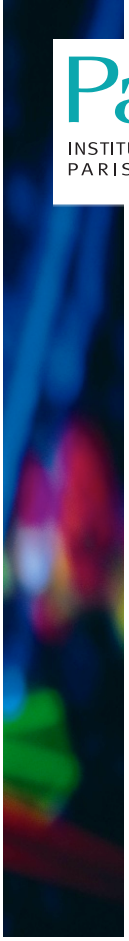
Protecting data confidentiality combining data fragmentation, encryption, and dispersal over a distributed environment.

Katarzyna Kapusta



ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY

TELECOM ParisTech

2018-ENST-0061

EDITE - ED 130

Doctorat ParisTech

T H È S E

pour obtenir le grade de docteur délivré par

TELECOM ParisTech

Spécialité « Informatique et Réseaux »

Katarzyna Kapusta

Soutenue le 20 Décembre 2018

Protecting data confidentiality combining data fragmentation, encryption, and dispersal over a distributed environment.

Directeur de la thèse : Prof. **Gérard MEMMI**

Jury

M. Frédéric CUPPENS, Prof., Télécom Bretagne	Rapporteur
M. Roberto DI PIETRO, Prof., Hamad Bin Khalifa University	Rapporteur
M. Louis GOUBIN, Prof.,University of Versailles-St-Quentin-en-Yvelines	Rapporteur
Mme Samia BOUZEFRANE, Dr., Conservatoire National des Arts et Métiers	Examinateur
Mme Nora CUPPENS, Prof., Télécom Bretagne	Examinateur
M. Artur JANICKI, Dr., Warsaw University of Technology	Examinateur
M. Jean LENEUTRE, Dr., Télécom ParisTech	Examinateur

T H È S E

TELECOM ParisTech
école de l'Institut Mines-Télécom - membre de ParisTech
46 rue Barrault 75013 Paris - (+33) 1 45 81 77 77 - www.telecom-paristech.fr

Contents

Acknowledgements

Firstly, I would like to express my most sincere gratitude to my supervisor Prof. Gérard Memmi for the tremendous support of my PhD study and research, for his patience, motivation, immense knowledge, and dedication to students. Thank you for leading me through such a splendid journey. With your leading, the past four years have been a fantastic adventure that will be unique and important in my life.

I am grateful to all the jury members: Prof. Frédéric Cuppens, Prof. Roberto di Pietro, Prof. Louis Goubin, Dr Samia Bouzefrane, Prof. Nora Cuppens, Dr Artur Janicki, and Dr Jean Leneutre for attending my dissertation, and, in particular, the reviewers of the manuscript: Prof. Frédéric Cuppens, Prof. Roberto di Pietro, Prof. Louis Goubin, for their important suggestions and comments.

Thanks also to all the colleagues and staffs in Télécom ParisTech for all help and collaboration in technical knowledge and administrative tasks.

Last but not least, I would like to express my gratitude to my family for encouraging and supporting me so many years. Thanks to my friends I met in France for the memorable moments we shared.

Abstract

Combining data encryption, fragmentation, and dispersal is a proven but not widely used way of reinforcing data confidentiality, availability, and integrity in distributed storage environments. Nowadays, it is being reconsidered with the emergence of the cloud storage and the growing interest in internet-of-things. Multiple questions arise within these two new contexts: How to guarantee data protection even in a situation when the attacker acquired the encryption key? How to protect outsourced data against a curious storage provider and malicious attackers? How to provide data confidentiality in a sensor network where real-time data collection is not possible?

This thesis dissertation revisits state-of-the-art fragmentation techniques making them faster and cost-efficient. The main focus is put on increasing data confidentiality without deteriorating the processing performance. The ultimate goal is to provide a user with a set of fast fragmentation methods that could be directly applied inside an industrial context to reinforce the confidentiality of the stored data and/or accelerate the fragmentation processing.

First, a rich survey on fragmentation as a way of preserving data confidentiality is presented. It introduces two new definitions dividing frag-

mentation into bitwise and structurewise. Relevant techniques are described in details and compared in terms of data protection level, storage overhead, and complexity. Performance benchmarks confirm the theoretical complexity evaluation. Not only techniques, but also relevant academic and commercial systems are presented. Their main characteristics are described. The academic and commercial approaches are compared. Several recommendations are given on the design of an efficient fragmentation system.

Second, the family all-or-nothing transforms is extended with three new proposals. They all aim at protecting encrypted and fragmented data against the exposure of the encryption key but are designed to be employed in three different contexts: for data fragmentation in a multi-cloud environment, a distributed storage system, and an environment composed of one storage provider and one private device. Complexity evaluation and performance benchmark show that they are the fastest of all of the relevant techniques.

Third, a way of accelerating fragmentation is presented that achieves better performance than data encryption using the most common symmetric-key encryption algorithm. This gain in performance is achieved by limiting the encryption processing, applying an all-or-nothing transform over partially encrypted data, and relying on the protection provided by data dispersal. This contribution addresses the need of users disposing of access to independent storage sites and prioritizing the speed of processing.

Fourth, a lightweight fragmentation scheme based on data encoding, permuting, and dispersing is introduced. It totally gets rid of data encryption allowing the fragmentation to be performed even faster; up to twice as fast as data encryption. The proposal revisits the use of the perfect secret sharing,

trading its security for scalability. In contrast to other contributions, this proposal comes with an alternative to symmetric encryption. It was motivated by the fact that in some cases a lightweight data protection combined with dispersal may be sufficient.

Finally, fragmentation inside sensor networks is revisited, particularly in the Unattended Wireless Sensor Networks. The main focus in this case is put not solely on the fragmentation performance, but also on the reduction of storage and transmission costs by using the data aggregation. This is motivated by the fact that contrary to fragmentation in the cloud, fragmentation inside a sensor networks has to take into account the limited energy capacities of the sensors' batteries. When compared with relevant state-of-the art techniques, the proposed scheme reduces by at least half the number of stored and transmitted bits.

Résumé

Dans cette thèse de doctorat, les méthodes classiques de protection de données sont revisitées au vu de l'émergence du cloud public d'une part et de l'intérêt croissant pour l'Internet des Objets (IoT) d'autre part. Plusieurs questions se posent dans ces deux nouveaux contextes. Comment garantir la protection des données même dans le cas où l'attaquant a acquis la clé de chiffrement? Comment protéger ses données externalisées contre non seulement des attaquants malveillants, mais aussi contre un fournisseur de stockage curieux qui voudrait utiliser les données pour ses propres intérêts ou applications? Comment assurer la confidentialité des données dans un réseau constitué de capteurs pour lequel la collecte de données en temps réel n'est pas possible?

La combinaison du chiffrement, de la fragmentation et de la dispersion des données constitue un moyen éprouvé mais peu répandu de renforcer la confidentialité, l'intégrité et la disponibilité des données dans des environnements de stockage distribués. Pendant longtemps, la non-popularité de cette solution a essentiellement été liée à deux causes. Premièrement, avant l'introduction du cloud public, l'utilisation de la fragmentation était limité aux centres de données privés. Hors, la fragmentation n'est efficace en terme

de protection que par l'usage de plusieurs nœuds de stockage indépendants. Aujourd'hui, l'architecture distribué du cloud et de l'Internet des Objets ouvrent de nouvelles possibilités de dispersion: un simple utilisateur peut se permettre de disperser ses données sur plusieurs serveurs localisés dans le monde entier. Deuxièmement, les techniques de fragmentation classiques détériorent notablement les performances du système et souvent augmentent le volume de la donnée stockée de façon importante.

La thèse a commencé dans le cadre du projet européen ITEA2 CAP. Elle se concentre sur l'étude des techniques de fragmentation et de dispersion en se proposant de les rendre plus rapides et/ou moins gourmandes en mémoire. Le but ultime est de fournir à un utilisateur un ensemble de méthodes de fragmentation rapide pouvant être directement appliqué dans un contexte industriel afin de renforcer la confidentialité des données stockées ou d'accélérer le processus de protection.

Sur la fragmentation comme moyen de protection de données

Une étude approfondie sur la fragmentation en tant que moyen de préserver la confidentialité des données a été effectuée et a donné comme résultat une vaste analyse et organisation de l'état de l'art. Cet état d'art commence par la description du processus de fragmentation. Par fragmentation de données on comprend le processus de transformation quelconque d'une donnée en un ensemble de fragments comprenant un seuil c'est à dire un nombre minimal de fragments nécessaires pour la reconstruction de la donnée initiale.

Ainsi définie, la fragmentation inclut les méthodes ayant pour but d'assurer la résilience de la donnée (utilisant la réplication ou les codes correcteurs d'erreurs). Ensuite, les descriptions des travaux pertinents sont organisés en discernant les méthodes de fragmentation qui utilisent ou non des informations sur la structure de donnée. Dans le premier cas, les données sont divisées en sous-ensembles de bits de différents niveaux de confidentialité qui seront protégés de différentes manières, c'est-à-dire dispersés sur des serveurs physiques caractérisés par différents niveaux de fiabilité. Ce type de fragmentation sera surtout appliqué pour fragmenter des bases de données. Dans le second cas, les données sont simplement traitées comme un ensemble de bits consécutifs, chacun ayant un niveau de confidentialité égal. Ce type de fragmentation peut être appliqué sur tous types de données.

Les techniques de fragmentation sans considération de la structure de données pertinentes sont décrites en détail et comparées en termes de niveau de protection des données, de surcharge de stockage et de complexité. Des évaluations expérimentales ont confirmé les estimations théoriques. Parmi les techniques présentées on retrouve le partage de secret (notamment le schéma de partage de secret proposé par Shamir), les algorithmes de dispersion d'information (notamment celui de Rabin), ainsi que les schémas de fragmentation basé sur le chiffrement symétrique (entre autres la modification du partage de secret de Krawczyk, les méthodes de type "tout-ou-rien" et la technique AONT-RS). On peut observer que même si le partage de secret garantit un haut degré de confidentialité de données, il reste peu pratique pour la fragmentation de grandes volumes de données puisqu'il mène à une augmentation considérable de la taille de la donnée fragmentée. Les méth-

odes basées sur le chiffrement permettent de limiter la surcharge de mémoire en garantissant un niveau de confidentialité suffisant contre un attaquant limité en terme de puissance de calcul. De plus, contrairement aux algorithmes de dispersions, ils passe à l'échelle en terme de nombre de fragments.

Non seulement les techniques, mais aussi six systèmes académiques appliquant la fragmentation sans regard de la structure des données (Delta-4 (1991), PASIS (2000), GridSharing (2005), POTSHARDS (2009), DepSky (2013) et CDStore (2016)) et trois systèmes commerciaux (Unisys's SecureParser (2005), IBM Cloud Storage (précédemment Cleversafe, 2011) et Symform (2011)) ont été analysés et comparés. L'accent est mis sur la façon de renforcer la confidentialité des données utilisée par ces systèmes. Cependant, d'autres caractéristiques propre à un système basé sur la fragmentation de donnée sont également décrites, comme la résilience des données, la gestion de placement des clés et des fragments, l'intégrité et l'authentification des données, la fiabilité des nœuds de stockage, la taille des fragments, ainsi que la déduplication des fragments. On peut observer que l'utilisation de la fragmentation a changé au cours des années et diffère selon l'origine du projet. Il y a encore dix ans, les solutions reposaient presque exclusivement sur le partage de secrets pour la confidentialité et la réplication pour la résilience des données, tandis que les systèmes récents remplacent le partage de secret par le chiffrement et la réplication par les codes correcteurs d'erreurs. A la différence des produits commerciaux, les propositions académiques ont tendance à être plus originales, même au prix d'une augmentation excessive du volume des données stockées et d'une diminution drastique des performances.

Les techniques de fragmentation exploitant la structure de données sont

divisés en deux groupes. La première groupe réunit les travaux autour de la fragmentation orientée objets: une approche de conception de logiciel décomposant l'architecture d'une application en plusieurs fragments de différents niveaux de confidentialité et favorisant le placement des objets confidentiels sur des machines de confiance. Le deuxième groupe applique la fragmentation au base de données relationnelles. Cet approche est majoritairement réalisée dans un environnement de type multi-cloud. Les n-uplets de la base de données sont segmentés en plusieurs fragments pour séparer les valeurs non-confidentielles qui, une fois associées, pourraient dévoiler de l'information confidentielle, notamment qui dé-anonymiseraient l'information stockée dans la base. Les valeurs contenant de l'information confidentielle sont ensuite chiffrées.

Pour conclure, quelques recommandations sont données sur la conception d'un système de fragmentation efficace. On recommande notamment la façon de disperser la données selon son niveau de confidentialité et selon son volume (voir Figure 1).

Protection des données chiffrées contre l'exposition de la clé de chiffrement

L'externalisation du stockage de données dans le cloud constitue sans doute un défi en terme de sécurité. D'une part, les fournisseurs doivent faire face quotidiennement à un grand nombre d'attaques externes. Chaque fuite de données majeure risque d'être signalée dans les médias en nuisant à leur réputation. D'autre part, du point de vu d'un utilisateur du service, la menace

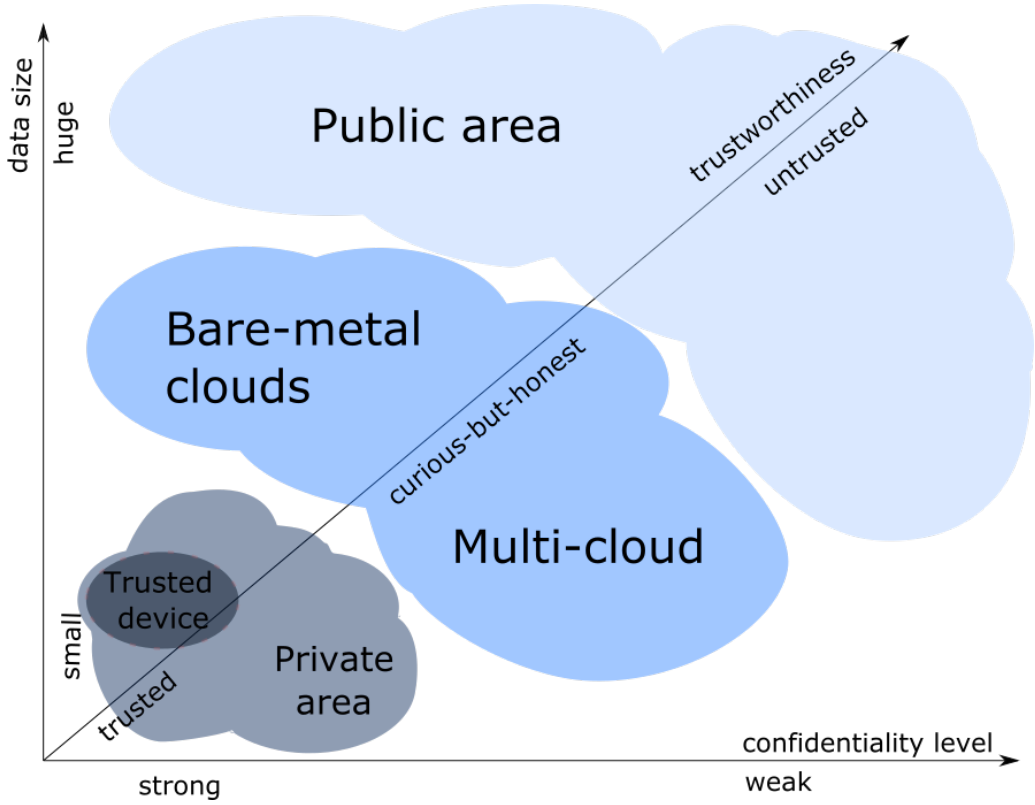


Figure 1: *Dispersion des données selon leurs niveaux de confidentialité.*

peut venir non seulement d'un attaquant externe, mais aussi du fournisseur lui même. En effet, un fournisseur malhonnête pourrait être tenté d'exploiter les données stockées, par exemple pour des raisons économiques ou politiques ou même par pure malveillance. Ainsi, la solution recommandée (entre autres par le Règlement européen Général sur la Protection des Données (RGPD)) est de chiffrer les données avant de les confier à un service de stockage.

D'un autre point de vu, grâce à sa nature hautement distribuée, le cloud ouvre de nouvelles possibilités pour renforcer la protection des données qui, en plus d'être protégées par le chiffrement, peuvent être maintenant facile-

ment et efficacement dispersées sur un grand nombre de serveurs situés sur des sites de stockage différents. Un tel traitement non seulement ralentit un attaquant externe (qui doit accéder à plusieurs sites de stockage protégés par des mécanismes de sécurité différents), mais protège également contre une utilisation abusive des données stockées par un seul fournisseur (qui ne possédera pas la données complète sans une coalition avec *tous* les fournisseurs).

Le renforcement de la protection des données en utilisant une combinaison de chiffrement, de fragmentation et de dispersion a déjà été proposé par plusieurs solutions de stockage durant les dernières années. Cependant, la majorité d'entre eux ne prêtent pas beaucoup d'attention à la manière dont les fragments sont construits à partir de la donnée chiffrée. La fragmentation est généralement effectuée de manière simple, les fragments étant formés à partir de gros morceaux consécutifs de donnée. Un tel traitement ne protège pas contre des adversaires qui, en plus d'avoir accès à un sous-ensemble des domaines de stockage, ont réussi d'obtenir la clé de chiffrement. Ce type d'adversaire sera en effet capables de déchiffrer une partie des informations dispersées.

De nos jours, l'accès par l'attaquant à la clé de chiffrement (on parlera d'exposition à la clé de chiffrement) est une menace réelle surtout pour des données de long cycle de vie ou des données partagées par de nombreux utilisateurs. D'une part, cette exposition peut être le résultat d'une mauvaise génération de clé résultant en une clé facilement concevable ou reproductible. D'autre part, elle peut être due à l'utilisation par les attaquants de portes dérobées dans les logiciels de génération de clé, ou encore due à une forme de corruption ou de coercition. De plus, lorsque nous considérons des données

ayant de longs cycles de vie, la longueur de la clé de chiffrement utilisée peut devenir insuffisante après plusieurs années en raison des progrès réalisés dans le développement de nouvelles machines toujours plus puissantes ainsi que dans le domaine de la cryptanalyse.

Afin d'empêcher un attaquant en possession de la clé de chiffrement de déchiffrer même une petite partie de l'information, la donnée sera premièrement transformée avec une méthode dite "tout-ou-rien" (*all-or-nothing*) et ensuite fragmentée en plusieurs fragments puis dispersée. La transformation "tout-ou-rien" produit un texte chiffré déchiffrable seulement s'il est complet. Ainsi, la totalité des fragments est nécessaire pour la défragmentation. Une fois que les fragments sont dispersés sur au moins deux sites de stockage indépendants, la donnée est protégée contre un attaquant incapable de collecter tous les fragments. Cela est vrai même si l'attaquant parvient à obtenir la clé de chiffrement car un déchiffrement partiel des données (déchiffrer uniquement la partie des données contenues dans un seul fragment) est impossible.

Plusieurs algorithmes de type "tout-ou-rien" ont déjà été proposés dans la littérature. Cependant, nécessitant un traitement créant des liens entre les blocs du texte chiffré, ils restent moins rapides que le chiffrement symétrique combiné avec une fragmentation simple de la donnée chiffrée. Dans le Chapitre 3 de la thèse, la famille des algorithmes de type "tout-ou-rien" se voit agrandie de trois propositions visant à atteindre un surcoût de performance optimisé voire négligeable: le "Secure Fragmentation and Dispersal" (SFD), le "Circular *all-or-nothing*" (CAON), et le "Selective *all-or-nothing*" (SAON). Ayant le même but, chacune des propositions est conçue pour être utilisée dans un contexte différent.

SFD est une méthode de fragmentation et de dispersion d'un texte obtenu en exploitant les propriétés d'un chiffrement symétrique par blocs. Elle ne nécessite aucune transformation de la donnée chiffrée, juste une simple segmentation. En effet, les bits de la donnée sont dispersés sur plusieurs fragments. Cependant, cette dispersion n'est pas faite d'une façon aléatoire mais suit des règles précises exploitant les propriétés d'un chiffrement par bloc et ayant pour but de renforcer la confidentialité des fragments. Premièrement, les blocs consécutifs du texte chiffré sont séparées entre différents fragments. Cela est motivé par le fait que le déchiffrement d'un bloc nécessite dans la plus part des cas la présence du prédécesseur de ce bloc. Ainsi, la séparation des blocs consécutifs renforce la protection et il n'est plus possible de déchiffrer un seul fragment, même si la clé de chiffrement est connue. Deuxièmement, les bits d'un seul bloc sont aussi répartis entre différents fragments qui seront dispersés sur plusieurs serveurs. Cela est motivé par le fait que le déchiffrement d'un bloc incomplet n'est pas possible. Les fragments finaux sont dispersés sur plusieurs serveurs dans plusieurs clouds. Les fragments contenant des blocs consécutifs de la donnée chiffrée sont séparés entre différent clouds préférablement indépendants. Cette forme de fragmentation et de dispersion renforce la confidentialité de la donnée en forçant un attaquant de connaitre la liste des serveurs et d'obtenir l'accès aux services de stockage utilisés. Un attaquant ayant obtenu l'accès à un seul cloud ne peut pas déchiffrer les données qu'il a obtenu même s'il possède la clé utilisée pendant le chiffrement (cela est vrai pour un fournisseur de stockage malhonnête qui voudrait déchiffrer les données qui lui ont été confiées). SFD ne nécessite aucun traitement de données spécifique juste une simple dispersion de bits qui

peut être implémentée d'un façon très rapide en rendant le surcout provenant de la dispersion négligeable. Cette technique est particulièrement adaptée aux environnements multi-clouds permettant une dispersion des fragments sur des sites de stockage indépendants.

Contrairement au SFD, CAON peut être appliqué sur tou type de chiffrement. Cet algorithme crée des dépendances entre les blocs consécutifs du texte chiffré en les xorant entre eux. Ainsi, une chaîne de dépendances entre blocs du texte chiffré est créée. Après cette transformation, la donnée est fragmentée en plusieurs fragments qui vont être ensuite dispersés sur plusieurs serveurs ou sites de stockage indépendants. Pour défragmenter la donnée initiale, il est nécessaire de rassembler la totalité ces fragments. En conséquence, un attaquant n'ayant pas obtenu tous les fragments ne peut pas procéder aux déchiffrement de la donnée et cela même s'il possède la bonne clé. En comparaison avec l'algorithme le plus rapide de l'état de l'art, la transformation utilisée par CAON réduit de moitié le nombre de XOR exigé en plus du chiffrement de la donnée. L'étude de la complexité théorique a été confirmée par des tests de performance (voir Figure 2).

SAON vise à protéger les données chiffrées contre l'exposition de la clé de chiffrement même si l'utilisateur a accès à un seul service de stockage. La donnée chiffrée est transformée en deux fragments: un grand fragment public et un petit fragment privé. Les deux fragments sont nécessaires au déchiffrement de la donnée initiale. Un utilisateur conserve le fragment privé et externalise le fragment public. Ainsi, l'utilisateur peut profiter des avantages du cloud public sans se soucier de l'exposition de la clé de chiffrement. En effet, sans le fragment privé, la donnée contenue dans le fragment public

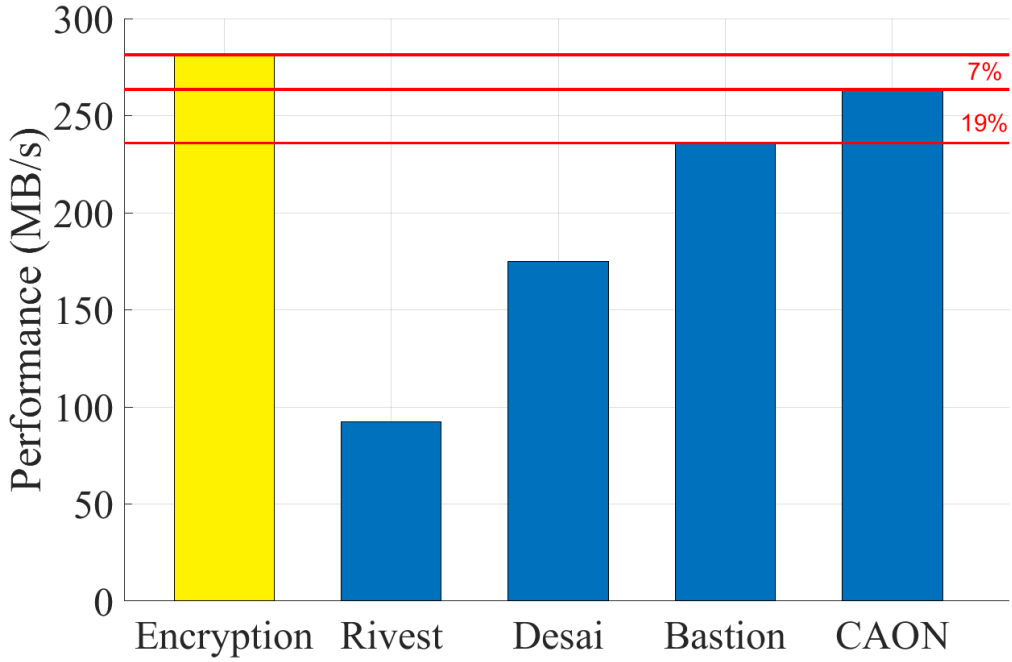


Figure 2: *Tests de performance de l'algorithme CAON.*

reste inutilisable. La transformation de la donnée en fragments est composée d'une combinaison de SFD et d'une transformation rapide de type "tout-ou-rien" appliquée seulement à une partie de la donnée. En conséquence, SAON est plus rapide qu'une transformation "tout-ou-rien" appliquée à la totalité de la donnée. Les performances dépendent de la taille du fragment privé qui est un paramètre de SAON à la main de l'utilisateur.

En conclusion, les schémas présentés dans le Chapitre 3 résolvent efficacement le problème de la protection des données chiffrées contre l'exposition de la clé de chiffrement. Chacun de ces schémas correspond à un contexte de stockage différent. Le renforcement de la confidentialité se traduit par un surcoût de performance presque négligeable. Les propositions présentées

restent les plus rapides de la famille des algorithmes de type "tout-ou-rien" publiés: cela a été démontré par une analyse de complexité et confirmé par des tests de performance (voir Figure 3).

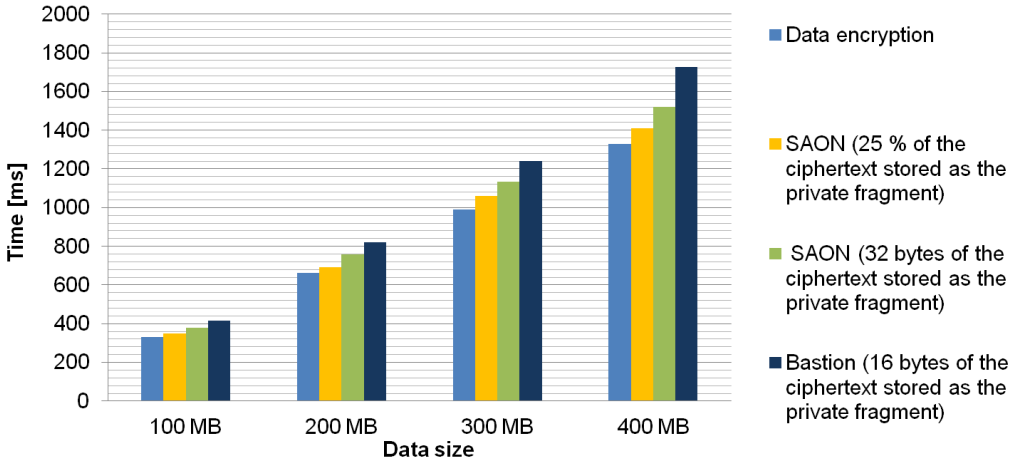


Figure 3: *Tests de performance de l'algorithme SAON.*

Accélération de la fragmentation

Le renforcement de la confidentialité des données entraîne inévitablement un surcoût en termes de performances (même s'il peut être réduit au point d'être négligeable, comme dans le cas des algorithmes présentés dans le Chapitre 2). Pour certains cas d'usage, même une petite surcharge s'avère inacceptable car la rapidité de traitement est critique. Le choix de la bonne technique de fragmentation est souvent un compromis entre les performances souhaitées, le niveau de protection des données et la surcharge de stockage. Le Chapitre 3 introduit le schéma PE-AON permettant d'accélérer le processus de fragmentation en le rendant plus rapide que le chiffrement symétrique de la don-

née combiné à une simple fragmentation. Il transforme les données en un ensemble de fragments, qui sont tous nécessaires à la reconstruction des données. Contrairement aux algorithmes présentés précédemment, seulement une partie (la quantité exacte dépend du niveau de protection souhaité) de la donnée initiale est chiffrée. Les blocs de la donnée sont ensuite mélangés à l'aide d'une transformation de type "tout-ou-rien". Les fragments finaux sont formés à partir de ce mélange. La protection des données est assurée par la dispersion. Une reconstruction complète de la donnée initiale n'est possible qu'une fois tous les fragments rassemblés.

Éviter partiellement le chiffrement améliore les performances, car une partie des opérations de chiffrement par blocs est remplacée par des XORs. Par conséquent, PE-AON est plus rapide que le chiffrement symétrique combiné à une fragmentation simple (où les données sont simplement divisées en gros morceaux de bits consécutifs). Lorsque le rapport entre le nombre de fragments chiffrés et non chiffrés est judicieusement choisi, les données contenues dans les fragments sont protégées contre l'exposition de la clé de chiffrement. Intuitivement, de meilleures performances se font au détriment d'une protection plus faible que celle fournie par le chiffrement complet combiné à une transformation "tout-ou-rien". Dans les schémas présentés dans le Chapitre 2, un attaquant est censé pouvoir compromettre tous les emplacements de stockage sauf un. En revanche, PE-AON n'est efficace que contre un attaquant plus limité, notamment résidant sur un seul site de stockage. Ainsi, PE-AON doit être traité comme une méthode de fragmentation rapide pour une protection de données légère, permettant un gain de performance considérable (voir les résultats de performance sur la Figure 4). Cela est

particulièrement utile dans les cas où l'utilisateur dispose d'un obstacle de dispersion important, par exemple, il peut séparer les données sur plusieurs clouds notoirement indépendants.

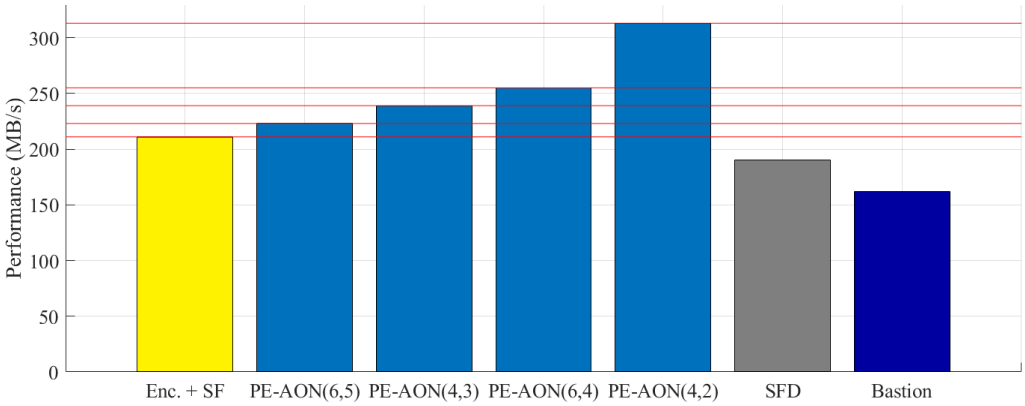


Figure 4: *Tests de performance de l'algorithme PE-AON.*

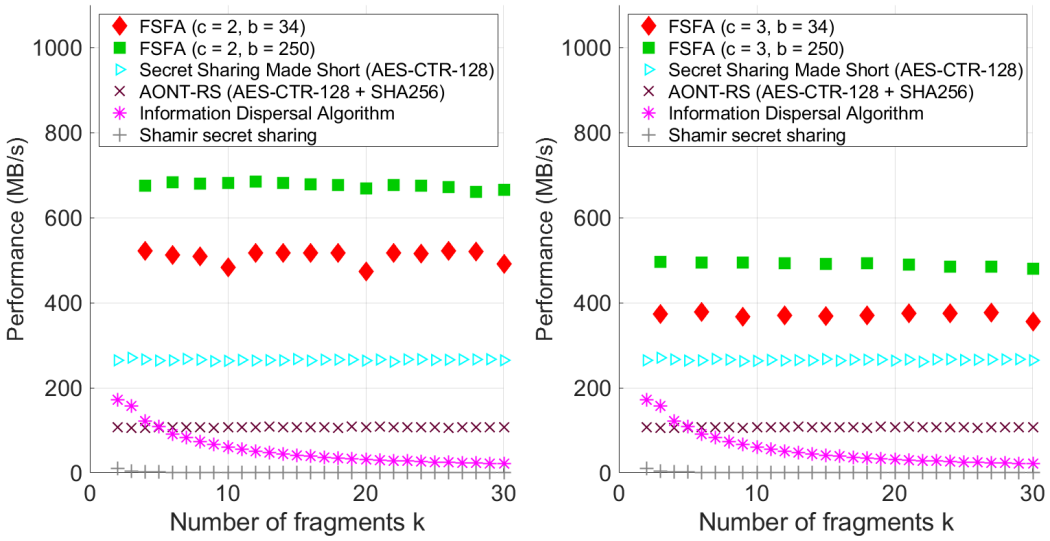
Protection des données par une combinaison de chiffrement, fragmentation et dispersion de données

La solution habituelle pour assurer la confidentialité des données externalisées consiste à chiffrer les données avant de les envoyer au fournisseur de stockage. L'utilisation d'un algorithme de chiffrement symétrique assure normalement de fortes garanties de confidentialité. Cependant, le chiffrement a toujours un coût. Grâce aux récents progrès réalisés dans le domaine du matériel, comme l'intégration du jeu d'instructions AES (AES-NI) dans de nombreux processeurs Intel, la vitesse de chiffrement doit être considérée comme une

variable qui évolue avec les progrès technologiques. Cependant, suivre cette progression est coûteuse et l'ensemble des utilisateurs doit être considéré comme étant hétérogène dans son comportement et, par conséquent, une quantité non négligeable de données n'est pas chiffrée avant d'être envoyée dans le cloud.

Le Chapitre 4 introduit le Fast and Scalable Fragmentation Algorithm (FSFA), un algorithme de fragmentation léger basé sur l'encodage, la permutation et la dispersion des données. Il élimine totalement le chiffrement des données, ce qui permet une fragmentation encore plus rapide: jusqu'à deux fois plus rapide que le chiffrement des données en utilisant l'algorithme de chiffrement le plus répandu (AES-NI) (voir les résultats présenté sur la Figure 5). Contrairement à d'autres contributions présentées dans cette thèse, cette proposition constitue une alternative au chiffrement symétrique. Elle était motivée par le fait que, dans certains cas (ex. transmission temps réel), le temps d'exécution du chiffrement n'est pas acceptable et qu'une fragmentation de données peut-être suffisante. Elle est destinée à être utilisée dans un environnement multi-clouds dans lequel les fournisseurs des données sont considérés comme curieux - ils essaieront d'examiner les données qui leur ont été confiées - mais ne feront pas l'effort de collaborer avec d'autres fournisseurs. Le but de l'algorithme est de fragmenter les données entre les clouds de manière à ce que les fragments reçus par un seul fournisseurs soient pratiquement inutiles.

Dans un premier temps, FSFA disperse la donnée initiale sur plusieurs fragments. Les données sont ensuite encodées en utilisant une version modifiée de l'algorithme de partage de secret de Shamir. L'encodage crée des



(a) (b)

Figure 5: *Tests de performance de l'algorithme FSFA pour deux configurations: 2 clouds utilisés (gauche) et 3 clouds utilisés (droite).*

dépendances entre les fragments. Au final, les données encodées sont permutées en utilisant des permutations pseudo-aléatoires pour complexifier le décodage.

FSFA pourrait être considéré comme un cas particulier d'une méthode plus générale de protection de données. En effet, on peut imaginer différentes variantes de chaque étape de l'algorithme. Du point de vue de la mise en œuvre, les performances pourraient être améliorées en exploitant pleinement diverses possibilités de parallélisation du traitement.

Fragmentation dans les réseaux de capteurs

Dans le monde de l'Internet des Objets, les réseaux de capteurs sans fil sont largement utilisés pour rassembler toutes sortes d'informations environnementales. Dans une approche classique, ils fonctionnent en mode temps réel où, juste après l'acquisition de la donnée, les capteurs la déplacent vers un nœud statique du réseau appelé *"sink"*. Cependant, dans certaines situations, la présence de ce nœud statique ne peut pas être assurée, par exemple, lorsque des capteurs sont déployés dans des zones vastes ou hostiles telles que des parcs nationaux, des zones frontalières, etc. Par conséquent, le terme *Unattended Wireless Sensor Networks* a été introduit pour définir une classe de réseaux de capteurs où les données sont stockées dans les capteurs en attendant d'être ramassées par un *sink* mobile (par exemple un drone) qui visite périodiquement les noeuds du réseau.

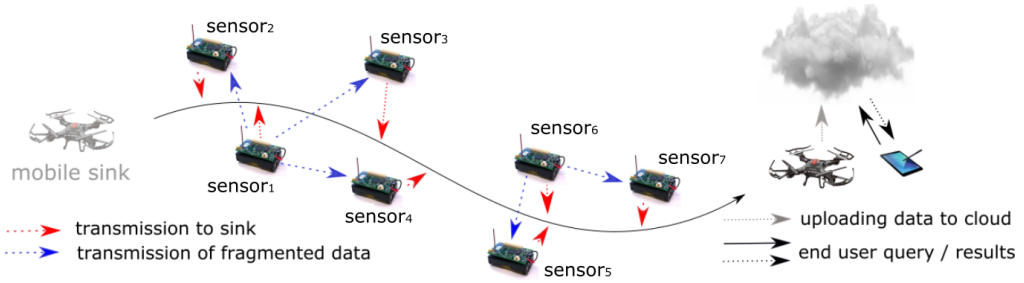
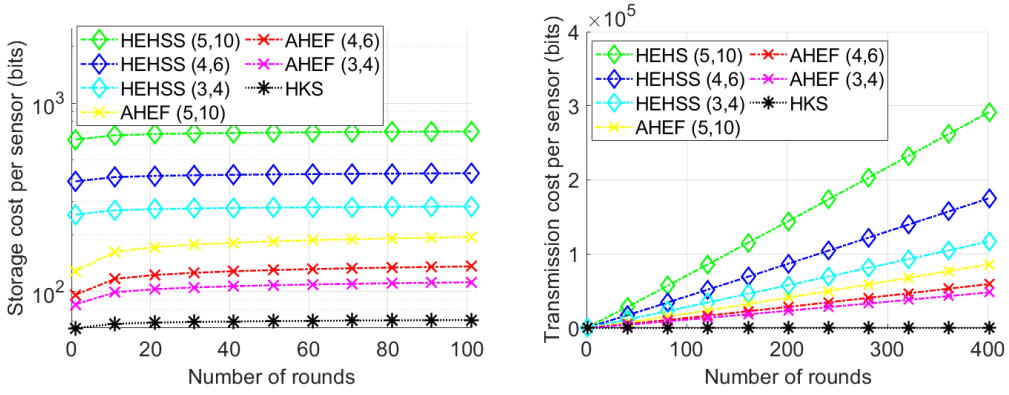


Figure 6: *Fragmentation dans les UWSN.*

Les données conservées à l'intérieur des nœuds peuvent être exposées à divers types d'attaquants désirant lire, détruire ou corrompre les informations stockées. Plusieurs stratégies de protection de données ont déjà été proposées. Par exemple, pour assurer la survie des données, celles-ci peuvent

être répliquées et dispersées sur différents nœuds. Une approche différente assurant, en outre, la confidentialité des données consiste à faire en sorte que les capteurs chiffrent, fragmentent et dispersent les données sur leurs voisins (voir le schéma d'architecture présenté Figure 6). La reconstruction des données est alors impossible à moins qu'un seuil donné de fragments soit rassemblé. Ainsi, les données sont protégées contre un attaquant incapable de compromettre le nombre requis de capteurs entre les visites consécutives du *sink* mobile.



(a) *SO - fragmentation techniques.* (b) *SO - homomorphic techniques.*

Figure 7: *Occupation de mémoire et cout de transmission par capteur.*

Plusieurs schémas de fragmentation aux *UWSN* ont été introduits dans la littérature. Ils proposent un processus basé sur le partage de secret ou une combinaison de chiffrement et de codes correcteurs d'erreurs. Étant donné que la consommation d'énergie est un problème important au sein d'*UWSN*, certaines propositions utilisent des schémas homomorphes afin de réduire les coûts de stockage et de transmission. Le schéma Additively Homomorphic Encryption and Fragmentation (AHEF) présenté au Chapitre 6

revient sur le schéma HEHSS. Dans les deux schémas, les données fragmentées et chiffrées sont agrégées dans les nœuds des voisins. Cependant, AHEF améliore considérablement le schéma HEHSS - au lieu d'utiliser le partage de secrets pour la fragmentation des données, AHEF utilise un algorithme de dispersion d'informations homomorphe. Cette modification a un impact considérable sur les coûts indirects de stockage et les coûts de transmission qui en résultent. En comparaison avec le schéma de fragmentation le plus proche de l'état de l'art, ces coûts son divisés par au moins deux (voir les résultats de la comparaison présentés sur la Figure 7). En effet, les fragments obtenus à l'aide du partage de secret ont la même taille que les données elles-mêmes. La dispersion des informations produit des fragments de taille réduite grâce à l'utilisation des propriété homomorphes. Ainsi, une augmentation des données à protéger est évitée. En outre, AHEF permet deux méthodes d'agrégation de fragments: l'agrégation de fragments provenant du même capteur et l'agrégation de fragments provenant du même groupe de capteurs.

Conclusion et travaux futurs

Après la réalisation d'un état de l'art détaillé, six schémas ont été proposés pour améliorer les techniques de pointe en matière de protection des données par fragmentation dans deux types d'environnements distribués. Ils ouvrent la porte à plusieurs pistes de recherche et soulèvent quelques questions en suspens qu'il convient de traiter dans les travaux futurs.

Sommaire des contributions

Au Chapitre 2, une enquête approfondie sur la fragmentation en tant que moyen de préserver la confidentialité des données a été effectué et a donné comme résultat une vaste analyse de l'état de l'art. Cet état de l'art a été organisé en discernant les méthodes de fragmentation prenant en considération ou non la structure de données. Les techniques pertinentes ont été décrites en détail et comparées en termes de niveau de protection des données, de surcharge de stockage et de complexité. Les algorithmes de fragmentation ont été implémentés et comparés pour valider leur complexité théorique. Non seulement les techniques, mais aussi les systèmes académiques et commerciaux pertinents ont été analysés et comparés. Au final, plusieurs recommandations ont été données sur la conception d'un système de fragmentation efficace.

Au Chapitre 3, la famille des algorithmes de type '*all-or-nothing*' (tout-ou-rien) a été agrandie avec trois nouvelles propositions. Elles visent toutes à protéger les données chiffrées et fragmentées contre l'exposition de la clé de chiffrement. Ils sont conçus pour être utilisés dans trois contextes différents: pour la fragmentation des données dans un environnement multi-cloud, un système de stockage distribué quelconque et un environnement composé d'un seul fournisseur de stockage et un dispositif privé. L'évaluation de la complexité et les critères de performance montrent que les algorithmes présentés sont les plus rapides de toutes les techniques pertinentes connues.

Au Chapitre 4, une manière d'accélérer la fragmentation a été présentée, qui offre de meilleures performances que le chiffrement de données en utilisant l'algorithme de chiffrement à clé symétrique le plus courant (AES-NI). Ce

gain de performances est obtenu en limitant le traitement de chiffrement: avant la fragmentation une transformation tout-ou-rien est appliquée sur les données partiellement chiffrées qui crée des liens entre les données claires et les données chiffrées. Les fragments sont ensuite dispersés de façon à renforcer leur protection. Cette contribution répond au besoin des utilisateurs donnant la priorité à la vitesse de traitement.

Au Chapitre 5, un schéma de fragmentation léger basé sur le codage, la permutation et la dispersion des données a été introduit. Il élimine totalement le chiffrement des données, ce qui permet une fragmentation encore plus rapide: jusqu'à deux fois plus rapide que le chiffrement de données. La proposition revisite l'utilisation du partage de secret. Contrairement à d'autres contributions, cette proposition est une alternative au chiffrement symétrique. Elle était motivée par le fait que, dans certains cas (ex. transmissions temps réel), le temps d'exécution du chiffrement peut ne pas être acceptable et que la dispersion peut-être suffisante.

Au Chapitre 6, la fragmentation au sein des réseaux de capteurs a été réexaminée, en particulier dans les réseaux de capteurs sans fils. Dans ce cas, l'accent est mis non seulement sur la performance en matière de fragmentation, mais également sur la réduction des coûts de stockage et de transmission grâce à l'agrégation de données. Ceci est motivé par le fait que, contrairement à la fragmentation dans le cloud, la fragmentation au sein d'un réseau de capteurs doit prendre en compte les capacités énergétiques limitées des batteries des capteurs. Par rapport aux techniques pertinentes, le schéma proposé réduit d'au moins la moitié le nombre de bits stockés et transmis.

Travaux futurs

Plusieurs travaux futurs peuvent être envisagés, entre autres:

- L'intégration des algorithmes proposés avec les librairies standards utilisés pour le chiffrement des données améliorerait encore plus les performances de la fragmentation. De plus, les performances pourraient être améliorées en exploitant pleinement diverses possibilités de parallélisation du traitement.

- Transformation du Fast and Scalable Fragmentation Algorithme (FSFA) en une méthodologie plus générale combinant plusieurs mécanismes légers afin de protéger les données fragmentées. En effet, diverses modifications pourraient être apportées, par exemple le schéma de partage de secret de Shamir utilisé pendant l'encodage des données pourrait être remplacé par un autre algorithme.

- Sécurisation de la transmission des fragments. Il a été supposé que l'utilisateur avait accès à un nombre suffisant d'emplacements de stockage séparés physiquement ou indépendants, c'est-à-dire différents centres de données du même fournisseur de stockage ou de différents fournisseurs de stockage. Cependant, la question de savoir comment assurer la séparation des fragments n'est pas triviale. Par conséquent, les travaux futurs viseront à garantir une distribution sécurisée des fragments de données de la zone de confiance où la fragmentation se produit jusqu'à leur destination de stockage.

- Traitement des fragments. La question d'un traitement sécurisé est

inévitable dans le cas de données externalisées. Le chiffrement homomorphe constitue un moyen théorique de résoudre ce problème, mais trouve que des domaines d'applications limités vu son manque de performance. Le calcul multiparti et le chiffrement interrogeable semblent une direction beaucoup plus prometteuse pour les années futures. Une piste de recherche adapterait les techniques de calcul multiparti et de chiffrement interrogeables existantes à la nature fragmentée des données.

- Conception d'une solution complète de stockage. À l'avenir, les algorithmes de fragmentation présentés pourraient être intégrés aux systèmes existants afin de les enrichir d'un mécanisme de protection de données supplémentaire ou d'une alternative de fragmentation rapide et légère. Par exemple, ils pourraient être intégrés au système HAIL, offrant une haute disponibilité et une intégrité optimale pour le stockage sur le cloud ou le projet multi-cloud DepSky. De plus, les schémas tout-ou-rien présentés au Chapitre 3 pourraient être utilisés comme moyen de créer des dépendances entre des données dans des systèmes de gestion d'accès rapide.

- Évaluation des coûts de distribution (en terme d'énergie) des fragments dans un environnement de type *UWSN*. De plus, la distribution des fragments pourrait être développer, par exemple les fragments pourraient être distribués d'une façon *multi-hop*.

Une partie de ces travaux a déjà commencé.

Chapter 1

Introduction

This chapter introduces the motivation behind this dissertation and announces the presented contributions.

1.1 Background and Motivation

Efficient protection of data confidentiality is not a recent challenge. Actually, the first methods aiming at obfuscating or encoding sensitive information date back to antiquity, like the scytale transposition or the Caesar's cipher [Kah67]. For centuries, humanity has been developing various techniques of data protection until the invention of cryptography in its recent form in the last few decades. Although historical and currently applied methods obviously differ from each other, they share the same principle: they transform some sensitive input data using a secret known as the key and/or a secret method (in modern cryptography only the key is secret, according to the Kerckhoff's principle [Ker83]). In order to reconstruct the initial in-

formation, two elements are required: the transformed data and the key that allows the data to revert to its original state. In symmetric-key algorithms, being currently the most common way of providing data confidentiality, the secret required during the data reconstruction corresponds to the one used during data transformation. As data confidentiality is only achieved when the secret key is separated from the transformed data, it is crucial to keep the key in a secure place that belongs to the data owner or people to the data was entrusted to. Clearly, data becomes vulnerable once the secret key is compromised. Thus, the main difficulty that a potential attacker has to overcome consists in obtaining the secret, which in practice is equivalent to accessing the storage location that keeps it, performing a brute-force search over all possibilities of the secret (or guessing it knowing some additional information if the secret is far from being pseudo-random).

Several questions arise: How can the security of the storage location keeping the secret key be guaranteed? What if an attacker can guess the secret without compromising the place that stores it? What will happen if the secret key is lost or destroyed? Currently, key management solutions try to address these questions by creating secure key stores accessible only to authorized users. As an alternative, the key can be fragmented into several fragments and spread over several separate locations or distributed among several shareholders. Therefore, an attacker has to compromise not only one key store but several of them. However, even the most sophisticated key management system may not help if the key was generated in an incorrect way, is easily guessable, or the attacker is able to bribe or coerce the holder of the secret.

The problem of secure data protection could be approached from a different perspective. Instead of basing the protection solely on the secure storage of the key, data can be in addition fragmented and dispersed, in a way that all of its fragments are needed for the initial data reconstruction. Such processing makes the importance of the data fragments comparable to a secret key and obliges the attacker to compromise not a single key store, but multiple storage locations. Such data fragmentation could be divided into two categories. In the first one, there is no other way to reconstruct the initial data apart of gathering all of the data fragments as dependencies are created between fragments of data. In the second one, the knowledge of the secret key allows to reconstruct a part of the information from a single fragment; fragmentation is performed in a simple way that does not create dependencies between data fragments. In both types of data fragmentation, in order to make data resilient to intentional or accidental fragments loss, only a threshold of the fragments can be required for the data reconstruction.

Fragmentation as a way of providing or reinforcing data confidentiality is not a recent idea. It can be found in Shamir's [Sha79] and Blakley's [Bla79] seminal papers from late 70s , addressing the problem of secure storage and management of an encryption key. Few years later, a more architectural technique was proposed with a design separating sensitive data from non-confidential fragments and dispersing them over separated devices[DBF91]. In the following decades, the idea of fragmentation was applied to self-securing and archival storage [SGMV09, SGS+00], as well as to relational database systems [ABG+05, CVF+10] as a way of ensuring user's data privacy. Recently, it is revisited in the context of multi-cloud architectures

allowing an easy and efficient data separation [BCCBF13, BGJ+13, HIK+13, DCdVEF+14]. A more detailed insight into those techniques and systems is given in Chapter 2 containing a detailed survey on fragmentation as a way of data protection.

Apart of the reinforcement of data protection, several arguments come out in favour of applying data fragmentation inside distributed storage. First, fragmentation allows to reduce, and in some cases event remove, the costs usually dedicated to the key management. A centralized secure key store is not needed anymore or becomes less critical as the dispersal plays a major role in the data protection process. Second, fragmentation is already widely used for other purposes. Indeed, it is currently applied in systems using the RAID storage technology or error-correction codes. Moreover, it was demonstrated that when applied in a multi-cloud environment it improves not only data confidentiality, but also availability and integrity [BCQ+13]. Third, fragmentation enables a more efficient data deduplication and thus can be successfully employed inside systems for version control or storage of backup data [LQLL16]. Last but not least, fragmentation techniques of the all-or-nothing type may be used in fast access management systems [BDCdVF+16]. Access revocation of outsourced data shared between multiple users is slow as it requires to re-encrypt the data using a fresh key. Re-encrypting only a fragment of data in order to revoke the access accelerates the process.

Although data fragmentation presents multiple benefits, it also comes with several limitations that can discourage from its deployment. First, the majority of secure data fragmentation techniques is slower than symmetric encryption performed over the whole data, which is the recommended tech-

nique for protection of the data confidentiality [SSS+07]. Some of them, like perfect secret sharing or XOR-splitting, lead in addition to a huge increase in the volume of the stored data and/or terribly lack of scalability. Moreover, dispersing data using not one but several communication channels may increase the latency. Second, an efficient protection based on fragmentation requires the access to at least several physically separated servers and thus may lead to an increase of costs.

Nevertheless, what was seen as a major obstacle not more than ten years ago, becomes less restrictive with the recent technology development. On the one hand, even mere users have access to devices with spectacular capacities in terms of computing power. Data processing can be accelerated using parallelization and new hardware is adapted to improve the speed of encryption operations (see the AES intrinsic instruction set AES-NI [1]). Moreover, the transmission capacities enormously improved since the time when fragmentation was being introduced. On the other hand, fragmentation is not anymore solely related to private data centers composed of multiple servers as other architectures also enable data dispersal. First, even mere users have access to a multitude of storage devices distributed across the word as storage providers offer solutions that are cost-efficient and easy to use. Second, in a world of internet-of-things, we are surrounded by networks composed of thousands of sensor nodes used for measuring and storing all kind of environmental variables.

The complexity of the fragmentation processing or the accessibility of

[1] https://www.intel.com/content/www/us/en/architecture-and-technology/advanced-encryption-standard–aes-/data-protection-aes-general-technology.html

a distributed architecture are old issues that nowadays become less significant. However, at the same time new challenges are raised. Especially, preserving data confidentiality and privacy in the presence of sophisticated attackers is becoming a serious and relevant problem [KSLC17]. Users witness data breaches leading to exposure of sensitive data and consequently start to doubt into the capacity of their providers to protect the stored data (see cases like the data theft of 3 billions of Yahoo users [2], the infamous Equifax data breach that exposed the sensitive information of 143 millions of Americans [3], or the late SingHealth theft of personal information of 1.5 millions patients [4]). Moreover, cases of misusing of personal data were recently revealed (see Facebook, Cambridge Analytica case [5]). International regulations, like the recently introduced European General Data Protection Regulation [VVdB], recommend to at least encrypt the data before outsourcing them, making the data owners (and not the storage providers) responsible for the key management. Still, mere users are anxious that they private data could be exposed or misused for commercial or political reasons. Moreover, enterprises of all size fear that a data breach will inevitably make them loose their reputation. For both categories of users, fragmentation as a way of reinforcing the data protection seems to be a promising track.

[2] https://www.reuters.com/article/us-yahoo-cyber/yahoo-says-all-three-billion-accounts-hacked-in-2013-data-theft-idUSKCN1C82O1

[3] https://www.ftc.gov/equifax-data-breach

[4] https://www.straitstimes.com/singapore/personal-info-of-15m-singhealth-patients-including-pm-lee-stolen-in-singapores-most

[5] https://en.wikipedia.org/wiki/Facebook-Cambridge_Analytica_data_scandal

1.2 Contributions and Dissertation Overview

This dissertation revisits known fragmentation techniques, particularly the Rivest's all-or-nothing transform [Riv97] or Shamir's secret sharing [Sha79], making them usable for users looking for secure but at the same time cost-efficient and fast storage solutions. The main focus is put on increasing data confidentiality without deteriorating fragmentation performance. This is especially motivated by the fact that a detailed survey on data fragmentation (presented in Section 2) demonstrated that although multiple fragmentation techniques were proposed since the late 70s, they are not widely used for storage of large data as they are slow, difficult to integrate, or lead to an increase in overall data volume.

The ultimate goal of this thesis is to provide a set of methods that could be directly applied inside an industrial context to reinforce the confidentiality of the stored data. As current standards imply the use of symmetric encryption for protection of data confidentiality, the majority of the proposed methods leave symmetric encryption as its core component and reinforces the protection by using additional mechanisms. Thus, the contributions are not in contrast with the usual process but rather inherit from it. Issues of data availability or integrity are not treated in this thesis as separate research tracks provide solutions that could be integrated within the proposed methods.

Chapter 2 contains a rich survey on data fragmentation as a way of preserving data confidentiality. It introduces two new definitions dividing fragmentation into bitwise and structurewise. Relevant fragmentation techniques are divided according to these definitions and described in detail. They are

compared in terms of data protection levels, storage overhead, and complexity. Performance benchmarks confirm the theoretical complexity evaluation. In addition to techniques, academic and commercial systems applying data fragmentation are presented. The main characteristics of the systems are described and the academic and commercial approaches are compared. Finally, recommendations are given on the design of an efficient fragmentation system.

In Chapter 3, the family of all-or-nothing transforms is extended with three new proposals. They all aim at protecting encrypted data against the exposure of the encryption key; encrypted data are transformed into fragments so the decryption of even a single ciphertext block is impossible unless all the fragments are gathered. The proposed schemes are the fastest of all the state-of-the-art techniques as they require a very low number of operations in addition to data encryption. Moreover, a fast fragmentation solution is introduced addressing the needs of users that do not want or cannot afford to use of a multitude of storage sites (like those using using a basic account inside a public cloud). It allows them to protect their data against the curiosity of the cloud or potential external attackers.

Chapter 4 modifies some of the schemes presented in Chapter 3 making their fragmentation processing even faster than data encryption using the most common symmetric-key encryption algorithm. This gain in performance is achieved by limiting encryption processing and relying on the protection provided by data dispersal. This contribution addresses the need of users prioritizing the speed of processing.

Chapter 5 introduces a lightweight fragmentation scheme based on data

encoding, permutation, and dispersal that totally gets rid of data encryption. This allows the fragmentation processing to be performed even faster - up to twice as fast as data encryption. The proposal revisits the use of the Shamir's secret sharing, trading its perfect security for scalability. In contrast to other contributions, this proposal comes with an alternative to symmetric encryption. It was motivated by the fact that in some cases, a lightweight data protection combined with dispersal may be sufficient as suggested in [CBHK15, BLU+15]. It should be considered as a first sketch of a fragmentation methodology that would have to be investigated in more detail before being deployed in an industrial context.

Chapter 6 revisits data fragmentation inside sensor networks, particularly in the Unattended Wireless Sensor Networks. The main focus in this case is put not solely on the fragmentation performance but also on the reduction of storage and transmission costs by using the data aggregation. This is motivated by the fact that, contrary to fragmentation in the cloud, fragmentation inside a sensor network has to account for the limited energy capacities of the sensors' batteries. When compared with relevant state-of-the art techniques, the proposed scheme reduces by at least half the number of stored and transmitted bits.

Chapter 7 contains a summary of the research contributions as well as a detailed insight into future work. It opens the way to several new research tracks.

Bibliography

[ABG+05] G. Aggarwal, M. Bawa, P. Ganesan, H. Garcia-molina, K. Kenthapadi, R. Motwani, U. Srivastava, D. Thomas, and Y. Xu. Two can keep a secret: A distributed architecture for secure database services. In *In Proc. CIDR*, 2005.

[BCCBF13] Anis Bkakria, Frédéric Cuppens, Nora Cuppens-Boulahia, and José M. Fernandez. *Confidentiality-Preserving Query Execution of Fragmented Outsourced Data*, pages 426–440. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[BCQ+13] Alysson Bessani, Miguel Correia, Bruno Quaresma, Fernando André, and Paulo Sousa. Depsky: Dependable and secure storage in a cloud-of-clouds. *Trans. Storage*, 9(4):12:1–12:33, November 2013.

[BDCdVF+16] Enrico Bacis, Sabrina De Capitani di Vimercati, Sara Foresti, Stefano Paraboschi, Marco Rosa, and Pierangela Samarati. Mix&slice: Efficient access revocation in the cloud. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 217–228, New York, NY, USA, 2016. ACM.

[BGJ+13] J. M. Bohli, N. Gruschka, M. Jensen, L. L. Iacono, and N. Marnau. Security and privacy-enhancing multicloud architectures. *IEEE Transactions on Dependable and Secure Computing*, 10(4):212–224, July 2013.

[Bla79] George R. Blakley. Safeguarding Cryptographic Keys. In *Proceedings of the 1979 AFIPS National Computer Conference*, volume 48, pages 313–317, June 1979.

[BLU+15] William J. Buchanan, David Lanc, Elochukwu Ukwandu, Lu Fan, Gordon Russell, and Owen Lo. The future internet: A world of secret shares. *Future Internet*, 7(4):445, 2015.

[CBHK15] P. Cincilla, A. Boudguiga, M. Hadji, and A. Kaiser. Light blind: Why encrypt if you can share? In *2015 12th International Joint Conference on e-Business and Telecommunications (ICETE)*, volume 04, pages 361–368, July 2015.

[CVF+10] Valentina Ciriani, Sabrina De Capitani Di Vimercati, Sara Foresti, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. Combining fragmentation and encryption to protect privacy in data storage. *ACM Trans. Inf. Syst. Secur.*, 13(3):22:1–22:33, July 2010.

[DBF91] Y. Deswarte, L. Blain, and J. C. Fabre. Intrusion tolerance in distributed computing systems. In *Proceedings. 1991 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 110–121, May 1991.

[DCdVEF+14] Sabrina De Capitani di Vimercati, Robert F. Erbacher, Sara Foresti, Sushil Jajodia, Giovanni Livraga, and Pierangela Samarati. *Encryption and Fragmentation for Data Confiden-*

tiality in the Cloud, pages 212–243. Springer International Publishing, Cham, 2014.

[HIK+13] Aleksandar Hudic, Shareeful Islam, Peter Kieseberg, Sylvi Rennert, and Edgar R. Weippl. Data confidentiality using fragmentation in cloud computing. *International Journal of Pervasive Computing and Communications*, 9(1):37–51, 2013.

[Kah67] David Kahn. *The codebreakers: the story of secret writing*. 1967.

[Ker83] Auguste Kerckhoffs. *La cryptographie militaire*, volume 9. Journal des sciences militaires, 1883.

[KSLC17] G. O. Karame, C. Soriente, K. Lichota, and S. Capkun. Securing cloud data under key exposure. *IEEE Transactions on Cloud Computing*, pages 1–1, 2017.

[LQLL16] M. Li, C. Qin, J. Li, and P. P. C. Lee. Cdstore: Toward reliable, secure, and cost-efficient cloud storage via convergent dispersal. *IEEE Internet Computing*, 20(3):45–53, May 2016.

[Riv97] Ronald L. Rivest. All-or-nothing encryption and the package transform. In *In Fast Software Encryption, LNCS*, pages 210–218. Springer-Verlag, 1997.

[SGMV09] Mark W. Storer, Kevin M. Greenan, Ethan L. Miller, and Kaladhar Voruganti. Potshards: a secure, recoverable, long-

term archival storage system. *Trans. Storage*, 5(2):5:1–5:35, June 2009.

[SGS+00] John D. Strunk, Garth R. Goodson, Michael L. Scheinholtz, Craig A. N. Soules, and Gregory R. Ganger. Self-securing storage: Protecting data in compromised system. In *Proceedings of the 4th Conference on Symposium on Operating System Design & Implementation - Volume 4*, OSDI'00, Berkeley, CA, USA, 2000. USENIX Association.

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979.

[SSS+07] Karen Scarfone, Murugiah Souppaya, Matt Sexton, et al. Guide to storage encryption technologies for end user devices. *NIST Special Publication*, 800:111, 2007.

[VVdB] Paul Voigt and Axel Von dem Bussche. *The EU General Data Protection Regulation (GDPR)*, volume 18. Springer.

Chapter 2

Data protection by means of fragmentation in distributed storage systems

2.1 Introduction

Protecting data confidentiality using a combination of data fragmentation and dispersal is not a recent idea. In fact, one of the first secret sharing techniques can be found in Adi Shamir's [Bla79] or George Blakley's [Bla79] seminal papers from the late 70s. In these still timely proposals, fragments of the encryption key are distributed over several collaborators. The key reconstruction is only possible when a given threshold of the fragments is being gathered. Around the same time, the French LAAS-CNRS laboratory designed an architecture for secure data storage and processing that distributed data fragments over multiple physical devices of different trust-

worthiness levels [DBF91]. During the following decades, data fragmentation and dispersal were considered in a multitude of various contexts: protection of archival data [SGMV09], proactive data storage [SB05], preserving privacy of relational database systems [ABG+05, BCCBF13, CVF+10], securing commercial object storage solutions [RP11, SJH05, TG11a], and reinforcing data confidentiality, integrity, or availability in multi-cloud environments [BCQ+13, BGJ+13, DCdVEF+14, HIK+13]. Nowadays, a new generation of solutions based on data fragmentation and dispersal is emerging [1]. They address the concerns of users who fear the exposure of their outsourced data but still want to store data in public areas for obvious reasons of cost reduction.

This chapter presents relevant fragmentation techniques that aim to ensure confidentiality or privacy of stored data. It also gives an overview of the historical as well as current distributed storage systems applying the described techniques. In order to better organize the descriptions, two definitions of two different types of data fragmentation are introduced:

Definition 1 ***Bitwise fragmentation*** *is defined as the process of data fragmentation without regards to the data structure; data is just treated as set of consecutive bits, each of which is of equal confidentiality level.*

Definition 2 ***Structurewise fragmentation*** *is defined as the process of data fragmentation with regards to the data structure; data is divided into subsets of bits of different confidentiality levels that will be protected in dif-*

[1] http://wikibon.org/wiki/v/3_Tenants_of_Security_and_the_Role_of_Information_Dispersal

ferent ways, i.e. dispersed over physical devices characterized by different trustworthiness levels.

Presented techniques and systems are divided into two groups according to those two definitions. The first group addresses the user's need for storing data without making any assumptions about its type. Typically, it will include all kinds of object storage methods and systems. The second group gathers mainly applications of data fragmentation inside relational database storage systems.

2.2 Data Concepts and Notation

Different terminology is applied in different works; for instance, fragments are sometimes called chunks, shards, or shares. The following data concepts, notation, and terminology are introduced in order to unify and facilitate the description of relevant works.

Data Concepts

The presented fragmentation techniques perform two kind of operations on their input data: they can segment the data into physically separate fragments or they can transform the data (i.e. encrypt or encode). Therefore, data concepts will be divided into two groups corresponding to those two kind of operations.

Concepts connected with data fragmentation:

- **Data (D or d)**: An initial vector of data bits of size $|D|$ bits. When the data is represented as an integer it is denotes as d.

- **Fragment (F)**: The final fragment, a vector of bits of size $|F|$ bits, a result of the data fragmentation.

- **Share ($SHARE$)**: an intermediary fragment of size $|SHARE|$ bits in the case when the data fragmentation process is performed in two steps.

Concepts connected with data transformation:

- **Block (B, P or C)**: a sequence of $|B|$ bits corresponding to the classical concept of a data block. When referring to a plaintext block, the block is denoted as P and, when referring to a ciphertext, the block is denoted as C. When it is not specified if the block is a plaintext or a ciphertext block, it is referred to as simply B.

- **Plaintext ($PLAIN$)**: when the data D is designed to be encrypted, it is denoted as plaintext $PLAIN$ composed of p plaintext blocks (the plaintext is already padded if needed).

- **Ciphertext ($CIPH$)**: encrypted plaintext composed of c ciphertext blocks. The number of blocks inside the ciphertext is equal to $c = p+1$ as an initialization vector is added at the beginning of the ciphertext.

- **Transformed ciphertext ($CIPH'$)**: when an additional processing is applied to the ciphertext after the data encryption, $CIPH'$ denotes the transformed ciphertext.

- **Key (K)**: An encryption key of size $|K|$ bits in the case when a symmetric encryption algorithm is used.

Notation

- n: The total number of final data fragments obtained after the fragmentation process.

- k: In the case when a threshold scheme (in which not all the fragments are required for data reconstruction) is applied, k denotes the threshold: the minimum number of fragments out of the n final fragments that is required for the reconstruction of the data. When a scheme has $kmax$ different thresholds, they are denoted as $k_1, \ldots, k_{kmax}$.

2.3 Bitwise Fragmentation

This section presents an overview of most notable bitwise fragmentation techniques and systems providing data confidentiality. First, it will describe the fragmentation techniques. Second, it will portray important characteristics and elements proper to bitwise fragmentation systems: data resilience, key and fragments' location management, integrity authentication, trustworthiness of the storage nodes, fragment size and decoys, and data deduplication. Finally, it will explain individual descriptions of nine selected bitwise fragmentation storage systems: six academic and three commercial solutions.

2.3.1 Bitwise Fragmentation Techniques

During bitwise fragmentation, initial data D is transformed into n fragments $F_0, \ldots, F_{n-1}$ that are later dispersed over n different physical locations over multiple physically separated storage servers inside of a data center or several independent storage providers. Data reconstruction is only possible when a threshold of k of the fragments is reached. Therefore, protection provided by the data fragmentation depends mainly on the two parameters k and n defining the dispersion scope. On the one hand, a value of k close to n makes the fragments gathering harder as it gives less choice to an attacker. In a particular case, when k equals n, all fragments are needed for data recovery and the attacker has to compromise all of the chosen storage sites. On the other hand, a lower value of k increases data availability as it makes data reconstruction feasible even if some of the fragments are lost or corrupted. Therefore, a wise choice of the k and n setting has to take into consideration the characteristics of the environment in which the fragments will be stored (the issue of providing data resilience inside a distributed storage system is described in more details in Section 2.3.2).

Fragmentation algorithms can be very roughly organized in three groups with a decreasing confidentiality level. The first group includes secret sharing schemes providing the highest level of secrecy - no information can be deduced from a single data fragment. The second group contains computationally secure algorithms that are usually based on symmetric encryption. In this case, an attacker with enough time and computational resources may deduce some information from fewer fragments than the minimum amount required for data reconstruction. The last group gathers all kind of lightweight

fragmentation techniques replacing encryption with data shredding and dispersal in order to improve the fragmentation performance.

The following subsections describe in detail the most relevant fragmentation techniques used for data protection inside historical and modern distributed systems.

Secret Sharing Schemes and Information Dispersal Algorithms

The perfect (or information-theoretically secure) secret sharing schemes transform data D into n fragments, each of a size equal to the size of D. Any k of those fragments are sufficient to recover original information while $k-1$ fragments do not provide any information about the initial data. This is true even if an attacker possesses unlimited computational resources. However, information-theoretical security comes at the price of a large storage overhead as the size of a single data fragment is equal to the size of the data. Therefore, secret sharing schemes are often judged too impractical for voluminous data and are rather used for protection of small data, typically encryption keys. Nevertheless, three of five academic systems proposals described later in Section 2.3.3, adopt perfect secret sharing for data protection, judging the increase of storage as an acceptable cost. POTSHARDS and GridSharing chose XOR-splitting because of its fast performance. PASIS in some situations also accepts the use of perfect security.

In contrast to perfect secret sharing schemes, information dispersal algorithms are space-efficient. They fragment data D into n of size $\frac{|D|}{k}$ each. Like in secret sharing, any k of such fragments is needed for the data reconstruction. Nevertheless, the gain in storage is at the cost of secrecy as

initial data patterns are preserved inside the data fragments resulting in a low confidentiality level. Thus, information dispersal algorithms are mainly used for resilience purpose (as they can resist a loss of $n - k$ fragments) and more rarely are considered as a way of data protection [BLU+15].

Ramp schemes are somehow hybrid (k_1, k_2, n)-threshold schemes situated between secret sharing and information dispersal. Data fragments are protected like in the case of secret sharing, but only until k_1 fragments are gathered. A complete data reconstruction is possible when k_2 fragments are gathered.

Following paragraphs describe in detail two relevant secret sharing schemes (Shamir's secret sharing and XOR-splitting), the Rabin's information dispersal algorithm, as well as linear ramp schemes.

Shamir's secret sharing (SSS) In his seminal work from the late 70s, Shamir introduced a perfect secret sharing scheme [Sha79] that 40 years later still finds multiple applications in the data protection domain. A (k,n)-threshold SSS takes as input data d (represented as an integer) and transforms it into n fragments $F_0, \ldots, F_{k-1}$, any k of which are needed for data reconstruction. SSS is based on polynomial interpolation; it exploits the fact that given k unequal points $x_0, \ldots, x_{k-1}$ and arbitrary values $y_0, \ldots, y_{k-1}$ there is at most one polynomial $y(x)$ of degree less or equal to $k - 1$ such that $y(x_i) = y_i, i = 0, \ldots, k-1$. More precisely, Shamir's scheme uses modular arithmetic. The set of integers modulo a prime number m forms a field in which interpolation is possible. In order to encode data d, a prime number m greater than d and n is picked. A polynomial $y(x)$ of degree $k - 1$ is

constructed where $k-1$ coefficients $coeff_0$,...,$coeff_{k-2}$ are randomly chosen from a uniform distribution over the integers in $[0, m)$. y_i values are computed modulo m. The n computed points $F_i = (x_i, y_i), i = 0, \ldots, k-1$ are the final fragments that will be distributed over n different locations or owners.

In SSS, fragmentation consists in evaluating n times a polynomial of degree $k-1$. The complexity of computing a value at a single point is $O(k)$ when the Horner's scheme is applied. Therefore, it takes $O(kn)$ to compute a polynomial of degree $k-1$ at n points. During data defragmentation, the constant term of the interpolating polynomial is computed using, for instance, the Lagrange interpolation. Unlike the fragmentation, this is an operation quadratic in function of k.

When the data is large, Shamir's advice is to break it into smaller chunks and apply the fragmentation process to each of the chunks separately. Implementations of the scheme usually optimize its performance by performing the operations in the finite field $GF(2^8)$ as it is adapted to the nature of byte computations[2].

SSS was primarily designed to protect secret keys - an encryption key is fragmented and the fragments are distributed among shareholders. In this case, quadratic complexity and n-fold increase in storage are acceptable but, in the context of distributed storage of larger data, they may be a serious obstacle.

Around the same time that Shamir presented his scheme, Blakley [Bla79] published his own scheme relying on the fact that any n nonparallel (n-1)-

[2] http://manpages.ubuntu.com/manpages/xenial/man7/gfshare.7.html

dimensional hyperplanes intersect at only one specific point. However, it did not find wide application inside distributed storage systems.

XOR-splitting XOR-splitting is an information-theoretically secure scheme relying on the one-time pad encryption technique [Sha49]. During fragmentation of data D, $k-1$ random fragments $F_0, \ldots, F_{k-2}$ of size $|D|$ bits each are generated. The last fragment, F_{k-1}, is obtained by exclusive-oring all the fragments with the the data $F_{k-1} = (\bigoplus_0^{k-2} F_i) \oplus D$. In contrast to Shamir's scheme, xor-split does not provide data redundancy in addition to secrecy and the loss of a single fragment makes the data unrecoverable. Therefore, in order to achieve resilience of protected data, it has to be combined with a complementary technique like data replication (the combination of techniques used in the GridSharing system) or a (k, n)-threshold secret sharing scheme (the combination of techniques chosen by the POTSHARDS system). Like in the case of all perfect secret sharing schemes, XOR-splitting leads to a n-fold increase in storage as the fragments are of the size of the data itself. Its advantages are the high confidentiality level. XOR-splitting processing is theoretically very fast as exclusive-or is one of the quickest operations to implement. However, the increase in storage connected with the generation of $k-1$ random fragments can make it slow and not scalable in practice. The idea behind XOR-splitting is very similar to the one behind the Karnin-Greene-Hellman method [DKWGH83, KK03].

Information Dispersal Algorithms (IDA) Rabin introduced the concept of an information dispersal algorithm (IDA) [Rab89] at the end of the 80s. An IDA divides data D of size $|D|$ bits into n fragments. Each fragments

contains $\frac{|D|}{k}$ bits of the data. Any k fragment of those n fragments suffices for reconstruction. More precisely, data D is represented as a collection of k-element vectors. Each of those vectors is transformed into an n-element vector by being multiplied by a $k \times n$ nonsingular dispersal matrix DM. The n-elements of each transformed vector are then dispersed over n fragments $F_0, \ldots, F_{n-1}$. The n rows of the dispersal matrix DM are usually attached within the fragments. Data reconstruction consists of multiplying any k of the fragments by the inverse of a $k \times k$ matrix built from any k rows of the matrix DM.

Rabin's IDA is mainly used for fault-tolerant storage and information transmission as its confidentiality level is too low. Indeed, fragmented data cannot be explicitly reconstructed from fewer than the k required fragments [Li12] though some information about the content of the initial data is leaked. Data patterns are preserved inside fragments when the same matrix is reused to encode different data vectors. A similar problem occurs when using the Electronic Code Book block cipher mode for block cipher encryption [Dwo01]. Even with this weakness, IDA is still sometimes being considered as one the techniques that could be applied in a multi-cloud environment [BLU+15]. When applied to already encrypted data, IDA adds redundancy and reinforces confidentiality.

Data parsing [CBHK15, SJH05] is sometimes seen as a subcategory of information dispersal. Data (encrypted or not) are shredded and distributed with a bit granularity over k fragments. A dispersal key is required to define the parsing pattern. A relative drawback of this solution resides in the fact that it operates at bit level while in many high level programming languages

the byte is the smallest addressable unit of memory. Therefore, an efficient implementation of this kind of solution is a challenge at the programming level and may necessitate the use of a dedicated hardware element (like in [SJH05]).

Ramp schemes Ramps schemes, first introduced in [BM85], are situated between perfect secret sharing and information dispersal. They break data into n fragments, such that any k_2 of them allow data recovery and fewer than k_1 reveal no information at all. The main idea is to gain storage efficiency by relaxing security requirements. One of the simplest ramp schemes modifies the Shamir's scheme by using only k_1 random coefficients inside the encoding polynomial. Data D is used to generate the remaining coefficients of the polynomial. A different way of implementing a linear ramp scheme consists of modifying the Rabin's information dispersal algorithm. Each data vector is composed of k_2 elements, k_1 of which are random and $k_2 - k_1$ belong to the initial data.

In distributed storage, ramp schemes have been considered in the context of distributed storage by the authors of PASIS and CDStore systems. In [LQLL14], a modification of the linear ramp scheme designed for data dispersal in a cloud-of-clouds was introduced: the Convergent Ramp Secret Sharing Scheme (CRSSS). CRSSS replaces random information inside a classical ramp scheme with deterministic hashes generated from the initial data. Such processing allows further fragment deduplication.

Fast and Scalable Fragmentation Algorithm (FSFA) The fast and scalable fragmentation Algorithm (FSFA) is situated somewhere between se-

Table 2.1: *A comparison between relevant techniques in terms of the number of operations required during data fragmentation and defragmentation. (add. - additions, sub. - subtractions, mult. - multiplications, div. - divisions, xors - exclusive-ors)*

Algorithm	**Fragmentation**	**Defragmentation**
Shamir's SS	$Dn(k-1)$ add.	$D(k-1)$ add.
	$Dn(k-1)$ mult.	$D2(k-1)^2 + 2n$ sub.
		$D2(k-1)^2 + 2n - 1$ mult.
		$Dn + 1$ div.
XOR-splitting	Dn xors	Dn xors
Rabin's IDA	Dn mult.	Dk mult.
	Dn add.	Dk add.
Ramp scheme	$D(1 + \frac{k_1}{k_2})n$ mult.	$D(1 + \frac{k_1}{k_2})k$ add.
	$D(1 + \frac{k_1}{k_2})n$ add.	$D(1 + \frac{k_1}{k_2})k$ mult.
FSFA	$D(k-1)$ add.	$D(k-1)$ add.
	$D(k-1)$ mult.	$D(k-1)$ mult.

cret sharing and information dispersal. It is introduced in detail in Chapter 5. It transforms data into k interdependent fragments that all have to be gathered in order to reconstruct the initial information. Its process is a mix of data encoding based on a modification of the Shamir's scheme, data permutation, and data dispersal. A performance comparison with related works demonstrates it can be much faster than fragmentation techniques based on

Table 2.2: *A comparison between relevant techniques in terms of overall required storage, integrated resilience, and provided data confidentiality level. A low confidentiality level preserves data patterns inside the fragments. A lightweight protection does not preserve patterns.*

Algorithm	**Storage**	**Resilience**	**Confidentiality**
Shamir's SS	$\|D\|n$	Yes	Perfect
XOR-splitting	$\|D\|n$	No	Perfect
Rabin's IDA	$\|D\|$	Yes	Low
Ramp scheme	$\|D\|(1+\frac{k_1}{k_2})$	Yes	Perfect up to k_1, then low
FSFA	$\|D\|+\|B\|$, $\|D\| \gg \|B\|$	No	Lightweight

symmetric encryption while producing reasonable storage overhead.

Comparison between presented techniques Table 2.1 and Table 2.2 contain a comparison of the techniques in terms of complexity, storage, resilience and confidentiality level. The following algorithms are compared: Shamir's secret sharing, XOR-splitting, Rabin's information dispersal algorithm, the linear ramp scheme based on an information dispersal algorithm, and the fast and scalable fragmentation algorithm. Figure 2.1 shows a performance benchmark confirming the theoretical evaluation. All techniques were implemented in JAVA language using GF(2^8) which allows for multiplications using look-up tables and replaces addition/subtraction with exclusive-ors.

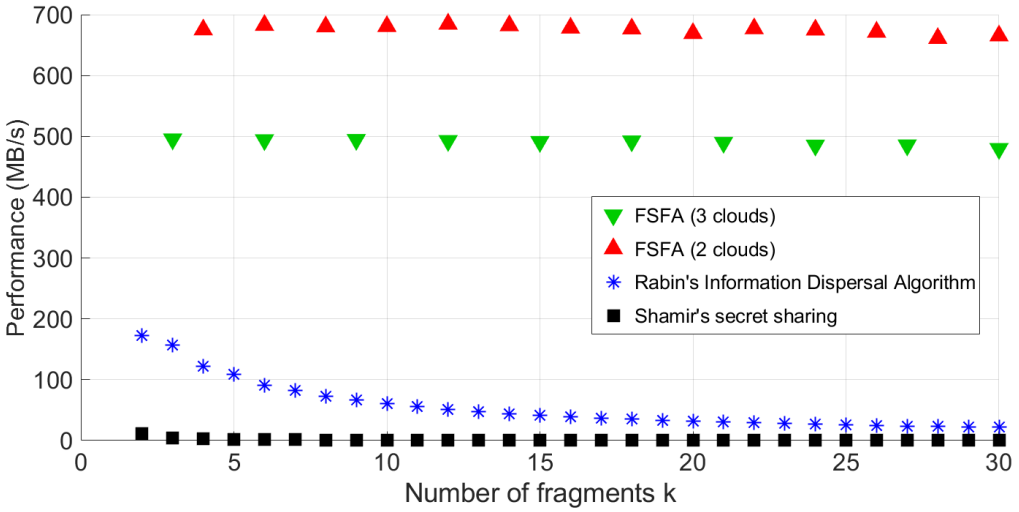


Figure 2.1: *Performance comparison between Shamir's secret sharing (SSS), Rabin's information dispersal algorithm (IDA), and the fast and scalable fragmentation scheme (FSFA, presented in two configurations, a more precise comparison of FSFA with relevant techniques is presented in Chapter 5). SSS and IDA are slow and not scalable while FSFA achieves a good performance that does not decrease with the growing numbers of fragments. A linear ramp scheme based on Shamir's scheme will have a similar or slightly faster performance compared to SSS depending on the chosen threshold. A linear ramp scheme based on IDA will have a performance similar or slightly slower than the IDA depending on the chosen threshold. XOR-splitting in practice achieves the same performance as well implemented SSS. In presented configuration* $k = n$.

Fragmentation Techniques Using Symmetric Encryption

A different group of fragmentation techniques reunites methods based on symmetric encryption. Such techniques first encrypt the data using a sym-

metric encryption algorithm (like AES). Then, they fragment data into n fragments, k of which are needed for data reconstruction. Finally, the encryption key is securely fragmented and transmitted withing the fragments. The big advantage of such techniques is that they provide data confidentiality without leading to an increase of storage.

Secret Sharing Made Short (SSMS) Krawczyk was the first one to introduce a fragmentation scheme combing data encryption and dispersal [Kra94] at the beginning of the 90s. In his proposal named Secret Sharing Made Short (SSMS), data D are first encrypted using a symmetric encryption algorithm with a random encryption key K. Encrypted data are then fragmented using an information dispersal algorithm like the Rabin's IDA into n fragments any k of which are sufficient for data reconstruction. The random key K used during data encryption is fragmented using a (k, n)-threshold perfect secret sharing scheme (usually Shamir's secret sharing) and fragments of the encryption key are attached to the n data fragments. In contrast to the perfect secret sharing, the storage overhead of SSMS does not depend on data size $|D|$ but is only equal to the size of the key $|K|$ per data fragment: Such storage overhead is negligible when larger data are being fragmented.

SSMS defines a fragmentation methodology, leaving implementation details to the user. In modern implementations [BCQ+13], systematic error-correction Reed-Solomon codes [RS60] are used during the information dispersal step. They fragment data in a way that k fragments are formed from large chunks of encrypted data (data are just fragmented in a straightforward way) and $n - k$ fragments are added for data resilience. Systematic error-

correction codes are faster than classic information dispersal algorithm, but they leave large chunks of encrypted data inside the fragments. Therefore an attacker in possession of the encryption key is able to decrypt a portion of information from less than k fragments.

AONT-RS Like SSMS, the AONT-RS method [CLM17, RP11] combines symmetric encryption with data dispersal. The main difference between these two methods lies in the key management. Similarly to SSMS, in AONT-RS data D are first encrypted using a random encryption key and a symmetric encryption algorithm. In a next step, encrypted data are fragmented into k fragments in a straightforward way (data are just divided into chunks composed of consecutive bits) and $n - k$ additional fragments are generated using systematic Reed-Solomon codes [RS60]. In contrast to SSMS, in AONT-RS the encryption key is not fragmented using Shamir's scheme but exclusive-ored with the hash of the encrypted data (so it is unrecoverable in the absence of the complete ciphertext). AONT-RS was clearly inspired by the all-or-nothing transform (AONT) introduced by Rivest [Riv97] (described in detail in Section 2.3.1). However, unlike Rivest's proposal, it does not protect data against the exposure of the encryption key.

CAONT-RS [LQLL16] slightly modifies AONT-RS in order to allow a fragments' deduplication. It replaces the random key used during data encryption with a key generated from the cryptographic hash of the initial data.

All-Or-Nothing Family of Algorithms

An all-or-nothing process mainly aims at protecting encrypted data against the exposure of the encryption key. It creates dependencies inside the ciphertext in a way that its partial decryption is infeasible. Therefore, once the ciphertext is being fragmented and dispersed, encrypted data contained inside the fragments is protected against the key exposure.

Historical (introduced by Rivest [Riv97]) all-or-nothing processing is composed of two steps: an all-or-nothing transform (denoted as AONT) preprocessing applied before data encryption and the data encryption. The whole two step process is denoted in the literature as all-or-nothing (AON) [KSLC17]. The majority of recent all-or-nothing schemes operate differently: they are first encrypting the data and then applying a transform over it to create dependencies. Thus, they avoid the data re-encryption. The following paragraphs present the most relevant all-or-nothing schemes.

Rivest's and Desai's all-or-nothing transforms Rivest was the first one to propose an all-or-nothing process [Riv97]. In his proposal, data are encrypted twice: first during the pre-processing step named the all-or-nothing transform (AONT) and second after the transformation. During this preprocessing step, the plaintext composed of p input blocks is transformed into a sequence of $c = p+1$ output blocks. First, each input block P_i is encrypted using a random key K: $C_i = P_i \oplus E(K, i)$, where $0 \leq i \leq p-1$ (E is a symmetric encryption function). Second, a hash of each output block is computed: $H_i = hash(K_{pub}, C_i \oplus i)$, where $1 \leq i \leq c-1$ using a publicly known key K_{pub} (H is a keyed hash function). Third, the last output block

is computed as an exclusive-or of K and of all hashes: $C_{c-1} = K \bigoplus_{i=1}^{n} H_i$. Such transformed plaintext is then encrypted using a key K_{enc}. Rivest's pre-processing protects against the exposure of the key K_{enc} which was used during the second step of processing. However, it does not protect against a situation where an attacker managed to acquire the random key K used during the pre-processing AONT in addition to the encryption key used after the pre-processing. Rivest's proposal requires two rounds of encryption (one during pre-processing and one after) that could be a burden for performance.

Desai [Des00] proposed a modification to Rivest's proposal replacing the round computing hashes. The last output block C_{c-1} is obtained as an exclusive-or of the random key and of all previously obtained p output blocks: $C_{c-1} = K \bigoplus_{i=0}^{c-2} C_i$. Such processing improves the performance.

Bastion Among the latest developments, Bastion [KSLC17] is an AON composed of the data encryption step and of a linear transform applied to the encrypted data. It ensures that the initial ciphertext blocks cannot be recovered as long as the adversary has access to all but two output blocks. More specifically, the ciphertext $CIPH$ is multiplied by a square matrix A of dimensions $c \times c$, such that: (i) all diagonal elements are set to 0 and (ii) the remaining off-diagonal elements are set to 1 (such a matrix is invertible and $A = A^{-1}$ so the inverse transform: $CIPH = A^{-1} \cdot CIPH' = A \cdot CIPH'$). The multiplication $CIPH' = A \cdot CIPH$ ensures that each output block C'_i will depend on all output blocks C'_j except from C'_i,$i = j$. Bastion achieves much better performance than AONs as it does not require an additional encryption round. Thanks to a wise implementation, the transform applied

after the encryption uses only $2c$ exclusive-or operations ($3c-1$ exclusive-ors are made in total, counting $c-1$ exclusive-ors from the CTR mode).

Mix&Slice Mix&Slice [BDCdVF+16] is an approach to enforce access revocation on data stored at external cloud providers. Dependencies are created inside encrypted data so re-encrypting even a small portion of the outsourced data with a fresh key revokes the access to a user who does not possess the new key. The algorithm used for the data transformation could be seen as a particular case of an all-or-nothing scheme as it is characterized by the same property: data decryption is not possible without the possession of the whole ciphertext. In Mix&Slice, the transformation into final output messages is performed using multiple encryption rounds - each encryption round re-encrypts (and thus creates dependencies between) a different subset of the input data.

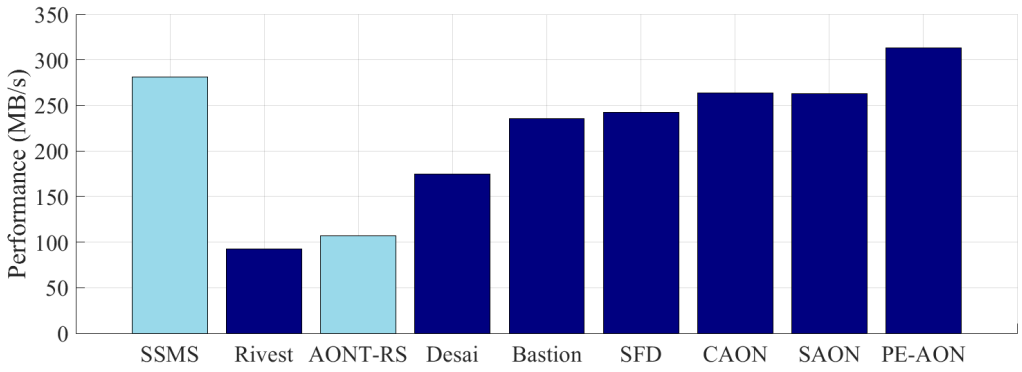


Figure 2.2: *Performance comparison between techniques based on symmetric encryption. In light blue: techniques that do not provide protection against key exposure. In dark blue: techniques that provide protection against key exposure.*

Table 2.3: *Comparison between relevant techniques in terms of number of operations required for data fragmentation (encryption is measured in the number of block operations b.o.), number of exclusive-ors (including those used during encryption), operations used for key management, and data defragmentation. n - total number of fragments, k-number of fragments required for data reconstruction, p- number of plaintext blocks, C- ciphertext block, K- encryption key, r_{MS}- (in Mix&Slice) number of encryption rounds, e- (in PE-AON) number of fragments on which encryption will be applied*

Algorithm	**Enc. [b.c.]**	**xors**	**Key m.**	**K.E.P.**
SSMS	p	p	$SSS(K)$	No
SSMS + IDA	p	$np + p + n$	$SSS(K)$	Yes
AONT-RS	p	p	$Hash(D) \oplus K$	No
Rivest's AON	$3p + 1$	$3p$	-	Yes
Desai's AON	$2p + 1$	$2p$	-	Yes
Bastion	p	$3p + 1$	-	Yes
Mix&Slice	pr_{MS}	pr_{MS}	-	Yes
SFD	p	p	-	Yes
CAON	p	$2p + k - 1$	$C_{p+1} \oplus K$	Yes
SAON	p	from p to $2p$	-	Yes
PE-AON	$\frac{e}{k}p$	$\frac{e}{k}p + 2(p + 1)$	-	Yes

Secure fragmentation and dispersal Presented in detail in Chapter 3, the Secure Fragmentation and Dispersal scheme (SFD) [KM18a] can be also

seen as a form of an all-or-nothing fragmentation. Instead of creating dependencies between the ciphertext blocks, it fragments the data along the dependencies created during a block cipher encryption.

Circular all-or-nothing (CAON) The Circular All-Or-Nothing (CAON) [KM18b] algorithm presented in detail in Chapter 3 improves the Bastion proposal by reducing the complexity of the linear transform applied after the data encryption by a factor of 2.

Selective All-Or-Nothing (SAON) The Selective All-Or-Nothing (SAON) [KM18c] scheme, presented in detail in Chapter 3, is a fast way of protecting data against key exposure in a single cloud environment. It achieves good performance thanks to combining the SFD with the Bastion scheme.

Partial encryption and all-or-nothing (PE-AON) The partial encryption and all-or-nothing (PE-AON) scheme, presented in detail in Chapter 4, improves the fragmentation performance of encryption based fragmentation schemes: it applies encryption only to a subset of the data and then exclusive-ors encrypted and non-encrypted data fragments.

Comparison between presented techniques A comparison between techniques presented in the section is presented in Table 2.3. A performance benchmark is shown in Figure 2.2. All relevant schemes were implemented in similar ways. AES was used as the symmetric encryption algorithm. SHA256 was used as the hash function inside the AONT-RS. More detailed benchmarks comparing algorithms presented in this thesis (SFD, CAON, SAON,

and PE-AON) with the state-of-the-art techniques are shown in Chapter 3 and Chapter 4.

2.3.2 Characteristics of bitwise fragmentation systems

The previous section gave an overview of existing techniques providing data protection by means of fragmentation. This section focuses on other aspects intrinsically connected with fragmentation that will be later analyzed in Section 2.3.3 during the description of the storage systems.

Data Resilience

Any kind of distributed storage system should ensure data resilience as it has to be prepared for the loss or alteration of a part of its data in case of an attack or an incident. In a system applying fragmentation, the ratio between the total number of fragments (n) and the number of fragments required for the data reconstruction (k) should depend mainly on two factors: the trustworthiness of the storage devices and the estimated longevity of the system. Indeed, data dispersed over unreliable machines (i.e. inside a peer-to-peer storage system) are more likely to be lost or altered. At the same time, it is easier to ensure data survival if the longevity of the system is measured in years rather than decades.

Several techniques may be applied to ensure data resilience, like data replication, threshold secret sharing, information dispersal, and systematic error-correction codes. The choice of a suitable technique is not straightforward as multiple factors like the performance of the technique or its impact on the confidentiality and storage requirements have to be taken into account.

Replication is the easiest and fastest solution but also quite inefficient in terms of memory occupation. Its main advantage is that it ensures high data availability as the replicas of fragments are immediately ready to be used and no special processing is required in case of data recovery. Among systems presented in Section 2.3.3, it can be found in GridSharing, DepSky (only as an option), and IBM Object Cloud Storage (only for small data, for which the gain in performance prevails over storage blow-up).

Threshold schemes, like the previously presented Shamir's scheme, provide not only data protection but also resilience. However, in this case resilience comes at an extremely high cost as not only does the performance of such schemes drastically decrease with the increasing number of fragments, but their storage overhead is comparable to data replication. Therefore, they are almost exclusively used for the protection of small data, especially encryption keys. One of the rare uses of secret sharing for ensuring the resilience of larger data can be found in the POTSHARDS system, designed to protect archival data for decades.

Information dispersal algorithm adds resilience without leading to an excessive storage overhead. They perform better than threshold schemes but similarly lack of scalability (performance decreases with the increasing number of fragments). Therefore, recent systems use rather systematic error-correction codes (especially Reed-Solomon codes [RS60]). The principle of systematic error-correction codes is similar to the one of an information dispersal algorithm (both can be seen as matrix multiplication). However, they allow users to save computations by only generating resilient $n-k$ fragments while keeping k fragments as direct chunks of the data.

Key and Fragments' Location Management

The use of a classical key management store may not be necessary inside a bitwise fragmentation system. Indeed, data fragmentation method do not require any encryption key (i.e. when secret sharing is used to create the fragments) or disperses the key within the data fragments (i.e when SSMS or AONT-RS techniques are applied). Thus, of all the systems described later in Section 2.3.3, only Delta-4 and Symform store encryption keys in a separate trusted area.

In systems based on fragmentation, the encryption key is somehow replaced by a *map* - a piece of information mapping information about the stored data with the corresponding fragments' locations and ordering. Even if the map is less critical than a key, its possession clearly facilitates the work of an attacker. Thus, it should be stored in a trusted location, fragmented over independent nodes, or given to the user. The last solution is risky as the probability that the user will loss the map increases with the supposed data longevity [SGMV09]. In order to increase the protection level, the map may be also encrypted or fragmented [PBL91].

The loss of the map is a critical situation. A straightforward solution to this problem would be to broadcast a request to all of the storage nodes in order to discover fragments location. However, this will work only if fragments have a piece of information attached to them describing their origin and order.

An interesting design was proposed by POTSHARDS system, where a user possesses a primary map of their data fragments and a secondary distributed map, named *approximate pointers*, is attached to the fragments.

Therefore, this make the data reconstruction possible even if the user lost their map. However, the secondary map includes a sort of honeypot making the emergency recovery much more time consuming than the standard procedure.

Data Integrity and Authentication

Dispersed fragments may be altered, especially when they are stored on a multitude of untrusted devices. The most common solution addressing the data integrity issue is to compute a digest of the data that will be transmitted to the client within the fragments. The client can then verify the integrity of the defragmented data by computing its hash and comparing it with the one that was received within the fragments. When authentication of the fragments is also required, a keyed-hash message authentication code can be used or the data can be signed using public-key cryptography.

An alternative way of ensuring data integrity was implemented in GridSharing using a voting system. It makes the system resilient to a Byzantine fault; even if a certain amount of storage nodes are corrupted, the system still behaves correctly. GridSharing replicates fragments and distributes them of multiple storage nodes. During data retrieval, several replicas of the same fragments are sent to the client and the most frequent among the set of replicas is considered to be the right one. On one hand, such a technique is very inefficient in terms of storage capacity and transmission costs. On the other hand, over-requesting of the fragments may slightly improve the defragmentation performance. Even if the GridSharing approach does not directly require the computation of cryptographic hashes, it still demands a

comparison between data fragments.

PASIS presented an interesting approach against unauthorized data changes. It audits the storage nodes in order to detect data modifications and restore the data from backup when necessary. Storage nodes are recording any kind of data modifications. In the case of a data loss or undesired data modification, the state of the node will be reverted.

Trustworthiness of the Storage Nodes

Roughly, three principal levels of device trustworthiness can be distinguished: *trusted*, *curious-but-honest*, and *untrusted*. Trusted devices are used for the processing (fragmentation, defragmentation, mapping) of data. They also store maps and encryption keys if necessary. In any kind of system there must be at least one such trusted device (often it is the user's device). Intuitively, communication to and from this component must be appropriately secured as a man-in-the-middle attack would expose fragments to an attacker making the dispersal obstacle useless (it is recommended to use separate channels for the distributions of distinct fragments). Commercial solutions apply TLS to secure communication between the trusted zone and the storage nodes. In the historical Delta-4 project, a less typical approach was proposed where fragments belonging to different data are mixed together during the transmission and thus obfuscate the communication.

Curious-but-honest (or semi-honest) devices try to learn as much as possible about the stored data. However, they behave correctly, do not modify the data, and execute protocols as specified. A storage provider like a cloud is often assumed to be curious-but-honest. The category of untrusted devices

gathers all kind of storage nodes that will not only look at the stored data, but can also deviate from the defined protocols.

Fragment Size and Decoys

Knowledge of the fragments' size opens the door to a potential side-channel attack. Not only does it give an attacker information about the estimated size of the data, but it can also help identify fragments belonging to the same data as most likely they will be equal in size. A possible countermeasure could be having a fixed fragment size inside the whole storage system. This implies pre-processing data - segmenting larger data into smaller chunks while padding smaller data. Such a solution has already been applied, for instance, in the Symform peer-to-peer storage system. Data could also be divided into chunks of various sizes that would obfuscate the size of the data.

Decoys can be defined as fragments aiming to mislead an intruder. They may be implemented in two ways. The first way, consists in generating random fragments that do not belong to any initial data and mixing them with valid fragments. The randomly-generated portion of data obfuscates the exact amount of stored data but inevitably increases storage requirements. A more efficient way of proceeding would be to inject invalid entries to the map with fake locations of the data fragments. These pre-existing fragments (but matching different data files) are now decoys. This technique was adopted by POTSHARDS where, in its secondary map, only one entry out of four lead to a valid fragment.

Data Deduplication

Data deduplication exploits content similarity in order to reduce the overall storage volume [LQLL14, QD02a]. It is especially efficient in the case of archival systems storing large amounts of similar data like backup or log files. Applied inside a bitwise fragmentation system, deduplication prohibits the use of randomness during data fragmentation. Therefore, usual bitwise fragmentation techniques have to be modified in order to produce the same output fragments for same input data. To make a technique deduplicable, the random element used during fragmentation (typically the encryption key) is replaced by deterministic data constructed from the hash of the data [LQLL14].

A naive implementation of data deduplication opens the door to side-channel attacks. Indeed, an attacker eavesdropping the communication between a user and the system nodes may deduce if the user's data exists already inside the system. A two stage deduplication, like the one proposed by CDStore, provides a solution to this problem.

2.3.3 Systems applying bitwise fragmentation

Nine bitwise fragmentation systems were selected among relevant works: six academic solutions and three commercial products. Their main focus is to protect data confidentiality using fragmentation. Systems applying fragmentation solely for a data resilience purpose were put aside. The particularities of each of the systems are described and a comparison between the academic and commercial approaches can be found at the end of this section.

Academic Systems

Six academical systems were described in chronological order: Delta-4, Grid-Sharing, POTSHARDS, PASIS, Depsky, and CDStore.

Delta-4 (LAAS-CNRS, INRIA, and Univerisity of Newcastle upon Tyne) [PBL91] Delta-4 was a European project - one of the first to address the need for a dependable distributed storage system that could resist not only accidental faults but also intentional intrusions. In this proposal, the storage environment is composed of three different sites: a user site performing data fragmentation, an archive site storing the fragmented data, as well as a security site handling authentication, authorization, and key management. The user and security site have to be trusted. Individual nodes of the archive site are not trusted and can be a subject to accidental faults and malicious intrusions. However, the archive site overall is supposed to be trusted.

Delta-4 transforms data using the fragmentation-redundancy-scattering (FRS) technique [DBF91, FDP86]. In a pre-processing step data are cut into chunks of equal size and chunks are encrypted with a block cipher. Encrypted data are then distributed over k fragments using decimation at the byte level; consecutive bytes of data are distributed over separate fragments. Final fragments are then replicated and the replicas are sent to the archive site that distributes them over its storage nodes using a randomized algorithm obfuscating the fragments' locations. Data replication was chosen because of performance reasons as, thirty years ago, CPU cycles were too scarce to efficiently execute error-correction or information dispersal. During

the transmission to the archive site, fragments belonging to different data are mixed, forming a single data flow (if necessary such data flow may be artificially created by injecting decoys).

After two decades, the FRS technique was implemented inside two distributed systems: one peer-to-peer [BSL+06] and one using a central server [CP11].

PASIS (Carnegie Mellon University [WBS+00, GKB+01, SGS+00]) PASIS shares the same objective as Delta-4: designing a storage system capable of handling node failures and activities of malicious users. In order to achieve its goal, it combines data fragmentation with dynamic self-maintenance. Its architecture is composed of storage nodes (vulnerable to attacks and intrusions) equipped with repair agents and agents integrated within client devices (supposed to be inside trusted areas). Agents fragment and disperse data over the storage nodes. A fragment is identified by the name of its storage node and its local name on that node. A dedicated directory service maps the name of data stored over storage nodes to their fragments. Therefore, a careful naming of stored files can obfuscate relations between fragments.

In PASIS, the data fragmentation method is not predefined, but adapted to the type of stored data (details of the right fragmentation choice are presented in [WBP+01]). A wide range of techniques and their combinations is taken into account: secret sharing, encryption, ramp schemes etc. The choice of the right fragmentation technique is a compromise between the desired performance, data availability, and data confidentiality.

Self-securing storage nodes are the PASIS particularity. Each node implements a repair agent that internally keeps track of all changes at the node during a given interval of time. Keeping historical information allows the detection of intrusions and prevents intruders from destroying or tampering with stored data. Systems administrators have a window of time to recognize malicious activity and rebuild the system using the history pool.

PASIS comes up with two interesting suggestions aimed at improving the speed of data retrieval. The first suggestion consists of over-requesting data fragments: asking storage nodes for more than the k fragments during the data retrieval. This way, only the k first fragments that arrived first are defragmented. However, at the same time, the bandwidth usage is increased. The second suggestion is to prioritize nodes that have responded first during recent retrievals.

GridSharing (Georgia Institute of Technology [SB05]) GridSharing proposes a distributed system architecture that organizes storage nodes in the form of a logical grid of $k \times n$ dimensions. It can handle up to cr of node crashes and will behave correctly even if le of the nodes are honest-but-curious and by are Byzantine faulty. The exact values of cr, le, and by depend on the grid dimensions and the data distribution technique.

GridSharing combines XOR-splitting with replication. First, data are fragmented into k fragments, all of which are required for data reconstruction. Each row of the k rows receives a single fragment and replicates it n times over its columns. During defragmentation, a client broadcasts their request to the grid. As an answer, they will receive multiple replicas of the same

fragments from which they will choose the most frequent answers. Such processing leads to a very high overhead of storage and transmission costs (in the example presented in the paper, data are replicated more than 60 times).

An original concept of renewal of data fragments is described. Indeed, the random fragments generated using XOR-splitting could be replaced with a new set after some time. Therefore, an attacker would dispose of less time to gather the required fragments.

POTSHARDS (University of California [SGMV09]) POTSHARDS is a proposal for an archival storage architecture able to last years or even decades. Its basic concept is to distribute fragmented data over several co-operating providers (named organizations).

POTSHARDS fragments data in a two steps process. First, data are fragmented using XOR-splitting into a set of shards. All shards are then required for data reconstruction. To add resilience, each shard is fragmented using Shamir's secret sharing (this results in a considerable increase of storage overhead as secret sharing is applied twice). The final fragments are distributed across independent storage providers. POTSHARDS assures data integrity by the use of algebraic signatures.

Authors motivate the choice of secret sharing with two reasons. First, key management can be expensive over the years as it requires key replacements. Moreover, an encryption key may be lost making the data unrecoverable. Second, even the strongest encryption is only computationally secure and may become become insufficient over a finite period of time taking into ac-

count the fast development of new technologies.

After data fragmentation and dispersal, a user receives a primary map containing the locations of the data fragments. Because it is possible that this primary map will be lost after decades, a secondary map is attached to the fragments in the form of *approximate pointers*. Each fragment contains four pointers to other fragments, but not all of them point at the fragments of the same data. This allows data reconstruction even if the primary map is not anymore available. However, data reconstruction using approximate pointers is time consuming and requires access to multiple different storage providers and nodes.

In the event of a partial data loss, a special protocol allows the providers to reconstruct the lost data without revealing too much of their content where data used during the reconstruction are encrypted and only the fail-over archive receives the corresponding encryption keys.

DepSky (University of Lisbon [BCQ$^+$13]) DepSky is an academic system built on commercial clouds. It fragments and disperses data over several providers in a way that only a subset of the fragments is required in order to reconstruct the initial data. This *cloud-of-clouds* aims at improving data availability, confidentiality, and integrity, as well as protecting against a vendor lock-in problem. Its architecture is composed of clients (a software installed at the user site) reading and writing data stored over four commercial clouds (Amazon S3, Windows Azure, Nirvanix, and Rackspace). To deal with the heterogeneity of cloud interfaces, data are encapsulated inside special data units, the exact implementation of which depends on the archi-

tecture of the storage provider. Each data unit contains information about its content including the data version or signature.

Two protocols for the distribution of data units are proposed: DepSky-A and DepSky-CA. DepSky-A does not provide any data confidentiality but just disperses replicas of data over the clouds in order to increase the data availability. As data are replicated, the overall storage blowup is equal to number of the replicas. DepSky-CA is a secure and space efficient improvement of the DepSky-A. Data processing follows the Krawczyk's Secret Sharing Made Short method. First, data are encrypted using symmetric encryption (AES). They are then encoded into n fragments by the use of an optimal erasure code (Reed-Solomon). The encryption key is partitioned into n fragments using secret sharing (via the Shoenmakers' Publicly Verifiable Secret Sharing scheme [Sch99]) and these key fragments are attached to data. Data integrity is ensured by the use of n digests (one digest for each cloud) stored inside of the metadata (SHA-1 was used for cryptographic hashes and RSA for signatures). The system allows for the replacement of secret sharing by a more traditional key distribution infrastructure.

Depsky's authors demonstrated an improvement of the perceived availability and (in most cases) of the access latency when compared to cloud providers individually. The monetary cost of the data dispersal was estimated at twice the cost of using a single data storage provider.

CDStore (The Chinese University of Hong Kong [LQLL16]) CDStore deduplicates backup-data stored in a multi-cloud environment. The system architecture is composed of CDStore clients integrated with users' ser-

vices and CDStore servers belonging to the cloud storage providers. Data are first divided into shares of variable size that are then transformed into fragments. CDStore keeps only the fragments that are different from those that have already been archived. In order to enable such deduplication processing, fragmentation of two identical input data has to result in two identical sets of fragments [QD02b] (as already described in Section 2.3.2). Therefore, authors of CDStore introduces two fragmentation techniques allowing data deduplication: CAONT-RS and CRSSS [LQLL14]. Both modify existing fragmentation techniques (AONT-RS and RSSS) by replacing their random elements with deterministic data generated from the data hash. Moreover, for performance reasons, CAONT-RS uses an all-or-nothing transformation based on *optimal asymmetric encryption padding* (OAEP) [BR95, Boy99] instead of the one initially proposed in [RP11].

CDStore comes with a two stage deduplication method to resist side-channel attacks. When deduplication is implemented in a naive way, an attacker can deduce information about the stored data by observing fragments being updated to the cloud. With CDStore, produced fragments are deduplicated at the client side between themselves as a first step. Then, the remaining fragments are transferred to the CDStore servers that perform the second deduplication step: they will keep only the fragments that are different from those already inside the clouds.

Commercial Systems

Three commercial systems were chosen for the survey: the IBM Cloud Object Storage (existing[3]), the Symform peer-to-peer system (discontinued[4]), and Unisys's Secure Parser® (existing[5]). All of them reinforce the protection provided by symmetric encryption using data fragmentation.

IBM Cloud Object Storage [RP11] IBM Cloud Object Storage[6] was one of the first commercial products to use fragmentation and dispersal for reinforcing data confidentiality. Created by the Cleversafe startup as a private cloud storage solution aimed at achieving petabyte scalability and reliability, it was acquired by IBM and adapted to be integrated within a hybrid cloud storage [7].

Data inside IBM Cloud Storage are fragmented into around 20-30 fragments using the AONT-RS technique combining symmetric encryption with Reed-Solomon codes (presented in detail in Section 2.3.1). Such fragments are dispersed over random storage nodes. AES-256 is used as the algorithm for symmetric encryption and SHA-256 is used to compute the hash of the data that will be exclusive-ored with the encryption key. Data resilience is achieved as, thanks to the Reed-Solomon error correction codes, not all of the data fragments are required for the data reconstruction. A canary is dispersed within the fragments in order to ensure data integrity. For per-

[3]https://www.ibm.com/cloud/object-storage

[4]https://www.theregister.co.uk/2016/06/17/quantum_file_sync_and_share_sinks/

[5]https://www.unisys.com/offerings/security-solutions/news%20release/latest-release-unisys-stealth-security-extends-protection-to-include-medical-iot-devices

[6]https://www.ibm.com/cloud-computing/products/storage/object-storage/

[7]http://www.pcworld.com/article/3130792/ibms-cleversafe-storage-platform-is-becoming-a-cloud-service.html

formance reasons, in the case of smaller data, the fragmentation process is replaced with data encryption and replication.

AONT-RS does not require the use of a key store as the encryption key is exclusive-ored with the data hash and dispersed within the fragments. Therefore, the storage costs can be reduced by the costs of the key management. However, the key is somehow replaced by a map that allows users (and attackers who find it) to obtain the location of the fragments.

Symform peer-to-peer storage Symform[8] was a peer-to-peer solution offering distributed storage space. It used the distributed nature of its architecture to enhance the confidentiality of the stored data. Symform's fragmentation method was based on the RAID-96^{TM} patented [TG11a, TG11b, TG14] technology. RAID-96^{TM} first divides data into 64MB shares composed of consecutive data bits. It then encrypts the shares using the AES algorithm with a 256-bits key. The encryption key is generated from the hash of the data inside the share, allowing deduplication. In the next step, each of the encrypted shares is shredded into 64 fragments of size 1MB each and 32 additional fragments are generated using Reed-Solomon codes. Finally, the 96 fragments are dispersed across 96 randomly chosen devices. Data reconstruction is possible if any 64 out of 96 fragments are gathered.

Symform was one of the rare fragmentation systems to use a centralized key store situated in a trusted element. The store kept not only encryption keys but also the information about the locations of the data fragments.

[8] http://www.symform.com

Unisys's SecureParser® Unisys's SecureParser® [Joh07, SJH05] aims at improving classical data protection techniques using a combination of encryption, data shredding, and error-correction codes. It is composed of software and hardware components.

First, data are encrypted using the AES algorithm with a random key. Second, encrypted data are fragmented at bit level using a random splitting key (the splitting key parses data and disperses its bits over fragments). Therefore, final fragments are composed of random bits of encrypted data. Both keys, encryption and splitting, are transformed using an all-or-nothing transform and distributed within the fragments. Resilience is added to the data using error-correction codes. An authentication value is also added to the data fragments.

SecureParser® introduces the interesting concept of the mandatory fragment: a fragment that is necessary for a correct reconstruction of the data regardless of the k and n specification. It can be useful in a situation when the user would like to quickly refresh the fragmented data as they will only have to replace the mandatory fragment. Such processing proposal od data refreshment was already proposed by GridSharing (for a description of GridSharing, see Section 2.3.3). Nevertheless, a mandatory fragment should be seen as especially sensitive data. Indeed, in a situation when an attacker will want to make the data unavailable, they will most probably target the mandatory fragment.

Academic Approach vs. Commercial Products

It can be observed that the approach to data fragmentation changed over the years. First, solutions based their protection almost solely on secret sharing for confidentiality and replication for data resilience while recent systems apply data encryption for confidentiality and error-correction for data resilience.

In contrast to commercial products, academic proposals tend to be more original even at the cost of an excessive (like in GridShard or POTSHARDS) increase of storage volume or a drastic decrease of fragmentation performance. They introduce novel solutions, such as repair agents in PASIS, or approximate pointers in POTSHARDS, which deserve more experimenting and probing. They look to the future by trying to be prepared for the possibility that the currently recommended size of the encryption key will no longer be enough.

Commercial products hold to recommended NIST encryption standards (AES with a key size of 128 or 256 bits) that they combine with some additional fragmentation and dispersal processing. They tend to avoid data replication, caring about the storage costs.

2.4 Exploiting data structures, multi-level confidentiality, and machine trustworthiness

In a case where the data structure is known, the fragmentation process can proceed by taking into account the varying need for secrecy along different

subsets of the input data. Confidential data can be separated from the non-sensitive part of the information and consequently different type of processing can be applied to these two data subsets; for instance only confidential data could be encrypted saving computations. Moreover, such fragmentation could allow a reduction in storage costs as there is no need to provide a specific secure architecture (often expensive) in order to store a piece of data that does not reveal anything confidential to an attacker. The idea of structure-wise fragmentation was first proposed at the end of the last century by the authors of the object-oriented fragmentation-redundancy-scattering [FDRR92] technique. It was later modified to suit database storage [ABG+05, CVF+10] and cloud computing technology [BCCBF13, DCdVEF+14, HIK+13].

Separating data along its confidentiality levels is simple to imagine but not easy to implement. The decomposition process often requires user interaction and cannot be automatized. It is the users' responsibility to provide the rules of confidentiality for each set of data types. It is still possible, however, that even after a careful data fragmentation, a combination of two or more non-confidential data fragments will reveal some confidential information [CVF+10]. Moreover, information from outside could be used to exploit the non-confidential data (for instance, de-anonymizing it). Therefore, a recent research track aims at automatizing the data separation; new ways of separating data in two fragments, confidential and non-confidential, without user interaction were recently developed and presented in [QM15, Han18]. These were inspired by selective encryption and use the discrete cosine and discrete wavelet transforms.

2.4.1 Object-oriented Data Fragmentation

In the 90s, authors of the fragmentation-redundancy-scattering (FRS) technique (described in Section 2.3.3) introduced an object-oriented version that could be used to fragment applications composed of objects [FDRR92, FP95]. Confidential objects are fragmented using a recursive algorithm until they are broken into fragments that do not reveal any confidential information about the functions of the application. Resilience is then added through the use of error processing techniques (like error correcting codes). According to the authors, adding resilience to objects could also be done by anticipating the application design at the early stage of design. At the end, the fragmented data is scattered over various workstations. Leftover fragments, still holding confidential information (even after being fragmented inside the first step), are encrypted or stored on trusted devices. All remaining pieces are distributed over untrusted sites. Obviously, data processing or defragmentation are performed only on trusted sites.

The object-oriented FRS is hard to implement and thus was not later developed. Indeed, decomposing an application into confidential and non-confidential objects raises a challenge for the developers. Its distribution over multiple nodes may very probably slow down the execution performance.

2.4.2 Database Fragmentation

A database as a service [9] delivers similar functionality to a classic, relational, or NoSQL database management systems while providing flexibility

[9] https://www.technologyreview.com/s/414090/designing-for-the-cloud/

and scalability of a hosted in a cloud on-demand platform. The DBaaS user does not have to be concerned with database provisioning issues; it is the cloud provider's responsibility to maintain, backup, upgrade and handle the physical failures of the database system. Therefore, it is easy to see that simplicity and cost effectiveness are the biggest advantages of such a solution.

At the same time, owners lose control over their outsourced data. Such situations create new security and privacy risks, especially when the stored data contains sensitive information like health or financial records. Consequently, securing database services has become a need of paramount importance. A straightforward solution to the problem lies in the encryption of the whole database, successfully implemented in [PRZB12]. However, performance overhead and query processing limitations may be the drawbacks of such a blunt approach. Moreover, the ability of encrypted databases to provide provable security guarantees is sometimes questioned [GRS17].

A way of protecting the privacy of a database without using encryption is seen in anonymization techniques aimed at guaranteeing the *k-anonymization* [Swe02], *t-closeness* [LLV07] and *l-diversity* [MKGV07] of records inside a database. They allow the release of a non-encrypted database containing personal information while ensuring some degree of individual privacy. Some progress on this subject, mainly for health data, has recently been performed among others in [BCBC+14].

Database fragmentation promises an interesting alternative to full database encryption or full anonymization. One of the first works on the subject [ABG+05], introduces a distributed architecture for preserving data privacy. End users communicate through a trusted client (as all of the presented sys-

tems must include at least one trusted element in their architecture - the one in which the fragmentation and defragmentation processes will occur) with two non-trusted servers belonging to two different storage providers (as presented in Figure 2.3). The use of different storage providers ensures the physical separation of the information being protected. By construction, storage providers do have access to the information that users entrust them with. Even if they are well aware that they should not incorrectly interact with the user's data and its integrity without endangering their own business, it is a common assumption to suppose them to be *honest-but-curious*; they have the ability to observe, move, and replicate stored data, especially behind the virtualization mechanism. In [ABG+05], the outsourced data is partitioned among the two untrusted servers in a way that content at any one server does not breach data privacy. In order to obtain valuable information, an adversary must gain access to both databases. By analogy, the system is also protected from insider attacks and the curiosity of the providers as long as they do not ally together. On top of that, queries involving only one of the fragments are executed much more efficiently than on encrypted data.

Another work [CVF+10, DCdVEF+14] protects sensitive information by mixing encryption and fragmentation. It defines confidentiality constraints as a subset containing one or more relation attributes. A constraint involving only one attribute implies that the value of the attribute is sensitive and the only way of protecting it is the use of encryption. On the other hand, multi-attribute constraints specify that only associations between attributes of a given constraint are sensitive. In that case, there is no need to encrypt all the attributes values because confidentiality can be ensured by fragmentation.

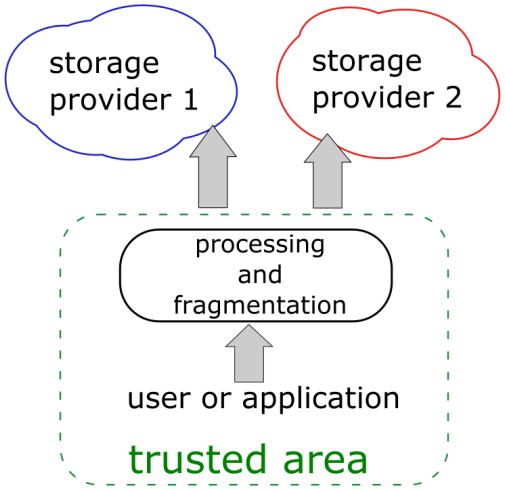


Figure 2.3: *Fragmentation of the database is performed inside a trusted area [ABG+05] and database fragments are then dispersed over two independent storage providers.*

In [CVF+10, DCdVEF+14], three scenarios of fragmenting a relation are presented. In the first one, a relation is divided into two fragments which does not contain sensitive combination of unencrypted attributes. In the second scenario, the relation is split into multiple fragments in a way that any query can always be evaluated on one of the fragments; each fragment contains unencrypted attributes that do not violate confidentiality constraints, as well as the encrypted representation of all other attributes. The last fragmentation scenario avoids the use of encryption by introducing a trusted area (belonging to the data owner) for the storage of sensitive portion of data.

For each scenario, the authors present fragmentation metrics supporting the definition of an appropriate fragmentation algorithm. Fragmentation metrics can aim at minimizing the number of fragments, maximizing affinity

between attributes stored in one fragment or minimizing querying costs.

Recently, Bkakria [BCCBF13] generalized this approach to a database containing multiple relations. It introduced a new confidentiality constraint for the protection of relationships between two tables. Sensitive associations between relations are secured by the protection of primary key/foreign key relationships and the separation of the involved relations. Relations are transformed into secure fragments in which subsets of attributes respecting confidentiality constraints are stored in plaintext while all others are encrypted. Bkakria introduced a parameter for evaluating the query execution cost and proposed a query transformation and optimization model for executing queries on distributed fragments. He also focuses on the issue of preserving data unlinkability while executing queries on multiple fragments. Indeed, providers have to build a coalition and then deduct information by observing query execution. To avoid such situations, he proposes the use of an improved Private Information Retrieval (PIR) [OG10] technique which allows querying a database without revealing query results to service providers. Results of implementation of the proposed approach are presented. Although the modified PIR solution is much faster than its predecessor, the processing time of record retrieval from multiple fragments is considerably slower in comparison with querying a single fragment.

The idea of splitting a database into fragments stored at different cloud providers was also proposed by Hudic [HIK$^+$13]. In his approach, a database is first normalized and then several security levels (high, medium, low) are attributed to relations. Based on these three levels and specific user requirements, data is encrypted, stored at local domain or distributed between

providers.

Database fragmentation methods presented in this thesis remain limited by the fact that the number of fragments does not exceed a few dozen. Moreover, in each case, the proposed fragmentation algorithms require user interaction in order to define the data confidentiality level.

2.5 Issues and Recommendations

Several issues have to be taken into account while designing a storage system basing its data protection on fragmentation and dispersal. First, a right separation of the fragments has to be ensured. A situation where data is fragmented but there is no control over the fragments locations is a weak solution as it does not guarantee that the dispersal will constitute a sufficient obstacle for an attacker. Indeed, the majority of cloud providers use virtualization which prevents the end user from such control. Dispersing data over independent providers is a rapid solution but it can entail an increase in latency costs [BCQ+13, HIK+13]. A single provider possessing several storage sites, like the Amazon S3, may also be used. Such coarse-grained solutions are suitable for users looking for simplicity. A more sophisticated means of data dispersion would use bare-metal clouds like Rackspace OnMetal [10], that abandoned virtualization and thus allows full control over the physical location of the stored data.

A suggested dispersal strategy is presented in Figure 2.4. A small amount of confidential information is stored inside trusted devices. Fragments of data

[10] https://www.rackspace.com/cloud/servers/onmetal

that are not confidential but could reveal some information are gathered and dispersed over several physically separated servers or independent cloud providers. A large amount of non-confidential data then goes to a public cloud as it this the most cost efficient solution.

Fragmentation may increase latency inside the storage system as it usually requires additional data processing, multiple transmission channels, and in some cases it increases the amount of data to be stored and transmitted. However, parallelization of the processing may be applied in order to compensate for the performance overhead coming from the fragmentation. Fragments over-requesting as well as keeping track of the most responsive nodes also help to accelerate the data reconstruction process. When performance becomes a critical issue, lightweight fragmentation techniques like the FSFA (proposed in the Chapter 5) could be applied. A pre-processing step consisting in structurewise fragmentation may also help to speed up the overall processing as it allows the extraction and protection solely of the confidential portion of the stored information.

Last but not least, structurewise fragmentation strongly depends on user guidance for the definition of the confidentiality levels and consequently on the data nature. Designing an algorithm for automatically or semi-automatically separating confidential data from non-sensitive pieces would make the structurewise fragmentation process much faster and easier to use. This last idea has been successfully developed by Qiu [QM15] for the selective encryption of images using a general purpose GPU.

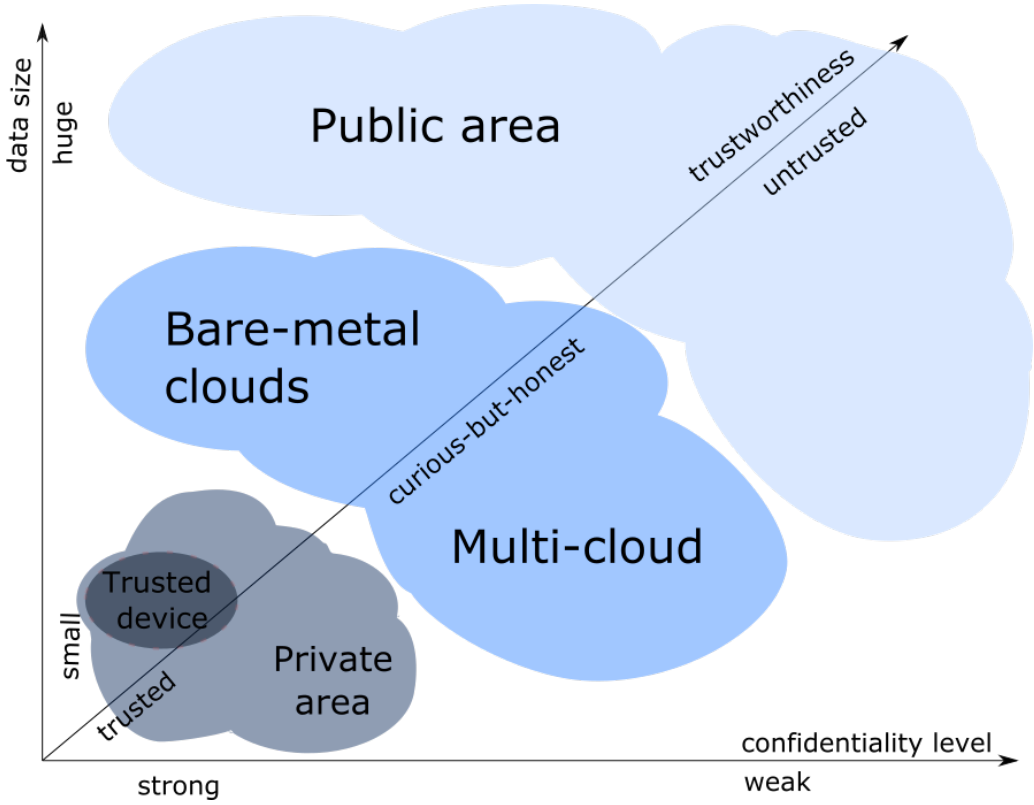


Figure 2.4: *Dispersing data according to the trustworthiness of the storage devices. First, a small portion of sensitive data is kept inside a private area. Second, fragments of data are dispersed over a multi-cloud environment or physically separated servers. Such fragments do not leak confidential information unless the are gathered. Third, a large amount of non-confidential data is stored inside a public untrusted area..*

2.6 Summary

In this chapter, related works from the domain of data protection by means of fragmentation were presented. Fragmentation is divided into two cate-

gories: bitwise and structurewise. Bitwise fragmentation regroups all kind of data fragmentation techniques operating on data without regards for the data structure. Structurewise fragmentation gathers methods that fragments data into fragments of different confidentiality levels by analyzing their structure. Relevant works are organized into two groups according to those two definitions.

The first section of the survey focuses on bitwise fragmentation. Relevant techniques are presented including secret sharing, information dispersal, as well as multiple schemes based on symmetric encryption used in recent distributed systems. They are compared in terms of storage requirements, performance, and provided level of data confidentiality. Performance benchmark help the positioning of the techniques. Several elements proper to bitwise fragmentation systems are portrayed like data resilience, key and fragments' location management, integrity, data defragmentation, trustworthiness of the machines, the concept of decoys, fragment size, and data deduplication. Finally, descriptions of eight selected storage systems applying bitwise fragmentation techniques are presented.

The second section gathers works applying structurewise fragmentation: the historical object-oriented fragmentation-redundancy-scattering and various proposals connected with fragmentation of relational databases, including database anonymization and private information retrieval.

Bibliography

[ABG+05] G. Aggarwal, M. Bawa, P. Ganesan, H. Garcia-molina, K. Kenthapadi, R. Motwani, U. Srivastava, D. Thomas, and Y. Xu. Two can keep a secret: A distributed architecture for secure database services. In *In Proc. CIDR*, 2005.

[BCBC+14] Maxime BERGEAT, Nora CUPPENS-BOULAHIA, Frédéric Cuppens, Noémie JESS, Françoise DUPONT, Said Oulmakhzoune, and Gaël De Peretti. A French Anonymization Experiment with Health Data. In *PSD 2014 : Privacy in Statistical Databases*, Eivissa, Spain, September 2014.

[BCCBF13] Anis Bkakria, Frédéric Cuppens, Nora Cuppens-Boulahia, and José M. Fernandez. *Confidentiality-Preserving Query Execution of Fragmented Outsourced Data*, pages 426–440. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[BCQ+13] Alysson Bessani, Miguel Correia, Bruno Quaresma, Fernando André, and Paulo Sousa. Depsky: Dependable and secure storage in a cloud-of-clouds. *Trans. Storage*, 9(4):12:1–12:33, November 2013.

[BDCdVF+16] Enrico Bacis, Sabrina De Capitani di Vimercati, Sara Foresti, Stefano Paraboschi, Marco Rosa, and Pierangela Samarati. Mix&slice: Efficient access revocation in the cloud. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer*

and Communications Security, CCS '16, pages 217–228, New York, NY, USA, 2016. ACM.

[BGJ+13] J. M. Bohli, N. Gruschka, M. Jensen, L. L. Iacono, and N. Marnau. Security and privacy-enhancing multicloud architectures. *IEEE Transactions on Dependable and Secure Computing*, 10(4):212–224, July 2013.

[Bla79] George R. Blakley. Safeguarding Cryptographic Keys. In *Proceedings of the 1979 AFIPS National Computer Conference*, volume 48, pages 313–317, June 1979.

[BLU+15] William J. Buchanan, David Lanc, Elochukwu Ukwandu, Lu Fan, Gordon Russell, and Owen Lo. The future internet: A world of secret shares. *Future Internet*, 7(4):445, 2015.

[BM85] George Blakley and Catherine Meadows. *Security of Ramp Schemes*, pages 242–268. Springer Berlin Heidelberg, Berlin, Heidelberg, 1985.

[Boy99] Victor Boyko. On the security properties of oaep as an all-or-nothing transform. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '99, pages 503–518, London, UK, UK, 1999. Springer-Verlag.

[BR95] Mihir Bellare and Phillip Rogaway. *Optimal asymmetric encryption*, pages 92–111. Springer Berlin Heidelberg, Berlin, Heidelberg, 1995.

[BSL+06] Rudi Ball, Vicki Spurrett, Rogério De Lemos, Rudi Ball, Vicki Spurrett, and Rogério De Lemos. Dependable and secure storage in pervasive peer-to-peer systems, 2006.

[CBHK15] P. Cincilla, A. Boudguiga, M. Hadji, and A. Kaiser. Light blind: Why encrypt if you can share? In *2015 12th International Joint Conference on e-Business and Telecommunications (ICETE)*, volume 04, pages 361–368, July 2015.

[CLM17] Liqun Chen, Thalia M. Laing, and Keith M. Martin. Revisiting and extending the aont-rs scheme: A robust computationally secure secret sharing scheme. In Marc Joye and Abderrahmane Nitaj, editors, *Progress in Cryptology - AFRICACRYPT 2017*, pages 40–57, Cham, 2017. Springer International Publishing.

[CP11] A. B. Chougule and G.A. Patil. Implementation & analysis of efrs technique for intrusion tolerance in distributed systems. *International Journal of Computer Science Issues*, 8(1), July 2011.

[CVF+10] Valentina Ciriani, Sabrina De Capitani Di Vimercati, Sara Foresti, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. Combining fragmentation and encryption to protect privacy in data storage. *ACM Trans. Inf. Syst. Secur.*, 13(3):22:1–22:33, July 2010.

[DBF91] Y. Deswarte, L. Blain, and J. C. Fabre. Intrusion tolerance in

distributed computing systems. In *Proceedings. 1991 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 110–121, May 1991.

[DCdVEF+14] Sabrina De Capitani di Vimercati, Robert F. Erbacher, Sara Foresti, Sushil Jajodia, Giovanni Livraga, and Pierangela Samarati. *Encryption and Fragmentation for Data Confidentiality in the Cloud*, pages 212–243. Springer International Publishing, Cham, 2014.

[Des00] Anand Desai. The security of all-or-nothing encryption: Protecting against exhaustive key search. In *Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '00, pages 359–375, London, UK, UK, 2000. Springer-Verlag.

[DKWGH83] Ehud D. Karnin, Jonathan W. Greene, and Martin Hellman. On secret sharing systems. *Information Theory, IEEE Transactions on*, IT-29:35 – 41, 02 1983.

[Dwo01] Morris J. Dworkin. Nist sp 800-38a, recommendation for block cipher modes of operation: Methods and techniques. Technical report, United States, 2001.

[FDP86] J. M. Fray, Y. Deswarte, and D. Powell. Intrusion-tolerance using fine-grain fragmentation-scattering. In *1986 IEEE Symposium on Security and Privacy*, pages 194–194, April 1986.

[FDRR92] Jean-Charles Fabre, Yves Deswarte, Brian Randall, and Brian R. Designing secure and reliable applications using fragmentation-redundancy-scattering: an object-oriented approach, 1992.

[FP95] J. C. Fabre and T. Perennou. Fragmentation of confidential objects for data processing security in distributed systems. In *Proceedings of the Fifth IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems*, pages 395–403, Aug 1995.

[GKB+01] G. R. Ganger, P. K. Khosla, M. Bakkaloglu, M. W. Bigrigg, G. R. Goodson, S. Oguz, V. Pandurangan, C. A. N. Soules, J. D. Strunk, and J. J. Wylie. Survivable storage systems. In *DARPA Information Survivability Conference amp; Exposition II, 2001. DISCEX '01. Proceedings*, volume 2, pages 184–195 vol.2, 2001.

[GRS17] Paul Grubbs, Thomas Ristenpart, and Vitaly Shmatikov. Why your encrypted database is not secure. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*, HotOS '17, pages 162–168, New York, NY, USA, 2017. ACM.

[Han18] Qiu Han. An efficient data protection architecture based on fragmentation and encryption. *CoRR*, abs/1803.04880, 2018.

[HIK+13] Aleksandar Hudic, Shareeful Islam, Peter Kieseberg, Sylvi Rennert, and Edgar R. Weippl. Data confidentiality using

fragmentation in cloud computing. *International Journal of Pervasive Computing and Communications*, 9(1):37–51, 2013.

[Joh07] R. Johnson. The unisys stealth solution and secureparser: A new method for securing and segregating network data. Unisys Corporation white paper, 2007.

[KK03] Kamil Kulesza and Zbigniew Kotulski. On automatic secret generation and sharing for karin-greene-hellman scheme. In *Artificial intelligence and security in computing systems*, pages 227–238. Springer, 2003.

[KM18a] K. Kapusta and G. Memmi. Enhancing data protection with a structure-wise fragmentation and dispersal of encrypted data. In *The 17th International Joint Conference on Trust, Security and Privacy in Computing and Communications (IEEE TrustCom)*, August 2018.

[KM18b] Katarzyna Kapusta and Gerard Memmi. Poster: Circular aon: A very fast scheme to protect encrypted data against key exposure. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18. ACM, 2018.

[KM18c] Katarzyna Kapusta and Gerard Memmi. Selective all-or-nothing transform: Protecting outsourced data against key exposure. In *Proceedings of the 10th International Sympo-*

sium on Cyberspace Safety and Security, CSS '18. Springer, 2018.

[Kra94] Hugo Krawczyk. Secret sharing made short. In *Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '93, pages 136–146, London, UK, 1994. Springer-Verlag.

[KSLC17] G. O. Karame, C. Soriente, K. Lichota, and S. Capkun. Securing cloud data under key exposure. *IEEE Transactions on Cloud Computing*, pages 1–1, 2017.

[Li12] Mingqiang Li. On the confidentiality of information dispersal algorithms and their erasure codes. *CoRR*, abs/1206.4123, 2012.

[LLV07] N. Li, T. Li, and S. Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *2007 IEEE 23rd International Conference on Data Engineering*, pages 106–115, April 2007.

[LQLL14] Mingqiang Li, Chuan Qin, Patrick P. C. Lee, and Jin Li. Convergent dispersal: Toward storage-efficient security in a cloud-of-clouds. In *Proceedings of the 6th USENIX Conference on Hot Topics in Storage and File Systems*, HotStorage'14, pages 1–1, Berkeley, CA, USA, 2014. USENIX Association.

[LQLL16] M. Li, C. Qin, J. Li, and P. P. C. Lee. Cdstore: Toward reliable, secure, and cost-efficient cloud storage via convergent dispersal. *IEEE Internet Computing*, 20(3):45–53, May 2016.

[MKGV07] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. L-diversity: Privacy beyond k-anonymity. *ACM Trans. Knowl. Discov. Data*, 1(1), March 2007.

[OG10] Femi Olumofin and Ian Goldberg. Privacy-preserving queries over relational databases. In *Proceedings of the 10th International Conference on Privacy Enhancing Technologies*, PETS'10, pages 75–92, Berlin, Heidelberg, 2010. Springer-Verlag.

[PBL91] D. Powell, I. Bey, and J. Leuridan, editors. *Delta Four: A Generic Architecture for Dependable Distributed Computing*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1991.

[PRZB12] Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. Cryptdb: Processing queries on an encrypted database. *Commun. ACM*, 55(9):103–111, September 2012.

[QD02a] Sean Quinlan and Sean Dorward. Venti: A new approach to archival storage. In *Proceedings of the Conference on File and Storage Technologies*, FAST '02, pages 89–101, Berkeley, CA, USA, 2002. USENIX Association.

[QD02b] Sean Quinlan and Sean Dorward. Venti: A new approach to archival storage. In *Proceedings of the Conference on File and Storage Technologies*, FAST '02, pages 89–101, Berkeley, CA, USA, 2002. USENIX Association.

[QM15] Han Qiu and Gerard Memmi. Fast selective encryption methods for bitmap images. *Int. J. Multimed. Data Eng. Manag.*, 6(3):51–69, July 2015.

[Rab89] Michael O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *J. ACM*, 36(2):335–348, April 1989.

[Riv97] Ronald L. Rivest. All-or-nothing encryption and the package transform. In *In Fast Software Encryption, LNCS*, pages 210–218. Springer-Verlag, 1997.

[RP11] Jason K. Resch and James S. Plank. Aont-rs: Blending security and performance in dispersed storage systems. In *Proceedings of the 9th USENIX Conference on File and Stroage Technologies*, FAST'11, pages 14–14, Berkeley, CA, USA, 2011. USENIX Association.

[RS60] I. S. Reed and G. Solomon. Polynomial Codes Over Certain Finite Fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.

[SB05] Arun Subbiah and Douglas M. Blough. An approach for fault tolerant and secure data storage in collaborative work

environments. In *Proceedings of the 2005 ACM Workshop on Storage Security and Survivability*, StorageSS '05, pages 84–93, New York, NY, USA, 2005. ACM.

[Sch99] Berry Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '99, pages 148–164, London, UK, UK, 1999. Springer-Verlag.

[SGMV09] Mark W. Storer, Kevin M. Greenan, Ethan L. Miller, and Kaladhar Voruganti. Potshards: a secure, recoverable, long-term archival storage system. *Trans. Storage*, 5(2):5:1–5:35, June 2009.

[SGS+00] John D. Strunk, Garth R. Goodson, Michael L. Scheinholtz, Craig A. N. Soules, and Gregory R. Ganger. Self-securing storage: Protecting data in compromised system. In *Proceedings of the 4th Conference on Symposium on Operating System Design & Implementation - Volume 4*, OSDI'00, Berkeley, CA, USA, 2000. USENIX Association.

[Sha49] C. E. Shannon. Communication theory of secrecy systems. *The Bell System Technical Journal*, 28(4):656–715, Oct 1949.

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979.

[SJH05] Sabre A. Schnitzer, Robert A. Johnson, and Henry Hoyt. Secured storage using secureparser^{TM}. In *Proceedings of the 2005 ACM Workshop on Storage Security and Survivability*, StorageSS '05, pages 135–140, New York, NY, USA, 2005. ACM.

[Swe02] Latanya Sweeney. K-anonymity: A model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5):557–570, October 2002.

[TG11a] B. Tabbara and P. Garg. Shared community storage network. patent application number: 20100020718, patent number: 7869383, Jan 2011.

[TG11b] B. Tabbara and P. Garg. Shared community storage network. continuation patent application number: 20110246652, Oct 2011.

[TG14] B. Tabbara and P. Garg. Shared community storage network. continuation patent application number: 20140143425, May 2014.

[WBP+01] Jay J Wylie, Mehmet Bakkaloglu, Vijay Pandurangan, Michael W Bigrigg, Semih Oguz, Ken Tew, Cory Williams, Gregory R Ganger, and Pradeep K Khosla. Selecting the right data distribution scheme for a survivable storage system (cmu-cs-01-120). 2001.

[WBS+00] Jay J. Wylie, Michael W. Bigrigg, John D. Strunk, Gregory R. Ganger, Han Kiliççöte, and Pradeep K. Khosla. Survivable information storage systems. *Computer*, 33(8):61–68, August 2000.

Chapter 3

Fragmentation as a way of protecting encrypted data against key exposure

3.1 Introduction and Motivation

The rapid growth of the cloud data storage market raises both security challenges and opportunities. On the one hand, cloud providers deal with a large number of external attacks on a daily basis. Each major data leak is loudly reported in the media, damaging their reputation. On top of that, the threat from a user point of view may not only come from the outside but also from a curious, malicious, or careless insider. On the other hand, never before could users access so many storage devices at such a low cost. This highly distributed nature of cloud storage opens up new possibilities for strengthening data protection: data can be fragmented and dispersed over a large

number of servers on independent sites [BCQ+13, BLU+15]. Such processing not only slows down an external attacker but also enhances users' privacy against the misuse of their personal data (as with the infamous Facebook and Cambridge Analytica case [1]) as it limits the possibility of data exploitation by a malicious insider or of compromising data confidentiality by an honest-but-curious storage provider.

As presented in Chapter 2, several recent storage solutions already fragment and disperse encrypted data to reinforce the protection level [KM15]. However, the majority of them do not pay much attention to the way the fragments are constructed from encrypted data. Fragmentation is usually performed in a straightforward manner where fragments are formed from large consecutive chunks of encrypted data. Such processing does not protect against powerful adversaries in possession of encryption keys with access to a subset of the storage domains (and thus able to decrypt a part of the dispersed information).

Nowadays, key exposure becomes a real threat [KSLC17]. It may be the result of an easily guessable or reproducible key generation but even relying on secure mechanisms may be insufficient as a secure key may be acquired in various ways, e.g., using backdoors, bribes, or coercion. Moreover, when we consider data with long life cycles (like archive data that are supposed to be kept for decades or even more), the length of the encryption key which may be recommended when data are first stored may not be sufficient anymore a decade later due to progress in hardware development and cryptanalysis.

In order to prevent an attacker in possession of the encryption key from

[1] https://en.wikipedia.org/wiki/Facebook-Cambridge_Analytica_data_scandal

decrypting even part of a ciphertext, data can be fragmented into a set of k fragments (needed for data reconstruction). Once the fragments are dispersed over two or more independent storage sites, the data is protected against an attacker unable to gain access to the totality of the storage sites. This is true even if the attacker managed to obtain the encryption key as a partial data decryption (decrypting just the portion of data contained inside a single fragment) is impossible.

To make the decryption of less than the totality of data fragments impossible, an all-or-nothing transform can be applied as a pre-processing step [Riv97] before data encryption. Alternatively, an information dispersal algorithm can be used to form the final fragments from encrypted data [Rab89] or a linear transform may be applied over the encrypted data to create dependencies between the ciphertext blocks [KSLC17]. These three methods reinforcing data protection come with a performance cost. The all-or-nothing transform in particular leads to a considerable decrease in performance (as it requires at least two rounds of block cipher encryption). A linear transform achieves better performance but still requires some operations in addition to data encryption. Finally, information dispersal lacks terribly in scalability.

Motivated by recent attacks on user's privacy, this chapter treats the problem of protecting outsourced data against cryptographic material exposure. It introduces three fast schemes reinforcing confidentiality of fragmented data: the Secure Fragmentation and Dispersal (SFD), the Circular All-or-Nothing (CAON), and the Selective All-or-Nothing (SAON) scheme.

The Secure Fragmentation and Dispersal scheme is a method applied after data encryption in order to create fragments resistant to the exposure of

the encryption key. It operates on a ciphertext obtained with block cipher encryption with a mode of operation creating dependencies between consecutive ciphertext's blocks. In contrast to SFD, the Circular All-or-Nothing (CAON) can be applied over any kind of ciphertext. Both SFD and CAON significantly increase the computational effort that an attacker would have to apply in order to recover even a single block from an incomplete set of data fragments. In contrast to similar techniques, they do not compromise performance or scalability in processing. Finally, the Selective All-or-Nothing (SAON) scheme adresses the needs of users that cannot or do not want to use multiple storage providers. It has similar properties with SFD and CAON but in a single cloud scenario.

3.2 Secure Fragmentation and Dispersal

The main idea behind Secure Fragmentation and Dispersal (SFD, published in [KM18a]) is to derive from known properties of block ciphers (and more particularly from their chaining modes of operation), a new method for the fragmentation and dispersal of encrypted data that could produce fragments resistant to the exposure of the cryptographic material. In SFD, encrypted data are organized into a set of fragments in two steps. Each step increases the difficulty of a brute-force search that an attacker in the possession of the encryption key and some of the fragments would have to perform in order to decrypt even part of the initial data. SFD is an easy and efficient alternative to the family of information dispersal algorithms and to the all-or-nothing transform pre-processing. It can be seen as a generalized methodology of

data parsing [SJH05]. When integrated within existing techniques, it enriches them with an additional layer of data protection by securing encrypted data in a situation of key exposure.

3.2.1 Data Concepts, Notation, and Prerequisites

The proposed approach identifies the following basic data structures that mostly correspond to classical concepts concerning a symmetric block cipher [Dwo01]:

- **Block (P or C)**: a sequence of bits of size $|B|$ corresponding to the classical concept of block. A plaintext block is denoted as P while a ciphertext block is denoted as C.
- **Sub-block (SB)**: a sequence of bits of size $|SB|$ contained in a block.
- **Plaintext ($PLAIN$)**: initial data composed of p plaintext blocks (already padded if needed).
- **Ciphertext ($CIPH$)**: encrypted plaintext composed of c ciphertext blocks. $c = p + 1$ as an initialization vector is added at the beginning of the ciphertext.
- **Share ($SHARE$)**: result of the first fragmentation step in the SFD method. An intermediary fragment; it is composed of $\frac{c}{k_1}$ blocks.
- **Fragment (F)**: result of the final fragmentation step. Each fragment is composed of $\frac{c}{k_1}$ sub-blocks.

Notations

Plaintext $PLAIN$ is composed of p input blocks $P_1, ..., P_p$. It is encrypted into ciphertext $CIPH$ composed of $c = p + 1$ blocks $C_0, C_1, ..., C_{c-1}$, where C_0 corresponds to the initialization vector. A ciphertext block C_i comes from the encryption of the plaintext block P_i (except for the pseudo-random initialization vector that is just appended as the first block C_0). In the first fragmentation step, blocks of the ciphertext $CIPH$ are dispersed over k_1 shares $SHARE_0, ..., SHARE_{k_1-1}$. In the second fragmentation step, a share $SHARE_j$ is fragmented into k_2 fragments $F_0^j, ..., F_{k_2-1}^j$. Each block is composed of k_2 sub-blocks. A sub-block l inside a block C_i is denoted as $C_i(l), l = 0, \ldots, k_2 - 1$.

Prerequisites

The ciphertext $CIPH$ is to be obtained using a block cipher with a mode of operation creating dependencies between consecutive blocks (like the widely used Cipher Block Chaining). The motivation behind this requirement will be detailed in Section 3.2.2.

Simply put, it is supposed that k_1 is a divisor of the number of blocks inside the ciphertext c and that the size of the sub-block $|SB|$ is a divisor of the size of the block $|B|$. This way, all the fragments will be of equal size. If this requirement is not fulfilled, two possibilities could be applied. One would consist in having fragments of different sizes. Another would require the use of padding. This would guarantee that the fragments are of equal size.

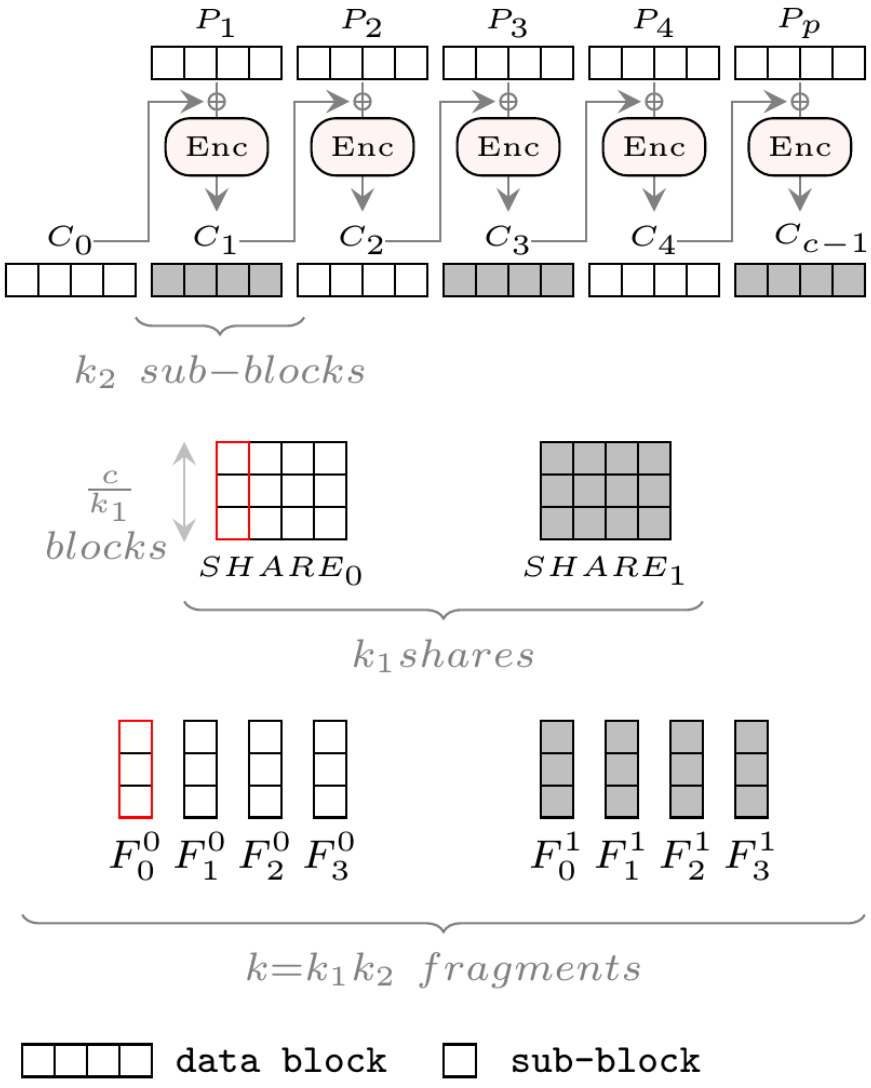


Figure 3.1: *Data fragmentation, example for $k_1 = 2$ and $k_2 = 4$. Plaintext blocks are encrypted into corresponding ciphertext blocks using the CBC mode. Consecutive ciphertext blocks are dispersed over k_1 different shares. Each of the k_2 sub-blocks of a block is dispersed to a different final fragment.*

3.2.2 Description of the Scheme

A pseudo-code of the SFD scheme is presented in Figure 3.2. The goal of this process is to shred data into fragments making data recovery from an incomplete set of fragments as hard as possible without leading to decreased performance. The difficulty of the recovery is measured in the feasibility of a brute-force search attack with less than the required number of fragments.

Splitting Ciphertext into Shares

A symmetric block cipher transforms (encrypt or decrypt) a fixed-length group of bits called block. To operate on data larger than a single block, an operation that repeatedly applies the single-block procedure - named mode of operation - has to be defined. The simplest mode of operation, the Electronic Code Book, encrypts each of the ciphertext blocks separately. Such processing was proven to be insecure as it transforms identical plaintext blocks into identical ciphertext blocks. One way of dealing with this lack of diffusion is re-using the output of the encryption of a block for the encryption of the next block (for the first block that does not have a predecessor, an initialization vector is generated that plays the role of the previous block). In such mode of operation, e.g., in Cipher Block Chaining, it is infeasible for an adversary in possession of the encryption key to decrypt C_i without possessing C_{i-1}. Indeed, C_{i-1} can take on any possible values in $\{0,1\}^{|b|}$ as it is an output of a secure block cipher (or a pseudo-random initialization vector); consequently, P_i could take any of the $2^{|b|}$ possibilities depending on C_{i-1}. Therefore, it is *infeasible* for the adversary to recover a set of non-consecutive ciphertext blocks without possessing their predecessors. This leads to the definition of

the following dispersal property:

Property 1 (Dispersal of blocks) *If a ciphertext was obtained using a mode of operation implying chaining between blocks, then dispersing its consecutive blocks over separate shares makes the data recovery infeasible for an adversary that does not have access to consecutive shares.*

The first step of the SFD scheme disperses the ciphertext blocks over shares to comply with the Dispersal Property 1. More precisely, c blocks of the ciphertext c are distributed over k_1 shares $share_0, \ldots, share_{k_1-1}$ in a way that C_i is assigned to the share $share_j \iff i \bmod k_1 = j$. A share $SHARE_j$ contains all the predecessor blocks of those contained in share $SHARE_{(j+1) \bmod k_1}$ and it is therefore necessary for recovering blocks inside $SHARE_{(j+1) \bmod k_1}$. This proves that the fragmentation function presented in Figure 3.2 verifies our Dispersal Property 1.

1: **function** FRAGMENTATION($CIPH, k_1, k_2$)
2: **for** each block C_i inside the ciphertext $CIPH$ **do**
3: Compute share index $j = i \bmod k_1$
4: Disperse C_i to $SHARE_j$
5: **for** $l = 0, \ldots, k_2 - 1$ **do**
6: Disperse $C_i\,(l)$ to fragment F_l^j

Figure 3.2: *Pseudo-code of the fragmentation function transforming a ciphertext $CIPH$ into $k = k_1 k_2$ fragments in a single pass. For each block a share index is computed indicating to which of k_1 shares the block belongs. Then, sub-blocks of the block are dispersed over k_2 fragments.*

Splitting Shares into Fragments

A symmetric cryptographic function operating on blocks guarantees that a ciphertext block depends on every bit of the corresponding plaintext block. Moreover, modern symmetric encryption functions, like AES, guarantee that the absence of i bits from the plaintext block and of o bits from the ciphertext block does not permit to easily recover the plaintext (or the ciphertext) block. A brute-force search generating and verifying all the $2^{min(i,o)}$ possible configurations for the missing bits would have to be performed [ABM14, BDCdVF+16]. Therefore, the following property can be formulated:

Property 2 (Dispersal of sub-blocks) *Dispersing sub-blocks of ciphertext blocks contained inside a share over separate fragments increases the difficulty of the ciphertext decryption for an adversary possessing an incomplete set of fragments belonging to consecutive shares.*

This property is behind the motivation of the second step of the SFD scheme. A block C_i belonging to the share $SHARE_j$ and composed of k_2 sub-blocks is dispersed over the k_2 fragments $F_0^j, \ldots, F_{k_2-1}^j$ in a way that a sub-block $C_i(l)$ is assigned to the fragment F_l^j. Thus, each single ciphertext block inside a share is uniformly spread over k_2 fragments. After the splitting of the shares, $k = k_1 k_2$ fragments are obtained that will be dispersed over k different physical locations.

The size of a sub-block does not necessarily have to be a divisor of the size of the block. A solution could be imagined, similar to the data parsing found in [SJH05], where a block is composed of several sub-blocks of different sizes or where bits are spread unevenly over fragments without taking into account

the block structure of the ciphertext. However, this would lead to fragments of different sizes (as well as of different importance since larger fragments would contain more information). Moreover, such processing would require a sort of dispersal key similar to the splitting key used by the Unisys's Secure Parser® (described in Section 2.3.3).

The SFD scheme is particularly adapted for dispersal of data obtained using block cipher encryption with a mode of operation like the Cipher Block Chaining (CBC), the Output Feedback (OFB) or the Cipher Feedback (CFB). Such modes of operation creates dependencies between consecutive data blocks that are exploited by SFD. Indeed, when SFD is applied within the Counter (CTR) mode (that is closer to a stream cipher than to a block cipher), the two dispersal properties are not fully satisfied. Nevertheless, the dispersal of the ciphertext sub-blocks will still slow an attacker from the recovery of consecutive sub-blocks of plaintext. Moreover, a special dispersal procedure could be imagined for the CTR. In the CTR mode, it is not possible to decrypt the ciphertext without possessing the right pseudo-random initialization vector that is used as the counter during the encryption process and is usually appended at the beginning of the ciphertext. Thus, dispersing the initialization vector over the fragments, using, for instance, the Shamir's secret sharing, would slow down an attacker that managed to obtain the encryption key but not the right initialization vector.

Fragment Dispersal

After fragmentation, the ciphertext is transformed into $k = k_1 k_2$ final fragments. The data protection provided by fragmentation is fully enabled only

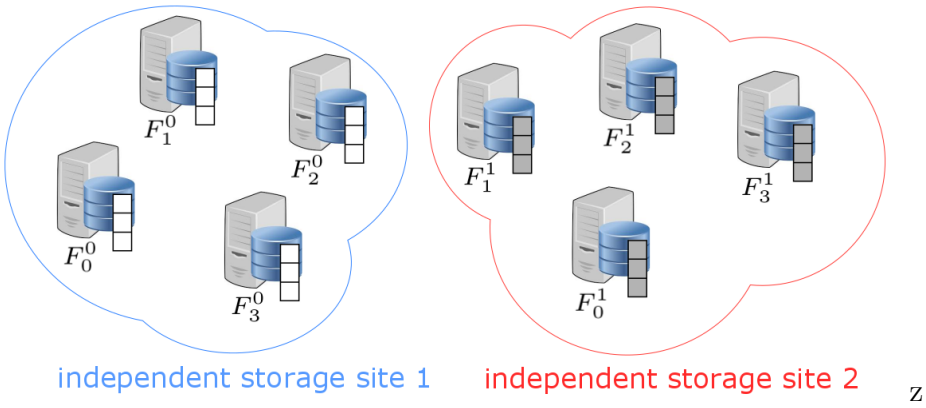
when the fragments are stored in k different physical locations: without the data dispersal, an attacker needs only resolve the fragment order, which is not a significant obstacle.

The simplest fragment distribution is to disperse each fragment to a random location. It is also the only possible way when a user dispose of k physical storage devices that are equally hard to access (like k servers inside the same cluster). Nowadays, a user has the opportunity to disperse data over multiple servers on several independent sites. They can maximize the degree of data protection by following two recommendations:

Recommendation 1 (Dispersing fragments of consecutive shares) *Every two sets of fragments allowing the recovery of consecutive shares (containing consecutive ciphertext blocks) should be dispersed over independent storage sites - e.g., over different cloud providers.*

Recommendation 2 (Dispersing fragments of the same share) *A set of fragments allowing the reconstruction of a share should be dispersed over separate storage locations - e.g., multiple physical servers.*

Figure 3.3 presents an example secure fragment dispersal that follows the two recommendations. In a trivial case when $k_1 = 1$, the scheme becomes identical to a data parsing solution. A choice of $k_2 = 1$ is suitable when a user would like to limit the number of fragments (for instance because they dispose of a limited number of storage locations or they fear latency costs).



z

Figure 3.3: *Dispersal of fragments presented in the example from Figure 3.1. Fragments are dispersed over $k_1 = 2$ independent storage sites. Each storage site spreads its fragments over $k_2 = 4$ physically separated servers.*

3.2.3 Comparison with Relevant Works

The SFD scheme was implemented and compared with relevant works: Secret Sharing Made Short (implemented in two versions: with and without Rabin's Information Dispersal Algorithm), AONT-RS, and Rivest's All-or-nothing scheme.

Implementation Details

All schemes were implemented in JAVA using the following resources: JDK 1.8 on DELL Latitude E6540, X64-based PC running on Intel® Core™ i7-4800MQ CPU @ 2.70 GHz with 8 GB RAM, under Windows 7. The standard *javax.crypto* library was used. Data samples of 200MB were used for each measurement. AES-NI was enabled.

During benchmark tests, relevant algorithms were compared in a $k = n$

configuration (all fragments are necessary for data recovery) as the generation of redundant fragments would have the same impact on each method's performance. In each method, Reed-Solomon systematic error correction codes could be applied to obtain additional $n - k$ fragments. Details of the implementation of each algorithm are as follows:

SSMS: Data are encrypted using AES-CBC with a 128 bits key and fragmented in a straightforward way into k consecutive data chunks. The encryption key is fragmented into k fragments using Shamir's secret sharing and attached to the fragments. For larger data, the time taken on key fragmentation has a negligible impact on the overall performance. Thus, the performance of SSMS is practically equivalent to symmetric data encryption.

SSMS with Rabin's IDA: Data are encrypted using AES-CBC with a 128 bits key and divided into vectors of k bytes. Each data vector is then multiplied by a $k \times k$ dispersal matrix. Fragments are formed from the results of vector multiplication by the dispersal matrix. The encryption key is fragmented using Shamir's scheme and attached to the fragments.

Rivest's AON: Data are first encrypted using an inner cipher: AES-CBC with a 128 bits. A SHA-256 hash of the encrypted data is calculated and exclusive-ored with the encryption key. Data fragments are then re-encrypted using an outer cipher: AES-ECB with a 128 bits key (such insecure mode of operation can be used as the data was already encrypted).

AONT (AONT-RS without the use of RS): Data are encrypted using AES-CBC with a 128 bits key. A SHA-256 hash of the encrypted data is calculated, exclusive-ored with the encryption key, and attached to the

data. Encrypted data are then fragmented in a straightforward way into k consecutive chunks. The processing is similar to the Rivest's AON but skips the second encryption round.

Bastion scheme: Data is encrypted using AES-CBC with a 128 bits key. Bastion's linear transform is then applied over the data, exclusive-oring each ciphertext blocks with all other blocks. Encrypted and transformed ciphertext is then fragmented in a straightforward way into k fragments composed of large chunks of consecutive data bits.

Secure Fragmentation and Dispersal: The SFD scheme was integrated within SSMS and AONT. It replaced the straightforward fragmentation of these two methods. This integration was done in a coarse-grained manner where data encryption and data fragmentation are performed in two different steps. A fine-grained integration would directly interface fragmentation steps after each block encryption. Moreover, it is clear from the algorithm's description that the fragmentation can be parallelized. Fragmentation performance was measured in various configuration of k_1 and k_2. The configuration choice had no visible impact on the scheme's performance.

Comparison with Relevant Works

The performance benchmark presented in Figure 3.4 shows that replacing straightforward fragmentation with the proposed SFD scheme leads to a reasonable constant performance overhead (around 6% percent for the AONT-RS scheme and 11% for SSMS) given the chosen coarse-grained integration. The SFD scheme's was compared to Rivest's all-or-nothing and a SSMS version using Rabin's information dispersal algorithm for data fragmentation.

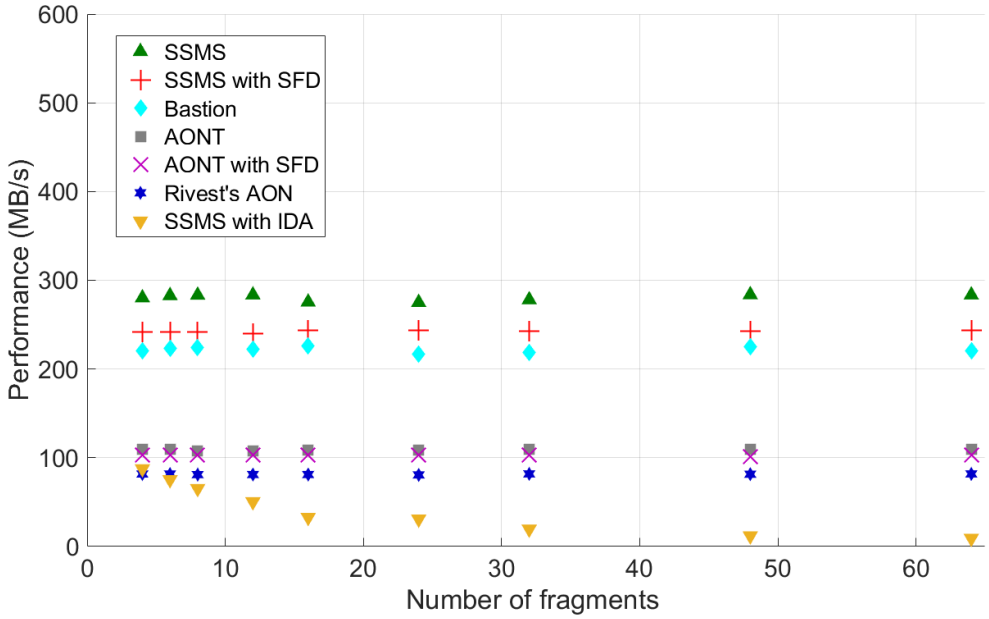


Figure 3.4: *Performance benchmark: throughput of different fragmentation methods in function of the number of fragments k. SSMS and AONT are treated as baseline; they do not provide protection against key exposure. Adding the SFD to the AONT method decreases the performance of the original technique by 5.5%. Adding the SFD to SSMS decreases the performance of the original technique by ∼11%. SFD method is ∼27% faster than the Rivest's all-or-nothing transform and ∼10% faster than the Bastion scheme. In contrast to Rabin's information dispersal, its performance does not decrease with the growing number of fragments.*

SFD is much faster than Rivest's all-or-nothing proposal. Unlike an information dispersal algorithm, SFD is scalable with the number of fragments.

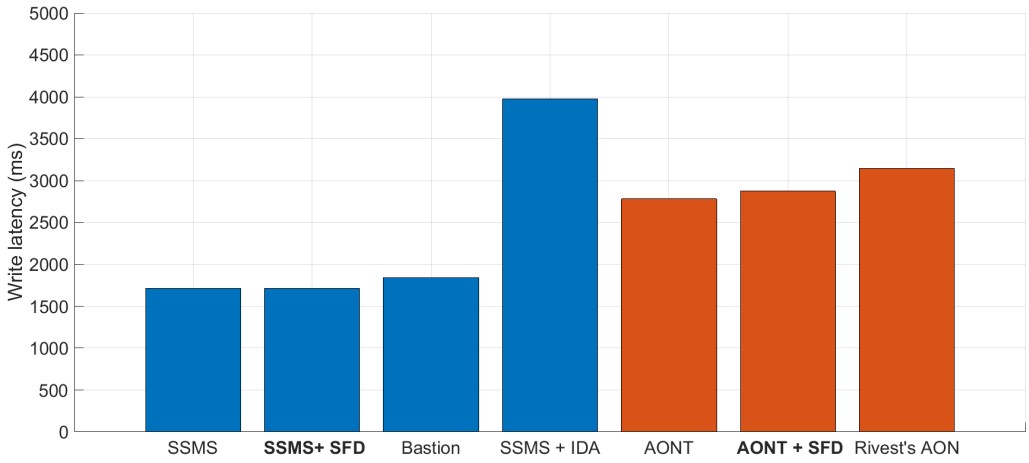


Figure 3.5: *Performance comparison of relevant fragmentation algorithms in an end-to-end scenario (data fragmentation and dispersal over 4 independent Amazon S3 storage sites, $k_1 = 2$ and $k_2 = 4$ for all the algorithms). Tests performed using 200 MB data samples. For the end user, replacing straightforward fragmentation in SSMS or AONT-RS with SFD has an negligible impact on the performance. Strengthening data protection using SFD is faster than using an IDA or the Rivest's all-or-nothing method.*

Impact on the Performance on the Client Side

The performance of relevant techniques was measured in an end-to-end scenario where data was not only fragmented but also dispersed over several independent storage locations. More precisely, algorithms presented in Figure 3.4 were integrated within the DepSky multi-cloud environment [BCQ+13]. Data were fragmented and dispersed over 4 independent Amazon S3 storage sites [2]. Figure 3.5 shows the performance result of the experiment. The

[2] https://aws.amazon.com/s3

performance cost of replacing straightforward fragmentation with SFD is negligible in an end-to-end scenario and therefore unnoticeable for the end user. Indeed, no performance overhead was observed when applying secure dispersal in addition to the AONT-RS technique. Applying secure dispersal in addition to SSMS resulted in a negligible decrease of performance of $\sim$1%. In contrast to SFD, applying an IDA in combination to SSMS decreases the performance twice as much ($\sim$ 95% performance overhead in a $k = 8$ configuration) while the Rivest's pre-processing transform is 12% slower than AONT-RS.

3.3 Circular All-Or-Nothing

The Circular All-or-Nothing (CAON, published in [KM18b]) algorithm addresses the same problem of data protection against key exposure as the Secure Fragmentation and Dispersal (SFD) scheme but in a different way. In contrast to SFD, it is compatible with all kinds of symmetric encryption algorithms. Similar to the Bastion scheme presented in [KSLC17], it transforms the encrypted data by creating dependencies between ciphertext blocks. It improves the Bastion's scheme by reducing the number of exclusive-or operations required in addition to the data encryption to half.

3.3.1 Data Concepts and Notation

The CAON algorithm identifies the following basic data structures that mostly correspond to the classical concepts concerning a symmetric block cipher [Dwo01]:

- **Block (P or C)**: a sequence of bits of size $|B|$ corresponding to the classical concept of block. A plaintext block is denoted as P and a ciphertext block is denoted as C.
- **Transformed ciphertext block (C')**: a transformed ciphertext block after CAON was applied to the ciphertext.
- **Plaintext ($PLAIN$)**: initial data composed of p plaintext blocks (already padded if needed).
- **Ciphertext ($CIPH$)**: encrypted plaintext composed of c ciphertext blocks. $c = p + 1$ as an initialization vector is added at the beginning of the ciphertext.
- **Ciphertext ($CIPH'$)**: the ciphertext $CIPH$ after being transformed using CAON.
- **Fragment (F)**: a fragment, the result of the final fragmentation step. Each fragment is composed of $\frac{c}{k}$ blocks.

Notations

Plaintext $PLAIN$ is composed of p input blocks $P_1, \ldots, P_p$. It is encrypted into ciphertext $CIPH$ composed of $c = p + 1$ blocks $C_0, C_1, \ldots, C_{c-1}$, where C_0 corresponds to the initialization vector. A ciphertext block C_i comes from the encryption of the plaintext block P_i (except for the initialization vector that is appended as the first block C_0). The CAON transform transforms the ciphertext $CIPH$ into $CIPH'$ composed of c blocks $C'_0, C'_1, \ldots, C'_{c-1}$. $CIPH'$ is then fragmented into k fragments $F_0, \ldots, F_{k-1}$.

3.3.2 Description of the Algorithm

CAON operates on a ciphertext $CIPH = C_0, \ldots, C_{c-1}$ coming from the encryption of a plaintext $PLAIN = P_1, \ldots, P_p$ using an encryption key K. The ciphertext $CIPH$ is transformed into $CIPH' = C'_0, \ldots, C'_{c-1}$ inside the function TransformCAON and then fragmented into k fragments $F_0, \ldots, F_{k-1}$ in a way that the decryption of a single fragment is not possible unless all the fragments are gathered. An example illustrating the whole process is shown in Figure 3.9.

Linear transform

The pseudo-code of the linear transform applied to the ciphertext $CIPH$ is presented in Figure 3.6. The processing is done block by block and starts from the last ciphertext block C_{c-1}. Each ciphertext block C_i is transformed into C'_i by being exclusive-ored with its predecessor C_{i-1}. The first block C_0 does not possess a predecessor. Instead, it is exclusive-ored with $k-1$ already processed blocks. Such *circular* chaining makes the reconstruction of any block $C_i, i = 1, \ldots, c-1$ from C'_i impossible without the reconstruction of C_{i-1}.

C_0 is a special block as it is found at the beginning of the chain of blocks. Once reconstructed, it allows the user to "break" the chain and starting from C_1, begin the reconstruction of other blocks. Therefore, C_0 is exclusive-ored with $k-1$ pre-transformed special blocks. The indices of the special blocks $i_1, \ldots, i_{k-1}$ are chosen (i) with gaps at least 1 between them, (ii) and such that, in the next step, the k blocks $C'_0, C'_{i_1}, \ldots, C'_{i_{k-1}}$ all end up in k pairwise different fragments.

Pseudo-code of the inverse transform ReconstructCAON is presented in Figure 3.7. It starts after the ciphertext $CIPH'$ was reassembled from the k fragments. First, C_0 is obtained by exclusive-oring C'_0 with the $k-1$ special blocks. Remaining blocks are then reconstructed by exclusive-oring each block with its already recovered predecessor $C_i = C'_i \oplus C_{i-1}$ starting from C_1.

1: **function** TransformCAON($CIPH, k$)
2: **for** each $i = c-1, \ldots, 1$ **do**
3: Compute $C'_i = C_i \oplus C_{i-1}$
4: First block: $C'_0 = C_0 \bigoplus_{j=1}^{k-1} C'_{i_j}$

Figure 3.6: *Pseudo-code of the linear transform creating dependencies between consecutive blocks of the ciphertext and between the first block C_0 and $k-1$ pre-transformed blocks that will be later dispersed over different fragments (here we choose the $k-1$ special blocks with indices i_j, chosen (i) with gaps at least 1 between them, (ii) and such that, during dispersal, C'_0 and the C'_{i_j} all end up in k pairwise different fragments.*

1: **function** ReconstructCAON($CIPH', k$)
2: First block: $C_0 = C'_0 \bigoplus_{j=1}^{k-1} C'_{i_j}$
3: **for** each block C_i, $i = 1, \ldots, n$ **do**
4: Compute $C_i = C'_i \oplus C_{i-1}$

Figure 3.7: *Pseudo-code of the function reconstructing the initial ciphertext. No block can be reconstructed without first reconstructing C_0.*

```
1: function DisperseBlocks(CIPH', k)
2:     Disperse shares of the block C_0 over k different fragments
3:     for i = 1, ..., c − 1 do
4:         Disperse C'_i to fragment F_j, j = i (mod k)
```

Figure 3.8: *Pseudo-code of the function dispersing the transformed ciphertext* $CIPH'$ *over* k *fragments* $FRAG = F_0, \ldots, F_{k-1}$.

Managing the Key

A variation of the CAON transform hides the encryption key K inside an additional ciphertext block (like in Desai's AONT presented in Section 2.3.1). An additional ciphertext block is computed as $C'_c = C_{c-1} \oplus K$. Once a user possesses the whole transformed ciphertext, they can not only reconstruct the blocks but also the encryption key. This variation could be applied to facilitate key management in distributed storage systems, as it does not require the use of a separate key store (similar solutions are already used in the AONT-RS technology [RP11] of the IBM Cloud Object Storage and in multiple systems using secret sharing for managing keys).

Fragmentation and Dispersal of the Transformed Ciphertext

In order to fully enable the protection against key exposure, transformed ciphertext $CIPH'$ has to be fragmented into at least two fragments that will be stored over independent storage locations. Fragmentation of $CIPH'$ has to follow one rule: k cipher blocks required for the reconstruction of C_0 have to be dispersed over separate fragments. A straightforward fragmentation can be applied where the ciphertext is cut into k chunks of equal sizes,

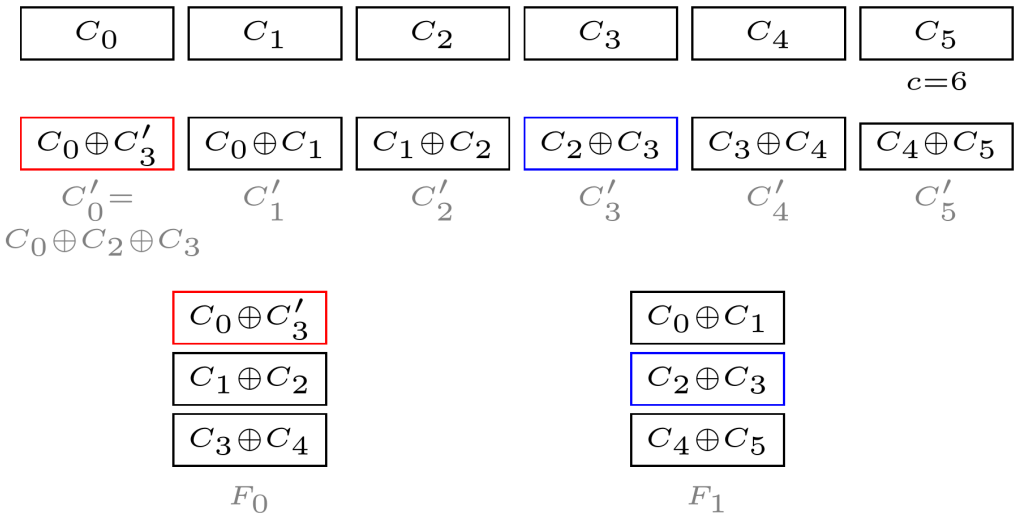


Figure 3.9: *Example for $k = 2$. A ciphertext composed of $c = 6$ blocks is transformed using CAON transform and then dispersed over two storage sites. The block C'_3 was chosen as the special block. Therefore, C'_0 and C'_3 are separated over different fragments. The fragmentation process presented in the Figure, reinforces data protection by dispersing consecutive blocks (consecutive blocks are stored on different sites).*

each containing one of the critical blocks. It can be shown that when the fragments are wisely created, CAON behaves like an all-or-nothing; it forces an adversary to gather all the fragments in order to decrypt even a single ciphertext block.

3.3.3 Comparison with Relevant Works

CAON was compared with relevant works in terms of the amount of computations and ability to protect against key exposure. Results of the comparison are shown in the Table 3.1. As a baseline, the CTR block cipher encryp-

Table 3.1: *Comparison with relevant works in terms of the number of block cipher operations (block op.), number of exclusive-ors, and the ability to provide a key exposure protection (K.E.P). For Bastion and CAON, the number of exclusive-ors coming only from the linear transforms applied after encryption is pointed out. p - number of blocks in the plaintext. c - number of blocks in the ciphertext.*

Algorithm	Block op.	Exclusive-ors	K.E.P.
CTR Enc.	c-1 b.c.	c-1	No
Rivest AONT	2(c-1) b.c.	3(c-1)	No
Desai AONT	c-1 b.c.	2(c-1)	No
Rivest AON	3c-2 b.c.	3(c-1)	Yes
Desai AON	2c-1 b.c.	2(c-1)	Yes
Bastion	c-1 b.c.	3c-1 [Transform: 2c]	Yes
CAON	**c-1 b.c.**	**2c+k-3** [Transform: c+k-2]	**Yes**

tion requires $c-1$ block cipher operations and $c-1$ exclusive-or operations. Rivest and Desai apply encryption during their AONT pre-processing step that comes before the proper encryption. Thus, the complete AON processing composed of AONT and actual encryption doubles (Desai) or triples (Rivest, as the hash of data is computed) the number of block ciphers operations in comparison to normal data encryption. By contrast, the Bastion scheme only applies a linear transform over the encrypted data without increasing the number of block cipher operations. Bastion's transform uses $2c$

exclusive-or operations. Counting with the encryption step, Bastion scheme requires $3c - 1$ exclusive-ors. Similarly to Bastion, CAON applies only a linear transform on the ciphertext. However, its transform uses only $c + k - 2$ exclusive-ors (almost 50% less, as k is usually a small number). This results in a total of $2c + k - 3$ exclusive-ors.

Rivest and Desai AONTs do not protect against key exposure; an attacker possessing the key is able to decrypt the transformed data. AONs encrypt data already preprocessed with AONT making them resistant to key exposure unless the random key used during the pre-processing is also exposed. Bastion protects transformed ciphertext against key exposure unless all but two blocks are exposed. CAON protects data against key exposure unless all the k fragments are being exposed.

Performance Results

Implementation details: Relevant algorithms were implemented using the same programing style in JAVA with JDK 1.8 on DELL Latitude E6540, X64-based PC running on Intel® Core™ i7-4800MQ CPU @ 2.70 GHz with 8 GB RAM, under Windows 7. A standard *javax.crypto* library was used. A random data sample was used for each measurement and each presented result is an average of 30 measurements. AES-CTR-128 was used as the algorithm for symmetric encryption. AES-NI was enabled. Results are somewhat consistent with those presented in [KSLC17] when taking into account the difference between AES and AES-NI (factor of 3 in performance) as well as differences between hardware platforms.

The comparison is presented in Figure 3.10. CAON is the fastest of the

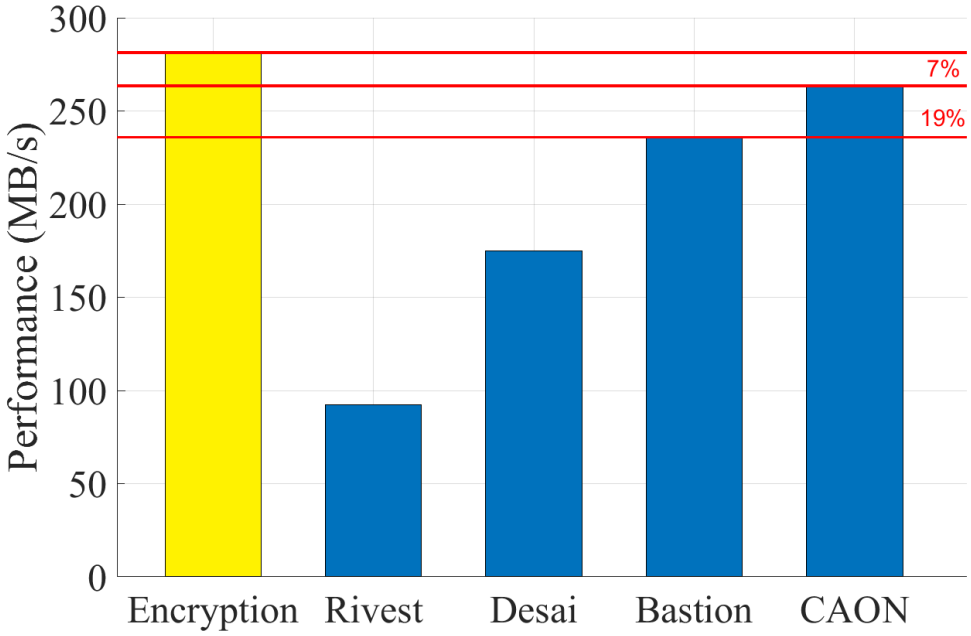


Figure 3.10: *Performance comparison. CAON achieves the best performance among techniques protecting fragmented data against key exposure. AES-CTR-128 was used for encryption. Rivest's and Desai's are presented in AON configurations.*

four schemes, protecting encrypted data against key exposure. Protection against key leakage is achieved with an overhead of only 7% against a simple data encryption. The second fastest scheme, Bastion, results in an overhead of around 19% in comparison to data encryption. It could be assumed that a fine-grained implementation of CAON could make its overhead negligible.

3.4 Selective All-Or-Nothing

The Secure Fragmentation and Dispersal (SFD) scheme and the Circular All-or-Nothing (CAON) algorithm are only efficient when data are equally distributed over at least two independent storage sites. Such solutions may not be acceptable for casual users as dispersing data over independent storage providers may increase storage costs: A study on the cost of using multiple storage providers show that dispersing data over multiple providers is twice the cost of using a single cloud [BCQ+13]. The Selective All-or-Nothing (SAON, published in [KM18c]) scheme addresses the needs of users that would like to benefit from a single cost-efficient storage solution but fear the exposure of their data.

To be specific, SAON transforms a ciphertext into two fragments: a small confidential *private* fragment that will be kept on the user's private device and a large *public* fragment that will be uploaded to an inexpensive storage site. This separation into two fragments is remotely inspired by selective encryption. Decryption of the public fragment is *infeasible* without the private fragment. Consequently, outsourced data are protected against key exposure.

3.4.1 Data Concepts, Notations, and Prerequisite

The following key data components are introduced:

- **(Input or Output) Block (P or C):** a sequence of bits of size $|B|$ corresponding to the classical concept of a block. When referring to a plaintext block, the block is denoted as P, and when referring to a ciphertext the block is denoted as C.

- **Plaintext** ($PLAIN$): initial data composed of p plaintext blocks.
- **Ciphertext** ($CIPH$): encrypted plaintext composed of c ciphertext blocks. $c = p + 1$ as an initialization vector is added at the beginning of the ciphertext.
- **Share** ($SHARE$): intermediary result of the fragmentation of the ciphertext (it corresponds to the concept of share in the Secure Fragmentation and Dispersal scheme).
- **Private Fragment** (F_{priv}): a small data fragment that will be stored at user chosen device.
- **Public Fragment** (F_{pub}): a large data fragment that will be uploaded to a public cloud.

Notations

Plaintext $PLAIN$ is composed of p input blocks $P_1, ..., P_p$. It is encrypted into ciphertext $CIPH$ composed of c blocks $C_0, C_1, ..., C_{c-1}$ ($c = p + 1$), where C_0 corresponds to the initialization vector of the cipher. A ciphertext block C_i comes from the encryption of the plaintext block P_i (except for the initialization vector C_0). In a first step, blocks of the ciphertext $CIPH$ are separated into two shares $SHARE_0^{pub}$ and $SHARE_1$ of size $\frac{c}{2}$ each. A ciphertext block C_i that was attributed to the share $SHARE_j$ is also denoted as C_i^j. In a second step, $SHARE_1$ is fragmented into $SHARE_{10}^{priv}$ and $SHARE_{11}$. $SHARE_{11}$ is then transformed using an all-or-nothing transform and fragmented into $SHARE_{110}^{priv}$ and $SHARE_{111}^{pub}$. The upper index

of a share denotes if a share will be stored at a private or public storage location. Final data stored at a private location are denoted as F_{priv} and data stored at a public storage site are denoted as F_{pub}.

Prerequisite

SAON uses the first fragmentation step of the Secure Fragmentation and Dispersal (SFD) scheme. Therefore, as in SFD, the ciphertext $CIPH$ should be obtained as a result of symmetric encryption using a block cipher with a mode of operation creating dependencies between consecutive ciphertext blocks, (for instance the Cipher Block Chaining mode).

3.4.2 Description of the Scheme

The SAON scheme is composed of three steps. The first step includes the encryption of the plaintext and its transformation into two interdependent shares. The second step operates only on one of the obtained shares. In the final phase, private and public fragments are formed.

Step 1: Encryption and Blocks Separation

Step 1 of the scheme directly applies the first fragmentation step of the SFD scheme for $k = 2$. Plaintext $PLAIN$ is encrypted into ciphertext $CIPH$ using a symmetric block cipher with a mode of operation that reuses the output of the encryption of a previous block during the encryption of the current block, for instance the Cipher Block Chaining or the Cipher Feedback mode. Consecutive blocks of the ciphertext are then separated over two shares, $SHARE_0^{pub}$ and $SHARE_1$. Both shares are necessary in order to

1: **function** FRAGMENTATION($PLAIN$)
2: **Step 1: Encryption and blocks dispersal**
3: Encrypt plaintext $PLAIN$ into ciphertext $CIPH$
4: **for** each pair (C_{i-1}, C_i) such that i is odd **do**
5: C_{i-1} goes to $SHARE_0^{pub}$ and C_i goes to $SHARE_1$
6: **Step 2: All-or-nothing transform on a subset of SHARE$_1$**
7: Fragment $SHARE_1$ into $SHARE_{10}^{priv}$ and $SHARE_{11}$
8: Apply a linear *all-or-nothing* transform over $AON(SHARE_{11})$
9: Fragment $SHARE_{11}$ into $SHARE_{110}^{priv}$ and $SHARE_{111}^{pub}$
10: **Step 3: Forming the final public and private fragments**
11: Form the private fragment $F_{priv} = SHARE_{10}^{priv} + SHARE_{110}^{priv}$
12: Form the public fragment $F_{pub} = SHARE_0^{pub} + SHARE_{111}^{pub}$

Figure 3.11: *Pseudo-code of the fragmentation algorithm transforming a plaintext into a public and a private fragment.*

decrypt the ciphertext $CIPH$. Therefore, an attacker in possession of the encryption key and only one of these two shares will not be able to decrypt even one ciphertext block.

Remark 1 *After Step 1, a user could already save $SHARE_1$ as the private fragment and upload $SHARE_0^{pub}$ to the cloud as the public fragment. However, it would oblige them to keep 50% of the total ciphertext on their private storage device. The second step goes further by transforming $SHARE_1$ into a private and a public fragment and by consequence, increasing the size of the data that can be safely stored in the cloud.*

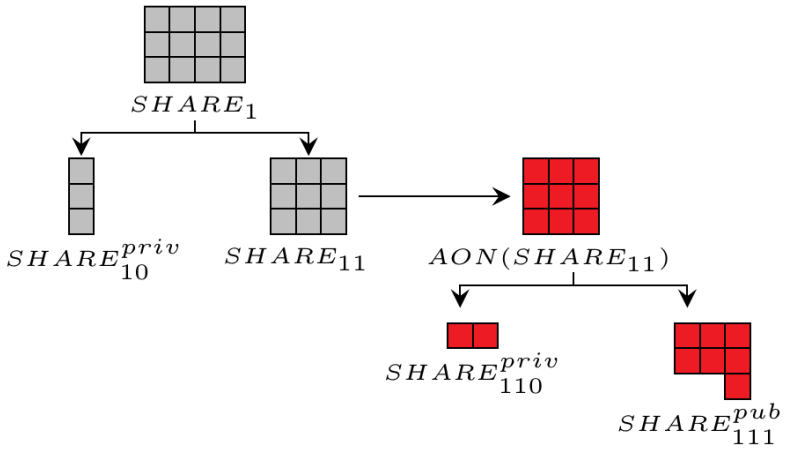


Figure 3.12: *Transformation of $SHARE_1$ into public and private fragments. $SHARE_1$ is first fragmented into $SHARE_{10}^{priv}$ and $SHARE_{11}$. An all-or-nothing transform is applied to $SHARE_{11}$ before fragmenting it into a private $SHARE_{110}^{priv}$ and a public $SHARE_{111}^{pub}$. By choosing a larger $SHARE_{10}^{priv}$ a user improves the performance of the algorithm at the cost of an increase of storage on the private device.*

Step 2: All-or-Nothing Transform of a Subset of Data Contained inside $SHARE_1$

Step 2 operates only on data of $SHARE_1$ ($SHARE_0$ equally could be chosen for further processing instead of $SHARE_1$). It transforms $SHARE_1$ into two small private shares ($SHARE_{10}^{priv}$ and $SHARE_{110}^{priv}$) and one public share $SHARE_{111}^{pub}$. The transformation process is illustrated in Figure 3.12.

Fragmenting $SHARE_1$ into $SHARE_{10}^{priv}$ and $SHARE_{11}$ $SHARE_1$ is fragmented into $SHARE_{10}^{priv}$ and $SHARE_{11}$. $SHARE_{10}^{priv}$ will not be processed but directly stored at a private storage device. $SHARE_{11}$ will be

transformed using an all-or-nothing transform and then fragmented into a private and a public part. The choice of the size of both shares is left to the user. The only requirement is an even number of blocks inside $SHARE_{11}$ (this is necessary for the correctness of the *all-or-nothing* linear transform that will be later applied). On the one hand, a larger $SHARE_{10}^{priv}$ will lead to an increased occupation of the private device's memory. On the other hand, a larger $SHARE_{11}$ will increase the computation overhead as the complexity of the all-or-nothing transform depends on the size of the data on which it is applied.

1: **function** AON($SHARE_{11}$)
2: $SUM = 0$
3: **for** each block C_i inside the share $SHARE_{11}$ **do**
4: $SUM = SUM \oplus C_i$
5: **for** each block C_i inside the share $SHARE_{11}$ **do**
6: $C'_i = SUM \oplus C_i$

Figure 3.13: Pseudo-code of the linear all-or-nothing transform applied to the share $SHARE_{11}$. Each block C_i of the ciphertext is transformed into $C'_i = \bigoplus_0^{m-1} C_l, l \neq i$, where m is the number of blocks contained in the $SHARE_{11}$. In order to achieve linear complexity, the processing is performed in two passes. In the first pass, all ciphertext blocks are exclusive-ored together in order to obtain the value SUM. In the second pass, each block is exclusive-ored with the value SUM.

Transforming $\mathbf{SHARE_{11}}$ using an all-or-nothing transform An all-or-nothing linear transform (the Bastion's all-or-nothing transform was chosen) is applied over $SHARE_{11}$. The goal of this process is to create dependencies between every sub-block of data contained in that share in a way that a correct reconstruction of the share is impossible even if all but two sub-blocks are missing. The pseudo-code of the processing is presented in Figure 3.13.

Fragmenting $\mathbf{SHARE_{11}}$ into $\mathbf{SHARE^{priv}_{110}}$ and $\mathbf{SHARE^{pub}_{111}}$ After the all-or-nothing transformation, the absence of any two sub-blocks inside $SHARE_{11}$ makes the correct reconstruction of this share impossible. In the next step, $SHARE_{11}$ is fragmented into a private share $SHARE^{priv}_{110}$ and a public share $SHARE^{pub}_{111}$. $SHARE^{priv}_{110}$ will be stored in the user's private device as a part of the private fragment and $SHARE^{pub}_{111}$ will be uploaded to a public storage service. Without $SHARE^{priv}_{110}$, data contained inside $SHARE^{pub}_{111}$ are useless to an attacker. Indeed, without $SHARE^{priv}_{110}$, the $SHARE_1$ cannot be reconstructed. Consequently, as even a block of ciphertext data cannot be decrypted without the $SHARE_1$, it is impossible to decrypt the ciphertext without $SHARE^{priv}_{110}$. Obviously, one could imagine a brute-force search over the possible values of $SHARE^{priv}_{110}$. Therefore following recommendation is being formulated:

Recommendation 3 (Recommendation for the size of $\mathbf{SHARE^{priv}_{110}}$) *SAON applies the Bastion's all-or-nothing transform on data inside the $SHARE_{11}$ at the level of sub-blocks. Therefore, it efficiently protects $SHARE_{11}$ against a situation of key exposure unless an attacker acquires all but two*

blocks of $AON(share_{11})$*. Therefore, the size of* $SHARE_{110}^{priv}$ *should be twice as large as a sub-block. However, the size of* $SHARE_{110}^{priv}$ *should also be large enough to prevent a brute-force search of all possible values. Therefore,* $SHARE_{110}^{priv}$ *should contain at least 32 bytes of transformed data: Such a choice leaves all but two data blocks in the public share. The space of the brute-force search is then the same as for a 256-bits encryption key (*2^{256} *possible values).*

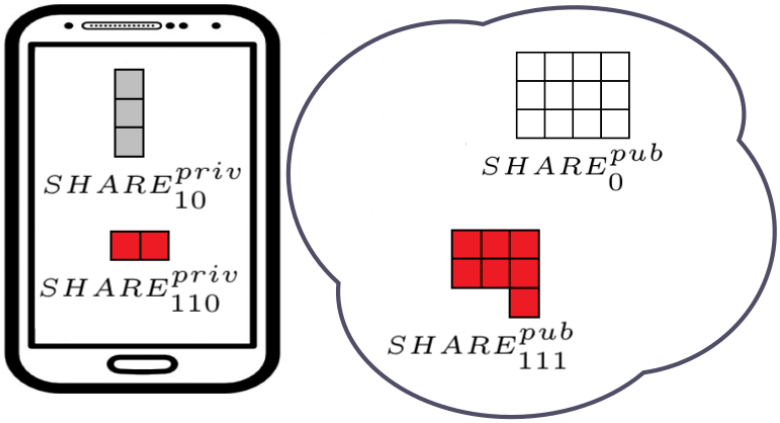


Figure 3.14: *Dispersing private and public fragments (here, on a smartphone with limited capacity and a public cloud) coming from the SFD fragmentation (white and grey shares) and the all-or-nothing transform (red shares).*

Step 3: Forming the Private and the Public Fragment

In a final step, private and public fragments are formed. As shown in Figure 3.14, public fragment is composed of $SHARE_{0}^{pub}$ (coming from the separation of consecutive blocks) and of $SHARE_{111}^{pub}$ (coming from the all-or-nothing transformation). The private fragment is composed of $SHARE_{10}^{priv}$

(coming from the fragmentation of $SHARE_1$) and of $SHARE_{110}^{priv}$ (coming from the all-or-nothing transformation). The public fragment is resistant to a key exposure attack. Indeed, a block inside $SHARE_0^{pub}$ cannot be decrypted without a predecessor that is either stored as a part of $SHARE_{10}^{priv}$ or transformed inside $SHARE_{111}^{pub}$ which in turn is unrecoverable without $SHARE_{110}^{priv}$.

3.4.3 Comparison with Relevant Works

Theoretical Comparison

One can combine block-wise fragmentation of the ciphertext with the application of an all-or-nothing transform on just a part of the ciphertext in order to minimize the amount of computations. In a case where the all-or-nothing transform is applied on the totality of the ciphertext at the level of blocks, $2 \times \frac{|CIPH|}{|B|}$ exclusive-or operations have to be performed. In the case of the proposed algorithm, the all-or-nothing transform is applied only over $SHARE_{11}$, a subset of $SHARE_1$. Thus, it requires $2 \times \frac{|SHARE_{11}|}{|B|}$ exclusive-or operations. As the size of $SHARE_{11}$ is smaller or equal than the size of $SHARE_1$ (which is half of the total size of the ciphertext), it saves at least half of the operations by applying the block-wise fragmentation.

Performance Benchmark

Implementation details Relevant algorithms were implemented in JAVA using the following resources: JDK 1.8 on DELL Latitude E6540, X64-based PC running on Intel® Core™ i7-4800MQ CPU @ 2.70 GHz with 8 GB RAM,

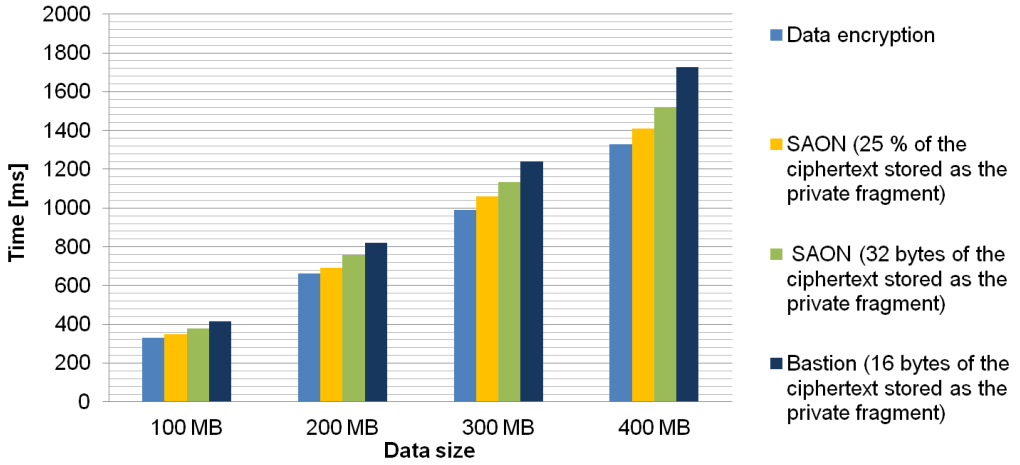


Figure 3.15: *Performance results for various sizes of the data sample and for two sizes of the private fragment. AES-CBC with 128 bit key was used for data encryption. SAON was compared with simple encryption and with the Bastion scheme (where the all-or-nothing transform is applied over the whole ciphertext). For the smallest amount of data stored as a private fragment (16 bytes), the performance overhead of SAON is two times better than Bastion's. The gain becomes greater with the increase of the private fragment (the larger the private fragment, the smaller the data on which the all-or-nothing transform is applied).*

under Windows 7. The standard *javax.crypto* library was used. AES-NI was enabled. A random data sample was used for each measurement.

Results of the performance comparison are presented in Figure 3.15. The processing time for two configurations of SAON is shown:

1. where the size of the private fragment is minimized and only 16 bytes are stored at the user's device.

2. where the private fragment is 25% the size of the ciphertext and where 75% of the data is outsourced to the cloud.

The performance overhead of the SAON is 13% for configuration 1 and of 6% for configuration 2 in comparison to a simple encryption of a plaintext. By contrast, Bastion's scheme (where the whole ciphertext is transformed using an all-or-nothing transform) leads to an overhead of approximately 20%. Increasing the size of the private fragment improves the performance of SAON as it decreases the size of data on which the AONT is applied. However, the performance gain becomes less interesting when the private fragment is larger than 25% of the ciphertext. Indeed, a private fragment containing 25% of the ciphertext already results in a negligible performance overhead of 6%.

The presented performance results were obtained using AES in the Cipher Block Chaining (AES-CBC) mode of operation. We also compared the performance of Bastion's scheme using AES in Counter Mode (AES-CTR) with SAON using AES-CBC. Results were similar to the comparison with Bastion's scheme using AES-CBC.

3.5 Summary

This chapter presented three fast novel ways of protecting encrypted data against key exposure: the Secure Fragmentation and Dispersal Scheme, the Circular All-or-nothing algorithm, and the Selective All-Or-Nothing scheme. All of them fragment a ciphertext in two or more fragments, all of which are needed to be gathered in order to start the decryption process. The difference

between them lies in the context of their application.

Secure Fragmentation and Dispersal (SFD) operates on data encrypted using block ciphers with a mode of operation creating dependencies between consecutive ciphertext blocks. It breaks data into a set of fragments, all of which are needed for data reconstruction: Consecutive blocks are dispersed over different shares who are then shredded over different final fragments. The exact protection level will depend on the number of storage locations that an attacker is able to compromise. SFD was integrated within two known fragmentation methods: Secret Sharing Made Short and AONT-RS. Benchmark tests show an acceptable impact on overall performance. In contrast to similar techniques, the scheme is scalable with the growing number of fragments and does not require any additional operations apart from data dispersal.

The Circular all-or-nothing (CAON) algorithm operates on all kinds of encrypted data. It creates dependencies between consecutive ciphertext blocks. Each block is exclusive-ored with its predecessor forming a sort of chain. Such transformed ciphertext is securely broken into at least two fragments that will be dispersed over independent storage sites. Inverting the chaining transform and recovering the initial ciphertext is only possible once all the fragments are gathered. CAON improves the state-of-the art linear transform by halving the number of required exclusive-or operations in addition to data encryption. The performance evaluation, confirming theoretical results, shows that CAON is the fastest of relevant schemes.

Selective All-or-nothing (SAON) aims at protecting encrypted data that will be outsourced to only a single storage provider. A ciphertext is trans-

formed into two fragments: a large public fragment and a small private fragment. Both fragments are necessary for the decryption of the ciphertext. A user keeps the private fragment on an independent device of their choice and uploads without fear the larger fragment to a public (inexpensive) cloud. Results show good performance: SAON achieves better performance than simply applying the fastest of the known all-or-nothing transforms over the whole ciphertext. Moreover, a user may vary the size of the private fragment in order to balance memory occupation and performance overhead.

In conclusion, the schemes presented in this chapter efficiently address the problem of encrypted data protection against key exposure. Each of these schemes corresponds to a different storage context. The reinforcing of confidentiality comes at a price of an almost negligible performance overhead. They are the fastest of the existing family of all-or-nothing algorithms.

Bibliography

[ABM14] Elena Andreeva, Andrey Bogdanov, and Bart Mennink. Towards understanding the known-key security of block ciphers. In Shiho Moriai, editor, *Fast Software Encryption*, pages 348–366, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[BCQ+13] Alysson Bessani, Miguel Correia, Bruno Quaresma, Fernando André, and Paulo Sousa. Depsky: Dependable and secure storage in a cloud-of-clouds. *Trans. Storage*, 9(4):12:1–12:33, November 2013.

[BDCdVF+16] Enrico Bacis, Sabrina De Capitani di Vimercati, Sara Foresti,

Stefano Paraboschi, Marco Rosa, and Pierangela Samarati. Mix&slice: Efficient access revocation in the cloud. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 217–228, New York, NY, USA, 2016. ACM.

[BLU+15] William J. Buchanan, David Lanc, Elochukwu Ukwandu, Lu Fan, Gordon Russell, and Owen Lo. The future internet: A world of secret shares. *Future Internet*, 7(4):445, 2015.

[Dwo01] Morris J. Dworkin. Nist sp 800-38a, recommendation for block cipher modes of operation: Methods and techniques. Technical report, United States, 2001.

[KM15] K. Kapusta and G. Memmi. Data protection by means of fragmentation in distributed storage systems. In *2015 International Conference on Protocol Engineering (ICPE) and International Conference on New Technologies of Distributed Systems (NTDS)*, pages 1–8, July 2015.

[KM18a] Katarzyna Kapusta and Gérard Memmi. Enhancing data protection in a distributed storage environment using structure-wise fragmentation and dispersal of encrypted data. In *17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications / 12th IEEE International Conference On Big Data Science And Engineering, TrustCom/BigDataSE 2018, New York, NY, USA, August 1-3, 2018*, pages 385–390, 2018.

[KM18b] Katarzyna Kapusta and Gerard Memmi. Poster: Circular aon: A very fast scheme to protect encrypted data against key exposure. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18. ACM, 2018.

[KM18c] Katarzyna Kapusta and Gerard Memmi. Selective all-or-nothing transform: Protecting outsourced data against key exposure. In *Proceedings of the 10th International Symposium on Cyberspace Safety and Security*, CSS '18. Springer, 2018.

[KSLC17] G. O. Karame, C. Soriente, K. Lichota, and S. Capkun. Securing cloud data under key exposure. *IEEE Transactions on Cloud Computing*, pages 1–1, 2017.

[Rab89] Michael O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *J. ACM*, 36(2):335–348, April 1989.

[Riv97] Ronald L. Rivest. All-or-nothing encryption and the package transform. In *In Fast Software Encryption, LNCS*, pages 210–218. Springer-Verlag, 1997.

[RP11] Jason K. Resch and James S. Plank. Aont-rs: Blending security and performance in dispersed storage systems. In *Proceedings of the 9th USENIX Conference on File and Stroage*

Technologies, FAST'11, pages 14–14, Berkeley, CA, USA, 2011. USENIX Association.

[SJH05] Sabre A. Schnitzer, Robert A. Johnson, and Henry Hoyt. Secured storage using secureparser™. In *Proceedings of the 2005 ACM Workshop on Storage Security and Survivability*, StorageSS '05, pages 135–140, New York, NY, USA, 2005. ACM.

Chapter 4

Accelerating fragmentation by combining partial encryption with an all-or-nothing transform

4.1 Introduction and Motivations

In the previous chapter, it was demonstrated that the protection of encrypted data against key exposure will always require some additional pre- [Riv97] or post-processing [KSLC18, KM18a, KM18b, KM18c]. Therefore, such data confidentiality reinforcement inevitably leads to a performance overhead (that for some algorithms can be reduced to the point of being negligible). For some users, even a small overhead may not be acceptable as they are looking for the quickest solutions. Indeed, the choice of the right fragmenta-

tion technique is often a compromise between the desired performance, data protection level, and storage overhead [WBP+01].

This short chapter introduces a solution for accelerating the fragmentation process, making it faster than symmetric data encryption: the Partial Encryption and All-Or-Nothing (PE-AON) scheme. It transforms data into a set of fragments, all of which are required for data reconstruction. In contrast to previously presented schemes, the PE-AON does not fully encrypt data. Instead, it encrypts only a part of the plaintext and then fragments it into k fragments. Plaintext and ciphertext blocks inside the fragments are then blended using an all-or-nothing linear transform applying only exclusive-ors operations. Data protection is provided by dispersal as a complete reconstruction of the initial plaintext and ciphertext blocks is only possible once the fragments are gathered.

Partially avoiding encryption ameliorates the performance as a part of the block cipher operations is replaced by exclusive-ors. Therefore, PE-AON is faster than symmetric encryption combined with straightforward fragmentation (where data are just divided into large chunks of consecutive bits). When the ratio between the number of encrypted and non-encrypted fragments is wisely chosen, data inside the fragments are protected against the exposure of the encryption key. Intuitively, better performance comes at the cost of weaker protection than the one provided by full encryption combined with an all-or-nothing transformation. In previously presented schemes, an attacker is presumed to be able to compromise all but one storage location. By contrast, PE-AON is efficient only against an attacker presumed to be, depending on the configuration, at all but three storage sites or just at a

single location. Thus, PE-AON should be treated as a fast fragmentation method for lightweight data protection, allowing for a gain in performance. It is especially useful in a situation where the user can combine PE-AON with a significant dispersal obstacle, for instance they can separate data over multiple non-colluding clouds.

4.1.1 Data Concepts, Notation, and Prerequisites

The proposed approach identifies the following basic data structures that mostly correspond to the classical concepts concerning a symmetric block cipher [Dwo01]:

- **Block (P or C or B)**: a sequence of bits of size $|B|$ corresponding to the classical concept of block. When referring to a plaintext block, the block is denoted as P, and when referring to a ciphertext the block is denoted as C. When a block can be a ciphertext or a plaintext, it is referred to as simply B.

- **Transformed block (B')**: a block B (P or C) transformed using the all-or-nothing transform.

- **Plaintext ($PLAIN$)**: initial data composed of p plaintext blocks (already padded if needed).

- **Partially encrypted plaintext ($PCIPH$)**: partially encrypted plaintext composed of $p + 1$ blocks in total. It is one block longer than the plaintext as an initialization vector is added during the block cipher encryption. It contains ef ciphertext blocks and $(k - e)f$ plaintext

blocks. e is the value defining the number of encrypted blocks inside the ciphertext. f denotes the number of blocks inside a fragment.

- **Fragment (F)**: result of data fragmentation. Each fragment is composed of $f = \frac{p+1}{k}$ blocks.

Notations

Plaintext $PLAIN$ is composed of p input blocks $P_1, \ldots, P_p$. It is partially encrypted into $PCIPH$ composed of $p+1$ blocks $B_0, \ldots, B_{p+1}$ (one block is appended at the beginning of the $PCIPH$ and corresponds to the initial vector). ef first blocks of $PCIPH$ are ciphertext blocks and the remaining blocks are plaintext. $PCIPH$ is fragmented into k fragments $F_0, \ldots, F_{k-1}$ of size $f = \frac{p+1}{k}$ blocks each. A block B inside the fragments is transformed using an all-or-nothing processing into B'.

Prerequisites

It is assumed that the number of fragments k is a divisor of the number of blocks $p+1$ inside $PCIPH$, so the produced fragments will be of equal sizes. This prerequisite can be easily satisfied using a padding solution.

The recommended value of e should be close to k. Intuitively, the lower it is, the lower is the data protection level. The minimum possible value of e is 3; this ensures that each plaintext block will be exclusive-ored with at least two ciphertext blocks and thus protected against key exposure. Exclusive-oring a plaintext block with only one ciphertext block would be a weaker solution, somehow similar to a one-time pad.

4.2 Description of the Scheme

PE-AON is composed of two steps. The first step comprises partial encryption of the plaintext and its fragmentation. The second step blends encrypted and non-encrypted data inside the fragments. Both steps are described in detail in following subsections.

1: **function** PARTIALLYENCRYPTANDFRAGMENT($PLAIN, e, k$)
2: Transform $PLAIN$ into $PCIPH$:
3: **for** each plaintext block P_i, where $i = 1, \ldots, ef - 1$ **do**
4: Encrypt block P_i
5: Fragment $PCIPH$ composed of $B_0, \ldots, B_p$ blocks into k fragments composed of f consecutive blocks each:
6: **for** $j = 0, \ldots, k - 1$ **do**
7: Disperse blocks from $B_{j \times f}$ to $B_{j \times f + f}$ to fragment F_j

Figure 4.1: *Pseudo-code of the first step. Plaintext is partially encrypted (only the first ef blocks are encrypted) and then fragmented into k fragments.*

4.2.1 Step 1: Partial Encryption and Fragmentation

Pseudo-code of the first step - PARTIALLYENCRYPTANDFRAGMENT- is presented in Figure 4.1. Initial data $PLAIN$ composed of p plaintext blocks are partially encrypted into $PCIPH$. Only the first ef blocks are encrypted using a symmetric block cipher. $PCIPH$ is composed of one block more than $PLAIN$ as, during the encryption process, an initial vector is generated and appended as the first block. $PCIPH$ is then fragmented into k

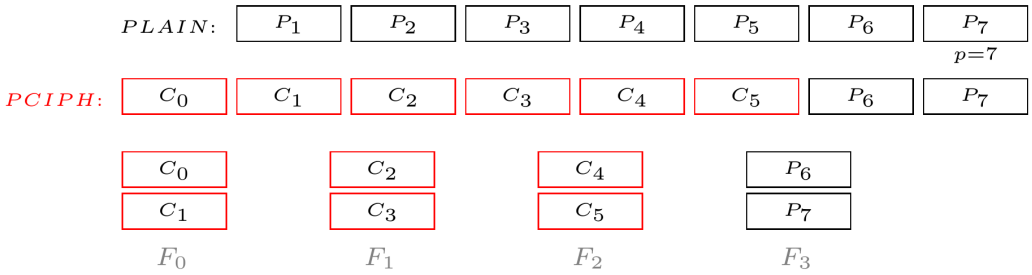


Figure 4.2: *Example for a plaintext composed of $p = 7$ blocks, $k = 4$ fragments, and the number of encrypted fragments $e = 3$. First step of PE-AON partially encrypts a plaintext $PLAIN$ composed of p blocks into $PCIPH$ composed of $p + 1$ blocks. First block of $PCIPH$, C_0, corresponds to the ciphertext block containing the initial vector. Later, PE-AON fragments $PCIPH$ into k fragments of size $f = 2$ blocks each.*

fragments $F_0, \ldots, F_{k-1}$: e of the fragments contain ciphertext blocks and $k - e$ of the fragments contain plaintext blocks. An example illustrating the whole process is shown in Figure 4.2.

4.2.2 Step 2: Blending Plaintext and Ciphertext Blocks

Pseudo-code of the second step - the all-or-nothing transform AON - is presented in Figure 4.3. Blocks inside of the k fragments formed during the first step are processed in a "row by row" fashion. Sets composed of k blocks are formed, where each of the k blocks belongs to a different fragment. Each set is composed of e ciphertext blocks and of $k-e$ plaintext blocks. Blocks inside a set are transformed using Bastion's all-or-nothing linear transform [KSLC18] (this transform could be replaced in the future by a different method applying the same all-or-nothing principle). As a result, each block inside the

1: **function** AON($F_0, \ldots, F_{k-1}$)
2: Proceed in a 'row' by 'row' fashion:
3: **for** $i = 0, \ldots, f-1$ **do**
4: Compute the exclusive-or of all blocks in a 'row':
5: $SUM = \bigoplus_0^{j=k-1} B_{f \times j+i}$
6: **for** $j = 0, \ldots, k-1$ **do**
7: Exclusive-ors it with each value inside the current 'row':
8: $B'_{f \times j+i} = B_{f \times j+i} \oplus SUM$

Figure 4.3: *Pseudo-code of the second step of the PE-AON. An all-or-nothing transform is applied over the k fragments exclusive-oring plaintext and ciphertext blocks. Blocks are processed by 'rows' of k blocks (each block in a row comes from a different fragment). For each 'row' of k blocks a SUM value containing the exclusive-or of the k blocks is first computed. The SUM value is then exclusive-ored with each value in the row. As a result, a block inside the 'row' is exclusive-ored with all* $k-1$ *other blocks from the same 'row'.*

set is exclusive-ored with all other blocks. Therefore, plaintext blocks are exclusive-ored with pseudo-random ciphertext blocks that will protect them. When the minimum possible value of e is 3 (as suggested in the prerequisites), each of the plaintext blocks will be exclusive-ored with at least 2 ciphertext blocks. An example illustrating the result of the all-or-nothing process applied on the fragments is presented in Figure 4.4.

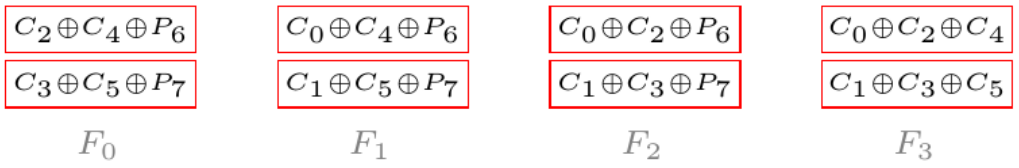


Figure 4.4: *Fragments after applying the all-or-nothing transform, example for $k = 4$ and $e = 3$. Each of the plaintext blocks is exclusive-ored with two ciphertext blocks. All fragments have to be gathered in order to reconstruct the initial data. However, some information is leaked when $k - 1$ fragments are gathered. Therefore, it is important to ensure a secure separation of the fragments.*

4.2.3 Step 3: Dispersing Fragments

In a final step, transformed fragments are dispersed over independent storage sites. The dispersal technique depends on the values of e and k. Two cases may be distinguished.

- For $e = k - 1$: Fragments should be dispersed in a way that no more than $k - 2$ of the fragments are stored at a single storage site. This requirement comes from the property of the chosen all-or-nothing transform [KSLC18], which applied on a set of blocks protects them unless all but two blocks are gathered. When exactly $k - 1$ fragments are gathered, some information about the data is being leaked (it is possible to reconstruct some of the ciphertext or plaintext blocks). An alternative solution to the problem would perform the all-or-nothing exclusive-ors at the level of sub-blocks and not blocks.

- For $e < k - 1$: all fragments have to be dispersed over independent storage

sites. As the number of plaintext fragments increases, it is possible that same combinations of ciphertext blocks will be used to protect different plaintext blocks. Therefore, the data protection level is lower in such a configuration and fragments are only secure when the attacker is assumed to be present only within a single storage site. This is the price that must be payed for acceleration of the fragmentation processing.

Table 4.1: *Comparison with relevant works in terms of the number of block cipher operations (block op.), number of exclusive-ors, the number of storage sites that an attacker is assumed to be able to compromise, and the ability to provide a key exposure protection (K.E.P). All the operations are presented in function of the number of plaintext blocks p.*

Algorithm	Block op.	Exclusive-ors	Compromised Sites	K.E.P.
Encryption	p b.c.	p	-	No
Bastion	p b.c.	$3p+1$	$k-1$	Yes
CAON	p b.c.	$2p+k-1$	$k-1$	Yes
SFD	p b.c.	p	$k-1$	Yes
PE-AON	ef b.c.	$ef+2(p+1)$	$k-2$ if $e=k-1$	Yes
			1 if $e<k-1$	Yes

4.3 Comparison with Relevant Works and Performance Evaluation

PE-AON was compared with relevant works in terms of complexity and data protection levels. A performance benchmark confirmed the complexity evaluation.

4.3.1 Theoretical Comparison with Relevant Works

The PE-AON scheme was compared with relevant works in terms of the amount of computations, the number of storage sites where the attacker is assumed to be present, and the ability to protect against key exposure. Results are shown in Table 4.1. Symmetric encryption is used as a baseline; it requires p block cipher operations and p exclusive-or operations when applied on a plaintext composed of p blocks. Bastion's scheme applies only a linear transform over the encrypted data without increasing the number of block cipher operations. Its transform uses $2(p+1)$ exclusive-or operations. Counting with the encryption step, the Bastion scheme requires $3p+1$ exclusive-ors. Similarly to Bastion, CAON requires some additional exclusive-ors in addition to encrypting data. Secure Fragmentation and Dispersal (SFD) does not require additional operations to data encryption since it just disperses data over fragments.

PE-AON requires only ef block cipher operations as it does not encrypt the totality of the data contained inside the fragments. It performs $cf + 2(p+1)$ exclusive-or operations: ef during the encryption of part of the data and $2(p+1)$ during the all-or-nothing transform applied in the second step.

Encryption and straightforward fragmentation does not protect against key exposure; an attacker possessing the key is able to decrypt the transformed data. Bastion, CAON, and SFD protects fragments unless all of the storage sites are being compromised. The level of data protection in PE-AON is lower since data protection level was traded for better performance. Indeed, it is assumed that the attacker is only present in a single storage site. The only exception is when $e = k-1$, then the attacker may be present in up to $k-2$ storage sites. This could easily be reinforced by making the fragments resistant to an attacker present in up to $k-1$ locations. The only change that would have to be done would be to make the all-or-nothing transform preform exclusive-ors at the sub-block and not the block level. This will be a part of the future work.

4.3.2 Performance Benchmark

Implementation details Relevant algorithms were implemented using the same programming style in JAVA with JDK 1.8 on DELL Latitude E6540, X64-based PC running on Intel® Core™ i7-4800MQ CPU @ 2.70 GHz with 8 GB RAM, under Windows 7. Standard *javax.crypto* library was used. A random data sample was used for each measurement and each presented result is an average of 30 measurements. AES-NI with 128 bits key was used for encryption.

Performance comparisons between relevant algorithms are presented in Figure 4.5. The performance of PE-AON is shown in 4 configurations. In all configurations, PE-AON outperforms encryption and straightforward fragmentation. Therefore, it is also faster than all schemes presented in Chapter 3

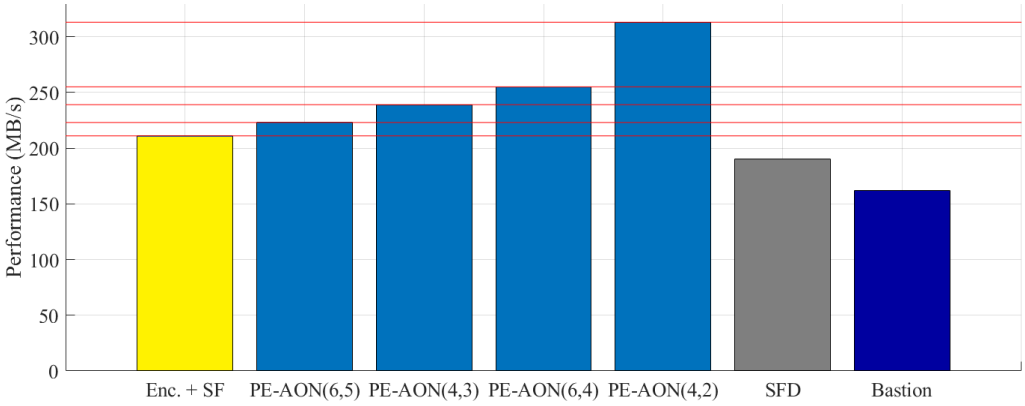


Figure 4.5: *Performance benchmark. PE-AON was measured in different configurations of* (k, e)*. In all of them, it is faster than encryption and straightforward fragmentation. It is also faster than two other fragmentation schemes selected for the comparison: Secure Fragmentation and Dispersal (SFD) and the Bastion's scheme. The protection level of PE-AON is lower but still may be sufficient in some situations. For instance, for PE-AON(6,5) fragments are protected (even against key exposure) unless the attacker is present in 4 or more storage sites, which seems a reasonable attack model.*

(as they all perform full data encryption).

4.4 Summary

This short chapter presented the Partial Encryption and All-Or-Nothing (PE-AON) scheme; a novel algorithm for fast and secure data fragmentation. Initial data are partially encrypted and fragmented. Further, fragments are transformed using an all-or-nothing transform that blends encrypted and non-encrypted fragments. Each plaintext block is protected by at least two

ciphertext blocks. In PE-AON, the desired performance and protection level can be adjusted by varying the number of encrypted blocks. However, the speed up of the fragmentation process comes at the cost of a decrease in data protection levels. Fragments are protected, even against key exposure, but only when the presumed attacker is present in one of the storage sites at most. For a particular configuration, this dispersal requirement is relaxed and the attacker may be present in up to all but two fragments storage locations (in the future, this could easily be changed to all but one by slightly modifying the all-or-nothing transform).

Bibliography

[Dwo01] Morris J. Dworkin. Nist sp 800-38a, recommendation for block cipher modes of operation: Methods and techniques. Technical report, United States, 2001.

[KM18a] Katarzyna Kapusta and Gérard Memmi. Enhancing data protection in a distributed storage environment using structure-wise fragmentation and dispersal of encrypted data. In *17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications / 12th IEEE International Conference On Big Data Science And Engineering, TrustCom/BigDataSE 2018, New York, NY, USA, August 1-3, 2018*, pages 385–390, 2018.

[KM18b] Katarzyna Kapusta and Gerard Memmi. Poster: Circular aon: A very fast scheme to protect encrypted data against key expo-

sure. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18. ACM, 2018.

[KM18c] Katarzyna Kapusta and Gerard Memmi. Selective all-or-nothing transform: Protecting outsourced data against key exposure. In *Proceedings of the 10th International Symposium on Cyberspace Safety and Security*, CSS '18. Springer, 2018.

[KSLC18] G. O. Karame, C. Soriente, K. Lichota, and S. Capkun. Securing cloud data under key exposure. *IEEE Transactions on Cloud Computing*, pages 1–1, 2018.

[Riv97] Ronald L. Rivest. All-or-nothing encryption and the package transform. In *In Fast Software Encryption, LNCS*, pages 210–218. Springer-Verlag, 1997.

[WBP+01] Jay J Wylie, Mehmet Bakkaloglu, Vijay Pandurangan, Michael W Bigrigg, Semih Oguz, Ken Tew, Cory Williams, Gregory R Ganger, and Pradeep K Khosla. Selecting the right data distribution scheme for a survivable storage system (cmu-cs-01-120). 2001.

Chapter 5

A fast and scalable fragmentation algorithm for lightweight data protection

5.1 Introduction and Motivations

The usual solution to ensuring outsourced data confidentiality is to encrypt data before sending it to the storage provider. Using a good symmetric encryption algorithm ensures strong confidentiality guarantees. At the same time, encryption comes at a performance price. Thanks to recent advancements in the hardware development like the integration of the AES instruction set (AES-NI) in many processors, the speed of encryption is being improved. Nevertheless, the overhead is still too important for some users (one reason could be that not all possess powerful devices) and, consequently, a non-negligible amount of data is stored insecurely in the cloud. Data frag-

mentation and dispersal enables a different type of protection mechanism that does not necessarily require classical cryptographic techniques [CBHK15]. Indeed, data shredding and dispersal over different storage sites intuitively provides some data protection as an attacker on a single storage site is unable to obtain the totality of the information. Combined with data permuting and encoding, it can be considered a lightweight method for providing data confidentiality.

This chapter introduces the Fast and Scalable Fragmentation Algorithm (FSFA), a novel approach for data protection in an environment (composed of several independent storage sites) that takes full advantage of the possibilities that lie within data fragmentation and dispersal. It transforms user's data into multiple interdependent fragments. Recovery of even the smallest part of a single fragment depends on an equivalent size of content inside one or more different fragments. In addition, data are shredded before and permuted after the encoding to increase the difficulty of data recovery from an incomplete set of data fragments. FSFA achieves better performance than relevant techniques (including data encryption and straightforward fragmentation) and does not make use of a key. It addresses the needs of a user fearing the disclosure of their outsourced data but desiring the storage solution to be as fast, scalable, and inexpensive as possible.

In the considered threat model, a single cloud provider is *honest-but-curious* - they will try to look at the data they were entrusted with but will not make the effort to contact other cloud providers (who are supposed to be unknown) in an attempt to recover the data. A cloud site may also be vulnerable to external attacks leading to data leakage. In such a situation,

the goal of the algorithm is to fragment the data between the clouds in a way that the fragments received by a single cloud are practically useless.

Another assumption is that in a situation where the choice of an appropriate fragmentation method is a compromise between performance, memory overhead, and data protection, the user favors the performance. Indeed, if the data are very sensitive or even critical, additional protection methods could be applied like perfect secret sharing or symmetric encryption (obviously this would decrease the performance of the solution). In such a case, some fragments could be stored in a private and trusted site, leading to an increased storage cost.

5.2 Data Concepts, Prerequisites, and Notation

The following key data components are introduced with their size in number of bits and their dimensions in terms of the number of elements of which they are directly composed (i.e., a structure S is $|S|$ bits long and composed of s elements):

- **Sub-block (SB):** a sequence of size $|SB|$ bits.
- **(Input or Encoded) Block (B):** a sequence of bits of size $|B|$ composed of $b = \frac{|B|}{|SB|}$ sub-blocks; an input block belongs to the original input data; an encoded block is a result of encoding of an input block.
- **Data (D):** an input data of size $|D|$ bits composed of $d = \frac{|D|}{|B|}$ data blocks

- **Permutation array (PA):** an array of size $|PA|$ bits containing b values: all natural numbers in range $[0, ..., b-1]$ appearing in a random order.

- **Initial Pseudo-Random Block (IPB):** pseudo-random block used as a first block of a fragment. It comes from the xor-split of a permutation array.

- **Fragment (F):** a fragment composed of $f = \frac{|F|}{|B|} = \frac{d}{k}$ input blocks at the beginning of the algorithm; then composed of the same number f of encoded blocks plus one IPB at the end of the algorithm.

5.2.1 Prerequisites and Index Notations

Size of data: The number of blocks inside the data d should be a multiple of the number of fragments k. This can be achieved using padding.

Number of sub-block inside a block: When the number of sub-blocks inside a block is greater than the maximum value that can be encoded on $|SB|$ bits, the size of permutation arrays is greater than the block size. To keep the size of permutation arras equal to the size of a block, the maximum size of the block should not be greater than the maximum value that can be represented on $|SB|$ bits, $max(\#b) = 2^{|sb|}$.

Parameters k and c: In order to facilitate computation, the number of fragments k should be chosen as a multiple of c.

Notation

A fragment is denoted by F_j where j is an integer in $[0, \ldots, k-1]$. A block inside a fragment F_j is denoted by B_i^j, where i is an integer in $[0, \ldots, f]$. A

sub-block at the position v inside a block B_i^j is denoted by $B_i^j(v)$, where v is an integer in $[0, ..., b-1]$. A value at the position t inside a permutation array PA_r is denoted by $PA_r(t)$. An initial pseudo-random block is denoted as IPB and comes from the split of a permutation array. IPB_z coming from the split of a permutation array PA_r is denoted by $IPB_{r,z}$, where z is an integer in $[0, ..., c-1]$ such that $j \pmod c = z$. By convention, an $IPB_{r,z}$ is also the first block B_0^j of the fragment $j = r \times c + z$.

5.2.2 Definitions

FSFA creates dependencies between fragments at the level of data blocks and sub-blocks using a modified version of Shamir's secret sharing. Dependencies are not equally strong between all the fragments but each fragment is directly dependent on $c-1$ other fragments. To create such dependencies, each block of a fragment is encoded using $c-1$ previously encoded blocks from $c-1$ fragments. In order to facilitate the description of the algorithm, the definitions of *neighbor fragments* and *parents blocks* are introduced, defining data structures used during the encoding of a fragment and a block respectively.

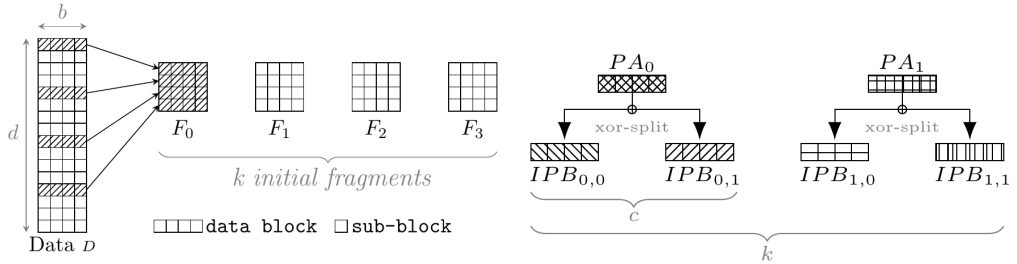
Definition 3 ***Neighbor fragments*** *A fragment F_j from the set of k fragments $F_0, \ldots, F_{k-1}$ possesses $c-1$ neighbor fragments used during its encoding:*

$F_{(j+1) \bmod k}, .., F_{(j+c-1) \bmod k}$

Definition 4 ***Parent blocks*** *A block B_i^j belonging to a fragment F_j such that $i > 0$ possesses $c-1$ parent blocks inside its neighbor fragments:*

$B_{i-1}^{(j+1) \bmod k}, \ldots, B_{i-1}^{(j+c-1) \bmod k}$

Dependencies between blocks are created in the form of a chain where an encoded block is re-used as a parent block during the encoding of the next block. Input blocks do not possess natural predecessors. Instead, initial pseudo-random blocks (IPBs) are used as their parent blocks.



(a) (b)

Figure 5.1: *Illustration for $c = 2$ and $k = 4$.* ***Left:*** *Dispersal of input data. Each fragment receives $\frac{1}{4}$ of the input data d. Any pair of adjacent blocks is distributed to different fragments.* ***Right:*** *Splitting $\frac{k}{c} = 2$ permutation arrays into k IPBs (each permutation array is split into c IPBs). As an example, $IPB_{1,0}$ will be appended to the fragment F_2.*

5.3 Forming Fragments

This section details how data are encoded into k fragments and how these k fragments are dispersed over c independent storage locations. A pseudo-code summarizing the encoding can be found in Figure 5.2. The defragmentation process is not described as it is the direct inverse of the fragmentation (it is also characterized by the same performance).

1: **function** FragmentData(D, c, k)
2: $F_0, \ldots, F_{k-1}$=FormFragments(D, k)
3: $PA_0, \ldots, PA_{\frac{k}{c}-1}$ =GeneratePermutations(c, k, b)
4: $B_0^0, \ldots, B_0^{k-1}$=SplitPermutationsIntoIPBs$\left(PA_0, \ldots, PA_{\frac{k}{c}-1}\right)$
5: **while** all f blocks of each fragment are not processed **do**
6: **for** each block B_i^j of a fragment F_j **do**
7: x=PickEvaluationPoint
8: $ParentBlocks$ =SelectParents$\left(B_i^j\right)$
9: EncodeAndPermuteBlock$\left(ParentBlocks, PA_{j \mod \frac{k}{c}}, x, B_i^j\right)$

Figure 5.2: *Pseudo-code of the function transforming input data D into a set of k fragments, that will be dispersed over k separate locations belonging to at least c independent storage sites.*

5.3.1 Data Distribution over Fragments

In a first step , data $D = B_1, \ldots, B_d$ are distributed over k fragments $F_0, \ldots, F_{k-1}$ in such a way that B_i is assigned to $F_j \iff i \bmod k = j$ (FormFragments function). The number of blocks inside the data is a multiple of k so each fragment receives exactly $\frac{|D|}{k}$ of the input data (illustrated in Figure 5.1). This method of proceeding was chosen as it allows to start the encoding of first distributed data blocks before the whole data are distributed over fragments in a pipelined manner. Data distribution over fragments could also be performed more simply by just dividing data into k consecutive chunks of size $\frac{|D|}{k}$ each.

5.3.2 Generating Permutations

Before beginning to encode data, permutation arrays have to be generated and split into initial pseudo-random blocks (IPBs). GeneratePermutations function generates $\frac{k}{c}$ random permutation arrays $PA_0, \ldots, PA_{\frac{k}{c}-1}$ of length b containing all natural numbers from the range $[0, ..., b-1]$ appearing in a pseudo-random order. The role of a permutation array is to mix up the positions of sub-blocks after block encoding. This is done in order to slow down the recovery of the relationships between encoded sub-blocks (see Section 5.4 for more explanation).Multiple permutation arrays are needed as a fragment has to use a different permutation array than its neighbors.

Function SplitPermutationsIntoIPBs xor-splits each permutation array into c IPBs (illustrated in Figure 5.1). Obtained k (because $\frac{k}{c} \times c = k$) IPBs are distributed over k fragments in a way that IPB_{rz} is assigned to $F_j \iff r \times c + z = j$. $IPB_{r,z}$ becomes the first block B_0^j of a fragment j, when $j = r \times c + z$. Recovery of each permutation array requires all c corresponding IPBs. Therefore, the following dispersal recommendation is formulated:

Recommendation 4 (Dispersing IPBs) *Fragments containing IPBs allowing the recovery of a permutation array should be dispersed over independent storage locations.*

Remark 2 (Generating initial pseudo-random blocks (IPBs)) *In order to minimize memory overhead, IPBs are used as shares allowing the recovery of permutation arrays while, at the same time, being initial pseudo-random blocks used during the encoding of fragments. A different solution*

would consist in having both permutation shares and IPBs separated.

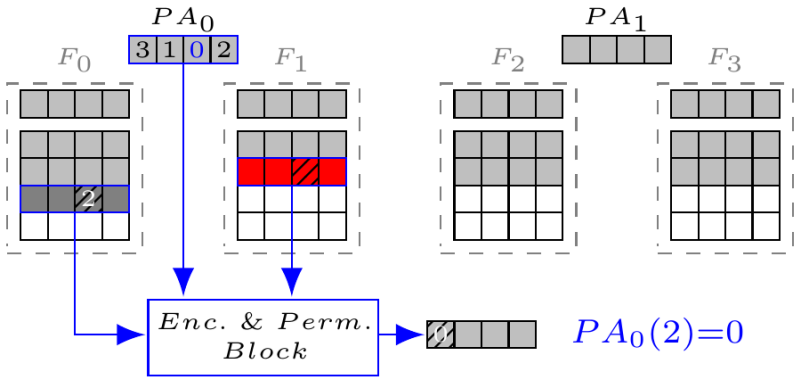


Figure 5.3: *Encoding fragments, example for $c = 2$ and $k = 4$. Fragments are encoded simultaneously, block by block. Input blocks (white) are encoded into encoded blocks (light grey). A current input block (dark grey) is encoded using $c-1$ parent blocks (red) from its neighbor fragments. A current sub-block (dark grey, striped) is encoded using $c-1$ sub-blocks from parent blocks (red, striped). After encoding, sub-blocks are permuted according to a given permutation. When $c = 2$, each fragment possesses one neighbor (as an example, F_1 is the neighbor of F_0).*

5.3.3 Encoding Fragments

Encoding processing is sequential and performed on all fragments simultaneously, block by block (illustrated in Figure 5.3). It creates dependencies between a fragment and its neighbors. More precisely, a block B_i^j inside a fragment F_j is encoded using $c-1$ parent blocks from neighbor fragments of F_j. The processing is sequential and its philosophy could be roughly compared to the Cipher Block Chaining mode - once a block is encoded it becomes

a parent block to another input block. The first input block of each of the fragments does not posses natural parents (a similar problem occurs in CBC mode, where an initialization vector is introduced as the first block). Thus, the k first blocks (IPBs) are used as parent blocks. As they are composed of pseudo-random values, they introduce pseudo-randomness to the fragments' encoding. In order to increase the ratio of pseudo-randomness inside encoded data, fresh IPBs could be generated after encoding a portion of input data.

Encoding and Permuting Blocks

Input blocks are encoded and permuted inside the EncodeAndPermuteBlock function (illustrated in Figure 5.3, pseudo-code in Figure 5.4), taking as input an input block B_i^j (where $j = 0, \ldots, k$ and $i = 1, \ldots, \#f$) to be encoded, its parent blocks, an evaluation point x, and the permutation array $pa_{j \bmod c}$ that will be used to permute sub-blocks of the block. Parent blocks are selected from neighbor fragments inside the SelectParentBlocks function according to Definition 4. Parent blocks are the last $c-1$ encoded (or permutation) blocks from neighboring fragments. The evaluation point x is selected inside the PickEvaluationPoint function. It is an integer in range of $[2, \ldots, 2^{|SB|-1}]$ ($2^{|SB|-1}$ being the maximum value that can be encoded on $|SB|$ bits). It is considered as a known value.

An input block is encoded sub-block by sub-block. The encoding procedure is based on a modification of Shamir's secret sharing. For each sub-block $B_i^j(v)$, where $v = 0, \ldots, b-1$, an encoding polynomial is being constructed. $c-1$ sub-blocks from parent blocks positioned at the same index v to the currently encoded sub-block are selected as the coefficients of this polyno-

1: **function** EncodeAndPermuteBlock($ParentBlocks$,$PermArray$,x,B_i^j)
2: **for** $v = 0 : b - 1$ **do**
3: $a_0, \ldots, a_{c-2}$=SelectCoefficients($ParentBlocks$)
4: $B_i^j(v) = b_i^j(v) + xa_0 + ... + x^{c-1}a_{c-2}$
PermuteSubBlocks$\left(PermArray, B_i^j\right)$

Figure 5.4: Pseudo-code of the function EncodeAndPermuteBlock.

mial (function SelectCoefficients). The result of the evaluation of the polynomial at the evaluation point x is the encoded sub-block. In contrast to Shamir's scheme, the encoding polynomial is evaluated at only one point as just one point in addition to $c-1$ coefficients is sufficient for the decoding.

Intuitively, an encoded block and its parent blocks should be stored over separated locations as, reunited together, they allow the decoding of the input block (in presence of the right permutation array). The following recommendation on blocks dispersal is formulated:

Recommendation 5 (Dispersing blocks) *A block and its $c-1$ parent blocks should be dispersed over c independent storage locations.*

Encoded sub-blocks within a block are permuted using one of the permutation arrays. An encoded sub-block $B_i^j(v)$ goes to position $w = PermArray(v)$. Permuting sub-blocks mixes up relationships between sub-blocks inside a block increasing the difficulty of data recovery from an incomplete set of fragments.

5.3.4 Dispersing Fragments

Fragmentation produces k final fragments $F_0, ..., f_{k-1}$ of size $f + 1$ each (f input blocks and one IPB). Such final fragments should be dispersed over at least c independent storage sites, e.g. independent cloud providers. The dispersal procedure is defined by a single rule: a fragment and its neighboring fragments cannot be stored at a single site. The total number of fragments k is the choice of the user - a higher value of k reinforces data protection as it allows for multiple fragments dispersed over a single site. A weaker variation of the dispersal algorithm where only one storage location is used could be also considered; instead of misleading a curious provider, a user can upload the data fragments from c different accounts. In the considered scenario, it is assumed that the user does not have to care about data availability or integrity as they are usually guaranteed while signing the Service-Level Agreement.

5.4 Security Analysis

Each storage site receives $\frac{k}{c}$ non-neighbor fragments containing uniform and independent data (resistance to frequency analysis was confirmed by an extended empirical analysis not presented in this paper). An attacker situated in less than the totality of the sites can undertake two actions: decode a portion of data from obtained fragments or verify if data inside received fragments match some presumed data. To satisfy their curiosity, they have to overcome a combination of three obstacles: data fragmentation and dispersal, permutation, and encoding.

Data Dispersal The first and simplest obstacle is data dispersal. A single provider receives k fragments containing only a portion of encoded input data of size $\frac{|D|}{c}$ in total. Even decoded, information contained inside the fragments is sampled (result of FormFragments function) and incomplete. Moreover, if the cloud does not receive any information about the ordering of the fragments, there are $\left(\frac{k}{c}\right)!$ possibilities of fragments reassembling.

Data encoding Data encoding creates dependencies between blocks of fragments and introduces pseudo-randomness to the data transformation thanks to IPBs. The following lemma is formulated where, by *infeasible* decoding, it is understood that for an encoded sub-block of size $|SB|$, an attacker must consider $2^{|SB|}$ possible values of the input sub-block:

Lemma 1 (Sub-block encoding) *Decoding of an encoded sub-block* $B_i^j(v)$,$j = 0, \ldots, k$,$i = 1, \ldots, \#f$, *without any knowledge about the input data and the* $c-1$ *values of sub-blocks used during its encoding is infeasible.*

Proof 1 *The procedure encoding sub-blocks of first input blocks directly applies Shamir's secret sharing scheme where* $c-1$ *pseudo-random sub-blocks from IPBs are used as coefficients of encoding polynomials. Encoding results are outputs of an information-theoretically secure scheme so they may be considered pseudo-random since it is supposed that an adversary has no knowledge about the input data. They can be reused as encoding coefficients. The value of an encoded sub-block of size* $|SB|$ *depends on the* $c-1$ *pseudo-random values of size* $|SB|$. *An adversary possessing an incomplete set of coefficients has* $2^{|SB|}$ *possibilities for each of the coefficients to consider. De-*

pending on these coefficients, the input sub-block may take any of the $2^{|SB|}$ *values.*

Data permuting Permutations were introduced in order to protect against a powerful adversary that possessed a fragment and all but one of its neighboring fragments, as well as acquired partial or total knowledge about the input data to which the fragments belong. Such an attacker may undertake two actions: to recover the missing data part or to verify if the fragments correspond to the data. Without block permutations, they would be able to recover the first pseudo-random block of the missing fragments by reversing the encoding procedure. They can then proceed to the verification or to a partial data recovery. Permuting blocks increases the difficulty of reversing the encoding procedure. Indeed, to recover values of a missing block, an adversary has to check all the combinations of permutations of the sub-blocks. The following lemma is formulated:

Lemma 2 (Defragmentation of permuted data) *For an adversary possessing some knowledge about the input data contained inside a fragment, the difficulty of defragmentation or verification of a fragment without the presence of all of its neighbors increases with the number of sub-blocks inside a block and decreases with the knowledge of neighboring fragments.*

Proof 2 *Let's first consider a situation where blocks are not permuted, but just encoded. For each encoded sub-block of a block it is possible to construct one polynomial equation of degree* $c-1$*, where known or unknown sub-blocks from parent blocks are used as coefficients. Recursively, these coefficients may also be represented as polynomial equations so, at the end, all sub-blocks*

may be represented in function of previously encoded sub-blocks as well as of the first pseudo-random sub-blocks coming from IPBs. For a single block of b sub-blocks, a system of b equations is obtained. The difficulty of solving this system of equations depends on the amount of knowledge about input data and the amount of possessed IPBs. When data are permuted, b! possible permutations exist, and as do many equally probable systems of equations.

For a permutation array of size b, b! possible permutations exist. If the blocks are composed of few sub-blocks, a brute-force search over all permutation possibilities is feasible. However, a $b = 34$ *results in* 2.95×10^{38} *permutation array possibilities, which is comparable to the number of tries that are required to perform a brute-force attack on a 128-bit symmetric encryption key (*2^{128} *gives* 3.4×10^{38} *possibilities). An increase of the size of the block slightly affects the storage space but also improves the performance of the fragmentation process (performance results presented in Section 5.6).*

5.5 Complexity Analysis and Storage Requirements

Table 5.1 shows an overview of complexity considerations and storage requirements of concerned fragmentation schemes and of our proposal (FSFA). Algorithms can be divided into two groups. The first group relies on symmetric encryption for data protection and combines it with a key hiding or dispersal method that prevents the key (and therefore the initial data) recovery until k fragments have been gathered. It includes all variations of the all-or-nothing-transform and Secret Sharing Made Short. The second

Table 5.1: *Runtime and storage requirements of relevant algorithms. Poly (n, k, d): cost of encoding data d into n fragments using a polynomial of degree $k-1$. Matrix (n, k, d): cost of multiplying data d by a dispersal matrix DM of dimension $n \times k$. Encrypt (d): cost of using symmetric encryption. Hash: cost of data hashing. RS: cost of applying a Reed-Solomon error correction codes. FragmentData (d, c, k): cost of data processing in FSFA. (D - initial data, $|D|$ - size of d, $|K|$ - symmetric key size, $|B|$ - block size in FSFA, $|DM|$ - dispersal matrix in IDA, k - required number of fragments, n - total number of fragments)*

Scheme	Runtime Fragmentation	Runtime Redundancy	Storage Without Red.	Storage With Red.
SSS	Poly(n,k,D)	-	$k\|D\|$	$n\|D\|$
IDA	Matrix(n,k,D)	-	$\|D\| + \|DM\|$	$\frac{n}{k}\|D\| + \|DM\|$
SSMS	Enc(D) + Poly(n,k,K)	RS(n,k,D)	$\|D\| + k\|K\|$	$n\left(\frac{\|D\|}{k} + \|K\|\right)$
AONT-RS	Enc(D) + Hash(D)	RS(n,k,D)	$\|D\| + \|K\|$	$\frac{n}{k}(d + \|K\|)$
FSFA	FragmentData(d,c,k)	RS(n,k,D)	$\|D\| + k\|PA\|$	$\frac{n}{k}\left(\|D\| + k\|B\|\right)$

group, comprising of Shamir's secret sharing and information dispersal, encodes data using a system of equations which is incomplete when less than $k-1$ fragments are present. Their big problem is the lack of scalability when the number of fragment k is growing, as a growing k entails a growing polynomial degree (SSS) or a growing dimension of the dispersal matrix (IDA). FSFA overcomes the scalability issue by introducing the c parameter. Data are dispersed over k fragments, but encoded using a polynomial of degree c.

The following subsections give more detail about the complexity and storage requirements of analyzed algorithms. A precise evaluation is hard because of the variety of implementations.

SSS and IDA: SSS computes n values of a polynomial ($Poly$) for data d of size $|D|$. Its performance depends on the values of k, n, and $|D|$. Evaluating a polynomial is usually done using the Horner's scheme, which is a $O(k)$ operation. The cost of an IDA equals to the cost of multiplying data D by a $k \times n$ dispersal matrix (DM). In both cases, data are usually first divided into smaller chunks and processed in a chunk by chunk fashion. They strongly benefit from an implementation in finite field arithmetic of the field $GF(2^8)$.

SSMS and AONT-RS: Performance of AONT-RS depends on the chosen encryption ($Encrypt$) and hash ($Hash$) algorithms, as well as on the data size and Reed-Solomon implementation. Wisely implemented, SSMS applies the same mechanisms as AONT-RS: symmetric encryption ($Encrypt$) and Reed-Solomon (RS) codes for redundancy. Instead of hiding the key inside the hash of the whole data, SSMS disperses it within the fragments using Shamir's scheme ($Poly$) applied only on the key. When SSMS is applied on data much larger than a symmetric key, the time taken by the key fragmentation is negligible.

FSFA: $FragmentData(d, c, k)$ is composed of several steps: generating and splitting permutations, data distribution, data encoding, and data permutation. The most consuming operation is the sub-blocks encoding. It takes $c - 1$ additions and $c - 1$ multiplications to encode a single sub-block, as the Horner's scheme for evaluating a polynomial is used. The procedure is repeated for all the sub-blocks inside the data, so at the end $db(c - 1)$

additions and the same number of multiplications are needed to encode all of the data. Because a $GF(2^8)$ finite field is used (like for SSS and IDA), a lookup table is used to replace the multiplications and the additions are replaced by exclusive-ors. Permuting sub-blocks may be implemented as a constant time operation. Data dispersal function FormFragments is an $O(k)$ operation. Being very simple and applied only once, data dispersal and permutations generation and splitting have a negligible effect on the algorithm performance. Additional fragments (if needed) are generated inside an optional procedure RS, which is exactly the same as the one used for SSMS and AONT-RS. FSFA produces $k|B|$ bits of storage overhead (k pseudo-random IPBs). A larger data block increases this overhead, but at the same time improves data protection and performance, as it allows a better parallelization of encoding. The defragmentation procedure is fully parallelizable, as a block may be decoded without waiting for decoding of predecessors.

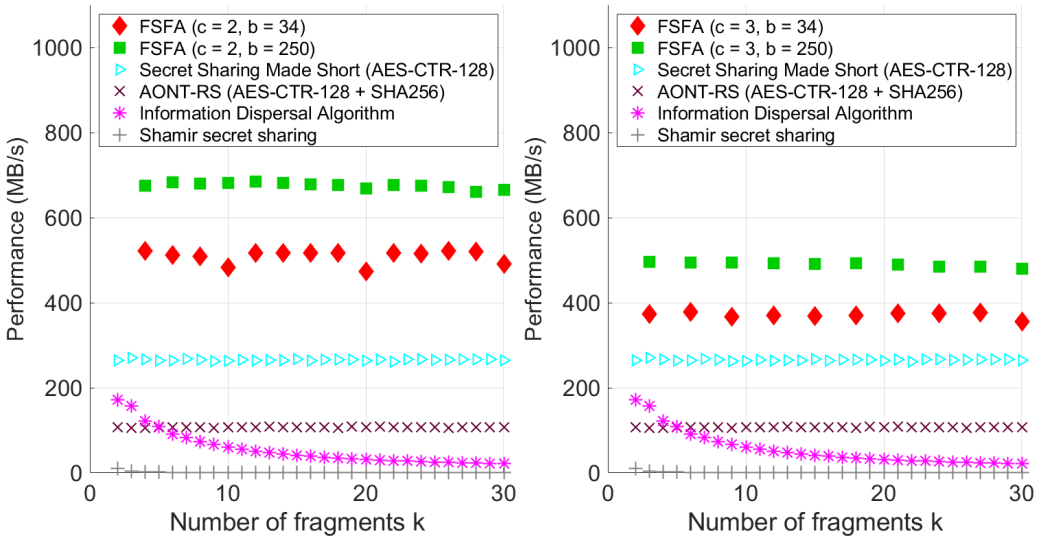
5.6 Comparison with Relevant Works

The proposed algorithm was compared with the state-of-the art fragmentation techniques presented in Chapter 2. All schemes were implemented in JAVA using following resources: JDK 1.8 on DELL Latitude E6540, X64-based PC running on Intel® Core™ i7-4800MQ CPU @ 2.70 GHz with 8 GB RAM, under Windows 7. *javax.crypto* library was used to implement cryptographic mechanisms. Throughput was measured on random data samples of 100MB.

Implementation Details Similarly to SSS and IDA, the proposed algorithm can be implemented in any Galois Field $GF\left(2^Q\right)$. Q is usually selected according to the word size of processors and can be 8, 16, 32 or 64-bit. The presented version was implemented in $GF\left(2^8\right)$ enabling the use of only logical operations. The same field was used for the implementations of SSS and IDA. The AES-128 in the CTR mode was used as the symmetric encryption algorithm inside AONT-RS and SSMS. The AES-NI instruction set was enabled. SHA256 was used as the hash algorithm inside AONT-RS.

The performance of FSFA was measured for four different configurations: for two different values of c (2 and 3) and two different choices of block size (34, and 250 bytes: a block size of 34 bytes makes the recovery of a permutation array similar to performing a brute-force search on a 128-bits key, a block size of 250 optimizes the performance). Results are shown in Figure 5.5. FSFA achieves much better performance than the state-of-art techniques. It is up to 200% (for $c = 2$) faster than the fastest of the relevant works (SSMS with AES). As the cost of fragmentation and key splitting is negligible in SSMS, the performance of SSMS is equivalent to the performance of the algorithm used to encrypt the data. Thus, FSFA achieves better performance than data encryption with AES. AONT-RS is slower than SSMS (as hashing data is more costly than applying Shamir's scheme to split the key). In contrast to other algorithms, IDA and SSS do not scale with the number of fragments k.

Presented techniques were integrated within the DepSky multi-cloud environment [BCQ+13]. Replacing symmetric encryption with FSFA resulted in a gain of $\tilde{2}$0-30% in performance on the client side. The results of an end-to-end performance comparison depend on multiple factors including the SLA



(a) (b)

Figure 5.5: *Performance benchmark for $c = 2$ (left) and $c = 3$ (right).*

and the data size.

5.7 Summary

In this chapter, the Fast Scalable Fragmentation Algorithm (FSFA) for data protection through fragmentation, encoding, and dispersal was introduced and analyzed. Data transformation into fragments relies on a combination of secret sharing and data permutation. It produces a small storage overhead proportional to the number of fragments, which is negligible in relation to larger data. Defragmentation of dispersed data requires the gathering of all fragments, which is possible only by acquiring locations and different access rights of several independent storage sites. Being keyless, the scheme may be used by a user fearing key exposure. Unlike variations of the all-

or-nothing transform, the scheme is adapted for data streaming use cases. Performance benchmarks show that the scheme can be more than 200% faster than state-of-the art comparable and widely renown techniques. The scheme is particularly well adapted for data dispersal in a multi-cloud environment where non-colluding cloud providers ensure the physical separation between data fragments.

FSFA could be seen as a particular case of a more general method for data protection combining fragmentation, encryption, and data dispersal [MKQ15]. Modifications in the way of data dispersal over fragments, data encoding, or data permuting could be done (i.e., initial encoded blocks could be generated in a separated step, Shamir's secret sharing could be replaced with a different secret sharing scheme). From an implementation point of view, performance could be improved by fully exploiting various possibilities of parallelization of the processing.

Bibliography

[BCQ+13] Alysson Bessani, Miguel Correia, Bruno Quaresma, Fernando André, and Paulo Sousa. Depsky: Dependable and secure storage in a cloud-of-clouds. *Trans. Storage*, 9(4):12:1–12:33, November 2013.

[CBHK15] P. Cincilla, A. Boudguiga, M. Hadji, and A. Kaiser. Light blind: Why encrypt if you can share? In *2015 12th International Joint Conference on e-Business and Telecommunications (ICETE)*, volume 04, pages 361–368, July 2015.

[MKQ15] G. Memmi, K. Kapusta, and H. Qiu. Data protection: Combining fragmentation, encryption, and dispersion. In *2015 International Conference on Cyber Security of Smart Cities, Industrial Control System and Communications (SSIC)*, pages 1–9, Aug 2015.

Chapter 6

Data protection by means of fragmentation in unattended wireless sensor networks

6.1 Introduction and Motivations

In a world of the internet of things, wireless sensor networks are widely employed to capture all kinds of environmental information. In a classic approach, they operate in real-time mode where, right after the acquisition, sensors move data to a network static node named the *sink*. However, in some situations, the presence of the sink may not be ensured. For instance, this may happen when sensors are deployed over huge or hostile areas like national parks, battlefields, international border zones etc. Therefore, the term *Unattended Wireless Sensor Networks* (UWSN) was introduced in [DPMS+08] to define a class of sensor networks where data is stored inside the sensors

waiting to be collected by a *mobile sink* (for instance a drone) periodically visiting the network.

UWSNs raise security challenges as data kept inside the nodes may be exposed to various types of attackers wanting to read, destroy, or corrupt the stored information. Several strategies for data protection were already proposed. For instance, in order to achieve data survivability, data may be replicated and moved around the network like in [DPMS+09]. A different approach ensuring, in addition, data confidentiality, consists in making the sensors encrypt, fragment, and disperse data over their neighbors. Data reconstruction is then impossible unless a given threshold of k out of n data fragments is gathered. Data are protected against an attacker unable to compromise the required amount of sensors between consecutive visits of the sink.

Several fragmentation schemes adapted to be used inside UWSN were introduced. They propose a process based on secret sharing ([ROL09a]) or a combination of data encryption and error-correction codes ([KMN17, RRZ08, WRLZ09]). As energy consumption is an important issue inside UWSN, some of the proposals use additively homomorphic schemes in order to reduce storage and transmission costs ([ROL09b, ROL09a]). The Additively Homomorphic Secure Fragmentation Scheme (AHEF) presented in this chapter revisits the HEHSS scheme by [ROL09a]. In both schemes, fragmented and encrypted data are aggregated inside the neighbors' nodes. However, AHEF significantly improves the HEHSS scheme - instead of using additively homomorphic secret sharing for data fragmentation, AHEF uses an additively homomorphic information dispersal algorithm. This change

has a big impact on the resulting storage overhead and transmission costs. Indeed, fragments obtained using secret sharing has the same size as the data itself. Information dispersal produces fragments of optimal size (see the explanation in [Rab89]). Thus, an increase in the data to be protected is avoided. In addition, AHEF allows two means of fragment aggregation: aggregation of fragments from the same sensor and aggregation of fragments coming from the same cluster of sensors.

6.2 Related Work

This section gives an overview of relevant works from the domain of data protection and secure aggregation inside UWSN. It especially focuses on fragmentation schemes used inside UWSN to ensure data confidentiality and availability. During the descriptions, the following terminology is used: A sensor $sensor_i$, $i = 0, \ldots, s-1$, captures data during events named *rounds*, denoted as *round*. r_{max} denotes the number of rounds between consecutive visits of the mobile sink. D_i^r denotes the data of size $|D_i^r|$ bits captured by a single sensor S_i during the round event r. When a fragmentation scheme is used, data D_i^r is fragmented into n fragments, k of which are needed for data reconstruction.

6.2.1 Moving Data around the Network

As presented in [DPMS+08, DPMS+09], attackers come in different flavors. A *curious* attacker will try to learn as much as possible about the stored data. A *polluter* will try to mislead the sink by introducing fraudulent data.

A *search-and-erase* or *search-and-replace* will destroy or modify certain target data. Finally, an *eraser* will erase as much data as possible.

The right defense strategy has to take into account the type of attacker. In [DPMS+09] three non-cryptographic strategies were introduced and analyzed in the context of a search-and-erase and eraser attackers: DO-NOTHING (DN), MOVE-ONCE (MO), and KEEP-MOVING (KM). In DN, captured data are just waiting inside the sensor for the mobile sink. In MO, a sensor offloads newly captured data to some other randomly picked sensor right after capture. In KM, data are moved continuously; each sensor moves each data item individually to another sensor at each round. Moreover, MO and KM may be combined with data replication in order to increase the probability of data survival. For an attacker of type search-and-replace, the choice of strategy will depend on the frequency of sink visits (MO or KM). MO is more efficient than KM when $r_{max} < \frac{s}{k-1}$. When no replication is applied, DN is the best strategy against an eraser. However, data migration becomes better than DN even with a single replica. More works on ensuring data survivability in UWSN using replication were done in [DPV11, ADPG17].

6.2.2 HybridS

HybridS [RRZ08, WRLZ09] is a scheme for secure and dependable storage inside UWSN combining secret sharing with erasure coding. A sensor encrypts round data using a random key. Then, the Reed-Solomon scheme is used to encode the encrypted data into n fragments, k of which are needed for data reconstruction. The secret key is also fragmented into n fragments using a secret sharing scheme like Shamir's scheme presented in [Sha79]. Finally,

the sensor distributes the produced data and key fragments over n randomly selected neighbors. Such processing achieves lower storage and transmission costs than simple data replication. Indeed, the only overhead comes from the key fragments that are the size of the encryption key (typically 128 bits). When the round data is large, the HybridS achieves lower storage and transmission costs than fragmenting data using secret sharing. However, when the round data is very small, the overhead coming from key fragments may be significant. In ([WRLZ09]) a similar scheme (RSSS) combining secret sharing with data encryption was presented. The main difference with HybridS is that it provides data integrity by including algebraic signatures within data fragments.

6.2.3 Homomorphic Key-evolution Scheme

The Homomorphic Key-evolution Scheme (HKS) [ROL09b] provides backward and forward secrecy of data stored inside the sensors by combining dynamic key generation with data aggregation. At each round, a sensor updates the encryption key K_i^r by hashing the key used during the previous round: $K_i^r = h(K_i^{r-1})$. As previous keys cannot be obtained from the current key, forward secrecy is provided. HKS encrypts data using additively homomorphic encryption presented in [CMT05] and thus significantly reduces the volume of data stored inside the sensors. Moreover, to decrypt the aggregated data, an attacker is obliged to have all the dynamic keys used to encrypt them. Thus, backward secrecy is provided. The goal of HKS it to protect the network from curious intruders - those who want to read the stored data. It is inefficient against attackers who want to erase data as no

data resilience is provided.

6.2.4 Homomorphic Encryption and Homomorphic Secret Sharing

The Homomorphic Encryption and Homomorphic Secret Sharing (HEHSS) [ROL09a] scheme improves the HKS scheme by providing data reliability in addition to forward and backward secrecy. Round data is first fragmented into a set of fragments n using an additively homomorphic scheme (Shamir's scheme [Sha79]). Fragments are then encrypted using additively homomorphic encryption. Finally, fragments are distributed over n sensor neighbors where they can be aggregated with fragments coming from previous rounds. Thanks to data aggregation, storage overhead, as well as the cost of data transmission to the mobile sink, it is lower than in RSSS. However, the use of secret sharing increases the size of the data fragments, as Shamir's scheme produces fragments of size $|D_i^r|$ (leading to an n-fold increase in data volume). Therefore, the transmission costs during communication with neighbors are increased.

6.3 Problem Formulation

This section presents assumptions about the considered sensor network architecture and the anticipated adversaries.

6.3.1 Network Model

A scheme of the network architecture is presented in Figure 6.1. In the considered scenario, the network is composed of s sensor nodes. Each sensor node has a fixed location and is denoted as $sensor_i$, where $i = 0, 2, \ldots, s-1$. Data captured by the sensors is collected by one or more authorized mobile sinks visiting the network periodically. To simplify the description of the proposed approach, it is considered that only one mobile sink is present inside the network. We assume that a sensor captures data r_{max} times between consecutive visits of the sink (this is a simplified approach as sensors could have different intervals between rounds, i.e. when the data capture would be triggered by an event). The event of data capture by the sensor is denoted as a round r, where $r = 0,\ 1,\ \ldots,\ r_{max} - 1$. At a round r, a sensor $sensor_i$ captures data denoted as D_i^r.

Sensors are limited in terms of computational power and memory. Apart from capturing and processing data, it is assumed that the sensors are able to communicate with at least n others sensors that are located in their neighborhood. It is a simplified approach as not all sensors may have n other nodes in their neighborhood. Two solutions may be adopted in a situation of lack of neighbors. First, n can be variable and dependent on the sensor location (an example is shown in Figure 6.1). The same solution could be applied in a situation of loss of neighbors due to interferences, obstacles, or issues of sensor mobility. Second, a multi-hop transmission may be used for sensors without enough neighbours. In the worst case scenario, a sensor could function without neighbors for some period of time and then re-transmit collected data with the reappearance of neighbors. In order to save energy, a classic

sleep and live technique can be applied as sensors are alive only during and shortly after each round.

The mobile sink is assumed to be a trusted party that cannot be compromised. It is also assumed that the collected data is processed in an environment not limited in terms of computation power, memory constraints, or energy consumption. For instance, once collected, data could be sent to a cloud for further processing.

Static configuration is a simplified approach. A more sophisticated scheme would include a dynamic selection of neighbors. Such a dynamic network topology would be motivated by two factors. First, the sensors' nature may be mobile, leading their neighbors to continuously change. Second, in a static configuration, attackers who know the topology of the network will first try to compromise the set of sensors they know are their neighbors. A mobile network makes the fragment distribution complicated, but increases the level of data protection. Indeed, moving sensors naturally add randomness to the fragment distribution, confusing the attacker about the location of the fragments. Some works on data fragmentation using secret sharing inside mobile UWSN have already been done in [RG13], showing how parameters of the secret sharing should be chosen in function of the mobility degree of the network.

During the system's initialization, each sensor receives information about the location of its n neighbor's node (as at each round, the sensor will produce n data fragments and disperse them over its neighbors). Information about the network topology should be transmitted securely as it can provide hints to an attacker about the groups of sensors that will store fragments of the

data.

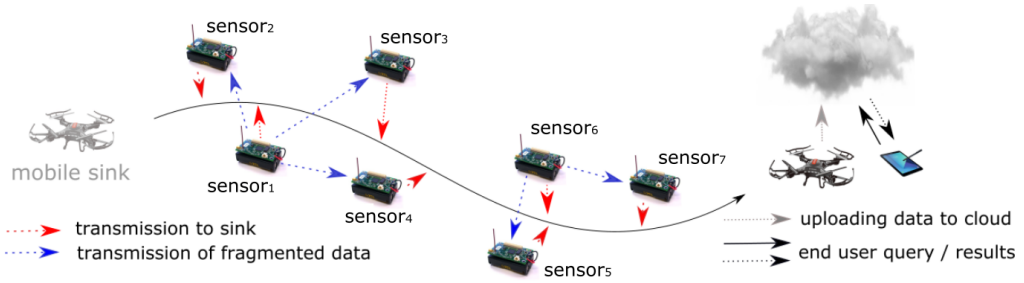


Figure 6.1: *Simplified scheme of the network architecture. Each sensor communicates with its neighbors (during each round, this is shown for just two sensors) and with the mobile sink (during data collection). Depending on the network density, some sensors may have less neighbors than the others; $sensor_1$ has three neighbors but $sensor_6$ only two.*

6.3.2 Threat Model

The threat model presumes that a roaming adversary is present inside the network, and has the ability to compromise most $k-1$ sensors during consecutive visits of a mobile sink. The setting of k and r_{max} must be carefully chosen during the network's dimensioning whilst also considering an estimation of the system's vulnerability. Once the attacker compromises a sensor, they are able to fully control it and consequently obtain the data collected during their occupation of the sensor. However, unless they manage to compromise a set of k sensors storing fragments of the same round data, they are not able to obtain the data collected before the compromise. There is no way for a sensor to distinguish if its neighbor is currently under attack or

not.

An attacker is assumed to not be able to compromise k sensors between consecutive visits of the sink. However, an additional countermeasure may be established, reducing this time interval to the time until initial data collection is done. It would consist of encrypting the first round data fragments stored at the sensor with a temporary key distributed during the system initialization and known to the end user. This temporary key would be deleted from the sensor's memory after the first round. This way, an attacker would only have time until the first round to compromise k sensors and obtain the temporary keys.

The attacker is assumed to be curious - they want to read the stored data. Therefore, the AHEF scheme presented in this chapter focuses mainly on providing data confidentiality and does not treat the problem of fragments authentication or integrity verification. However, it could be complemented with a different scheme treating these two issues, for instance the solution presented in [BPVW11].

6.4 The Proposed Scheme

The goal of the scheme is to protect the data stored inside the sensors until the arrival of the mobile sink while minimizing storage overhead, the complexity of processing, as well as the transmission costs.

6.4.1 System Initialization

During system initialization, a secure hash function (for instance SHA-512) denoted as $h(.)$ is chosen and preloaded to the sensors along with a symmetric cipher algorithm (for instance AES). In accordance with Kerckhoffs's principle, the hash and the cipher function are publicly known. Moreover, each sensor $sensor_i$ receives its own initial key denoted as K_i^0 (key distribution protocol may be based on one of the several solutions proposed in [GC15]). Initial key K_i^0 is refreshed after each visit of the mobile sink by being exclusive-ored with a nonce produced using a secure cryptographic Deterministic Pseudo Random Generator (DPRG) ([BK11]). The seed of the employed DPRG can be constructed by hashing the secret key.

Fragments Distribution and Aggregation

Two configurations of fragment distribution and aggregation are possible. In the first configuration, only data coming from the same sensor is aggregated. At each round, a sensor captures round data, fragments it into a set of n fragments, and disperses the fragments over its neighbors where they are aggregated with fragments distributed during previous rounds. In the second configuration, sensors are organized into clusters of n nodes. At each round each sensor from the cluster sends its data to the neighbors (that are also belonging to the cluster). Then, the data coming from the different nodes of the cluster is aggregated.

Key Evolution

At the beginning of each round, the round index r and the sensor's round key K_i^r are updated. The current round key is obtained by hashing the previous round key: $K_i^r = h\left(K_i^{r-1}\right)$, where $r = 0, \ldots, r_{max} - 1$ and K_i^0 is the initial key. Thanks to the one-way property of the hash function, attackers who compromise a sensor and obtain its current round key will not be able to derive the previously used round keys. Thus, *forward security* is provided. The mobile sink stores the initial key K_i of each sensor, allowing it to derive the round keys as needed.

6.4.2 Processing Round Data

At each round, data captured by a sensor is processed using the Additively Homomorphic Encryption and Fragmentation scheme (AHEF) composed of two steps: data fragmentation and data encryption.

Step 1 - Fragmentation of Round Data into Data Points

In the first step, round data is fragmented into n fragments, k of which are needed for data reconstruction. Instead of the additively homomorphic secret sharing that was applied in HEHSS, an additively homomorphic dispersal algorithm - similar to the one presented in [Kra94] - is used. More precisely, the first step consists of following operations:

1. Represent collected data D_i^r as a vector of k integers $D_i^r(j)$, where $j = 0, \ldots, k-1$ and $D_i^r(j) \in [0, \ldots, d_{max} - 1]$. The size of a each of the k data values is of $\frac{|D_i^r|}{k}$ bits.

2. Choose integer m in function of the predicted number of rounds r_{max}, aggregation configuration (only sensor or between cluster), and d_{max}: $m = 2^{\lceil log_2(r_{max}d_{max})\rceil}$ when the aggregation is performed only on data coming from the same sensor or $m = 2^{\lceil log_2(nr_{max}d_{max})\rceil}$ when the data will be aggregated inside a cluster of n sensors.

   The proposed scheme uses the additively homomorphic encryption scheme (AHE) presented in [CMT05]. In AHE, encryption/decryption operations are modified. Normally, encryption consists of exlusive-oring the data with a random keystream. In AHE, the exclusive-or operation is replaced with addition (encryption) or subtraction (decryption) modulo an integer m. The value of m has to be not only larger than the size of the data to be encrypted, but also has to take into account the number of rounds r_{max} and numbers of fragments that will be aggregated together at each round in order to prevent the overflow coming from the addition of multiple fragments. Indeed, with each round, the sum of the round data increases. Therefore, data fragments have to be large enough to contain the sum of all round data.

3. Construct an encoding polynomial $y_i^r(x) = \sum_{j=0}^{k-1} D_i^r(j)x^j \pmod m$. The k data values are used as the coefficients of this encoding polynomial.

4. Evaluate the polynomial $y_i^r(x)$ at n different evaluation points x_l, $x_l > 0$ and $l = 0, \ldots, n-1$ in order to obtain n different data points $P_{i,r}^l = (x_l, y_i^r(x_l))$ that are the result of the fragmentation of round data D_i^r. The round data can be obtained back by interpolating the polynomial using any k of the n points.

Step 2 - Transforming Points into Fragments using Additively Homomorphic Encryption

Fragments from the first step have to be encrypted in order to be protected. The second step therefore encrypts each fragment using additively homomorphic encryption:

1. Generate a set of n pseudo-random keystreams $T_{i,r}$ using a stream cipher and the current round key K_i^r. Each of the n keystream $T_{i,r}(l)$, $l = 0, \ldots, n-1$ is a value from in range $[0, m-1]$.

2. Transform the point $P_{i,r}^l$ into the final fragment $F_{i,r}^m$ by encrypting the y-axis values of the point. The encryption process entails the modular addition of the y-axis value of the point and of its corresponding keystream:
   $F_{i,r}^l = (x_l, AHE(y_l, T_{i,r}(l))) = (x_l, y_l + T_{i,r}(l) \pmod m)$

6.4.3 Fragments Aggregation

AHEF allows the aggregation of fragments not only between different rounds but also between different sensors. Indeed, any two fragments may be aggregated if they were obtained using the same evaluation points (x-axis values). This is possible because of the additively homomorphic properties of both fragmentation and encryption. Additively homomorphic fragmentation allows the addition of multiple points of different polynomials evaluated within the same set of evaluation points. Furthermore, additively homomorphic encryption allows the addition of multiple encrypted points even if they were encrypted using different keys.

In the proposed simplified architecture described in Section 6.3.1, it is presumed that during the static configuration of the network each sensor receives information about its n neighbors. The sensor is then constantly transmitting fragments evaluated at the same x-axis value to the same neighbor so fragments coming from a same sensor but from different rounds can be easily aggregated.

It is also possible to aggregate fragments coming from different rounds of the same sensor but also fragments coming from different rounds of different sensors. In order to enable the aggregation of fragments coming from different sensors, sensors should be organized in clusters of n sensors. Inside such a cluster, each sensor will receive fragments obtained using the same x-axis evaluation point in order to enable a correct data aggregation. Thus, an information about the evaluation points to be used during fragmentation along with a map associating sensors with those points should be given during the initialization phase.

6.4.4 Data Defragmentation

Data reconstruction is performed after the collection of at least k final fragments by the mobile sink. Each fragment is a sum of r_{max} component fragments (or nr_{max} when cluster aggregation is used): $F^l_{AggSensor} = \sum_1^{r_max} F^l_{i,r}$ or $F^l_{AggCluster} = \sum_1^{r_max} \sum_1^n F^l_{i,r}$. First, fragments are individually decrypted to obtain the aggregated points. Then, the sum of round data is interpolated from the aggregated points.

Step 1: Decryption of Fragments

In the additively homomorphic encryption (AHE), data decryption consists of subtraction modulo m of the keystream from the encrypted data. Thus, decryption of a final fragment consists in performing r_{max} (or nr_{max} when cluster aggregation is used) subtractions modulo m of all the keystreams that were used during the encryption of single fragments components of the final sum.

Step 2: Interpolation

After decryption of the k fragments, k points containing aggregation of r_{max} (or nr_{max}) points are obtained. The next step consists in interpolating a polynomial using those k points and reconstructing the polynomial's coefficients. This can be done using one of the standard interpolating methods, for instance the Lagrange interpolation. The interpolation is a more complex operation than the evaluation used during the fragmentation step (it has an arithmetic operational complexity of $O(k^2)$ that can be reduced to $O(klog^2k)$; the Horner's scheme can be used to reduce the complexity of polynomial evaluation to $O(k)$). However, interpolation is not a problem as long it is performed outside the sensors and there is no real-time constraint (which is the case of the presented scenario).

6.5 Comparison with Relevant Works

AHEF was analyzed in terms of storage overhead and transmission costs as well as in terms of performance of data processing. Results were compared

Table 6.1: *Quantitative analysis of relevant schemes. (*) - A version of RSSS where the key is fragmented and transmitted among the data. (1) - Aggregation is performed only on fragments coming from the same sensor. (2) - Sensors are organized into clusters of n nodes and corresponding fragments from the n sensors are being aggregated.*

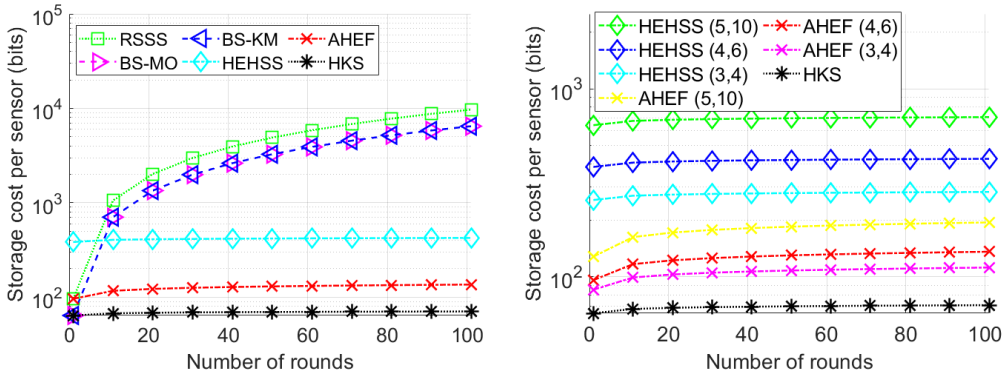
Scheme	Storage Overhead	Transmission Costs	Resilience	Agg.
Data M. [DPMS+08] [DPMS+09]	$\sum_{r=1}^{r_{max}} \|D_i^r\|$	DN: $\sum_{r=1}^{r_{max}} \|D_i^r\|$ MO: $2\sum_{r=1}^{r_{max}} \|D_i^r\|$ KM: $\sum_{j=1}^{r_{max}} \sum_{r=1}^{j} \|D_i^r\|$	No	No
RSSS [RRZ08] [WRLZ09]	$\frac{n}{k} \sum_{r=1}^{r_{max}} \|D_i^r\|$	$\frac{(n+k)}{k} \sum_{r=1}^{r_{max}} \|D_i^r\|$	Yes	No
RSSS (*)	$\frac{n}{k} \sum_{r=1}^{r_{max}} (\|D_i^r\| + k\|key\|)$	$\frac{(n+k)}{k} \sum_{r=1}^{r_{max}} (\|D_i^r\| + k\|K\|)$	Yes	No
HKS [ROL09b]	$\|D_i^r\| + log_2(r_{max})$	$\|D_i^r\| + log_2(r_{max})$	No	Yes
HEHSS (1) [ROL09a]	$n(\|D_i^r\| + log_2(r_{max}))$	$(n+k)(\|D_i^r\| + log_2(R))$	Yes	Yes
HEHSS (2)	$(\|D_i^r\| + log_2(r_{max}n))$	$(n+k)(\|D_i^r\| + log_2(r_{max}n))$	Yes	Yes
AHEF (1)	$n(\frac{\|D_i^r\|}{k} + log_2(r_{max}))$	$n(\frac{\|D_i^r\|}{k} + log_2(r_{max}))$	Yes	Yes
AHEF (2)	$\frac{1}{k}(\|D_i^r\| + log_2(r_{max}n))$	$(n+k)(\frac{\|D_i^r\|}{k} + log_2(r_{max}n))$	Yes	Yes

with relevant works.

6.5.1 Storage Overhead

A comparison of storage overhead (SO) and transmission costs (TC) of relevant schemes was done. Table 6.1 presents a summary of the quantitative analysis. To better illustrate the difference between algorithms, simulations of SO and TC are presented in Figures 6.2 and 6.3. During simulations, a data sample of 1000 bits was created at each round. It was supposed that the end user is interested by the average value of the captured data as well as by their variance.

Intuitively, for all schemes, the storage overhead depends on the size of round data $|D_i^r|$ and the number of rounds r_{max}. As presented in Figure 6.2, data aggregation allows a significant saving of SO. Indeed, for Basic Scheme and RSSS at each round, the SO increases by the size of round data $|D_i^r|$. HKS, HEHSS, and AHEF allow data aggregation. For each of these schemes, at each round, the storage overhead increases only by the value coming from the bits added to avoid overflow during aggregation of data fragments. HEHSS applies secret sharing in addition to encryption which, in contrast to encryption, increases the size of the data (the size of each data fragment is equal to the size of the round data, so it leads to a n-fold increase of SO during the first round). AHEF deals with this problem by replacing Shamir's scheme with Krawczyk's information dispersal.

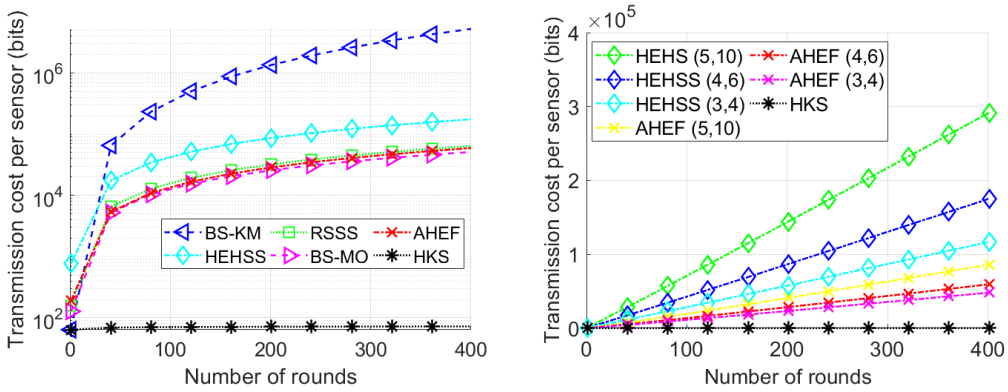


(a) *SO - fragmentation techniques.* (b) *SO - homomorphic techniques.*

Figure 6.2: *Storage cost per sensor. Data aggregation allows a significant reduction in storage cost. (a) HKS does not provide data resilience and it also comes with limited data protection. AHEF achieves a lower storage cost per sensor (factor* $\approx \frac{1}{k}$*) than HEHSS since it replaces secret sharing with information dispersal (b).*

6.5.2 Transmission Costs

Transmission costs are expressed as the sum of bits transmitted from the sensor to its neighbors (during fragmentation) and of bits of stored data transmitted to the sink during data collection. As presented in Figure 6.3, TC for the Basic Scheme in the KEEP-MOVING mode are much higher than for the rest of the schemes (for which the increase is linear in function of the number of rounds). When $k = n$, RSSS and Basic Scheme in the mode MOVE-ONCE have equal TC. Schemes allowing data aggregation comes with a significantly lower TC. HKS has the lowest TC; it does not spread data over the sensor's node since its only TC is the cost of uploading data to the



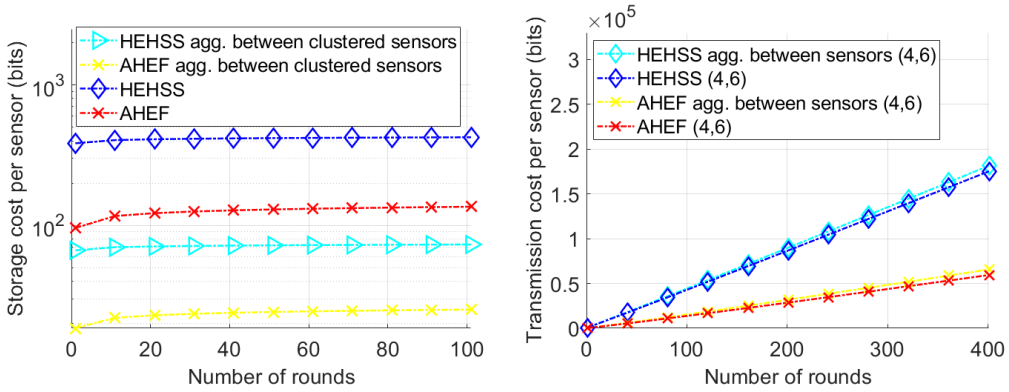
(a) *TC - fragmentation techniques.* (b) *TC - homomorphic techniques.*

Figure 6.3: *Transmission costs (TC) per sensor. Fragmenting data instead of replicating and applying error correction codes helps limit the transmission costs while providing data resilience (a). AHEF significantly reduces TC in comparison to HEHSS as its data fragments are k-times smaller than in HEHSS. HKS does distribute data to its neighbors (and thus does not provide data resilience) so the only TC is the transmission to the sink (b).*

sink. For AHEF, the cost is twofold: in addition to the sink's transmission, at each round, data are diffused over neighbors. Fragment size in HEHSS is k times larger than in AHEF (due to the use of Shamir's scheme). Therefore, at each round, its TC are k times larger than these of AHEF.

6.5.3 Performance Benchmark

Implementation details Relevant algorithms were implemented in JAVA using the following resources: JAVA 1.8 and Matlab 2011b on DELL Latitude E6540, X64-based PC running on Intel® Core™ i7-4800MQ CPU @ 2.70



(a) *Storage overhead.* (b) *Transmission costs.*

Figure 6.4: *Comparison of two aggregation variants: (1) Aggregation of fragments only coming from a single sensor and (2) Aggregation of data coming from a single sensor and some of its neighbors. This demonstrates that aggregating fragments from multiple sensors leads to an increase in transmission costs but a decrease of storage costs.*

GHz with 8 GB RAM, under Windows 7. AES-CTR with 128-bits key was used to generate the pseudo-random keystreams and SHA256 as the hash function.

Benchmark results Execution time for data processing for $r_max = 10000$ rounds was measured for each algorithm in two configurations: when all the fragments are required for data recovery ($n = k$) and when redundant fragments are generated ($n = 1.5k$). The time between consecutive rounds was not taken into account as it is the time when a sensor rests in sleep mode. Results are shown in Figure 6.5. The proposed scheme (AHEF) achieves better performance than HEHSS and RSSS. The performance's gain increases with the number of fragments. The clear lack of scalability of HEHSS and

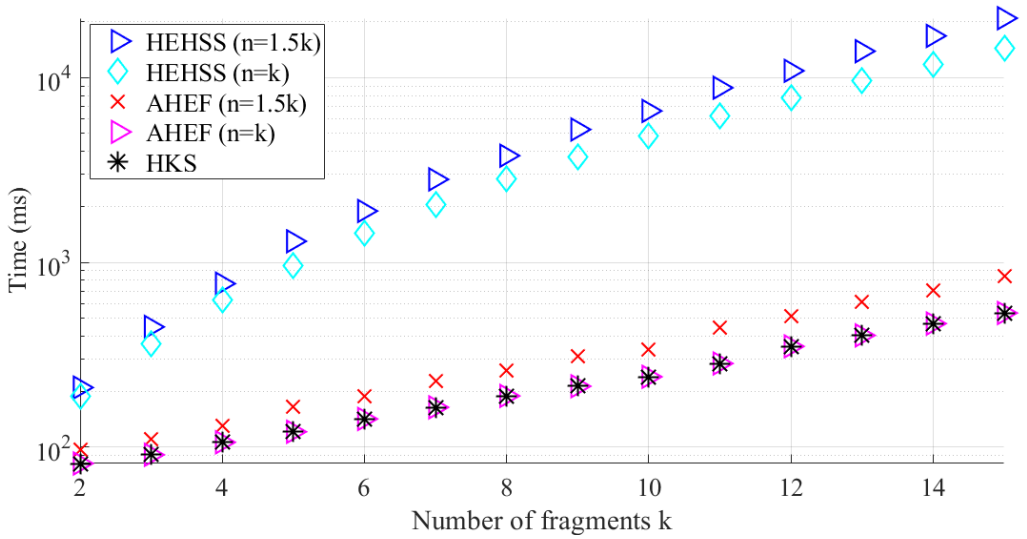


Figure 6.5: *Performance benchmark. AHEF achieves a better performance than HEHSS and RSSS. It comes with a slightly higher performance overhead than HKS as it adds fragmentation to the encryption. HKS does not provide data resilience, which is why it was not included in the comparison for n=1.5k.*

RSSS is mainly caused by the use of Shamir's secret sharing. In contrast to HKS, AHEF provides resilient data and generates interdependent fragments. Thus, it comes with a slightly larger performance overhead.

6.6 Summary

This chapter focuses on a different kind of data fragmentation - the one performed inside Unattended Wireless Sensor Networks. It introduces an Additively Homomorphic Encryption and Fragmentation (AHEF) scheme. The scheme allows the preservation of the backward and forward secrecy

of data collected inside the sensors during multiple collecting rounds. It combines dynamic key evolution with additively homomorphic encryption and additively homomorphic fragmentation.

AHEF replaces additively homomorphic secret sharing used in state-of-the-art techniques with additively homomorphic information dispersal. This change has a significant impact on the volume of data stored inside the sensors as well as on the transmission costs. Both are reduced by a factor of at least 2 (the exact gain will depend on the number of fragments and consequently on the chosen number of neighbors of each of sensors). Two configurations of data aggregation are presented: aggregating data produced by a single sensor and aggregating data produced by a cluster of several sensors. The storage overhead and the transmission costs are compared in both configurations. AHEF considerably reduces the number of required computations allowing sensors to limit the use of their energy not only because of the decreased amount of transmission in comparison to state-of-the-art techniques, but also because of a less energy consuming data fragmentation procedure.

Bibliography

[ADPG17] Giulio Aliberti, Roberto Di Pietro, and Stefano Guarino. Epidemic data survivability in unattended wireless sensor networks. *J. Netw. Comput. Appl.*, 99(C):146–165, December 2017.

[BK11] Elaine B Barker and John Michael Kelsey. *Recommendation for random number generation using deterministic random bit generators (revised)*. US Department of Commerce, Technology

Administration, National Institute of Standards and Technology, Computer Security Division, Information Technology Laboratory, 2011.

[BPVW11] J. Bohli, P. Papadimitratos, D. Verardi, and D. Westhoff. Resilient data aggregation for unattended wsns. In *2011 IEEE 36th Conference on Local Computer Networks*, pages 994–1002, Oct 2011.

[CMT05] C. Castelluccia, E. Mykletun, and G. Tsudik. Efficient aggregation of encrypted data in wireless sensor networks. In *The Second Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services*, pages 109–117, July 2005.

[DPMS+08] Roberto Di Pietro, Luigi V. Mancini, Claudio Soriene, Angelo Spognardi, and Gene Tsudik. Catch me (if you can): Data survival in unattended sensor networks. In *Sixth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom 2008), 17-21 March 2008, Hong Kong*, pages 185–194, 2008.

[DPMS+09] Roberto Di Pietro, Luigi V. Mancini, Claudio Soriente, Angelo Spognardi, and Gene Tsudik. Data security in unattended wireless sensor networks. *IEEE Trans. Comput.*, 58(11):1500–1511, November 2009.

[DPV11] Roberto Di Pietro and Nino Vincenzo Verde. Epidemic data

survivability in unattended wireless sensor networks. In *Proceedings of the Fourth ACM Conference on Wireless Network Security*, WiSec '11, pages 11–22, New York, NY, USA, 2011. ACM.

[GC15] Danilo de Oliveira Gonçalves and Daniel G. Costa. A survey of image security in wireless sensor networks. *Journal of Imaging*, 1(1):4–30, 2015.

[KMN17] Katarzyna Kapusta, Gerard Memmi, and Hassan Noura. Secure and resilient scheme for data protection in unattended wireless sensor networks. In *1st Cyber Security in Networking Conference, CSNet 2017, Rio de Janeiro, Brazil, October 18-20, 2017*, pages 1–8, 2017.

[Kra94] Hugo Krawczyk. Secret sharing made short. In *Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '93, pages 136–146, London, UK, UK, 1994. Springer-Verlag.

[Rab89] Michael O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *J. ACM*, 36(2):335–348, April 1989.

[RG13] Di Pietro Roberto and Stefano Guarino. Data confidentiality and availability via secret sharing and node mobility in uwsn. In *2013 Proceedings IEEE INFOCOM*, pages 205–209, April 2013.

[ROL09a] Y. Ren, V. Oleshchuk, and F. Y. Li. A distributed data storage and retrieval scheme in unattended wsns using homomorphic encryption and secret sharing. In *2009 2nd IFIP Wireless Days (WD)*, pages 1–6, Dec 2009.

[ROL09b] Y. Ren, V. Oleshchuk, and F. Y. Li. Secure and efficient data storage in unattended wireless sensor networks. In *2009 3rd International Conference on New Technologies, Mobility and Security*, pages 1–5, Dec 2009.

[RRZ08] W. Ren, Y. Ren, and H. Zhang. Hybrids: A scheme for secure distributed data storage in wsns. In *2008 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, volume 2, pages 318–323, Dec 2008.

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979.

[WRLZ09] Q. Wang, K. Ren, W. Lou, and Y. Zhang. Dependable and secure sensor data storage with dynamic integrity assurance. In *IEEE INFOCOM 2009*, pages 954–962, April 2009.

Chapter 7

Conclusions and Future Work

This chapter summaries the research contributions presented in this thesis dissertation. It also gives a detailed insight into future work.

7.1 Summary of contributions

This dissertation addresses the problem of preserving data confidentiality using a combination of data fragmentation, encryption, and dispersal.

Chapter 2 contains a detailed survey of data fragmentation as a way of preserving data confidentiality. Two types of fragmentation categories were introduced: bitwise and structurewise. We observe that fragmentation techniques can be divided into these two categories. The bitwise fragmentation category includes techniques like secret sharing, information dispersal, and various schemes based on symmetric encryption. The structurewise fragmentation category gathers the historical object-oriented fragmentation-redundancy-scattering approach together with multiple schemes fragmenting

relational databases in order to preserve user's privacy. Described techniques are compared in terms of performance, storage overhead, and provided data protection level. Performance benchmarks give an empirical comparison in order to confirm theoretical evaluations. In addition to data fragmentation techniques, systems applying data fragmentation are presented. Included here are not only academic but also commercial systems, demonstrating the growing interest of industries in fragmentation as a way of providing data protection. Elements proper to such solutions like the management of fragments' locations are pointed out and portrayed. Recommendations on the design of a secure fragmentation architecture are given at the end of the chapter.

Chapter 3 extends the all-or-nothing family of algorithms with three schemes that are the fastest among known state-of-the-art schemes: Secure Fragmentation and Dispersal (SFD), Circular-All-Or-Nothing (CAON), and Selective All-Or-Nothing (SAON). All of them encrypt then transform and disperse data over multiple storage locations. The main goal is to protect the obtained fragments against the exposure of the cryptographic material, especially the encryption key, without compromising fragmentation performance. They achieve a very moderate performance overhead (between 6 and 10%) in comparison to simple data encryption. This is possible as they require only few operations in addition to data encryption. As SFD does not perform any operation apart from data fragmentation and CAON uses only one exclusive-or per one bit of data, it is actually hard to imagine a fragmentation algorithm that would achieve the same properties with less operations. The three schemes share the same main goal but are dedicated to be applied in

three different contexts. Both SFD and CAON transform data into multiple fragments of equal size. They could be applied in a data center composed of a multitude of servers or by users having access to several storage sites. The difference between the two schemes lies in the fact that they are designed to be applied on different kinds of encrypted data. SFD operates on data encrypted using block ciphers with a mode of operation applying chaining between ciphertext blocks. CAON can be employed on any kind of encrypted data and thus may be applied on data encrypted using streamciphers. SAON differs from SFD and CAON, in that it fragments the transformed data into only two fragments: a small fragment designated to be stored on a trusted device and a large fragment meant to be outsourced. SAON suits the needs of users having access to only a single storage site, i.e. a public cloud.

Schemes presented in Chapter 3 are fast but cannot be faster than their key component - symmetric data encryption applied over the whole input data. Consequently, accelerating the fragmentation process even more requires limiting the number of block cipher operations. This is the main idea behind the Partial Encryption and All-Or-Nothing (PE-AON) scheme presented in Chapter 4. In a first step, PE-AON encrypts only a part of the plaintext. In a second step, it blends the encrypted and non-encrypted data using an all-or-nothing transform based solely on exclusive-ors operations. The speed up of the processing comes at the cost of tightened dispersal requirements; fragments are protected (even against key exposure) but only when an attacker is assumed to have access to at most one of the storage sites. The only exception from this rule is when the amount of non-encrypted data is equal to a size of a single fragment. In such configurations, the attacker

has to compromise the totality of the storage sites in order to reconstruct the initial data and is able to recover some information about the data when accessing all but two storage locations. It is an especially interesting configuration as it allows a fragmentation processing faster than encryption while protecting the fragments against key exposure.

Chapter 5 introduces the Secure and Scalable Fragmentation (FSFA) scheme - a lightweight fragmentation method that, instead of symmetric encryption uses a combination of data encoding, permutation, and dispersal. It produces a small storage overhead proportional to the number of fragments, which is negligible in relation to larger data. Performance benchmarks show that the scheme can be more than two times faster than symmetric encryption. The scheme is particularly well adapted for data dispersal in a multi-cloud environment, where non-colluding cloud providers ensure the physical separation between data fragments. It could be seen as a particular case of a more general method of lightweight data protection combining fragmentation, data encoding, and data dispersal.

Nowadays, the internet-of-things enables data fragmentation and distribution within a different type of architecture - the one composed of a multitude of sensors nodes with tight energy and memory constraints. Chapter 6 treats the problem of providing data confidentiality by means of fragmentation inside Unattended Wireless Sensor Networks. It introduces the Additively Homomorphic Encryption and Fragmentation Scheme (AHEF) that can be used to fragment data inside a sensor before dispersing them over a number of the sensor's neighbours. By replacing additively homomorphic secret sharing used in state-of-the-art techniques with additively homomorphic

information dispersal, AHEF considerably reduces the volume of data stored inside the sensors (more at least 50%), as well as the transmission costs. In addition, it decreases the number of required computations allowing sensors to save the energy of the sensors.

7.2 Future Work

Six schemes were proposed improving the state-of-the-art techniques for data protection by means of fragmentation in two kinds of distributed environments. They open the door to several research tracks and raise a few open questions which need to be addressed in future work.

Fine-grained implementation of the algorithms

The presented schemes were evaluated not only in terms of theoretical complexity but also in terms of execution time. The implementation was realized in a coarse-grained manner as the libraries containing basic algorithms (like the block cipher AES) were not modified. Integrating the proposed schemes within standard mechanisms would improve their performance. For instance, the block dispersal performed by the SFD could be directly implemented inside the function encrypting data using a block cipher and not in a second step after the data encryption (the way it is currently implemented). Moreover, performance could be improved by fully exploiting various possibilities for parallelization of the processing.

Transforming FSFA into a methodology

The Fast and Scalable Fragmentation Algorithm (FSFA) presented in Chapter 5 could be transformed into a more general methodology combining several lightweight mechanisms in order to protect the fragmented data. Indeed, various modifications for methods of data dispersal over fragments, data encoding, or data permutation could be done. First, instead of Shamir's secret sharing, another secret sharing scheme could be applied. Second, the amount of employed pseudo-random data (in FSFA, presented as the Initial Pseudo-random Blocks) could be adjusted to the desired level of data protection. Third, a more sophisticated means of data permutation could be imagined. Although some preliminary cryptanalysis work was performed, a more detailed security analysis comparing it with the state-of-the-art should be performed before its deployment in a industrial context.

The dispersal obstacle

During the descriptions of the dispersal of the fragments, it was assumed that the user has the access to a sufficient number of physically separated or independent storage locations, i.e. different data centers of the same storage provider or different storage providers. However, the question of how to ensure the separation of the fragments is not trivial. First, fragments should not be distributed using a single channel in order to make the man-in-the-middle attack inefficient. Ideally, a separate encrypted communication channel should be established between the trusted user's device and each of the storage locations. Second, the dispersal obstacle is enabled by the difficulty to access the selected number of storage location. Therefore, even a coarse-

grained dispersal over different storage providers may be not challenging for the attacker, for instance if the user uses the same credentials to access each of storage sites. The same situation will occur if the data are dispersed over virtually separated but physically identical storage servers. Therefore, future work will focus on ensuring a secure distribution of the data fragments from the trusted area where the fragmentation occurs to their final destinations.

Processing of fragmented data

The question of a secure processing is unavoidable in the case of outsourced data. A theoretical way of addressing this issue could be seen in the use of homomorphic encryption [Gen09]. However, Fully Homomorphic Encryption is currently impractical and Somewhat Homomorphic Encryption has limited applications areas. Multi-party computation and searchable encryption seems a much more promising direction for following years [ABB+15]. A research track would adapt existing multi-party computation and searchable encryption techniques to the fragmented nature of the data.

Designing a complete bitwise fragmentation storage systems

In the future, presented fragmentation algorithms could be integrated inside existing systems in order to enrich them with an additional data protection mechanism or with a fast and lightweight fragmentation alternative. For instance, they could be integrated within the HAIL system providing high-availability and integrity for cloud storage [BJO09] or the DepSky multi-cloud project [BCQ+13]. Moreover, the all-or-nothing schemes presented in Chapter 3 could be used as a way of creating dependencies between data in

fast access management systems, like the Mix&Slice [BDCdVF+16].

Fragments Distribution and Energy Cost Evaluation of Fragmentation in UWSN

The schemes presented in Chapter 6 were evaluated in terms of storage overhead, transmission costs, as well as performance. In the future, an analysis of the energy consumption should be performed. Indeed, energy optimization plays a crucial role inside UWSN as sensors are powered by batteries. Thus, theoretical results showing that the AHEF scheme requires less computations than other techniques should be confirmed by an empirical measurement of the sensors' energy consumption. Such measurements should take into account not only the energy required for the data fragmentation, but also the energy required to disperse the fragments. The first works on this subject, addressing the optimal distribution of data fragments, are already in progress [CKL18].

A static configuration of the UWSN where the sensor sends data only to its closest neighbors facilitates the description and evaluation of the AHEF scheme. However, this network model could be extended in several aspects. First, the proposed fragmentation scheme could be adapted to a dynamic network model where the nodes are mobile and thus neighbors of a node are constantly changing. Second, a multi-hop data dispersal protocol could be established that would wisely balance between the dispersal scope of the data fragments and the energy consumption of the sensors. A work on this subject is already in progress [LKMJ19].

Bibliography

[ABB+15] Dave Archer, Dan Bogdanov, Sasha Boldyreva, Seny Kamara, Florian Kerschbaum, Yehuda Lindell, Steve Lu, Jesper Buus Nielsen, Rafail Ostrovsky, Jakob I. Pagter, Ahmad-Reza Sadeghi, and Adrian Waller. Future directions in computing on encrypted data, 2015.

[BCQ+13] Alysson Bessani, Miguel Correia, Bruno Quaresma, Fernando André, and Paulo Sousa. Depsky: Dependable and secure storage in a cloud-of-clouds. *Trans. Storage*, 9(4):12:1–12:33, November 2013.

[BDCdVF+16] Enrico Bacis, Sabrina De Capitani di Vimercati, Sara Foresti, Stefano Paraboschi, Marco Rosa, and Pierangela Samarati. Mix&slice: Efficient access revocation in the cloud. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 217–228, New York, NY, USA, 2016. ACM.

[BJO09] Kevin D Bowers, Ari Juels, and Alina Oprea. Hail: A high-availability and integrity layer for cloud storage. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 187–198. ACM, 2009.

[CKL18] Hong-Beom Choi, Young-Bae Ko, and Keun-Woo Lim. Energy-aware distribution of data fragments in unattended wireless sensor networks. In *Proceedings of the Third Inter-*

national Conference on Security of Smart Cities, Industrial Control System and Communications, SSIC '18, 2018.

[Gen09] Craig Gentry. A fully homomorphic encryption scheme, 2009.

[LKMJ19] Keun-Woo Lim, Katarzyna Kapusta, Gerard Memmi, and Woo-Sung Jung. Multi-hop data fragmentation in unattended wireless sensor networks. In *Submitted to the International Conference on Information Processing in Sensor Networks (IPSN 2019)*, 2019.

Appendix A

Empirical Analysis of FSFA

This appendix presents an empirical security evaluation of the Fast and Scalable Fragmentation Algorithm (FSFA) introduced in Chapter 5. Results show that data inside fragments produced by FSFA achieve a good level of uniformity and independence. Contrary to an information dispersal algorithm, this scheme does not preserve patterns inside data fragments.

All tests were performed using Matlab environment on textual data samples provided by the French post office LaPoste[1]. An example of one of such data sample is shown in Figure A.1a. Its corresponding fragment is presented in Figure A.1c and compared to the one obtained using an IDA (Figure A.1b).

Probability Density Function. Frequency counts close to a uniform distribution testify data have a good level of mixing. This means that each byte value inside a fragment should have an occurrence probability close to $\frac{1}{v} = 0.0039$, where v is the number of possible values (256 for a byte). In

[1] http://www.laposte.fr/

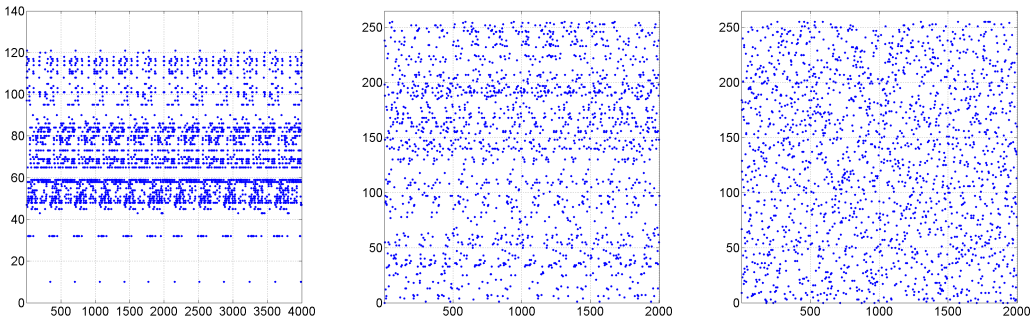


Figure A.1: *Byte value distribution of a textual data sample (a) and distribution of one of its fragment after applying an IDA (b) and after applying the proposed approach (c), k=2. Data patterns are preserved after the use of an IDA. Fragment (c) contains all possible byte values and does not contain visible data patterns. The x-axis shows the byte position inside the sample, the y-axis shows the byte value at position x.*

Figure A.2a, the probability density function (PDF) of a data sample and two of its fragments are shown. Results for the fragments are spread over the space and have a distribution close to uniform. It demonstrates that the occurrence probability of byte values is close to 0.0039.

Entropy. Information entropy is a measure of unpredictability of information content [Cac97]. In a good fragmentation scheme the entropy of the fragments should be close to ideal. Figure A.2b shows the entropy value for three different fragmented data samples for different fragmentation algorithm. The entropy value of the fragments generated using FSFA was comparable with the entropy of fragments generated using SSMS and much

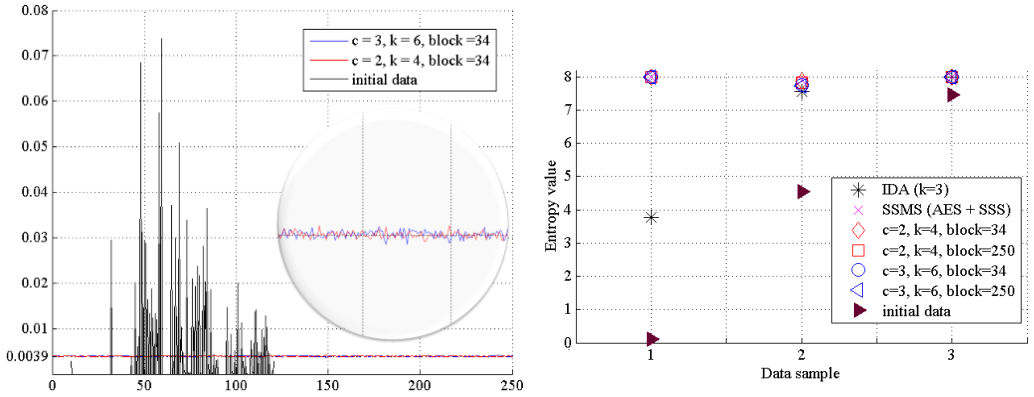
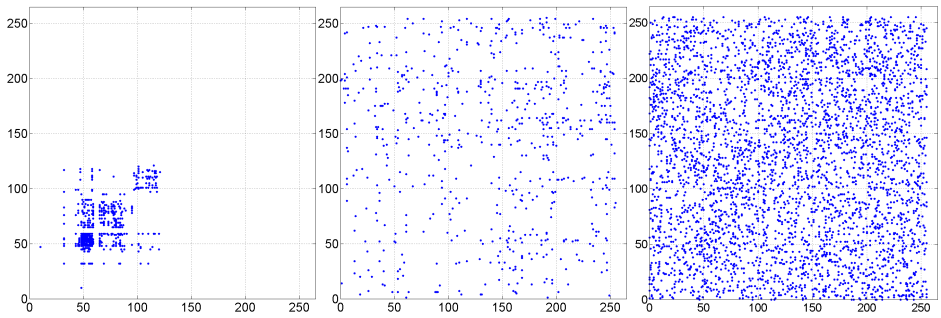


Figure A.2: ***Left:*** *Probability density function comparison for data and their two different fragments. The x-axis shows possible byte values in the sample, the y-axis shows the probability of occurrence of a value. For fragmented data, the occurrence probability is close to the one of a uniform distribution (0.039).* ***Right:*** *Entropy comparison for three data samples (text, image). The maximum entropy value is equal to 8. Entropy of fragments obtained using IDA depends strongly on the entropy of the input data.*

higher than the one obtained using an IDA.

Chi-squared test. The uniformity of byte values of data inside fragments was validated by applying a chi-squared test [Coc52]. For a significance level of 0.05, the null hypothesis is not rejected and the distribution of the fragment data is uniform if $\chi^2_{test} \leq \chi^2_{theory}(255, 0.05) \approx 293$. The test was applied on fragmentation results of 15 different data samples for a fragment size of 1000 bytes. For all samples, the tests was successful.



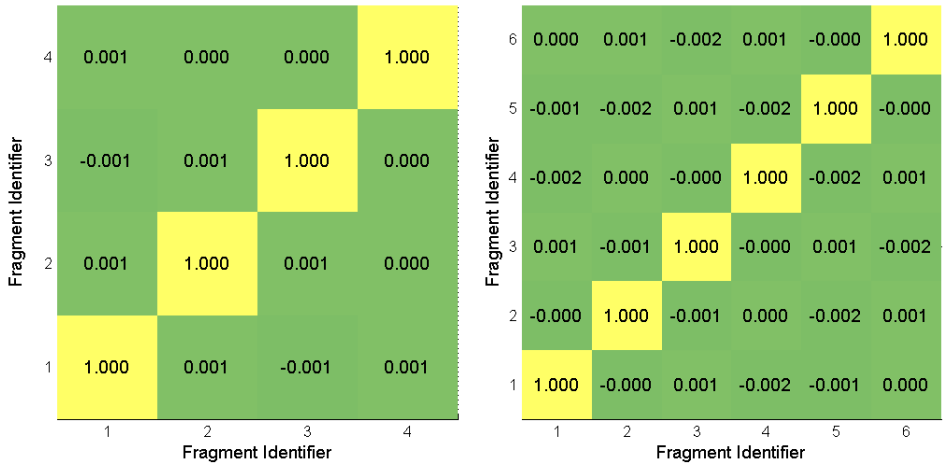
(a) *Original data* (b) *IDA* (c) *Proposed scheme*

Figure A.3: *Recurrence plots for data from Figure A.1. The x-axis shows data values, the y-axis shows data values with a delay* $t = 1$.

Recurrence. A recurrence plot serves to estimate correlation inside data [RN88]. Considering data vector $x = x_1, x_2, ..., x_m$ a vector with delay $t \geq 1$ is constructed $x(t) = x_{1+t}, x_{2+t}, ..., x_{m+t}$. A recurrence plot shows the variation between x and $x(t)$. In Figure A.3, such plots for a data sample and its fragments obtained by applying an IDA and the proposed scheme are shown. Using the proposed scheme, data inside the fragments are more uniformly distributed.

Correlation. Correlation coefficients between fragments were measured and were close to 0. This demonstrates that even neighboring fragments are not correlated with each other and thus confirms the independence property of the scheme.

Difference. Each fragment should be significantly different from the initial data and from other fragments of the same fragmentation result. Bit difference between a data sample and each of its fragments was measured and



(a) (b)

Figure A.4: *Correlation between fragments. Measured for* $c = 2$ *(a) and* $c = 3$ *(b). Block size was equal to 34 bytes. Correlation coefficients are close to 0, even between neighbor fragments.*

was close to 50%. The same result was obtained for the difference between fragments themselves.

Bibliography

[Cac97] Christian Cachin. Entropy measures and unconditional security in cryptography, 1997.

[Coc52] William G. Cochran. The χ^2 test of goodness of fit. *Ann. Math. Statist.*, 23(3):315–345, 09 1952.

[RN88] Joseph L. Rodgers and Alan W. Nicewander. Thirteen Ways to Look

at the Correlation Coefficient. *The American Statistician*, 42(1):59–66, 1988.

Appendix B

Publications, Talks, and Student Projects

This appendix lists the publications that were published during the work on this dissertation. It also contains a list of relevant talks (seminars, poster's presentations). Several student projects were also realized in the framework of this work, they are listed out at the end.

The work presented in this dissertation was partially founded by the ITEA2-CAP WP3 European project.

Publications:

1. Katarzyna Kapusta, Gérard Memmi, and Hassan Noura: Additively Homomorphic Encryption and Fragmentation Scheme for Data Aggregation inside Unattended Wireless Sensor Networks, Annals of Telecommunications, 2018. [accepted to be published, selected among the best papers of the CSnet'18 conference]

2. Keun-Woo Lim, Katarzyna Kapusta, Gérard Memmi, and Woo-Sun Jung, Multi-hop Data Fragmentation in Unattended Wireless Sensor Networks, the International Conference on Information Processing in Sensor Networks (IPSN), Montreal, Canada 2019. [submitted]

3. Katarzyna Kapusta and Gérard Memmi: POSTER: CAON:A Very Fast Scheme to Protect Encrypted Data against Key Exposure. ACM Conference on Computer and Communications Security (CCS), Toronto, Canada, 2018.

4. Katarzyna Kapusta and Gérard Memmi: Selective All-Or-Nothing Transform: Protecting Outsourced Data Against Key Exposure. 10th International Symposium on Cyberspace Safety and Security (CSS), Amalfi, Italy, 2018. [paper invite by the General Chair for the special issue of International Journal of High Performance Computing and Networking]

5. Katarzyna Kapusta and Gérard Memmi: A Fast and Scalable Fragmentation Algorithm for Data Protection Using Multi-storage over Independent Locations. Security and Trust Management (STM, Esorics' workshop), Barcelona, Spain, 2018.

6. Katarzyna Kapusta and Gérard Memmi: Enhancing Data Protection in a Distributed Storage Environment Using Structure-Wise Fragmentation and Dispersal of Encrypted Data. TrustCom/BigDataSE, New York, USA, 2018.

7. Katarzyna Kapusta, Gérard Memmi, and Hassan Noura: Secure and

resilient scheme for data protection in unattended wireless sensor networks. 1st Cyber Security in Networking Conference (CSNet), Rio de Janeiro, Brazil, 2017.

8. Katarzyna Kapusta, Gérard Memmi, and Hassan Noura: POSTER: A Keyless Efficient Algorithm for Data Protection by Means of Fragmentation. ACM Conference on Computer and Communications Security (CCS), Vienna, Austria, 2016.

9. Katarzyna Kapusta and Gérard Memmi: Data protection by means of fragmentation in distributed storage systems. International Conference on Protocol Engineering and International Conference on New Technologies of Distributed Systems (ICPE/NTDS), Paris, France, 2015 .

10. Gérard Memmi, Katarzyna Kapusta, Patrick Lambein, and Han Qiu: Data Protection: Combining Fragmentation, Encryption, and Dispersion, ITEA2-CAP WP3 Final Report, arXiv:1512.02951, 2016.

Talks:

1. Accelerating all-or-nothing transforms. ETH, Zurich, December 2018.

2. Secure Data Fragmentation and Dispersal. Warsaw University of Technology, Warsaw, June 2018.

3. A Keyless Efficient Algorithm for Data Protection by Means of Fragmentation. SnT, Luxembourg, January 2017.

4. Fragmentation for data protection. LINCS, Paris, March 2016.

5. (Poster presentation) Fragmentation for data protection. The International Symposium on Research in Attacks, Intrusions and Defenses (RAID), Paris, September 2016.

Student Projects:

1. Recherche sur la Protection des Données par Fragmentation, Nathalie ENFRIN, professional thesis, 6 months, 2018.

2. Protection des données dans les réseaux de capteurs de type UWSN, Alexandre GUICHANDUT, Matteo BROWANG, and Paul-Ernest MARTIN, 2 weeks, 2018.

3. Protection des données dans les environnements distribués (Cloud), Yohan CHALIER, Gauthier CHATAING, and Antoine URBAN, 2 weeks, 2017.

4. Protection des données dans les environnements distribués (Cloud), Selim BEN AMAR, Abdessalam BOULAHDID, Faycal FASSI-FEHRI, Cedric OSORNIO GLEASON, Quentin LUTS, and Arnaud DUBESSAY, 2 weeks, 2018.

List of Figures

List of Tables

List of Abbreviations

AHEF Additively Homomorphic Encryption and Fragmentation scheme

AHE Additively Homomorphic Encryption

AONT-RS All-Or-Nothing Transform and Reed-Solomon codes

AONT All-Or-Nothing Transform

AON All-Or-Nothing

CAON Circular All-Or-Nothing

FRS Fragmentation-Redundancy-Scattering

FSFA Fast and Secure Fragmentation Algorithm

HEHSS Homomorphic Encryption and Homomorphic Secret Sharing

IDA Information Dispersal Algorithm

PE-AON Partial Encryption and All-Or-Nothing

RSSS Reed-Solomon codes and Secret Sharing based scheme

SFD Secure Fragmentation and Dispersal

SO Storage Overhead

SSS Shamir's Secret Sharing

TC Transmission Costs

UWSN Unattended Wireless Sensor Network

WSN Wireless Sensor Network

XOR-splitting eXclusive-or splitting