1, x)$
  2.2: Let $u = f_k(x)$
  2.3: Output (u, v)

Figure 3.4 – PCF for VOLE over a the ring $\mathcal{R}$ based on FSS for scalar multiples

3.3 *Programmable* PCGs/PCFs

We now introduce an important property that PCG and PCF can enjoy, and which has some interesting impact on our constructions. The PCG protocols for OLE presented in Chapter 5 will verify this property, and it is one of the reasons for their study.

At a high level, a *programmable* PCG allows the generation of multiple PCG keys such that part of the correlation generated remains the same across different instances. Consider the following example: imagine Alice, Bob, and Charlie want to perform computations together, equipped with a programmable PCG for OLE between two parties. Alice can decide to execute the PCG protocol with Bob and produce numerous pairs $(u, [\![\Delta \cdot u]\!])$ and $(\Delta, [\![\Delta \cdot u]\!])$, respectively for Alice and Bob. Alice can then execute the PCG protocol a second time, but with Charlie this time, and produce numerous pairs $(u, [\![\Gamma \cdot u]\!])$ and $(\Gamma, [\![\Gamma \cdot u]\!])$. Note that the value u is deliberately the same in both PCG calls: Alice can *program* parts of the output of the correlation to be u. Here, the terminology can be misleading, but the meaning is that Alice can fix part of her output to be equal to the result she obtained with Bob.

Programmable PCGs are necessary to construct n-party correlated pseudorandomness from the 2-party correlated pseudorandomness generated via the PCG. Informally, this is because when expanding n-party shares (e.g. of Beaver triples) into a sum of 2-party shares, the sum will involve many "cross terms"; using programmable PCGs allows maintaining consistent pseudorandom values across these cross terms. This will be the object of Section 3.3.1.

The formal definition of a programmable PCG is given in Appendix A.

Remark 3.3.1 (On random correlations and programmability)**.** Programmable PCG on random correlations may seem a little paradoxical. Indeed, if we assumed at the beginning that the parties were given random instances of the target correlation, they can now fix some of the inputs they will receive. For example, in the case OLE (giving some $(u, [\![\Delta \cdot u]\!]_0)$ to Alice and $(\Delta, [\![\Delta \cdot u]\!]_1)$ to Bob), a programmable PCG can be used to fix the values u and Δ. Note that when a party decides to fix the value of some parts of its correlation, it looks like we are just creating a PCG for a variant of the original correlation, where some inputs are fixed. Nevertheless, it is a bit different because the value that is fixed has to have a small description and therefore cannot be simply sampled randomly beforehand.

3.3.1 Application 1: (N-party) Multiplication Triples Generation for Arithmetic Circuit

Theorem 3.3.1. *Assume the existence of a programmable* PCG $\mathcal{P}$ *for* OLE *between two parties. Let denote its running time to produce* T OLE*s instances by* $\mathcal{P}(T)$*. Then there exists a semi-honest* N*-party protocol for securely evaluating an arithmetic circuit* C *over* $\mathbb{F}_q$ *with* T *multiplication gates, in the preprocessing models, such that:*

- *During the preprocessing phase, the cost is* $N(N-1)\mathcal{P}(T)$*.*
- *In the online phase, which is non-cryptographic, the cost of communication is* $2 \cdot N \cdot T$ *elements of* $\mathbb{F}_q$*.*

Proof. Consider the parties $P_1, \ldots, P_N$. Remember that using two random OLEs, one can construct a Beaver triple (see Lemma 2.5.3). We use programmability to obtain N-party multiplication triples via the following steps:

1. Each party P_i gets two random values (x_i, y_i). We define $X = \sum_i x_i$ and $Y = \sum_j y_j$.

2. Each pair of parties $(P_i, P_j)_{1 \leqslant i,j \leqslant N, i \neq j}$ performs the programmable protocol for (2-party) OLE with programmable inputs (x_i, y_j), and obtains shares of $x_i \cdot y_j$. We denote the share of P_i as $[\![x_i \cdot y_j]\!]_i$.
3. Let $K_i = \sum_{j=1}^{N} [\![x_i \cdot y_j]\!]_i + [\![x_j \cdot y_i]\!]_i + x_i \cdot y_i$. The K_i are shares of the product:

$$X \cdot Y = \sum_{1 \leqslant i,j \leqslant N} x_i \cdot y_j = \sum_{i=1}^{N} K_i.$$

Note that the importance of programmability arise here at step 2: we want to obtain the additive sharing of all the possible pairs $(x_i y_j)$, with $1 \leqslant i, j \leqslant N$. Therefore, the party i wants to fix its value x_i in each of the PCG protocols he executes to compute the additive sharing of $x_i y_j$ for all the y_j, $j \neq i$. The parties use the programmable PCG $\mathcal{P}$ to generate short seeds for each of the $N \cdot (N-1)$ (2-party) OLE they need. In the online phase, they locally expand the seeds to obtain T instances of (N-party) multiplication triples. The parties can execute the (N-party) GMW protocol using the multiplication triples and evaluate the circuit.

The cost of communication in the online phase is derived from the GMW algorithm using the multiplication triples. For each multiplication gate, each party must send two field elements, resulting in a cost of $2 \cdot N \cdot T$. □

3.3.2 Application 2: Secure Computation with Circuit-Dependent Preprocessing

We discuss now another important application of programmable PCG: circuit-dependent preprocessing is a variation of the standard Beaver's circuit randomization technique with multiplication triples. It has been investigated in recent works, such as [DNNR17; Cou19]. The idea is to preprocess multiplications in a way that depends on the structure of the circuit and leads to an online phase that requires just *one opening per multiplication gate*, instead of two when using multiplication triples. PCGs for OLEs do not directly enable reducing the preprocessing phase of secure computation with circuit-dependent correlated randomness: at a high level, this stems from the fact that since the correlated randomness depends on the topology of the circuit, it cannot be compressed beyond the description size of this topology. Nevertheless, PCGs enable *batch* secure computation (*i.e.* securely computing many copies of the same circuit on different input) with silent preprocessing in the circuit-dependent correlated randomness setting, by using PCGs to compress a batch of correlations for a given gate across all circuits.

Theorem 3.3.2. *Assume the existence of oblivious transfer and a programmable* PCG $\mathcal{P}$ *for* OLE *between two parties. Let us denote its running time to produce T* OLE*s instance by $\mathcal{P}(T)$. There exists a semi-honest 2-party protocol for securely evaluating T copies of an arithmetic circuit C over $\mathbb{F}$ with S multiplication gates, in the preprocessing model, such that:*

— *During the preprocessing phase, the computational cost is $2S\mathcal{P}(T)$.*

— *The online phase is non-cryptographic and has communication cost of $2 \cdot S \cdot T$ elements of $\mathbb{F}$.*

Proof. Let C be an arithmetic circuit over $\mathbb{F}$ consisting of fan-in two addition and multiplication gates. Each wire w is assigned a mask r_w during the offline phase. The masks are designed as follows:

— If w is an input wire, r_w is chosen at random;

— If w is the output wire of a multiplication gate, $r_w \in \mathbb{F}$ is chosen at random;

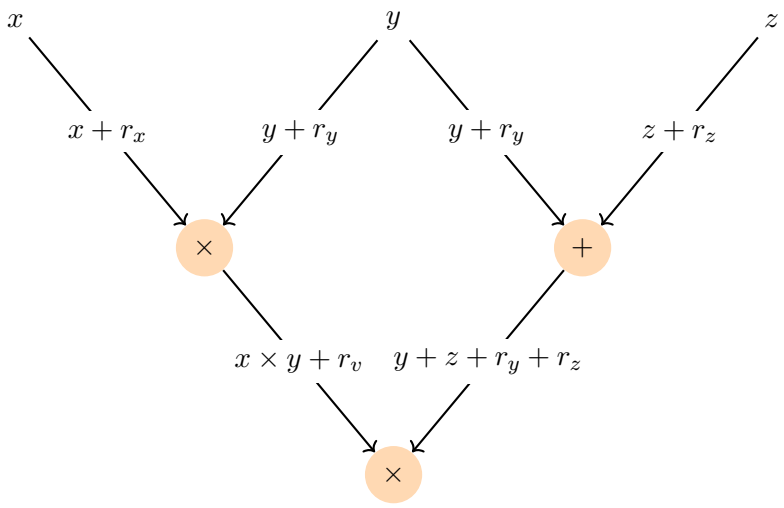

Figure 3.5 – Representation of the arithmetic circuit with its label

- If w is the output wire of an addition gate with input wires u and v, then $r_w := r_u + r_v$;
- For each multiplication gate, we assign a value $s_{u,v}$, such that on input wires u and v, $s_{u,v} := r_u \cdot r_v$.

The masks are not known by the parties, but they obtain random additive shares of each r_w for any input and output wire of multiplication gates, as well as $s_{u,v}$ for the multiplication gates.

When the online phase begins, both parties hide their secret values with random masks. Before the protocol starts, a party that has an input x sample a random r_x, and send $x + r_x$ to the other party, along with a share of r_x. The invariant of the online phase is that through the protocol, for each wire, parties know exactly the value $x + r_x$, where r_x is the mask of this wire (for which parties have additive sharing), and x is the real value that is computed by the circuit passing through the wire w. The invariant is preserved through each gate because of the following:

- For an addition gate, parties know $x + r_x$ and $y + r_y$. Then the parties add locally those values to obtain $x + y + r_x + r_y$, with $r_x + r_y$ being indeed the output mask for the addition gate.
- For a multiplication gate with r_w denoting its output wire's mask, parties know $x + r_x$ and $y + r_y$. The parties can locally compute their share $[\![(x + r_x) \cdot r_y + (y + r_y) \cdot r_x + r_x \cdot r_y + r_w]\!]$ (the formula can be a little bit different if we are not over $\mathbb{F}_2$). By exchanging one bit of information, they reconstitute this value. Adding up $(x + r_x) \cdot (y + r_y)$, they obtain in clear $x \cdot y + r_w$ where r_w is the mask of the output wire of this multiplication gate.

In the end, we have to perform $2S$ different calls to $\mathcal{P}$ to create the (2-party) multiplication triples seeds. In the online phase, we gain a factor 2 in communication because each party only has to send a bit of information for each of the multiplication gates. Since there are $S \cdot T$ multiplication gates in total, the communication cost in the online phase is $2 \cdot S \cdot T$. □

Chapter 4

The *linear test* framework

The linear test framework was formally introduced in 2020 in [BCGI+20a], but its core idea was folklore for a long time. This framework analyzes the decision Syndrome Decoding assumption, which tackles the difficulty of distinguishing $(\mathbf{H}, \mathbf{y})$ from $(\mathbf{H}, \mathbf{He})$, where $\mathbf{y} \xleftarrow{\$} \mathbb{F}_q^{n-k}$ and $\mathbf{e} \xleftarrow{\$} \mathcal{S}(\mathbb{F}_q^n, t)$. The main takeaway is that the majority of attacks on search Syndrome Decoding — for example, information set decoding or statistical decoding — perform only linear operations on the syndrome, these operations depending solely on the matrix $\mathbf{H}$. By defining a general framework that abstracts this notion, we can prove general lower bounds on the time complexity of all attacks falling into that category. This is therefore an important tool in analyzing the syndrome decoding assumption and is central to the analysis of our construction. In this chapter, we recall the definition of the linear test framework, its purpose, and some important results that we use. We analyze the details of the constructions and provide insights on how to demonstrate that an attack fits into the framework.

Outline of the current chapter

4.1 Introduction

We showed in Chapter 3 how PCFs and PCGs can be built using the syndrome decoding assumption (defined in Definition 2.4.7). Syndrome decoding is a very common hardness assumption in cryptography. It was introduced by McEliece in 1978 [McE78] and proved NP-complete by [BMT78]. Nowadays, many frameworks use codes and the syndrome decoding assumption for efficient protocols, and several of the accepted schemes in the fourth round of NIST's call for quantum-resistant primitives rely on it ([AABB+22b; AABB+22a; ABCC+22]). Syndrome decoding is also equivalent to the *Learning Parity with Noise* (LPN) assumption when the number of samples is fixed (see Remark 2.4.4), which is another widely used assumption in cryptography, notably introduced by [BFKL94]. A variant of SD was also studied in the context of learning theory and random constraint satisfaction problems [FGKP09; Fei02].

Syndrome Decoding is therefore part of the landscape of computer science and especially cryptography for a long time. Needless to say, the assumption as been gone over with a fine-tooth comb, and many attacks have been created since the sixties. Among those attacks, we can identify:

- BKW-style of attacks [BKW00; Lyu05; LF06; EKM17].
- Covering-codes attacks [ZJW16; BV16; BTV15; GJL20].
- Information set decoding attacks [Pra62; Ste89; FS09; BLP11; MMT11; BJMM12; MO15a; EKM17; BM18].
- Statistical decoding attacks [Al 01; FKI07; Ove06a; DT17a; CDMT22b].
- Birthday-based attacks [Wag02; Kir11].
- Linearization attacks [BM97; Saa07; AG11].
- Attacks based on finding correlations with low-degree polynomials [ABGK+14; BR17].

There exist other attacks that are not listed, but this offers already a good overview of the landscape.

Considering the above, proving that a given variant of syndrome decoding is secure may appear daunting, as one must consider various attack techniques. Fortunately, there is a trick: all the attacks presented above share a common feature. Indeed, these attacks consist of different types of linear operations on the syndrome, with these linear operations depending only on the public matrix $\mathbf{H}$. Therefore, we can encompass all the attacks above into one common framework, denoted as the *linear test framework*. This concept was more or less folklore for a long time before it was first remarked by [ADIN+17] and formally stated in [BCGI+20a]. Since its introduction, this formalism has gained interest and has been utilized in numerous works [CRR21; BCGI+22; CD23; BCCD23; RRT23]. This chapter formally presents the linear test framework, explains how resistance against linear tests can be linked to a minimal distance property, and provides precise examples of why specific attacks fit into the model. The chapter will partially follow Section 3 of [CRR21].

4.2 Formal definition

Recall the definition of bias introduced in Definition 2.2.1:

Definition 4.2.1 (Bias of a Distribution)**.** *Given a distribution $\mathcal{D}$ over $\mathbb{F}^n$ and a vector $\mathbf{v} \in \mathbb{F}^n$, the bias of $\mathcal{D}$ with respect to $\mathbf{v}$, denoted $\mathsf{bias}_{\mathbf{v}}(\mathcal{D})$, is equal to*

$$\mathsf{bias}_{\mathbf{v}}(\mathcal{D}) = |\mathbb{P}_{\mathbf{x}\sim\mathcal{D}}[\mathbf{v}^\intercal \cdot \mathbf{x} = 0] - \mathbb{P}_{\mathbf{x}\sim\mathcal{U}_n}[\mathbf{v}^\intercal \cdot \mathbf{x} = 0]| = \left|\mathbb{P}_{\mathbf{x}\sim\mathcal{D}}[\mathbf{v}^\intercal \cdot \mathbf{x} = 0] - \frac{1}{|\mathbb{F}|}\right|,$$

where $\mathcal{U}_n$ denotes the uniform distribution over $\mathbb{F}^n$. The bias of $\mathcal{D}$, denoted $\mathsf{bias}(\mathcal{D})$*, is the maximum bias of $\mathcal{D}$ with respect to any nonzero vector $\mathbf{v}$.*

We introduce formally the intuition that we gave during the introduction.

Definition 4.2.2 (Linear Tests Framework)**.** *Let $\mathbb{F}$ represent an arbitrary finite field, and let $\mathcal{D} = \{\mathcal{D}_{m,n}\}_{m,n\in\mathbb{N}}$ denote a family of noise distributions over $\mathbb{F}^n$. Let $\mathfrak{C}$ be a probabilistic code generation algorithm such that $\mathfrak{C}(m,n)$ outputs a matrix $\mathbf{H} \in \mathbb{F}^{m\times n}$. The* Linear Tests Framework *consists in the following game (represented in Figure 4.1):*

- *Sample $\mathbf{H} \xleftarrow{\$} \mathfrak{C}(n,m)$.*
- *Send $\mathbf{H}$ to the adversary $\mathcal{A}$.*
- *Let the adversary an* unbounded time *to compute an* attack vector *$\mathbf{v} \in \mathbb{F}^m$.*

*Let λ denote a security parameter. Let $\varepsilon, \delta : \mathbb{N} \to [0,1]$ be two functions. A $(\mathcal{D}, \mathfrak{C}, \mathbb{F})$-$\mathsf{SD}(m,n)$ assumption with dimension $m = m(\lambda)$ and $n = n(\lambda)$ samples is said to be (ε,δ)-*secure in the linear tests framework *if for any adversary $\mathcal{A}$ with unlimited time which, on input a matrix $\mathbf{H} \in \mathbb{F}^{m\times n}$, outputs a nonzero attack vector $\mathbf{v} \in \mathbb{F}^m$, it holds that*

$$\Pr_{\mathbf{H} \xleftarrow{\$} \mathfrak{C}(m,n),\ \mathbf{v} \leftarrow \mathcal{A}(\mathbf{H})} [\ \mathsf{bias}_{\mathbf{v}}(\mathcal{D}_{\mathbf{H}}) \geqslant \varepsilon(\lambda)\ ] \leqslant \delta(\lambda), \tag{4.1}$$

where $\mathcal{D}_{\mathbf{H}}$ is the distribution induced by sampling $\mathbf{e} \xleftarrow{\$} \mathcal{D}_{m,n}$, and outputting the SD *sample $\mathbf{He}$.*

This means that for any adversary $\mathcal{A}$ that outputs any attack vector $\mathbf{v}$, the probability that the distribution $\{\mathbf{He} \mid \mathbf{e} \leftarrow \mathcal{D}_{m,n}\}$ is biased, with respect to the vector $\mathbf{v}$, above $\varepsilon(\lambda)$ is lower than $\delta(\lambda)$. Let us properly analyze the implications of Equation (4.1). This bound acts as an upper limit on the advantage that an adversary running in polynomial time can have in distinguishing $\mathbf{He}$ from random. To bound the advantage of the adversary, consider the following: with probability $1 - \delta(\lambda) \approx 1$, we obtain a bias of at most $\varepsilon(\lambda)$, resulting in an advantage of at most $\varepsilon(\lambda)$. In the remaining case, which occurs with probability $\delta(\lambda)$, we only know that the bias is not bounded by $\varepsilon(\lambda)$. Here, we can upper bound the advantage by $\delta(\lambda)$. Therefore, the upper bound on the advantage of an adversary whose attack falls into the linear tests framework is $\delta(\lambda) + \varepsilon(\lambda)$. We will subsequently choose ε and δ to be exponentially small in the security parameter λ.

Remark 4.2.1 (From a Bias to an Actual Distinguisher)**.** How can we transform the information on the maximum bias associated with an attack vector $\mathbf{v}$ into an actual distinguisher? Suppose the distinguisher can pick to N attack vectors $\mathbf{v_1}, \ldots, \mathbf{v_N}$, all depending on $\mathbf{H}$. Then, we sample the error vector $\mathbf{e}$. The adversary learns the value associated with each attack vector, $\mathbf{v_1}^\intercal \cdot \mathbf{He}, \ldots, \mathbf{v_N}^\intercal \cdot \mathbf{He}$. The adversary can distinguish by looking at the number of zeros in this list. Each of the $\mathbf{v_i}^\intercal \cdot \mathbf{He}$ can provide distinguishing power of at most ε. Therefore, to obtain a noticeable effect, a simple union bound indicates that at least $N = \mathsf{poly}(\lambda)/\varepsilon$ samples are needed, where λ is a security parameter. Because the adversary is given the freedom to choose different $\mathbf{v_i}$, the results are highly correlated, and we cannot provide a stronger bound on the number of samples required. We argue, therefore, that a given distinguisher should take time $T = aN$, where a is the cost of the multiplication by a vector $\mathbf{v_i}$ (in $O(m)$).

Conjecture 4.2.1. *All the attacks on the search version of* SD *listed in the introduction can be cast on an attack on decision which fits in the linear test framework. That is, for every attack in the list which*

finds the error vector in time T, we can find an attack vector $\mathbf{v}$ *such that the associated bias is bounded by* $\mathsf{poly}(\lambda)/T$. *As a consequence, any* SD *assumption that is provably* (ε, δ)*-secure against linear tests would require at least* $\mathsf{poly}(\lambda)/\varepsilon$.

We propose in Section 4.4 a precise analysis showing why some attacks fall into the linear tests framework and how their bias relates to the running time of the attack they represent.

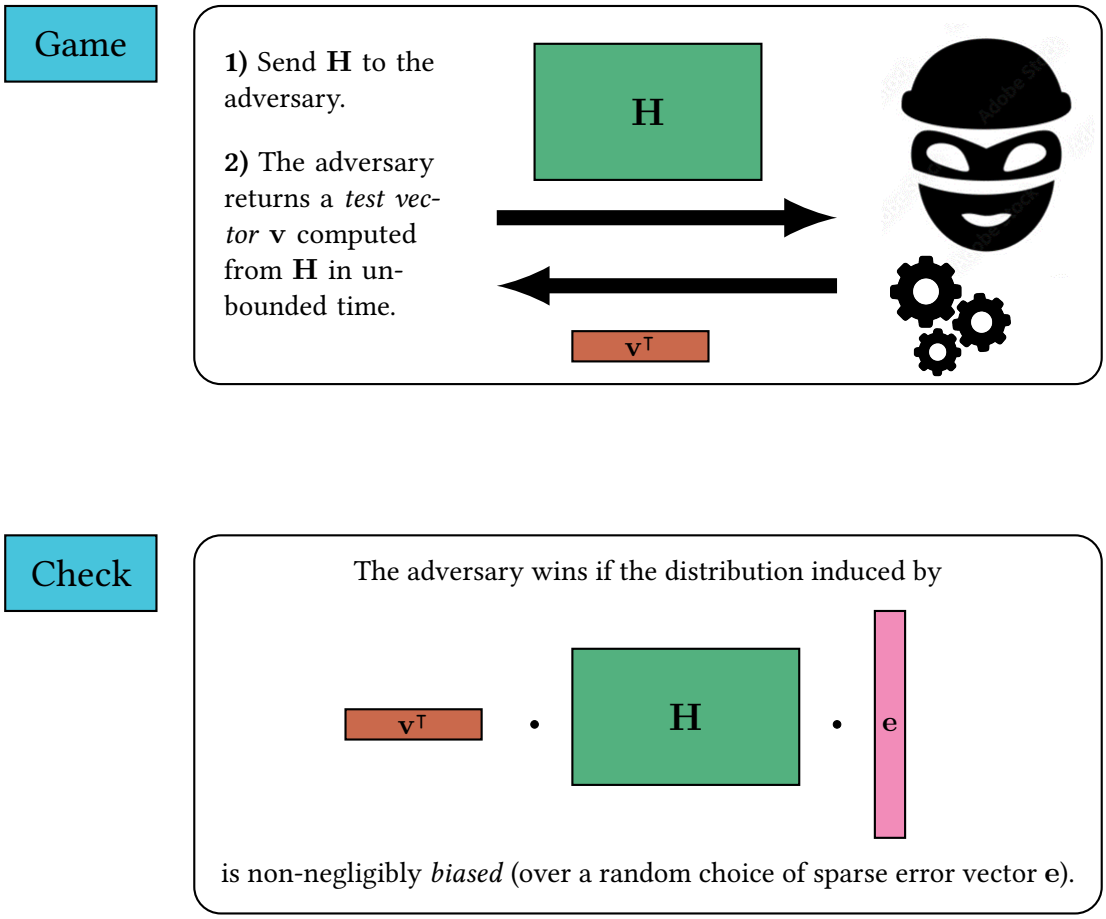

Figure 4.1 – Representation of the linear test framework.

4.3 Properties of the Linear Test Framework and discussion

We thereafter present an important property of the linear tests framework that we will use later in our constructions: if the code produced by the rows of $\mathbf{H}$ has a good minimum distance, then we can have resistance in the linear test framework. We start by providing an intuition of why it is the case. The linear test framework asks to analyze the bias of $\mathbf{He}$ with respect to the attack vector $\mathbf{v}$. Because $\mathbf{e}$ is a sparse vector, if $\mathbf{v}^\intercal\mathbf{H}$ is also sparse, the result is likely to be significantly biased. However, $\mathbf{v}^\intercal\mathbf{H}$ can be viewed as a codeword in the code generated by the rows of $\mathbf{H}$. Therefore, the condition that $\mathbf{v}^\intercal\mathbf{H}$ is not sparse implies that the code produced by the rows of $\mathbf{H}$ must have a high minimum distance.

Proposition 4.3.1 (The importance of the minimal distance). *Let $\mathcal{D} = (\mathcal{D}_{m,n})_{m,n\in\mathbb{N}}$ denote a family of noise distributions over $\mathbb{F}^n$. Let $\mathfrak{C}$ be a probabilistic matrix generation algorithm such that $\mathfrak{C}(n,m) \to \mathbf{H} \in \mathbb{F}^{m\times n}$. We denote by $d_{\mathbf{H}}$ the minimal distance of the code generated by the rows of the matrix $\mathbf{H}$. Then, for any integer d, the $(\mathcal{D}, \mathfrak{C}, \mathbb{F})$- $\mathsf{SD}(m,n)$ assumption with dimension $m = m(\lambda)$ and $n = n(\lambda)$ is $(\varepsilon_d, \delta_d)$-secure in the linear tests framework with*

$$\varepsilon_d = \max_{wt(\mathbf{u})\geqslant d} \mathsf{bias}_{\mathbf{u}}(\mathcal{D}_{m,n}),$$

and

$$\delta_d = \Pr_{\mathbf{H} \xleftarrow{\$} \mathfrak{C}(m,n)} [\, d_{\mathbf{H}} < d \,].$$

Proof. The proof follows directly from Equation (4.1), utilizing the formula of total probability under the assumption that $d_{\mathbf{H}} \geqslant d$:

$$\Pr[\,\mathsf{bias}_{\mathbf{v}} > \varepsilon_d\,] = \underbrace{\Pr[\,\mathsf{bias}_{\mathbf{v}} > \varepsilon_d \mid d_{\mathbf{H}} < d\,]\Pr[\,d_{\mathbf{H}} < d\,]}_{<\delta_d} + \underbrace{\Pr[\,\mathsf{bias}_{\mathbf{v}} > \varepsilon_d \mid d_{\mathbf{H}} \geqslant d\,]\Pr[\,d_{\mathbf{H}} \geqslant d\,]}_{0}$$

□

If the code produced by the rows of $\mathbf{H} \xleftarrow{\$} \mathfrak{C}$ has a high minimum distance with high probability, and given a suitable choice of noise distribution, we can achieve an exponentially small bias ε_d and an exponentially small probability of deviation from this bias δ. More formally, given a noise distribution $\mathcal{E}_n$ and an integer d, we focus on the maximum bias of $\mathcal{E}_n$ induced by every codeword, such that $\varepsilon_d(\mathcal{E}_n) = \max_{wt(\mathbf{u})>d} \mathsf{bias}_{\mathbf{u}}(\mathcal{E}_n)$. We will show that the bias decreases exponentially with the parameter d. For example consider the case of Bernoulli noise, with parameters t/n, that we denote by $\mathcal{B}_{n,t}$: it assigns independently each entry of the vector a 1 with probability t/n.

Lemma 4.3.1. *For any integer d,*

$$\varepsilon_d(\mathcal{B}_{n,t}) \leqslant (1-2t/n)^{d+1}/1 \leqslant \exp(-2(d+1)t/n)/2.$$

Proof. The claim is a direct result of the piling-up lemma (Lemma 2.2.2): indeed, we are interested in the bias of $\langle \mathbf{c}, \mathbf{e}\rangle$, where $\mathbf{e} \xleftarrow{\$} \mathcal{B}_{n,t}$, which is a xor of $wt\,(c) > d$ Bernoulli samples of rate t/n. □

In the case of the Bernoulli distribution, the bias decreases exponentially with the integer d.

Example 4.3.1 (Resistance of random code, from [CRR21]). It is broadly assumed by the community that the SD assumption is hard when $\mathbf{H} \in \mathbb{F}_2^{m\times n}$ is random. We show how we can easily bring evidence for that claim in the linear attack framework. Consider a uniformly random parity-check matrix $\mathbf{H} \in \mathbb{F}_2^{m\times n}$, sampled via the Bernoulli distribution and associated with Bernoulli rate $1/2$. The minimal distance of the code generated by the rows of $\mathbf{H}$ is equal to $d_{\mathbf{H}}$ as soon as $d_{\mathbf{H}}$ columns of $\mathbf{H}^*$ are independent, where $\mathbf{H}^*$ is the parity-check matrix of the code induced by the rows of $\mathbf{H}$. Because $\mathbf{H}$ is random, $\mathbf{H}^*$ is also random. The probability that d random vectors over $\mathbb{F}_2^m$ are linearly independent is at least

$$\prod_{i=0}^{d-1} \frac{2^m - 2^i}{2^m} \geqslant (1-2^{d-1-m})^d \geqslant 1 - 2^{2d-m}.$$

To ensure that there do not exist d dependent random vectors over $\mathbb{F}_2^m$, we must consider all the $\binom{n}{d}$ possibilities. The probability the minimal distance of the code generated by the rows of $\mathbf{H}$ is greater than d is at least $1 - \binom{n}{d} \cdot 2^{2d-m} \geqslant 1 - 2^{(2+\log(n))d-m}$. Using Lemma 4.3.1, we bound ε_d by $\exp(-2(d+1)t/m)/2$.

Setting $d = O(n/\log(m))$ is sufficient to obtain a probability of $\delta_d = 1 - 2^{-O(n)}$, and $\varepsilon_d \leqslant e^{-\Omega(t/\log(m))}$. Therefore, any attack fitting into the linear test framework, with noise rate r, requires on the order of $e^{\Omega(t/\log(m))}$ iterations.

Remark 4.3.1 (On the value δ_d)**.** In this manuscript, we consider both ε and δ_d to be exponentially small in d. However, this condition is not always necessary for δ_d. For example, in [BCGI+22; RRT23], the authors consider a δ_d that is polynomially small in d. This is because, in the case of PCGs, the parties can sample the matrix $\mathbf{H}$, and take the bet that the matrix has a good minimal distance. With probability $1 - \delta_d$, this is the case, and we have the guarantee that the bias will never be bigger than ε. This setup can be interesting for some applications. In contrast, for PCFs, this is not exactly feasible, as the matrix is virtually generated on-the-fly.

4.4 Attacks inside the linear test framework

In this subsection, we demonstrate how a linear test can be precisely derived from an actual attack, thereby showing why the aforementioned attacks fit within the framework of linear tests. We will focus on standard information set decoding (ISD) techniques, which are the ones of interest for our applications. These techniques are thoroughly analyzed in Appendix B. For each attack algorithm, we will recall its formal description and exhibit an attack vector representing the linear operations performed during the attack. Next, we will compute the bias associated with each attack vector and, using Remark 4.2.1, compare it with the actual running time of the attack. Some of the computations are considered free within the linear test model: all costs associated with finding the vector $\mathbf{v}$ are discarded. The dominant costs (exponential) remain the same when the attack vector is exactly the one associated with the attack. In the following we will work over $\mathbb{F}_2$ for the sake of simplicity, but it can be generalized to bigger fields.

4.4.1 The exhaustive search

We start easy with the exhaustive search: it consist to randomly guess the error. The algorithm is described on Figure 4.2.

Proposition 4.4.1. *Algorithm Figure 4.2 expected running time is equal to* $T_{\text{Exhaustive}}\binom{n}{t}$.

Proof. The probability to pick the right vector is $\frac{1}{\binom{n}{t}}$. The expected running time is the inverse of that probability. □

We identify a linear attack that captures the essence of the exhaustive search attack. The attack vector corresponding is produced by Figure 4.3.

Proposition 4.4.2. *The bias induced by the vector* $\mathbf{v}$ *produced by the algorithm of Figure 4.3 is* $\approx \frac{1}{2\binom{n}{k}}$.

Proof. The bias is the distance to $1/2$ of $\Pr[\, \mathbf{v}^\intercal \cdot \mathbf{s} = 0 \,]$.

INPUT: Matrix $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$, vector $\mathbf{s} \in \mathbb{F}_2^n$, target weight t.
OUTPUT: $\mathbf{e} \in \mathbb{F}_2^{n-k}$.
$\mathcal{A}$ ($\mathbf{H}$):
1: Sample $\mathbf{e'} \in \mathbb{F}_2^n$ of weight t.
2: **While** $\mathbf{s} \neq \mathbf{He'}$:
4.1: Sample $\mathbf{e'} \in \mathbb{F}_2^n$ of weight t.
3: **Return** $\mathbf{e}$.

Figure 4.2 – Exhaustive Search Algorithm

INPUT: Matrix $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$.
OUTPUT: $\mathbf{v} \in \mathbb{F}_2^{n-k}$.
linearExhaustiveSearch($\mathbf{H}$):
1: Sample $\mathbf{e'} \in \mathbb{F}_2^n$ of weight t.
2: Define $\mathbf{v}$ as an orthogonal vector to $\mathbf{H} \cdot \mathbf{e'}$.
3: **Return** $\mathbf{v}$.

Figure 4.3 – Exhaustive search linear attack vector

$$\Pr\left[\, \mathbf{v}^\intercal \cdot \mathbf{s} = 0 \,\right] = \Pr\left[\, \mathbf{v}^\intercal \mathbf{He} = 0 \mid \mathbf{e'} \neq \mathbf{e} \,\right] \Pr\left[\, \mathbf{e'} \neq \mathbf{e} \,\right] + \Pr\left[\, \mathbf{v}^\intercal \mathbf{He} = 0 \mid \mathbf{e'} = \mathbf{e} \,\right] \Pr\left[\, \mathbf{e'} = \mathbf{e} \,\right].$$

$$\Pr\left[\, \mathbf{v}^\intercal \cdot \mathbf{s} = 0 \,\right] = \Pr\left[\, \mathbf{v}^\intercal \mathbf{He} = 0 \mid \mathbf{e'} \neq \mathbf{e} \,\right] \left(1 - \frac{1}{\binom{n}{t}}\right) + \frac{1}{\binom{n}{t}}.$$

Note that $\mathbf{H}$ is initially sampled as a random matrix: therefore $\mathbf{He}$ is equal to a sum of t random columns. On the other hand, $\mathbf{v}$ is chosen uniformly in the orthogonal complement of $\mathbf{He'}$. Therefore, the part of $\mathbf{H}$ that influences the choice of $\mathbf{v}$ depends only on the column of $\mathbf{H}$ corresponding to $\mathbf{e'}$. Note therefore that in the sum of columns equal to $\mathbf{He}$, there is at least one column that was not taken into account when sampling $\mathbf{v}$. This column is random by assumption and therefore, we can conclude that

$$\Pr\left[\mathbf{v}^\intercal \mathbf{He} = 0 \mid \mathbf{e'} \neq \mathbf{e}\right] = \frac{1}{2}.$$

Therefore, we obtain that

$$\mathsf{bias}_{\mathsf{Exhaustive}}(\mathbf{v}) = \frac{1}{\binom{n}{t}} - \frac{1}{2}\left(1 - \frac{1}{\binom{n}{t}}\right) = O\left(\frac{1}{2\binom{n}{t}}\right).$$

This lead to the expected bias. □

4.4.2 Prange Algorithm

We now consider the simplest ISD algorithm: the Prange algorithm, displayed in Figure 4.4. More information can be found in Appendix B. For a subset $\mathcal{I} \subset [0, n-1]$, we denote by $\overline{\mathcal{I}}$ the complementary set $[0, n-1] \backslash \mathcal{I}$. For a vector $x \in \mathbb{F}_q^n$, $x_{\mathcal{I}}$ denotes the vector restricted to the entries indexed by $\mathcal{I}$, that is $x_{\mathcal{I}} = (x_i)_{i \in \mathcal{I}}$.

INPUT: Matrix $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$, Vector $\mathbf{s} \in \mathbb{F}_2^n$, target weight t.
OUTPUT: $\mathbf{e} \in \mathbb{F}_2^{n-k}$.
$\mathcal{A}$ ($\mathbf{H}$):

1: Pick randomly a subset $\mathcal{I} \subset [0, n-1]$, $|\mathcal{I}| = k$, until $\mathbf{H}_{\overline{\mathcal{I}}}$ is full rank.
2: Perform a Gaussian elimination to compute a non-singular matrix $\mathbf{U} \in \mathbb{F}_q^{(n-k)\times(n-k)}$ such that $\mathbf{U}\mathbf{H}_{\overline{\mathcal{I}}} = \mathbf{I}_{n-k}$. Write $\mathbf{U}\mathbf{H}_{\mathcal{I}} = \mathbf{H_1}$.
3: Construct $\tilde{\mathbf{e}} = \mathbf{U}\mathbf{s}$ and check whether $wt(\tilde{\mathbf{e}}) = t$. If it's not the case, go to step 1.
4: Let $\mathbf{e'}$ such that $\mathbf{e'}_{\overline{\mathcal{I}}} = \tilde{\mathbf{e}}$ and $\mathbf{e'}_{\mathcal{I}} = 0$.
5: **return** $\mathbf{e'}$

Figure 4.4 – Prange Algorithm

Proposition 4.4.3. *The expected running time of the Prange algorithm is equal to* $n(n-k)^2\binom{n}{t}/\binom{n-k}{t}$. *(see Corollary B.2.1)*

We identify an attack vector $\mathbf{v}$ that captures the essence of the Prange algorithm attack. The attack vector corresponding is produced by Figure 4.5.

INPUT: Matrix $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$.
OUTPUT: $\mathbf{v} \in \mathbb{F}_2^{n-k}$.
linearAttackPrange($\mathbf{H}$):

1: Pick randomly a subset $\mathcal{I} \subset [0, n-1]$, $|\mathcal{I}| = k$, until $\mathbf{H}_{\overline{\mathcal{I}}}$ is full rank.
2: Via Gaussian elimination, compute the corresponding non-singular matrix $\mathbf{U} \in \mathbb{F}_q^{(n-k)\times(n-k)}$ such that $\mathbf{U}\mathbf{H}_{\overline{\mathcal{I}}} = \mathbf{I}_{n-k}$.
3: Choose a unit vector $\mathbf{u}$.
4: **return** $\mathbf{v} = \mathbf{U}^\intercal\mathbf{u}$.

Figure 4.5 – Prange Linear Attack

Proposition 4.4.4. *The bias of the attack vector* $\mathbf{v}$ *produced by the algorithm of Figure 4.5 is equal to* $\frac{n-k-2t}{n-k} \cdot \frac{\binom{n-k}{t}}{2\binom{n}{t}}$.

Proof. The bias is just the distance to $1/2$ of the probability $\Pr[\mathbf{v}^\intercal \cdot \mathbf{s} = 0]$. Let $\mathcal{G}$ be the "event the information set picked is such that $\mathbf{e}_{\mathcal{I}} = 0$" . In the following, $\mathcal{E}^c$ denote the complement of an event $\mathcal{E}$.

$$\Pr[\,\mathbf{v}^\intercal \cdot \mathbf{s} = 0\,] = \Pr[\,\mathbf{v}^\intercal \cdot \mathbf{s} = 0 \mid \mathcal{G}\,] \times \Pr[\mathcal{G}] + \Pr[\,\mathbf{v}^\intercal \cdot \mathbf{s} = 0 \mid \mathcal{G}^c\,] \times \Pr[\,\mathcal{G}^c\,].$$

$$\Pr[\,\mathbf{v}^\intercal \cdot \mathbf{s} = 0\,] = \frac{t}{n-k} \times \frac{\binom{n-k}{t}}{\binom{n}{t}} + \Pr[\mathbf{v}^\intercal \cdot \mathbf{s} = 0 \mid \mathcal{G}^c] \times \left(1 - \frac{\binom{n-k}{t}}{\binom{n}{t}}\right).$$

This left us with $\Pr[\,\mathbf{v}^\intercal \cdot \mathbf{s} = 0 \mid \mathcal{G}^c\,]$ to analyze.

$$\Pr[\,\mathbf{v}^\intercal \cdot \mathbf{s} = 0 \mid \mathcal{G}^c\,] = \Pr[\,\mathbf{u}^\intercal \cdot \mathbf{U}\mathbf{H}\mathbf{e} = 0 \mid \mathcal{G}^c\,] = \Pr\left[\,\mathbf{u}^\intercal \cdot (\mathbf{e}_{\overline{\mathcal{I}}} + \mathbf{U}\mathbf{H}_{\mathcal{I}}\mathbf{e}_{\mathcal{I}}) = 0 \mid \mathbf{e}_{\mathcal{I}} \neq 0\,\right].$$

As $\mathcal{I}$ is sampled randomly, and because $\mathbf{H}$ is initially sampled as a random matrix, the matrix $\mathbf{H}_1 = \mathbf{U}\mathbf{H}_{\mathcal{I}}$ can also be considered to be random. Then the $\mathbf{U}\mathbf{H}_{\mathcal{I}}\mathbf{e}_{\mathcal{I}}$ shall produce a well-balanced column with high probability, and therefore

$$\Pr\left[\,\mathbf{u}^\intercal \cdot (\mathbf{e}_{\overline{\mathcal{I}}} + \mathbf{U}\mathbf{H}_{\mathcal{I}}\mathbf{e}_{\mathcal{I}}) = 0 \mid \mathbf{e}_{\mathcal{I}} \neq 0\,\right] = \frac{1}{2}.$$

We obtain therefore,

$$\Pr[\,\mathbf{v}^\intercal \cdot \mathbf{s} = 0\,] = \frac{t}{n-k} \times \frac{\binom{n-k}{t}}{\binom{n}{t}} + \frac{1}{2} \times \left(1 - \frac{\binom{n-k}{t}}{\binom{n}{t}}\right).$$

We can deduce

$$\mathsf{bias}_{\mathsf{Prange}}(\mathbf{v}) = \frac{n-k-2t}{n-k} \cdot \frac{\binom{n-k}{t}}{2\binom{n}{t}}.$$

We can again compare this to the expected time running time of the Prange algorithm: $n(n-k)\binom{n}{t}/\binom{n-k}{t}$, which is naturally polynomially larger than $1/\mathsf{bias}_{\mathsf{Prange}}(\mathbf{v})$. This is expected because in the linear attack framework we are not taking into account the different polynomial costs induced during the attack (in this case the cost of the Gaussian elimination). This is because we assume that the adversary has an unlimited time in order to compute the attack vector $\mathbf{v}$. □

4.4.3 Birthday decoding algorithm

We now consider the birthday decoding algorithm. It builds on the famous so called birthday paradox. The algorithm is described on Figure 4.6.

Proposition 4.4.5. *The expected running time of the Birthday decoding algorithm is equal to* $2\binom{n}{t}/\binom{n/2}{t/2} + \binom{n}{t}/2^{n-k} = 2\binom{n/2}{t/2} + \binom{n}{t}/2^{n-k}$.

Proof. Let C denote the number of column operations for the steps 2 and 3. The costs of the creation of the lists S_i is $2\binom{n/2}{t/2}$. Indeed, we are just enumerating the vectors $\mathbf{e_i} \in \mathbb{F}_2^{n/2}$. From the two lists of size L, we can constitute L^2 possible pairs The size of the interection is about $L^2/2^{n-k}$, and therefore,

INPUT: Matrix $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$, Vector $\mathbf{s} \in \mathbb{F}_2^n$.
OUTPUT: $\mathbf{e'} \in \mathbb{F}_2^{n-k}$.
$\mathcal{A}$ (**H**):

1: Choose a subset $\mathcal{I} \subset [0, n-1]$ of size $n-k$. Let $\mathbf{H}_0 = \mathbf{H}_{\mathcal{I}}$ and $\mathbf{H}_1 = \mathbf{H}_{\overline{\mathcal{I}}}$.
2: For $i \in \{0,1\}$, create the sets $S_i = \left\{ \mathbf{H}_i \mathbf{e_i} + (1-i)\mathbf{s} \mid \mathbf{e_i} \in \mathbb{F}_2^{n/2}, wt(\mathbf{e_i}) = t/2 \right\}$.
3: Find the intersection between the two lists and extract the corresponding $(\mathbf{e_0}, \mathbf{e_1})$. If there is no intersection go to step 1.
4: Construct $\mathbf{e'}$ such that $\mathbf{e'}_{\mathcal{I}} = \mathbf{e_0}$ and $\mathbf{e'}_{\overline{\mathcal{I}}} = \mathbf{e_1}$.
5: **return** $\mathbf{e'}$.

Figure 4.6 – Birthday decoding algorithm

the computation of the interesection can be done in $\binom{n/2}{t/2}^2 / 2^{n-k}$. Remark that a solution can be found only if the Hamming weight e is such that $wt(e_{\mathcal{I}}) = w/2$. This is the case with probability $P = \frac{\binom{n/2}{t/2}}{\binom{n}{t}}$. To obtain the expectancy of the total number of operation we devide therefore the number of operation in the case 2 and 3 by the probability. This gives

$$C/P = \left(2\binom{n/2}{t/2} + \binom{n/2}{t/2}^2 / 2^{n-k} \right) \cdot \frac{\binom{n}{t}}{\binom{n/2}{t/2}^2} = 2\frac{\binom{n}{t}}{\binom{n/2}{t/2}} + \frac{\binom{n}{t}}{2^{n-k}}.$$

We end the proof with the approximations $\binom{n}{t} \sim \binom{n/2}{t/2}$. □

Remark 4.4.1. The left-hand term $\binom{n}{t}/2^{n-k}$ is important to take into account only when we expect multiple solutions. In our case, it will be negligible.

We identify an attack vector $\mathbf{v}$ that captures the essence of the Birthday decoding algorithm. We consider it in the simpler case where $k = n/2$ for convenience. The attack vector corresponding is produced by the algorithm of Figure 4.7.

Proposition 4.4.6. *The bias of the attack vector* $\mathbf{v}$ *produced by the algorithm of Figure 4.7 is equal to* $\Pr[\, \mathbf{v}^\intercal \cdot \mathbf{s} = 0 \,] = \frac{(n-2t)\cdot\sqrt{\frac{2}{nt(n-t)}}}{2\binom{n/2}{t/2}}$.

Proof. Let us analyze $\Pr[\mathbf{v}^\intercal \cdot \mathbf{s} = 0]$. Let $\mathcal{G}$ be the event $wt(e_{\mathcal{I}}) = t/2$, and $\mathcal{H}$ be the event $\mathbf{e'_1} = \mathbf{e_1}$.

$$\Pr[\, \mathbf{v}^\intercal \cdot \mathbf{s} = 0 \,] = \Pr[\, \mathbf{v}^\intercal \cdot \mathbf{s} = 0 \mid \mathcal{G} \,] \cdot \Pr[\, \mathcal{G} \,] + \Pr[\, \mathbf{v}^\intercal \cdot \mathbf{s} = 0 \mid \mathcal{G}^c \,] \cdot \Pr[\, \mathcal{G}^c \,]$$

We compute, for n and t going to infinity (via the Stirling formula):

$$\Pr[\, \mathcal{G} \,] = \frac{\binom{n/2}{t/2}^2}{\binom{n}{t}} \sim \sqrt{\frac{2n}{\pi \cdot t \cdot (n-t)}}.$$

Moreover,

INPUT: Matrix $\mathbf{H} \in \mathbb{F}_2^{(n/2)\times n}$.

OUTPUT: $\mathbf{v} \in \mathbb{F}_2^{n/2}$.

linearAttackBirthday($\mathbf{H}$):

1: Choose a subset $\mathcal{I} \subset [0, n-1]$ of size $n/2$ until $\mathbf{H}_\mathcal{I}$ is full rank. Let $\mathbf{H}_0 := \mathbf{H}_\mathcal{I}$ and $\mathbf{H}_1 := \mathbf{H}_{\overline{\mathcal{I}}}$.

2: Sample $\mathbf{e_1'} \xleftarrow{\$} \mathbb{F}_2^{n/2}$, $wt\left(\mathbf{e_1'}\right) = t/2$

3: Compute H_0^{-1}.

4: Search a row $\mathbf{v}^\intercal$ of H_0^{-1} such that $\mathbf{v}^\intercal \mathbf{H}_1 \mathbf{e_1} = 0$
▷ Such a vector exist with high probability:
▷ The probability that a row of H_0^{-1}, which is random, does not vanish $\mathbf{H}_1 e_1'$ is $1/2$.
▷ Therefore, with probability $1 - 1/2^{n/2}$, $\mathbf{v}$ exists.

5: **return** $\mathbf{v}$

Figure 4.7 – Birthday Linear Attack

$$\Pr\left[\, \mathbf{v}^\intercal \cdot \mathbf{s} = 0 \mid \mathcal{G}^c \,\right] = \Pr\left[\, \mathbf{u}^\intercal \mathbf{e_0} + \mathbf{u}^\intercal \mathbf{H}_0^{-1} \mathbf{H}_1 (\mathbf{e_1} - \mathbf{e_1'}) = 0 \mid \mathcal{G}^c \,\right].$$

Because $\mathbf{u}^\intercal \mathbf{H}_0^{-1} \mathbf{H}_1 (\mathbf{e_1} - \mathbf{e_1'})$ amounts to a non-empty sum of product of Bernoulli variables with parameter $1/2$. $\mathbf{u}^\intercal \mathbf{e_0}$ on the other side is simply a biased Bernoulli. It follows that:

$$\Pr\left[\, \mathbf{v}^\intercal \cdot \mathbf{s} = 0 \mid \mathcal{G}^c \,\right] = \frac{1}{2}. \tag{4.2}$$

Further we compute:

$$\Pr\left[\, \mathbf{v}^\intercal \cdot \mathbf{s} = 0 \mid \mathcal{G} \,\right] = \Pr\left[\, \mathbf{v}^\intercal \cdot \mathbf{s} = 0 \mid \mathcal{G}, \mathcal{H} \,\right] \cdot \Pr\left[\, \mathcal{H} \,\right] + \Pr\left[\, \mathbf{v}^\intercal \cdot \mathbf{s} = 0 \mid \mathcal{G}, \mathcal{H}^c \,\right] \cdot \Pr\left[\, \mathcal{H}^c \,\right].$$

Note that

$$\Pr[\mathcal{H}] = \frac{1}{\binom{n/2}{t/2}}.$$

Moreover

$$\Pr\left[\, \mathbf{v}^\intercal \cdot \mathbf{s} = 0 \mid \mathcal{G}, \mathcal{H} \,\right] = \Pr\left[\, \mathbf{u}^\intercal \mathbf{e_0} = 0 \mid \mathcal{G}, \mathcal{H} \,\right] = 1 - t/n.$$

and

$$\Pr\left[\, \mathbf{v}^\intercal \cdot \mathbf{s} = 0 \mid \mathcal{G}, \mathcal{H}^c \,\right] = \Pr\left[\, \mathbf{v}^\intercal \cdot \mathbf{s} = 0 \mid \mathcal{G}, \mathcal{H}^c \,\right] = 1/2$$

where the last equality stands for the same reasons as for Equation (4.2). Adding everything together yields

$$\Pr\left[\, \mathbf{v}^\intercal \cdot \mathbf{s} = 0 \,\right] \sim \frac{(n-2t) \cdot \sqrt{\frac{2}{nt(n-t)}}}{2\binom{n/2}{t/2}} + 1/2.$$

We again verify that the inverse of the bias acts as an appropriate lower bound on the execution time of the birthday paradox algorithm. We obtain again the same dominant factor $\binom{n/2}{t/2}$. □

Chapter 5

PCGs for OLE correlations

In 2020, the first PCG (see Definition 3.1.3) to generate OLEs (see Definition 2.5.6) was proposed by [BCGI+20b]. However, it was still limited, both by its efficiency and by the fact that it could generate OLEs only over large fields. We provide another construction, using the QA-SD assumption (defined Section 2.4.3), to almost entirely get rid of the constraint. This construction is a natural generalization of the previous one, and offers much clearer insights on the security of the primitive. In this chapter, we compare the two known constructions by describing a general framework that encompass both. Additionally, we examine important optimizations to make this new QA-SD-based construction more efficient. The results presented come from two works produced in collaboration with Maxime Bombar, Dung Bui, Geoffroy Couteau, Alain Couvreur, and Sacha Servan-Schreiber, respectively accepted at *Crypto 2023* and available on eprint [BCCD23; BBCC+24].

Outline of the current chapter

5.1 State of the Art

Pseudorandom Correlation Generators (PCGs) were formally introduced in [BCGI+17; BCGI18; BCGI+19b] as a method to generate random correlations while minimizing communication overhead. Since then, this line of work has proven to be an attractive new area of research, attracting significant interest and yielding promising outcomes. Various directions have been explored since the original introductory articles. PCG targeting the Oblivious Transfer correlation (OT) has been extensively studied for the past six years, making good use of the balance between the simplicity of OT and its great importance [1]. As mentioned in Section 3.1, successive works build their PCGs for OT using function secret sharing (FSS, see Section 2.5.3.5), introduced by [BGI16]. With this powerful primitive, parties can easily share sparse vectors. Then, using a syndrome decoding-like assumption (SD, see Section 2.4.2), the parties can transform a sparse vector into a pseudorandom vector by multiplying with some well-chosen matrix $\mathbf{H}$. Because the additive sharing is linear, it appears that we can obtain additive shares of pseudorandom long vectors. It became immediately apparent that the choice of the matrix $\mathbf{H}$ was critical to propose a trade-off between efficiency and security. This has been the subject of most of the work done since then.

[BCGI18; BCGI+19b] initially proposed the use of LDPC codes or quasi-cyclic codes, which offer a nice balance between efficiency and security. These classes of codes have been studied for a long time, especially the syndrome decoding assumption in the context of cryptographic primitives ([Ale03; AABB+22a]). Subsequent research to find the best code continued under increasingly exotic assumptions (though still proven to resist most known attacks) with the following works [BCGI+20a; BCGI+22; RRT23]. The authors constructed their PCGs using variants of syndrome decoding, tailored for respectively *Variable Density Codes*, *Expand and Accumulate Codes*, and *Expand-Convolute Codes*. The former additionally achieves the property of providing PCF construction, as will be elaborated in Chapter 7. While all these papers originated from the code as a basis for constructing the PCG, [CRR21] attempted to construct the code iteratively, via computer simulations and heuristics to achieve more aggressive results (the proposed code was finally proven insecure by [RRT23], but the approach remains interesting). All in all, the best PCGs for OT achieve, as demonstrated in [BCGI+22; RRT23], up to 10 million OT correlations per second, on one core of a standard laptop, while remaining *silent* as requested by the PCG framework (meaning communication in $\log(m)$ for m generated OT, see Section 3.1). The *silent* requirement is crucial to the construction. Other techniques, not *silent*, can achieve faster OT generators: the SoftSpoken OT expansion protocol by Roy [Roy22] yields a faster OT generation reaching up to 30 million OTs on localhost. Nonetheless, this comes with a trade-off of increased communication, up to $64m$ for m generated OT (other communication/computation trade-offs are possible, see [Roy22, Table 1]). When considering N parties, the communication scales with a $N(N-1)$. As our ultimate goal is to execute MPC protocols with multiple players, this is not practical as soon as $N \sim 100$.

Concerning more involved correlations such as OLE or multiplication triples, the state of the art is less substantial. Constructing efficient PCG for OLE is the focus of the current chapter. In 2019, [BCGI+19b] proposed two ideas in order to achieve the OLE correlation. Both were inefficient, with very large seed size, and only efficient when generating huge batches. The first one was based on homomorphic secret sharing from ring-LWE ([BKS19]), the second was based on LPN and suffered from big computational costs. In 2020, an efficient solution was found [BCGI+20b], matching the best non-silent protocol [KPR18]. Nevertheless, we will show that this construction was unfortunately created with an inherent tedious constraint since it is usable only over large fields. The following sections will, in particular, provide insights on how to bypass (almost entirely)

1. they are building blocks of Beaver Triples which are important for the protocols used in Section 2.5.4.1

this constraint, using our work from *Crypto2023* [BCCD23]. Note that the previously mentioned works [BCGI+20a; BCGI+22] could be used as a PCG for OLEs (we follow what was outlined in Section 3.1.3.1). Concretely, each of the constructions builds on the fact that it is hard to distinguish $\mathbf{H} \cdot \mathbf{e}$ from random, where $\mathbf{H}$ is a carefully chosen sparse or structured matrix. Additionally $\mathbf{e_1} \cdot \mathbf{e_2}^\intercal$ can be shared using an appropriate FSS scheme since $\mathbf{e_1}, \mathbf{e_2}$ and $\mathbf{e}$ are all $\lambda \log(n)$-sparse vectors [2]. In this scenario, extracting the diagonal of $\mathbf{H e_1 e_2^\intercal H}^\intercal$ (which is the component-wise product of $\mathbf{He_1}$ and $\mathbf{He_2}$, hence gives n pseudorandom OLEs) does not require computing the full matrix, and scales in $\mathsf{poly}(\lambda) \cdot \tilde{\Omega}(n)$. This basic construction will be explained in detail Section 5.2, in the case of a random matrix $\mathbf{H}$. Remark that in these constructions, no constraints on the size of the field are required. Nevertheless, a closer look reveals some bad news. Indeed, the constant overhead turns out to be prohibitive in these cases. For example, in our recently published work of [CD23], we explained that they need a security parameter λ (number of noisy coordinates) as big as 350 for the assumption to be valid. Remember that if the Hamming weight of the noise vector $\mathbf{e}$ is equal to $\lambda \log(n)$, the number of FSS calls is then $\lambda^2 \log(n)^2$. Taking $n = 2^{30}$, this entails around 10^8 invocations of FSS calls for each OLE produced which is highly impractical.

How to Read this Chapter

We recommend the reader to consult Chapter 3 for a general understanding of the PCG primitive. Additionally, we will use extensively syndrome decoding and its variants, presented in Sections 2.4.2, 2.4.2.2 and 2.4.3. This chapter is structured as follows: Section 5.2 describes a first unfruitful construction of PCG for OLE and argues for the need for constructions using structured matrices, and therefore structured rings. Secondly, Section 5.3 presents a general framework for a PCG for OLE over a general ring $\mathcal{R}$, as long as $\mathcal{R}$ satisfies certain properties. Section 5.4 explains how both [BCGI+20a; BCCD23] fit into the general framework. Section 5.5 addresses some efficiency optimizations from [BCCD23] and [BBCC+24]. Finally, in the last section Section 5.6, we present a full MPC protocol from start to finish, FOLEAGE, which is fully optimized and state-of-the-art in numerous aspects according to our work [BBCC+24].

5.2 A First Unfructuous Tentative

In this section, we sketch out a simple method to obtain PCG for OLE, and argue for its shortcomings. This idea was briefly mentioned in Section 5.1 when explaining how the design of the PCGs for OLE from [BCGI+20a; BCGI+22; RRT23]. First, we recall in figure 5.1 the ideal functionality we expect from the PCG.

Figure 5.2 describes the protocol. Let $\mathbf{H}$ be a public random matrix. Party σ starts by generating $\mathbf{e_\sigma}$, a sparse vector of weight t. Then, party σ computes $\mathbf{x_\sigma} = \mathbf{He_\sigma}$ for $\mathbf{e_\sigma}$. Using FSS, parties obtain small FSS keys $(K_0, K_1) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, 1^n, A, B)$, where B denote the position of the non zero position in the matrix $\mathbf{e_0} \cdot \mathbf{e_1}^\intercal$, represented as a vector of size t^2, and A the associated list of non-coefficients.

.The seed given by the PCG to party σ is then $\mathsf{k}_\sigma = (K_\sigma, \mathbf{e_\sigma})$. From the seed, party σ can compute $\mathbf{u_\sigma} = \mathsf{SPFSS.FullEval}(\sigma, \mathbf{e_0} \cdot \mathbf{e_1}^\intercal)$. The protocol stops after that party σ computes $\mathbf{z_\sigma} = \mathsf{Diag}(\mathbf{H u_\sigma H}^\intercal)$. Here, $\mathsf{Diag}(\mathbf{A})$ returns the diagonal of the matrix $\mathbf{A}$.

This protocol is *correct*, as we can verify the following:

$$\mathbf{z_0} + \mathbf{z_1} = \mathsf{Diag}(\mathbf{H u_0 H}^\intercal) + \mathsf{Diag}(\mathbf{H u_1 H}^\intercal),$$

2. Meaning element non-zero on exactly $\lambda \log(n)$ entries.

Ideal Functionality $\mathsf{PCG}_{\mathsf{OLE}}$

PARAMETERS: Security parameter λ, $\mathsf{PCG}_{\mathsf{OLE}} = (\mathsf{PCG}_{\mathsf{OLE}}.\mathsf{Gen}, \mathsf{PCG}_{\mathsf{OLE}}.\mathsf{Expand})$.

FUNCTIONALITY:

1: Sample $(k_0, k_1) \leftarrow \mathsf{PCG}_{\mathsf{OLE}}.\mathsf{Gen}(1^\lambda)$.

2: Output k_σ to party P_σ for $\sigma \in \{0, 1\}$.

3: Party σ expands locally $(x_\sigma, z_\sigma) \leftarrow \mathsf{PCG}_{\mathsf{OLE}}.\mathsf{Expand}(\sigma, k_\sigma)$, with $x_0 x_1 = z_0 + z_1$.

Figure 5.1 – Functionality OLE.

and by definition of the SPFSS primitive it can be rewritten as follows:

$$\mathbf{z_0} + \mathbf{z_1} = \mathsf{Diag}(\mathbf{H e_0} \cdot \mathbf{e_1}^\intercal \mathbf{H}^\intercal) = \mathsf{Diag}(\mathbf{x_0} \cdot \mathbf{x_1}^\intercal).$$

Looking at the i-th entries of the vectors we obtain

$$\mathbf{z_0^i} + \mathbf{z_1^i} = \mathbf{x_0^i} \cdot \mathbf{x_1^i}.$$

Remark that each entries of the vectors define an OLE.

Moreover, the protocol is also *secure*. First, upon examining their definitions $\mathbf{x_0}$ and $\mathbf{x_1}$ are plain syndromes. When the matrix $\mathbf{H}$ is random, we are exactly within the setting of standard Syndrome Decoding presented in Section 2.4.2 and Definition 2.4.7. The assumption guarantees exactly the pseudorandomness of $\mathbf{x_0}$ and $\mathbf{x_1}$. Second, assuming the SPFSS scheme is secure, both $\mathbf{z_0}$ and $\mathbf{z_1}$ are pseudorandom. Therefore, we obtain the security requirements demanded by PCG.

Finally, it should be noted that $\mathbf{e_\sigma}$ is sparse and therefore of short description, and the FSS keys K_σ are also of short description. Therefore, the protocol satisfies also the requirements about the size of the seed k_σ.

5.2.1 A Dream End

Sadly, this construction is not efficient and cannot be used in practice. Looking at the definition of $\mathbf{z_\sigma}$, the parties have to compute $\mathbf{H u_\sigma H}^\intercal$, for a $\mathbf{u_\sigma}$ *not sparse* (even *pseudorandom*). Since we assume that $\mathbf{H}$ is a random matrix, the cost of the matrix multiplication is in $\Omega(n^\epsilon)$, with $\epsilon \approx 2.8$ from Strassen's algorithm [STR69] (further improvements on ϵ assumes absurd memory size and are therefore not taken into account here). The cost of the multiplication, performed here twice, is a deal-breaker for the method, as the target number n of OLEs to generate is very large ($\approx 2^{30}$) (see Chapter 1). To be more precise about the computation:

$$\mathbf{H u_\sigma H}^\intercal = \mathbf{H e_0 e_1}^\intercal \mathbf{H}^\intercal.$$

Let index the t non-zero coordinates of $\mathbf{e_0}$ by the set $\{i_0, \ldots, i_{t-1}\}$. We do similarly for $\mathbf{e_1}$, which is indexed by the set $\{j_0, \ldots, t_{j-1}\}$

$$\mathbf{H e_0} = \sum_{k=0}^{t-1} \mathbf{H}_{i_k}$$

Tentative construction of PCG for OLE

PARAMETERS:
Security parameter λ, noise weight $t = t(\lambda)$. An FSS scheme (SPFSS.Gen,SPFSS.FullEval) for sums of t^2 point functions, with domain $[0 \dots n)$ and range $\mathbb{F}_q$.

PUBLIC INPUT: Random matrix $\mathbf{H} \in \mathbb{F}_q^{(n-k)\cdot n}$.

$\mathsf{PCG.Gen}(1^\lambda)$:

1: **foreach** $\sigma \in \{0,1\}$:
  1.1: $\mathbf{e_\sigma} \leftarrow \{\mathbf{e} \in \mathbb{F}_q^n, wt(\mathbf{e}) = t\}$
2: Let $A = \{(\mathbf{e_0} \cdot \mathbf{e_1^\intercal})[i][j] \mid i,j \in [n], (\mathbf{e_0} \cdot \mathbf{e_1^\intercal})[i][j] \neq 0\}$ and $B = \{i \cdot n + j \mid i,j \in [n], (\mathbf{e_0} \cdot \mathbf{e_1^\intercal})[i][j] \neq 0\}$
3: Sample FSS keys $(K_0, K_1) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, 1^n, A, B)$
  ▷ B denote the position of the non zero position in the matrix $\mathbf{e_0} \cdot \mathbf{e_1^\intercal}$,
  ▷ represented as a vector of size t^2, and A the associated list of non-coefficients.
4: Let $\mathsf{k}_\sigma = ((K_\sigma), \mathbf{e_\sigma})$.
5: Output $(\mathsf{k}_0, \mathsf{k}_1)$.

$\mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma)$:

1: Parse k_σ as $((K_\sigma), \mathbf{e_\sigma})$.
2: Define $\mathbf{x_\sigma} = \mathbf{H}\mathbf{e_\sigma}$.
3: Compute $\mathbf{u_\sigma} \leftarrow \mathsf{SPFSS.FullEval}(\sigma, K_\sigma)$.
4: Compute $\mathbf{z_\sigma} = \mathsf{Diag}(\mathbf{H}\mathbf{u_\sigma}\mathbf{H}^\intercal)$.
  ▷ $\mathsf{Diag}(\mathbf{A})$ denote the diagonal of the matrix $\mathbf{A}$.
5: Output $\mathbf{x_\sigma}, \mathbf{z_\sigma}$.

Figure 5.2 – Tentative PCG for OLE over $\mathbb{F}_q$

and

$$\mathbf{He_1} = \sum_{k=0}^{t-1} \mathbf{H}_{j_k}.$$

Because $\mathsf{Diag}(\cdot)$ is linear, we obtain

$$\mathsf{Diag}(\mathbf{x}\mathbf{y}^\intercal) = \sum_{0 \leqslant l,k \leqslant t-1} \mathsf{Diag}(\mathbf{H}_{i_k}\mathbf{H}_{j_l}^\intercal).$$

Looking at the core calculations, we have to perform all these distinct vector multiplications, while $\mathbf{H}$ is perfectly random. To reduce the number of calculations, one could consider using a more structured or sparse matrix $\mathbf{H}$. Indeed, with such a matrix, certain steps could be avoided and, consequently, the costs of the total multiplication would be reduced. This is the idea behind what

was presented in Section 5.1, to use very sparse matrices to reduce the computation cost (even if that was still not enough). In what follows we will not suppose the matrix $\mathbf{H}$ to be sparse, but suppose that it is structured. More precisely, we will consider a random matrix on a special ring $\mathcal{R}$ which achieves some properties. The ring will convey the structure.

5.3 A PCG for OLE Framework

In this section, we describe a general framework for overcoming the efficiency problem encountered in the previous section. We recommend reading Section 2.4.2.2 for a better understanding of this section. What follows is an abstraction of the framework used by [BCGI+20b] and [BCCD23; BBCC+24] to produce a PCG scheme for OLE correlations.

5.3.1 *Appropriate* Ring

We describe a certain class of ring $\mathcal{R}$ upon which we know how to create a PCG.

Definition 5.3.1. *We say that a ring $\mathcal{R}$ is* appropriate *when:*

- *$\mathcal{R}$ is equipped with a structure of $\mathbb{F}_q$ vector space, inducing a canonical notion of* sparsity *after the choice of a basis, as explained in Section 2.4.2.2.*
- *$\mathcal{R}$ is isomorphic[3] to a product of n copies of $\mathbb{F}_q$.*
- *A variant of Syndrome Decoding is valid in $\mathcal{R}$, and we call it $\mathcal{R}$-SD (see Definition 2.4.9).*
- *The multiplication of two elements of $\mathcal{R}$ can be performed efficiently.*
- *The sparsity defined on $\mathcal{R}$ enjoys* weak stability*: the result of a multiplication of two t-sparse elements shall produce a t'-sparse element, $t' \leqslant t^2$.*

Theorem 5.3.1. *Let $\mathcal{R}$ be an* appropriate *ring. We assume that we have access to an* SPFSS *primitive, which is a secure* FSS *scheme for sums of point functions. Then there exists a generic scheme to construct a* PCG *to produce one* OLE *correlation over $\mathcal{R}$, and the protocol is described in Figure 5.3.1.*

The PCG construction can be summed up as follows:

1. Choose an *appropriate* ring $\mathcal{R}$, and a public element $a \in \mathcal{R}$.
2. The parties generate private t-sparse elements $(e^0_\sigma, e^1_\sigma) \in \mathcal{S}(\mathcal{R}, t)$
3. The parties compute $x_\sigma = ae^0_\sigma + e^1_\sigma$. According to the specific associated Syndrome Decoding assumption x_σ appears to be random.
4. The parties use Function Secret Sharing (see Section 2.5.3.5) to obtain shares of the products (the $e^0_0 \cdot e^0_1, e^1_0 \cdot e^0_1, e^0_0 \cdot e^1_1, e^1_0 \cdot e^1_1$). Thanks to the *weak stability*, these products are t^2-sparse.
5. The parties use their shares of the sparse products to obtain a secret sharing of $x_0x_1 = a^2(e^0_0e^0_1) + a(e^1_0e^0_1 + e^0_0e^1_1) + e^1_0e^1_1$.

This protocol is typical of the general approach to create a *silent* PCG. Each party creates an element x_σ with a hidden sparse structure. The structure enables the construction of the correlation from scratch because of the very powerful Functions Secret Sharing (FSS) primitives. Because of the hidden structure of x_0 and x_1, the product x_0x_1 also has a hidden structure. More precisely, in this case, we have $x_0x_1 = a^2(e^0_0e^0_1) + a(e^1_0e^0_1 + e^0_0e^1_1) + e^1_0e^1_1$. This expression is linear in the

3. as a ring

products of the sparse elements $(e_0^i e_1^j)_{i,j\in\{0,1\}}$. Therefore, if there exists a method for sharing these products, then we obtain a method to compute compressed additive shares of $z_\sigma = x_0 x_1$. This is exactly what FSS can offer, as it provides a compressed seed that can be expanded into a sharing of the sparse elements. We guarantee that the elements x_σ appear pseudorandom because of the $\mathcal{R}$-SD assumption over the chosen $\mathcal{R}$ holds.

Remark 5.3.1 (Compression Factor)**.** In the informal presentation above we have used the standard and simple version of Syndrome Decoding of the form $x_\sigma = a e_\sigma^0 + e_\sigma^1$. A more general version, used in the protocol described in Figure 5.3.1, consists of using the more general module version of the assumption, mentioned in Definition 2.4.6. In this version, the players hold vectors $\mathbf{e_\sigma} = (e_\sigma^0, \cdots, e_\sigma^{c-1})^\intercal$, and vectors $\mathbf{a_\sigma} = (1, a_1 \ldots, a_{c-1})^\intercal$ with $c \leqslant 1$ denoting the *compression factor*. The assumption then states that

$$\langle \mathbf{a}, \mathbf{e_\sigma} \rangle = e_\sigma^0 + e_\sigma^1 a_1 + \cdots + e_\sigma^{c-1} a_{c-1}$$

is indistinguishable from random. The influence of the compression factor will be discussed in Sections 5.5 and 5.6.

We give thereafter a proof of Theorem 5.3.1.

Proof. Refer to Figure 5.3.1 for the definition of the different variables. First, we argue for the *correcteness* of the protocol. Remark that $\mathbf{u} = \mathbf{u_0} + \mathbf{u_1}$, which therefore can be rewritten as $\mathbf{u} = \mathbf{e_0} \otimes \mathbf{e_1}$. Each entry of $\mathbf{u}$ is therefore equal to one of the $e_0^i e_1^j$. Moreover, observe that

$$z_0 + z_1 = \langle \mathbf{a} \otimes \mathbf{a}, \mathbf{u_0} + \mathbf{u_1} \rangle = \langle \mathbf{a} \otimes \mathbf{a}, \mathbf{e_0} \otimes \mathbf{e_1} \rangle = \langle \mathbf{a}, \mathbf{e_0} \rangle \cdot \langle \mathbf{a}, \mathbf{e_1} \rangle = x_0 \cdot x_1.$$

The next-to-last equality is straightforward to check. Note that here, $\langle \mathbf{a}, \mathbf{e_\sigma} \rangle$ is a $\mathcal{R}$-SD sample, with fixed random $\mathbf{a}$ and independent secret $\mathbf{e_\sigma}$.

Now, we argue for the *security reduction* of the proof. The subsequent analysis follows the same path as the original analysis of [BCGI+20b], but adapted to our more general study case. Because the proof is symmetrical, we prove the security reduction only in the case $\sigma = 1$. Consider the pair $(\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda)$ associated with the expanded outputs (x_0, z_0) and (x_1, z_1). The security guarantee of the PCG requires that

$$\{(\mathsf{k}_1, x_0, z_0)\} \approx \left\{(\mathsf{k}_1, \tilde{x_0}, \tilde{z_0}) \mid \tilde{x_0} \xleftarrow{\$} \mathcal{R}, \tilde{z_0} = \tilde{x_0} \cdot x_1 - z_1\right\}.$$

This result can be shown via a sequence of hybrid distributions.

First, remark that $\{(\mathsf{k}_1, x_0, z_0)\} \approx \{(\mathsf{k}_1, x_0, x_0 \cdot x_1 - z_1)\}$. What we would like to do here is to replace x_0 by $\tilde{x_0}$ using the $\mathcal{R}$-SD assumption. However, we cannot do that directly because $\mathsf{k}_1 = ((K_\sigma^{i,j})_{i,j\in[0\ldots c)}, (\mathbf{A_\sigma^i}, \mathbf{c_\sigma^i})_{i\in[0\ldots c)})$ depends directly on x_0. Nevertheless, what we can do is first to replace, step by step, the FSS keys $K_1^{i,j}$ in k_1 with simulated keys generated only with the range and the domain of the function. Because we suppose the FSS scheme is secure, we maintain indistinguishability at each step. Then, when all the keys have been replaced by simulated keys, we can replace x_0 by a fresh $\tilde{x_0} \xleftarrow{\$} \mathcal{R}$, and we obtain $\{(\tilde{\mathsf{k}_1}, \tilde{x_0}, \tilde{x_0} \cdot x_1 - z_1)\}$. This is still secure because we removed all dependencies on x_0 in k_1, and because of the $\mathcal{R}$-SD assumption. Finally, we transform back the simulated keys given by the FSS by the original keys. This is allowed by the assumption that the FSS scheme we use is secure (indistinguishability works both ways). Finally, we proved

General construction of Ring-OLE

PARAMETERS:
Security parameter λ, noise weight $t = t(\lambda)$, compression factor $c \geqslant 2$, $\mathcal{R}$ an *appropriate* ring, equipped with a basis $\mathbf{B}$. An FSS scheme (SPFSS.Gen,SPFSS.FullEval) for sums of t^2 point functions, with domain $[0 \dots |\mathbf{B}|)$ and range $\mathbb{F}_q$.

PUBLIC INPUT: $c-1$ random ring elements $a_1, \dots, a_{c-1} \in \mathcal{R}$.

$\mathsf{PCG.Gen}(1^\lambda)$:

1: **foreach** $\sigma \in \{0,1\},\ i \in [0 \dots c)$:
  1.1: $\mathbf{A}^{\boldsymbol{i}}_{\boldsymbol{\sigma}} \leftarrow (b_1, \dots, b_t)_{b_i \in \mathbf{B}}$ and $\mathbf{c}^{\boldsymbol{i}}_{\boldsymbol{\sigma}} \leftarrow (\mathbb{F}_q^*)^t$.
    ▷ Each pair $\mathbf{A}^{\boldsymbol{i}}_{\boldsymbol{\sigma}}, \mathbf{c}^{\boldsymbol{i}}_{\boldsymbol{\sigma}}$ defines a sparse element of $\mathcal{R}$
    ▷ The former indicates the elements of the basis that describes it.
    ▷ The latter their associated coefficients.
  1.2: Output $\mathbf{x}_{\boldsymbol{\sigma}}, \mathbf{z}_{\boldsymbol{\sigma}}$.
2: **foreach** $i, j \in [0 \dots c)$:
  2.1: Sample FSS keys $(K_0^{i,j}, K_1^{i,j}) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, 1^n, \mathbf{A}^{\boldsymbol{i}}_{\mathbf{0}} \otimes \mathbf{A}^{\boldsymbol{j}}_{\mathbf{1}}, \mathbf{c}^{\boldsymbol{i}}_{\mathbf{0}} \otimes \mathbf{c}^{\boldsymbol{j}}_{\mathbf{1}})$.
    ▷ For all $i, j \in [0 \dots)$, $\mathbf{A}^{\boldsymbol{i}}_{\mathbf{0}} \otimes \mathbf{A}^{\boldsymbol{j}}_{\mathbf{1}}, \mathbf{c}^{\boldsymbol{i}}_{\mathbf{0}} \otimes \mathbf{c}^{\boldsymbol{j}}_{\mathbf{1}}$, represent
    ▷ the product of two sparse elements of $\mathcal{R}$.
3: Let $\mathsf{k}_\sigma = ((K_\sigma^{i,j})_{i,j \in [0 \dots c)}, (\mathbf{A}^{\boldsymbol{i}}_{\boldsymbol{\sigma}}, \mathbf{c}^{\boldsymbol{i}}_{\boldsymbol{\sigma}})_{i \in [0 \dots c)})$.
4: Output $(\mathsf{k}_0, \mathsf{k}_1)$.

$\mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma)$:

1: Parse k_σ as $((K_\sigma^{i,j})_{i,j \in [0 \dots c)}, (\mathbf{A}^{\boldsymbol{i}}_{\boldsymbol{\sigma}}, \mathbf{c}^{\boldsymbol{i}}_{\boldsymbol{\sigma}})_{i \in [0 \dots c)})$.
2: **foreach** $i \in [0 \dots c)$,
  2.1: Define the element of $\mathcal{S}(\mathcal{R}, t)$,

$$e_\sigma^i = \sum_{j \in [0 \dots t)} \mathbf{A}^{\boldsymbol{i}}_{\boldsymbol{\sigma}}[j] \cdot \mathbf{c}^{\boldsymbol{i}}_{\boldsymbol{\sigma}}[j].$$

3: Compute $x_\sigma = \langle \mathbf{a}, \mathbf{e}_{\boldsymbol{\sigma}} \rangle$, where $\mathbf{a} = (1, a_1, \cdots, a_{c-1})^\intercal, \mathbf{e}_{\boldsymbol{\sigma}} = (e_\sigma^0, \cdots, e_\sigma^{c-1})^\intercal$.
4: **foreach** $i, j \in [0 \dots c)$,
  4.1: Compute $u_{\sigma, i+cj} \leftarrow \mathsf{SPFSS.FullEval}(\sigma, K_\sigma^{i,j})$ and view it as a c^2 vector of elements in $\mathcal{S}(\mathcal{R}, t^2)$.
    ▷ Let $\mathbf{u}_{\boldsymbol{\sigma}} = (u_{\sigma, i+cj})_{i,j \in [0 \dots)}$
5: Compute $z_\sigma = \langle \mathbf{a} \otimes \mathbf{a}, \mathbf{u}_{\boldsymbol{\sigma}} \rangle$.
6: Output x_σ, z_σ.

Figure 5.3 – PCG for OLE over an *appropriate* $\mathcal{R}$.

$$\left\{(\mathsf{k}_1, x_0, z_0)\right\} \approx \left\{(\mathsf{k}_1, \tilde{x_0}, \tilde{z_0}) \mid \tilde{x_0} \xleftarrow{\$} \mathcal{R}, \tilde{z_0} = \tilde{x_0} \cdot x_1 - z_1\right\},$$

and this concludes the proof.

□

Theorem 5.3.2. *Given an* appropriate *ring* $\mathcal{R} \simeq \mathbb{F}_q^n$*, there exists a* PCG *construction based on* $\mathcal{R}$-SD *producing* n *independent* OLE*s over* $\mathbb{F}_q$*.*

Proof. We have already proven that when working over an *appropriate* ring $\mathcal{R}$, the protocol described on Theorem 5.3.1 achieves creating a single OLE over $\mathcal{R}$. To obtain numerous fresh instances of the OLE correlations, the core idea of the construction is to use the isomorphism ϕ between $\mathcal{R}$ and $\mathbb{F}_q^n$. The existence of such an isomorphism is a condition for $\mathcal{R}$ being *appropriate*. Then, because of the pseudorandomness of (x_σ, z_σ), all the entries of $\phi(x_\sigma, z_\sigma)$ computed by the isomorphism are indistinguishable from random, and satisfy still the OLE correlation. Thus, this concludes the proof. □

Up to now, the literature has come up with two distinct constructions, based on two different rings $\mathcal{R}$.

- In [BCGI+20b], the authors chose $\mathcal{R}$ to be a quotient ring of $\mathbb{F}_q[X]/P(X)$ for a polynomial P that split perfectly.
- In [BCCD23], we chose $\mathcal{R}$ to be a group algebra $\mathbb{F}_q[\mathbb{G}]$, for a specific group $\mathbb{G}$.

In Section 5.4, we discuss the choice of $\mathcal{R}$, and the limitations of the constructions.

5.3.2 Additional Property

We prove that the protocol satisfies an important property.

Proposition 5.3.1. *The* PCG *thus defined is a* programmable PCG *(see Section 3.3).*

Proof. For the formal definition of programmable PCG, refer to Appendix A. Some adjustments are needed to show properly that the PCG is indeed programmable. Our goal is to show that players can decide the value of their first input (in this case x_σ), by controlling some additional inputs (ρ_σ). Formally, we want to show the existence of a computable function $\psi_\sigma : \{0,1\}^* \to (\mathcal{S}(\mathcal{R}, t))^n$, such that

$$\Pr_{\substack{\rho_0, \rho_1 \xleftarrow{\$} \{0,1\}^\kappa, (k_0, k_1) \leftarrow \mathsf{PCG.Gen}(1^\lambda, \rho_0, \rho_1) \\ (R_0, S_0) \leftarrow \mathsf{PCG.Expand}(0, k_0), \\ (R_1, S_1) \leftarrow \mathsf{PCG.Expand}(1, k_1)}} \left[\, R_0 = \psi_0(\rho_0), \quad R_1 = \psi_1(\rho_1) \,\right] \geqslant 1 - \mathsf{negl}(\lambda).$$

Let $\rho_\sigma = \left\{\mathbf{A}_{\boldsymbol{\sigma}}^{\boldsymbol{i}}, \mathbf{c}_{\boldsymbol{\sigma}}^{\boldsymbol{i}}\right\}$ be the additional input. It is a representation of the vector of elements in $\mathcal{S}(\mathcal{R}, t)$, $\mathbf{e}_{\boldsymbol{\sigma}} = (e_\sigma^0, \ldots, e_\sigma^{c-1})$. x_σ is defined as $x_\sigma = \langle \mathbf{a}, \mathbf{e}_{\boldsymbol{\sigma}} \rangle$. This directly defines the two functions ψ_σ, and proves the programmability property of a *programmable* PCG. Concerning the programmable security, we provide a proof once again from a series of hybrid distributions, exactly in the same fashion we already did for the security reduction of the construction in the proof of Section 5.3.1.

The proof is symmetrical depending on the choice of σ. Hereafter, we suppose that $\sigma = 1$. We want to show that the following distributions

$$\left\{ (\mathsf{k}_1, (\rho_0, \rho_1)) \;\middle|\; \rho_0, \rho_1 \xleftarrow{\$} \{0,1\}^\kappa, (\mathsf{k}_0, \mathsf{k}_1) \leftarrow \mathsf{PCG.Gen}(1^\lambda, \rho_0, \rho_1) \right\} \text{ and}$$
$$\left\{ (\mathsf{k}_1, (\rho_0, \rho_1)) \;\middle|\; \rho_0, \rho_1, \tilde{\rho_0} \xleftarrow{\$} \{0,1\}^\kappa, (\mathsf{k}_0, \mathsf{k}_1) \leftarrow \mathsf{PCG.Gen}(1^\lambda, \tilde{\rho_0}, \rho_1) \right\}$$

are computationally indistinguishable. The different steps of the series of hybrid distributions are as follows:

- Add $\tilde{\rho_0}$ to be sampled randomly.
- For $0 \leqslant i, j \leqslant c-1$, replace each FSS key $K_1^{i,j}$ of k_1 by a fresh randomly simulated key of the same range and domain. Under the assumption that our FSS scheme is secure, all the hybrid distributions are indistinguishable from one another.
- Using the local $\mathcal{R}$-SD assumption, replace $\mathsf{PCG.Gen}(1^\lambda, \rho_0, \rho_1)$ by $\mathsf{PCG.Gen}(1^\lambda, \tilde{\rho_0}, \rho_1)$.
- We transform back the simulated keys of k_1 by their original, thanks to the FSS security assumption (again, the indistinguishability works in both directions).

This concludes the proof. □

5.4 Choice of the Ring $\mathcal{R}$

In this section, we discuss the different candidates $\mathcal{R}$ that could be used in Figure 5.3.1. We recommend reading Sections 2.4.2.2 and 2.4.3 for a better understanding of this section. We will present the two choices chronologically: first, we will suppose that $\mathcal{R} = \mathbb{F}_q[X]/(P(X))$, for a polynomial $P \in \mathbb{F}_q[X]$ that splits into distinct monomials over $\mathbb{F}_q$, then we will consider $\mathcal{R} = \mathbb{F}_q[\mathbb{G}]$, that is, a group algebra for some abelian group $\mathbb{G}$.

5.4.1 Using $\mathcal{R} = \mathbb{F}_q[X]/(P(X))$

Here we focus on the original construction by [BCGI+20b]. In this construction, the authors chose to take $\mathcal{R}$ to be a quotient ring:

$$\mathcal{R} = \mathbb{F}_q[X]/(P(X)),$$

where P denotes a polynomial of degree n that can completely *split*, meaning that it verifies $P(X) = \prod_{i=0}^{n-1}(X - r_i)$ for distinct r_i's in $\mathbb{F}_q$. We will present why this choice is interesting, and what its limitations are.

Proposition 5.4.1. $\mathcal{R} = \mathbb{F}_q[X]/(P(X))$ *is* almost appropriate*: it satisfies all the conditions to be an* appropriate *ring except the last one (*weak stability*).*

Proof. There are four different requirements to prove.

1. $\mathbb{F}_q[X]/(P(X))$ is an $\mathbb{F}_q$-vector space structure, and therefore, it possesses a canonical notion of sparsity: the monomials form a basis of $\mathbb{F}_q[X]/(P(X))$, and a polynomial $e \in \mathbb{F}_q[X]/(P(X))$ is said to be t-sparse when it contains no more than t monomials in its description.

2. Because we supposed that P splits completely into linear factors, we obtain that $\mathbb{F}_q[X]/(P(X)) \simeq \mathbb{F}_q^n$ using the Chinese Remainder Theorem.
3. The equivalent of Syndrome Decoding in $\mathbb{F}_q[X]/(P(X))$ is then the standard Ring Syndrome Decoding assumption (or Ring-LPN in the literature). This assumption has been analyzed in various works ([BL12; DP12; LP15; GJL15]) but not exactly in the case of a polynomial P that splits completely. The authors provided an *ad hoc* analysis of this assumption, exactly tailored to their needs.
4. There has been a long tradition of works focused on performing very fast polynomial multiplications in $\mathbb{F}_q[X]/(P(X))$, by using FFT, breaking the barrier of n^2.

□

Unfortunately, $\mathbb{F}_q[X]/(P(X))$ does not satisfy entirely *weak stability*, which guarantees that a product of two sparse elements is sparse. The authors address this problem by tweaking the protocol slightly: they generate sparse elements e_0, e_1 of $\mathbb{F}_q[X]/(P(X))$, but they consider them living in $\mathbb{F}_q[X]$, in which they are still sparse. In $\mathbb{F}_q[X]$, *weak stability* is trivially achieved. From there, the authors performed the FSS scheme over the product e_0e_1 seen in $\mathbb{F}_q[X]$ (of maximum degree known $= 2n$). Each party obtains a $u' = \mathsf{SPFSS}(e_0e_1) \in \mathbb{F}_q[X]$, which they can then transform into $u = u' \mod P$. Because of the modulo is linear, it is as if we could use FSS for non-sparse polynomials. As the goal was to obtain an additive sharing of the product e_0e_1, the protocol can continue unchanged. The security is not threatened by the change, nor is this efficiency.

The cost of generating 2^{20} OLEs was estimated to be 10 seconds [BCGI+20b, Section 9], resulting in approximately 10^5 OLEs produced per seconds in this context. This is still about 2 orders of magnitude slower than the state of the art concerning OT.

5.4.1.1 A Tedious Constraint

The scheme suffers from a major limitation, stemming inherently from the structure of the ring. $\mathbb{F}_q[X]/(P(X)) \simeq \mathbb{F}_q^n$ by the Chinese Remainder theorem. n is the degree of the polynomial, but also the number of distinct roots of P, and finally the number of OLEs generated. This means that in order to generate n OLEs, we need to have n distinct roots in $\mathbb{F}_q$. This is possible only if there is *enough space* in $\mathbb{F}_q$, therefore

$$q \geqslant n.$$

We can convince ourselves that it is a tedious constraint. Indeed, recall from Section 2.5.4.2 that standard functions of interest require $n \sim 2^{30}$. Therefore, in the case $\mathcal{R} = \mathbb{F}_q[X]/(P(X))$, the protocol produces OLEs only over very large finite fields.

Remark 5.4.1 (About the choice of polynomial)**.** As stated in the description of $\mathbb{F}_q[X]/(P(X))$, we want to choose a polynomial P of degree n that splits entirely, in order to obtain as many copies of $\mathbb{F}_q$ as possible. Nevertheless, one must be cautious when deciding on which polynomial P to use. In [BCCD23] we remarked that some restrictions, not noticed by [BCGI+20b], have to be considered: namely, one could think of using the polynomial $P(X) = X^q - X$ which splits entirely and has for roots all the elements in $\mathbb{F}_q$. This would be a natural choice in order to maximize the number of OLEs produced. Unfortunately, this does not work: consider an $\mathcal{R}$-SD sample $(a, ae + f)$, $a \in \mathcal{R}$ and $e, f \in \mathcal{S}(\mathbb{F}_q[X]/(P(X)), t)$ t-sparse elements of $\mathbb{F}_q[X]/(P(X))$. Since there is no constant coefficient in $X^q - X$, multiplying two non-constant monomials will never result in a polynomial with a non-zero constant term. Therefore, the evaluation at 0 and the multiplication in $\mathbb{F}_q[X]/(P(X))$ commute. Consequently, $(ae + f)(0) = a(0)e(0) + f(0)$. Because e and f are sparse, their constant

coefficient is likely to be zero, and therefore, the evaluation at 0 is biased. Needless to say, this is not the case for a random element of $\mathbb{F}_q[X]/(P(X))$. Note that it does not mean that this framework does not work. If one is a bit cautious and chooses $P(X) = X^{q-1} - 1$, then the previous attack does not apply; while being still able to split $P(X) = \prod_{r \in \mathbb{F}_q^*}(X - r)$. This comes at the (minor) cost of not maximizing the number of copies that $\mathbb{F}_q$ could provide.

5.4.2 Using $\mathcal{R} = \mathbb{F}_q[\mathbb{G}]$

In what follows, we aim to address one of the main issues that the previous constructions suffer from: the lower bound on q. Let $\mathcal{R}$ be a group algebra:

$$\mathcal{R} = \mathbb{F}_q[\mathbb{G}]$$

for $\mathbb{G}$ the following an abelian group:

$$\mathbb{G} = \prod_{i=1}^{n} \mathbb{Z}/(q-1)\mathbb{Z}.$$

Necessary background on group algebra and quasi-abelian codes is given in Section 2.4.3. Recall that given a group $\mathbb{G}$, $\mathbb{F}_q[\mathbb{G}]$ denotes the set of all the formal sums of elements of $\mathbb{F}_q[\mathbb{G}]$ with coefficients $a_i \in \mathbb{F}_q$. We will demonstrate the five different requirements to establish that $\mathbb{F}_q[\mathbb{G}]$ is an *appropriate* ring which includes proving the new security assumption, *i.e.* the QA-SD assumption.

Proposition 5.4.2. *$\mathcal{R} = \mathbb{F}_q[\mathbb{G}]$ comes with a canonical basis, given by the elements of the group $\mathbb{G}$ and therefore enjoys a canonical notion of sparsity. , it respects the* weak stability*: the multiplication of two t-sparse elements is at most t^2-sparse.*

Proof. An element $x \in \mathbb{F}_q[\mathbb{G}]$ is considered t-sparse if at most t of its coefficients a_i are non-zero.

We aim to prove that the product of two elements of $\mathcal{S}(\mathbb{F}_q[\mathbb{G}], t)$ is in $\mathcal{S}(\mathbb{F}_q[\mathbb{G}], t^2)$. Let e, f be two sparse elements. Let S_e, S_f be the two sets such that $S_e, S_f \subset \mathbb{G}, |S_e| = |S_f| = t$, and $e = \sum_{g \in S_e} e_g g$ and $f = \sum_{g \in S_e} f_g g$. Then, the product $e \cdot f$ can be expressed using only $S_e \bullet S_f \coloneqq \{gh \mid g \in S_e, h \in S_f\}$ as a basis. Since $|S_e \bullet S_f| < |S_e| \cdot |S_f| = t^2$, we deduce that *weak stability* is achieved. □

Proposition 5.4.3. *If $\mathbb{G}$ is a finite abelian group, $\mathcal{R} = \mathbb{F}_q[\mathbb{G}]$ is isomorphic to a product of $\mathbb{F}_q$, $q > 2$.*

Proof. It was proven in Proposition 2.4.3, that assuming that for any abelian group $\mathbb{G} = \prod_{i=1}^{n} \mathbb{Z}/q_i\mathbb{Z}$, $q_i \geqslant 3$, we have

$$\begin{aligned}\mathbb{F}_q[\mathbb{G}] = \mathbb{F}_q\left[\prod_{i=1}^{n} \mathbb{Z}/q_i\mathbb{Z}\right] &\simeq \mathbb{F}_q[\mathbb{Z}/q_1\mathbb{Z}] \otimes_{\mathbb{F}_q} \cdots \otimes_{\mathbb{F}_q} \mathbb{F}_q[\mathbb{Z}/q_n\mathbb{Z}] \\ &\simeq \bigotimes_{i=1}^{n} \mathbb{F}_q[X_i]/(X_1^{q_i} - 1) \simeq \mathbb{F}_q[X_1, .., X_n]/(X_1^{q_1} - 1, \ldots, X_n^{q_n} - 1).\end{aligned}$$

Note that we set $\mathbb{G} = \prod_{i=1}^{n} \mathbb{Z}/(q-1)\mathbb{Z}$. Thus, we have $\mathcal{R} = \mathbb{F}_q[\mathbb{G}] \simeq \mathbb{F}_q[X_1, \ldots, X_n]/(X_1^{q-1} - 1, \ldots, X_n^{q-1} - 1)$. The elements of $\mathbb{F}_q^*$ are precisely the roots of the polynomial $X_i^{q-1} - 1$. Therefore,

we can write $X_i^{q-1} - 1 = \prod_{a \in \mathbb{F}_q^*}(X_i - a)$, for all $1 \leqslant i \leqslant n$ and, by the Chinese Remainder Theorem, we get

$$\mathbb{F}_q[X_1, \ldots, X_n]/(X_1^{q-1} - 1, \ldots, X_n^{q-1} - 1) \simeq \prod_{i=1}^{(q-1)^n} \mathbb{F}_q.$$

□

Remark 5.4.2 (About the tedious constraint of $\mathbb{F}_q[X]/(P(X))$)**.** We showed in the previous section that $\mathcal{R} = \mathbb{F}_q[X]/(P(X))$ suffers from a limitation that restricts the construction to very large fields. This is no longer the case when using $\mathcal{R} = \mathbb{F}_q[\mathbb{G}]$. We generate $T = (q-1)^n$ OTs when using the CRT isomorphism, and it does not depend solely on q. Therefore, to produce OLEs over small fields, we can increase the size of the group n.

5.4.2.1 Security of QA-SD Assumption

It remains to prove that the current $\mathcal{R}$-SD assumption is valid for $\mathcal{R} = \mathbb{F}_q[\mathbb{G}]$. This leads us to examine the reliability of the QA-SD assumption, as defined in Section 2.4.3, which we restate below.

Definition 5.4.1 ((Decisional) Quasi-Abelian Syndrome Decoding assumption)**.** *Let $(n, k, t) = (n(\lambda), k(\lambda), t(\lambda))$ be parameters polynomial in the security parameter λ, and let $\mathbb{G}$ be an abelian group. The goal of the* decisional Quasi-Abelian Syndrome Decoding problem *is to distinguish, with a non-negligible advantage, between the distributions*

$$\begin{array}{lll} \mathcal{D}_0: & (a, s) & \text{where } a, s \xleftarrow{\$} \mathbb{F}_q[\mathbb{G}], \\ \mathcal{D}_1: & (a, a \cdot e_1 + e_2) & \text{where } a \xleftarrow{\$} \mathbb{F}_q[\mathbb{G}] \text{ and } e_i \xleftarrow{\$} \mathcal{S}(\mathbb{F}_q[\mathbb{G}], t)). \end{array}$$

The *Decisional Quasi-Abelian Syndrome Decoding over a ring* assumption, denoted by QA-SD, states that this problem is hard. That is, for every polynomial-time algorithm $\mathcal{A}$, it holds that

$$\left| \Pr_{a \xleftarrow{\$} \mathbb{F}_q[\mathbb{G}],\ (e_0,e_1) \xleftarrow{\$} \mathcal{S}(\mathbb{F}_q[\mathbb{G}],t)} [\, \mathcal{A}(a, ae_0 + e_1) = 0 \,] - \Pr_{(a,s) \xleftarrow{\$} \mathbb{F}_q[\mathbb{G}]} [\, \mathcal{A}(a, s) = 0 \,] \right| \leqslant \mathsf{negl}(\lambda).$$

where negl denotes a negligible function.

Following the template provided in Section 4.2, our objective is to demonstrate the security of these assumptions against linear attacks, as they encompass the majority of the attacks against a Syndrome Decoding like assumption.

Conjecture 5.4.1. *The* QA-SD *is secure against linear attacks.*

Proof. Let $\mathbf{H} = (\mathbf{I}_n \mid \mathbf{M}_a)$ be the matrix associated with the problem, as defined in Section 2.4.3, Definition 2.4.10, where $\mathbf{I}_n$ is the identity matrix of size n. In the case of the module variant, we would take $\mathbf{H} = (\mathbf{I}_n \mid \mathbf{M}_{a_1} \mid \cdots \mid \mathbf{M}_{a_{c-1}})$. We use here the classical equivalence (see Proposition 4.3.1) between resistance against linear tests and the high minimum distance of the code generated by $\mathbf{H}$. Our goal is to prove that with high probability over the choice of $\mathbf{H}$, the minimum distance of the code generated by $\mathbf{H}$ is large.

Several restricted versions of the QA-SD assumption have been previously analyzed. Remark that the assumption can be equivalent to a standard assumption for some specific choices of $\mathbb{G}$: when taking $\mathbb{G} = \{1\}$ the assumption is the standard SD assumption for random codes, that have been

studied for years. Recall from Proposition 2.4.2 that it is well known that random codes reach the Gilbert-Varshamov (GV) bound (you can refer to the works of [Pie67; BF02; Deb23a]). After the pure random code, the focus has been on whether an analog result could be achieved by quasi-cyclic code. This class of codes has been studied by Kasami [Kas74] and then Gaborit and Zémor [GZ06], in which they introduced a GV-like bound in order to prove that quasi-cyclic codes enjoy high minimal distance. Finally, in 2015, Fan and Lin [FL15] improved the result for every quasi-abelian codes.

Theorem 5.4.1 ([FL15, Theorem 2.1]). *Let $\mathbb{G}$ be a finite abelian group, and let $(\mathcal{C}_\ell)_\ell$ be a sequence of random quasi-$\mathbb{G}$ codes of length $\ell \in \mathbb{N}$ and rate $r \in (0,1)$. Let $\delta \in (0, 1 - \frac{1}{q})$. Then,*

$$\lim_{\ell \to \infty} \Pr\left[\frac{d_{min}(\mathcal{C}_\ell)}{|\mathbb{G}|} > \delta\ell\right] = \left\{ \begin{array}{ll} 1 & \text{if } r < 1 - h_q(\delta); \\ 0 & \text{if } r > 1 - h_q(\delta); \end{array} \right\},$$

and both limits converge exponentially fast. The above probability is taken over the uniform choice of a generator matrix $\mathbf{G}_\ell \in \mathbb{F}_q[\mathbb{G}]^{k \times \ell}$ of $\mathcal{C}_\ell$.

This result is asymptotic, as it is often the case in coding theory. Actually, the coefficient in the exponent depends on the size of the group $\mathbb{G}$, and therefore, the bigger the group, the more likely we are to get a good linear distance. Nonetheless, we could hope to improve this result in order to fit better our use case: in the theorem, while the rate is fixed, (k, ℓ) are growing; whereas we consider in our application a matrix with a fixed number of blocks (equal to c the compression factor). This is a lead for future work. Then, one has to be careful that having a matrix $\mathbf{H}$ with a nice minimal distance when considered over $\mathbb{F}_q[\mathbb{G}]$ entails a matrix $\mathbf{H}$ with a nice linear distance when seen over $\mathbb{F}_q$. In the case of random quasi-$\mathbb{G}$ codes, it is likely to occur. □

In addition to this result, group algebras offer efficient ways to perform multiplications.

Proposition 5.4.4. *Fast operations are possible in $\mathcal{R} = \mathbb{F}_q[\mathbb{G}]$.*

Proof. We defer the proof to Section 5.5 where we analyze in detail the multiplication over $\mathbb{F}_q$. □

All the conditions have been proven. Therefore,

Proposition 5.4.5. $\mathcal{R} = \mathbb{F}_q[\mathbb{G}]$ *is* appropriate.

Remark 5.4.3 (Link with $\mathcal{R} = \mathbb{F}_q[X]/(P(X))$). Following Remark 5.4.1, note that in the case $P(X) = X^{q-1} - 1$, $\mathbb{F}_q[X]/(P(X))$ is a group algebra, as it corresponds to $\mathbb{F}_q[\mathbb{G}]$ in the case $\mathbb{G} = \mathbb{F}_q^*$.

5.4.2.2 Limitation of the Construction

Using the QA-SD assumption and taking $\mathcal{R}$ to be the group algebra $\mathbb{F}_q[\mathbb{G}]$, $\mathbb{G} = \prod_{i=1}^{n} \mathbb{Z}/(q-1)\mathbb{Z}$, $q \geqslant 3$ has offered us the possibility to create a PCG to generate many OLEs over $\mathbb{F}_q$, for $q \geqslant 3$. Therefore, we have managed to shrink down the condition $q > n$, with n being the number of OLEs produced to $q \geqslant 3$, covering all cases except for $q = 2$. This is in fact inherent to the structure of the group algebra itself. Looking at $(\mathbb{F}_2)^n$ (product of n copies of $\mathbb{F}_2$) there is only one invertible element (the full 1 vector). We are looking for a group algebra that is isomorphic to $(\mathbb{F}_2)^n$. This means that there will be only one invertible element in the group algebra as well. On the other side, a group algebra has only one invertible element when $|\mathbb{G}| = n = 1$, meaning that we would produce only 1 OLE over $\mathbb{F}_q$. For this reason, the framework is interesting only when $q \geqslant 3$.

One might think that this is not a problem, as OLEs over $\mathbb{F}_2$ are just OTs, and we already have efficient constructions for OTs. While this is true, an important reason to push this specific PCG construction down to $q = 2$ is that it enjoys, as shown in Section 5.3, the programmability property, a property that known constructions of PCG for OT do not satisfy. As sketched in part Section 3.3 the programmability property is essential for silent secure computation both for the malicious setting or to do MPC with more than 2 players.

Next, we investigate other unfruitful choices of $\mathcal{R}$ to obtain the PCG over $\mathbb{F}_2$.

5.4.3 Other Possible $\mathcal{R}$

In this section, we question ourselves about other choices of $\mathcal{R}$ that may lead to interesting results and illustrate some attempts that end up getting broken. A natural candidate in order to reach production of OLEs over $\mathbb{F}_2$ is the set of boolean functions

$$\mathcal{R} = \mathbb{F}_2[X_1, \dots, X_n]/(X_1^2 - X_1, \dots, X_n^2 - X_n).$$

This structure has been deeply studied (mostly in symmetric cryptography) and is isomorphic to 2^n copies of $\mathbb{F}_2$, due to the fact that $X^2 - X = X(X + 1)$. As before, we can obtain fast operations in $\mathcal{R}$ thanks to the use of FFT (usually called fast Walsh or Hadamard transform in symmetric cryptography). Additionally, there exists a canonical notion of sparsity. For a function $f \in \mathcal{R}$, we can express $f(X_1, \dots, X_n) = \sum_{\mathbf{u} \in \mathbb{F}_2} a_{\mathbf{u}} X^{\mathbf{u}}$, with $a_{\mathbf{u}} \in \mathbb{F}_2$, and where $X^{\mathbf{u}} = {X_1}^{\mathbf{u}_0} X_2^{\mathbf{u}_2} \dots X_n^{\mathbf{u}_n}$. As before, we can consider an element to be sparse if it has few $a_{\mathbf{u}} \neq 0$. All in all, boolean functions seem to be a very solid candidate in order to achieve PCG for OLE over $\mathbb{F}_2$. Unfortunately, the construction does not support security analysis.

5.4.3.1 An Attack Against Boolean Function Syndrome Decoding

This attack is is, in fact, the very same attack idea presented in Remark 5.4.1. Concretely, consider being given an OLE sample $(a, as + e)$, with $a \xleftarrow{\$} \mathcal{R}$, and $s, e \xleftarrow{\$} \mathcal{S}(\mathcal{R}, t)$ two t-sparse elements of the set of boolean functions. There exists a very natural bias: since s and e are sparse, their constant term is very likely to be non-zero, and therefore, with high probability $s(0) = e(0) = 0$. Since there is no constant in $X^2 - X$, the multiplication and the evaluation in 0 commute, and therefore $(as + e)(0) = a(0)s(0) + e(0)$ which is 0 with high probability.

5.4.3.2 A Possible Solution

Taking a step back, what causes the bias here is the fact that a random monomial chosen among all the possible monomials is unlikely to be of small degree, moreover 0. An idea in order to thwart this problem would be to consider other noise distributions when sampling the error vectors. Indeed, if we force the sparse vector $e \in \mathcal{S}(\mathcal{R}, t)$ to have a constant number of monomials of each degree exactly, the previous attacks would not work: we can increase the probability that a constant coefficient appears in e, to be close to $1/2$. Then the sparse elements would not be biased toward 0 when evaluated in 0. When looking at the vector representation of such an error vector, a structure of *variable density* appears, as the number of monomials grows with the degrees at first, and then decreases again. Obviously, such a construction would request a tailored proof of syndrome decoding in the case the noise follows a *variable density* shape. This is similar to what will be done in Chapter 7, where we consider the *Variable Density Syndrome Decoding* VDLPN assumption. This question is still open but is a very exciting line of work to delve into.

5.5 Further Optimizations for the QA-SD Construction

In this section, we present different optimizations that allow the PCG based on QA-SD to be really efficient, and thus the construction to be usable in practice.

5.5.1 High-level Idea for Fast Operations in $\mathcal{R} = \mathbb{F}_q[\mathbb{G}]$

Going back to the construction described in Section 5.3, at the end of Figure 5.3.1, the parties obtain one OLE over $\mathcal{R}$. Recall that (x_σ, z_σ) are the output of party σ.

5.5.1.1 Fast Operations

A requirement for the ring $\mathcal{R}$ to be *appropriate* is to have fast operations. The question is then, for $x, y \in \mathcal{R} = \mathbb{F}_q[\mathbb{G}] \simeq \mathbb{F}_q[X_1, \ldots, X_n]/(X_1^{q-1} - 1, \ldots, X_n^{q-1} - 1)$, is $x \cdot y$ fast to compute? The existence of the isomorphism between $\mathcal{R}$ and $(\mathbb{F}_q)^n$ motivates us to seek for a Fast Fourier Transform (FFT). This is what was done in the case $\mathcal{R} = \mathbb{F}_q[X]/(P(X))$. The goal is to transform the elements of $\mathcal{R}$ in elements of $(\mathbb{F}_q)^T$, where $T = (q-1)^n$,compute the equivalent of the multiplication with the two elements in $(\mathbb{F}_q)^T$, and then by using the inverse isomorphism, recover the result of the multiplication in $\mathcal{R}$.

5.5.1.2 Fast Evaluations

Recall that in Theorem 5.3.2, we proved that it was possible to obtain a PCG for a single OLE over $\mathcal{R} = \mathbb{F}_q[\mathbb{G}]$. Further, we obtain a PCG for OLEs over $\mathbb{F}_q$ by using the Chinese isomorphism between $\mathbb{F}_q[\mathbb{G}]$ and $\mathbb{F}_q^T$, where $T = (q-1)^n$. Given $x \in \mathbb{F}_q[\mathbb{G}]$, the isomorphism returns the full evaluation of x seen as a polynomial over the full domain except zero. That is to say, from $(x_\sigma, z_\sigma) \in \mathcal{R} \simeq \mathbb{F}_q[X_1, \ldots, X_n]/(X_1^{q-1} - 1, \ldots, X_n^{q-1} - 1)$ given by Figure 5.3.1, we want to evaluate both of them over $(\mathbb{F}_q^\times)^n$. Therefore, we want this fast evaluation algorithm, both for the FFT and for the final evaluation in order to obtain the numerous OLEs over $\mathbb{F}_q$.

5.5.1.3 Working Directly with the Evaluations

To sum up, on one side our multiplications in the group algebra require the FFT to be fast, first in a multi-evaluation form, and then in the interpolation form. On the other side, what we desire as an output of the protocol is the evaluations of the polynomials x_σ, z_σ. That being said, a natural optimization is to work directly with the evaluations. For $f \in \mathcal{R}$, let $\mathsf{Eval}_n(f) = \big(f(x_1, \ldots, x_n) \mid (x_1, \ldots, x_n) \in (\mathbb{F}_q^\times)^n\big)$ be the list of all the possible evaluations. The evaluation isomorphism between $\mathcal{R}$ and $(\mathbb{F}_q)^T$ is a ring isomorphism such that $\mathsf{Eval}_n(x \cdot y) = \mathsf{Eval}_n(x) \odot \mathsf{Eval}_n(y)$ where $\odot$ denotes the component-wise or Schur product. In what follows we show how we can avoid the intermediate steps of the multi-evaluation-then-interpolation. Instead of constructing polynomials x_σ, z_σ, we focus on constructing directly their polynomial evaluations.

Instead of giving the parties the description of the coefficients of the polynomials $a_i \in \mathbf{a}$, we can give them the vectors of all the evaluations of all the polynomials, i.e., giving them $\mathsf{Eval}_n(a_i)$, for all $1 \leqslant i \leqslant c-1$. Because we can write $x_\sigma = e_\sigma^0 + e_\sigma^1 a_1 + \cdots + e_\sigma^{c-1} a_{c-1}$, it follows that all the evaluations of x_σ can be obtained from $\mathsf{Eval}_n(e_\sigma^i)$ and $\mathsf{Eval}_n(a_i)$. As a result, computing $\mathsf{Eval}_n(x_\sigma)$ amounts to compute $\mathsf{Eval}_n(e_\sigma^i)$. For the computation of the evaluation of sparse elements $(e_\sigma^i)_{0 \leqslant i \leqslant c-1}$, we will use an FFT algorithm explained in the next subsection. We consider also taking advantage of the sparsity of e_σ^i, because intuitively there are less evaluation to perform, nevertheless the FFT remains counterintuitively better, when just doing a straightforward full evaluation. More refined strategies might exist.

The computation of $\mathsf{Eval}_n(z_\sigma)$ is a little trickier. As mentioned above, $x_0 \cdot x_1$ can be seen as a function of degree 2 in $\mathbf{e_0}, \mathbf{e_1}$, and therefore a function of degree 1 in $(\mathbf{e_0} \otimes \mathbf{e_1})$, with constant coefficients depending solely on $\mathbf{a}\otimes\mathbf{a}$. Because $\mathsf{Eval}_n(a_i), 0 \leqslant i \leqslant c-1$ is already given to the parties, the computation of the coefficients from $\mathbf{a}\otimes\mathbf{a}$ can be obtained using only c^2 multiplications. It remains to compute the evaluations of the additive shares of the polynomials $(e_0^i \cdot e_1^j)_{0\leqslant i,j\leqslant c-1}$. There are c^2 such polynomials shared among the players, and we can view each share as a random polynomial. Therefore, each player has to compute the full evaluation of c^2 pseudorandom polynomials. Once again, we will use FFT in this case, and it is explained next subsection. In Figure 5.4 we represent the PCG framework tailored to our setting, in the specific case $q = 4$.

5.5.2 Fast Evaluation over $\mathbb{F}_q$

Given a polynomial P with n variables, the parties want to compute $\mathsf{Eval}_n(P)$, that is, to evaluate P over $\left(\mathbb{F}_q^\times\right)^n$ where $\mathbb{F}_q^\times$ is the field of q elements without 0. Here, we adapt the standard divide-and-conquer style algorithm (see for example the seminal work of [CT65]) to our case. Remark that

$$P(X_1, \ldots, X_n) = \sum_{i=0}^{q-2} X_n^i P_i(X_1, \ldots, X_{n-1}).$$

Instead of classically dividing our problem into 2 sub-problems, we divide it into $q-1$ sub-problems. This is a q-ary generalization of a standard FFT algorithm adapted to our case. Then:

$$\mathsf{Eval}_n(P) = \left\{ \sum_{k=0}^{(q-1)^{n-1}} x^i \mathsf{Eval}_{n-1}(P_i)[k] \mid 0 \leqslant i \leqslant q-2, x \in \mathbb{F}_q^\times \right\}$$

Denote by $\mathcal{C}(\mathsf{Eval}_n(P))$ the number of operations carried out to obtain all the $(q-1)^n$ evaluations on the set $\mathbb{F}_q^\times$. Then we have $\mathcal{C}(\mathsf{Eval}_n(P)) = (q-1)\mathcal{C}(\mathsf{Eval}_{n-1}(P)) + (q-1)\cdot(q-1)^{n-1}$, which leads us to $\mathcal{C}(\mathsf{Eval}_n(P)) = n\cdot(q-1)^n$. This quick back-of-the-envelope calculation gives an estimate of the asymptotic complexity. A more careful count would focus on the cost of the distinct operations implemented in practice. Section 5.6 offers a more involved analysis of the evaluation algorithm in the precise case of $q = 4$, as done in [BBCC+24].

5.5.3 Regular Syndrome Decoding Optimization

Another common optimization comes from the FSS part. While we present this optimization in the case of QA-SD, with $\mathcal{R} = \mathbb{F}_q[\mathbb{G}]$, it was also used for $\mathcal{R} = \mathbb{F}_q[X]/(P(X))$ in [BCGI+20b]. The optimization requires that the number of errors in each subvector t divides the number $T = (q-1)^n$. Let $\mathbb{G}$ be an abelian group of size n. We view the elements of $\mathbb{F}_q[\mathbb{G}]$ in their vector representation, given by the isomorphism between $\mathbb{F}_q[\mathbb{G}]$ and $\mathbb{F}_q^T$. Since the first work on PCGs ([BCGI18; BCGI+19a]), it has been noted that in order to efficiently share a sparse vector of weight t, one could use *regular sparse vectors*.

Let $\mathbf{e_1}, \mathbf{e_2} \in \mathcal{S}(\mathbb{F}_q^T, t)$, representing two sparse elements in $\mathbb{F}_q[\mathbb{G}]$. We denote by $\mathbf{e}$ the vector corresponding to the product of those two elements. As stated in Proposition 5.4.2, $\mathbf{e}$ is a vector with at most t^2 non-zero elements. Recall from Definition 2.5.10 that, given a t^2-sparse vector $\mathbf{e} \in \mathcal{S}(\mathbb{F}_q^T, t^2)$, and an FSS scheme for point functions, one can decompose $\mathbf{e}$ as a sum of t^2 unit vectors, and call the DPF protocol on each one of them. As a result, players obtain t^2 sharing of unit elements, and thanks to the linearity of additive sharing, deduce a sharing of $\mathbf{e}$. This construction

Specific construction of $\mathsf{QA\text{-}SD}_{\mathsf{OLE}}$ for FOLEAGE

PARAMETERS: Noise weight $t = t(\lambda)$, compression factor c, ring $\mathcal{R} = \mathbb{F}_4[X_1, \ldots, X_n]/(X_1^3 - 1, \ldots, X_n^3 - 1)$. A SPFSS scheme $\mathsf{SPFSS} = (\mathsf{SPFSS.Gen}, \mathsf{SPFSS.FullEval})$ for sums of t^2 point functions, with domain $[0 \ldots 3^n)$ and range $\mathbb{F}_4$.

PUBLIC INPUT: $c - 1$ vectors of length 3^n over $\mathbb{F}_4$, corresponding to the result of $\mathsf{Eval}_n(a_i)$, for $a_1, \ldots, a_{c-1} \in \mathcal{R}$, therefore the full evaluation of the c elements a_i.

$\mathsf{PCG.Gen}(1^\lambda)$:

1: **foreach** $\sigma \in \{0,1\},\ i \in [0 \ldots c)$:

1.1: Sample random $\mathbf{p}_{\boldsymbol{\sigma}}^{\boldsymbol{i}} \xleftarrow{\$} \{(p_{\sigma,1}^i, \ldots, p_{\sigma,t}^i) \mid p_{\sigma,j}^i \in \{0,1,2\}^n\}$, and $\mathbf{v}_{\boldsymbol{\sigma}}^{\boldsymbol{i}} \xleftarrow{\$} (\mathbb{F}_4^\times)^t$.
▷ Each pair $(\mathbf{p}_{\boldsymbol{\sigma}}^{\boldsymbol{i}}, \mathbf{v}_{\boldsymbol{\sigma}}^{\boldsymbol{i}})$ represent a sparse element of $\mathcal{R}$.
▷ $\mathbf{p}_{\boldsymbol{\sigma}}^{\boldsymbol{i}}[j]$ is the power of a monomial, and $\mathbf{v}_{\boldsymbol{\sigma}}^{\boldsymbol{i}}[jm]$ the associated coefficient.
▷ [Optimization]: $\mathbf{p}_{\boldsymbol{\sigma}}^{\boldsymbol{i}}$ can be sampled from regular noise distribution. See Section 5.5.3.

2: **foreach** $i, j \in [0 \ldots c)$:

2.1: Sample FSS keys $(K_0^{i,j}, K_1^{i,j}) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, 1^n, \mathbf{p}_{\mathbf{0}}^{\boldsymbol{i}} \boxplus \mathbf{p}_{\mathbf{1}}^{\boldsymbol{j}}, \mathbf{v}_{\mathbf{0}}^{\boldsymbol{i}} \otimes \mathbf{v}_{\mathbf{1}}^{\boldsymbol{j}})$.
▷ If using regular noise as an optimization, then
▷ SPFSS is for the sum of t point functions with domain $[0, \ldots, 3^n/t)$.

3: Let $\mathsf{k}_\sigma = ((K_\sigma^{i,j})_{i,j \in [0 \ldots c)}, (\mathbf{p}_{\boldsymbol{\sigma}}^{\boldsymbol{i}}, \mathbf{v}_{\boldsymbol{\sigma}}^{\boldsymbol{i}})_{i \in [0 \ldots c)})$.

4: Output $(\mathsf{k}_0, \mathsf{k}_1)$.

$\mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma)$:

1: Parse k_σ as $((K_\sigma^{i,j})_{i,j \in [0 \ldots c)}, (\mathbf{p}_{\boldsymbol{\sigma}}^{\boldsymbol{i}}, \mathbf{v}_{\boldsymbol{\sigma}}^{\boldsymbol{i}})_{i \in [0 \ldots c)})$.

2: **foreach** $i \in [0 \ldots c)$,

2.1: Define over $\mathbb{F}_4$ the polynomial:

$$e_\sigma^i(X) = \sum_{j \in [0 \ldots t)} \mathbf{v}_{\boldsymbol{\sigma}}^{\boldsymbol{i}}[j] \cdot \mathbf{X}^{\mathbf{p}_{\boldsymbol{\sigma}}^{\boldsymbol{i}}[j]}$$

2.2: Compute $\mathsf{Eval}_n(e_\sigma^i)$.

3: From $\mathsf{Eval}_n(e_\sigma^i)$ and $\mathsf{Eval}_n(a_i)$, compute $\mathsf{Eval}_n(x_\sigma)$.

4: **foreach** $i, j \in [0 \ldots c)$,

5.1: Compute $u_{\sigma,i+cj} \leftarrow \mathsf{SPFSS.FullEval}(\sigma, K_\sigma^{i,j})$ and view it as a c^2 vector of element in $\mathcal{R}$.

5: **foreach** $j \in [0 \ldots c^2)$,

6.1: Compute $\mathsf{Eval}_n(u_{\sigma,j})$.

6: Compute $\mathsf{Eval}_n(z_\sigma)$, with $z_\sigma = \langle \mathbf{a} \otimes \mathbf{a}, \mathbf{u}_{\boldsymbol{\sigma}} \rangle$, for $\mathbf{a} = (1, a_1, \ldots, a_{c-1})^\intercal$, $\mathbf{u}_{\boldsymbol{\sigma}} = (u_{\sigma,i+cj})_{i,j}$.

7: Output $(\mathsf{Eval}_n(x_\sigma), \mathsf{Eval}_n(z_\sigma))$.

Figure 5.4 – QA-SD-based PCG for OLE over $\mathcal{R}$ from evaluations of functions.

leads to a key size in $O(t^2 \cdot (\log(n)\lambda + \log(q)))$ and the number of operations of the full domain evaluation is dominated by $t^2 n$ group operations and evaluations of a PRG.

Consider now vectors $\mathbf{e_1}, \mathbf{e_2} \in \mathbb{F}_q^T$, such that both vectors can be written as $(\mathbf{u_1} /\!/ \cdots /\!/ \mathbf{u_t})$ where $\mathbf{u_i}$ denotes a unit vector of size n/t and $||$ denotes the vertical concatenation. We say that $\mathbf{e_1}$ and $\mathbf{e_2}$ are *blockwise regular*(see Definition 2.4.8). Let $\mathbf{e} \in \mathcal{S}(\mathbb{F}_q^T, t^2)$ be the vector representing the product of $\mathbf{e_1 e_2}$. Write $\mathbf{e} = (\mathbf{e^{(1)}} /\!/ \cdots /\!/ \mathbf{e^{(t)}})$. Now we want to show that for all i, $\mathbf{e^{(i)}}$ contains at most t non-zero coordinates. To see this, we will name the different blocks $B_0, \ldots, B_{t-1}$ each one being of size $(q-1)^n/t$. We denote by $\mathbf{X^p} \coloneqq X_1^{p_1} \cdot X_2^{p_2} \cdots X_n^{p_n}$. We can use the lexicographic ordering on the monomials, and because $t = (q-1)^k$ for some k (t divides T), a block B_i correspond to all monomials $\mathbf{X^p}$, such that the first k coordinates of $\mathbf{p}$ represent the $(q-1)$-nary decomposition of the integer i (over k q-bits).

In the case of $q = 4$ that we will use in the next section, for $n = 4$ and $t = 9$, the $3^4 = 81$ monomials are split into 9 blocks $B_0, \ldots, B_8$ of size 9, and a monomial $\mathbf{X^p}$ lies in B_6 in and only if $\mathbf{p}$ is of the form $(2, 0, \star, \star)$ with $\star \in \{0, 1, 2\}$, where $[2\|0]$ is the ternary decomposition of the integer 6.

Now, consider $i \in \{0, \ldots, (q-1)^k - 1\}$, and let $\mathbf{X^p}$ be a monomial appearing in $\mathbf{e}$ with a nonzero coefficient. In particular, the k entries of $\mathbf{p}$ can be parsed as a $q-1$-ary decomposition of i, which we denote by $[i]_{q-1}$. It is clear that $\mathbf{X^p}$ is of the form $\mathbf{X^{p_0+p_1}}$ where $\mathbf{p_0}$ (resp. $\mathbf{p_1}$) identifies one of the t nonzero monomials in $\mathbf{e_0}$ (resp. $\mathbf{e_1}$), and the sum is taken modulo $q-1$ component-wise. In particular, there are at most t^2 such monomials, and for each nonzero monomial in $\mathbf{e_1}$ contributing to $\mathbf{X^p}$ of $\mathbf{e_0}$, the first k entries $[i_0]_{q-1}$, there corresponds at most one nonzero monomial in $\mathbf{e}_1$ contributing to $\mathbf{X^p}$, namely $\mathbf{X^{p-p_0}}$.[4] In other words, the monomial $\mathbf{X^p}$ can be produced by *at most* t possible pairs of monomials $(\mathbf{X^{p_0}}, \mathbf{X^{p_1}})$, whose first k entries are $([i_0]_{q-1}, [i]_{q-1} - [i_0]_{q-1})$, with i_0 ranging over $\{0, \ldots, t-1\}$.

Example. Let $n = 3$ and $t = 3$. Set $\mathbf{e}_0 \coloneqq X_3^2 + X_1X_2X_3 + X_1^2$ (which corresponds to positions $(0,0,2), (1,1,1),$ and $(2,0,0)$) and $\mathbf{e}_1 \coloneqq 1 + X_1 + X_1^2$ (which corresponds to positions $(0,0,0), (1,0,0),$ and $(2,0,0)$). Then,

$$\mathbf{e}_0 \cdot \mathbf{e}_1 = \underbrace{(1 + X_3^2 + X_2X_3)}_{\in B_0} + \underbrace{(X_1 + X_1X_3^2 + X_1X_2X_3)}_{\in B_1} + \underbrace{(X_1^2 + X_1^2X_3^2 + X_1^2X_2X_3)}_{\in B_2}.$$

Now, in order to obtain an additive sharing of the vector $\mathbf{e}$, one can directly perform the DPF protocol on the $\mathbf{e^{(i)}}$ directly, each of length n/t, shaving a factor t to obtain seeds of size $O(t \cdot (\log(n)\lambda + \log q))$, and computation costs dominated by tn group operations and evaluations to a PRG. We call this noise distribution a *regular noise distribution* as it forces the non-zero coordinates of $\mathbf{e}$ to be distributed with a pattern, and call r- QA-SD or more generally r- $\mathcal{R}$-SD the associated regular Quasi-Abelian Syndrome Decoding assumption, or the regular Syndrome Decoding over a Ring.

Obviously, such a distribution has to be handled with caution. In [BCGI+20b], the authors provide proof that the Ring Syndrome Decoding assumption was secure against a large class of attacks in the regular case. The proof of the resistance of QA-SD against linear tests is in fact true for a very large group of distribution among which the regular distribution. Additionally, we conduct a thorough cryptanalysis of the QA-SD assumption in Chapter 7, where the regularity is of course taken into account.

4. Note that the corresponding monomial $\mathbf{X^{p_1}}$ might not appear in $\mathbf{e}_1$.

5.6 FOLEAGE: A Full Framework

In this section we give an analysis of a full PCG for OLEs construction, FOLEAGE, following [BBCC+24]. This construction beats the state of the art in producing OLEs. We will give a high-level description of the full framework, and succinctly precise the different techniques used. For more details, we refer to the full article, available on eprint [BBCC+24].

5.6.1 High-level Description of FOLEAGE

As we previously stated, the PCG produced via the use of a Group Algebra falls short of constructing numerous OLEs over $\mathbb{F}_2$, the unique missing stone of the framework. On the other hand, all the $q \geqslant 3$ are suitable via this method. The motivation for this work was the following result: we show that using this programmable PCG for generating OLEs over $\mathbb{F}_4$, we can construct multiplication triples over $\mathbb{F}_2$. The cost of this transformation is only a single bit of communication per triple and per party in the preprocessing phase.

In order to obtain the best possible scheme from top to bottom, we optimized different parts of the constructions tailored for $q = 4$. First, we constructed an FSS scheme designed for $q = 4$, by improving the seed distribution algorithm, by using a ternary DPF. Second, we took advantage of having reduced q to use the standard early termination technique that was already pointed out by [BGI16]. Finally we used the FFT algorithm sketched in Section 5.5.2 and discussed different implementations, and described a simple way to execute all the c^2 evaluations in parallel.

Additionally, an involved cryptanalysis of the QA-SD assumption was done, and this is the object of Chapter 6. In short, we have shown new ways to perform the cryptanalysis of the QA-SD assumption that were overlooked in [BBCC+24], and proved that our initial set of parameters achieved only 118 bits of security (for a target security of 128 bits).

5.6.2 Using OLE over $\mathbb{F}_4$

What follows is the technique that allows to go from multiplication triples over $\mathbb{F}_4$ to multiplication triples over $\mathbb{F}_2$. Remember that players can easily obtain multiplication triples over $\mathbb{F}_4$ using their OLEs over $\mathbb{F}_4$ given by the PCG (see Lemma 2.5.3). Once converted into $\mathbb{F}_2$-triples, these triples can be used within standard GMW protocol in the standard way described in Section 2.5.4.1.

Let $(\llbracket a \rrbracket^4, \llbracket b \rrbracket^4, \llbracket ab \rrbracket^4)$ be a Beaver triple over $\mathbb{F}_4$. Writing $x = x(0) + \theta \cdot x(1)$ for any $x \in \mathbb{F}_4$, with θ a root of the polynomial $X^2 + X + 1$ (hence $\theta^2 = \theta + 1$), we have

$$\begin{aligned} a \cdot b =& a(0)b(0) + a(1)b(1) + \theta \cdot (a(0)b(1) + a(1)b(0) + a(1)b(1)) \\ \implies & (ab)(0) = a(0)b(0) + a(1)b(1). \end{aligned}$$

The parties can reconstruct $b(1)$ at the cost of sending a single bit of communication per party from their share $\llbracket b \rrbracket^4 = \llbracket b(0) \rrbracket^2 + \theta \cdot \llbracket b(1) \rrbracket^2$. With the knowledge of $b(1)$ the parties can locally compute shares of $a(0)b(0)$ as follows:

$$\llbracket a(0)b(0) \rrbracket^2 = \llbracket ab \rrbracket^4(0) + b(1) \cdot \llbracket a \rrbracket^4(1).$$

We let all parties set their multiplication triple over $\mathbb{F}_2$ to be $(\llbracket a(0) \rrbracket^2, \llbracket b(0) \rrbracket^2, \llbracket ab \rrbracket^4(0) + b(1) \cdot \llbracket a \rrbracket^4(1))$. Note that the only communication consists of reconstructing $b(1)$. As neither $a(0), b(0)$ is related to $b(1)$, the reconstruction does not harm security.

Here we obtain an enhanced protocol in the preprocessing model: the parties run the PCG to be given the OLE over $\mathbb{F}_4$, locally obtain their multiplication triples over $\mathbb{F}_4$, and then broadcast one bit

per gate. They then obtain multiplication triples over $\mathbb{F}_2$, and can finish by using the standard GMW protocol (see Section 2.5.4).

5.6.3 Ternary DPFs for Enhanced Seed Distributions

An important improvement concerns the seed distribution algorithm, and more exactly the DPF algorithm (see Definition 2.5.9). The DPF seed distribution algorithm is constructed based on the Doerner-shelat protocol [Ds17], which requires the parties to hold binary shares of the non-zero positions in the noise. The idea behind this optimization is to get this first step for free. Remark that over $\mathbb{F}_4[\mathbb{G}] = \mathbb{F}_4[X_1, \ldots, X_n]/(X_1^3 - 1, \ldots, X_n^3 - 1)$ a monomial can be written as $\mathbf{X}^{\mathbf{k}} := \prod_{i=1}^{n} X_i^{k_i}$, where $\mathbf{k} = (k_1, \ldots, k_n) \in (\mathbb{Z}_3)^n$. In other words, we can express the position of a coefficient $c_{\mathbf{k}}$ as $\mathbf{k}$. Now, observe that given two monomials $m_0 = \mathbf{X}^{\mathbf{k_0}}, m_1 = \mathbf{X}^{\mathbf{k_1}}$, it appears immediately that $m_0 \cdot m_1 = \mathbf{X}^{\mathbf{k_0}+\mathbf{k_1} \mod 3}$. This remark is not without significance, as it means that the parties holding e_0, e_1 respectively have already a sharing of the non-zero positions in the noise. There is still a problem though, as a careful reader would have noticed that the sharing of the position we obtain is over $\mathbb{Z}_3$, whereas Doerner-shelat requests it in its binary form. In order to take advantage of this trick, we developed a generalized version of Doerner-shelat using shares over $\mathbb{F}_3$ of the non-zero positions as a basis.

We give a high-level idea of this new construction of Doerner-shelat, more detail can be found in [BBCC+24, Section 5]. Doerner-shelat algorithm classically uses a PRG $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$, constructing a binary-tree. Because our path is given modulo 3, what we want is to generalize this notion with a *ternary tree*. Given a node, its three children are this time computed using a length-tripling PRG $G : \{0,1\}^\lambda \to \{0,1\}^{3\lambda}$ on the node value. This requires adapting the standard DPF construction (see [BGI15; BGI16]), and to do again the proof of security.

With this new construction of Doerner-shelat, the path to a leaf is given exactly by the leaf position written as an $\mathbb{Z}3$-vector. In essence, the new construction requires 3 corrections words instead of 1 in the binary case, as well as two 1-out-of-3 OTs instead of one 1-out-of-2 OT, for each level of the tree. This entails small drawbacks. The correction words increase the seed size by a factor of 1.5, and the change of oblivious transfers costs also a little in communication and computations ($2(ct)^2 \log_3(D/t)$ 1-out-of-3 OTs of 3λ-bit strings instead of $2(ct)^2 \log_2(D/t)$ 1-out-of-2 OTs of 2λ-bit strings). On the other hand, the optimization completely removes of the additional step which consist giving the parties a share of the nonzero positions, saving a cost of $2(ct)^2 \cdot \log(D/t)$ oblivious transfers in $\log(D/t)$ rounds (i.e., half of the total number of rounds and OTs). Because the number of PRG evaluations one performs when evaluating a ternary tree is smaller than when evaluating a binary tree, expanding the PCG seeds also becomes 20% faster. In addition, because the tree is ternary, it is also shorter, and therefore the number of rounds in the protocol in this modified Doerner-shelat protocol is reduced (from $\log_2(D/t)$ to $\log_3(D/t)$).

5.6.4 Early Termination

Another optimization concerns the evaluation by the players of their SPFSS shares. Indeed, note that the construction of the DPF via the GGM tree (see Section 2.5.3.5) produces leaves of the size of the security parameter $\lambda = 128$ bits (security parameter of the PRG used for the GGM tree). In [BGI16] the authors point out that in the case of small output group, *early termination* can be used. This is exactly the use case we are in, as we want each single leaf to encode a value in $\mathbb{F}_4$. Currently, the size of the leaf is therefore too big compared to the size of the element we want to represent. If we use the standard protocol without optimizations, we would be using the 128 bits of the leaf to represent just one element of $\mathbb{F}_4$ spoiling at the same time $128 - 2 = 126$ bits. *Early termination*

simply entails truncating the tree 6 levels earlier and setting the value of the new 128-bit leaf on the special path to encode a unit vector consisting of zeroes except on the exact 2 bits where it equals to the correct value of $\mathbb{F}_4$ element. This essentially boils down to "hard-coding" the end of the path directly into the leaf, as illustrated in Figure 5.5. This means that the computational costs of the SPFSS evaluations is divided by $64 = 2^6$, since we stop 6 levels earlier. In addition, we also reduce the communication costs.

This simple trick was known for some time and already used in Private Information Retrieval (PIR), for example in [BGI16]. Nevertheless, it is the first time that one can apply it to PCG: prior PCG constructions were always built over large fields, and therefore this optimization was not possible. It is thanks to our first work ([BCCD23]) that the trick reappears in this context. The same limitations arise in the silent OT extension protocols [BCGI+19b; BCGI+19a; CRR21; BCGI+22; RRT23], which are also limited by the DPF evaluations. Indeed, this optimization could not be applied, because the DPF is used to output "authenticated" shares of a (potentially small) field element with a (large) MAC, which requires the leaves to encode 128 bits output values.

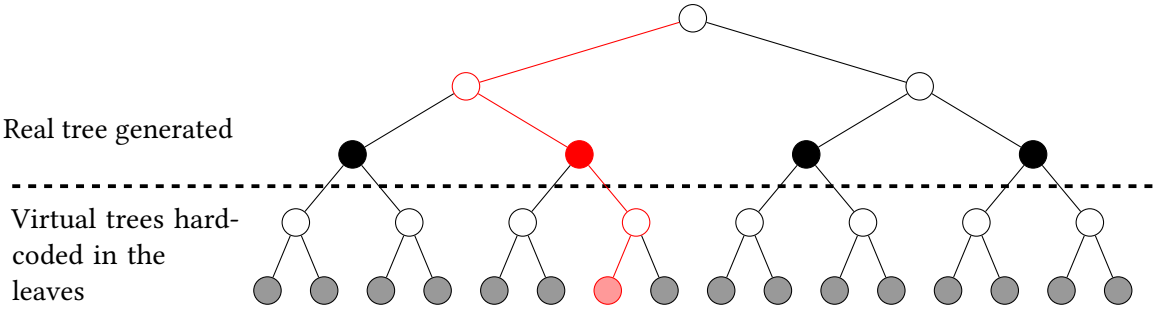

Figure 5.5 – Early termination example in the case we truncate only two steps earlier. Solid black nodes represent "zero" leaves, whereas solid red leaves can take on any value.

5.6.5 Concrete Evaluation of the Polynomials with FFT

In this section, we show how to implement concretely the FFT in our case $q = 4$. Once again, we represent an element of $\mathbb{F}_4$ by its natural canonical representation using 2 bits: given $x \in \mathbb{F}_4$, we write $x(0)$ and $x(1)$ to denote the two $\mathbb{F}_2$ coefficients of x, with $x = x(0) + \theta \cdot x(1)$. Given a machine word of 64 bits, we represent a vector of size 32 over $\mathbb{F}_4$, such that the even indexed bits are high order and the odd indexed bits are low order. This is illustrated in Figure 5.6.

As shown in Section 5.5.2, we can compute the evaluations via a simple recursive algorithm. It is displayed in Figure 5.7. We have also considered a non-recursive approach, but it does not yield substantial gain and therefore the main construction presented in FOLEAGE uses the recursive algorithm, which has the advantage of keeping everything simple.

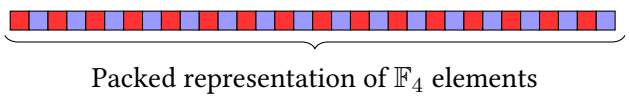

Figure 5.6 – Representation of a vector of $\mathbb{F}_4$ elements. Red blocks represent the high-order bits while the blue blocks represent low-order bits.

Fast-Evaluation algorithm

PARAMETERS: $n > 0$ an integer, $P \in \mathbb{F}_3[X_1, \ldots, X_n]/(X_1^3 - 1, \cdots, X_n^3 - 1)$, a polynomial with n variables.

FastEval(n, P):

1: **if** $n = 1$ **then**
  1.1: **return** $\{P(1), P(\theta), P(\theta + 1)\}$
2: **else**
  2.1: Write $P(X_1, \ldots, X_n) = P_0(X_1, \ldots, X_{n-1}) + X_n P_1(X_1, \ldots, X_{n-1}) + X_n^2 P_2(X_1, \ldots, X_{n-1})$.
  2.2: $S \leftarrow \varnothing$
  2.3: $\forall i \in \{0, 1, 2\}, S_i \leftarrow$ FastEval$(n - 1, P_i)$
  2.4: **foreach** $i \in [|S_0|]$:
    2.4.1: $f_j(X) \coloneqq S_0[j] + S_1[j]X + S_2[j]X^2$.
    2.4.2: $S \leftarrow S \cup \{f_j(1), f_j(\theta), f_j(\theta + 1)\}$
  2.5: **return** S

Figure 5.7 – Fast evaluation of a polynomial in n variables.

5.6.5.1 Concrete Cost of the Computation

At each step in the algorithm of Figure 5.7, we evaluate a polynomial of degree 2, with coefficients in $\mathbb{F}_4$, over the values $\{1, \theta, \theta + 1\}$. For a polynomial $f(X) = a + bX_i + cX_i^2$,

- in the case $X_i = 1$, then the evaluation of the polynomial becomes $a + b + c$.
- in the case $X_i = \theta$, the evaluation becomes $(a + c) + \theta \cdot (b + c)$.
- in the case $X_i = \theta + 1$, the evaluation becomes $(a + b) + \theta \cdot (b + c)$.

Because we want anyway to compute all the evaluations, it is natural to reuse the different common intermediate calculations. The three different evaluations can therefore be obtained via the following steps:

1. compute $a + b, a + c, b + c$.
2. compute $\theta \cdot (b + c)$.
3. compute $a + b + c$.
4. compute $(a + c) + \theta \cdot (b + c))$, and $(a + b) + \theta \cdot (b + c))$.

for a total of 12 classical bit-by-bit XOR over $\mathbb{F}_2$, and a multiplication by θ (which is nothing more than the shift on the right).

5.6.5.2 Taking Advantage of Vectorization

Today's processors offer XOR operations for machine words of size 64 bits. Remember that, in our representation, we pack 32 elements of $\mathbb{F}_4$ in a single machine word. Therefore, this general XOR operation offers us high parallelization power. A way to use it is to perform multiple FFTs

in parallel. Concretely, we can therefore perform up to 32 FFTs in parallel, as each machine word contains a single coefficient of the same monomial for each of the c^2 polynomials we want to evaluate. Assuming that c^2 is smaller than 32, results in a reduction of computational complexity by a factor of c^2, without any drawbacks. [5]

Therefore, the cost of the evaluation of a single polynomial being of $16n \cdot 3^{n-1}$ XOR operations, this optimization reduces the overall cost of obtaining the full evaluation of the c^2 polynomials to be $16 \left\lceil c^2/64 \right\rceil n \cdot 3^{n-1}$.

5.6.6 Performances

We then present the performances of FOLEAGE, as presented in [BBCC+24]. The choice of parameters comes from an involved cryptanalysis that can be found in Chapter 6, and more especially the script discussed in Section 6.3. We also provide a script giving the parameters we should use to guarantee security [6]. We also implemented an open-source prototype of the PCG construction in C, the open-source code of the benchmarks is available online [7]. We display thereafter a table comparing FOLEAGE to the state of the art, taken directly from [BBCC+24, Section 1]. Refer to [BBCC+24, Section 7] for more information on the implementation.

	Communication	`localhost`	**LAN**	**WAN**
Multi-party setting ($N = 10$)				
SoftSpoken ($k = 2$)	134 GB	342s	1192s	12207s
SoftSpoken ($k = 4$)	67 GB	405s	596s	6104s
SoftSpoken ($k = 8$)	34 GB	1900s	**1900s**	3052s
			*298s	
RRT	6.3 GB	2619s	**2619s**	**2619s**
			*50.3s	*515s
FOLEAGE	0.7 GB	1463s	**1463s**	**1463s**
			*5.6s	*57.9s
Two-party setting ($N = 2$)				
SoftSpoken ($k = 2$)	15 GB	38s	119s	1221s
SoftSpoken ($k = 4$)	7.5 GB	45s	60s	610s
SoftSpoken ($k = 8$)	3.7 GB	211s	**211s**	**211s**
RRT	258 KB	292s	**292s**	**292s**
FOLEAGE	33.5 MB	81s	**81s**	**81s**

Table 5.1 – Comparison of state-of-the-art protocols to generate N-party Beaver triples over $\mathbb{F}_2$ for $N = 10$ and $N = 2$ parties.

The `localhost` column reports the runtimes (ignoring communication) for generating 10^9 triples. All protocols run on one core of AWS `c5.metal` (3.4GHz CPU); all runtimes averaged across ten trials. "Communication" denotes the number of bits communicated per party for 10^9

5. In practice, using larger machine words has an impact by increasing stack usage, but this is only observed when performing an FFT over very large polynomials.

6. https://github.com/mbombar/estimator_folding

7. https://github.com/sachaservan/FOLEAGE-PCG

triples. LAN and WAN refer to the theoretical time required to generate 10^9 triples over a 1 Gbps and 100 Mbps network respectively, with respective delays 1ms and 40ms. Numbers in **bold red** indicate that the bottleneck cost is the local computation. The star * indicates the maximum theoretical throughput with more computational power (e.g., using multiple cores). Since each party computes $2 \cdot (N-1)$ expansions for the PCG in parallel for an N-party Beaver triple, the running time is divided by C when using C cores whenever $C \leqslant 2 \cdot (N-1)$.

Chapter 6

Cryptanalysis of the QA-SD assumption

Conjecture 5.4.1 attempted to establish the security of the QA-SD assumption against linear tests, which encompass the majority of effective attacks against SD-like assumptions. Further cryptanalysis is needed, first because the conjecture is not fully proved, second because the security parameters given by the conjecture would not be optimal. To achieve an efficient construction, it is a standard practice to select security parameters so that the best known attacks fail. In this chapter, we describe all the different tools that an adversary can use to attack the QA-SD constructions. We describe a strategy that attempts to exploit all structures present in the QA-SD construction and identify the parameters that offer security in the special case $q = 4$. This is based on the following work [BBCC+24, Section 6], co-authored with Maxime Bombar, Dung Bui, Geoffroy Couteau, Alain Couvreur, and Sacha Servan-Schreiber.

Outline of the current chapter

For a better understanding of the content of this chapter, we invite the reader to have a look at Section 2.4.3, where the QA-SD assumption is described, in addition to the necessary knowledge on group algebra. Refer to Section 5.4.2 for more information on the context in which we use this assumption. Additionally, please refer to Appendix C for attacks on generic syndrome decoding.

6.1 Generalities

We recall the problem at hand. Let us consider the QA-SD$(c, t, \mathbb{G})$ assumption, which asks to distinguish $((a_1, \ldots, a_{c-1}), s \coloneqq e_0 + a_1 e_1 + \cdots + a_c e_c)$ from the uniform distribution over $\mathbb{F}_q[\mathbb{G}]^c \times \mathbb{F}_q[\mathbb{G}]$, where $a_i \xleftarrow{\$} \mathbb{F}_q[\mathbb{G}]$, and $e_i \xleftarrow{\$} \mathcal{S}(\mathbb{F}_q[\mathbb{G}], t)$ (t-sparse elements of $\mathbb{F}_q[\mathbb{G}]$). As already mentioned in Conjecture 5.4.1, the attack version can be represented as finding the vector $\mathbf{e}$ in the following equation:

$$\begin{pmatrix} \mathbf{I} & \mathbf{M}_{a_1} & \ldots & \mathbf{M}_{a_{c-1}} \end{pmatrix} \begin{pmatrix} \mathbf{e}_0 \\ \vdots \\ \mathbf{e}_{c-1} \end{pmatrix} = \mathbf{s}, \tag{6.1}$$

where $\mathbf{M}_{a_i}$ is the matrix representation of the multiplication by the element a_i with respect to the basis given by $\mathbb{G}$, for some arbitrary ordering of $\mathbb{G}$, and $\mathbf{e}_i$ are sparse vectors of Hamming weight t. In the following, we explore the best attacks to solve the search variant.

6.1.1 Generic Attacks on SD

First, note that we can apply the well-known attacks against SD, because an instance of QA-SD is just a specific case of SD. The best classical attacks against SD were already mentioned in Chapter 4: *Information Set Decoding Attacks* (ISD) and *Dual attacks* (or Statistical Attacks). Appendix B presents a concrete analysis of the ISD algorithms. The most recent ISD algorithms (starting from [MO15b]) are not considered: although they achieve good asymptotic results, they suffer from prohibitive constants and are impractical for our range of parameters.

Dual attacks are a new alternative to ISD, introduced in [Jab01; Ove06b]. Further optimizations have been explored [DT17b; CDMT22a; MT23; CDMT24]. Although ISD techniques have been considered the best for solving the decoding problem, in some specific regimes dual attacks achieve better results. However, these improvements apply only for a rate below $1/2$ [CDMT24, Figure 1]. As we are working with a high code rate of $1 - 1/c$ (remember that the parity-check matrix is a $|\mathbb{G}| \times c \cdot |\mathbb{G}|$ matrix), we decided to not take these attacks into account. For the same reason, we do not take into account generic attacks on LPN such as the ones of BKW [BKW03] or Arora-Ge[AG11].

6.1.2 Possible Leverages

Nevertheless, the QA-SD construction, especially in the use case we consider, is far from generic, and many optimizations can be applied. Here we present the different advantage an adversary can use.

6.1.2.1 A (c, t)-blockwise regularity

First, note that in the description of Equation (6.1), the noise $\mathbf{e} \coloneqq (\mathbf{e}_0^\intercal, \ldots, \mathbf{e}_{c-1}^\intercal)^\intercal$ is (c, t)-blockwise-regular[1], with exactly t non-zero coordinates in each subvector $\mathbf{e}_j, 0 \leqslant j \leqslant c - 1$.

1. see Definition 2.4.8

This kind of regularity has been gaining increasing attention since its use in the NIST submission SDitH [AFGG+23] (in the case of unstructured codes). We use the following result by [FJR22].

Theorem 6.1.1 ([FJR22, Theorem 1]). *Let n, k, t be positive integers such that $n > k$, $n > ct$, and c divides n. If there is an algorithm solving a random instance of the (c,t)-blockwise regular decoding problem with code length n, dimension k and ct errors, in time T and with probability ε_c, then there exists an algorithm which solves a random instance of the standard decoding problem with the same parameters, in time T, and with probability ε_1 with*

$$\varepsilon_1 \geqslant \alpha \cdot \varepsilon_c \qquad \alpha \coloneqq \frac{\binom{n/c}{t}^c}{\binom{n}{ct}} < 1$$

To fully understand how we can use the theorem, let $T_1 = T/\varepsilon_1$ and $T_c = T/\varepsilon_c$. T_1 and T_c are respectively the attack expected time in the standard case and in the (c,t)-blockwise regular case. The theorem states that $T_1 \times \alpha \leqslant T_c$. We can see this inequality as a lower bound on the time it takes to solve the (c,t)-blockwise regular variant. Even if the bound seems really conservative, the c-regular variant is a new assumption that has not received deep and thoughtful analysis from the community. Therefore, we will be cautious when choosing our parameters and assume every time that $T_c = T_1\alpha$. This allows us to estimate the cost by using a generic ISD technique and then applying the (c,t)-regular penalty by multiplying by α.

6.1.2.2 Exploiting the Algebraic Structure: The DOOM Strategy and Folding Attacks

The adversary can also exploit the quasi-abelian structure of the code via the Decoding One out Of Many (DOOM) strategy. This strategy, from [Sen11], explains how an adversary can speed up the attack computational time by $\sqrt{N}$ given N distinct instances of the decoding problem. Thanks to the structure our parity-check matrix $\mathbf{H}$, we can obtain a maximum of $|\mathbb{G}|$ instances: it suffices to consider all the mappings $x \in \mathbb{F}_q[\mathbb{G}] \to g \cdot x$ for $g \in \mathbb{G}$. A multiplication by an element g induces a permutation of the basis, which does not change the weight of the error. Therefore from a single instance we can deduce $|\mathbb{G}|$ distinct instances. Using these $|\mathbb{G}|$ instances with the DOOM attack results in a gain of a factor $\sqrt{|\mathbb{G}|}$.

Another approach to take advantage of the quasi-abelian structure is to consider folding attacks. Folding attacks involve reducing the sample modulo some ideal of $\mathbb{F}_q[\mathbb{G}]$ to obtain a smaller instance of the decoding problem. When doing this, the instance size is reduced, but the noise level does not decrease significantly (it depends on the number of possible collisions). If the folded noise rate (number of errors in the folded error vector divided by the size of the folded code) does not increase too much, then the attack can offer significant advantages. Folding attacks will be precisely analyzed in Section 6.2.

6.2 Concrete analysis of the attacks

6.2.1 What is a Folding?

Let $\mathbb{H}$ be a subgroup of $\mathbb{G}$. An element of $\mathbb{G}$ can be written as $\bar{g} + h$ for $\bar{g} \in \mathbb{G}/\mathbb{H}$ and $h \in \mathbb{H}$. We can induce a natural projection from $\mathbb{G} \to \mathbb{G}/\mathbb{H}$:

$$\pi_{\mathbb{H}}: \begin{cases} \mathbb{F}_q[\mathbb{G}] & \longrightarrow & \mathbb{F}_q[\mathbb{G}/\mathbb{H}] \\ \sum_{g\in\mathbb{G}} a_g g & \longmapsto & \sum_{\bar{g}\in\mathbb{G}/\mathbb{H}} \left(\sum_{h\in\mathbb{H}} a_{g+h}\right) \bar{g}. \end{cases}$$

$\pi_{\mathbb{H}}$ is a morphism of algebras. Let us write an element of $\mathbb{F}_q[\mathbb{G}]$ as a vector of length $|\mathbb{G}|$. It is composed of $|\mathbb{G}/\mathbb{H}|$ sub-vectors of size $|\mathbb{H}|$ (corresponding to the coefficients of the coset of $\mathbb{H}$). The projection compresses these vectors by summing up their entries, and as a consequence reduces the size of the original vector by $|\mathbb{H}|$. The projection is illustrated in Figure 6.1.

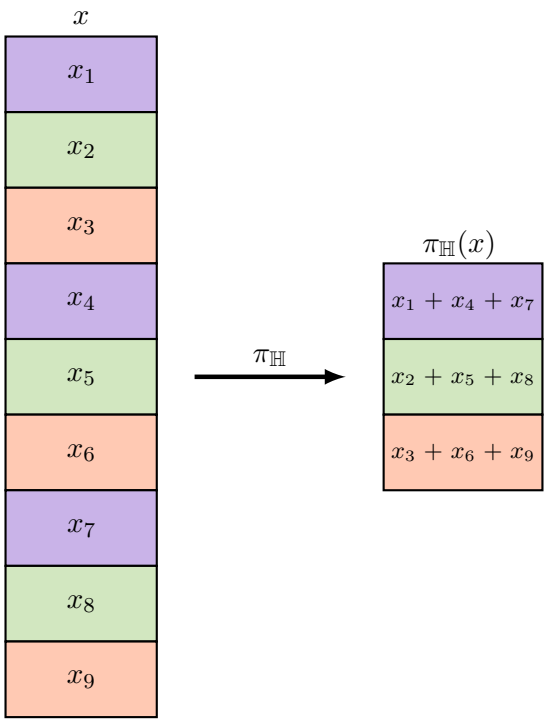

Figure 6.1 – Example representation of $\pi_{\mathbb{H}}$ for a group $\mathbb{G}$ of size 9 and a subgroup $\mathbb{H}$ of size 3. Each coset of $\mathbb{H}$ is represented by a different color.

Let $\mathbf{y} \coloneqq (y_1, \ldots, y_c) \in \mathbb{F}_q[\mathbb{G}]^c$. The folding of the vector $\mathbf{y}$ with respect to $\mathbb{H}$ is defined as follows: $\mathrm{Fold}_{\mathbb{H}}(\mathbf{y}) \coloneqq (\pi_{\mathbb{H}}(y_1), \ldots, \pi_{\mathbb{H}}(y_c)) \in \mathbb{F}_q[\mathbb{G}/\mathbb{H}]^c$. Let $(\mathbf{a}, \langle \mathbf{a}, \mathbf{e}\rangle + e_0)$ be a QA-SD$(c, t, \mathbb{G})$ instance. Because $\pi_{\mathbb{H}}$ is morphism of algebras, it follows that

$$\pi_{\mathbb{H}}\left(e_0 + \sum_{i=1}^{c-1} a_i e_i\right) = \pi_{\mathbb{H}}(e_0) + \sum_{i=1}^{c-1} \pi_{\mathbb{H}}(a_i)\pi_{\mathbb{H}}(e_i) \in \mathbb{F}_q[\mathbb{G}/\mathbb{H}].$$

This defines exactly another syndrome, with the error vector being $\mathrm{Fold}_{\mathbb{H}}(\mathbf{e})$ and corresponding parity check matrix

$$\begin{pmatrix} \pi_{\mathbb{H}}(\mathbf{I}) & \mathbf{M}_{\pi_{\mathbb{H}}(a_1)} & \ldots & \mathbf{M}_{\pi_{\mathbb{H}}(a_{c-1})} \end{pmatrix} \begin{pmatrix} \pi_{\mathbb{H}}(e_0) \\ \vdots \\ \pi_{\mathbb{H}}(e_{c-1}) \end{pmatrix}. \tag{6.2}$$

Therefore, by applying the projection morphism on the syndrome, we obtained a new syndrome of a *folded instance*. Let's analyze the underlying code of the new instance. There are still c different blocks, and therefore the code is of size $c(|\mathbb{G}|/|\mathbb{H}|)$, while the dimension of the code is $(c-1)(|\mathbb{G}|/|\mathbb{H}|)$. As a result, the code rate remains equal to $1-1/c$. Thus, the structure is mostly the same but the size has been reduced by $|\mathbb{H}|$. As for the noise, things are a bit different. The noise retains its c-regular structure because the folding is applied to each bloc independently. The question that immediately arises is: what is the weight of the folded error vector? For a given subgroup $\mathbb{H}$, we are summing, for each $\bar{g} \in \mathbb{G}/\mathbb{H}$ the coordinates of the elements $\bar{g}+h$ for each element $h \in \mathbb{H}$. Therefore, collisions are possible, but very rare if the number of errors in the original vector $\mathbf{e}$ is small. The weight of the folded error vector is therefore similar to the original. Still, the overall size of this new QA-SD instance is smaller than the original one, so the noise rate increases. To obtain a proper bound, it is important to compute the probability distribution of the weight of the folded vector. This will be discussed in the Section 6.2.3.

6.2.2 High-Level Description of the Attack

In light of the information given above, we provide a first description of the attack. A natural attack is to (1) randomly select a subgroup $\mathbb{H}$ of $\mathbb{G}$, (2) consider the folded instance associated to $\mathbb{H}$, (3) use a generic algorithm against SD on the folded (and thus reduced) instance, (4) apply the penalty coming from the (c,t)-blockwise regularity and the $\sqrt{|\mathbb{G}|/|\mathbb{H}|}$ gain from the DOOM strategy. Note that we ensure the penalty is applied to the reduced instance.

The choice of the subgroup $\mathbb{H}$ will be discussed later. In one word, one must ensure the solution to the decoding problem remains unique, while also selecting the largest possible $\mathbb{H}$ to reduce the size of the folded instance.

To estimate the time a generic algorithm would take on the folded instance, one must analyze the folding function and, more precisely, the distribution of the weight of the folded error vector: the smaller the number of errors, the lower the cost of finding a solution. This analysis also led to another, more effective attack: the adversary can "bet" that for a certain subgroup $\mathbb{H}$ the folded vector has a very small weight, enabling a very quick resolution via generic decoding. Performing the decoding algorithm with small noise can be so efficient that it compensates for the loss incurred from trying many subgroups. Next, we formally introduce the folding function and the distribution of the weight of a folded error.

6.2.3 Distribution of the Weight of a Folded Vector

We want to compute $\Pr_{e \xleftarrow{\$} \mathcal{S}(\mathbb{F}_q[\mathbb{G}],t)}[\, w_H(\pi_{\mathbb{H}}(e)) = u \,]$. We express the probability by counting the vectors that satisfy the condition:

$$\Pr_{e \xleftarrow{\$} \mathcal{S}(\mathbb{F}_q[\mathbb{G}],t)}[\, w_H(\pi_{\mathbb{H}}(e)) = u \,] = \frac{A_{\mathbb{H}}(t,u)}{\binom{|\mathbb{G}|}{t}(q-1)^t},$$

where

$$A_{\mathbb{H}}(t,u) \coloneqq \left| \left\{ e \in \mathbb{F}_q[\mathbb{G}] \mid w_H(e) = t \text{ and } w_H(\pi_{\mathbb{H}}(x)) = u \right\} \right|.$$

Let us analyze $A_{\mathbb{H}}(t,u)$. Let $e \in \mathbb{F}_q[\mathbb{G}]$ of Hamming weight t and let $\ell \coloneqq |\mathbb{H}|$. A coordinate of $\pi_{\mathbb{H}}(e)$ indexed by a coset $\bar{g} = g_0 + \mathbb{H} \in \mathbb{G}/\mathbb{H}$ is zero if and only if the sum of the corresponding entries in e is zero, meaning the subvector $\tilde{x}$ induced by the g_0+h positions belongs to the $[\ell, \ell-1]_q$-parity

code denoted by $\mathcal{C}_\ell$. The weights of the subvectors define a partition of t into parts of size at most ℓ: we call this partition the *signature* of e, and we denote it $\sigma = (\sigma_1, \ldots, \sigma_{|\mathbb{G}|/|\mathbb{H}|})$, $\sum_i \sigma_i = t$. Our strategy is to count the number of $e \in \mathbb{F}_q[\mathbb{G}]$ with $wt\,(e) = t, wt\,(\mathrm{Fold}_{\mathbb{H}}(e)) = u$ of signature σ, and summing for all possible signature. Given a signature $\sigma = (\sigma_1, \ldots, \sigma_{|\mathbb{G}|/|\mathbb{H}|})$, and assuming without loss of generality that only the first u parts of the partition contribute to the weight of the folded error vector, the total number of such e that satisfy the above condition is given by:

$$\prod_{j=1}^{u} \theta(\sigma_j, \ell) \prod_{j=u+1}^{|\mathbb{G}/\mathbb{H}|} \nu(\sigma_j, \ell),$$

where $\nu(\omega, \ell)$ is the number of codewords of $\mathcal{C}_\ell$ of Hamming weight ω and $\theta(\omega, \ell)$ is the number of vectors in $\mathbb{F}_q^\ell \setminus \mathcal{C}_\ell$ of Hamming weight ω. Therefore, we can now write:

$$A_{\mathbb{H}}(t, u) = \binom{|\mathbb{G}/\mathbb{H}|}{u} \sum_{\substack{\sigma_1 + \cdots + \sigma_{|G/\mathbb{H}|} = t \\ 0 \leqslant i_j \leqslant \ell}} \left(\prod_{j=1}^{u} \theta(\sigma_j, \ell) \prod_{j=u+1}^{|\mathbb{G}/\mathbb{H}|} \nu(\sigma_j, \ell) \right).$$

Define $P_{\nu,\ell} \coloneqq \sum_{\omega=0}^{\ell} \nu(\omega, \ell) X^\omega$ and $P_{\theta,\ell} \coloneqq \sum_{\omega=0}^{\ell} \theta(\omega, \ell) X^\omega$. Then the above expression is the coefficient of X^t in the polynomial $\binom{|\mathbb{G}/\mathbb{H}|}{u} P_{\theta,\ell}^u(X) P_{\nu,\ell}^{|\mathbb{G}/\mathbb{H}|-u}(X)$, which we denote by

$$[X^t] \left(\binom{|\mathbb{G}/\mathbb{H}|}{u} P_{\theta,\ell}^u(X) P_{\nu,\ell}^{|\mathbb{G}/\mathbb{H}|-u}(X) \right).$$

This is convenient because closed formulas of $P_{\nu,\ell}$ and $P_{\theta,\ell}$ exist (from a corollary of McWilliams' identity, [MS86]).

Lemma 6.2.1 ([CT19, Lemma 1])**.** *We have*

$$P_{\nu,\ell}(X) = \frac{1}{q}\Big((1 + (q-1)X)^\ell + (q-1)(1-X)^\ell\Big),$$

and

$$P_{\theta,\ell}(X) = \frac{q-1}{q}\Big((1 + (q-1)X)^\ell - (1-X)^\ell\Big).$$

Therefore we can conclude:

Proposition 6.2.1. *Let $0 \leqslant t \leqslant |\mathbb{G}|$ and $0 \leqslant u \leqslant \min(t, |\mathbb{G}/\mathbb{H}|)$. When e is uniformly distributed over the elements of $\mathbb{F}_q[\mathbb{G}]$ of Hamming weight t, then*

$$\Pr_{e \xleftarrow{\$} \mathcal{S}(\mathbb{F}_q[\mathbb{G}], t)} [\, w_H(\pi_{\mathbb{H}}(e)) = u \,] = \frac{\binom{|\mathbb{G}/\mathbb{H}|}{u} [X^t] \left(P_{\theta,\ell}^u(X) P_{\nu,\ell}^{|\mathbb{G}/\mathbb{H}|-u}(X) \right)}{\binom{|\mathbb{G}|}{t} (q-1)^t}.$$

6.2.4 Formal Analysis of the Attack

With knowledge of the weight distribution of the folded error, we can estimate the cost of our first attack idea: taking a subgroup $\mathbb{H}$ and applying the best ISD algorithm cost on the resulting folded instance. This would cost an average of $\sum_{w=0}^{t} (\mathrm{Cost}_{\mathrm{Decoding}}(\omega) + \mathrm{Cost}_{\mathrm{Folding}}) \cdot \Pr[\, w_H(\mathrm{Fold}_{\mathbb{H}}(x)) = \omega \,]$, where $\mathrm{Cost}_{\mathrm{Decoding}}$ is the complexity of the best decoding algorithm for the target weight ω in the

folded code, and $\text{Cost}_{\text{Folding}}$ is the cost of computing the folding operation, which is actually linear in $c|\mathbb{G}|$.

Nevertheless, knowing this distribution motivates a more promising attack: start by guessing the noise weight of the vector, say w_0, then pick a random subgroup $\mathbb{H}$ and apply the best decoding algorithm associated with the folded vector. We bound the number of iterations of the decoding algorithm (since the guess could be wrong) to the number of iterations required to decode w_0 errors using generic algorithms. If no solution is found, it means that the assumption was wrong for the chosen subgroup $\mathbb{H}$, and one can try another $\mathbb{H}$. Using this attack, an adversary can decode in time $\frac{\text{Cost}_{\text{Decoding}}(\omega_0) + \text{Cost}_{\text{Folding}}}{\Pr[\, w_H(\text{Fold}_{\mathbb{H}}(x)) = \omega_0 \,]}$.

Given a particular syndrome x, the adversary would then choose the target weight ω_0 to minimize this ratio and achieve the best complexity.

How to choose $\mathbb{H}$? It is natural to pick $\mathbb{H}$ as large as possible, to reduce the size of the instance on which we perform the generic algorithm. However, we must consider that the noise rate increases when we fold, as stated above. If the noise rate is too high, the solution to the decoding problem is not unique anymore. More concretely, recall the Gilbert-Varshamov (GV) bound, which, given a code of rate R, is defined as $\delta_{GV} = h_q^{-1}(1-R)$ where $h_q(x) \coloneqq -x \log_q\left(\frac{x}{q-1}\right) - (1-x)\log_q(1-x)$ for $x \in [0, 1-1/q]$, is the q-ary entropy function (see Proposition 2.4.2). A standard result states that if the noise rate is bellow δ_{GV} the uniqueness of the solution is assured with high probability; otherwise there will be exponentially many solutions (see [Deb23a, Chapter 2]). The problem with the latter case is that each potential solution to the folded instance must be tested to see if it also solves the original problem, which is inefficient compared to when the noise rate is below the GV bound. In the former scenario (noise rate bellow δ_{GV}), the single solution found translates directly to a solution of the original instance. Therefore, it is preferable to fold until the noise rate reaches the GV bound, but not further.

6.3 Concrete Parameters

The parameters are chosen to match the FOLEAGE construction. We take $\mathbb{G} = (\mathbb{Z}/3\mathbb{Z})^n$. The efficient attack presented above is much more efficient for small values of c, which is why we chose $c = 4$ in our setting. This allows us to maintain small parameters while achieving more than 128 bits of security (while still keeping some security margin). We also provide a script in SageMath [Ste+24] to compute a set of parameters for a given instance of QA-SD [2]. Concretely, given a size and noise level, the script uses the formulae proven in Section 6.2.3 to compute the weight distribution of the folded vector, determine the best target noise w_0, and then calculate the complexity of the attack. Results are displayed in Section 6.3. This cryptanalysis revealed that the original parameters of QA-SD, as presented in [BCCD23] were too optimistic, achieving only 118 bits of security compared to the intended 128. This script was used to derive the parameters presented in Table 5.1. Note that in [BCCD23], all the parameters were for $q = 3$. Here t is the number of errors per block, while in [BCCD23] it was the total number of errors. n_{fold} and k_{fold} are respectively the length and dimension of the folded code. N_{iter} is the number of different foldings necessary to run the attack, and ω_0 is the optimal target weight.

2. https://github.com/mbombar/estimator_folding.

n	c	t	$(n_{\text{fold}}, k_{\text{fold}}, \omega_0)$	$(\text{N}_{\text{iter}}, \text{Cost}_{\text{Decoding}})$	Number of subgroups	Actual security
25	4	16	$(2048, 1536, 54)$	$(2^{14}, 2^{89})$	2^{145}	118
30	4	16	$(2048, 1536, 54)$	$(2^{14}, 2^{89})$	2^{190}	118
35	4	16	$(2048, 1536, 54)$	$(2^{14}, 2^{89})$	2^{235}	118

Table 6.1 – Re-estimation of the security for the parameters given in [BCCD23].

Chapter 7

A WPRF for PCF constructions

The first PCF construction was proposed in [BCGI+20a]. They explained how to construct a PCF from a particular WPRF that can be efficiently shared between two parties; we say that the WPRF is FSS-friendly (see Section 3.2.1). They provided such an FSS-friendly WPRF using a new variant on Syndrome Decoding, called Variable Density Syndrome Decoding (VDSD). While being an important first step theoretically, their approach was not intended to be efficient, and their security parameters were huge ($\sim 10^7$). In this chapter, we explain in detail the construction of the VDSD assumption. We recap the first construction by [BCGI+20a], and point out some errors in their proof, which we address with corrections. Subsequently, we show how to make VDSD efficient by slightly changing the underlying assumption and providing a better analysis. This is based on an article published at PKC2023 [CD23], co-authored with Geoffroy Couteau.

Outline of the current chapter

7.1 State of the Art

A Pseudorandom Correlation Function (PCF) is a highly powerful yet challenging primitive to construct. It was formally introduced by [BCGI+20a]. A first proof of feasibility can be provided via heavy primitives using threshold fully-homomorphic encryption [DHRW16; BCGI+20a]. This type of construction is still limited for efficiency reasons.

To our knowledge, there are two main solutions to construct usable PCFs. The first one was introduced in [BCGI+20a]: a high-level idea was presented in Section 3.2. It is built using the combination of two primitives: (1) a Function Secret Sharing (FSS) scheme for functions in a particular class $\mathcal{C}$ (which enables splitting a function f into two additive shares $f = f_0 + f_1$, see Section 2.5.3.5), and (2) a weak pseudorandom function that can be constructed via functions in $\mathcal{C}$ (we then say it is a FSS-friendly WPRF). In practice, the WPRF is built using a variant of the Syndrome Decoding assumption due to its great compatibility with FSS. Indeed, we can build a weak pseudorandom function by interpreting the knowledge of $n - k$ different evaluation of the WPRF as a product $\mathbf{H} \cdot \mathbf{e}$, where $\mathbf{e}$ is very sparse and $\mathbf{H}$ is a special matrix. [BCGI+20a] first proposed this construction with a matrix composed of different blocks with decreasing density, making it the *Variable Density Syndrome Decoding* (VDSD) assumption. Their result was not meant to be efficient: the security parameter w was fixed to approximately 10^7, leading to impractical computational and communication costs. The construction was of more theoretical interest as it could be built in a complexity class as low as depth-2 AC$[\oplus]$ [1]. In [CD23], we demonstrated that by tweaking the assumption a bit (and thus losing the nice complexity class property), the VDSD assumption could, in fact, lead to close-to-practical PCF constructions ($w \approx 380$, resulting in approximately 500 OT/s with key size 3MB, or even 4000 OT/s for their aggressive parameters). The VDSD assumption will be the focus of this chapter. Still, the best PCF construction was proposed by [BCGI+22], in which the authors built over another Syndrome Decoding-like assumption, the *expand and accumulate* assumption. This assumption consists to define the parity-check matrix $\mathbf{H} = \mathbf{BA}$, where $\mathbf{B} \in \mathbb{F}_2^{n \times \mathsf{poly}(n)\cdot}$ is a matrix with sparse rows, and $\mathbf{A} \in \mathbb{F}_2^{\mathsf{poly}(n) \times \mathsf{poly}(n)}$ is the accumulator matrix, that is such that $\mathbf{xA} = (\mathbf{x}_1, \mathbf{x}_1 + \mathbf{x}_2, \ldots, \mathbf{x}_1 + \cdots + \mathbf{x}_n)$. Under the assumption that SD holds for this specific class of matrices, the authors proved that they can build a PCF that achieved state-of-the-art result of producing about 43k OT/s (3900 PRG evaluations per PCF outputs on a standard laptop) with a key size of approximately 650kB. Using more aggressive settings they reached $\approx 120k$OT/s for key size 370kB.

The second construction technique for PCF originated from decisional composite residuosity, and was put forward by [OSY21]. The scheme has the significant advantage of using the Paillier-based construction of homomorphic secret sharing and does not require any new, tailored-made assumptions but only the composite residuosity assumption. It also boasts the impressive property of having a *public-key setup*, which, when done, allows any party to start generating random correlations without any prior agreement. This is a property that the previous constructions did not possess. Still, it suffers from several downsides: (1) it is primarily limited to producing OTs, whereas the previous technique could be used for OLE and Beaver Triples, (2) the Decisional Composite Residuosity assumption is not post-quantum, whereas SD-like assumptions are supposed to be (even if this could be a bold assumption given that we are talking about non-standard types of SD), and finally (3) the efficiency considerations are not as favorable, lagging behind both VDSD and EASD based constructions by several orders of magnitude (producing around 1 OT/s). Another approach for public-key PCF for OT was recently advanced by [BCMP+24], based on the Naor-Reingold PRF. The

1. Functions that can be computed with circuit of depth 2, with AND and general XOR.

scheme achieves competitive results, even among non-public key PCF, producing about 15k-40k OT/s on a standard computer with keys of 30kB.

This chapter presents part of the work from [BCGI+20a] and [CD23]. First, Section 7.2 explains the different challenges that one can encounter when using SD-like assumptions as a way of creating the FSS-friendly WPRF we need, then explain how *Variable Density Syndrome Decoding* (VDSD) offers a nice solution, and finally discusses briefly the proof of security of VDSD, which originally contained some errors. In Section 7.3, we present a slightly changed VDSD assumption along with a better analysis coming from [CD23], which makes the VDSD usable in practice. Finally, Section 7.4 broaches some possible future improvements and further work on VDSD.

7.2 Variable Density Syndrome Decoding

For a better understanding of the content of this chapter, we invite the reader to have a look at Section 3.2, which introduces PCF and their constructions formally, and Chapter 4, as we will use the linear test framework as a security guarantee. For a better understanding of the motivations behind PCF, refer to Section 2.5.4.1.

7.2.1 The Search for an FSS-friendly WPRF: a Challenge

We have shown in Section 3.2 that PCF constructions for different correlations (VOLE, OT, Beaver Triples) exist, as long as we have access to an FSS-friendly WPRF, that is a WPRF that can be efficiently additively shared among the parties. Therefore, the search for a PCF construction boils down to the search for an FSS-friendly WPRF. Remember that known FSS constructions are pretty limited: we know how to share point functions, and therefore multi-point functions but more involved functions are for now out of reach. Next, we present iteratively different FSS-friendly WPRF tentative constructions that do not work, but which will provide some insights on the definition of VDSD. As for the PCG constructions, we make these WPRFs (and therefore the PCFs they construct) rely on some particular variant of the Syndrome Decoding assumption.

First, how can we construct a WPRF *under the* SD *assumption?* Let $\mathcal{F} = \{f_{\mathbf{k}}\}_{\mathbf{k}\in\{0,1\}^\lambda}$ be a pseudorandom function. Let $\mathbf{k} \xleftarrow{\$} \{0,1\}^\lambda$ be the secret key associated with the WPRF, with λ being the security parameter. The adversary is given n different samples of the WPRF, that is , n different pairs of the form $(\mathbf{x}^{(i)}, f_{\mathbf{k}}(\mathbf{x}^{(i)}))$,as explained in Section 2.3. A natural way to link WPRF to SD is to define $f_{\mathbf{k}}(\mathbf{x}) \coloneqq \langle \mathbf{k}, \mathbf{x}\rangle$, that is the inner product between $\mathbf{k}$ and $\mathbf{x}$. In that case, the adversary knows $(\mathbf{x}^{(i)}, \langle \mathbf{k}, \mathbf{x}^{(i)}\rangle)_{1\leqslant i\leqslant n}$. The transformation into an SD instance shape is straightforward. This is nothing else than giving to the adversary $(\mathbf{H}, \mathbf{H}\cdot\mathbf{e})$ where we defined $\mathbf{H}$ and $\mathbf{e}$ as follows:

$$\mathbf{H} = \begin{bmatrix} \mathbf{x}^{(1)} \\ \vdots \\ \mathbf{x}^{(n)} \end{bmatrix}, \qquad \text{and} \qquad \mathbf{e} = \mathbf{k}.$$

To sum up, it involves to view each input of the WPRF as a row in a matrix. As for the noise vector, it is defined as $\mathbf{e} = \mathbf{k}$. Therefore, an adversary able to distinguish the output of the WPRF would be able to distinguish $(\mathbf{H}, \mathbf{H}\cdot\mathbf{e})$ from $(\mathbf{H}, \mathbf{y})$ where $\mathbf{y} \xleftarrow{\$} \{0,1\}^n$: he would be able to solve the SD assumption.

We examine methods to share the WPRF function defined like this. First, we can consider the noise vector ($\mathbf{e} = \mathbf{k}$) dense but small, to be able to share it. Here "dense" means that it has about half of its entries non-zero and "small" means polynomial in the size of the security parameter λ. The

noise vector $\mathbf{e}$, being short, could be shared using FSS or even naïvely. However, a problem arises: the number of queries to the WPRF an adversary can make must be bounded by the size of the noise vector. Indeed, if the number of queries n is greater than the size of the key, then we would be able to solve the associated decoding problem using simple Gaussian Elimination techniques. The function would no longer be a WPRF ($\mathbf{H}$ would then be more tall than large). This bound on the number of queries goes against the spirit of a WPRF (and consequently of the spirit of the PCF): *we want the adversary to be able to request a number of evaluations of the function exponential in the security parameters λ.*

Therefore, we want the noise element in the matrix-vector representation of SD to be exponential in λ, to match the exponential in λ number of queries we want to give to the adversary. One solution would be to define the error vector not as equal to the key, but rather as equal to an exponential expansion of it. Therefore, we introduce the following function $\mathsf{Expd} : \{0,1\}^{\mathsf{poly}(\lambda)} \to \{0,1\}^{\exp(\lambda)}$, which takes a short seed $\mathbf{k}$ of size polynomial in the security parameter, and returns a very long (exponential in λ) but sparse vector. For example, a t-sparse exponential-sized vector can be described with a $t \cdot \mathsf{poly}(\lambda)$ vector (each of the t non-zero positions in $\mathbf{e}$ can be described with $\mathsf{poly}(\lambda)$ bits). Let Expd be the function that associates such a small vector to its t-sparse exponential-size equivalent.

$\mathbf{k}$ is still polynomial in the security parameter λ, and because Expd simply amounts to evaluating point functions on their entire domain, it can be efficiently shared via FSS schemes. At the same time, because the noise vector is now exponential in the security parameter, we can sustain an exponential number of queries from the adversary.

The WPRF would then be defined as $f_{\mathbf{k}}(\mathbf{x}) = \langle \mathsf{Expd}(\mathbf{k}), \mathbf{x} \rangle$. The associated syndrome decoding can be expressed as:

$$\left\{ \mathbf{H} \coloneqq \begin{bmatrix} \mathbf{x}^{(1)} \\ \vdots \\ \mathbf{x}^{(n)} \end{bmatrix}, \mathbf{H} \cdot \mathsf{Expd}(\mathbf{k}) \;\middle|\; \mathbf{k} \xleftarrow{\$} \mathbb{F}_2^{\mathsf{poly}(\lambda)} \right\} \approx \left\{ \mathbf{H}, \mathbf{y} \mid \mathbf{y} \xleftarrow{\$} \mathbb{F}_2^n \right\}.$$

Still, there is a caveat: in this situation, *the computational cost becomes an issue.* To compute the function, the parties (1) use an FSS scheme to share $\mathbf{e} = \mathbf{e_0} + \mathbf{e_1}$, and (2) multiply their respective $\mathbf{e}_{\boldsymbol{\sigma}}$ by $\mathbf{H}$. However, since the matrix $\mathbf{H}$ is exponentially large in λ, the multiplication would entail a prohibitive squared-exponential cost. Note that if $\mathbf{e}$ is sparse, it is not the case of the $\mathbf{e}_{\boldsymbol{\sigma}}, \sigma \in \{0,1\}$ which are dense.

To reduce the number of operations, a solution would be to make the rows of $\mathbf{H}$ sparse as well. To achieve this, we replace the rows of $\mathbf{H}$ by $\mathsf{Expd}(\mathbf{x})$, for $\mathbf{x} \in \{0,1\}^{\mathsf{poly}(\lambda)}$. The function would then be defined by $f_{\mathbf{k}}(\mathbf{x}) = \langle \mathsf{Expd}(\mathbf{k}), \mathsf{Expd}(\mathbf{x}) \rangle$, and the associated SD assumption could be written as:

$$\left\{ \mathbf{H} \coloneqq \begin{bmatrix} \mathsf{Expd}(\mathbf{x}^{(1)}) \\ \vdots \\ \mathsf{Expd}(\mathbf{x}^{(n)}) \end{bmatrix}, \mathbf{H} \cdot \mathsf{Expd}(\mathbf{k}) \;\middle|\; \mathbf{k} \xleftarrow{\$} \mathbb{F}_2^{\mathsf{poly}(\lambda)} \right\} \approx \left\{ \mathbf{H}, \mathbf{y} \mid \mathbf{y} \xleftarrow{\$} \mathbb{F}_2^n \right\}.$$

This would resolve the problem of the multiplication because each player would multiply their share e_σ by an exponential-but-sparse matrix $\mathbf{H}$. Unfortunately, this approach also fails: *in trying to achieve all these properties, we neglected to ensure that the result is still pseudorandom.* This is not the case here by any means: the inner products of two sparse vectors are likely to be biased toward 0, and therefore, it is not a WPRF. Next, we show how to navigate around all these constraints and find a tradeoff between these two non-working cases that achieves security.

7.2.2 A Solution: Variable Density Syndrome Decoding (a.k.a LPN)

In this subsection, we present a solution on how to design an FSS-friendly WPRF built under an SD-like assumption. We present the *regular* VDSD assumption from [BCGI+20a]. Other variants exist, but we will not delve into the details in this manuscript (see [BCGI+20a, section 6.1]).

Remark 7.2.1 (A note on terminology)**.** [BCGI+20a] introduced the terminology *Variable Density Learning Parity with Noise* (VDLPN). We have decided to replace this name with *Variable Density Syndrome Decoding* (VDSD) because LPN with fixed number of samples and SD are, in fact, equivalent in this setting and because coding theory is at the core of this thesis (see Remark 2.4.4).

7.2.2.1 VDSD intuition

In the previous subsection, we did not manage to find a proper pair of (matrix, vector) for which the associated syndrome decoding assumption is secure and provides a useful FSS-friendly WPRF. Nonetheless, the tentative solutions highlight some of the requirements we desire for our syndrome decoding assumption, and therefore for $\mathbf{H}$ and $\mathbf{e}$:

- We need exponentially many samples, and therefore $\mathbf{e}$ shall be exponentially long.
- We have to share the noise; therefore $\mathbf{e}$ has to be sparse (it is a consequence of FSS: we only know how to share sparse vectors).
- To be able to compute the matrix-vector multiplication, we need the matrix $\mathbf{H}$ to be sparse.
- However, multiplying a sparse matrix by a sparse vector does not produce pseudorandom elements.

We can identify two extremes that are computationally-wise acceptable and FSS-friendly: (1) a case where both the matrix and the error vectors are dense but small, which suffers from a drastic limitation in the number of calls to the WPRF; (2) a case where both the matrix and the error are sparse and exponentially long, but for which the WPRF is strongly biased (toward 0). *The intuition of* VDSD *is to get the best of both worlds.* What does this mean regarding attacks? We have shown that depending on the number of samples requested by the adversary, these two cases can be insecure. However, note that if the adversary requests an exponential number of evaluations of the WPRF, case (1) is broken but not case (2). Conversely, if the adversary requests a small number of evaluations of the WPRF, case (2) can be broken but not case (1). Therefore, it seems that each different pair (matrix, vector) protects against some attacks if not all.

It is then natural to wonder if a concatenation of a short but dense matrix and a sparse but long matrix, joined with a short and dense noise concatenated with a sparse and long noise, could resist attacks: for a given attack, *it suffices that only one of the two constituent matrices is resistant to this attack for the global matrix to be resistant.* Note that we are not considering all the attacks *in between* these two extremes, that is, those which take advantage of a moderate but not low number of calls to the WPRF. Therefore, this motivates the idea of taking a matrix composed of D blocks, each one providing resistance over a particular span, such that for every attack, there will be a block $\mathbf{H}_i$ that resists the attack, ensuring the resistance of the global matrix. More regularity is introduced in rVDSD, mostly for the sake of proof easiness and scheme simplicity.

7.2.2.2 Formal definition of VDSD

Let λ be a security parameter. We fix three parameters: a *sparsity* parameter $w = w(\lambda)$ (controlling the number of ones per row of a block), a *block* parameter $D = D(\lambda)$ (controlling the number

of blocks), and a bound $n = n(\lambda)$ on the number of samples. The reader can think of w, D as being $\Omega(\lambda)$, with $D < w$, and $n = 2^D$ for concreteness. We set $\mathsf{par} \coloneqq (w, D, n)$.

We will now describe a distribution of matrices and error vectors that will solve our problem. The matrix $\mathbf{H}$ will be described via a succession of distributions: first for the rows of sub-blocks, then for the sub-blocks, and finally for the matrix classes we are interested in.

Definition 7.2.1. *We denote by $\mathcal{S}_{1,2^i}$ the distribution of unit vectors of size 2^i.*

The construction is as follows:

- Let $\mathcal{K}_{w,i}$ be the distribution of random w-regular vectors over $\mathbb{F}_2^{w \cdot 2^i}$, i.e., the concatenation of w vectors sampled from $\mathcal{S}_{1,2^i}$.
- Each block of the matrix is then produced by the distribution that we call $\mathcal{H}^i_{\mathsf{par}}$ over $\mathbb{F}_2^{n \times (w \cdot 2^i)}$, where each row of the block is sampled independently from $\mathcal{K}_{w,i}$.
- Consequently, we denote by $\mathcal{H}_{\mathsf{par}}$ the distribution over $\mathbb{F}_2^{n \times 2n \cdot w}$, obtained by sampling the different blocks $\mathbf{H}_i \xleftarrow{\$} \mathcal{H}^i_{\mathsf{par}}$ for $i = 1$ to D and outputting $\mathbf{H} = [\mathbf{H}_1 || \cdots || \mathbf{H}_D]$, where $||$ denotes the horizontal concatenation. Note that the width of a block double each time i increases.
- Eventually, we denote by $\mathcal{N}_{\mathsf{par}}$ the noise distribution obtained by sampling each block of the noise $\mathbf{e_i}^\top$ according to $R_{w,i}$ and outputting $\mathbf{e} \leftarrow (\mathbf{e_1} /\!/ \cdots /\!/ \mathbf{e_D}) \in \mathbb{F}_2^{2n \cdot w}$ where $/\!/$ is this time the vertical concatenation.

The matrix $\mathbf{H}_i$ sampled from $\mathcal{H}^i_{\mathsf{par}}$ is:

$$\mathbf{H}_i = \begin{bmatrix} & & \overbrace{\phantom{\mathbf{u}^i_{1,w}}}^{2^i \text{ columns}} \\ \mathbf{u}^i_{1,1} & \cdots & \mathbf{u}^i_{1,w} \\ \vdots & \ddots & \vdots \\ \mathbf{u}^i_{n,1} & \cdots & \mathbf{u}^i_{n,w} \end{bmatrix} \coloneqq \left[\mathbf{M}^i_1, \cdots, \mathbf{M}^i_w\right],$$

where $(\mathbf{u}^i_{k,j})_{1 \leqslant k \leqslant n, 1 \leqslant j \leqslant w}$ are sampled from the distribution $\mathcal{S}_{1,2^i}$, and are unit vectors over $\mathbb{F}_2^{2^i}$. The $\mathbf{M}_i$ are matrices of size $2^{n \times 2^i \cdot w}$, and are defined by the equation. Thus, there are w non-zero entries per row. Eventually, the matrix $\mathbf{H}$ sampled from $\mathcal{H}_{\mathsf{par}}$ is a horizontal concatenation of the $\mathbf{H}_i$:

$$\mathbf{H} = \underbrace{\begin{bmatrix} \mathbf{H}_1 & \cdots & \mathbf{H}_D \end{bmatrix}}_{w \dot{2}^{D+1} \text{ columns}}.$$

The term *variable density* refers to the fact that the density of 1's in each block $\mathbf{H}_i$ is $1/2^i$ by construction: it decreases exponentially with i. Let $\mathbf{H}$ sampled from the distribution $\mathcal{H}_{\mathsf{par}}$, we denote by $\mathcal{O}_{\mathsf{par}}(\mathbf{H})$ the distribution which samples $\mathbf{e} \xleftarrow{\$} \mathcal{N}_{\mathsf{par}}$ and returns $\mathbf{H} \cdot \mathbf{e}$. We display the matrix $\mathbf{H}$ in Figure 7.1.

We can now define VDSD. Not surprisingly, it is again a particular restriction of SD for the special case of using the distribution $\mathcal{H}_{\mathsf{par}}$ and $\mathcal{N}_{\mathsf{par}}$.

Definition 7.2.2 ($\mathsf{rVDSD}(w, D, n)$)**.** *The regular Variable Density Syndrome Decoding* (rVDSD) *assumption, with parameters* $\mathsf{par} = (w, D, n)$*, denoted* $\mathsf{rVDSD}(w, D, n)$*, states that:*

$$\left\{(\mathbf{H}, \mathbf{b}) \mid \mathbf{H} \xleftarrow{\$} \mathcal{H}_{\mathsf{par}}, \mathbf{e} \xleftarrow{\$} \mathcal{N}_{\mathsf{par}}, \mathbf{b} \leftarrow \mathbf{H} \cdot \mathbf{e}\right\} \approx \left\{(\mathbf{H}, \mathbf{b}) \mid \mathbf{H} \xleftarrow{\$} \mathcal{H}_{\mathsf{par}}, \mathbf{b} \xleftarrow{\$} \mathbb{F}_2^n\right\}.$$

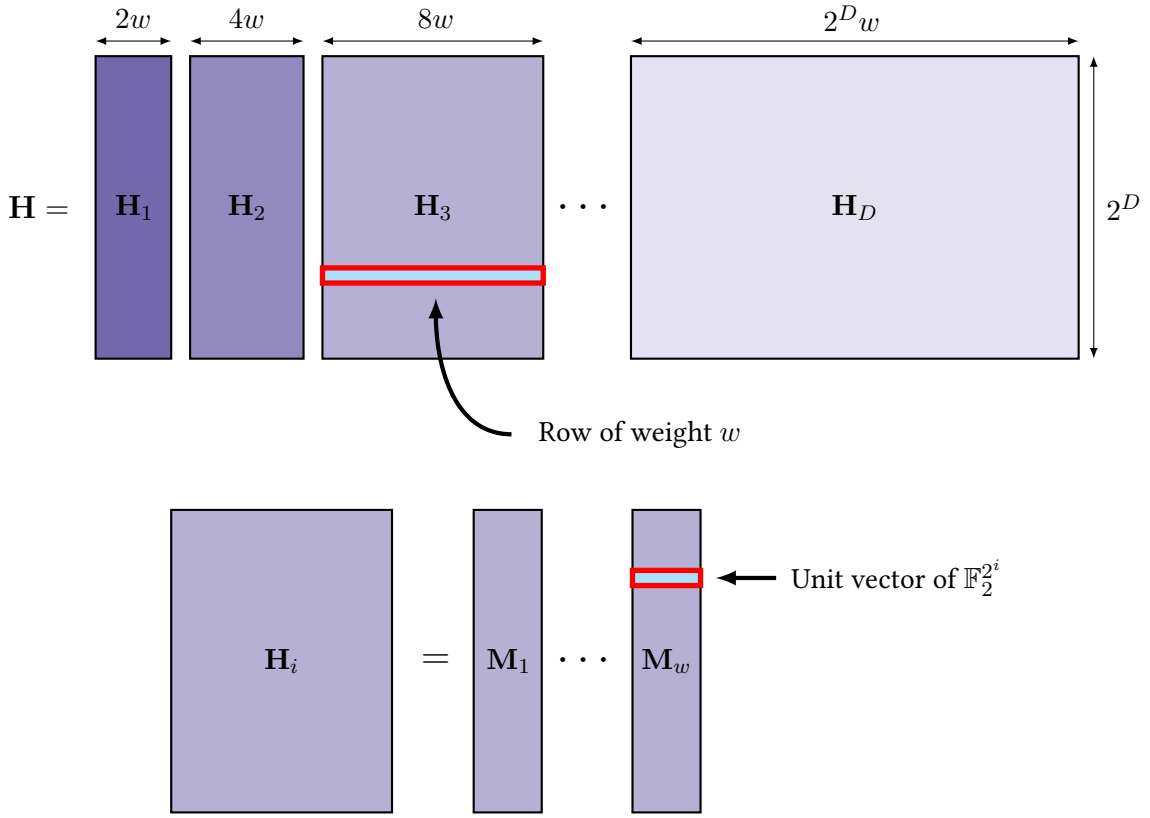

Figure 7.1 – Representation of the parity-check matrix used in rVDSD.

7.2.2.3 A WPRF Candidate from the rVDSD Assumption

We show that rVDSD naturally defines an FSS-friendly WPRF. Fix parameters $\mathsf{par}(\lambda) = (w(\lambda), D(\lambda), n(\lambda) = 2^{D(\lambda)})$. Recall that a vector from the distribution $\mathcal{N}_{\mathsf{par}}$ is, in fact, the vertical concatenation of D vectors $(\mathbf{e_i})_{1 \leqslant i \leqslant D}$, where $\mathbf{e_i}$ is the transpose vector of a vector from the distribution $\mathcal{K}_{w,i}$. Moreover, $\mathcal{K}_{w,i}$ is the concatenation of w unit vectors over $\mathbb{F}_2^{2^i}$, where each of them can be generated with i random bits (encoding the index of the nonzero entry). Therefore, sampling a vector $\mathcal{N}_{\mathsf{par}}$ requires exactly $w \cdot \sum_{i=1}^{D} i = w \cdot D(D-1)/2$ random bits, which is still polynomial in the security parameter. Let $\mathcal{N}_{\mathsf{par}}(r)$ denote the vector $\mathbf{e}$ sampled from $\mathcal{N}_{\mathsf{par}}$ using randomness r. Then the WPRF can be constructed as below:

- Key size: $\mathbf{k} \in \{0,1\}^{\pi(\lambda)}$ with $\pi(\lambda) = \rho(\lambda) = w \cdot D(D-1)/2$
- Input size : $\mathbf{x} \in \{0,1\}^{\rho(\lambda)}$ with $\rho(\lambda) = w \cdot D(D-1)/2$
- $F_{\mathbf{k}}(\mathbf{x})$: on input $\mathbf{x} \in \{0,1\}^{\rho(\lambda)}$, sample $\mathbf{h}^{\top} \leftarrow \mathcal{N}_{\mathsf{par}}(\mathbf{x})$ and output $\langle \mathbf{h}, \mathcal{N}_{\mathsf{par}}(\mathbf{k}) \rangle$

Theorem 7.2.1 ([BCGI+20a]). *Suppose that* rVDSD(par) *holds. Then the above construction is an* n*-query* WPRF*, with input length and key length equal at* $w \cdot D(D-1)/2$.

The next subsection will cover exactly how the authors established the security of the rVDSD assumption. Then, the following section will analyze a slightly different assumption and also prove its security.

7.2.3 Original Proof of Security, Errors and Fixing

The security of the PCF boils down to the security of the underlying FSS scheme and WPRF scheme. [BCGI+20a] introduced the *linear attack framework* (see Section 4.1) to prove that the WPRF based on rVDSD is secure against a large class of attacks. Next, we present their attempt to prove the security of rVDSD against linear tests, highlight some errors in their analysis, and explain how to fix the mistakes.

7.2.3.1 Overview of the Original Proof of Security

Warning: The following present the main idea of the security proof from [BCGI+20a]. It contains some errors, that we will repeat here in order to explain the correction. In what follows, two equations are therefore wrong: Equations (7.1) and (7.2). Nevertheless, the general idea of the proof remains valid, and we will fix the mistakes.

The goal of this analysis is to show that the rVDSD assumption cannot be broken by any *linear test*, which captures, in particular, *almost* all attacks against SD. We restate Definition 4.2.2 and Equation (4.1) with the exact notation introduced with rVDSD:

Theorem 7.2.2 (Resistance against linear tests)**.** *There exist constants* (Γ, μ, ν)*, such that for any large enough* w*, any* $\Gamma \cdot D \leqslant w, := 2^D, \mathsf{par} := (w, D, n)$*, it holds that*

$$\Pr_{\mathbf{H} \xleftarrow{\$} \mathcal{H}_{\mathsf{par}}} [\ \mathsf{bias}(\mathcal{O}_{\mathsf{par}}(\mathbf{H})) > \mu^w\ ] \leqslant \nu^w,$$

where $\mathcal{O}_{\mathsf{par}}(\mathbf{H})$ denotes the distribution which samples $\mathbf{e} \xleftarrow{\$} \mathcal{N}_{\mathsf{par}}$ and returns $\mathbf{H} \cdot \mathbf{e}$. Recall that this theorem states that with high probability (at least $1 - \nu^w$), over the choice of a possible exponential number ($n = 2^D$) of random inputs $(\mathbf{x^{(1)}}, \ldots, \mathbf{x^{(n)}})$, a distinguisher that computes a linear function of the entire output string $\mathbf{y} = (f_{\mathbf{k}}(\mathbf{x^{(1)}}), \ldots, f_{\mathbf{k}}(\mathbf{x^{(n)}}))$ has an advantage smaller than μ^w to distinguish the string from uniform. Note that the choice of the linear function can depend arbitrarily on $(\mathbf{x^{(1)}}, \ldots, \mathbf{x^{(n)}})$.

It was stated that the goal of the construction will be to ensure that, for a given number of samples chosen, there exists a matrix sub-block that offers resistance against this attack. This is equivalent to saying that for all attack vectors $\mathbf{v}$, there exists a specific block $\mathbf{H}_i$ that offers protection against it. In practice, for attack vectors of weight $wt(\mathbf{v}) \in [2^{i-1}, 2^i]$, the block $\mathbf{H}_i$ will be the one ensuring security. Therefore, we want to analyze, for $wt(\mathbf{v}) \in [2^{i-1}, 2^i]$ and $\mathbf{H}_i \xleftarrow{\$} \mathcal{H}^i_{\mathsf{par}}$ the bias of the distribution $\mathcal{O}^i_{\mathsf{par}}(\mathbf{H}_i) := \{\mathbf{v}\mathbf{H}_i\mathbf{e_i} \mid \mathbf{e_i} \xleftarrow{\$} \mathcal{K}_{w,i}\}$. To bound the bias of $\mathcal{O}^i_{\mathsf{par}}(\mathbf{H}_i)$ the authors thus analyzed the bias induced by all the w smaller sub-blocks of $\mathbf{H}_i$, and introduced a notion of *good* and *bad* matrices:

Definition 7.2.3. *Take* $i > 0$*. Given a matrix* $\mathbf{M}_j \in \mathbb{F}_2^{n \times 2^i}$*,* $\mathbf{M}_j$ *is judged* bad *with respect to a vector* $\mathbf{v} \in \mathbb{F}_2^n$ *if*

$$wt\left(\mathbf{v}^\top \cdot \mathbf{M}_j\right) \notin \left[\frac{2^i}{5}, \frac{2^{i+2}}{5}\right].$$

Moreover, given w *matrices* $(\mathbf{M}_1, \ldots, \mathbf{M}_w)$ *in* $\mathbb{F}_2^{n \times 2^i}$*, we let* $N_{\mathbf{v}}(\mathbf{M}_1, \ldots, \mathbf{M}_w)$ *be the number of matrices that are bad with respect to* $\mathbf{v}$ *among* $\mathbf{M}_1, \ldots, \mathbf{M}_w$*.*

This means that a matrix is said to be "bad" with respect to a vector $\mathbf{v}$ if the bias it induces against the test vector $\mathbf{v}$ is too large: here large means $> 3/10 = 1/2 - 1/5$, the constant is chosen arbitrarily.

The goal of the proof is to guarantee that, with high probability, at least half of the matrices $\mathbf{M}_j$ of $\mathbf{H}_i$ are good, thereby obtaining an upper bound on the bias of the full block $\mathbf{H}_i$. This is what is stated in the following lemma.

Lemma 7.2.1. *There is a constant C, such that for any $1 \leqslant i \leqslant D$, and for any vector $\mathbf{v} \in \mathbb{F}_2^n$ such that $wt(\mathbf{v}) \in [2^{i-1}, 2^i]$, it holds that*

$$\Pr_{[\mathbf{M}_1,\ldots,\mathbf{M}_w] \xleftarrow{\$} \mathcal{H}^i_{\mathsf{par}}} \left[N_{\mathbf{v}}(\mathbf{M}_1, \ldots, \mathbf{M}_w) \geqslant \frac{w}{2} \right] \leqslant 2^{-C \cdot 2^i \cdot w}.$$

The above lemma states that the bad situation where more than half of the $\mathbf{M}_j$ are bad is very unlikely. As a consequence, with high probability, for any fixed vector $\mathbf{v}$ of weight in $[2^{i-1}, 2^i]$, the distribution $\mathcal{O}^i_{\mathsf{par}}(\mathbf{H}_i)$ has low bias. Note that the probability of this bad occurrence is so low that it remains low even after a union bound over *all* vectors $\mathbf{v}$ of weight in $[2^{i-1}, 2^i]$. Hence, looking at the output $\mathbf{H} \cdot \mathbf{e} = \sum_i \mathbf{H}_i \cdot \mathbf{e}_i$, each component $\mathbf{H}_i \cdot \mathbf{e}_i, 1 \leqslant i \leqslant D$ will guarantee low-bias against all attack vectors whose weight lies in $[2^{i-1}, 2^i]$. The XOR of these independent samples will inherit the low-bias of all its components, and therefore resist all linear tests.

To prove the previous lemma, it is necessary to bound the number of bad matrices. In [BCGI+20a], the authors reformulate the event that a matrix $\mathbf{M}_j$ is *bad* as a *balls and bins* problem. Let $\mathbf{H}_i \xleftarrow{\$} \mathcal{H}^i_{\mathsf{par}}$. By definition $\mathbf{H}_i = [\mathbf{M}_1, \cdots, \mathbf{M}_w]$, with $\mathbf{M}_j \in \mathbb{F}_2^{n \times 2^i}, 1 \leqslant j \leqslant w$. Recall that by definition of $\mathcal{H}^i_{\mathsf{par}}$, the rows of each $\mathbf{M}_j$ are generated independently from $\mathcal{S}_{1,2^i}$. By considering that each column of $\mathbf{M}_j$ can be seen as a bin, sampling a row of $\mathbf{M}_j$ is equivalent to taking 2^i empty bins and throwing a ball randomly into one of the 2^i bins. Then, for a vector $\mathbf{v}$ of weight $l \in [2^{i-1}, 2^i]$, the result of $\mathbf{v}^\top \cdot \mathbf{M}_j$ is equivalent to throwing l balls into 2^i bins and emptying all the bins containing an even number of balls (this is because we work over $\mathbb{F}_2$). One can also imagine that the bins work modulo 2, and therefore turn out to be empty after an even number of balls. Consequently, the event $wt\left(\mathbf{v}^\top \cdot \mathbf{M}_j\right) \notin \left[\frac{2^i}{5}, \frac{2^{i+2}}{5}\right] := \mathcal{I}_i$ is equivalent to the following event: after randomly throwing l balls into 2^i bins, the number T of bins that contain an odd number of balls satisfies $T \notin \mathcal{I}_i$.

We therefore define the following experiment:

- Take 2^i bins and throw l balls into the bins in w consecutive phase.
- Each time that l balls have been thrown, we check that the proportion of the number of bins that contain an odd number of balls is between $1/5$ and $4/5$, and clear out the bins.
- In the end, we return *failure* if more than $w/2$ of the w checks have failed.

To bound the probability of returning a failure, we define the following *cost function* $\Phi\left(\mathbf{X}_{\mathbf{1,1}}, \cdots, \mathbf{X}_{\boldsymbol{l,w}}\right) := \sum_{k=1}^{w} \left(2^{i-1} - \left|wt\left(\bigoplus_{j=1}^{l} \mathbf{X}_{\boldsymbol{j,k}}\right) - 2^{i-1}\right|\right)$, where each $\mathbf{X}_{\boldsymbol{j,k}}, 1 \leqslant j \leqslant l, 1 \leqslant k \leqslant w$, is the random variable corresponding to the bin in which the j-th ball of the k-th phase was thrown (seen as a length-2^i unit vector with a 1 at the bin position). The $\mathbf{X}_{\boldsymbol{j,k}}$ are independent. Bounding the number of bad matrices, the authors claimed, amounts to bounding Φ. More precisely they claim that [2]

$$\Pr_{[\mathbf{M}_1,\ldots,\mathbf{M}_w] \xleftarrow{\$} \mathcal{H}^i_{\mathsf{par}}} \left[n_{\mathbf{v}}(\mathbf{M}_1, \ldots, \mathbf{M}_w) \geqslant \frac{w}{2} \right] \leqslant \Pr\left[\Phi(X_{1,1}, \cdots, X_{l,w}) < \frac{w \cdot 2^i}{10}\right]. \quad (7.1)$$

2. This equation turns out to be false after precise examination — the constants are not correct — but it does not change the spirit of the proof.

Afterwards, it suffices to bound Φ to conclude. The claim is that the following bound holds:

$$\Pr\left[\Phi(\mathbf{X_{1,1}},\ldots,\mathbf{X_{l,w}}) < \frac{w\cdot 2^i}{10}\right] \leqslant 2^{-C\cdot 2^i\cdot w}. \tag{7.2}$$

The authors chose Φ as a suitable function to analyze: it is 2-Lipschitz, meaning that changing one of the input of Φ can only change its output by at most 2. Strong concentration bounds are known for Lipschitz functions, such as the McDiarmid inequality Lemma 2.2.3. Applying the McDiarmid inequality directly gives us the desired result. The last remaining tool is to establish a lower bound on the expectancy of Φ, as it appears in the McDiarmid inequality. This lower bound is stated in the following lemma:

Lemma 7.2.2.

$$\mathbb{E}\left[\Phi\left(\mathbf{X_{1,1}},\ldots,\mathbf{X_{l,w}}\right)\right] \geqslant \frac{w\cdot 2^i}{5}.$$

Assuming Lemma 7.2.2, the McDiarmid inequality provides a bound on Φ, which translates to a bound on $n_{\mathbf{v}}$ by Equation 7.1. A union bound over all vectors of weight between $[2^{i-1}, 2^i]$ allows to conclude:

$$\Pr_{[\mathbf{M}_1,\cdots,\mathbf{M}_w]\xleftarrow{\$}\mathcal{H}^i_{\mathsf{par}}}\left[\exists \mathbf{v}\in\mathcal{S}_{i,n}, N_{\mathbf{v}}(\mathbf{M}_1,\ldots,\mathbf{M}_w) \geqslant \frac{w}{2}\right] \leqslant 2^{D\cdot 2^i}\cdot 2^{-C\cdot w\cdot 2^i} \leqslant 2^{-a\cdot w},$$

with $a = \frac{C}{2} > D$. The proof ends with a last union bound over all matrices $\mathbf{H_i}$, for $1 \leqslant i \leqslant D$.

Some notations. In the following, and for the rest of the chapter, we will denote by $\mathbf{X_{j,k}}$ the bin into which the j-th ball of the k-th phase is thrown ($\mathbf{X_{j,k}}$ is a unit vector). Given a test vector $\mathbf{v}\in\mathbb{F}_2^n$ of weight $wt(\mathbf{v}) = l$, we define $R_{i,l,k} = wt\left(\mathbf{v}^\top\cdot\mathbf{M}\right) = wt\left(\bigoplus_{j=1}^l \mathbf{X_{j,k}}\right)$. That is, $R_{i,l,k}$ it is the number of bins that contain an odd number of 1 in the k-th phase; we usually write it $R_{l,k}$ when i is clear from the context. We further define $Z_{i,l,k}$ as $Z_{i,l,k} = |2^{i-1} - R_{l,k}|$ (also usually written $Z_{l,k}$). Eventually, we denote by $S_{i,n}$ the set of vectors $\mathbf{v}\in\mathbb{F}_2^n$ with $wt(\mathbf{v})\in[2^{i-1},2^i]$.

7.2.3.2 Errors in the Proof

As already mentioned, while the proof strategy seems sound and appropriate, it contains some errors that invalidate the proof. The first error is minor and can be corrected with a simple change of constant. However, fixing the second error is more delicate.

- The first error appeared in Equation (7.1): the constant in the left hand part is incorrect. Fortunately, it seems to be due to a reversed inequality and can be straightforwardly fixed by adjusting the constant.
- The second and main error appears in the proof of Equation (7.2). The error stems from an incorrect analysis in Lemma 7.2.2. After calculating an upper bound on the expectation $\mathbb{E}\left[wt\left(\bigoplus_{j=1}^l \mathbf{X_{j,k}}\right)\right]$, the authors deduced a bound on $\mathbb{E}\left[\left|2^{i-1} - wt\left(\bigoplus_{j=1}^l \mathbf{X_{j,k}}\right)\right|\right]$. However, a bound on $\mathbb{E}[Z]$ does not imply a bound on $\mathbb{E}[|Z-b|]$ in general (and typically when Z is "concentrated away" from b). We believe the error stems from a use of the Jensen inequality in the wrong direction. Up to the choice of the constant $1/5$ (the proof actually only requires any constant below $1/2$), the lemma remains true; however, proving the lemma fundamentally requires characterizing the shape of the distribution of the random variable $wt\left(\bigoplus_{j=1}^l \mathbf{X_{j,k}}\right)$. This turns out to be non-trivial.

7.2.3.3 Fixing the Main Error

Next, we provide a sketch of how we managed to fix the main error. The new proof correcting all the errors can be found in Appendix C. As stated previously, the main error stemmed from using the Jensen inequality in the wrong direction. Fortunately, it is possible to obtain a similar result from a deeper analysis (although with worse constant parameters). What we aim to achieve is to measure exactly how concentrated the random variable Z is from the expectancy. Our main contribution to this analysis is the proof of the following lemma:

Lemma 7.2.3 (Fixing of the second error)**.** *For all $n \in \mathbb{N}$, there exists $\beta < 1/2$ such that $\mathbb{E}[\mathbb{Z}_{l,k}] < \beta \cdot n$.*

Note that by design the inequality $\mathbb{E}[\mathbb{Z}_{l,k} \leqslant 1/2 \cdot n]$ holds, but we need more than that. The further away from $1/2$ we are, the better the efficiency parameters.

Sketch. The proof consists in finding an upper bound on both $\Pr[R_{l,k} \geqslant p \cdot n]$ and $\Pr[R_{l,k} \leqslant (1-p) \cdot n]$ for $p \in [\frac{1}{2}, 1]$ and using it to find the the bound on $\mathbb{E}[Z_{l,k}] = \sum_{j=0}^{2^{i-1}-1} \Pr\left(\left|R_{l,k} - 2^{i-1}\right| > j\right)$.

Lemma 7.2.4. *Let $n = 2^i > 2^7$, $l \in [2^{i-1}, 2^i]$ and $\mu = (1 - \frac{1}{n})^l$. There exists $0.5 \leqslant p \leqslant 1$ such that with $\theta = \frac{pn - l/2}{\mu} - 1$, it holds that*

$$\max\left(\Pr\left[R_{l,k} \geqslant pn\right], \Pr\left[R_{l,k} \leqslant (1-p)\, n\right]\right) \leqslant 2 \exp\left(-\frac{\theta^2 \mu^2 (n - \frac{1}{2})}{n^2 - \mu^2}\right).$$

To prove this lemma, we use the Occupancy Bound for balls and bins from Lemma 2.2.4. The occupancy bound is about the proportion of empty bins but can be shrewdly transformed to relate to our specific problem, which focuses on parity in bins. This concludes the sketch. □

The end of the proof is the same as in the original proof, up to handling separately the case of small i's. Refer to the Appendix C for the details.

7.3 Making VDSD Efficient

In this section, we explain the efforts undertaken to make VDSD efficient. We managed to reduce the security parameters from an unbearable $w \approx 10^7$ to $w \approx 380$, using three different kinds of optimizations. First, we noted that the analysis was imprecise for small i. This led to a loss in the parameters. The solution was to eliminate this side effect by replacing the first blocks of the construction with a truly random block. The second major gain in efficiency came from a much better analysis, which took into account blocks in a more precise manner. Finally, we also used some computer-based simulations to get a bound on the actual value of β, again increasing efficiency.

7.3.1 VDSD 2.0

Next, we present some minor changes in the rVDSD assumption that helped reduce the bad side effects while preserving all the desired properties. Where do the side effects come from? Remember that in the rVDSD assumption, the matrix $\mathbf{H}$ is a concatenation of D blocks $\mathbf{H}_i$, where each $\mathbf{H}_i$ is a concatenation of w sub-blocks $\mathbf{M}_i$, each sub-blocks having for rows unit vectors of size 2^i. The goal of the analysis was to bound the bias of $\mathbf{H}_i \mathbf{e_i}$ against attack vectors of size $\theta(2^i)$. However, the bounds from the new correct analysis (see Appendix C) turned out to be much worse for small

constant values of i. Moreover, for very small i, the general proof does not suffice, and we had to conduct a tailored analysis. Therefore, the analysis suffers from the first blocks $\mathbf{H}_1, \mathbf{H}_2, \ldots$. We propose to replace the $i^* - 1$ first blocks $\mathbf{H}_1||\mathbf{H}_2||\ldots||\mathbf{H}_{i^*-1}$ by a random matrix $\mathbf{R}$, for i^* a fixed small constant. In this new VDSD assumption, the matrix has therefore the following shape:

$$\mathbf{H} = \begin{bmatrix} \mathbf{R} & \mathbf{H}_{i^*} & \ldots & \mathbf{H}_D \end{bmatrix}.$$

Because $\mathbf{R}$ is a uniformly random matrix, it will resist linear tests as long as it has an appropriate width (number of columns). This is important because $\mathbf{He} = \mathbf{Re_r} + \sum_{i=i^*}^{D} \mathbf{H}_i \cdot \mathbf{e_i}$, with $\mathbf{e_r}$ being a uniformly random vector. The question is how wide this matrix has to be: if it is not too big, then the computations can still be performed.

Let t be the width of $\mathbf{R}$. We will show that for a vector $\mathbf{v}$ with small Hamming weight, $\mathbf{Re_r}$ has no bias. Returning to the framework of linear tests, we show that the matrix $\mathbf{R}$ resists linear tests. As shown in Section 4.3, resistance in the model of linear tests against vectors of weight below d is related equivalent to saying that the code generated by the rows of $\mathbf{R}$, which is a random linear code of dimension $2^D - t$, has minimum distance at least d. Random codes are well-studied, and bounds exist on the probability that a random code $\mathbf{S}$ of dimension $2^D - t$ and codeword length 2^D has a minimum distance inferior to some bound d. Indeed:

$$\Pr[\mathbf{S} \text{ has minimum distance } < d] \leqslant 2^{-t+H_2(d/2^D)\cdot 2^D},$$

where $H_2(x) = -x\log x - (1-x)\log(1-x)$ is the binary entropy function. Therefore, for a target probability of 2^{-128}, we need to pick $t = H_2(d/2^D) \cdot 2^D + 128$. In our case, we fixed $i^* = 5$. Therefore, we want to take a matrix $\mathbf{R}$ that will resist attacks of weight at most $2^{i^*-1} - 1 = 15$. For a number of samples $2^D = 2^{30}$, this results in $t = 541$.

We can therefore, define the WPRF as before, and note that it is still FSS-friendly: indeed, because the block $\mathbf{R}$ that we are adding is still small, the number of operations induced by it remains small even if it is dense. In fact, $\mathbf{R}$ has even a smaller description size than $\mathbf{H}_1||\ldots||\mathbf{H}_{i^*-1}$: the latter has exactly $w \cdot (i^* - 1)$ ones. The analysis of the security parameters results in a $w \approx 380$, in other words, 1520 ones in comparison to the 541 (at most) ones of $\mathbf{R}$.

Remark 7.3.1 (Droping the small complexity class property)**.** The WPRF from the [BCGI+20a] also enjoys the interesting property of being in a low complexity class: the WPRF is in fact in the depth-2 $\mathsf{AC}^0[\oplus]$ class, meaning it can be computed from a depth-2 circuit, with a layer of AND gates at the bottom and a single XOR gate at the top (both of arbitrary fan-in). In contrast, this VDSD 2.0 is no longer in this class: we sacrificed the perfect regularity of the matrix to gain efficiency.

7.3.2 Security Analysis

For the rest of the analysis, we assume that we start with $i \geqslant i^* = 5$. We stick with the notations introduced by the previous proof of security, in Section 7.2.3.1. The adversary chooses an attack vector $\mathbf{v}$ of Hamming weight $l \in [2^{i-1}, 2^i]$. We use the following random variable:

$$\mathbf{Z_{i,l,k}} = \left| wt\left(\bigoplus_{j=1}^{l} \mathbf{X_{j,k}}^{(i)} \right) - 2^{i-1} \right|.$$

Unlike the original proof (see Section 7.2.3.1), this time we aim for a much more direct strategy. Recall the distribution from Section 7.2.2.2: $\mathcal{O}_{\mathsf{par}}(\mathbf{H})$ is the distribution which samples $\mathbf{e} \xleftarrow{\$} \mathcal{N}_{\mathsf{par}}$

and returns $\mathbf{H} \cdot \mathbf{e}$. The goal of the proof is still to demonstrate that the bias induced by $\mathcal{O}_{\mathsf{par}}(\mathbf{H})$ is exponentially small with exponentially close to 1 probability. We will rewrite the bias using the above random variable $\mathbf{Z}_{\boldsymbol{i,l,k}}$, which we will analyze next. As before, we decompose the distribution $\mathcal{O}_{\mathsf{par}}(\mathbf{H}) = \bigoplus_{i \geqslant i^*} \mathcal{O}^i_{\mathsf{par}}(\mathbf{H}_i) \oplus \mathcal{O}_{\mathsf{par}}(\mathbf{R})$. Here $\mathcal{O}^i_{\mathsf{par}}(\mathbf{H}_i)$ is the distribution that samples $\mathbf{e}_{\boldsymbol{i}}$ (as a concatenation of w length-2^i unit vectors) and outputs $\mathbf{H}_i \cdot \mathbf{e}_{\boldsymbol{i}}$. Similarly $\mathcal{O}_{\mathsf{par}}(\mathbf{R})$ is the distribution that samples a random vector $\mathbf{e}_{\boldsymbol{r}}$ of length t and output $\mathbf{R} \cdot \mathbf{e}_{\boldsymbol{r}}$. For any test vector $\mathbf{v}$, we have $\mathsf{bias}_{\mathbf{v}}(\mathcal{O}_{\mathsf{par}}) \leqslant \mathsf{bias}_{\mathbf{v}}(\mathcal{O}^i_{\mathsf{par}})$. Therefore, we focus on bounding the bias against a test vector $\mathbf{v}$ of $\mathcal{O}^i_{\mathsf{par}}$. We have:

$$\mathsf{bias}_{\mathbf{v}}(\mathcal{O}^i_{\mathsf{par}}) = \left| \frac{1}{2} - \Pr\left[\, (\mathbf{v}^\top \cdot \mathbf{H}_i) \cdot \mathbf{e}_{\boldsymbol{i}} = 1 \, \right] \right|.$$

$(\mathbf{v}^\top \cdot \mathbf{H}_i) \cdot \mathbf{e}_{\boldsymbol{i}}$ is the XOR of w independent terms $(\mathbf{v}^\top \cdot \mathbf{H}_{\boldsymbol{i,j}}) \cdot \mathbf{e}_{\boldsymbol{i,j}}$ where each $\mathbf{e}_{i,j}$ is a length-2^i unit vector. Again, we can decompose the distribution $\mathcal{O}^i_{\mathsf{par}}$ into w smaller distributions $\mathcal{D}^{(1)}_{\mathsf{par}}(\mathbf{H}_i), \ldots, \mathcal{D}^{(w)}_{\mathsf{par}}(\mathbf{H}_i)$ such that $\mathcal{O}^i_{\mathsf{par}} = \bigoplus_{k=1}^{w} \mathcal{D}^{(k)}_{\mathsf{par}}(\mathbf{H}_i)$. Using the Piling-up lemma Lemma 2.2.2, obtaining an upper bound on the bias of $\mathcal{O}^i_{\mathsf{par}}$ is equivalent to upper bounding the bias against each of the smaller distributions.

$$\Pr\left[\bigoplus_{k=1}^{w} \mathcal{D}_k = 1 \right] = \frac{1}{2} \left(1 - \prod_{k=1}^{w} \left(1 - \frac{R_{i,l,k}}{2^{i-1}} \right) \right).$$

Therefore, we obtain an expression of the bias of $\mathcal{O}^i_{\mathsf{par}}$ in terms of the $Z_{i,l,k}$ random variables:

$$\mathsf{bias}_{\mathbf{v}}(\mathcal{O}^i_{\mathsf{par}}) = \frac{1}{2} \cdot \prod_{k=1}^{w} \frac{Z_{i,l,k}}{2^{i-1}}.$$

Fix any bound B. Then by the above,

$$\Pr[\, \mathsf{bias}_{\mathbf{v}}(\mathcal{O}^i_{\mathsf{par}}) > B \,] = \Pr\left[\prod_{k=1}^{w} Z_{i,l,k} > 2^{(i-1)w} \times (2B) \right].$$

We have finally managed to express our partial bias only with the variable $Z_{i,l,k}$. The study of this random variable will complete the proof. However, observe that on the right-hand side, the $Z_{i,l,k}$ appears in a product. This is not convenient, because usual concentration bounds concern primarily *sums* of random variables and not products. The key observation is that if a product of values is minimized by a bound $\mathcal{B}$, the sum of these values is minimized when all the values are equal and therefore minimized by $w\mathcal{B}^{1/w}$. In our case, this implies that whenever $\prod_{k=1}^{w} Z_{i,l,k} > 2^{(i-1)w} \times (2B)$, it necessarily further holds that

$$\sum_{k=1}^{w} Z_{i,l,k} > w \cdot \left(2^{(i-1)w} \times (2B) \right)^{1/w}.$$

Therefore

$$\Pr\left[\prod_{k=1}^{w} Z_{i,l,k} > 2^{(i-1)w} \times (2B) \right] \leqslant \Pr\left[\sum_{k=1}^{w} Z_{i,l,k} > w \cdot \left(2^{(i-1)w} \times (2B) \right)^{1/w} \right],$$

and finally,

$$\Pr\left[\ \mathsf{bias}_{\mathbf{v}}(\mathcal{O}^i_{\mathsf{par}}) > B\ \right] \leqslant \Pr\left[\sum_{k=1}^{w} Z_{i,l,k} > w \cdot 2^{(i-1)} \cdot c\right], \tag{7.3}$$

where $c = (2B)^{\frac{1}{w}}$. We managed successfully to obtain an expression using a sum of $Z_{i,l,k}$. As in the previous proof, we can now reintroduce the function $\Phi\left(\mathbf{X_{1,1}}, \ldots, \mathbf{X_{l,w}}\right)$:

$$\Phi\left(\mathbf{X_{1,1}}, \ldots, \mathbf{X_{l,w}}\right) = 2^{i-1} \cdot w - \sum_{k=1}^{w} Z_{i,l,k} = \sum_{k=1}^{w}\left(2^{i-1} - \left|wt\left(\bigoplus_{j=1}^{l} \mathbf{X_{j,k}}\right) - 2^{i-1}\right|\right).$$

This function is 2-Lipschitz, meaning that a change of input shall entail a difference in the output of at most 2, and therefore we could still apply the McDiarmids's inequality (see Lemma 2.2.3). First, note that:

$$\Pr\left[\ \Phi < \mathbb{E}[\Phi] - t\ \right] = \Pr\left[\sum_{k=1}^{w} Z_{i,l,k} > w \cdot \left(\mathbb{E}[Z_{i,l}] + 2^i \cdot \zeta\right)\right],$$

with $t = \zeta \cdot w \cdot 2^i$. Let β be a constant such that $\mathbb{E}[Z_{i,l}] \leqslant \beta \cdot 2^i$, where β is a constant < 0.5. Choosing ζ such that $c = 2(\beta + \zeta)$, we obtain that

$$\Pr\left[\sum_{k=1}^{w} Z_{i,l,k} > w \cdot 2^{(i-1)} \cdot c\right] = \Pr\left[\sum_{k=1}^{w} Z_{i,l,k} > w \cdot \left(\mathbb{E}[Z_{i,l}] + 2^i \cdot \zeta\right)\right] = \Pr\left[\ \Phi < \mathbb{E}[\Phi] - t\ \right].$$

Applying McDiarmid's inequality we obtain:

$$\Pr\left[\ \Phi < \mathbb{E}[\Phi] - t\right] < \exp\left(-\frac{2t^2}{4w \cdot l}\right).$$

Replacing t by $\zeta \cdot w \cdot 2^i$ gives:

$$\Pr\left[\sum_{k=1}^{w} Z_{i,l,k} > w \cdot 2^{i-1} \cdot c\right] < \exp\left(-w\frac{2^{2i-1}}{l} \cdot \zeta^2\right). \tag{7.4}$$

It remains then to put Equation (7.3) and Equation (7.4) together. Additionally we use that $c = (2B)^{\frac{1}{w}}$ and $c = 2(\beta + \zeta)$ to replace B by $\frac{1}{2}(2(\beta + \zeta))^w$.

$$\Pr\left[\ \mathsf{bias}_{\mathbf{v}}(\mathcal{O}^i_{\mathsf{par}}) > \frac{1}{2}(2(\beta + \zeta))^w\ \right] \leqslant \exp\left(-w\frac{2^{2i-1}}{l} \cdot \zeta^2\right).$$

Hence, by a union bound over all vectors of Hamming weight l,

$$\Pr\left[\ \exists \mathbf{v}, wt\left(\mathbf{v}\right) = l\ ,\ \mathsf{bias}_{\mathbf{v}}(\mathcal{O}^i_{\mathsf{par}}) > \frac{1}{2}(2(\beta + \zeta))^w\ \right] \leqslant \binom{n}{l} \exp\left(-w\frac{2^{2i-1}}{l} \cdot \zeta^2\right).$$

The proof continues by taking the union bound over all the vectors of weight in $[2^{i-1}, 2^i]$:

$$\Pr\left[\ \exists \mathbf{v}, wt\left(\mathbf{v}\right) \in [2^{i-1}, 2^i]\ ,\ \mathsf{bias}_{\mathbf{v}}(\mathcal{O}^i_{\mathsf{par}}) > \frac{1}{2}(2(\beta + \zeta))^w\ \right] \leqslant \sum_{l=2^{i-1}}^{2^i} \binom{n}{l} \exp\left(-w\frac{2^{2i-1}}{l} \cdot \zeta^2\right),$$

and it remains only to perform a single union bound for $i^* \leqslant i \leqslant D$:

$$\Pr\left[\exists i^* \leqslant i \leqslant D, \exists \mathbf{v}, wt(\mathbf{v}) \in [2^{i-1}, 2^i],\ \mathsf{bias}_{\mathbf{v}}(\mathcal{O}^i_{\mathsf{par}}) > \frac{1}{2}(2(\beta+\zeta))^w\right] \leqslant \sum_{i=i^*}^{D} \sum_{l=2^{i-1}}^{2^i} \binom{n}{l} \exp\left(-w\frac{2^{2i-1}}{l} \cdot \zeta^2\right).$$

This bound is significantly tighter than what was achieved with the previous proof. Compared to the original proof by [BCGI+20a], corrected in Appendix C, we gain several orders of magnitude. Why is that the case? In essence, this is because the previous proof relied on Lemma 2.2.1 to bound the bias of the XOR of independent distributions. However, this approach introduced some exponential slackness in the *number* of distributions involved. To overcome this slackness, the strategy was to only "count" the distributions that contribute the most to the bias, by identifying *good* distributions. It was shown that, over the choice of $\mathbf{H}$, a sufficient number of distributions would be good, and the lemma was applied selectively to these good distributions. This approach ensured that the slackness was compensated by the contribution of each distribution. If, instead, one tries to apply the lemmas to all distributions, the resulting bound is too loose and does not provide any usable guarantee.

Here, we manage to directly take into account for the contribution to the bias of *all* distributions by carefully rewriting the bias formula in terms of the $Z_{i,l,k}$ random variables, and by using a standard "optimization trick" to bound the product of the $Z_{i,l,k}$ in terms of their sum. This turns out to be the key to returning to the function Φ, which is Lipschitz, which we can bound without incurring any slackness in the number of distributions involved.

7.3.3 Optimization of the Parameters

The previous analysis introduced the value β as a constant such that $\mathbb{E}[Z_{i,l}] \leqslant \beta \cdot 2^i$. In the correction of the original proof (see Appendix C), β was proven to be $\leqslant 0.44$. In this section, we present better bounds found via computer analysis on β, and simultaneously optimize the different parameters to achieve the smallest value of w for which we can ensure 80-bit security against all test vectors, given a number of samples $N = 2^{30}$.

7.3.3.1 Optimization of β

We start by examining β via computer analysis. Our script is in Appendix D. Table 7.1 shows the value of β obtained for different choices of $n = 2^i$ and l. For larger values of n and a fixed $l = l(n)$, note that our estimate value for β barely increases (for $l < n$) or decreases (for $l \geqslant n$). The value l is taken to be in the interval $[2^{i-1}, 2^i]$. The $Z_{i,l}$ measures the distance to the mean (2^{i-1}) of the number of bins that contain an odd number of balls. For l smaller than 2^{i-1}, this number diverges from the middle value (2^{i-1}) because not enough balls are thrown. Therefore, we are considering only the case where l is lower bounded by 2^{i-1}.

Table 7.1 – Estimated value of β for different values of n and l, in a confidence interval of 99% (rounded value ± 0.002)

	$n = 32$	$n = 64$	$n = 512$	$n = 1024$	$n = 2048$
$l = \frac{n}{2}$	0.178	0.181	0.184	0.184	0.184
$l = \frac{3 \cdot n}{4}$	0.111	0.111	0.111	0.111	0.112
$l = n$	0.084	0.073	0.067	0.067	0.067

7.3.3.2 Fine-tuning ζ and w

Let us revisit our bound. We have shown:

$$\Pr(\exists i^* \leqslant i \leqslant D, \exists \mathbf{v}, wt(\mathbf{v}) \in [2^{i-1}, 2^i] \ , \ \mathsf{bias}_{\mathbf{v}}(\mathcal{O}^i_{\mathsf{par}}) > \frac{1}{2}(2(\beta+\zeta))^w) \leqslant \sum_{i=i^*}^{D} \sum_{l=2^{i-1}}^{2^i} \binom{n}{l} \exp\left(-w\frac{2^{2i-1}}{l} \cdot \zeta^2\right).$$

We want to find a w such that, simultaneously:

$$(2 \cdot (\beta+\zeta))^w/2 \leqslant 2^{-80} \qquad \text{and}$$
$$\sum_{i=i^*}^{D} \sum_{l=2^{i-1}}^{2^i} \binom{n}{l} \exp\left(-w\frac{2^{2i-1}}{l} \cdot \zeta^2\right) \leqslant 2^{-80}. \tag{7.5}$$

Note that

$$\binom{N}{l} \exp\left(-w\frac{2^{2i-1}}{l} \cdot \zeta^2\right) \leqslant \exp\left(\ln(N) \cdot l - w \cdot \frac{2^{2i-1}}{l} \cdot \zeta^2\right).$$

Suppose that $\ln(n) \cdot l < w \cdot \frac{2^{2i-1}}{l} \cdot \zeta^2$. For all $i \in [i^*, D]$, and then for all $l \in [2^{i-1}, 2^i]$ and $l^* \in [2^{i^*-1}, l/2^i \cdot 2^{i^*}]$ that

$$\exp\left(\ln(n) \cdot l - w \cdot \frac{2^{2i-1}}{l} \cdot \zeta^2\right) \leqslant \exp\left(\ln(n) \cdot l^* - w \cdot \frac{2^{2i^*-1}}{l^*} \cdot \zeta^2\right)$$

Let $\mathcal{B}$ be an upper bound on $\exp\left(\ln(N) \cdot l^* - w \cdot \frac{2^{2i^*-1}}{l^*} \cdot \zeta^2\right)$. Then

$$\sum_{i=i^*}^{D} \sum_{l=2^{i-1}}^{2^i} \binom{n}{l} \exp\left(-w\frac{2^{2i-1}}{l} \cdot \zeta^2\right) \leqslant (D - i^* + 1) \cdot 2^{i^*-1} \cdot \mathcal{B}.$$

In practice, we choose w such that $(D - i^* + 1) \cdot 2^{i-1} \cdot \mathcal{B} \leqslant 2^{-80}$. With $i^* = 5$ and $D = 30$, we obtain that $(D - i^* + 1) \cdot 2^{i-1} < 2^9$. Therefore, we want to find w such that

$$(2 \cdot (\beta+\zeta))^w/2 \leqslant 2^{-80} \qquad \text{and}$$
$$\exp\left(\ln(n) \cdot l^* - w \cdot \frac{2^{2i^*-1}}{l^*} \cdot \zeta^2\right) \leqslant 2^{-89}. \tag{7.6}$$

To find a suitable w, we calculate the required w for different values of l^* and take the worst one. In the following, we keep the number of samples $n = 2^D = 2^{30}$. Assume that $l^* = 2^{i^*-1}$. Then we compute w as follows:

- The second inequality can be rewritten as $\exp\left(2^{i^*-1} \cdot \left(\ln(2) \cdot 30 - 2 \cdot w \cdot \zeta^2\right)\right) \leqslant 2^{-80}$. We compute that $w \cdot \zeta^2 \geqslant 12.35$ is sufficient for the inequality to hold.
- Now set $w = \zeta^2/12.35$. The first inequality becomes $(2 \cdot (0.184 + \zeta))^{12.35/\zeta^2} \leqslant 2^{-80}$. The 0.184 constant correspond to the value of β when $l = 2^{i^*-1}$, from Table 7.1. We can solve numerically again, and obtain $\zeta \leqslant 0.219$.
- Conclude with $w = 12.35/\zeta^2 = 12.35/0.219^2 \approx 257$.

The same process can be conducted for $l^* = 2^{i^*}$:

- The second inequality can be rewritten as $\exp\left(2^{i^*} \cdot \left(\ln(2) \cdot 30 - \frac{1}{2}w \cdot \zeta^2\right)\right) \leqslant 2^{-89}$. We compute that $w \cdot \zeta^2 \geqslant 45.5$ is sufficient for the inequality to hold.
- Now set $w = \zeta^2/45.5$. The second inequality becomes $(2 \cdot (0.084 + \zeta))^{45.5/\zeta^2} \leqslant 2^{-80}$. The 0.084 constant correspond to the value of β when $l = 2^{i^*}$, from Table 7.1. We can solve numerically again and obtain $\zeta \leqslant 0.347$.
- Conclude with $w = 45.5/\zeta^2 = 45.5/0.347^2 \approx 380$.

We can perform exactly the same computation for all values of l^* in the interval $[2^{i^*-1}, l/2^i \cdot 2^{i^*}]$. The calculations show that the biggest w is obtained at the extremity of the interval for $l = 2^{i^*}$. We fix our security parameter w to be equal to 380. Going back to our inequality we have shown that

$$\Pr(\exists \mathbf{v}, wt(\mathbf{v}) \geqslant 2^4, \mathsf{bias}_{\mathbf{v}}(\mathcal{O}_{\mathsf{par}}) > 2^{-80}) \leqslant 2^{-80}.$$

Note that this computation is done for a fixed value $D = 30$. As you increase the value D, the parameter w needs to be adjusted accordingly. More concretely, the analysis shows that these two values are linearly connected: there should exist a value c such that $w = c \cdot D$. Unfortunately, the asymptotic analysis gives a very poor value of c. In Table 7.2, we provide the ratio $c = w/D$, to give an intuition of what the right constant should be.

Nevertheless, we can easily derive a lower bound c' that approximates c. We consider the bound $\exp\left(2^{i^*} \cdot \left(\ln(2) \cdot D - \frac{1}{2}w \cdot \zeta^2\right)\right) \leqslant 2^{-89}$. For the left-hand expression to be less than 1, we feed $w \cdot \zeta^2 > 2\ln(2) \cdot D$. Similarly, from the requirement $(2 \cdot (\beta + \zeta))^{2\ln(2) \cdot D/\zeta^2} \leqslant 2^{-80}$, we find another trivial bound on ζ: the left-hand side expression is less than 1 if and only if $\zeta < 1/2 - 0.084 = 0.416$. Therefore, this gives a lower bound on the value of w depending linearly on D only:

$$w > 2\ln(2)D/0.416^2 \sim 8D.$$

D	w	$c = w/D$
20	293	14.7
25	336	13.4
30	380	12.7
35	421	12
40	461	11.5

Table 7.2 – Security parameter w and the ratio w/D, for different values of D, computed with our method above.

7.3.3.3 Concrete Costs in Practice

We use the parameters above ($w = 380, D = 30$) and we compute the seed size and the evaluation time of this PCF constructed using this rVDSD 2.0 assumption. In the construction, we use the FSS primitive from [BGI16], which relies on the GGM tree and a PRG with target security parameter $\lambda = 128$.

The key seed is composed of the share of the short vector of size 541 (from Section 7.3.1), and the short DPF keys corresponding to the shares of the sparse vectors of $\mathbf{e_i}$. The size of a DPF key for a domaine 2^i and security target λ is equal to $i(\lambda + 2) + 2\lambda$ (see Section 7.2.2.3). Thus, the size of the seed is equal to $541 + \sum_{i=5}^{30} w \times (i(\lambda + 2) + 2\lambda) = 25 \cdot 10^6 = 3.12$MB.

Regarding the computational cost, the number of calls to a PRG is $w(i + 1)$ for each bloc i. Therefore, the overall the costs of one evaluation is $w(\sum_{i=5}^{30}(i + 1)) = 1.82 \cdot 10^5$ calls to a PRG. To give a rough runtime estimation, the PRG can be instantiated using two calls to fixed-key AES. We estimate that one byte of AES-128 can be computed in $\sim 1,3$ cycles ([MSY21]), among the 16 bytes required. Therefore, the number of cycles is given by $3.64 \cdot 10^5 \cdot 16 \cdot 1.3 = 7.57 \cdot 10^6$ cycles. Using a 3.8GHz processor, this amounts to roughly 500 PCF evaluations per second on a single core. We estimate that this on-paper estimation would not be far from reality because the computation requires no data access, and therefore cache misses are unlikely. The computation is parallelizable: using c cores increases this number to $500c$ evaluations per second. In the next and last section, we explain some possible aggressive variants that achieve even better results.

7.4 Further Improvements and Future Works

7.4.1 Some Refined Analysis and Possible Small Optimization

We present here how small changes in the analysis can further improve the construction. We note that in [CD23] the size of the random block $\mathbf{R}$ was chosen with too much precaution, and that it could be chosen larger. This observation stems from the following: the part of the key corresponding to the block $\mathbf{R}$ is only 541 bits, whereas the rest of the matrix costs $\sum_{i=5}^{30} w \times (i(\lambda + 2) + 2\lambda) = 25 \cdot 10^6$ bits for $D = 30$. By increasing the size of the block, we could hope to decrease the size of the key, at least initially. It is intuitively better to balance these two different parts. Computation must also be considered. When we increase i^*, it is initially good news because the number of PRG calls for one evaluation depends on i^*. However, we also have to add the costs of the scalar product of two vectors of the size of $\mathbf{R}$, which is linear in the size of $\mathbf{R}$. Additionally, we also apply a straightforward optimization: the number of sub-blocks of each matrix $\mathbf{H}_i$ does not have to be fixed to a single w: in fact, w can be a function of i. Therefore, we compute the different values of w_i, and derive from them the total seed size $|\mathbf{k}|$, and the number T of PCF evaluations per second. The value of w does not change much for larger values of i due to a threshold effect in the exponent (there is a multiplication by 2^i in the exponent, therefore the convergence is really fast).

Table 7.3 displays the values of $w, |\mathbf{k}|, T, |\mathbf{R}|$ we obtain depending on the value i^*. Appendix E provides a script for the computation of these parameters. We choose the value i^* that achieves the best trade-off in this case. We obtain that if we want to optimize the key size, one should take $i^* = 17$, associated with security parameter $w_i = 328$, a key size of 2.03MB, and a number of PCF evaluations per second of 782. We can also maximize the number of PCF evaluations per second by taking a slightly larger seed of 2.04MB, producing 799 PCF evaluations per second.

i^*	$w(i^*)$	$\lvert\mathbf{k}\rvert$	T	$\lvert\mathbf{R}\rvert$
5	374	2.71	576	541
6	347	2.67	584	949
7	337	2.63	593	1732
8	334	2.58	603	3234
9	330	2.53	615	6107
10	329	2.47	629	11598
11	329	2.41	645	22064
12	329	2.34	663	41976
13	329	2.27	684	79747
14	328	2.20	707	151194
15	328	2.13	732	285893
16	328	2.07	757	538907
17	328	2.03	782	1012164
18	328	2.04	799	1893136
19	328	2.14	798	3523992
20	328	2.40	764	6523488
21	328	2.97	681	11997914
22	328	4.08	552	21897081

Table 7.3 – Security parameters $w(i^*)$, key size $|\mathbf{k}|$ (in MB), number of PCF evaluations per second T and length of the matrix $|\mathbf{R}|$ depending on different values of i^*, in the case of $D = 30$

7.4.1.1 Discussion on the Size of the Blocks $\mathbf{H}_i$

In the construction, the size of each $\mathbf{H}_i$ is set to $w_i \cdot 2^i$. By doing this, we are more or less halving the density when moving from $\mathbf{H}_i$ to $\mathbf{H}_{i+1}$. We question whether other sizes could yield better results and propose fixing the size of the $\mathbf{H}_i$ to be equal to $w_i \cdot b^i$. Each $\mathbf{H}_i$ is made of w_i matrices $(\mathbf{M}_{i,k})_{1 \leqslant k \leqslant w_i}$ whose rows are of size b^i (instead of 2^i). We perform an exact generalization of the original security analysis (Section 7.3.2): the goal is to show that the i-th block resists against attack vectors of Hamming weight $l \in [b^{i-1}, b^i]$. The number of blocks constituting the matrix $\mathbf{H}$ also needs to be revisited. Previously, we aimed for D blocks, targeting a number of samples $n = 2^D$. Now, we will target D_{new} blocks such that $b^{D_{\text{new}}} = 2^D = n$. The two inequalities we have to satisfy become

$$(2 \cdot (\beta + \zeta))^w / 2 \leqslant 2^{-80} \qquad \text{and}$$

$$\exp\left(\ln(b) \cdot D_{\text{new}} \cdot l^* - w \cdot \frac{b^{2i^*-1}}{l^*} \cdot \zeta^2\right) \leqslant 2^{-80}(D_{\text{new}} - i^* + 1)b^{i^*}.$$

We perform different analyses to find the b that offers the best results in key size and number of PCF evaluations per second. The target number of samples in the original construction $n = 2^{30}$. It turns out that it is more advantageous to choose a larger b to reduce the number of blocks D_{new} of $\mathbf{H}$, benefiting from the fact that the security parameter w_i of the block i does not grow too fast at the beginning. After some point, the security parameter grows too much, and the trade-off starts to be bad. We decide to set b to be a power of 2, allowing us to reach exactly 2^{30} samples. Therefore, we consider $b \in \{4, 8, 32, 64, 1024\}$, corresponding to the number of blocks $D_{\text{new}} \in \{15, 10, 6, 5, 3\}$. The results associated with these cases are displayed in Table 7.4. We take into account the optimization from

Section 7.4.1, which consists of replacing the first i^* blocks of $\mathbf{H}$ by a tailored random matrix $\mathbf{R}$. Refer to Appendix E for the script computing these parameters.

b = 4	i^*	2	3	4	5	6	7	8	9	10	11
	$w(i^*)$	757	597	563	562	561	561	561	561	561	561
	$\lvert\mathbf{k}\rvert$	1.34	1.3	1.25	1.2	1.14	1.07	1.01	1.01	1.23	2.18
	T	1222	1255	1296	1350	1419	1505	1602	1666	1547	1062
	$\lvert\mathbf{R}\rvert$	218	541	1732	6107	$22 \cdot 10^3$	$80 \cdot 10^3$	$29 \cdot 10^4$	$10 \cdot 10^5$	$3.5 \cdot 10^6$	$1.2 \cdot 10^7$
b = 8	i^*	2	3	4	5	6	7	8	9	10	
	$w(i^*)$	1080	1014	1009	1009	1009	1009	1009	1009	1009	
	$\lvert\mathbf{k}\rvert$	1.18	1.11	1.03	0.94	0.88	1.13	3.28	15.96	73.15	
	T	1435	1509	1617	1766	1924	1754	789	180	40	
	$\lvert\mathbf{R}\rvert$	328	1431	$12 \cdot 10^3$	$80 \cdot 10^3$	$5.4 \cdot 10^5$	$3.5 \cdot 10^6$	$2.2 \cdot 10^7$	$1.2 \cdot 10^8$	$5.8 \cdot 10^8$	
b = 32	i^*	2	3	4	5	6					
	$w(i^*)$	3686	3676	3676	3676	3676					
	$\lvert\mathbf{k}\rvert$	1.78	1.55	1.32	2.39	27.4					
	T	994	1129	1338	965	105					
	$\lvert\mathbf{R}\rvert$	949	$22 \cdot 10^3$	$5.4 \cdot 10^3$	$1.2 \cdot 10^7$	$2.1 \cdot 10^8$					
b = 64	i^*	2	3	4	5						
	$w(i^*)$	7246	7241	7226	7226						
	$\lvert\mathbf{k}\rvert$	2.57	2.11	1.96	16.4						
	T	702	840	980	172						
	$\lvert\mathbf{R}\rvert$	1732	$80 \cdot 10^3$	$3.5 \cdot 10^5$	$1.2 \cdot 10^8$						
b = 1024	i^*	2	3								
	$w(i^*)$	26797	26797								
	$\lvert\mathbf{k}\rvert$	3.89	3.66								
	T	487	593								
	$\lvert\mathbf{R}\rvert$	$22 \cdot 10^3$	$1.2 \cdot 10^7$								

Table 7.4 – Security parameter $w(i^*)$, key size $|\mathbf{k}|$ (in MB) and number of PCF evaluations per second T and width of $\mathbf{R}$ depending on different values of i^* and different values of b.

We obtain that the best result corresponds to $b = 8$, $D_{\text{new}} = 10$, and with the random block $\mathbf{R}$ replacing the blocks $1 \dots i^* - 1 = 5$, of size 538907. This leads to a key size of 0.88MB and 1924 PCF evaluations per second.

Note that for this optimization, we examined what happens when taking the size of one block to be equal to $w_i b^i$, with a fix b. We leave for future work the question asking whether there is any gain from using different b_i.

7.4.2 The All-prefix Variant Optimization

[BCGI+20a] also presented an interesting optimization called the *all-prefix* variant. Here, we present this variant, explain why it is more efficient, and provide some insight for this aggressive variant. In the original construction, D different vectors $\mathbf{e_i}$ are sampled for constructing the error vector, each one being the concatenation of w unit vectors of size 2^i. Instead, let assume we start by sampling only $\mathbf{e_D}$, of size $2^{D \cdot w}$ - the biggest and sparser vector. Then we construct the $\mathbf{e_i}$ recursively as follows:

- Decompose $\mathbf{e_{i+1}} = [\mathbf{e_{(i+1),1}} /\!/ \cdots /\!/ \mathbf{e_{(i+1),w}}]$ into its w different parts, each $\mathbf{e_{i+1,j}}$ being a unit vector of size 2^{i+1} ($/\!/$ denote vertical concatenation).
- For $\mathbf{u} \in \mathbb{F}_q^{2^i}$ a unit vector define

$$\mathcal{F}(\mathbf{u}) \coloneqq \left[\mathbf{u_0} \oplus \mathbf{u_{2^{i-1}}}, \mathbf{u_1} \oplus \mathbf{u_{2^{i-1}+1}}, \dots, \mathbf{u_{2^{i-1}-1}} \oplus \mathbf{u_{2^i-1}}\right] \in \mathbb{F}_q^{2^{i-1}}.$$

We call this operation a *folding*[3] of the vector $\mathbf{u}$. It simply consists to cut your vector $\mathbf{u}$ in two parts, and to xor bit by bit the parts. The resulting vector is therefore twice smaller.

– Define $\mathbf{e_i} = \left[\mathcal{F}(\mathbf{e_{(i+1),1}}) \mathbin{/\!/} \cdots \mathbin{/\!/} \mathcal{F}(\mathbf{e_{(i+1),w}})\right]$.

This produces a valid rVDSD error vector $\mathbf{e}$. Similarly, we can construct the full matrix $\mathbf{H}$ by only sampling $\mathbf{H}_D$, and constructing the rows of $\mathbf{H}_i$ by applying the folding operation on the rows of $\mathbf{H}_{i+1}$. The advantages of this construction are twofold:

– First, it offers significant gains in both seed size and computational time. By doing so, the parties only need a secret sharing of the longest vector $\mathbf{e_D}$ as their seed. This leads to concrete improvements regarding the seed size: assuming the security parameter w is still equal to 380, it suffices to share $541 + 380 \cdot (30 \cdot 130 + 256) = 0.2$MB. This also impacts the main costs of the evaluation, which is the PRG evaluations to create the shares of the error vector. Note that the folding operation commutes with additive sharing (we are doing just XOR after all). Therefore, we just need to produce the shares of $\mathbf{e_D}$, and the shares of the smaller $\mathbf{e_i}$ can be obtained via the folding operations, which comes down to only small linear operations. Therefore, this would lead to several calls to AES equal to $2 \cdot 380 \cdot 31 = 2.3 \cdot 10^4$. Using a 3.8GHz computer, we estimate that this leads to 3950 evaluations per second.

– The folding operation is compatible with the additive sharing: if you have a sharing of a sparse vector you can construct the shares of the folding of the vector. The folding operation commutes with the additive sharing operation.

This optimization could be combined with the previous optimizations (Sections 7.4.1 and 7.4.1.1), but the results are not as good as one might hope. The point of the all-prefix optimization is to take advantage of reducing all the blocks to only one, and therefore the larger the number of blocks the better the reduction. Thus, the two optimizations compete (with the same goal: reducing the number of blocks to effectively share). The gain due to the size of the matrix $\mathbf{R}$ and the choice of the parameter b is, therefore, less convincing for many parameters. Nonetheless, with $b = 4$ and taking $i^* = 7$, we obtain a seed of size 0.17MB and 5106 PCF evaluations per second.

7.4.3 Security of the All-prefix Variant

In this subsection, we discuss the security analysis of the all-prefix variant. By defining the matrix $\mathbf{H}$ via this folding, the analysis of security presented in Section 7.3.2 no longer holds. Indeed, the previous proof (Section 7.3.2) assumes that the different blocks $\mathbf{H}_i$ are independent of one another, which is not the case in the all-prefix variant. Therefore, a new analysis must be conducted. Unfortunately, it has not proven yet to resist linear tests - this is why it is called an *aggressive* variant. Next, we discuss this variant of VDSD and provide some possible leads on how to prove its security.

7.4.3.1 Rewriting the Associated Assumption

For simplicity, we will assume that $b = 2$ and that no random matrix has been added to the construction. We can express the transformation that associate to $\mathbf{e_D}$ the full vector $\mathbf{e}$ by a matrix $\mathbf{X} \in \mathbb{F}_2^{((2^{D+1}-2)\cdot w \times 2^D \cdot w)}$

$$\mathbf{X} = \begin{pmatrix} \mathbf{X}_0 \\ \mathbf{X}_1 \\ \vdots \\ \mathbf{X}_D \end{pmatrix}.$$

3. Not to be mixed up with the folding operation defined in Section 6.2.1.

where

$$\mathbf{X}_i = \begin{pmatrix} \mathbf{J}_i & 0 \dots & 0 \\ 0 & \mathbf{J}_i \dots & 0 \\ \vdots & & \vdots \\ 0 & 0 \dots & \mathbf{J}_i \end{pmatrix} \qquad \text{and} \qquad \mathbf{J}_i = \begin{pmatrix} \mathbf{I}_i & \mathbf{I}_i & \dots & \mathbf{I}_i \end{pmatrix}.$$

The $\mathbf{X}_i$ are matrices of size $2^i \cdot w \times 2^D \cdot w$, the $\mathbf{J}_i$ are matrices of size $2^i \times 2^D$, and the $\mathbf{I}_i$ is the identity matrix of size 2^i. We have therefore:

$$\mathbf{X} \cdot \mathbf{e_D} = \mathbf{e} \qquad \text{and} \qquad \mathbf{H}_D \mathbf{X}^\top = \mathbf{H}.$$

Therefore, the expression of the associated rVDSD assumption in the aggressive variant can be rewritten as follows:

$$\mathbf{H} \cdot \mathbf{e} = \mathbf{H}_D \mathbf{X}^\top \cdot \mathbf{X} \cdot \mathbf{e_D}.$$

We compute

$$\mathbf{X}^\top \cdot \mathbf{X} = \begin{pmatrix} \mathbf{G} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \mathbf{G} \end{pmatrix},$$

where $\mathbf{G} \in \mathbb{F}_2^{2^D \times 2^D}$ is defined by:

$$\mathbf{G} = \sum_1^D \mathbf{G}_i \qquad \text{with} \qquad \mathbf{G}_i = \begin{pmatrix} \mathbf{I}_i & \mathbf{I}_i & \dots & \mathbf{I}_i \\ \mathbf{I}_i & \mathbf{I}_i & \dots & \mathbf{I}_i \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{I}_i & \mathbf{I}_i & \dots & \mathbf{I}_i \end{pmatrix}.$$

Let $\mathbf{H}_\mathcal{C} = \mathbf{H}_D \cdot \mathbf{X}^\top \cdot \mathbf{X}$. The matrix $\mathbf{H}_\mathcal{C}$ has the following shape:

$$\mathbf{H}_\mathcal{C} = \begin{pmatrix} \mathbf{c_{1,1}} & \dots & \mathbf{c_{1,w}} \\ \vdots & \ddots & \vdots \\ \mathbf{c_{l,1}} & \dots & \mathbf{c_{l,w}} \end{pmatrix}.$$

The $\mathbf{c_{i,j}}$ are called *circle* and enjoy special structures and symmetries that we will not discuss in this manuscript. You can see that the matrix $\mathbf{H}_\mathcal{C}$ is divided into w parts, each one independent from the other (as was $\mathbf{H}_D$). Therefore, we can try to analyze the bias of this new problem, with regular sparse noise $\mathbf{e_D}$, and with matrix $\mathbf{H}_\mathcal{C}$.

Let us be more precise about $\mathbf{c_{i,j}}$. Each such vector is of size 2^D. On top of the folding operation $\mathcal{F}$ that we defined earlier, we build the following $\mathcal{F}_{\text{join}}(\mathbf{x}, i)$. Given input $x \in \mathbb{F}_2^{2^D}$, it returns $[\mathcal{F}^i(x)|| \cdots ||\mathcal{F}^i(x)]$, which is a vector composed of 2^i vectors of size 2^{D-i}, produced by the successive application of the folding operation $\mathcal{F}$. Remember that $\mathbf{H}_D$ is formed with w blocks. Let $\mathbf{x_{k,j}}$ be the k-th row of the j-th block. Then $c_{i,j} = \sum_{i=1}^D \mathcal{F}_{\text{join}}(\mathbf{x_{k,j}}, i) \coloneqq \mathcal{F}^{(D)}(\mathbf{x_{k,j}})$.

7.4.3.2 The Assumption

We finish this chapter by giving a taste of the open problem we are facing and some of our reflection to solve it. This is still ongoing research.

Let $\mathcal{D}_{\text{allpref}}$ be the distribution that samples a matrix $\mathbf{H}_D$ and returns the matrix $\mathbf{H}_\mathcal{C}$. The all-prefix variant is therefore conditional on the validity of the following assumption:

Assumption 7.4.1. *Let* $\mathbf{H}_{\mathcal{C}} \xleftarrow{\$} \mathcal{D}_{allpref}$. *Then it holds that:*

$$\{\mathbf{H}_{\mathcal{C}}, \mathbf{H}_{\mathcal{C}} \cdot \mathbf{e_D} : \mathbf{e_D} \xleftarrow{\$} \mathcal{K}_{w,D}\} \approx \{\mathbf{H}_{\mathcal{C}}, y : y \xleftarrow{\$} \mathbb{F}_2^{2^D}\}.$$

In order to give evidence for this assumption, we can try to analyze it using the linear attack framework. This involves studying the bias of the distribution $\{\mathbf{v} \cdot \mathbf{H}_{\mathcal{C}} \cdot \mathbf{e_D} : \mathbf{e_D} \xleftarrow{\$} \mathcal{K}_{w,D}\}$. We can still try to analyze this construction with the balls and bins model: the process is the same as before, counting the number of bins that contain an odd number of balls. However, this time, several balls are thrown into different bins each time, making the analysis more tedious. Different possibilities of proof have been explored (without success) and are still ongoing:

- Analogous analysis as in the original proof, primarily a dead-end due to all the dependencies that appear and their incompatibility with the concentration bounds we use.
- A more "direct" approach, which involve expressing the bias probability with the number of concrete possibilities that achieve that event. This approach aims to see the problem as a combinatorial one. To do so, we analyzed deeply the structure of the vectors $\mathbf{c_{i,j}}$ and how they can be combined. Nevertheless, it has not yet led to something conclusive.
- An analysis of $\mathbf{H}_{\mathcal{C}}$ and, more specifically, trying to obtain its dual distance to apply Proposition 4.3.1. Note that here again, the $\mathbf{c_{i,j}}$ provide some interesting information about the matrix. In fact, it can be shown that $\mathbf{H}_{\mathcal{C}}$ is equal to

$$\mathbf{H}_{\mathcal{C}} = \mathbf{C} \times \begin{bmatrix} \mathbf{P}_1 & \dots & \mathbf{P}_w \end{bmatrix} \qquad \mathbf{C} = \begin{bmatrix} \mathcal{F}^{(D)}(\mathbf{u_0}) \\ \vdots \\ \mathcal{F}^{(D)}(\mathbf{u_{2^D-1}}) \end{bmatrix},$$

  where the $\mathbf{P}_i$ are matrices of permutations and $\mathbf{u_i}$ is a the unit vector of size 2^{2^D} with the non-zero coordinate at index i. This construction possesses some great structures, but no method to exploit them has been found yet, and we leave it for future work.

Conclusion and Open Questions

In this thesis, we investigated MPC protocols, specifically focusing on constructions based on the GGM protocol and the use of correlated pseudorandomness. We analyzed the line of work initiated by Boyle et al. [BCGI18; BCGI+19b; BCGI+20b; BCGI+20a], which targets two promising new tools in cryptography: Pseudorandom Correlation Generators (PCGs) and Pseudorandom Correlation Functions (PCFs). This thesis was articulated around three different axes:

1. We presented a construction of PCGs for OLE correlation, building on the work of [BCGI+20b]. They presented a PCG for OLE, whose security relies on the Ring Syndrome Decoding assumption ($\mathcal{R}$-SD), in the case $\mathcal{R} = \mathbb{F}_q[X]/(P(X))$, for P a polynomial that splits into linear distinct factors. This construction suffered from a problematic constraint: the OLEs are produced on a field $\mathbb{F}_q$, and q must satisfy $q > n$, where n is the number of OLEs created. We reduced the constraint to $q > 2$. This was achieved by choosing a different underlying structure $\mathcal{R} = \mathbb{F}_q[\mathbb{G}]$. We introduced and thoroughly studied the associated security assumption, namely the QA-SD assumption.

   Furthermore, we presented some techniques to produce PCGs for OLE over $\mathbb{F}_2$, removing the constraint altogether. Our construction, FOLEAGE, offers the best performance for the massive production of OLEs (producing over 12 million triples per second in the 2-party setting on one core of a commodity machine).

   Nevertheless, these results are not entirely satisfactory. Indeed, as we have shown in Section 5.4, the case $q = 2$ cannot be achieved natively from our construction; this is one of its major limitations. The solution we proposed to achieve $q = 2$ in FOLEAGE was more via an indirect route, which entailed a non-negligible communication overhead. This raises the natural question:

   Question 7.4.1. *Does there exist some ring $\mathcal{R}$ in which we can operate our protocol, creating* OLE*s over $\mathbb{F}_2$?*

   Different structures need to be investigated. We mentioned in Section 5.4.3 that choosing the set of Boolean functions

   $$\mathcal{R} = \mathbb{F}_2[X_1, \ldots, X_n]/(X_1^2 - X_1, \ldots, X_n^2 - X_n),$$

   paired with a variable density error vector, could be an interesting lead. This is something we will definitely be investigating in the future.

2. Additionally, we also studied a Pseudorandom Correlation Function (PCF) construction from [BCGI+20a]. The construction's security relies on a particular variant of syndrome decoding

in the case of a parity-check matrix and an error vector having a variable density structure. We explained the motivation and the description of the variant, the VDSD assumption. We proposed some improvements in the analysis to make the construction applicable in practice (the initial construction was more a proof-of-concept, and was by no means efficient), alongside correcting the initial proof of security. While our construction is not state-of-the-art compared to other PCFs, it still offers interesting aspects, such as the change of noise in the general assumption. This is typically something well worth exploring. As explained in Section 7.4, we have already pursued several improvements for this construction. It appears that the bounds we use may still be quite loose, and therefore some improvements in this area might already lead to interesting progress.

Mainly, we hope to find a way to prove the assumption at the end of Section 7.4:

Question 7.4.2. *Is the all-prefix variant from Section 7.4.2 secure?*

This has been a problem we studied during the thesis without conclusive results yet, but the question is worth pursuing.

3. Finally, the thesis also revolved around the study of different variants of the syndrome decoding assumption and their analysis. To help analyze possible attacks against them, or to prove the security of the assumptions we used, we pursued the linear attack framework. This framework, introduced by [BCGI+20a], says that for all linear attacks, we can associate an attack vector which depends only on $\mathbf{H}$, such that $\mathbf{v}^\intercal \cdot \mathbf{H} \cdot \mathbf{e}$ is biased. In order to be secure, the linear test framework does not beat around the bush: it requires that the bias associated with $\mathbf{v}$, for all possible attack vectors $\mathbf{v}$, is negligible. We showed that being secure in the linear test framework signifies that even the best attack vector will not provide a significant bias. We provided an analysis of the literature and explained how some classical algorithm fit within the framework. We can identify two different points to analyze further:

   1) It is always assumed that the adversary can find the best attack vector, and we can question this assumption, and 2) in order to recover the secret, an adversary would require multiple attack vectors, but the quality of each of them might be questionable.

   Both points are good questions, and we currently believe that the framework is overly conservative and that we could improve it by targeting its precise limitations.

Question 7.4.3. *Can we improve the linear test framework to make it less conservative and closer to what real attacks do in practice?*

Bibliography

[AABB+22a] Carlos Aguilar Melchor et al. *BIKE*. Round 4 Submission to the NIST Post-Quantum Cryptography Call, v. 5.1. Version 5.1. Oct. 2022. URL: https://bikesuite.org.

[AABB+22b] Carlos Aguilar Melchor et al. *HQC*. Round 4 Submission to the NIST Post-Quantum Cryptography Call. https://pqc-hqc.org/. Oct. 2022.

[ABCC+22] Martin Albrecht et al. *Classic McEliece (merger of Classic McEliece and NTS-KEM)*. https://classic.mceliece.org. Fourth round finalist of the NIST post-quantum cryptography call. Nov. 2022.

[ABGK+14] Adi Akavia, Andrej Bogdanov, Siyao Guo, Akshay Kamath, and Alon Rosen. "Candidate weak pseudorandom functions in $\mathsf{AC}^0 \circ \mathrm{MOD}_2$". In: *ITCS 2014*. Ed. by Moni Naor. ACM, Jan. 2014, pp. 251–260. DOI: 10.1145/2554797.2554821.

[ADIN+17] Benny Applebaum, Ivan Damgård, Yuval Ishai, Michael Nielsen, and Lior Zichron. "Secure Arithmetic Computation with Constant Computational Overhead". In: *CRYPTO 2017, Part I*. Ed. by Jonathan Katz and Hovav Shacham. Vol. 10401. LNCS. Springer, Heidelberg, Aug. 2017, pp. 223–254. DOI: 10.1007/978-3-319-63688-7_8.

[AFGG+23] Carlos Aguilar Melchor et al. *SDitH*. Round 1 Additional Signatures to the NIST Post-Quantum Cryptography: Digital Signature Schemes Call. May 2023. URL: https://sdith.org.

[AG11] Sanjeev Arora and Rong Ge. "New Algorithms for Learning in Presence of Errors". English. In: *Automata, Languages and Programming*. Ed. by Luca Aceto, Monika Henzinger, and Jiří Sgall. Vol. 6755. LNCS. Springer Berlin Heidelberg, 2011, pp. 403–415. ISBN: 978-3-642-22005-0. URL: http://dx.doi.org/10.1007/978-3-642-22006-7%5C_34.

[Al 01] A. Kh. Al Jabri. "A Statistical Decoding Algorithm for General Linear Block Codes". In: *8th IMA International Conference on Cryptography and Coding*. Ed. by Bahram Honary. Vol. 2260. LNCS. Springer, Heidelberg, Dec. 2001, pp. 1–8.

[Ale03] Michael Alekhnovich. "More on Average Case vs Approximation Complexity". In: *44th FOCS*. IEEE Computer Society Press, Oct. 2003, pp. 298–307. DOI: 10.1109/SFCS.2003.1238204.

[Azu67] Kazuoki Azuma. "Weighted sums of certain dependent random variables". In: *Tohoku Mathematical Journal, Second Series* 19.3 (1967), pp. 357–367.

[BBCC+24] Maxime Bombar, Dung Bui, Geoffroy Couteau, Alain Couvreur, Clément Ducros, and Sacha Servan-Schreiber. *FOLEAGE: $\mathbb{F}_4$OLE-Based Multi-Party Computation for Boolean Circuits*. Cryptology ePrint Archive, Paper 2024/429. https://eprint.iacr.org/2024/429. 2024. URL: https://eprint.iacr.org/2024/429.

[BCCD23] Maxime Bombar, Geoffroy Couteau, Alain Couvreur, and Clément Ducros. "Correlated Pseudorandomness from the Hardness of Quasi-Abelian Decoding". In: *CRYPTO 2023, Part IV*. LNCS. Springer, Heidelberg, Aug. 2023, pp. 567–601. DOI: 10.1007/978-3-031-38551-3_18.

[BCGI+17] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, and Michele Orrù. "Homomorphic Secret Sharing: Optimizations and Applications". In: *ACM CCS 2017*. Ed. by Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu. ACM Press, Oct. 2017, pp. 2105–2122. DOI: 10.1145/3133956.3134107.

[BCGI+19a] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. "Efficient Two-Round OT Extension and Silent Non-Interactive Secure Computation". In: *ACM CCS 2019*. Ed. by Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz. ACM Press, Nov. 2019, pp. 291–308. DOI: 10.1145/3319535.3354255.

[BCGI+19b] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. "Efficient Pseudorandom Correlation Generators: Silent OT Extension and More". In: *CRYPTO 2019, Part III*. Ed. by Alexandra Boldyreva and Daniele Micciancio. Vol. 11694. LNCS. Springer, Heidelberg, Aug. 2019, pp. 489–518. DOI: 10.1007/978-3-030-26954-8_16.

[BCGI+20a] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. "Correlated Pseudorandom Functions from Variable-Density LPN". In: *61st FOCS*. IEEE Computer Society Press, Nov. 2020, pp. 1069–1080. DOI: 10.1109/FOCS46700.2020.00103.

[BCGI+20b] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. "Efficient Pseudorandom Correlation Generators from Ring-LPN". In: *CRYPTO 2020, Part II*. Ed. by Daniele Micciancio and Thomas Ristenpart. Vol. 12171. LNCS. Springer, Heidelberg, Aug. 2020, pp. 387–416. DOI: 10.1007/978-3-030-56880-1_14.

[BCGI+22] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Nicolas Resch, and Peter Scholl. "Correlated Pseudorandomness from Expand-Accumulate Codes". In: *CRYPTO 2022, Part II*. Ed. by Yevgeniy Dodis and Thomas Shrimpton. Vol. 13508. LNCS. Springer, Heidelberg, Aug. 2022, pp. 603–633. DOI: 10.1007/978-3-031-15979-4_21.

[BCGI18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. "Compressing Vector OLE". In: *ACM CCS 2018*. Ed. by David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang. ACM Press, Oct. 2018, pp. 896–912. DOI: 10.1145/3243734.3243868.

[BCMP+24] Dung Bui, Geoffroy Couteau, Pierre Meyer, Alain Passelègue, and Mahshid Riahinia. *Fast Public-Key Silent OT and More from Constrained Naor-Reingold*. Cryptology ePrint Archive, Paper 2024/178. https://eprint.iacr.org/2024/178. 2024. URL: https://eprint.iacr.org/2024/178.

[Bea91] Donald Beaver. "Secure Multiparty Protocols and Zero-Knowledge Proof Systems Tolerating a Faulty Minority". In: *Journal of Cryptology* 4.2 (Jan. 1991), pp. 75–122. DOI: 10.1007/BF00196771.

[Bea92] Donald Beaver. "Efficient Multiparty Protocols Using Circuit Randomization". In: *CRYPTO'91*. Ed. by Joan Feigenbaum. Vol. 576. LNCS. Springer, Heidelberg, Aug. 1992, pp. 420–432. DOI: 10.1007/3-540-46766-1_34.

[Bea95] Donald Beaver. "Precomputing Oblivious Transfer". In: *CRYPTO'95*. Ed. by Don Coppersmith. Vol. 963. LNCS. Springer, Heidelberg, Aug. 1995, pp. 97–109. DOI: 10.1007/3-540-44750-4_8.

[BF02] Alexander Barg and G. David Forney. "Random codes: Minimum distances and error exponents". In: *IEEE Trans. Inf. Theory* 48.9 (2002), pp. 2568–2573. DOI: 10.1109/TIT.2002.800480. URL: https://doi.org/10.1109/TIT.2002.800480.

[BFKL94] Avrim Blum, Merrick L. Furst, Michael J. Kearns, and Richard J. Lipton. "Cryptographic Primitives Based on Hard Learning Problems". In: *CRYPTO'93*. Ed. by Douglas R. Stinson. Vol. 773. LNCS. Springer, Heidelberg, Aug. 1994, pp. 278–291. DOI: 10.1007/3-540-48329-2_24.

[BGI15] Elette Boyle, Niv Gilboa, and Yuval Ishai. "Function Secret Sharing". In: *EUROCRYPT 2015, Part II*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9057. LNCS. Springer, Heidelberg, Apr. 2015, pp. 337–367. DOI: 10.1007/978-3-662-46803-6_12.

[BGI16] Elette Boyle, Niv Gilboa, and Yuval Ishai. "Function Secret Sharing: Improvements and Extensions". In: *ACM CCS 2016*. Ed. by Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi. ACM Press, Oct. 2016, pp. 1292–1303. DOI: 10.1145/2976749.2978429.

[BJMM12] Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. "Decoding Random Binary Linear Codes in $2^{n/20}$: How 1 + 1 = 0 Improves Information Set Decoding". In: *EUROCRYPT 2012*. Ed. by David Pointcheval and Thomas Johansson. Vol. 7237. LNCS. Springer, Heidelberg, Apr. 2012, pp. 520–536. DOI: 10.1007/978-3-642-29011-4_31.

[BKS19] Elette Boyle, Lisa Kohl, and Peter Scholl. *Homomorphic Secret Sharing from Lattices Without FHE*. Cryptology ePrint Archive, Report 2019/129. https://eprint.iacr.org/2019/129. 2019.

[BKW00] Avrim Blum, Adam Kalai, and Hal Wasserman. *Noise-Tolerant Learning, the Parity Problem, and the Statistical Query Model*. 2000. arXiv: cs/0010022 [cs.LG].

[BKW03] Avrim Blum, Adam Kalai, and Hal Wasserman. "Noise-tolerant learning, the parity problem, and the statistical query model". In: *Journal of the ACM (JACM)* 50.4 (2003), pp. 506–519.

[BL12] Daniel J. Bernstein and Tanja Lange. *Never trust a bunny*. Cryptology ePrint Archive, Report 2012/355. https://eprint.iacr.org/2012/355. 2012.

[BLP11] Daniel J. Bernstein, Tanja Lange, and Christiane Peters. "Smaller Decoding Exponents: Ball-Collision Decoding". In: *CRYPTO 2011*. Ed. by Phillip Rogaway. Vol. 6841. LNCS. Springer, Heidelberg, Aug. 2011, pp. 743–760. DOI: 10.1007/978-3-642-22792-9_42.

[BM18] Leif Both and Alexander May. "Decoding Linear Codes with High Error Rate and Its Impact for LPN Security". In: *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018*. Ed. by Tanja Lange and Rainer Steinwandt. Springer, Heidelberg, 2018, pp. 25–46. DOI: 10.1007/978-3-319-79063-3_2.

[BM82] Manuel Blum and Silvio Micali. "How to generate cryptographically strong sequences of pseudo random bits". In: *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*. 1982, pp. 112–117. DOI: 10.1109/SFCS.1982.72.

[BM90] Mihir Bellare and Silvio Micali. "Non-Interactive Oblivious Transfer and Applications". In: *CRYPTO'89*. Ed. by Gilles Brassard. Vol. 435. LNCS. Springer, Heidelberg, Aug. 1990, pp. 547–557. DOI: 10.1007/0-387-34805-0_48.

[BM97] Mihir Bellare and Daniele Micciancio. *A New Paradigm for Collision-free Hashing: Incrementality at Reduced Cost*. Cryptology ePrint Archive, Report 1997/001. https://eprint.iacr.org/1997/001. 1997.

[BMT78] E. Berlekamp, R. McEliece, and H. van Tilborg. "On the inherent intractability of certain coding problems (Corresp.)" In: *IEEE Transactions on Information Theory* 24.3 (1978), pp. 384–386. DOI: 10.1109/TIT.1978.1055873.

[BR17] Andrej Bogdanov and Alon Rosen. *Pseudorandom Functions: Three Decades Later*. Cryptology ePrint Archive, Report 2017/652. https://eprint.iacr.org/2017/652. 2017.

[BTV15] Sonia Bogos, Florian Tramer, and Serge Vaudenay. *On Solving LPN using BKW and Variants*. Cryptology ePrint Archive, Report 2015/049. https://eprint.iacr.org/2015/049. 2015.

[BV16] Sonia Bogos and Serge Vaudenay. *Optimization of LPN Solving Algorithms*. Cryptology ePrint Archive, Report 2016/288. https://eprint.iacr.org/2016/288. 2016.

[Can01] Ran Canetti. "Universally Composable Security: A New Paradigm for Cryptographic Protocols". In: *42nd FOCS*. IEEE Computer Society Press, Oct. 2001, pp. 136–145. DOI: 10.1109/SFCS.2001.959888.

[CD23] Geoffroy Couteau and Clément Ducros. "Pseudorandom Correlation Functions from Variable-Density LPN, Revisited". In: *PKC 2023, Part II*. Ed. by Alexandra Boldyreva and Vladimir Kolesnikov. Vol. 13941. LNCS. Springer, Heidelberg, May 2023, pp. 221–250. DOI: 10.1007/978-3-031-31371-4_8.

[CDMT22a] Kevin Carrier, Thomas Debris-Alazard, Charles Meyer-Hilfiger, and Jean-Pierre Tillich. "Statistical Decoding 2.0: Reducing Decoding to LPN". In: *Advances in Cryptology - ASIACRYPT 2022*. LNCS. Springer, 2022. URL: https://eprint.iacr.org/2022/1000.

[CDMT22b] Kevin Carrier, Thomas Debris-Alazard, Charles Meyer-Hilfiger, and Jean-Pierre Tillich. "Statistical Decoding 2.0: Reducing Decoding to LPN". In: *ASIACRYPT 2022, Part IV*. Ed. by Shweta Agrawal and Dongdai Lin. Vol. 13794. LNCS. Springer, Heidelberg, Dec. 2022, pp. 477–507. DOI: 10.1007/978-3-031-22972-5_17.

[CDMT24] Kevin Carrier, Thomas Debris-Alazard, Charles Meyer-Hilfiger, and Jean-Pierre Tillich. "Reduction From Sparse LPN to LPN, Dual Attack 3.0". In: *Advances in Cryptology - EUROCRYPT 2024*. LNCS. Springer, 2024. URL: https://eprint.iacr.org/2023/1852.

[Cou19] Geoffroy Couteau. "A Note on the Communication Complexity of Multiparty Computation in the Correlated Randomness Model". In: *EUROCRYPT 2019, Part II*. Ed. by Yuval Ishai and Vincent Rijmen. Vol. 11477. LNCS. Springer, Heidelberg, May 2019, pp. 473–503. DOI: 10.1007/978-3-030-17656-3_17.

[Cou23] Geoffroy Couteau. "Correlated Pseudorandomness in Secure Computation, Habilitation à Diriger des Recherches". In: (2023). URL: https://geoffroycouteau.github.io/assets/pdf/hdr.pdf.

[CRR21] Geoffroy Couteau, Peter Rindal, and Srinivasan Raghuraman. "Silver: Silent VOLE and Oblivious Transfer from Hardness of Decoding Structured LDPC Codes". In: *CRYPTO 2021, Part III*. Ed. by Tal Malkin and Chris Peikert. Vol. 12827. LNCS. Virtual Event: Springer, Heidelberg, Aug. 2021, pp. 502–534. DOI: 10.1007/978-3-030-84252-9_17.

[CT19] Rodolfo Canto-Torres and Jean-Pierre Tillich. "Speeding up decoding a code with a non-trivial automorphism group up to an exponential factor". In: *Proc. IEEE Int. Symposium Inf. Theory - ISIT 2019*. 2019, pp. 1927–1931.

[CT65] James W. Cooley and John W. Tukey. "An Algorithm for the Machine Calculation of Complex Fourier Series". In: *Math. Comput.* 19 (1965), pp. 297–301. DOI: 10.1090/S0025-5718-1965-0178586-1.

[Deb23a] Thomas Debris-Alazard. *Code-based Cryptography: Lecture Notes*. https://arxiv.org/abs/2304.03541. 2023. DOI: 10.48550/arXiv.2304.03541. arXiv: 2304.03541.

[Deb23b] Thomas Debris-Alazard. "Code-based cryptography: Lecture Notes. arXiv:2304.03541," in: (2023).

[DHRW16] Yevgeniy Dodis, Shai Halevi, Ron D. Rothblum, and Daniel Wichs. "Spooky Encryption and Its Applications". In: *CRYPTO 2016, Part III*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9816. LNCS. Springer, Heidelberg, Aug. 2016, pp. 93–122. DOI: 10.1007/978-3-662-53015-3_4.

[DNNR17] Ivan Damgård, Jesper Buus Nielsen, Michael Nielsen, and Samuel Ranellucci. "The TinyTable Protocol for 2-Party Secure Computation, or: Gate-Scrambling Revisited". In: *CRYPTO 2017, Part I*. Ed. by Jonathan Katz and Hovav Shacham. Vol. 10401. LNCS. Springer, Heidelberg, Aug. 2017, pp. 167–187. DOI: 10.1007/978-3-319-63688-7_6.

[DP12] Ivan Damgård and Sunoo Park. *How Practical is Public-Key Encryption Based on LPN and Ring-LPN?* Cryptology ePrint Archive, Report 2012/699. https://eprint.iacr.org/2012/699. 2012.

[Ds17] Jack Doerner and abhi shelat. "Scaling ORAM for Secure Computation". In: *ACM CCS 2017*. Ed. by Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu. ACM Press, Oct. 2017, pp. 523–535. DOI: 10.1145/3133956.3133967.

[DT17a] Thomas Debris-Alazard and Jean-Pierre Tillich. *Statistical decoding*. Slides of the ISIT talk. See https://tdalazard.io/slidesDecoStat.pdf. June 2017. URL: https://tdalazard.io/slidesDecoStat.pdf.

[DT17b] Thomas Debris-Alazard and Jean-Pierre Tillich. "Statistical decoding". In: *Proc. IEEE Int. Symposium Inf. Theory - ISIT 2017*. Aachen, Germany, June 2017, pp. 1798–1802.

[Dum89] Il'ya Dumer. "Two decoding algorithms for linear codes". In: *Probl. Inf. Transm.* 25.1 (1989), pp. 17–23.

[EKM17] Andre Esser, Robert Kübler, and Alexander May. "LPN Decoded". In: *CRYPTO 2017, Part II*. Ed. by Jonathan Katz and Hovav Shacham. Vol. 10402. LNCS. Springer, Heidelberg, Aug. 2017, pp. 486–514. DOI: 10.1007/978-3-319-63715-0_17.

[Fei02] Uriel Feige. "Relations between average case complexity and approximation complexity". In: *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*. STOC '02. Montreal, Quebec, Canada: Association for Computing Machinery, 2002, pp. 534–543. ISBN: 1581134959. DOI: 10.1145/509907.509985. URL: https://doi.org/10.1145/509907.509985.

[FGKP09] Vitaly Feldman, Parikshit Gopalan, Subhash Khot, and Ashok Kumar Ponnuswami. "On Agnostic Learning of Parities, Monomials, and Halfspaces". In: *SIAM Journal on Computing* 39.2 (2009), pp. 606–645. DOI: 10.1137/070684914. eprint: https://doi.org/10.1137/070684914. URL: https://doi.org/10.1137/070684914.

[FJR22] Thibauld Feneuil, Antoine Joux, and Matthieu Rivain. "Syndrome Decoding in the Head: Shorter Signatures from Zero-Knowledge Proofs". In: *CRYPTO 2022, Part II*. Ed. by Yevgeniy Dodis and Thomas Shrimpton. Vol. 13508. LNCS. Springer, Heidelberg, Aug. 2022, pp. 541–572. DOI: 10.1007/978-3-031-15979-4_19.

[FKI07] Marc Fossorier, Kazukuni Kobara, and Hideki Imai. "Modeling Bit Flipping Decoding Based on Nonorthogonal Check Sums With Application to Iterative Decoding Attack of McEliece Cryptosystem". In: *Information Theory, IEEE Transactions on* 53 (Feb. 2007), pp. 402–411. DOI: 10.1109/TIT.2006.887515.

[FL15] Yun Fan and Liren Lin. "Thresholds of Random Quasi-Abelian Codes". In: *IEEE Transactions on Information Theory* 61.1 (2015), pp. 82–90. DOI: 10.1109/TIT.2014.2368138.

[FS09] Matthieu Finiasz and Nicolas Sendrier. "Security Bounds for the Design of Code-Based Cryptosystems". In: *ASIACRYPT 2009*. Ed. by Mitsuru Matsui. Vol. 5912. LNCS. Springer, Heidelberg, Dec. 2009, pp. 88–105. DOI: 10.1007/978-3-642-10366-7_6.

[FS96] Jean-Bernard Fischer and Jacques Stern. "An Efficient Pseudo-Random Generator Provably as Secure as Syndrome Decoding". In: *EUROCRYPT'96*. Ed. by Ueli M. Maurer. Vol. 1070. LNCS. Springer, Heidelberg, May 1996, pp. 245–255. DOI: 10.1007/3-540-68339-9_22.

[Gen09] Craig Gentry. "Fully homomorphic encryption using ideal lattices". In: *41st ACM STOC*. Ed. by Michael Mitzenmacher. ACM Press, May 2009, pp. 169–178. DOI: 10.1145/1536414.1536440.

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. "How to Construct Random Functions (Extended Abstract)". In: *25th FOCS*. IEEE Computer Society Press, Oct. 1984, pp. 464–479. DOI: 10.1109/SFCS.1984.715949.

[GI14] Niv Gilboa and Yuval Ishai. "Distributed Point Functions and Their Applications". In: *EUROCRYPT 2014*. Ed. by Phong Q. Nguyen and Elisabeth Oswald. Vol. 8441. LNCS. Springer, Heidelberg, May 2014, pp. 640–658. DOI: 10.1007/978-3-642-55220-5_35.

[Gil99] Niv Gilboa. "Two Party RSA Key Generation". In: *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*. Vol. 1666. Lecture Notes in Computer Science. Springer, 1999, pp. 116–129. DOI: 10.1007/3-540-48405-1_8.

[GJL15] Qian Guo, Thomas Johansson, and Carl Löndahl. "A New Algorithm for Solving Ring-LPN With a Reducible Polynomial". In: *IEEE Transactions on Information Theory* 61.11 (2015), pp. 6204–6212. DOI: 10.1109/TIT.2015.2475738.

[GJL20] Qian Guo, Thomas Johansson, and Carl Löndahl. "Solving LPN Using Covering Codes". In: *Journal of Cryptology* 33.1 (Jan. 2020), pp. 1–33. DOI: 10.1007/s00145-019-09338-8.

[GJN23] Qian Guo, Thomas Johansson, and Vu Nguyen. *A New Sieving-Style Information-Set Decoding Algorithm*. Cryptology ePrint Archive, Paper 2023/247. https://eprint.iacr.org/2023/247. 2023. URL: https://eprint.iacr.org/2023/247.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. "How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority". In: *19th ACM STOC*. Ed. by Alfred Aho. ACM Press, May 1987, pp. 218–229. DOI: 10.1145/28395.28420.

[Gol09a] Oded Goldreich. "Foundations of cryptography: volume 2, basic applications". In: *Cambridge university press* (2009).

[Gol09b] Oded Goldreich. *Foundations of cryptography: volume 2, basic applications*. Cambridge university press, 2009.

[GZ06] Philippe Gaborit and Gilles Zémor. "Asymptotic improvement of the Gilbert-Varshamov bound for linear codes". In: *Proc. IEEE Int. Symposium Inf. Theory - ISIT 2006*. Seattle, USA, June 2006, pp. 287–291.

[HILL99] Johan HÅstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. "A Pseudorandom Generator from any One-way Function". In: *SIAM Journal on Computing* 28.4 (1999), pp. 1364–1396. DOI: 10.1137/S0097539793244708. eprint: https://doi.org/10.1137/S0097539793244708. URL: https://doi.org/10.1137/S0097539793244708.

[Jab01] Abdulrahman Al Jabri. "A statistical decoding algorithm for general linear block codes". In: *Cryptography and coding. Proceedings of the 8th IMA International Conference*. Ed. by Bahram Honary. Vol. 2260. LNCS. Cirencester, UK: Springer, Dec. 2001, pp. 1–8.

[Kas74] T. Kasami. "A Gilbert-Varshamov bound for quasi-cycle codes of rate 1/2 (Corresp.)" In: *IEEE Transactions on Information Theory* 20.5 (1974), pp. 679–679. DOI: 10.1109/TIT.1974.1055262.

[Kir11] Paul Kirchner. *Improved Generalized Birthday Attack*. Cryptology ePrint Archive, Report 2011/377. https://eprint.iacr.org/2011/377. 2011.

[KMPS94] Anil Kamath, Rajeev Motwani, Krishna V. Palem, and Paul G. Spirakis. "Tail Bounds for Occupancy and the Satisfiability Threshold Conjecture". In: *35th FOCS*. IEEE Computer Society Press, Nov. 1994, pp. 592–603. DOI: 10.1109/SFCS.1994.365732.

[KPR18] Marcel Keller, Valerio Pastro, and Dragos Rotaru. "Overdrive: Making SPDZ Great Again". In: *EUROCRYPT 2018, Part III*. Ed. by Jesper Buus Nielsen and Vincent Rijmen. Vol. 10822. LNCS. Springer, Heidelberg, Apr. 2018, pp. 158–189. DOI: 10.1007/978-3-319-78372-7_6.

[LB88] Pil Joong Lee and Ernest F. Brickell. "An Observation on the Security of McEliece's Public-Key Cryptosystem". In: *EUROCRYPT'88*. Ed. by C. G. Günther. Vol. 330. LNCS. Springer, Heidelberg, May 1988, pp. 275–280. DOI: 10.1007/3-540-45961-8_25.

[LF06] Éric Levieil and Pierre-Alain Fouque. "An Improved LPN Algorithm". In: *Security and Cryptography for Networks*. Ed. by Roberto De Prisco and Moti Yung. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 348–359. ISBN: 978-3-540-38081-8.

[LP15] Helger Lipmaa and Kateryna Pavlyk. "Analysis and Implementation of an Efficient Ring-LPN Based Commitment Scheme". In: *CANS 15*. Ed. by Michael Reiter and David Naccache. LNCS. Springer, Heidelberg, Dec. 2015, pp. 160–175. DOI: 10.1007/978-3-319-26823-1_12.

[Lyu05] Vadim Lyubashevsky. "The Parity Problem in the Presence of Noise, Decoding Random Linear Codes, and the Subset Sum Problem". In: *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*. Ed. by Chandra Chekuri, Klaus Jansen, José D. P. Rolim, and Luca Trevisan. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 378–389. ISBN: 978-3-540-31874-3.

[McD89] Colin McDiarmid. *On the method of bounded differences, in "Survey in Combinatorics,"(J. Simons, Ed.) London Mathematical Society Lecture Notes, Vol. 141*. 1989.

[McE78] Robert J. McEliece. *A public-key cryptosystem based on algebraic coding theory*. The Deep Space Network Progress Report 42-44. https://ipnpr.jpl.nasa.gov/progress_report2/42-44/44N.PDF. Jet Propulsion Laboratory, California Institute of Technology, Jan. 1978, pp. 114–116.

[MMT11] Alexander May, Alexander Meurer, and Enrico Thomae. "Decoding Random Linear Codes in $\tilde{\mathcal{O}}(2^{0.054n})$". In: *ASIACRYPT 2011*. Ed. by Dong Hoon Lee and Xiaoyun Wang. Vol. 7073. LNCS. Springer, Heidelberg, Dec. 2011, pp. 107–124. DOI: 10.1007/978-3-642-25385-0_6.

[MO15a] Alexander May and Ilya Ozerov. "On Computing Nearest Neighbors with Applications to Decoding of Binary Linear Codes". In: *EUROCRYPT 2015, Part I*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. LNCS. Springer, Heidelberg, Apr. 2015, pp. 203–228. DOI: 10.1007/978-3-662-46800-5_9.

[MO15b] Alexander May and Ilya Ozerov. "On Computing Nearest Neighbors with Applications to Decoding of Binary Linear Codes". In: *Advances in Cryptology - EUROCRYPT 2015*. Ed. by E. Oswald and M. Fischlin. Vol. 9056. LNCS. Springer, 2015, pp. 203–228.

[MR92] Silvio Micali and Phillip Rogaway. "Secure Computation (Abstract)". In: *CRYPTO'91*. Ed. by Joan Feigenbaum. Vol. 576. LNCS. Springer, Heidelberg, Aug. 1992, pp. 392–404. DOI: 10.1007/3-540-46766-1_32.

[MS86] Florence J. MacWilliams and Neil J. A. Sloane. *The Theory of Error-Correcting Codes*. Fifth. Amsterdam: North–Holland, 1986.

[MSY21] Jean-Pierre Münch, Thomas Schneider, and Hossein Yalame. "VASA: Vector AES Instructions for Security Applications". In: *Annual Computer Security Applications Conference*. 2021, pp. 131–145.

[MT23] Charles Meyer-Hilfiger and Jean-Pierre Tillich. "Rigorous Foundations for Dual Attacks in Coding Theory". In: *Theory of Cryptography Conference, TCC 2023*. LNCS. to appear. Springer Verlag, Dec. 2023. URL: https://eprint.iacr.org/2023/1460.

[OSY21] Claudio Orlandi, Peter Scholl, and Sophia Yakoubov. "The Rise of Paillier: Homomorphic Secret Sharing and Public-Key Silent OT". In: *EUROCRYPT 2021, Part I*. Ed. by Anne Canteaut and François-Xavier Standaert. Vol. 12696. LNCS. Springer, Heidelberg, Oct. 2021, pp. 678–708. DOI: 10.1007/978-3-030-77870-5_24.

[Ove06a] Raphael Overbeck. "Statistical Decoding Revisited". In: *ACISP 06*. Ed. by Lynn Margaret Batten and Reihaneh Safavi-Naini. Vol. 4058. LNCS. Springer, Heidelberg, July 2006, pp. 283–294.

[Ove06b] Raphael Overbeck. "Statistical decoding revisited". In: *Information security and privacy : 11th Australasian conference, ACISP 2006*. Ed. by Reihaneh Safavi-Naini Lynn Batten. Vol. 4058. LNCS. Springer, 2006, pp. 283–294.

[Pet10] Christiane Peters. "Information-Set Decoding for Linear Codes over F_q". In: *The Third International Workshop on Post-Quantum Cryptography, PQCRYPTO 2010*. Ed. by Nicolas Sendrier. Springer, Heidelberg, May 2010, pp. 81–94. DOI: 10.1007/978-3-642-12929-2_7.

[Pie67] John N. Pierce. "Limit distribution of the minimum distance of random linear codes". In: *IEEE Trans. Inform. Theory* 13.1 (1967), pp. 595–599.

[Pra62] Eugene Prange. "The use of information sets in decoding cylcic codes". In: *IRE Transactpons on Information Theory* 8.5 (1962), pp. 5–9.

[Roy22] Lawrence Roy. "SoftSpokenOT: Quieter OT Extension from Small-Field Silent VOLE in the Minicrypt Model". In: *CRYPTO 2022, Part I*. Ed. by Yevgeniy Dodis and Thomas Shrimpton. Vol. 13507. LNCS. Springer, Heidelberg, Aug. 2022, pp. 657–687. DOI: 10.1007/978-3-031-15802-5_23.

[RRT23] Srinivasan Raghuraman, Peter Rindal, and Titouan Tanguy. "Expand-Convolute Codes for Pseudorandom Correlation Generators from LPN". In: *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part IV*. Ed. by Helena Handschuh and Anna Lysyanskaya. Vol. 14084. Lecture Notes in Computer Science. Springer, 2023, pp. 602–632. DOI: 10.1007/978-3-031-38551-3_19. URL: https://doi.org/10.1007/978-3-031-38551-3%5C_19.

[Saa07] Markku-Juhani Olavi Saarinen. "Linearization Attacks Against Syndrome Based Hashes". In: *INDOCRYPT 2007*. Ed. by K. Srinathan, C. Pandu Rangan, and Moti Yung. Vol. 4859. LNCS. Springer, Heidelberg, Dec. 2007, pp. 1–9.

[Sen11] Nicolas Sendrier. "Decoding One Out of Many". In: *Post-Quantum Cryptography 2011*. Vol. 7071. LNCS. 2011, pp. 51–67.

[Sen23] Nicolas Sendrier. "Wave Parameter Selection". In: *Post-Quantum Cryptography*. Ed. by Thomas Johansson and Daniel Smith-Tone. Cham: Springer Nature Switzerland, 2023, pp. 91–110. ISBN: 978-3-031-40003-2.

[Shp09] Amir Shpilka. "Constructions of low-degree and error-correcting ε-biased generators". In: (2009).

[Ste+24] W. A. Stein et al. *Sage Mathematics Software (Version 10.2)*. http://www.sagemath.org. The Sage Development Team. 2024.

[Ste89] Jacques Stern. "A method for finding codewords of small weight". In: *Coding Theory and Applications* 388.10 (1989), pp. 106–113.

[STR69] V. STRASSEN. "Gaussian Elimination is not Optimal." In: *Numerische Mathematik* 13 (1969), pp. 354–356. URL: http://eudml.org/doc/131927.

[Tor17] Rodolfo Canto Torres. "Optimizing BJMM with Nearest Neighbors:Full Decoding in $2^{2/21n}$ and McEliece Security". In: *WCC Workshop on Coding and Cryptography WCC 2017* (2017).

[Wag02] David Wagner. "A Generalized Birthday Problem". In: *CRYPTO 2002*. Ed. by Moti Yung. Vol. 2442. LNCS. Springer, Heidelberg, Aug. 2002, pp. 288–303. DOI: 10.1007/3-540-45708-9_19.

[Yao82] Andrew C Yao. "Theory and application of trapdoor functions". In: *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*. IEEE. 1982, pp. 80–91.

[ZJW16] Bin Zhang, Lin Jiao, and Mingsheng Wang. "Faster Algorithms for Solving LPN". In: *EUROCRYPT 2016, Part I*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Vol. 9665. LNCS. Springer, Heidelberg, May 2016, pp. 168–195. DOI: 10.1007/978-3-662-49890-3_7.

Appendix A

Differed definitions related to PCG and PCF

In this section, we present the formal definitions of the programmability, introduced in Section 3.3.

Definition A.0.1 (Programmable PCG)**.** *A tuple of algorithms* $\mathsf{PCG} = (\mathsf{PCG.Gen}, \mathsf{PCG.Expand})$ *follows the syntax of a standard* PCG*, with the following modifications:* $\mathsf{PCG.Gen}(1^\lambda)$ *takes additional random inputs* $\rho_0, \rho_1 \in \{0,1\}^\kappa$*, for a fixed parameter* κ *of size* $\mathsf{poly}(\lambda)$*, is a* programmable PCG *for a simple bilinear 2-party correlation* C_e^n *(specified by a bilinear pairing* $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ *for some groups* $\mathbb{G}_1, \mathbb{G}_2$ *and* $\mathbb{G}_T$*) if the following holds:*

- ***Correctness.*** *The correlation obtained via:*

$$\left\{ ((R_0, S_0), (R_1, S_1)) \;\middle|\; \begin{array}{c} \rho_0, \rho_1 \xleftarrow{\$} \{0,1\}^\kappa, (k_0, k_1) \leftarrow \mathsf{PCG.Gen}(1^\lambda, \rho_0, \rho_1), \\ (R_\sigma, S_\sigma) \leftarrow \mathsf{PCG.Expand}(\sigma, k_\sigma) \text{ for } \sigma \in \{0,1\} \end{array} \right\}$$

  is computationally indistinguishable from $C_e^n(1^\lambda)$*.*

- ***Programmability****. There exist public efficiently computable functions* $\psi_0 : \{0,1\}^* \to \mathbb{G}_1^n$*,* $\psi_1 : \{0,1\}^* \to \mathbb{G}_2^n$ *such that*

$$\Pr_{\substack{\rho_0, \rho_1 \xleftarrow{\$} \{0,1\}^\kappa, (k_0, k_1) \leftarrow \mathsf{PCG.Gen}(1^\lambda, \rho_0, \rho_1) \\ (R_0, S_0) \leftarrow \mathsf{PCG.Expand}(0, k_0), \\ (R_1, S_1) \leftarrow \mathsf{PCG.Expand}(1, k_1)}} \left[R_0 = \psi_0(\rho_0), \quad R_1 = \psi_1(\rho_1) \right] \geqslant 1 - \mathsf{negl}(\lambda).$$

- ***Programmable security****. We define the following distributions:*

$$A = \left\{ (k_1, (\rho_0, \rho_1)) \;\middle|\; \rho_0, \rho_1 \xleftarrow{\$} \{0,1\}^\kappa, (k_0, k_1) \leftarrow \mathsf{PCG.Gen}(1^\lambda, \rho_0, \rho_1) \right\}$$

$$B = \left\{ (k_1, (\rho_0, \rho_1)) \;\middle|\; \rho_0, \rho_1, \tilde{\rho_0} \xleftarrow{\$} \{0,1\}^\kappa, (k_0, k_1) \leftarrow \mathsf{PCG.Gen}(1^\lambda, \tilde{\rho_0}, \rho_1) \right\}$$

$$C = \left\{ (k_0, (\rho_0, \rho_1)) \;\middle|\; \rho_0, \rho_1 \xleftarrow{\$} \{0,1\}^\kappa, (k_0, k_1) \leftarrow \mathsf{PCG.Gen}(1^\lambda, \rho_0, \rho_1) \right\}$$

$$D = \left\{ (k_0, (\rho_0, \rho_1)) \;\middle|\; \rho_0, \rho_1, \tilde{\rho_1} \xleftarrow{\$} \{0,1\}^\kappa, (k_0, k_1) \leftarrow \mathsf{PCG.Gen}(1^\lambda, \rho_0, \tilde{\rho_1}) \right\}$$

Security asks that $A \approx B$ *and* $C \approx D$.

Appendix B

Existing Generic Attacks on Syndrome Decoding

Outline of the current chapter

An important method to attack Syndrome Decoding problems in the general case is to look at the Information Set Decoding types of algorithms. Starting with the seminal work of Prange [Pra62], , this line of algorithms has been extensively studied and improved via Birthday-like paradox technique [Ste89], representation technique [MMT11], nearest neighbors [MO15a] or sieving [GJN23]. ISD algorithms are among the bests known attacks so far on purely random codes, and as such, the security analysis of the Syndrome Decoding assumption has to take into account the attack. We describe a general template in the next subsection, and will specify the different specifities afterwards.

B.1 General ISD framework.

We consider Syndrome Decoding in its search variant: given a matrix $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$ and $\mathbf{y} \in \mathbb{F}_q^{n-k}$, find $\mathbf{e} \in \mathbb{F}_q^n$ such that the weight of $\mathbf{e}$ achieves a certain target t.

ISD algorithms take their strength and their name from the concept of Information Set. For a set $\mathcal{I} \subset [0, n-1]$, we define by $\overline{\mathcal{I}}$ it's complementary in $[0, n-1] : \overline{\mathcal{I}} = [0, n-1] \backslash \mathcal{I}$. For a matrix $\mathbf{H}$, we denote by $\mathbf{H}_\mathcal{I}$ the restriction of $\mathbf{H}$ to the columns indicated by $\mathcal{I}$, and for a vector $\mathbf{e}$, we write $e_\mathcal{I}$ for the restriction of $\mathbf{e}$ to the entries indexed by $\mathcal{I}$. Given a matrix $\mathbf{H}$, an Information Set is a set $\mathcal{I} \subset [0, n-1]$, such that the $|\mathcal{I}| = k + \ell, \ell \in \mathbb{N}$, and $\mathbf{H}_{\overline{\mathcal{I}}}$ is full rank. The goal of an ISD would be to guess an Information Set such that errors are not in it. Because all the information are contained in the information set, it is clear we can break everything once you get the proper information set.

What follows is a template for ISD algorithm freely inspired from the lectures notes of [Deb23b].

Consider two parameters $0 \leqslant \ell \leqslant n-k$ and $0 \leqslant p \leqslant \min(t, k+\ell)$

1. Pick randomly a subset $\mathcal{I} \subset [0, n-1], |\mathcal{I}| = k + \ell$, until $\mathbf{H}_{\overline{\mathcal{I}}}$ is full rank.
2. Perform a Gaussian elimination to compute a non-singular matrix $\mathbf{U} \in \mathbb{F}_q^{(n-k)\times(n-k)}$ such that $\mathbf{U}\mathbf{H}_{\overline{\mathcal{I}}} = \begin{bmatrix} I_{n-k-\ell} \\ 0 \end{bmatrix}$, and compute $\tilde{\mathbf{s}} = \mathbf{U}\mathbf{s}$. Write $\mathbf{U}\mathbf{H}_\mathcal{I} = \begin{bmatrix} \mathbf{H_1} \\ \mathbf{H_2} \end{bmatrix}$.
3. Compute $\mathcal{L} \subset \{\mathbf{e_2} \in \mathbb{F}_q^{k+\ell}, wt(\mathbf{e_2}) = p, \mathbf{H_2}\mathbf{e_2} = \tilde{\mathbf{s}}_\mathcal{I}\}$ according to a given ISD-subroutine.
4. Find a $\mathbf{e_2} \in \mathcal{L}$ such that $\mathbf{e_1} = \tilde{\mathbf{s}}_{\overline{\mathcal{I}}} - \mathbf{H_1}\mathbf{e_2}$ has weight $t - p$. If no such $\mathbf{e_2}$ is found, return to Step 1. If it is found, return the vector $\mathbf{e'}$ such that $\mathbf{e'}_\mathcal{I} = \mathbf{e_2}$ and $\mathbf{e}_{\overline{\mathcal{I}}} = \mathbf{e_1}$.

Remark B.1.1. We did not take into account permutation matrix because it has no purposes aside pedagogical. While [Sen23] also offers a comparable general framework, it introduces a permutation matrix to denote an information set, a classic selection for clarity reasons. However, in practical application, this approach introduced overheads due to matrix multiplication. Consequently, we have opted not to pursue it.

An ISD algorithm is an iterative algorithm that loops until it finds a solution. The algorithm is therefore probabilistic, with its iterations being independent, and its complexity is given by the complexity of Steps 2 and 3, multiplied by the number of iterations (i.e., the inverse of the probability of finding an information set for which the procedure terminates). An instance of the general ISD algorithm is given by the choice of the parameters (ℓ, p) and of the ISD subroutine to compute $\mathcal{L}$. Next, based on the work of Sendrier [Sen23], we expose the practical complexity of the different variants of the ISD algorithm defined above and take into account some polynomial factors while being conservative.

Remark B.1.2. In this analysis, we do not consider the modern and more recent attacks on syndrome decoding using nearest neighbors search [Tor17; BM18], as they are not considered to be practical, even in the binary field case. To the best of our knowledge, there has be no work on extending these algorithms to the general case of $\mathbb{F}_q$, and we believe that it would not be practical in our range of parameters.

We denote by $T(n, k, q, *)$ the complexity of a given ISD, $*$ indicating here the different parameters of the variant considered to be tuned. As stated before an ISD algorithm is by essence probabilistic,

and its complexity can be formulated like this:

$$T(n,k,q,t,*) = \frac{T_G(n,k,q,*) + C_S(n,k,q,*)}{\mathcal{P}_S(n,k,q,t,*)},$$

where $T_G(n,k,q,*)$ counts the costs of the Gaussian elimination done at each iteration, $C_S(n,k,q,*)$ is the complexity of the subroutine chosen for this ISD, and $\mathcal{P}_S(n,k,q,t,*)$ the probability of randomly selecting an Information Set that led to a solution.

Remark B.1.3 (Gaussian Elimination Cost)**.** For our estimations, we consider the following cost

$$T_G(n,k,q,p,\ell) = (n-k-\ell)n.$$

In other words, we consider here that Gaussian elimination costs just as much as the size of the matrix, which is a very conservative lower bound.

B.2 Stern/Dummer

The algorithm proposed by Stern [Ste89] and independently by Dummer [Dum89] corresponds to the case where we consider the following subroutine:

1. We create two lists

$$\mathcal{L}_1 = \left\{ \left(\mathbf{e_0} = \begin{bmatrix} \mathbf{e'} \\ \mathbf{0} \end{bmatrix}, \mathbf{H_2 e_0} \right) \;\middle|\; \mathbf{e'} \in \mathbb{F}_q^{\frac{k+\ell}{2}}, wt\left(\mathbf{e'}\right) = \frac{p}{2} \right\}$$
$$\mathcal{L}_2 = \left\{ \left(\mathbf{e'_0} = \begin{bmatrix} \mathbf{0} \\ \mathbf{e'} \end{bmatrix}, \tilde{\mathbf{s}}_{\mathcal{I}} - \mathbf{H_2 e'_0} \right) \;\middle|\; \mathbf{e'} \in \mathbb{F}_q^{\frac{k+\ell}{2}}, wt\left(\mathbf{e'}\right) = \frac{p}{2} \right\}$$

   each list is of size $|\mathcal{L}_1| = |\mathcal{L}_2| = \binom{(k+\ell)/2}{p/2}(q-1)^{p/2}$.

   Complexity: $O(|\mathcal{L}_1|)$.

2. Search for all pairs $((\mathbf{e_0}, \mathbf{H_2 e_0}), (\mathbf{e'_0}, \tilde{\mathbf{s}}_{\mathcal{I}} - \mathbf{H_2 e'_0})) \in \mathcal{L}_1 \times \mathcal{L}_2$ such that $\mathbf{H_2} \cdot \mathbf{e_0} = \tilde{\mathbf{s}}_{\mathcal{I}} - \mathbf{H_2 e'_0}$, and construct the list

$$\mathcal{L} = \{\mathbf{e_0} + \mathbf{e'_0}, ((\mathbf{e_0}, \mathbf{H_2 e_0}), (\mathbf{e'_0}, \tilde{\mathbf{s}}_{\mathcal{I}} - \mathbf{H_2 e'_0}) \in \mathcal{L}_1 \times \mathcal{L}_2, \mathbf{H_2} \cdot \mathbf{e_0} = \tilde{\mathbf{s}}_{\mathcal{I}} - \mathbf{H_2 e'_0}\}.$$

   The list $\mathcal{L}$ created like this satisfies that

$$\mathcal{L} \subset \{\mathbf{e_2} \in \mathbb{F}_q^{k+\ell}, wt\left(\mathbf{e_2}\right) = p, \mathbf{H_2 e_2} = \tilde{\mathbf{s}}_{\mathcal{I}}\}.$$

   The size of the list is $|\mathcal{L}| = \binom{(k+\ell)/2}{p/2}^2 (q-1)^p q^{-l}$.

   Complexity: $O(\max(|\mathcal{L}_1|, |\mathcal{L}|))$.

3. We perform Step 4 of the general ISD framework to find e_1 of weight $t-p$.

   Complexity: $O(|\mathcal{L}|(n-k-\ell)(k+\ell))$.

Proposition B.2.1 (Stern Algorithm). *The complexity of an iteration of the Stern algorithm is given by*

$$C_S(n,k,q,p,l) = O(|\mathcal{L}|(n-k-\ell)(k+\ell) + \max(|\mathcal{L}_1|,|\mathcal{L}|) + |\mathcal{L}_1|)$$

with

$$|\mathcal{L}_1| = \binom{(k+\ell)/2}{p/2}(q-1)^{p/2} \qquad |\mathcal{L}| = \binom{(k+\ell)/2}{p/2}^2 (q-1)^p q^{-l}$$

and the probability of finding a solution is

$$\mathcal{P}_S(n,k,q,t,p,l) = \frac{\binom{k+\ell}{p}\binom{n-k-\ell}{t-p}}{\binom{n}{t}} \cdot \frac{\binom{(k+\ell)/2}{p/2}^2}{\binom{k+\ell}{p}} = \frac{\binom{(k+\ell)/2}{p/2}^2\binom{n-k-\ell}{t-p}}{\binom{n}{t}}.$$

Remark B.2.1. The probability of success is obtained as the probability of having an error vector that has exactly p 1's on the chosen information set (first fraction) multiplied by the probability that the error on the information set is equally distributed into the two halves of the corresponding vector $e_{\mathcal{I}}$.

Remark B.2.2. The complexity given at Step 1 and 2 is voluntarily given without any polynomial factor, for conservative purposes. In fact, for each element of the lists, we have to perform the computation of $\tilde{\mathbf{s}}_{\mathcal{I}} - \mathbf{H_2 e'_0}$. This costs naively $\ell(\ell+k)$ per element in the list (for a total cost of $O(\ell(\ell+k)|\mathcal{L}_1|)$. Second, we have to check for equality among vectors of size l, and this costs l for each element in the list (for a total cost of $O(\max(|\mathcal{L}_1|,|\mathcal{L}|)\cdot\ell)$. Nevertheless, some optimization exists for these Steps (see [Pet10]). Therefore, we chose to stick with $O(|\mathcal{L}_1|)$ and $O(\max(|\mathcal{L}_1|,\mathcal{L}|))$, as conservative lower bounds.

Corollary B.2.1 (Prange and Lee Brickell complexities). *The Prange algorithm [Pra62] is a very particular case of Stern algorithm with $p=0, \ell=0$ and no subroutine. In the case of Prange, we have*

$$C(n,k,q,t) = 0 \qquad \mathcal{P}_S(n,k,q,t) = \frac{\binom{n-k}{t}}{\binom{n}{t}}.$$

The same goes for the Lee-Brickell algorithm [LB88], by taking $\ell=0$ with only p to optimize, and no subroutine. In the case of Lee-Brickell we therefore have:

$$C(n,k,q,t) = 0 \qquad \mathcal{P}_S(n,k,q,t) = \frac{\binom{k}{p}\binom{n-k}{t-p}}{\binom{n}{t}}.$$

B.3 MMT

The MMT algorithm [MMT11] of May et al. introduced the *representation technique*. They remark that we can split a given vector $\mathbf{e}$ of weight p into R different sums of two vectors $\mathbf{e_1}$ and $\mathbf{e_2}$, of the same size but weight $p/2$, and disjoint support.

$$\mathbf{e} = \mathbf{e_1} + \mathbf{e_2}, \quad wt(\mathbf{e_1}) = p/2, \quad wt(\mathbf{e_2}) = p/2.$$

Remark that $\mathbf{H_2 e} = \tilde{\mathbf{s}}_{\mathcal{I}}$ implies $\mathbf{H_2 e_1} = \tilde{\mathbf{s}}_{\mathcal{I}} - \mathbf{H_2 e_2}$. Therefore they create $\mathcal{L}_{1,0} \subset \{\mathbf{H_2 e_1}, wt(\mathbf{e_1}) = p/2\}$ and $\mathcal{L}_{1,1} \subset \{\tilde{\mathbf{s}}_{\mathcal{I}} - \mathbf{H_2 e_2}, wt(\mathbf{e_2}) = p/2\}$ and look for possible collisions. R is called the number of representations, and we have that

$$R = \binom{p}{p/2}.$$

Let $0 \leqslant r = \log_q(R) \leqslant l$. The subroutine is as follows:

1. For $0 \leqslant i \leqslant 1$, construct the following lists

$$\mathcal{L}_{i,0} = \left\{ \left(\mathbf{e_0} = \begin{bmatrix} \mathbf{e'} \\ \mathbf{0} \end{bmatrix}, \mathbf{H_2 e_0} \right) \;\middle|\; \mathbf{e'} \in \mathbb{F}_q^{\frac{k+\ell}{2}}, wt\left(\mathbf{e'}\right) = \frac{p}{4} \right\},$$
$$\mathcal{L}_{i,1} = \left\{ \left(\mathbf{e'_0} = \begin{bmatrix} \mathbf{0} \\ \mathbf{e'} \end{bmatrix}, \tilde{\mathbf{s}}_{\mathcal{I}} - \mathbf{H_2 e'_0} \right) \;\middle|\; \mathbf{e'} \in \mathbb{F}_q^{\frac{k+\ell}{2}}, wt\left(\mathbf{e'}\right) = \frac{p}{4} \right\}.$$

   The size of each list is $|\mathcal{L}_{0,0}| = |\mathcal{L}_{0,1}| = |\mathcal{L}_{1,0}| = |\mathcal{L}_{1,1}| = \binom{(k+\ell)/2}{p/4}(q-1)^{p/4}$.

   Complexity: $O(|\mathcal{L}_{0,0}|)$.

2. For $0 \leqslant i \leqslant 1$, and given a fix set of r entries, search for all pairs $((\mathbf{e_0}, \mathbf{H_2 e_0}), (\mathbf{e'_0}, \tilde{\mathbf{s}}_{\mathcal{I}} - \mathbf{H_2 e'_0})) \in \mathcal{L}_{i,0} \times \mathcal{L}_{i,1}$ such that $\mathbf{H_2} \cdot \mathbf{e_0} = \tilde{\mathbf{s}}_{\mathcal{I}} - \mathbf{H_2 e'_0}$ on the r entries. Construct the list

$$\mathcal{L}_{i,2} = \left\{ \mathbf{e_0} + \mathbf{e'_0}, (\mathbf{e_0}, \mathbf{H_2 e_0}) \;\middle|\; (\mathbf{e'_0}, \tilde{\mathbf{s}}_{\mathcal{I}} - \mathbf{H_2 e'_0})) \in \mathcal{L}_{i,0} \times \mathcal{L}_{i,1}, \mathbf{H_2 e_0} = \tilde{\mathbf{s}}_{\mathcal{I}} - \mathbf{H_2 e'_0} \right\}.$$

   We have that

$$\mathcal{L}_{i,2} \subset \left\{ \mathbf{e_2} \in \mathbb{F}_q^{k+\ell} \;\middle|\; wt\left(\mathbf{e_2}\right) = p/2, \mathbf{H_2 e_2} = \tilde{\mathbf{s}}_{\mathcal{I}} \right\}.$$

   The size of the lists resulting from the merge is $|\mathcal{L}_{0,2}| = |\mathcal{L}_{1,2}| = |\mathcal{L}_{0,0}|^2/q^r$.

   Complexity: $O(\max(|\mathcal{L}_{0,0}|, |\mathcal{L}_{0,2}|))$.

3. Merge the two previous lists again, to obtain $\mathcal{L}$ which contains vectors of size $k+\ell$, weight p, and with the appropriate fixed value for the remaining $\ell - r$ coordinates of their syndromes. The size of the resulting list is $|\mathcal{L}| = |\mathcal{L}_{0,0}|^4/q^{l+r}$.

   Complexity: $O(\max(|\mathcal{L}_{0,2}|, |\mathcal{L}|))$.

4. We perform Step 4 of the general ISD framework to find $\mathbf{e_1}$ of weight $t - p$.

   Complexity: $O(|\mathcal{L}|(n-k-\ell)(k+\ell))$.

Proposition B.3.1 (General MMT algorithm)**.** *The complexity of an iteration of the MMT algorithm is given by*

$$C_S(n,k,q,p,l) = O(|\mathcal{L}|(n-k-\ell)(k+\ell) + \max(|\mathcal{L}_{0,0}|, |\mathcal{L}_{0,2}|, |\mathcal{L}|) + \mathcal{L}_{0,0})$$

with

$$|\mathcal{L}_{0,0}| = \binom{(k+\ell)/2}{p/4}(q-1)^{p/4} \qquad |\mathcal{L}_{0,2}| = \binom{(k+\ell)/2}{p/4}^2 (q-1)^{p/2} q^{-r}$$
$$|\mathcal{L}| = \binom{(k+\ell)/2}{p/4}^4 (q-1)^p q^{-\ell-r} \qquad R = \binom{p}{p/2}$$
$$r = \log_q(R)$$

and the probability of finding a solution is

$$\mathcal{P}_S(n,k,q,t,p,l) = \frac{\binom{k+\ell}{p}\binom{n-k-\ell}{t-p}}{\binom{n}{t}} \cdot \frac{\binom{(k+\ell)/2}{p/4}^4}{\binom{k+\ell}{p/2}^2}.$$

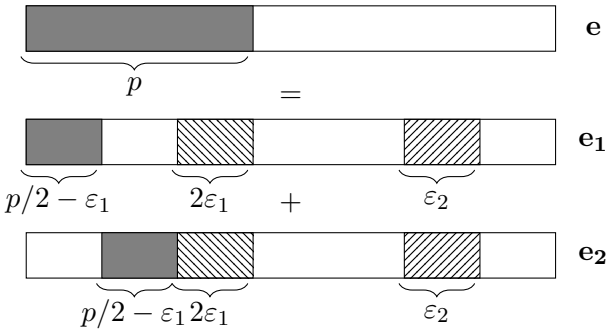

Figure B.1 – BJMM representations

Remark B.3.1. The probability of success is obtained as the probability of having an error vector that has exactly p errors on the chosen Information Set (first fraction) multiplied by the probability that $\mathbf{e_1}$ has $p/2$ errors equally distributed into its two halves, multiplied again by the probability that $\mathbf{e_1}$ has $p/2$ errors equally distributed into its two halves.

B.4 BJMM

The algorithm introduced by Becker et al. [BJMM12], proposes an improvement of the MMT algorithm, by increasing the number of representations one can get. Their idea follows the approach of MMT, except that an additional parameter ε is introduced. Let $p_1 = p/2 + \varepsilon$. Their idea is still to write a vector $\mathbf{e} \in \mathbb{F}_q^{k+\ell}$, of weight p as a sum.

$$\mathbf{e} = \mathbf{e_1} + \mathbf{e_2}$$

but now we want to allow their support to coincide on a proportion ε, and we ask that $wt(\mathbf{e_1}) = wt(\mathbf{e_2}) = p_1$.

The support of $\mathbf{e_1}$ and $\mathbf{e_2}$ can coincide in two different ways: either the sum of the entries vanishes, or they do not vanish. For this reason, we split $\varepsilon = \varepsilon_1 + \varepsilon_2$. For $\mathbf{e}$ to have a support of size p, the support of $\mathbf{e_1}$ and $\mathbf{e_2}$ should coincide on the same $2\varepsilon_1 + \varepsilon_2$ entries, and be distinct on the other $p/2 - \varepsilon_1$ entries, as represented in Figure B.1. The number of representations is therefore given by

$$R = \sum_{\varepsilon_1+\varepsilon_2=\varepsilon} \binom{p}{p/2-\varepsilon_1}\binom{p/2+\varepsilon_1}{2\varepsilon_1}\binom{k+\ell-p}{\varepsilon_2}(q-1)^{2\varepsilon_1+\varepsilon_2}.$$

Let $r = \log_q(R)$. The subroutine is as follows:

1. For $0 \leqslant i \leqslant 1$ construct the following lists

$$\mathcal{L}_{i,0} = \left\{ \left(\mathbf{e_0} = \begin{bmatrix} \mathbf{e'} \\ \mathbf{0} \end{bmatrix}, \mathbf{H_2 e_0} \right) \;\middle|\; \mathbf{e'} \in \mathbb{F}_q^{\frac{k+\ell}{2}}, wt(\mathbf{e'}) = \frac{p_1}{2} \right\},$$
$$\mathcal{L}_{i,1} = \left\{ \left(\mathbf{e'_0} = \begin{bmatrix} \mathbf{0} \\ \mathbf{e'} \end{bmatrix}, \tilde{\mathbf{s}}_{\mathcal{I}} - \mathbf{H_2 e'_0} \right) \;\middle|\; \mathbf{e'} \in \mathbb{F}_q^{\frac{k+\ell}{2}}, wt(\mathbf{e'}) = \frac{p_1}{2} \right\}.$$

The size of each list is $|\mathcal{L}_{0,0}| = |\mathcal{L}_{0,1}| = |\mathcal{L}_{1,0}| = |\mathcal{L}_{1,1}| = \binom{(k+\ell)/2}{p_1/2}(q-1)^{p_1/2}$.

Complexity: $O(|\mathcal{L}_{0,0}|)$.

2. For $0 \leqslant i \leqslant 1$, and given a fix set of r entries, search for all pairs $(\mathbf{e_0}, \mathbf{e'_0}) \in \mathcal{L}_{i,0} \times \mathcal{L}_{i,1}$ such that $\mathbf{H_2} \cdot \mathbf{e_0} = \tilde{\mathbf{s}}_{\mathcal{I}} - \mathbf{H_2}\mathbf{e'_0}$ on the r entries. Construct the list

$$\mathcal{L}_{i,2} = \left\{ \mathbf{e_0} + \mathbf{e'_0} \;\middle|\; (\mathbf{e_0}, \mathbf{H_2}\mathbf{e_0}), (\mathbf{e'_0}, \tilde{\mathbf{s}}_{\mathcal{I}} - \mathbf{H_2}\mathbf{e'_0})) \in \mathcal{L}_{i,0} \times \mathcal{L}_{i,1}, \mathbf{H_2} \cdot \mathbf{e_0} = \tilde{\mathbf{s}}_{\mathcal{I}} - \mathbf{H_2}\mathbf{e'_0} \right\}.$$

We have that

$$\mathcal{L}_{i,2} \subset \left\{ \mathbf{e_2} \in \mathbb{F}_q^{k+\ell} \mid wt\,(\mathbf{e_2}) = p_1/2, \mathbf{H_2}\mathbf{e_2} = \tilde{\mathbf{s}}_{\mathcal{I}} \right\}$$

The size of the lists resulting from the merge is $|\mathcal{L}_{0,2}| = |\mathcal{L}_{1,2}| = |\mathcal{L}_{0,0}|^2/q^r$.

Complexity: $O(\max(|\mathcal{L}_{0,0}|, |\mathcal{L}_{0,2}|))$.

3. Merge the two previous lists again, to obtain $\mathcal{L}$ which contains vectors of size $k+\ell$, weight p, and with the appropriate fixed value for the remaining $\ell - r$ coordinates of their syndromes. The size of the resulting list is $|\mathcal{L}| = |\mathcal{L}_{0,0}|^4/q^{l+r}$.

Complexity: $O(\max(|\mathcal{L}_{0,2}|, |\mathcal{L}|))$.

4. We perform Step 4 of the general ISD framework to find $\mathbf{e_1}$ of weight $t - p$.

Complexity: $O(|\mathcal{L}|(n-k-\ell)(k+\ell))$.

Proposition B.4.1 (BJMM algorithm)**.** *The complexity of an iteration of the BJMM algorithm is given by*

$$C_S(n,k,q,p,l,\varepsilon) = O(|\mathcal{L}|(n-k-\ell)(k+\ell) + \max(|\mathcal{L}_{0,0}|, |\mathcal{L}_{0,2}|, |\mathcal{L}|) + \mathcal{L}_{0,0})$$

with

$$|\mathcal{L}_{0,0}| = \binom{(k+\ell)/2}{p_1/2}(q-1)^{p_1/2} \qquad |\mathcal{L}_{0,2}| = \binom{(k+\ell)/2}{p_1/2}^2 (q-1)^{p_1} q^{-r}$$
$$|\mathcal{L}| = \binom{(k+\ell)/2}{p_1/2}^4 (q-1)^{2p_1} q^{-\ell-r} \qquad p_1 = p/2 + \varepsilon$$

and

$$R = \sum_{\varepsilon_1+\varepsilon_2=\varepsilon} \binom{p}{p/2-\varepsilon_1}\binom{p/2+\varepsilon_1}{2\varepsilon_1}\binom{k+\ell-p}{\varepsilon_2}(q-1)^{2\varepsilon_1+\varepsilon_2}$$

$$r = \log(R)$$

and the probability of finding a solution is

$$\mathcal{P}_S(n,k,q,t,p,l,\varepsilon) = \frac{\binom{k+\ell}{p}\binom{n-k-\ell}{t-p}}{\binom{n}{t}} \cdot \frac{\binom{(k+\ell)/2}{p_1/2}^4}{\binom{k+\ell}{p_1}^2}.$$

Remark B.4.1. The probability of success is obtained as the probability of having an error vector that has exactly p errors on the chosen Information Set (first fraction) multiplied by the probability that $\mathbf{e_1}$ has p_1 errors equally distributed into its two halves, multiplied again by the probability that $\mathbf{e_2}$ has p_1 errors equally distributed into its two halves.

Remark B.4.2. We want to stress the fact that in the case of the MMT and BJMM-like attacks, our estimations are quite conservative, as we do not take into account the costs implied by the memory calls. In our range of parameters, they should not be discarded in practice. Indeed, the memory size is about the size of the biggest lists computed, therefore

$$M(n, k, q, p, l) \sim \max(|\mathcal{L}_{0,2}|, |\mathcal{L}_{1,1}|, |\mathcal{L}|),$$

which can be be over 2^{50}, and entails an overhead in the computation time. This is often neglected in the literature.

Appendix C

Proving the Resistance against Linear Test of the Original VDSD Construction

We put forward a corrected detailed analysis of the resistance of the original rVDSD against linear tests. Our proof fixes the two errors in [BCGI+20a], at the cost of achieving worse constants, and being more involved. As before, we study individually the bias induced by the $\mathbf{H}_i$ components against vectors of weight close to 2^i. The general proof (Appendix C.1) will nevertheless focus only on large enough values of i: we assume $n = 2^i \geqslant 2^7$. The missing cases are handled separately in Appendix C.2.

Definition C.0.1 (δ-Bad Matrices). *Let $\mathbf{M} \in \mathbb{F}_2^{N \times w \cdot 2^i}$, build as a concatenation of w matrix M_j of size $N \times 2^i$. We say that a block $\mathbf{M}_j$ is* bad *with respect to a vector $\mathbf{v} \in \mathbb{F}_{2^N}$ if*

$$wt\left(\mathbf{v}^\top \cdot \mathbf{M}_j\right) = R_{l,k} \notin \left[\delta \cdot 2^i, (1-\delta) \cdot 2^i\right].$$

Stated in terms of $Z_{l,k}$, this condition rewrites to $Z_{l,k} \in \left[(1/2 - \delta) \cdot 2^i, 2^{i-1}\right]$.

We let $\mathsf{Bad}_{\delta,\mathbf{v},w}$ be the set of all $\mathbf{M}_j$ that are bad. *We let $\mathsf{Good}_{\delta,\mathbf{v},w}$ denote the complement of $\mathsf{Bad}_{\delta,\mathbf{v},w}$. Given vector $\mathbf{v}$, we also denote $B_{\delta,\mathbf{v},w} = \#\mathsf{Bad}_{\delta,\mathbf{v},w}$ and $G_{\delta,\mathbf{v},w} = \#\mathsf{Good}_{\delta,\mathbf{v},w} = w - B_{\delta,\mathbf{v},w}$.*

C.1 The general proof

We now formally prove Theorem 7.2.2. Let $\mathcal{O}^i_{\mathsf{par}}(\mathbf{H})$ be the distribution induced by sampling $\mathbf{e_i}$ (as a concatenation of w length-2^i vectors) and outputting $\mathbf{H}_i \cdot \mathbf{e_i}$. A sample from $\mathcal{O}^i_{\mathsf{par}}(\mathbf{H})$ can be further decomposed as $\bigoplus_{j \leqslant w} \mathbf{H}_{i,j} \cdot \mathbf{e_{i,j}}$ where the $\mathbf{e_{i,j}}$ are unit vectors. Let D_i denote the distribution of $\mathbf{H}_{i,j} \cdot \mathbf{e_{i,j}}$ (these terms are w samples from the same distribution).Let α be a constant. Then,

Lemma C.1.1. *If $B_{\delta,\mathbf{v},w} \leqslant \alpha \cdot w$, then*

$$\mathsf{bias}\left(\bigoplus_{i=1}^{w} D_i\right) \leqslant \frac{1}{2} \cdot ((1-2\delta)^{(1-\alpha)})^w.$$

Proof. By the piling-up lemma (Lemma 2.2.2),

$$\mathsf{bias}\left(\bigoplus_{i=1}^{w} D_i\right) \leqslant 2^{(1-\alpha)w-1} \cdot \left(\frac{1}{2}-\delta\right)^{(1-\alpha)\cdot w} \leqslant \frac{1}{2}\cdot((1-2\delta)^{(1-\alpha)})^w$$

□

Lemma C.1.1 provides an upper bound of the bias, which depends on the number of good matrices and their quality. We now show that the condition $B_{\delta,\mathbf{v}} \leqslant \alpha \cdot w$ holds with very high probability:

Lemma C.1.2. *For any* $\mathbf{v} \in S_{i,N}$*, there is a constant* C *such that*

$$\Pr\left[\, B_{\delta,\mathbf{v}} > \alpha \cdot w \,\right] \leqslant 2^{-C\cdot 2^i \cdot w}.$$

Proof. As in the original proof, we introduce the function Φ:

$$\begin{aligned}\Phi\left(\mathbf{X_{1,1}},\ldots,\mathbf{X_{l,w}}\right) &= \sum_{k=1}^{w}\left(2^{i-1}-\left|wt\left(\bigoplus_{j=1}^{l}\mathbf{X_{j,k}}\right)-2^{i-1}\right|\right)\\ &= 2^{i-1}\cdot w - \sum_{k=1}^{w} Z_{l,k}.\end{aligned}$$

We want to bound the probability of large bias by a bound on Φ. This is where the first error appeared in the previous proof.

Lemma C.1.3 (Correction of the first error)**.**

$$\Pr\left[\, B_{\delta,\mathbf{v}} \geqslant \alpha\cdot w \,\right] \leqslant \Pr\left[\, \Phi(\mathbf{X_{1,1}},\ldots,\mathbf{X_{l,w}}) < \gamma\cdot w\cdot 2^i \,\right],$$

with $\gamma = \frac{1}{2} - \alpha(\frac{1}{2}-\delta)$.

Proof. We assume that $B_{\delta,\mathbf{v}} \geqslant \alpha\cdot w$. This translates to a lower bound of the sum of the $Z_{l,k}$:

$$\sum_{k=1}^{w} Z_{l,k} \geqslant \alpha\cdot w\cdot\left(\frac{1}{2}-\delta\right)\cdot 2^i,$$

and finally an upper bound on Φ.

$$\Phi\left(\mathbf{X_{1,1}},\ldots,\mathbf{X_{l,w}}\right) \leqslant 2^{i-1}\cdot w - \alpha\cdot w\cdot\left(\frac{1}{2}-\delta\right)\cdot 2^i = 2^i\cdot w\cdot\left(\frac{1}{2}-\alpha\left(\frac{1}{2}-\delta\right)\right).$$

Setting $\gamma = \frac{1}{2}-\alpha(\frac{1}{2}-\delta)$ we get $\Phi(\mathbf{X_{1,1}},\ldots,\mathbf{X_{l,w}}) < \gamma\cdot w\cdot 2^i$, which proves that

$$\Pr\left[\, B_{\delta} \geqslant \alpha\cdot w \,\right] \leqslant \Pr\left[\, \Phi(\mathbf{X_{1,1}},\ldots,\mathbf{X_{l,w}}) < \gamma\cdot w\cdot 2^i \,\right].$$

□

It remains now to find an upper bound on the right hand side probability. As in the original proof, we used the bounded difference inequality. Since Φ is 2-Lipschitz, (this was proved in the original proof),

$$\Pr\left[\, \Phi(\mathbf{X_{1,1}},\cdots,\mathbf{X_{l,w}}) \leqslant \mathbb{E}[\Phi(\mathbf{X_{1,1}},\cdots,\mathbf{X_{l,w}})] - t \,\right] \leqslant \exp\left(-\frac{t^2}{2lw}\right).$$

We finally need a lower bound on $\mathbb{E}[\phi(\mathbf{X_{1,1}}, \cdots, \mathbf{X_{l,w}})]$. Recall that $\Phi\left(\mathbf{X_{1,1}}, \cdots, \mathbf{X_{l,w}}\right) = 2^{i-1} \cdot w - \sum_{k=1}^{w} Z_{l,k}$, so this reduces to bounding $\mathbb{E}[Z_{l,k}]$. Our main contribution in this analysis is the proof of the following lemma:

Lemma C.1.4 (Correction of the second error)**.** *For all $n \in \mathbb{N}$, there exists $\beta < 1/2$ such that $\mathbb{E}\left[\mathbb{Z}_{l,k}\right] < \beta \cdot n$.*

Proof. We first rewrite $\mathbb{E}[Z_{l,k}]$ using the standard fact that $\mathbb{E}[Z] = \sum_j \Pr[Z > j]$:

$$\begin{aligned}\mathbb{E}\left[Z_{l,k}\right] = \mathbb{E}\left[\left|R_{l,k} - 2^{i-1}\right|\right] &= \sum_{j=0}^{2^{i-1}-1} \Pr\left(\left|R_{l,k} - 2^{i-1}\right| > j\right) \\ &= \sum_{j=0}^{2^{i-1}-1} \Pr(R_{l,k} \geqslant j + 1 + 2^{i-1}) + \sum_{j=0}^{2^{i-1}-1} \Pr(R_{l,k} \leqslant 2^{i-1} - j - 1).\end{aligned}$$

While we can bound $\Pr\left[R_{l,k} \geqslant j + 1 + 2^{i-1}\right] + \Pr\left[R_{l,k} \leqslant 2^{i-1} - j - 1\right]$ by 1 (for every j the two events are disjoint), this only proves that $\mathbb{E}\left[Z_{l,k}\right] \leqslant 0.5 \cdot n$. Therefore, we are looking for better bounds on these two probabilities. Both bounds come from the Lemma 2.2.4; we prove each of them separately below.

Lemma C.1.5. *Let $\frac{1}{2} < p < 1$. Let θ such that $(1-p) \cdot n = \mu \cdot (1-\theta)$. Then,*

$$\Pr\left[\, R_{l,k} \geqslant pn \,\right] \leqslant 2 \exp\left(-\frac{\theta^2 \mu^2 (n - \frac{1}{2})}{n^2 - \mu^2}\right).$$

Proof. Let E be the random variable equal to the number of empty bins. Remark that when $E > x$ then $R_{l,k} < 2^i - x$. Then, it appears necessarily that

$$\Pr\left[\, R_{l,k} \geqslant pn \,\right] \leqslant \Pr[E \leqslant (1-p)n].$$

We can then use lemma 2.2.4, and establish the following

$$\Pr\left[\, R_{l,k} \geqslant pn \,\right] \leqslant 2 \exp\left(-\frac{\theta^2 \mu^2 (n - \frac{1}{2})}{n^2 - \mu^2}\right),$$

where we chose θ is chosen such that $(1-p)n = \mu \cdot (1-\theta)$. This proves the first bound we need. □

Let's focus now on the second bound.

Lemma C.1.6. *Let p such that $\frac{1}{2} < p < 1$. Let θ such that $(1-p) \cdot n = n - \mu(\theta+1) - \frac{l}{2}$. Then*

$$\Pr\left[\, R_{l,k} \leqslant (1-p)n \,\right] \leqslant 2 \exp\left(-\frac{\theta^2 \mu^2 (n - \frac{1}{2})}{n^2 - \mu^2}\right).$$

Proof. First, we state a simple but interesting result:

Lemma C.1.7. *Let U be the number of bins that contain exactly one ball. Then*

$$U/2 \geqslant n - E - \frac{l}{2}.$$

Proof. If there are E bins empty, and U bins with one ball only, then the remaining $l-U$ balls are contained in at most $(l-U)/2$ bins, and so

$$E + U + (l-U)/2 \geqslant \sum_i \text{number of bins containing i balls} = n.$$

and thus, $U/2 \geqslant n - E - \frac{l}{2}$. □

From there using once again lemma 2.2.4

$$\Pr\left[\, E \geqslant \mu(\theta+1) \,\right] \leqslant 2\exp\left(-\frac{\theta^2\mu^2(n-\frac{1}{2})}{n^2-\mu^2}\right)$$

$$\Pr\left[\, n - E - \frac{l}{2} \leqslant n - \mu(\theta+1) - \frac{l}{2} \,\right] \leqslant 2\exp\left(-\frac{\theta^2\mu^2(n-\frac{1}{2})}{n^2-\mu^2}\right).$$

We obtain

$$\Pr\left[\, U/2 \leqslant n - \mu(\theta+1) - \frac{l}{2} \,\right] \leqslant 2\exp\left(-\frac{\theta^2\mu^2(n-\frac{1}{2})}{n^2-\mu^2}\right),$$

because $n - E - l/2 \leqslant U/2$. Finally from $U/2 \leqslant R_{l,k}$ we get the desired bound

$$\Pr\left[\, R_{l,k} \leqslant n - \mu(\theta+1) - \frac{l}{2} \,\right] \leqslant 2\exp\left(-\frac{\theta^2\mu^2(n-\frac{1}{2})}{n^2-\mu^2}\right)$$

$$\Pr\left[\, R_{l,k} \leqslant (1-p)n \,\right] \leqslant 2\exp\left(-\frac{\theta^2\mu^2(n-\frac{1}{2})}{n^2-\mu^2}\right),$$

by choosing θ such that $(1-p)\cdot n = n - \mu(\theta+1) - \frac{l}{2}$. This proves the second bound we need. □

Now we have upper bounds for $\Pr\left[\, R_{l,k} \geqslant pn \,\right]$ and $\Pr\left[\, R_{l,k} \leqslant (1-p)n \,\right]$ for $1/2 < p < 1$. It remains to carefully choose *when* to start applying these bounds, i.e., when these bounds become better than the naive $\Pr\left[\, R_{l,k} \geqslant pn \,\right] + \Pr\left[\, R_{l,k} \leqslant (1-p)n \,\right] \leqslant 1$. Thus we want to find the lowest n for which there exists a proportion $1/2 \leqslant p \leqslant 1$ such that the following equation stands

$$\Pr\left[\, R_{l,k} \geqslant pn \,\right] + \Pr\left[\, R_{l,k} \leqslant (1-p)n \,\right] < 1. \tag{C.1}$$

Remark that we have the same formula for both bounds, but with different values for θ. Let us call them θ_1 and θ_2. For the first one (lemma C.1.5) we we should have θ_1 such that $(1-p)\cdot n = \mu\cdot(1-\theta_1)$. For the second one (lemma C.1.6) should have a θ_2 such that $(1-p)\cdot n = n - \mu(\theta_2+1) - \frac{l}{2}$. We remark that $\theta_1 > \theta_2$, thus the value of n we are looking for is coming the second inequality. The smallest n such that there exists $1/2 < p < 1$ verifying $\Pr[\, R_{l,k} \leqslant (1-p)n \,] < 1$ is $n = 2^7$. Then, using Table C.1 we can conclude that $n = 2^7$ is enough to find a proportion p, as we wanted.
From these calculations we can also show that $\beta < 0.47$, when $n > 2^7$. □

At this stage, we have shown that $\mathbb{E}[\, \phi(X_{1,1}, \ldots, X_{l,w}) \,] \geqslant 2^i \cdot w \cdot (1/2 - \beta)$, for some positive constant $\beta < 0.5$. Therefore,

$$\Pr\left[\, \phi(X_{1,1}, \ldots, X_{l,w}) \leqslant 2^i \cdot w \cdot (1/2-\beta) - t \,\right] \leqslant \exp\left(-\frac{t^2}{2lw}\right).$$

Table C.1 – Rounded value of $2\exp\left(-\frac{\theta^2\mu^2(n-\frac{1}{2})}{n^2-\mu^2}\right)$; for the two different θ, with $n = 128$ and $l = n$.

$1-p$	0.92	0.94	0.96	0.98
$\theta = \theta_1$	5×10^{-5}	2×10^{-6}	3×10^{-7}	4×10^{-8}
$\theta = \theta_2$	1.31	0.90	0.55	0.3

Let us define $\zeta = t/(2^i \cdot w)$. Then, the condition $2^i \cdot w \cdot (\frac{1}{2} - \beta) - t = 2^i \cdot w \cdot \gamma$ rewrites to $\gamma = (1/2 - \beta) - \zeta = 1/2 - \alpha(1/2 - \delta)$. Let us pick a concrete choice of value for $n \geqslant 2^7$: set $\alpha = 48/49$, $\delta = 1/100$. This gives us a value for $\gamma = 0.02$. As soon as $n \geqslant 2^7$, we know that $\beta \leqslant 0.47$, and thus $\zeta \geqslant 0.01$. Hence, there exists a constant C such that

$$\Pr\left[\, B_{\delta,\mathbf{v}} \geqslant \alpha \cdot w \,\right] \leqslant 2^{-C \cdot 2^i \cdot w}.$$

This concludes the proof of Lemma C.1.2. □

The end of the proof is the same as in the original proof, up to handling separately the case of small i's. The total number of vectors $\mathbf{v} \in S_{i,N}$ can be bounded by

$$\sum_{l=2^{i-1}}^{2^i} \binom{N}{l} \leqslant (2^i - 2^{i-1}) \cdot \frac{N^{2^i}}{(2^{i-1})!} \leqslant 2^{D \cdot 2^i}.$$

Hence, choosing constant such that $Cw/2 > D$, and setting $a = C/2$, by a union bound, we have

$$\Pr\left[\, \exists \mathbf{V} \in S_{i,N}, B_{\delta,\mathbf{v}} \geqslant \alpha \cdot w \,\right] \leqslant 2^{D \cdot 2^i} \cdot 2^{-C \cdot 2^i \cdot w} \leqslant 2^{-a \cdot w}.$$

We eventually use a union bound again on all values of $i \leqslant D$:

$$\Pr\left[\, \exists i \leqslant D, \mathbf{v} \in S_{i,N}, B_{\delta,\mathbf{v}} \geqslant \alpha \cdot w \,\right] \leqslant D \cdot 2^{-a \cdot w},$$

which, using Lemma C.1.1, rewrites to

$$\Pr\left[\; existsi \;\leqslant\; D, \mathbf{v} \;\in\; S_{i,N}, \mathsf{bias}_{\mathbf{v}}\left(\bigoplus_{j=1}^{w} D_j\right) \;\geqslant\; \frac{1}{2} \cdot ((1 - 2\delta)^{(1-\alpha)})^w \;\right] \;\leqslant\; D \cdot 2^{-a \cdot w}.$$

C.2 Handling the Corner Cases

In the proof, we have made two assumptions: $l \in [2^{i-1}, 2^i]$, and $n = 2^i \geqslant 2^7$ (recall that l is the Hamming weight of the test vector). Therefore, there is some corner cases that are not covered by the proof, the cases when the adversary attempts an attack with a vector of hamming weight $l \in [1, 63]$.

C.2.1 Case 1: l is odd.

We focus on the first submatrix of our matrix $\mathbf{H}$. The matrix has the following shape:

$$\mathbf{H}_1 = \begin{bmatrix} & & \overbrace{\phantom{\mathbf{u}^1_{1,w}}}^{\text{2 columns}} \\ \mathbf{u}^1_{1,1} & \cdots & \mathbf{u}^1_{1,w} \\ \vdots & \vdots & \vdots \\ \mathbf{u}^1_{N,1} & \cdots & \mathbf{u}^1_{N,w} \end{bmatrix},$$

where $(\mathbf{u^1_{k,j}})_{1\leqslant k\leqslant N, 1\leqslant j\leqslant w}$ are unit vectors over $\mathbb{F}_2^2$. Each row uniquely corresponds to a uniform w-bit vector (we say that it *encodes* this vector), where each bit indicates the position of the 1 in the 2-bit vector $\mathbf{u^1_{k,j}}$ (0 being left, and 1 being right). We denote the w-bit string encoded by the i-th row by x_i. Recall that $\mathbf{e_1}$ is distributed as a row of $\mathbf{H}_1$. We let K denote the string obtained by flipping all bits in the w-bit string that encodes $\mathbf{e_1}$. Observe that the inner product between the i-th row of $\mathbf{H}_1$ and $\mathbf{e_1}$ is equal to $\bigoplus_{j=1}^{w}(x_{i,j} \oplus K_j)$. Given $\mathbf{v}$, the vector $\mathbf{v}\cdot\mathbf{H}_1$ is the XOR of the l rows of $\mathbf{H}_1$ corresponding to nonzero entries in $\mathbf{v}$. Therefore, whenever l is odd, the value $\mathbf{v}\cdot\mathbf{H}_1\cdot\mathbf{e_1}$ is of the form $f(\mathbf{H}_1)\oplus(\bigoplus_{j=1}^{w}K_j)$, where f is some appropriate function. That is, value of $\mathbf{v}\cdot\mathbf{H}_1\cdot\mathbf{e_1}$ is masked by a uniformly random bit equal to $(\bigoplus_{j=1}^{w}K_j)$: it is therefore perfectly unbiased.

C.2.2 Case 2: l is even.

Let now focus on the second submatrix $\mathbf{H}_2$ of our matrix $\mathbf{H}$. It has the following shape:

$$\mathbf{H}_2 = \begin{bmatrix} & & \overbrace{\phantom{\mathbf{u^2_{1,1}}\quad\cdots\quad\mathbf{u^2_{1,w}}}}^{\text{4 columns}} \\ \mathbf{u^2_{1,1}} & \cdots & \mathbf{u^2_{1,w}} \\ \vdots & \vdots & \vdots \\ \mathbf{u^2_{N,1}} & \cdots & \mathbf{u^2_{N,w}} \end{bmatrix},$$

where $(\mathbf{u^2_{k,j}})_{1\leqslant k\leqslant N, 1\leqslant j\leqslant w}$ are unit vector over $\mathbb{F}_2^4$. This time, each row can be seen as encoding a *pair* of w-bits vector: the two bits at position i in both vectors encode together the position of the 1 in the unit length-four vector $\mathbf{u^2_{k,j}}$, in binary. For the i-th line, let us denote these vectors as $\mathbf{x_{i,0}}$ and $\mathbf{x_{i,1}}$. For the noise vector $\mathbf{e_1}$, who has the same structure, we also define two vectors $\mathbf{K_0}$ and $\mathbf{K_1}$, as before by flipping all bits of the two vectors encoded by $\mathbf{e_2}$. Now, given a test vector $\mathbf{v}$ with even Hamming weight l, the value $\mathbf{v}\cdot\mathbf{H}_2\cdot\mathbf{e_2}$ is equal to $\bigoplus_{i\in S}(\mathbf{x_{i,0}}\oplus\mathbf{K_0})\cdot(\mathbf{x_{i,1}}\oplus\mathbf{K_1})$, where S is the subset of nonzero entries in $\mathbf{v}$. Let $g(\mathbf{x_0},\mathbf{x_1}) = \bigoplus_{j=1}^{w} x_{i,0,j}\cdot x_{i,1,j}$. With this notation, we can rewrite $\mathbf{v}\cdot\mathbf{H}_2\cdot\mathbf{e_2}$ as

$$\bigoplus_{i\in S} g(\mathbf{x_{i,0}},\mathbf{x_{i,1}}) \oplus \left\langle \bigoplus_{i\in S}\mathbf{x_{i,0}}, \mathbf{K_1}\right\rangle \oplus \left\langle \bigoplus_{i\in S}\mathbf{x_{i,1}}, \mathbf{K_0}\right\rangle.$$

The leftmost term is independent of the secret noise vector. The above equation implies that if $\bigoplus_{i\in S}\mathbf{x_{i,0}}$ is not the all zero vector, then the above value is additively masked by one of the entries in $\mathbf{K_1}$, which is a uniformly random bit; hence, it is uniformly distributed (the above is a sufficient condition; the same condition with respect to $\mathbf{x_{i,1}}$ and $\mathbf{K_0}$ would also suffice). Furthermore, we can bound the probability that $\bigoplus_{i\in S}\mathbf{x_{i,0}} = \mathbf{0}$: let us call E_0 this event. For any fixed choice of the size-l subset S, the probability that $\bigoplus_{i\in S}\mathbf{x_{i,0}} = \mathbf{0}$ holds over a random choice of the vectors $\mathbf{x_{i,0}}$ is exactly 2^{-w}. By a straightforward union bound over all subsets S of size l,

$$\Pr[E_0] \leqslant 2^{-w}\cdot\binom{N}{l} \leqslant 2^{-w}\cdot N^l \leqslant 2^{l\cdot D - w},$$

where $N = 2^D$ is the number of rows in $\mathbf{H}$. Whenever l is a constant (recall that we assume here $l\leqslant 63$), the above probability is bounded by $2^{-\alpha_0 w}$ as soon as $w > \alpha_1 D$ for an appropriate choice of the constants α_0, α_1 (a quick calculation shows that these constants are *much* better than the ones involved in the case of large l, hence the final constants involved in our theorem remain the same as the constants achieved in the previous proof). This concludes the analysis of the corner cases.

Appendix D

Script for the optimization of β

Hereinafter the script written in python used to obtain simulated value of β. The number of simulations is chosen in order to reach a 99% confidence interval.

```
import math
import random
import statistics

n = 2048 # Number of bins
l = 1024# Number of balls
T = 100000# Number of simulations
List_Z = []

for j in range(T):
  #Repeat for each simulation
  Bins = [0 for k in range(n)]
  count_odd = 0
  #We throw the l balls
  for k in range(l):

    #Choice of the bin among the n
    r = random.randint(0,n-1)
    count_odd += (1-2*Bins[r])
    Bins[r] = (Bins[r] +1) % 2
  List_Z.append(abs(1/2-count_odd/n))
  #Z = n/2 - E[R], and R = count_odd/n.
mean_Z = statistics.mean(List_Z)
# To determine a confidence interval
stdev_Z = statistics.stdev(List_Z)


print("The␣confidence␣interval␣at␣99%␣is
:␣[{}␣-␣{},␣{}␣+␣{}]".format(
mean_Z,3*stdev_Z/math.sqrt(T),
mean_Z,3*stdev_Z/math.sqrt(T)))
```

Appendix E

Script for the optimization of i^* and w_i

Hereinafter the script written in python used to obtain the couple (i^*, w_i) reaching the best key size and number of PCF evaluations per second.

```
import math
import random
import statistics

def getBeta(betaList,i):
    #Return the element at index i-2 ;
    # or the last item if i is superior than the size of the list.
    if i-1> len(betaList):
    return betaList[len(betaList)-1]
    else:
    return betaList[i-2]

def H(x):
    #Binary entropy function
    return -x *math.log(x,2) - (1-x) *math.log(1-x,2)

def firstBound(D,level,i,l,precision,basis):
    #Return the closes lower bound on w * zeta^2. Precision is the number
    #of digit after the decimal point +1
    for wzetacarre in range(1*10**(precision),5000000*10**(precision)):
    #Recursion in order to find the wzetacarre
    c = (math.log(basis)*D*l-(((basis**(2*i-1))/l)*
    (wzetacarre/(10**(precision+1)))))/math.log(2)
    if (c<level): # first w * zeta^2 is bellow the requirement level.
        return wzetacarre/10**(precision+1)
    return wzetacarre/10**(precision+1)

def secondBound(beta,wzcarre,level,precision):
    # Return the value zeta. The precision level indicate the number of digit after
    #the decimal point +2
    for zeta in range(50*10**(precision),1*10**(precision),-1):
    c = math.log(2*(beta+(zeta/10**(precision+2))))*
    (wzcarre/(zeta/10**(precision+2))**2)/math.log(2)
    if (c<level):# First zeta bellow the requirement level
        return (zeta/10**(precision+2))
    return (zeta/10**(precision+2))

def findw(D,level1,level2,istar,precision,beta,b):
    #Finding the variable w with :
    # D : number of blocs
    # Level1 and level 2: Required security levels of the probability and of the bias
    # istar : blocs we are on
    # precision : level of precision for w zeta^2 and zeta,
    #in order to be close to the true w.
```

```
    # beta: value of beta in the case l = b^istar
    # number of samples such that b^D.
    wzetacarre = firstBound(D,level1,istar,b**istar,precision,b)
    zeta = secondBound(beta,wzetacarre,level2,precision)
    return math.ceil(wzetacarre/zeta**2) # equal to w.

def sizeR(istar,D,b):
    # Function computing the size of the bloc R
    #replacing the istar-1 first blocs among D blocs ;
    #for a number of samples equal to b^D.
    d = (b)**(istar-1)-1
    return round(H(d/(b)**D) * (b)**D + 128)

def tailleseed(tr,istar,D,lbd,b,betaList,level1,level2,precision):
    # Computation of the size of the seed for construction with :
    # tr : size of the bloc R replacing the istar-1 first blocs.
    # D : number of blocs
    # Level1 and level 2: Required security levels of the probability and of the bias
    # istar : blocs we are on
    # precision : level of precision for w zeta^2 and zeta,
    # in order to be close to the true w.
    # beta: value of beta in the case l = b^istar
    # number of samples such that b^D.
    #lbd : security parameters of DPF (=128).
    # betaList: list of beta depending on the size of the blocs.
    c = 0
    for i in range(istar,D+1):
    beta = getBeta(betaList,i)
    c += findw(D,level1,level2,i,precision,beta,b) * (i*(lbd + 2) + 2 *lbd)
    # size of the istar... D blocs.
    return round((c + tr)/8000000,2) # transformation in MB

def computationCost(tr,D,istar,b,betaiList,level1,level2,precision):
    # Computation of the size of the seed for construction with :
    # tr : size of the bloc R replacing the istar-1 first blocs.
    # D : number of blocs
    # Level1 and level 2: Required security levels of the probability and of the bias
    # istar : blocs we are on
    # precision : level of precision for w zeta^2 and zeta,
    # in order to be close to the true w.
    # beta: value of beta in the case l = b^istar
    # number of samples such that b^D.
    # betaList: list of beta depending on the size of the blocs.
    c = 0
    for i in range(istar,D+1):
    beta = getBeta(betaiList,i)
    c += findw(D,level1,level2,i,precision,beta,b)  * (i+1)
    cycles1 = 2*c*16*1.3  # number of cycle for the computation of
    cycle2 = tr/8*1.3
    # number of cycle for the computation of the scalar product of size r
    pcfEvalperS = round(3800000000/(cycles1+cycle2))
    #transformation in number of PCF evaluation per second.
    return pcfEvalperS




    T = 100 # number of repetition for the computation of the values beta
    #- faster if T is smaller but the result might be a tiny bit different
    level1 = -80 # Security requirements concerning the probability.
    level2 = -80 # Security requirements concerning the bias.
    precision = 2 # precision : level of precision for w zeta^2
    #and zeta, in order to be close to the true w.
    D1 = 30   # Original value of D = 30 ; because we wanted 2^30 samples.
```

```
    for e in [3]: #[1,2,3,5,6,10]:
    b = 2**e # 2^e for size of a subblocs.
    D = math.ceil(math.log(2)*D1/math.log(b))
    #Computation of the new number of blocs required,
    # such that the number of samples generated is 2^30 = b^D.
    betaList = []
    for istar in range(2,D//2):
    # We compute the values of beta depending on the blocs we are in.
    #We do not go until the end because it would take
    # to much time and stop when beta converges.
        n = b**istar
        l = n
        List_Z = []
        for j in range(T):
        #Repeat for each simulation
        Bins = [0 for k in range(n)]
        count_odd = 0
        #We throw the l balls
        for k in range(l):
            #Choice of the bin among the n
            r = random.randint(0,n-1)
            count_odd += (1-2*Bins[r])
            Bins[r] = (Bins[r] +1) % 2
        List_Z.append(abs(1/2-count_odd/n))
        #Z = n/2 - E[R], and R = count_odd/n.
        beta = statistics.mean(List_Z)
        betaList.append(beta)
    for istar in range(2,D+1): # For all istar in between 2 and D
    # we compute and display the seed size and the number of PCF evaluation per second.
        tr = sizeR(istar,D,b)
        print("D = {}, b = {}, beta used = {}, target level1 = {}, size of R = {}".format(
            D,b,getBeta(betaList,istar),level1-math.log((D-istar+1)*(b**(istar-1)),2),tr))
        print("")
        print("i^* = {}, w(i^*) = {}, seed size = {},
        PCF evaluations per second = {}".format(
        istar, findw(D,level1-math.log((D-istar+1)*(b**(istar-1)),2),
        level2,istar,3,getBeta(betaList,istar),b),
        tailleseed(tr,istar,D,128,b,betaList,level1,level2,precision),
        computationCost(tr,D,istar,b,betaList,level1,level2,precision)))
        print(" ")
        print(" ")
```