ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY

TELECOM ParisTech

2016-ENST-0016

EDITE - ED 130

Doctorat ParisTech

T H È S E

pour obtenir le grade de docteur délivré par

TELECOM ParisTech

Spécialité « Sécurité et Réseaux »

présentée et soutenue publiquement par

Pasquale PUZIO

le 25 Février 2016

Déduplication des données chiffrées dans le Cloud Computing

Directeur de thèse : **Refik MOLVA**
Co-encadrement de la thèse : **Sergio LOUREIRO**

Jury

Mme Nora CUPPENS, Directeur de Recherches, Telecom Bretagne	Rapporteur
M. Bruno MARTIN, Professeur, Université de Nice Sophia Antipolis	Rapporteur
Mme Isabelle CHRISMENT, Professeur, Université de Lorraine	Examinateur
M. Abdelmadjid BOUABDALLAH, Professeur, UTC Compiègne	Examinateur
M. Sergio LOUREIRO, PhD, SecludIT	Examinateur
M. Refik MOLVA, Professeur, EURECOM	Directeur de Thèse

TELECOM ParisTech
école de l'Institut Télécom - membre de ParisTech

THÈSE

TELECOM PARISTECH

Abstract

Computer Science and Networks

Eurecom

Doctor of Philosophy

Deduplication of Encrypted Data in Cloud Computing

by Pasquale PUZIO

In this manuscript we study in depth the problem of deduplicating encrypted data without compromising users' confidentiality. This problem has proved to be challenging and to the best of our knowledge existing solutions suffer from several drawbacks which make massive user adoption difficult. We first conduct a comprehensive study on real datasets aimed at highlighting the best practices in terms of data chunking and whether or not deterministic encryption may introduce a weakness with respect to confidentiality. Moreover, we take advantage of these findings in order to propose two novel and unique solutions to achieve secure data deduplication. In particular, PerfectDedup manages to combine the best both worlds, meaning that users can enjoy efficient client-side block-level deduplication while the underlying system assures full confidentiality for sensitive data.

Acknowledgements

First and foremost, I would like to express my deep and sincere gratitude to my academic supervisor, Professor Refik MOLVA at EURECOM, for providing an excellent guidance and a constant source of inspiration and motivation. Thanks to the countless and fruitful discussions behind every new idea, he transferred to me a bit of his knowledge, which I will proudly bring with me for the rest of my career. He has taught me the methodology to carry out research and to present the research works as clearly as possible. Regardless of the field in which I will develop my career, these principles will always be fundamental. It was a rare privilege and honor to work and study under his guidance.

I am very grateful to SecludIT, in particular to Sergio LOUREIRO and Frederic DONNAT, for trusting me since the beginning and giving me the opportunity to do research and express my value.

Furthermore, I desire to thank Melek ÖNEN, who has worked at my side on all my publications and gave a great contribution to the ideas I developed during my PhD thesis.

I would also like to thank the IT staff at EURECOM, in particular Jean-Christophe DELAYE and Pascal GROS, who have always been very kind to me by allowing me to have access to EURECOM users' data and conduct the data study on it.

I am extremely grateful to my parents for their love and sacrifices for educating me. Whatever I will manage to achieve in my life, it will be thanks to the good example they gave me.

I am very much thankful to my girlfriend for her love, understanding and continuing support to complete this thesis.

Finally, I would like to say thanks to all my friends and colleagues for their genuine support throughout this experience.

Résumé en Français

Déduplication

L'une des techniques qui ont été récemment adoptées par de nombreux fournisseurs de stockage Cloud est la déduplication, qui permet de réaliser des importantes économies d'espace de stockage, en particulier dans les applications de sauvegarde [1], [2]. La déduplication est basée sur l'observation suivante: il est très fréquent que les différents utilisateurs peuvent télécharger des fichiers identiques ou très similaires. Dans ce cas, le stockage de copies multiples des mmes segments de données générera un gaspillage inutile des ressources de stockage. Par conséquent, au lieu de simplement stocker les données téléchargées par les utilisateurs, le fournisseur de stockage Cloud peut d'abord vérifier si un segment particulier de données a déjà été téléchargé dans le passé: cela se fait généralement en comparant son identifiant avec une base de données contenant les identifiants de tous les segments de données stockées au Cloud. Le cas échéant, il est inutile de le stocker à nouveau depuis le fournisseur de stockage Cloud peut créer un lien logique pour le segment actuel de données. Grâce à l'efficacité d'une telle idée, malgré sa simplicité, la déduplication est devenu un standard de facto [3] dans tous les systèmes majeures de stockage grâce à sa capacité de réduire considérablement l'empreinte de données, réduisant ainsi les cots de stockage et de consommation de bande passante.

Déduplication des Données Chiffrées

Toutefois, comme indiqué ci-dessus, la demande des clients comprend également la confidentialité, qui de jour en jour devient un problème urgent pour les entreprises ainsi que pour les utilisateurs. Par conséquent, le principal défi dans le domaine du stockage Cloud est de savoir comment répondre aux besoins contradictoires des

clients et des fournisseurs qui, d'un point de vue de la confidentialité, ont besoin de moyens combinant en toute sécurité et efficacement le cryptage avec des techniques d'optimisation du stockage telles que la déduplication. Bien qu'il puisse sembler simple, mettre les deux techniques ensemble est loin d'être trivial: le résultat souhaité du cryptage est de rendre les données chiffrées (le cryptogramme) illisibles et impossibles à distinguer à partir des données d'origine (le texte en clair), alors que la déduplication nécessite que le fournisseur de stockage Cloud soit en mesure de comparer facilement deux segments de données (cryptées) et déterminer si elles sont identiques. En d'autres mots, la déduplication de données chiffrées avec une méthode standard ne sera pas possible, à moins que les clés de chiffrement sont divulgués aux fournisseurs de stockage Cloud, ce qui leur donnerait le pouvoir de lire les données des utilisateurs, donc compromettre la confidentialité des données. De cette question, le problème suivant se pose: comment pouvons-nous préserver la confidentialité des données sans empcher un fournisseur de stockage Cloud curieux de détecter les doublons?

La réponse à cette question est loin dtre triviale. Dans un scénario commun, deux utilisateurs qui ont les mmes données et que les souhaitez stocker en toute sécurité dans le Cloud, vont crypter leurs données avant d'effectuer le téléchargement. Malheureusement, une technique de cryptage standard rendrait la déduplication impossible puisque deux utilisateurs différents utiliseront deux clés différentes, résultant en deux cryptogrammes différents.

Chiffrement Convergent

Récemment, une contre-mesure simple mais efficace a été proposée comme une solution immédiate à ce problème: la clé de cryptage peut en fait tre généré de manière totalement déterministe et dépendant des données, de sorte que deux utilisateurs avec les mmes données finiront pour générer le mme cryptogramme sans avoir besoin d'un échange ou une coordination clé. Une telle solution est connue comme Convergent Encryption (chiffrement convergent) [4], [5].

Dans le Chiffrement Convergent, la clé de chiffrement est habituellement générée en calculant le hachage (sans clé) sécurisé des données. Une fois que la clé a bien été générée, un algorithme de chiffrement déterministe et symétrique est exécuté sur les données afin d'obtenir le cryptogramme. Bien qu'il semble tre un candidat idéal pour combiner la confidentialité et la déduplication, le Chiffrement Convergent a

déjà prouvé tre vulnérable à plusieurs faiblesses qui ne font pas un choix approprié pour garantir la confidentialité des données. Ces faiblesses sont inhérentes aux fonctions de base du Chiffrement Convergent lui-mme. étant donné que la clé de chiffrement est la sortie d'une fonction déterministe qui ne prend que les données à chiffrer en entrée, le fournisseur de stockage Cloud peut exécuter l'attaque suivante: étant donné un texte en clair, l'attaquant peut facilement générer la clé de cryptage correspondante (le hachage sécurisé), produire le cryptogramme et la comparer avec tous les cryptogrammes dans le stockage.

Approches Existantes pour la Déduplication des Données Chiffrées

Les Solutions Basées sur le Secret

Certaines solutions atteignent la confidentialité totale grâce à un secret (par exemple une clé de cryptage) stocké sur un composant de confiance. Selon le chemin, le secret peut tre utilisé pour d'autres données à chiffrer (de manière déterministe, de sorte que la capacité de détecter des doublons ne soit pas perdue) et donc rendre impossible pour un attaquant de générer la clé de cryptage à partir du texte en clair. Par conséquent, en utilisant un secret élimine la possibilité d'exploiter la faiblesse en raison du déterminisme du Chiffrement Convergent. Dans le cas de DupLESS, si un attaquant apprend le secret stocké sur le serveur de clés, la confidentialité ne peut plus tre garantie.

Protocole Côté Client

Une approche intéressante pour faire face au conflit entre la déduplication et de cryptage consiste à permettre aux utilisateurs de détecter de manière autonome si un segment de données est populaire en exécutant un protocole préservant la vie privée avec d'autres utilisateurs (peer-to-peer) ou le fournisseur de Stockage Cloud. Sur la base du résultat obtenu après l'exécution du protocole, l'utilisateur peut déterminer si un segment de données est un doublon et donc compléter la procédure de téléchargement en conséquence.

Au meilleur de notre connaissance, l'une des œuvres les plus récentes et pertinentes qui font usage de ce type d'approche est PAKE [6], dans lequel les auteurs affirment proposer le premier système qui permet d'obtenir la déduplication sécurisé sans l'aide d'aucun serveur supplémentaire. Dans [6], les auteurs introduisent un protocole de partage clé appelée PAKE visant à permettre aux utilisateurs d'effectuer en collaboration avec d'autres utilisateurs la déduplication côté client sans compter sur le soutien d'aucune entité de confiance.

Bien que les auteurs ont mis en œuvre et évalué un prototype de validation de concept qui a prouvé tre très efficace, ce chemin a des limitations inhérentes qui rendent son adoption en scénarios réels un vrai défis. En effet, comme tout autre système peer-to-peer, PAKE nécessite d'un nombre suffisant d'utilisateurs simultanément en ligne afin d'exécuter le protocole avec succès, sinon un potentiel doublon ne peut pas tre détecté, ce qui entrane une diminution importante du ratio de déduplication.

La Popularité des Données comme un Facteur de Différenciation pour le Chiffrement

Certains solutions reposent sur l'hypothèse assez réaliste que les données populaires, qui appartiennent à de nombreux utilisateurs et donc peuvent avoir un impact important sur les économies d'espace de stockage lorsqu'elles sont dédupliquées, sont peu susceptibles de contenir des informations sensibles. D'autre part, les données impopulaires, qui sont rares ou appartiennent à un faible nombre d'utilisateurs et ne doivent donc pas tre dédupliquées, peuvent contenir des informations sensibles et doivent tre protégées par un cryptage plus puissant. Sur la base de cette distinction, un système peut utiliser des mécanismes de chiffrement différents (chacun avec un compromis différent entre possibilité de dédupliquer et la confidentialité) en fonction de la popularité d'un segment de données.

A titre d'exemple de cette approche, les auteurs de [7] ont proposé une solution dans laquelle les données appartenant à plusieurs utilisateurs sont cryptés avec CE afin de permettre au CSP d'effectuer la déduplication. D'autre part, les données impopulaires sont plus susceptibles de contenir des informations sensibles, par conséquent, il est plus sr de chiffrer avec un cryptage sémantiquement sécurisé. Cependant, ce travail souffre de quelques inconvénients. Tout d'abord, le système

souffre d'un surcot de stockage et de transmission important. Deuxièmement, le système proposé dans [7] repose sur un composant de confiance qui fournit un service d'indexation pour toutes les données, à la fois populaires et impopulaires. Troisièmement, à la fois le client et le CSP doivent effectuer des opérations cryptographiques complexes basées sur la "threshold cryptography" sur potentiellement très grandes quantités de données.

Deduplication au Niveau Fichier vs Deduplication au Niveau Bloc

Enfin, un des critères essentiels qui joue un rôle important dans la détermination de la faisabilité d'un système est la capacité de traiter efficacement la déduplication au niveau bloc, ce qui signifie que la déduplication est réalisée avec une granularité plus fine, qui est au niveau des blocs plutôt que des fichiers entiers. Bien que cela puisse paratre simple, dans la pratique, ce n'est pas le cas pour tous les solutions présentées jusqu'à présent.

Contributions

Étude sur la Déduplication

Cette étude a été réalisée avec deux principaux objectifs à l'esprit. Tout d'abord, en raison de l'absence de telles études dans la littérature, nous voulions avoir une preuve tangible de l'efficacité de la déduplication dans des environnements réels. Par conséquent, nous avons mené une étude approfondie sur les différents ensembles de données réelles qui visait à mettre en évidence les meilleures techniques de déduplication et de fournir une bonne approximation des économies qui peuvent tre potentiellement atteintes. Deuxièmement, nous voulions étudier les conséquences en termes de confidentialité lorsque on utilise le chiffrement déterministe en conjonction avec la déduplication au niveau bloc. En particulier, nous nous sommes intéressés à vérifier si une (très) petite taille de bloc pourrait réduire l'entropie globale et donc accroître la vulnérabilité aux attaques statistiques.

La raison pour laquelle nous avons concentré nos efforts sur le chiffrement déterministe est double. Tout d'abord, la grande majorité des solutions qui ont été proposées

dans la littérature, ainsi que dans l'industrie, sont basées sur ce type de cryptage, donc l'étude aurait un impact plus important. Deuxièmement, comme mentionné précédemment, le déterminisme est une exigence posée par la déduplication afin de rendre les données identiques téléchargées par différents utilisateurs comparables après avoir été chiffré.

Cependant, il est bien connu que le chiffrement déterministe souffre d'une limitation inhérente qui l'empche d'atteindre la sécurité maxime (sémantique). Néanmoins, en raison de la nécessité d'tre en mesure de comparer les segments de données chiffrées et de détecter les doublons, le chiffrement déterministe reste une exigence.

à la suite de notre analyse, l'étude a fourni quelques indications précieuses par rapport aux économies de stockage atteintes grâce a la déduplication au niveau bloc et à la sécurité de la déduplication. Plus précisément, la déduplication au niveau bloc avéré tre beaucoup plus efficace que la déduplication au niveau des fichiers, mme en tenant compte du surcot de stockage pour les métadonnées. En outre, sur la base de nos résultats, nous pensons que les attaques statiques basés sur l'analyse de la fréquence restent peu probables, mme en utilisant des tailles de bloc basses mais réalistes.

ClouDedup

Comme une première tentative pour surmonter les problèmes liés à sécuriser la déduplication et de cryptage convergent, nous avons proposé ClouDedup [8], une architecture de stockage en nuage dans laquelle la confidentialité et la déduplication sont sous-traitées par des composants externes et le rôle du fournisseur de stockage est limité au stockage brut de segments de données cryptées.

ClouDedup repose sur deux principaux composants uniques:

1. un composant de cryptage sécurisé (Gateway), déployé chez un tiers de confiance ou, si le client est un organisme, dans les locaux du client et dans le périmètre de sécurité du client;

2. un composant supplémentaire nommé Metadata Manager qui est responsable de la gestion des clés et de déduplication.

La protection des données est garantie grâce à la couche de chiffrement mis en place par la Gateway, qui effectue un cryptage symétrique et déterministe sur le dessus du cryptage convergent, qui est effectué par les clients avant le téléchargement. En outre, ce cryptage est totalement transparent pour l'utilisateur final. En effet, dans la pratique, la Gateway a été mis en œuvre en tant que proxy HTTPS situé comme un "man-in-the-middle" entre les utilisateurs finaux et le Metadata Manager.

En plus de nos composants uniques, nous comptons également sur un fournisseur de Stockage Cloud. En ClouDedup nous transformons le fournisseur de stockage en un composant qui s'occupe de manier extrmement simple du stockage brut de segments de données chiffrées, alors que la déduplication et la gestion des clés sont gérées par le Metadata Manager et le cryptage est effectué par la Gateway. Afin de souligner l'indépendance entre le Metadata Manager et le fournisseur de Stockage Cloud, la communication entre ces composants a été mis en œuvre de façon à couplage lâche en utilisant des API basées sur le modèle REST.

La nécessité d'un composant dédié responsable de la gestion des clés (le Metadata Manager) est soulevée par la nécessité de stocker efficacement et en toute sécurité les clés de cryptage produites par la combinaison de la déduplication au niveau du bloc et le chiffrement convergent. Dans notre implémentation, le Metadata Manager utilise des technologies de pointe dans le domaine des bases de données afin de traiter efficacement cette question. En particulier, nous utilisons une base de données NoSQL open-source appelée REDIS [9] qui, selon notre analyse de la performance, avérée extrmement efficace et fiable.

Cependant, en dépit de sa capacité à apporter le meilleur des deux mondes, soit la confidentialité et des économies d'espace de stockage, ClouDedup nécessite de maintenir et de surveiller une architecture complexe o la confidentialité peut tre garantie tant que la clé secrète stockée sur la Gateway reste sre et n'est pas divulguée à toute entité malveillante.

PerfectDedup

Afin d'atteindre toute confidentialité sans introduire une architecture complexe qui découragerait l'adoption des utilisateurs, nous avons étudié la possibilité de réduire au minimum l'utilisation de composants supplémentaires et utiliser différents niveaux de protection en fonction de la popularité du segment de données. En effet, la

popularité est mesurée par le nombre de propriétaires d'un segment de données donné, afin de déterminer le niveau de confidentialité: plus le nombre d'utilisateurs possédant un segment de données est élevé, plus le segment est populaire.

Sur cette base, un seuil global de popularité est défini dans le système de sorte que chaque fois que le nombre de propriétaires d'un segment de données dépasse le seuil, le segment de données devient populaire. Nous supposons également que si populaire, un segment de données est susceptible d'tre non-confidentiel, par opposition à des segments de données rares qui peuvent contenir des informations sensibles et donc très confidentielles. Une fois que l'état d'un segment de données a été évalué, différentes techniques de cryptage sont utilisées, à savoir le chiffrement convergent est utilisé pour les segments de données populaires (afin de permettre la déduplication) et le cryptage sémantiquement sécurisé autrement (afin d'atteindre le plus haut niveau de confidentialité).

Détecter les segments de données populaires d'une manière préservant la vie privée oblige les utilisateurs à exécuter un protocole de recherche sécurisé avec un fournisseur de Stockage Cloud malveillant. Dans PerfectDedup [10], nous avons introduit un nouveau protocole de recherche sécurisé basé sur une version sécurisée de l'Hachage Parfait [11], grâce à laquelle l'utilisateur peut de manière autonome et en toute sécurité de détecter si oui ou non un segment de données peut tre dédupliquées par simple interroger le fournisseur de Cloud Storage. Une telle approche, qui nous appelons PerfectDedup, permet à l'utilisateur de détecter en toute sécurité si un segment de données peut tre dédupliqué sans révéler aucune information exploitable au fournisseur malveillant de stockage Cloud. Par conséquent, en PerfectDedup nous réalisons la deduplication sécurisé côté client au niveau du bloc, ce qui signifie que les utilisateurs peuvent également économiser en bande passante.

En plus des économies de bande passante, un autre avantage majeur de PerfectDedup par rapport à ClouDedup est l'architecture beaucoup plus simple. En effet, par opposition à ClouDedup, PerfectDedup ne nécessite pas de composants de confiance en ce qui concerne les opérations de chiffrement et de déduplication, car la déduplication des blocs est entièrement gérée côté client grâce à un nouveau protocole de recherche sécurisé et efficace basé sur une version sécurisée de Hachage Parfait. Cependant, de façon similaire à ClouDedup, les deux solutions sont construites sur le dessus de chiffrement convergent et visent à le renforcer en

éliminant la possibilité d'une attaque de type dictionnaire par un fournisseur de stockage Cloud curieux.

Étude sur la Déduplication

Chiffrement Déterministe Toute solution pratique pour la réalisation de la déduplication sécurisée des données cryptées ont besoin du chiffrement déterministe comme principal bloc de construction. Une telle exigence est soulevée par la nécessité de générer des cryptogrammes identiques a partir de blocs identiques, mme si le cryptage est effectué par les différents utilisateurs. Si une telle exigence n'est pas remplie, la déduplication devient impossible en raison de l'incapacité du fournisseur de Stockage Cloud de détecter que deux cryptogrammes différents sont le résultat du chiffrement de la mme donnée.

Attaques Statistiques Malheureusement, l'application du chiffrement déterministe sans aucune amélioration peut ouvrir la voie à une potentielle menace de confidentialité qui nécessite d'tre évalué. En effet, un adversaire avec un accès direct à la mémoire comme le fournisseur de Stockage Cloud, peut perpétrer une attaque grâce à la connaissance d'un certain nombre d'informations qu'il peut obtenir sans avoir besoin de décrypter les données des utilisateurs. Grâce à ces informations, un attaquant peut tre en mesure d'exécuter des attaques statistiques, principalement basées sur l'analyse de la fréquence, sur les cryptogrammes, afin de les lier aux données en claire et ainsi briser la protection offerte par la couche de chiffrement.

Dans le cadre de la déduplication au niveau bloc, étant donné qu'une technique de chiffrement totalement déterministe est nécessaire, une telle attaque peut permettre à l'attaquant d'exploiter la présence de données statiques (par exemple, les en-ttes prédéfinis) dans les données en claire afin de compromettre efficacement le cryptage pour une partie des données chiffrées et essayer de deviner la partie restante à travers une attaque de force brute de type dictionnaire. En outre, vue que la faisabilité des attaques basées sur la fréquence est liée à la distribution des segments de données, il pourrait y avoir une relation directe entre la taille de segment de données et sa distribution, ce qui signifie qu'une taille de segment de données trop faible peut rendre les attaques statistiques plus faciles. S'il existe une telle relation, nous cherchons à déterminer si l'augmentation de la taille des blocs peut atténuer ce problème et, si tel est le cas, quelle taille de bloc assure

un compromis optimal entre la confidentialité, ce qui est notre objectif principal, et le ratio de déduplication, qui devrait rester assez élevé pour ne pas perdre les avantages de la déduplication au niveau bloc.

Notre étude Compte tenu de ces considérations sur la probabilité d'une attaque statistique, nous avons décidé d'effectuer une étude approfondie afin d'évaluer si l'information divulguée en raison de l'utilisation du chiffrement déterministe en conjonction avec la déduplication peut effectivement mettre la confidentialité des données à risque.

Afin d'atteindre cet objectif et rendre l'étude encore plus précieuse, nous avons d'abord réalisé une série d'expériences avec deux principaux objectifs: d'abord, nous voulions avoir une preuve tangible de l'efficacité de la déduplication dans des scénarios réelles; deuxièmement, nous voulions mener notre étude de sécurité en utilisant une configuration de déduplication réaliste, y compris la taille moyenne d'un bloc pour la déduplication au niveau bloc. Ce dernier est un point de départ fondamental pour l'étude de la sécurité car elle impacte la distribution des fréquences des blocs et, en conséquence, la probabilité d'une attaque statistique.

Expériences et Résultats Pour les raisons mentionnées ci-dessus, nous avons réalise un certain nombre d'expériences sur plusieurs jeux de données réelles appartenant aux étudiants et le personnel de l'institut de recherche EURECOM. Le but de ces expériences était double. Tout d'abord, nous avons voulu vérifier les économies d'espace qui sont réalisables en mettant en place une solution de déduplication des données. En particulier, nous avons voulu savoir quels types de fichiers sont plus susceptibles de contenir des doublons et la technique de coupage qui peut obtenir les économies d'espace les plus élevées. En outre, nous avons effectué la mme analyse avec des techniques différentes de coupage et en changeant la taille des segments de données ainsi que la granularité.

Les résultats ont fourni le point de départ nécessaire pour l'étude de la sécurité, c'est à dire la technique de coupage la plus efficace et des tailles moyennes réalistes. À cet égard, une approche avec une taille variable et en moyenne de 4, 8 ou 16 KB avérée tre la stratégie la plus efficace. En outre, ces résultats ont fourni des conseils intéressants qui peuvent tre utilisés comme un support d'information supplémentaire lors de l'évaluation de la stratégie de déduplication qui doit d'tre utilisés dans un contexte réel.

Nous avons ensuite utilisé cette configuration pour analyser la distribution des segments de données popularité, mesurer l'entropie de l'ensemble des données et observer son évolution à travers différentes tailles de blocs moyens. A partir de cela, nous avons étudié l'existence des menaces à la confidentialité introduites par la déduplication et vérifié si et quand les informations détenues par le fournisseur sont suffisantes pour effectuer une attaque statistique basée sur l'analyse de la fréquence, qui a le pouvoir de causer la fuite d'informations sensibles sur les données des utilisateurs. En bref, le résultat de cette analyse a prouvé que les attaques statistiques basées sur la fréquence ne sont pas réalisables dans un tel contexte.

Au meilleur de notre connaissance, ceci est la première étude qui a pris en considération les attaques statistiques sur les données dédupliquées. Sur la base des résultats de cette étude, nous serons alors en mesure de concevoir des systèmes de déduplication de données sécurisée.

ClouDedup

En ClouDedup nous envisageons le conflit inhérent entre le chiffrement et la déduplication en concevant une nouvelle architecture, qui nous appelons de ClouDedup, o les fichiers sont protégés par deux couches de chiffrement déterministe: la première couche est un chiffrement convergent et est appliquée par les utilisateurs avant de télécharger le fichier; la deuxième couche est appliquée par un composant de confiance qui fait usage d'une clé secrété stockée localement et jamais partagée avec une autre entité. Une telle architecture vise également à déléguer différentes tâches à différentes composantes d'une façon qu'un composant sans la coopération d'un autre composant n'a pas de connaissances suffisantes pour compromettre la confidentialité des données. En d'autres termes, ce système permet d'obtenir un principe connu comme "single point of failure", qui dans ce contexte signifie que la compromission d'un seul composant ne peut pas compromettre complètement la sécurité de l'ensemble du système.

La sécurité de ClouDedup repose donc sur sa nouvelle architecture selon laquelle, en plus du fournisseur de stockage de base, un gestionnaire de métadonnées (Metadata Manager) et un composant de cryptage supplémentaire, appelée Gateway, sont définis: la Gateway empche les attaques connues contre le chiffrement

convergent et aussi protège la confidentialité des données; d'autre part, le Metadata Manager est responsable de la tâche de gestion de clés à cause de la nécessité de mémorisation d'un grand nombre de clés produites par la conjonction entre la déduplication au niveau bloc et le chiffrement convergent. Par conséquent, la déduplication est effectuée au niveau du bloc et nous définissons un mécanisme très efficace de gestion des clés afin d'éviter aux utilisateurs de stocker une clé par bloc.

Pour résumer les avantages de ClouDedup:

- ClouDedup assure la **déduplication au niveau bloc** et la **confidentialité des données**. La déduplication au niveau bloc rend le système plus **flexible et efficace**;
- ClouDedup préserve la **confidentialité et la vie privée, mme contre les fournisseurs de stockage Cloud potentiellement malveillants** grâce à une couche supplémentaire de chiffrement déterministe;
- ClouDedup offre une solution de gestion de clé efficace grâce au composant Metadata Manager;
- La nouvelle architecture définit plusieurs composants et un **seul composant ne peut pas compromettre** l'ensemble du système sans collusion avec d'autres composants;
- ClouDedup fonctionne de manière **transparente** avec les fournisseurs de stockage en nuage existants. En conséquence, ClouDedup est entièrement compatible avec les API de stockage standard et nimporte quel fournisseur de stockage Cloud peut tre facilement intégré dans notre architecture.

Évaluation

Fournisseur de Stockage Curieux Comme indiqué dans la section de modèle de menace, nous supposons qu'un attaquant, comme un fournisseur de stockage malveillant, dispose d'un accès complet au stockage. Si l'attaquant a seulement accès à la mémoire, il ne peut obtenir aucune information. En effet, les fichiers sont divisés en blocs et chaque bloc est tout d'abord chiffré avec un chiffrement convergent et ensuite encore chiffré avec une ou plusieurs clés secrètes, toujours

au moyen d'un mécanisme de chiffrement déterministe. Comme indiqué pendant notre étude, le chiffrement déterministe peut effectivement fournir toute confidentialité. De plus, aucune métadonnée (propriétaire du fichier, nom de fichier, la taille du fichier, etc.) est stockée au niveau du fournisseur de stockage Cloud. Il est clair que, grâce à cette configuration, l'attaquant n'est pas en mesure d'effectuer une attaque de type dictionnaire sur les fichiers prévisibles.

Metadata Manager Compromis Un scénario pire est celui dans lequel l'attaquant parvient à compromettre le gestionnaire de métadonnées et a donc accès à des données, des métadonnées et aussi des clés cryptées. Dans ce cas, la confidentialité sera toujours garantie puisque les clés des blocs sont cryptées avec les clés secrètes des utilisateurs et la clé secrète de la Gateway. La seule information que l'attaquant peut obtenir est la similitude des données et les relations entre les fichiers, les utilisateurs et les blocs. Cependant, comme les noms de fichiers sont cryptés par les utilisateurs, ces informations ne seraient d'aucune utilité pour l'attaquant, à moins qu'il ne parvient à trouver une correspondance avec un fichier prévisible en fonction de sa taille et popularité. En outre, comme indiqué dans notre étude, le chiffrement déterministe assure la confidentialité mme lorsqu'il est utilisé en conjonction avec la déduplication au niveau bloc. En effet, les attaques uniquement basées sur l'analyse de la fréquence des blocs ne semblent pas tre réalisables dans des scénarios réels.

Gateway Compromise Le système doit garantir la confidentialité mme dans le cas improbable o la Gateway est compromise. Un cryptage supplémentaire effectué par le Metadata Manager avant d'envoyer les données au fournisseur de stockage sera alors appliqué afin de garantir la protection des données; grâce à cette couche de cryptage donc la confidentialité est toujours garantie et les attaques de dictionnaire hors ligne ne sont pas possibles. D'autre part, si l'attaquant compromet la Gateway, seulement les attaques en ligne seraient possibles puisque ce composant communique directement avec les utilisateurs. L'effet d'une telle violation est limité puisque les données téléchargées par les utilisateurs sont cryptées avec un chiffrement convergent, qui offre la confidentialité pour les fichiers imprévisibles [5]. En outre, une stratégie de limitation du débit mis en place par le gestionnaire de métadonnées peut limiter les attaques par force brute en ligne effectuées par la passerelle.

Gateway et Metadata Manager Compromises Dans le pire scénario, l'attaquant

parvient à obtenir toutes les clés secrètes en compromettant la Gateway et le Metadata Manager. Dans ce cas, l'attaquant sera en mesure de supprimer les deux couches supplémentaires de cryptage et d'effectuer hors attaques de type dictionnaire sur les fichiers prévisibles. Cependant, puisque les données sont cryptées avec le chiffrement convergent par les utilisateurs, la confidentialité des fichiers imprévisibles est toujours garantie.

Attaquant externe Finalement, nous analysons l'impact d'un attaquant qui tente de compromettre les utilisateurs mais n'a pas accès à la mémoire. Si un attaquant arrive à compromettre un ou plusieurs utilisateurs, il peut tenter d'effectuer les attaques de type dictionnaire en ligne. Comme la Gateway et le Metadata Manager n'ont pas été compromises, l'attaquant ne peut récupérer que des données appartenant à l'utilisateur compromis grâce au mécanisme de contrôle d'accès. En outre, comme mentionné ci-dessus, la passerelle peut limiter des telles attaques en fixant un seuil maximal pour la vitesse avec laquelle les utilisateurs peuvent envoyer des requtes.

PerfectDedup

ClouDedup obtient la déduplication sécurisée au niveau bloc au prix d'exiger une architecture complexe o l'opération de chiffrement la plus cruciale est déléguée à un composant de confiance. En outre, comme mentionné dans la section d'analyse de la sécurité, un Metadata Manager peut coopérer avec un ou plusieurs utilisateurs afin de contourner la protection garantie par la couche de cryptage supplémentaire et exécuter avec succès des attaques de type COF et LRI.

A partir de ces deux inconvénients, nous visons à la conception d'un système, appelé PerfectDedup, avec une architecture plus simple o les utilisateurs pourraient évaluer de manière autonome si un bloc peut tre dédupliqué en exécutant un protocole confidentiel auprès d'un fournisseur de stockage Cloud malveillant. Une telle approche aurait l'avantage supplémentaire et non négligeable de permettre la déduplication côté client, qui apporte des économies de bande passante en plus des économies d'espace de stockage. Grâce a ce protocole, PerfectDedup combine en toute sécurité et efficacement la déduplication au niveau bloc (parmi les fichiers de tous les utilisateurs) et la confidentialité contre les fournisseurs de stockage Cloud potentiellement malveillants. En outre, ces objectifs sont atteints sans compter

sur une entité de confiance à l'égard de l'opération de chiffrement. Contrairement à ClouDedup, ce système permet également de bénéficier de la déduplication côté client, ce qui signifie qu'un client peut en toute sécurité vérifier si un bloc est un doublon avant de le télécharger et de le chiffrer.

Popularité des Données Dans PerfectDedup, nous proposons de régler les vulnérabilités du chiffrement convergent en tenant compte de la popularité [7] des données. Les données (blocs) stockées par un grand nombre d'utilisateurs, c'est à dire les données populaires, sont protégées avec la mécanisme de chiffrement convergent faible tandis que les données impopulaires, qui sont rares, sont protégés par un chiffrement sémantiquement sécurisé. Cette déclinaison de mécanismes de chiffrement se prte parfaitement à la déduplication efficace puisque les données populaires qui sont cryptées sous le chiffrement convergent sont aussi ceux qui ont besoin d'tre dédupliquées. Ce système assure également la sécurité des données sensibles grâce à la protection forte fournie par le chiffrement sémantiquement sécurisé alors que les segments de données populaires, qui souffrent des faiblesses du chiffrement convergent, sont beaucoup moins sensibles car ils sont partagée par plusieurs utilisateurs.

Néanmoins, cette approche pose un nouveau défi: les utilisateurs doivent décider de manière autonome de la popularité de chaque bloc avant de le stocker et le mécanisme par lequel la décision est prise ouvre la voie à une série de faiblesses très similaires à ceux avec le chiffrement convergent. L'objectif des chemins basés sur la popularité devient alors la conception d'un mécanisme sécurisé pour déterminer la popularité des données.

Nous proposons un nouveau schéma pour la déduplication sécurisée des données chiffrées, sur la base du principe de la popularité. Le principal composant de ce système est un mécanisme original pour détecter la popularité des segments de données d'une manière parfaitement sécurisée. Les utilisateurs peuvent rechercher les données dans une liste de blocs populaires stockées par le fournisseur de stockage Cloud (CSP) sur la base des identifiants des blocs de données calculées avec une fonction de hachage parfait (PHF). Grâce à cette technique, il n'y a aucune fuite d'informations sur les données impopulaires et en mme temps les données populaires sont très efficacement identifiés. Sur la base de cette nouvelle technique de détection de la popularité, notre système réalise la déduplication des données chiffrées au niveau des blocs d'une manière parfaitement sécurisée.

Les avantages de notre système peuvent tre résumées comme suit:

- PerfectDedup permet d'obtenir la réduction de la taille par la déduplication des données populaires;
- PerfectDedup repose sur des algorithmes de chiffrement symétrique, qui sont connus pour tre très efficace, mme lorsqu'ils sont utilisés avec de volumétries importantes;
- PerfectDedup atteint la déduplication au niveau des blocs, ce qui conduit à une augmentation des économies d'espace de stockage par rapport à la déduplication au niveau fichier [12];
- PerfectDedup ne nécessite aucune coordination ou initialisation entre les utilisateurs;
- PerfectDedup ne comporte pas de surcot de stockage pour les données impopulaires;

Analyse de Sécurité

Dans cette section, nous analysons la sécurité du schéma proposé, le CSP étant considéré comme le principal adversaire. Le CSP est "honnte, mais curieux", ce qui signifie qu'il effectue correctement toutes les opérations, mais il peut essayer de découvrir le contenu original des données impopulaires. Nous ne considérons pas les scénarios o le CSP se comporte d'une manière byzantine. Nous supposons que CSP ne peut pas agir de concert avec l'IS vu que nous faisons confiance à ce composant. étant donné que le but du CSP malveillant est de découvrir le contenu des blocs impopulaires, nous analysons en détail si (et comment) la confidentialité est garantie pour les données impopulaires pendant toutes les phases du protocole. Enfin, nous analysons aussi des attaques qui peuvent tre perpétrées par les utilisateurs eux-mmes et proposons des contre-mesures simples contre eux.

Sécurité des Blocs Stockés chez le CSP Par définition, un bloc impopulaire est crypté à l'aide d'un chiffrement symétrique sémantiquement sécurisé. La confidentialité des données impopulaires est ainsi garantie grâce à la sécurité du mécanisme de chiffrement utilisé.

Sécurité pendant le Popularity Check Les informations échangées au cours de la phase Popularity Check ne doivent pas révéler aucune information qui pourrait fuir l'identité d'un bloc impopulaire appartenant à l'utilisateur. L'identité d'un bloc impopulaire est protégée grâce à la propriété de One-Wayness de l'algorithme de Hachage Parfait sécurisé (PHF: Perfect Hash Function): la requte générée par le client ne contient pas l'ID de bloc impopulaire réel, mais un entier i qui est calculé en évaluant le Hachage Parfait sécurisé sur l'ID du bloc. Essayer de deviner en explorant tous les possibles résultats de la fonction de hachage sécurisée intégrée dans e PHF n'est pas faisable grâce à la propriété de One-Wayness de la fonction de hachage sécurisée (SHA-2 [13]). En plus de cela, lorsque le PHF est évalué sur l'ID d'un bloc impopulaire, il y aura certainement une collision entre l'ID du bloc impopulaire et l'ID d'un bloc populaire déjà stocké chez le CSP. Ces collisions servent de contre-mesure principale à la divulgation de l'ID de bloc impopulaire envoyé au CSP au cours de la recherche. Avec une hypothèse raisonnable, nous pouvons aussi considérer que la sortie de la fonction de hachage sécurisée (SHA-2) est aléatoire. Dans le cas d'une collision entre un ID de bloc impopulaire et l'ID d'un bloc populaire stocké au CSP, grâce au caractère aléatoire de la fonction de hachage sécurisée, la sortie d'un PHF fondé sur une telle fonction de hachage est répartie uniformément entre 0 et m. En supposant que la cardinalité de l'ensemble du domaine est beaucoup plus grande que la cardinalité de l'ensemble des identifiants de blocs populaires (ce qui est le cas si les ID de blocs populaires sont le résultat d'une fonction de hachage sécurisée), nous pouvons affirmer que le nombre de collisions pour chaque position du tableau de hachage est assez grande pour empcher un CSP malveillant d'inférer l'ID de bloc utilisé comme entrée pour le PHF. D'o les collisions peuvent effectivement cacher l'identité de blocs impopulaires auprès d'un fournisseur de stockage Cloud malveillant tout en gardant le protocole de recherche extrmement efficace et léger pour les utilisateurs.

Sécurité contre les potentielles Vulnérabilités du Protocole Nous considérons maintenant quelques attaques supplémentaires qui peuvent tre perpétrées par le CSP. Pour chacun d'eux, nous proposons des contre-mesures simples mais efficaces, qui sont faciles à mettre en œuvre et n'augmentent pas de manière significative le temps de calcul et de réseau.

Tout d'abord, nous considérons que le CSP peut pré-construire un PHF sur la base de certaines données spécifiques (provenant par exemple d'un dictionnaire) qui n'ont pas encore été téléchargées par les utilisateurs. Dans un tel scénario,

les clients détecterons leur bloc demandé comme populaire mme si il n'a jamais réellement été téléchargé par un utilisateur; un tel bloc sera ensuite stocké avec un niveau de protection inférieur. En tant que contre-mesure à une telle attaque, nous proposons que le IS attache une signature à chaque ID de bloc populaire sur lors de transition de popularité. Par conséquent, l'IS signera l'ID du bloc populaire avant d'tre stockés chez le CSP, permettant aux clients de vérifier l'authenticité de ces blocs lors de l'exécution du contrôle de popularité (Popularity Check). Une telle contre-mesure aurait un effet minimal sur les performances du système.

Une autre attaque que nous considérons est liée à l'attaque de confirmation de fichier auquel le chiffrement convergent est également vulnérable [14]. En effet, sur un Popularity Check, le CSP peut comparer la séquence des indices envoyés par le client avec la séquence produite par un fichier populaire donné F. Si les deux séquences correspondent, alors il y a une chance que le client est effectivement en train de télécharger F. Dans afin de cacher cette information de la CSP, le client peut ajouter un certain nombre d'indices aléatoires à la liste des indices étant envoyés lors du Popularity Check. Grâce au bruit résultant inclus dans la liste d'indices, l'identification du fichier cible par le CSP sera beaucoup plus difficile. Cette contre-mesure empche également le CSP d'exécuter l'attaque connu avec le nom "learn-the-remaining-information". En outre, le surcot généré par cette contre-mesure est négligeable en termes de bande passante et de puissance de calcul.

Conclusion

Étude sur la Deduplication

Tout d'abord, à partir de l'observation que la déduplication soulève une exigence pour le chiffrement déterministe, nous avons évalué si et dans quels scénarios la déduplication au niveau bloc peut ouvrir la voie à des attaques statistiques. Afin de répondre à cette question, nous avons effectué une analyse complète sur un ensemble de données réelles et représentatives qui a conduit à trois résultats intéressants:

- Après avoir comparé les techniques de coupage les plus communs sur plusieurs ensembles de données, les résultats montrent clairement que la déduplication

au niveau bloc atteint toujours les meilleurs ratios de déduplication. En outre, le surcot pour les métadonnées générées par la déduplication au niveau bloc ne supprime pas le gain en termes d'économies d'espace de stockage, ce qui est remarquable.

- Lorsqu'on utilise des tailles de blocs réalistes (allant de 4Ko à 32Ko), l'entropie globale ne semble pas diminuer quand on diminue la taille moyenne de bloc, donc il n'y a aucune preuve concrète qu'une attaque statistique basée sur l'analyse de fréquence peut tre réalisée avec succès. En outre, le nombre total de blocs est tel qu'une attaque statistique devient extrmement difficile.

- Comme un résultat supplémentaire, nous avons également analysé la popularité de chaque bloc dans l'ensemble de données afin d'en savoir plus sur la distribution de la popularité et trouver une valeur telle que les économies d'espace de stockage restent élevées alors qu'il n'y a aucun risque pour la confidentialité. Ceci est particulièrement utile pour les systèmes tels que PerfectDedup qui sont basés sur un seuil de popularité. Comme prévu, la distribution de popularité est loin d'tre uniforme. En effet, en augmentant légèrement le seuil de popularité à 2 ou 3 provoque une chute des économies d'espace de stockage de plus de 15 %, quelle que soit la technique de coupage en blocs utilisée. Par conséquent, sur la base de ces données, nous suggérons que le seuil de popularité ne devrait pas tre supérieur à 5.

Grâce à cette étude, d'un point de vue pratique, nous avons appris la difficulté de construire un système qui vise à stocker les métadonnées et peut évoluer jusqu'à très grandes volumétries de données, par exemple des milliers de Gigaoctets. Nous avons prouvé que le recours au stockage clé-valeur est une solution très efficace. Cependant, des contre-mesures supplémentaires doivent tre prises afin d'tre en mesure de traiter de grandes quantités de métadonnées qui ne peuvent pas tenir entièrement dans la mémoire.

ClouDedup

ClouDedup était notre première tentative de fournir une solution sre et pratique pour atteindre la confidentialité avec déduplication au niveau bloc. Nous avons conçu un système qui utilise un certain nombre de couches de chiffrement déterministe et symétrique supplémentaires afin de faire face aux faiblesses du

chiffrement convergent. Des couches supplémentaires de cryptage sont ajoutés par la Gateway et, éventuellement, par le Metadata Manager. Comme les cryptages supplémentaires sont symétriques, l'impact sur les performances est négligeable. En plus de cela, nous avons montré que c'est la peine d'effectuer la déduplication au niveau bloc au lieu de la déduplication au niveau fichier vu que les gains en termes d'espace de stockage ne sont pas impactés par le surcot de gestion des métadonnées, ce qui est minime.

Nous avons également montré que notre conception, dans laquelle aucun composant est complètement de confiance, empche tous les composants de compromettre la sécurité du système sans coopérer avec un autre composant. Notre solution empche également les fournisseurs de stockage Cloud malveillants de déduire le contenu original des données stockées en observant les accès ou en accédant à des métadonnées. En outre, nous avons implémenté un prototype complet, qui a montré que notre solution peut tre facilement et efficacement mise en œuvre avec des technologies existantes et généralisées.

Dans le cadre des travaux futurs, ClouDedup peut tre étendu avec plus de fonctionnalités de sécurité telles que les "Proofs of Retrievability" [15], l'intégrité des données [16], la recherche sur les données cryptées [17], [18] et le partage sécurisé de fichiers [19], ce qui est une caractéristique très appréciée dans le stockage Cloud. Dans un proche avenir, nous visons à compléter la mise en œuvre du prototype actuel afin d'en faire un système de stockage Cloud prt à la production et le déployer dans des scénarios réels. De plus, nous allons travailler sur la recherche d'optimisations en termes de bande passante, d'espace de stockage et de calcul.

PerfectDedup

Sur la base de l'expertise gagnée au cours de la conception et le développement de ClouDedup, nous avons décidé de travailler sur un nouveau projet visant à résoudre le mme problème tout en faisant face aux lacunes existantes, parmi lesquelles il y a:

- **Pas de déduplication coté client:** dans ClouDedup, les clients téléchargent toutes les données vers la Gateway qui les crypte et les transmet au Metadata Manager, o finalement la déduplication a lieu. Cela signifie que, le système ne permet pas de réaliser des économies de bande passante.

- **Architecture complexe:** ClouDedup nécessite de déployer une architecture complexe dans laquelle un composant, qui est la porte d'entrée, doit tre digne de confiance par rapport à l'opération de chiffrement. Cela signifie que si la Gateway est compromise la confidentialité n'est plus garantie.

Sur la base de ces lacunes, nous avons conçu un système qui garantit une totale confidentialité pour les fichiers confidentiels, tout en permettant à la déduplication au niveau bloc à la source pour les fichiers populaires. La principale innovation de notre système est un nouveau protocole de recherche sécurisée construit au sommet d'une version améliorée de Hachage Parfait. Au meilleur de notre connaissance, ceci est le premier travail qui utilise le Hachage Parfait pour un but différent de l'indexation de bases de données. Un composant de semi-confiance est employé à des fins de stockage des métadonnées concernant les données impopulaires et en fournissant un support pour détecter les transitions de popularité, ce qui signifie qu'un bloc vient d'atteindre le seuil de popularité.

Ce travail nous a permis d'explorer une approche inhabituelle mais intéressante consistant en l'atteinte de la confidentialité en profitant des collisions. Plus précisément, bien que les collisions sont généralement considérées comme un événement qui devrait tre évité, dans nos cas les collisions sont utiles car elles fournissent la protection principale contre les attaques d'un fournisseur de stockage Cloud curieux et permettent d'atteindre la confidentialité pour tous les utilisateurs. Par conséquent, cette contribution peut ouvrir la voie à de nouvelles approches basées sur l'utilisation de collisions comme un moyen de protéger les informations confidentielles.

Dans le cadre des travaux futurs, PerfectDedup peut tre optimisée afin de réduire le surcot introduit par la génération de la fonction de Hachage Parfait et sa transmission. En particulier, afin de réduire le temps de génération lorsqu'il s'agit de très grandes volumétries de données, le PHF doit tre dynamique, ce qui signifie qu'un ID d'un bloc peut tre inséré ou supprimé sans nécessairement générer de nouveau le PHF à partir de zéro. En plus de cela, il serait utile de disposer d'une solution qui permet à un utilisateur de mettre à jour de maniéré incrémentielle le PHF sans avoir à le télécharger entièrement à chaque modification.

Travaux Futurs

Au moment de l'écriture, la conception d'un système pleinement fonctionnel pour le stockage sr et efficace reste un défi ouvert. Nous suggérons que l'extension de l'un de nos programmes avec de nouveaux mécanismes visant à fournir des fonctionnalités de sécurité manquantes peut tre une direction de recherche fructueuse.

Par exemple, étant donné un système de déduplication sécurisé comme ClouDedup ou PerfectDedup, il serait positif pour les utilisateurs d'intégrer un certain nombre de fonctions de sécurité supplémentaires telles que les "Proofs of Retrievability" (PoR) [15], l'intégrité des données [16], recherche sur des données chiffrées [17], [18] et le partage sécurisé de fichiers [19]. Cependant, une telle intégration peut présenter plusieurs défis en raison de l'utilisation du chiffrement déterministe. En effet, la plupart des approches qui permettent d'atteindre les caractéristiques de sécurité mentionnées ci-dessus utilisent des techniques de chiffrement sémantiquement sécurisés, qui sont connues pour tre incompatibles avec la déduplication. En outre, bien qu'il existe certains solutions [18] basées sur le chiffrement déterministe, ils comptent sur la cryptographie asymétrique, qui est connue pour tre inefficace lors du chiffrement des fichiers volumineux.

En outre, en raison de contraintes pratiques, y compris les limites de stockage, notre étude des données a été réalisée sur un seul cliché de stockage pris à un moment donné. Toutefois, il serait très intéressant d'étudier l'évolution de la confidentialité et des économies d'espace de stockage au fil du temps. Plus précisément, on pourrait prendre plusieurs clichés de stockage à des moments différents (par exemple un cliché par semaine) et comparer leurs entropie et les économies qui peuvent tre atteintes. Cela nous permettra d'observer si et comment les ratios d'entropie et de déduplication changent au fil du temps.

Finalement, une autre option difficile pour les recherches futures est d'étudier si et comment les approches de déduplication sécurisées existantes peuvent tre appliquées dans le contexte des bases de données relationnelles, qui sont souvent utilisés dans des applications à grande échelle. Dans un tel contexte, la granularité et la taille des blocs changent, puisque dans les bases de données la déduplication doit tre effectuée sur les champs au lieu des blocs. Cela pourrait avoir des conséquences graves en ce qui concerne la confidentialité en raison du fait que les blocs seront probablement beaucoup plus petits, à savoir quelques octets au lieu de quelques kilooctets. En outre, afin de conserver les opérations qui sont

habituellement exécutées pendant des requtes à la base de données (par exemple des comparaisons), la solution devrait permettre d'effectuer ces opérations sur les données cryptées, en plus d'utiliser une méthode de chiffrement déterministe.

Contents

List of Figures

List of Tables

To my family, who have always believed in me and did everything to help me reach this important milestone...

To my girlfriend, who made the last three years the most beautiful of my life...

To all my friends and colleagues, who in different times and different ways have contributed to my success...

Chapter 1

Introduction

1.1 Cloud Computing

Since 2006, that is when Amazon introduced Elastic Simple Storage Service (Amazon S3) [20], Cloud Computing adoption has been constantly and exponentially rising. According to the predictions based on the most recent studies [21], this trend does not seem to be slowing down, rather it is expected to keep growing in the next years. What makes Cloud Computing extremely attractive to customers is the possibility of enjoying a resource with the maximum flexibility, hence minimizing the startup and setup cost and time. Cloud Computing leverages the advantages of virtualization technologies in order to build platforms which can satisfy customers' needs while eliminating the complexity of managing complex software and hardware infrastructures. The main reason behind the success of Cloud Computing can thus be linked to the following factors: flexibility, elasticity, low ownership costs and pay-per-use economic model.

Cloud providers can be classified depending on the level at which they provide services and resources. Traditionally, according to the taxonomy defined by NIST [22], Cloud providers are classified into IaaS (Infrastructure as a Service), PaaS (Platform as a Service) and SaaS (Software as a Service). IaaS providers are focused on providing low-level infrastructure services such as virtual servers, storage and network. For instance, this is the case of Amazon EC2 and Amazon S3. PaaS providers are focused on services that are mainly intended for developers. More precisely, "Platform" stands for the set of tools allowing developers to deploy applications without taking care of the underlying infrastructure, which is provisioned

by the Cloud provider. Last but not least, SaaS providers usually offer a service through an application hosted on a remote server and accessible on the Internet. Most of the time, the application is a web application and the remote server is a web server.

1.2 Cloud Storage Security

Cloud storage certainly is one of the most attractive opportunities offered by Cloud Computing. Indeed, both standard users and enterprises need a way to safely and securely store their data without having to maintain complex and expensive infrastructures on their premises. Also, with the widespread adoption of portable devices (e.g. smartphones and tablets), users need to ubiquitously access their data from any location and on any device. Cloud storage brings what is needed to provide the above-mentioned features. However, recent data leaks [23] and security incidents have introduced a general concern about Cloud Storage security and drawn attention to an urgent need for solutions that can effectively guarantee data confidentiality and protect users' data from malicious entities. For instance, thanks to the recent revelations of Edward Snowden [24] and other recent scandals, everyone is now aware of the potential threats posed by secret agencies, crackers, insiders and malicious users in general. Therefore, this is one more reason why we cannot rely on the assumption that Cloud storage providers are trusted entities, since they are vulnerable to several, either internal or external, threats which are not under the control of the user. In addition to that, a "curious" Cloud provider may use customers' data in order to perform lucrative activities such as data mining or any other activity involving direct access to stored data. An effective solution for data confidentiality must protect users' data against any external or internal threat and unauthorized access.

1.3 Secure Data Deduplication

1.3.1 Deduplication

Along with security and confidentiality, users also demand low ownership costs and high quality of service. On the other hand, Cloud storage providers constantly

look for techniques aimed at optimizing communication overhead, improving performance and most importantly reducing operational costs in order to increase their profit.

When it comes to cloud storage, these requirements can be met by reducing data footprint both in transit and at rest. In this regard, Cloud Storage providers have observed that users tend to upload a non-negligible amount of redundant data. That is the reason why all major Cloud storage providers have put in place several strategies aimed at detecting redundant data in order to save storage space and speed up data transfers between clients and cloud storage servers.

One of the techniques that has recently been adopted by many Cloud storage providers is deduplication, which has proved to achieve significant storage space savings, especially in backup applications [1], [2]. Deduplication arises from the following observation: it is very common that different users may upload identical or very similar files. In this case, storing multiple copies of the same data segments would be a useless waste of storage resources. Therefore, instead of merely storing data uploaded by users, the Cloud storage provider may first check whether a particular data segment has already been uploaded in the past: this is usually done by comparing its ID with a database containing the IDs of all data segments stored at the Cloud. If so, there is no need to store it again since the Cloud storage provider may create a logic link to the actual data segment, as illustrated in Figure 1.1. Thanks to the effectiveness of such an idea, despite its simplicity, deduplication has become a de-facto standard [3] in all major massive storage systems thanks to its ability of greatly reducing data footprint, hence reducing storage costs and bandwidth consumption.

1.3.2 Deduplication of Encrypted Data

However, as stated above, customers' demand also includes confidentiality, which day by day is becoming a pressing issue for enterprises as well as for standard users. Therefore, the main challenge in Cloud Storage becomes how to meet the conflicting needs of customers and providers which, from a confidentiality point of view, means securely and efficiently combining encryption with storage optimization techniques such as deduplication. Although it may seem straightforward, putting the two techniques together is far from being trivial: the desired result

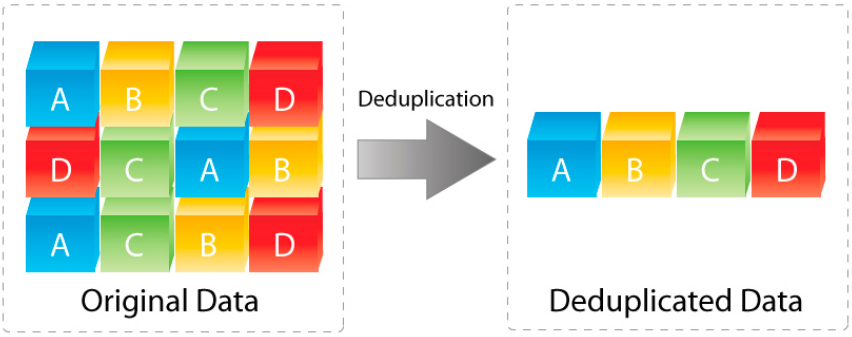

FIGURE 1.1: Illustration of data deduplication.

of encryption is to make encrypted data (the ciphertext) unreadable and indistinguishable from original data (the plaintext), whereas deduplication requires the Cloud storage provider to be able to easily compare two (encrypted) data segments and determine whether they are identical. In other words, as illustrated in Figure 1.2, in a standard scenario deduplication of encrypted data will not be feasible, unless encryption keys are leaked to the Cloud storage providers, which would give them the power to read users' data, hence compromise data confidentiality. From this issue, the following problem arises: how can we preserve data confidentiality without preventing an untrusted Cloud storage provider from detecting duplicate data? The answer to this question is not trivial. In a common scenario, two users who have the same data and want to securely store it in the Cloud, will encrypt their data before performing the upload. Unfortunately, a standard encryption technique would make deduplication impossible since two different users would use two different keys, resulting in two different ciphertexts.

1.3.3 Convergent Encryption

Recently, a simple but effective countermeasure has been proposed as a solution to this problem: the encryption key may be generated in a completely deterministic and data-dependent way, so that two users with the same data will end up with generating the same ciphertext without needing any key exchange or coordination. Such a solution is known as Convergent Encryption [4], [5] and is illustrated in Figure 1.2. In Convergent Encryption, the encryption key is usually generated by calculating the secure unkeyed hash of the data. Once the key has been generated,

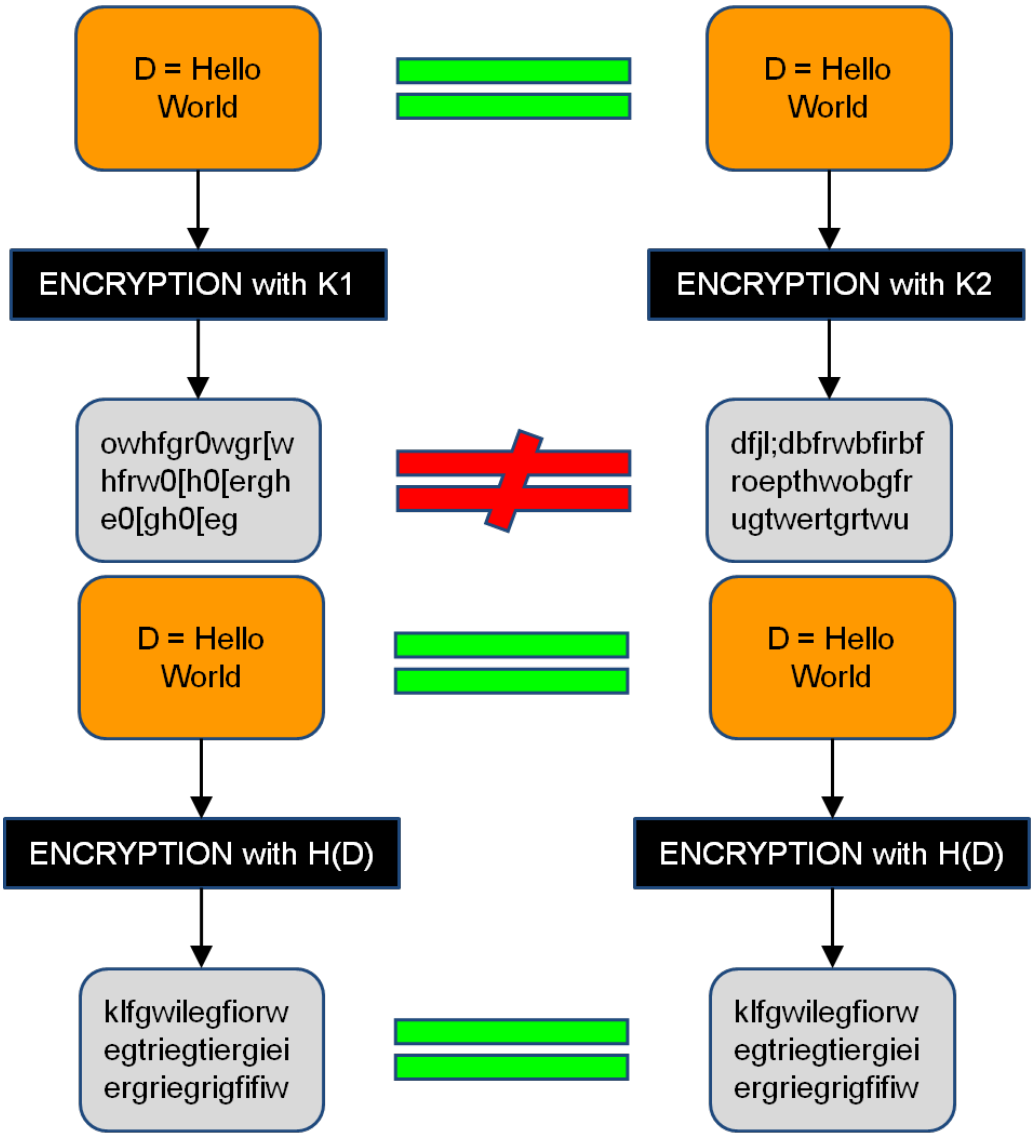

FIGURE 1.2: Example of the conflict between deduplication and encryption.

a symmetric deterministic encryption algorithm is executed over the data in order to obtain the ciphertext.

Although it seems to be an ideal candidate for combining confidentiality and deduplication, Convergent Encryption has proved to be vulnerable to several weaknesses which do not make it a suitable choice to guarantee data confidentiality. These weaknesses are inherent to the basic features of Convergent Encryption itself. Since the encryption key is the output of a deterministic function which only takes the data to encrypt as input, anyone, including the Cloud storage provider, may run the following attack: given a plaintext, the attacker may easily generate the corresponding encryption key (the secure hash), produce the ciphertext and

compare it with all ciphertexts in storage. Such an attack, which can be seen as an instance of the brute-force/dictionary attack, may effectively reveal whether a plaintext is already stored. Clearly, such a weakness is not acceptable since it severely undermines data confidentiality.

1.4 Existing Approaches for Secure Data Deduplication

As explained earlier, Convergent Encryption leverages determinism in both key generation and encryption in order to solve the conflict between deduplication and confidentiality. However, the weaknesses from which it suffers are not acceptable and need to be addressed.

Recently, several solutions have been proposed in both the literature and the industry. We now review the most relevant ones and categorize them according different criteria while briefly explaining their drawbacks. More details on each of these solutions will be given in next chapter.

1.4.1 Solutions based on Convergent Encryption

The first criteria we use for categorizing existing solutions is whether Convergent Encryption, or a slightly different variant, is involved in the data protection process. More precisely, some solutions may still rely on Convergent Encryption as a first layer of encryption or by changing the key generation mechanism.

As an example, this is the case of DupLESS [25] and Tahoe-LAFS [26], which is based on a privacy-preserving protocol based on pseudo-random functions running between the user and a trusted key server. Thanks to this protocol a user can generate a deterministic and data-dependent encryption key without disclosing any information on the file being encrypted, so that all users will encrypt the same file with the same key, resulting in the same ciphertext that can be easily deduplicated by the untrusted Cloud Storage Provider.

1.4.2 Solutions based on Secrecy

Some solutions achieve full confidentiality thanks to a secret (e.g. an encryption key) stored on a trusted component. Depending on the scheme, the secret can be used to further encrypt data (in a deterministic manner, so that the ability of detecting duplicates is not lost) or make impossible for an attacker to generate the encryption key starting from the plaintext. Therefore, using a secret eliminates the possibility of exploiting the weakness due to the determinism of Convergent Encryption. In the case of DupLESS, if an attacker learns the secret stored at the key server, confidentiality can no longer be guaranteed. Also, if the attacker has access to the storage (e.g. the Cloud Storage Provider), all stored files are subject to the same attacks to which Convergent Encryption is vulnerable.

1.4.3 Client-side Protocol

An interesting approach to cope with the conflict between deduplication and encryption consists of enabling users to autonomously detect whether a data segment is popular by running a privacy-preserving protocol with other users (peer-to-peer) or the Cloud Storage Provider. Based on the result obtained after the execution of the protocol, the user can assess whether a data segment is a duplicate and thus complete the upload procedure accordingly.

To the best of our knowledge, one of the most recent and relevant works that makes use of this kind of approach is PAKE [6], in which the authors claim to propose the first scheme that achieves secure deduplication without the aid of any additional independent servers. In [6], authors introduce a novel oblivious key-sharing protocol called PAKE aimed at allowing users to collaboratively perform client-side cross-user deduplication without relying on the support of any trusted entity. More precisely, before uploading a file the user runs this protocol with other users in order to autonomously discover whether a file is a duplicate and, if this is the case, securely obtain the encryption key with which it has been previously encrypted, so that the encrypted file can be deduplicated.

Interestingly, in order to reduce the number of users involved in the protocol, the authors introduce an optimization which implies a relaxed security definition. Indeed, each file is also associated to a short hash, namely 20 bits. Thanks to such a short hash, the user can select the subset of users with whom to run the

protocol. This set includes all users for whom there is a non-negligible chance that one of them uploaded the same file in the past. Disclosing such a short hash to the Cloud Storage Provider does not weaken confidentiality since the hash is short enough to provide no useful information to the Cloud Storage Provider. In practical terms, the collision rate is so high to make the complexity of such an attack comparable to guessing.

Although the authors implemented and evaluated a proof-of-concept prototype which proved to be both effective and efficient, the scheme presents am inherent limitation which makes its adoption in real scenarios challenging. Indeed, as any other peer-to-peer scheme, PAKE requires a sufficient number of users to be simultaneously online in order to run the protocol, otherwise a potentially duplicate may not be detected, resulting in a decrease of the deduplication ratio. In order to prove that this does not severely impact deduplication ratios, the authors assume that users stay online long enough and their status (offline/online) is uniformly distributed over the time. However, it is well-known that in real scenarios users have more irregular and hardly predictable behaviors, therefore it is unrealistic to make such an assumption on the distribution of users' statuses and predict that a given number of users will be online at a given time.

Another relevant work in this regard is iMLE [27], which proposes an elegant scheme for secure data deduplication based on an interactive protocol between the user an the cloud storage provider, in which the former is able to detect whether a file is a duplicate and the latter does not learn anything about the content of the file. However, as stated by the authors, the scheme is purely theoretical and its performance is far from being practical, hence this scheme cannot be adopted yet in real scenarios. The main reason behind this limitation is the extensive use of fully homomorphic encryption [28] in all phases of the interactive protocol, which is well-known to be still unpractical due to the prohibitive overhead in terms of computation and increase of data size.

1.4.4 Data Popularity as a differentiator for Encryption

Some schemes rely on the realistic assumption that popular data, which belong to many users and thus can have an important impact on storage space savings when being deduplicated, are unlikely to contain any sensitive information. On the other hand, unpopular data, which are unique or belong to a low number of users hence

do not necessarily have to be deduplicated, may contain sensitive information and must be protected with a stronger encryption. Based on this distinction, a scheme can use different encryption mechanisms (each with a different tradeoff between deduplicability and confidentiality) depending on the whether a data segment is considered to be popular.

As an example of this approach, authors in [7] proposed a scheme in which data belonging to multiple users are be encrypted with CE in order to enable the CSP to perform deduplication. On the other hand, unpopular data is more likely to contain sensitive information, hence it is safer to encrypt with a semantically-secure encryption. In order to implement this idea, this scheme makes use of a mixed cryptosystem combining convergent encryption and a threshold encryption scheme. However, this work suffers from a few drawbacks. First, the system suffers from a significant storage and bandwidth overhead. Indeed, for each unpopular file the user uploads two encrypted copies, one encrypted with a random symmetric key and one encrypted with the mixed encryption scheme. In scenarios with a high percentage of unpopular files, the storage overhead will be significant and nullify the savings achieved thanks to deduplication. Second, the system proposed in [7] relies on a trusted component which provides an indexing service for all data, both popular and unpopular. Third, both the client and the CSP have to perform complex cryptographic operations based on threshold cryptography on potentially very large data.

1.4.5 File-level vs Block-level

Finally, a crucial criteria which plays an important role in determining the practicality of a scheme is the ability of efficiently handling block-level deduplication, meaning that deduplication is performed with a finer granularity, that is at the level of blocks instead of whole files. Although this may seem straightforward, in practice this is not the case for all the schemes presented so far.

Indeed, in DupLESS, since the generation of the encryption key requires to run an interactive protocol between the user and the key server, extending the scheme to block-level deduplication would introduce a severe performance issue due to the need to run the protocol once per block, meaning that uploading a file may potentially require to run the protocol thousands of times. Similarly, in PAKE, since the user needs to run an instance of protocol for each file and each user,

extending this scheme to block-level deduplication would be unpractical. Also, in [7], because of the dependency on the indexing service, the effectiveness of the system is limited to file-level deduplication, which is known to achieve lower space savings than block-level deduplication.

1.5 Contributions

Starting from the aforementioned challenges, we aimed at achieving a comprehensive understanding of the scenario and then propose suitable approaches for the targeted problem. In order to do so, we first analyzed the results of various widespread deduplication techniques and most importantly their implications in terms of data confidentiality when combined with solutions that rely on deterministic encryption techniques.

Based on the indications provided by this study, we then designed and prototyped two secure yet efficient solutions in order to provide users and providers with practical mechanisms that effectively solve the aforementioned problem and are easy to integrate in real production environments. Also, for each prototype we conducted a deep performance analysis in order to compare them with traditional Cloud Storage solutions and assess their practicality in real environments. In particular, the first solution, which goes by the name of ClouDedup, led to the development of a fully-featured Cloud Storage system which is close to be proposed in the market.

1.5.1 Study on Deduplication

This study was conducted with two main objectives in mind. First, due to the lack of such studies in the literature, we wanted to have a tangible proof of the effectiveness of deduplication in real environments. Therefore, we conducted a comprehensive study on various real datasets which was aimed at highlighting the best deduplication technique(s) and providing a close approximation of the savings that can be potentially achieved. Second, we wanted to study the consequences in terms of confidentiality when using deterministic encryption in conjunction with block-level deduplication. In particular, we were interested in verifying whether

a (very) small block size could reduce the global entropy and thus increase the vulnerability to attacks.

The reason why we focused our efforts on deterministic encryption is twofold. First, the vast majority of the solutions that have been proposed in the literature as well as in the industry are based on this kind of encryption, therefore the study would have a bigger impact. Second, as mentioned earlier, determinism is a requirement raised by deduplication in order to make identical plaintexts uploaded by different users comparable after being encrypted.

However, it is well-known that deterministic encryption suffers from an inherent limitation which prevents it from achieving semantic security. In other words, a secure deduplication solution relying on deterministic encryption cannot provide the highest level of confidentiality at which we aim. Nevertheless, due to the necessity of being able to compare encrypted data segments and detect duplicates, randomness cannot be introduced in the encryption process, hence deterministic encryption remains a hard requirement and the conflict between deduplication and confidentiality arises.

Based on such a challenging scenario, we decided to study in depth the dependencies between deterministic encryption and block-level deduplication and assess whether and in which scenario a confidentiality threat may arise. In other words, we performed block-level deduplication on a real dataset using various realistic block sizes and for each of them we measured the difficulty of running a statistical attack.

As a result of our analysis, the study provided a few valuable indications with respect to block-level deduplication savings and security. More precisely, block-level deduplication proved to be much more effective than file-level deduplication, even taking into account the additional overhead. Also, based on our results, we argue that statistical frequency-based attacks remain unlikely even when using low yet realistic block sizes.

1.5.2 ClouDedup

As a first attempt to overcome the issues related to secure deduplication and convergent encryption, we proposed ClouDedup [8], a novel cloud storage architecture

in which the core confidentiality and deduplication features are outsourced to external components and the role of the Cloud Storage provider is limited to the storage of raw encrypted data segments.

ClouDedup is based on two main unique components:

1. a trusted encryption component (Gateway), deployed at a trusted third party or, if the customer is an organization, on the customer premises and within the security perimeter of the customer;
2. an additional component named Metadata Manager which is responsible for key management and deduplication.

Data protection is guaranteed thanks to the encryption layer introduced by the Gateway, which performs a symmetric and deterministic encryption on top of convergent encryption, which is performed by clients before the upload. Also, this encryption is totally transparent to the end user. Indeed, in practice the Gateway has been implemented as a HTTPS proxy located as a "man-in-the-middle" between end users and the Metadata Manager.

In addition to our unique components, we also rely on a Cloud Storage Provider. In ClouDedup we turn the Cloud Storage provider into an extremely simple component taking care of the raw storage of encrypted data segments, whereas deduplication and key-management are managed by the Metadata Manager and encryption is performed by the Gateway. In order to emphasize the independence between the Metadata Manager and the Cloud Storage Provider, the communication between these components has been implemented in a loosely coupled fashion by leveraging modern REST HTTPS-based APIs.

The need for a dedicated component responsible for key-management (the Metadata Manager) is raised by the necessity of efficiently and securely storing the encryption keys produced by the combination of block-level deduplication and convergent encryption. In our implementation, the Metadata Manager makes use of cutting-edge technologies in the field of databases together with and a very efficient implementation in order to effectively address this issue. In particular we use an open-source NoSQL database known as REDIS [9] which, according to our performance analysis, proved to be extremely efficient and reliable.

However, despite its ability to bring the best of both worlds, that is full confidentiality and storage space savings, ClouDedup requires to maintain and monitor a complex architecture in which confidentiality can be guaranteed as long as the secret key stored at the Gateway remains safe and is not disclosed to any malicious entity. Therefore, we aimed at finding a more advanced solution having the advantage of requiring a simpler architecture and directly involving the users in the deduplication operation.

1.5.3 PerfectDedup

In order to achieve full confidentiality without introducing a complex architecture which would discourage user adoption, we investigated the possibility of minimizing the use of additional components and using different levels of protection based on the popularity of the data segment. Indeed, we leverage popularity, which is measured as the number of owners of a given data segment, in order to determine the level of confidentiality: the higher the number of users owning a data segment, the higher its popularity. Based on that, a global popularity threshold is set in the system so that whenever the number of owners of a given data segment exceeds the threshold, the data segment becomes popular. We also assume that if popular, a data segment is likely to be non-confidential, as opposed to rare and unique data segments which may contain sensitive information and thus be confidential. Once the status of a data segment is assessed, different encryption techniques are employed, namely convergent encryption is used for popular data segments (in order to allow for deduplication) and semantically-secure encryption otherwise (in order to achieve the highest level of confidentiality).

Detecting popular data segments in a privacy-preserving manner requires users to run a secure lookup protocol against an untrusted Cloud Storage Provider. In PerfectDedup [10], we introduced a novel secure lookup protocol based on a secure version of Perfect Hashing [11], thanks to which the user can autonomously and securely detect whether or not a data segment can be deduplicated by simply querying the Cloud Storage Provider. Such an approach, which goes by the name of PerfectDedup, allows the user to securely detect whether a data segment can be deduplicated without revealing any exploitable information to the untrusted Cloud storage provider. Therefore, in PerfectDedup we achieve secure client-side

(our source-based) block-level deduplication, meaning that users can also save bandwidth.

In addition to bandwidth savings, another main advantage of PerfectDedup with respect to ClouDedup is the simpler architecture. Indeed, as opposed to ClouDedup, PerfectDedup does not require any trusted or independent servers with respect to the encryption and deduplication operations, since deduplication of blocks is entirely managed at client-side thanks to a novel, secure and lightweight lookup protocol based on a secure version of Perfect Hashing. However, similarly to ClouDedup, both solutions are built on top of convergent encryption and aim at strengthening it by eliminating the possibility of a dictionary attack by an untrusted Cloud storage provider.

To summarize, we designed, developed and analyzed the performance and the security of the two following solutions: ClouDedup and PerfectDedup. Both solutions share a very unique and valuable feature, that is the combination of full confidentiality and block-level deduplication, meaning that users and cloud storage providers can enjoy the advantages of deduplication without negatively affecting confidentiality. In the following chapters, after introducing a few concepts that are extensively mentioned in this thesis and comparing our work with existing approaches, for each solution we describe in detail the idea, the architecture, the protocol and finally we fully analyze its performance and security.

Chapter 2

State of the Art

In this chapter, we first introduce a few concepts that are mentioned throughout this manuscript and are the basis of the work conducted during this thesis. Later, we analyze in detail the most relevant approaches in the literature and thus compare our work with them.

2.1 Cloud Computing and Cloud Storage

As mentioned in the previous chapter, Cloud Computing providers leverage virtualization technologies in order to offer services and resources at different level of abstraction. What made Cloud Computing emerge was the innovative economic model proposed by Amazon. Cloud Computing comes with a "pay-as-you-go" model, meaning that customers only pay for the resources they actually use. In other words, as soon as a resource is dismissed or deactivated, it is not billed anymore. Such a feature gives customers the highest level of flexibility and scalability.

According to a recent market study performed in France [29], the Cloud Computing market has already exceeded the symbolic threshold of 1 billion Euros and its adoption is expected to keep growing exponentially in the next few years. Similar trends have been observed in other countries, especially in the United States where the adoption rate is currently close to 93% [30]. As shown in Figure 2.1, the study also takes into account Private Clouds, that is the set of Cloud-based infrastructures hosted on non-public platforms. Private Clouds are meant for providing the same benefits of Public Clouds (scalability, flexibility, lower costs, etc.) along

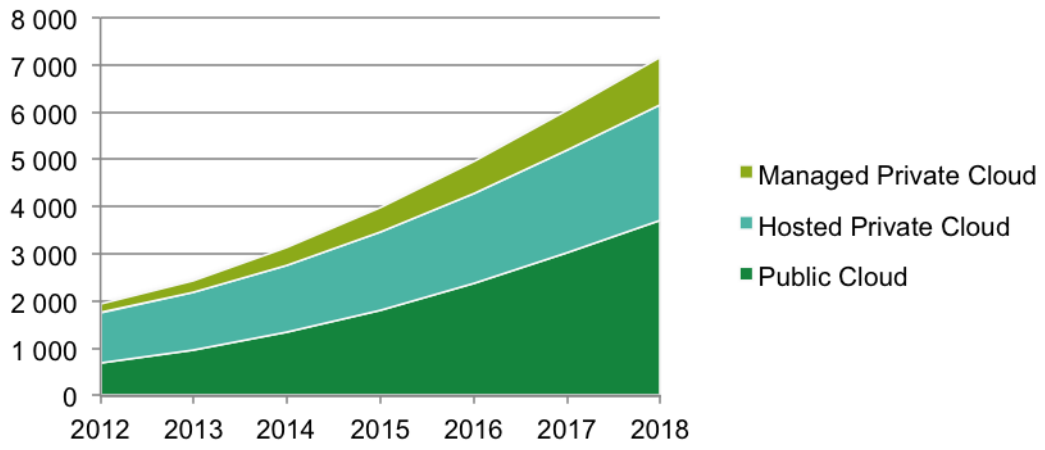

FIGURE 2.1: Cloud Computing growth in France according to the study conducted by cloudindex.fr.

with a better isolation and a finer control on the whole platform. For instance, this is often the case of those enterprises that decide to deploy their own Cloud platform on their premises in order to have more control and higher security assurance. Indeed, all studies [31] confirm that security and confidentiality concerns are the main reasons that refrain companies from outsourcing their infrastructures to Cloud Computing platforms.

In this thesis, we focus on Cloud storage, which belongs to the services usually offered by IaaS providers and arguably is the cloud service with the highest adoption rate. Cloud storage gives customers the attractive opportunity of using a remote and reliable storage for storing their data. In addition to that, data stored in the Cloud are accessible at anytime, on any device and from any location. This is made possible thanks to a set of different applications running on most of the existing devices (smartphones, tablets, laptops, web, etc.). Moreover, on top of these features, providers usually offer additional functionalities in order to facilitate file sharing and real-time collaboration. Among others, some of the major actors in this field are Dropbox [32], Box [33], Google Drive [34], OneDrive [35] and Amazon Cloud Drive [36]. Cloud storage is also a popular solution among developers. In fact, a growing number of applications (e.g. websites and mobile applications) are making use of Cloud storage containers in order to persistently store files (e.g. images) and databases. Most importantly, thanks to this strategy, developers can delegate the delicate task of providing a reliable storage and enjoy crucial features such as automatic backup, disaster recovery and geographical replication. In this

field, some of the major providers are Amazon S3 [20], Google Cloud Storage [37], Rackspace [38], Microsoft Azure [39] and OpenStack SWIFT [40].

Despite the high and increasing adoption rate, standard Cloud storage does not satisfy the needs of all customers. As an example, many enterprises explicitly require to be the only entity able to access and read their data, which may contain confidential and sensitive information. That is the main motivation behind the high number of Cloud storage providers that have recently proposed solutions that specifically address security, in particular confidentiality and privacy. These solutions achieve security thanks to encryption and other protocols aiming at protecting customers' data during all phases of the system life. As this field is constantly growing, it is not easy to point out an exhaustive list of the major providers. However, at the time of writing, some of the most noteworthy are Tresorit [41], Spideroak [42] and McAfee Personal Locker [43].

2.2 Asymmetric vs Symmetric Cryptography

Since the first contributions of Shannon [44] in 1949 and Hellman and Diffie [45] in 1979 (which introduced for the first time the concept of public-key cryptography), cryptography has constantly been an hot topic which has caught the attention of both research and industry. Nowadays, the systems that provide confidentiality and other security-related features are countless, and they are all based on cryptographic techniques that can be connected to two main categories: asymmetric and symmetric cryptography.

Generally, the main difference between the two categories lies on the fact that asymmetric encryption uses a pair of mathematically related keys, each of which decrypts the encryption performed using the other, whereas symmetric encryption requires the same key to be used for both encryption and decryption.

Asymmetric encryption is usually based on the hardness of some mathematical problem which cannot be efficiently solved in polynomial time. This is the reason why asymmetric cryptography has traditionally been labeled as more costly (in terms of computational load) than symmetric cryptography. Due to this difference, asymmetric cryptography found its success in applications based on digital signature, integrity verification and emails, whereby the "sender" does not need to share a secret with the "recipient" in order to send him an encrypted message

that only the recipient can decrypt. As an example, RSA [46] is an asymmetric encryption algorithm which is among the most widespread nowadays.

In contrast, symmetric cryptography can be considered the de-facto standard for guaranteeing data confidentiality, due to its better performance even when dealing with large amounts of data. In symmetric encryption, the same cryptographic key is used with the underlying algorithm (cipher) to encrypt and decrypt data, hence it must be kept secret and securely shared between the sender and the recipient. As opposed to asymmetric cryptography, in symmetric cryptography data are encrypted and decrypted by applying some sequence of complex operations which, chained together, have the property of making the final output impossible to invert, meaning that the original input cannot be obtained. As an example, AES (Advanced Encryption Standard) [47] a block-cipher which is the standard encryption method in data encryption, belongs to this family.

In the solutions we describe in this manuscript, when encrypting users' data we always refer to symmetric encryption, due to the necessity of being able to encrypt and decrypt large files as well as minimizing the performance impact for users.

2.3 Probabilistic vs Deterministic Encryption

In addition to the distinction between symmetric and asymmetric, an encryption algorithm can also be classified as probabilistic or deterministic. An encryption algorithm is probabilistic when it uses randomness while producing the ciphertext. More precisely, given a plaintext message M and an encryption key K, multiple executions of the encryption algorithm will yield different ciphertexts which, given the decryption key, can all be decrypted to the same plaintext M. On the other hand, deterministic encryption by definition does not involve any randomness, meaning that given a plaintext message M and an encryption key K, multiple executions of the encryption algorithm will always yield the same ciphertext.

This distinction implies that regardless of being symmetric or asymmetric, the degree of secrecy provided by an encryption algorithm may be affected by whether it is probabilistic or not. Indeed, a ciphertext encrypted with a pure deterministic encryption algorithm may leak some information to an attacker (e.g. an eavesdropper) who may be able to recognize known ciphertexts and perform a ciphertext-only attacks such as statistical or brute-force. The former, as shown in

the explicative example of Fig. 2.2, consists of finding patterns in the frequency of ciphertexts that can be linked to the frequency of known plaintexts. The latter would consist of encrypting random or chosen plaintexts and checking whether the resulting ciphertext matches the target ciphertext.

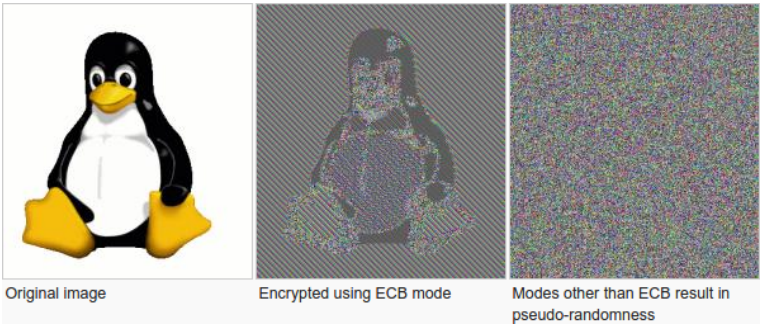

FIGURE 2.2: Example of weak deterministic encryption

The property that determines whether an encryption algorithm is vulnerable to this kind of leakage is known as semantic security. In short, semantic security requires that given the encryption of any message M and its size, an attacker cannot determine any partial information on the original message.

More formally, such a property can be defined as follows:

- The adversary (attacker) chooses two messages: M_0 and M_1.
- We encrypt one of these messages: $c = E(K, M_b)$. Given the ciphertext c, the adversary has to guess which message was ciphered.
- We define the following event: $M_b = \{$ adversary A decides that M_b is ciphered$\}$.
- The encryption function E is semantically secure if for all computationally bounded adversaries A the advantage $|Pr(M_0)Pr(M_1)|$ is negligible.

Depending on the family to which the encryption algorithm belongs, this requirement may have different implications. In asymmetric cryptography, semantic security means that it must be unfeasible for a computationally bounded (with respect to the hardness of the underlying mathematical problem) adversary to derive significant information about the plaintext when given only its ciphertext and the corresponding public encryption key. On the other hand, in symmetric cryptography semantic security implies that an adversary must not be able to compute

any function on the plaintext from its ciphertext. Such a requirement can also be seen as an adversary that given two plaintexts of equal length and their two respective ciphertexts, cannot determine which ciphertext belongs to which plaintext. In other words, the knowledge of a ciphertext and its size do not reveal any additional information on the plaintext. This concept is equivalent to the concept of perfect secrecy introduced by Shannon [44].

As an example of probabilistic encryption algorithm, it is worth mentioning the well-known ElGamal [48] asymmetric cryptographic scheme, which introduces randomness in the encryption process by encrypting the plaintext with the public key to the power of a random number. More precisely, given a message M and a public key defined by the tuple (G, q, g, h), the ciphertext (C_1, C_2) is calculated as $g^y, M\prime \cdot h^y$ and is equivalent to $(g^y, M\prime \cdot (g^x)^y)$, where x is the private key of the recipient (therefore it is kept secret), $M\prime$ is the element corresponding to M in the group G and y is a random value chosen by the sender at encryption time. When it comes to decryption, the recipient is given a tuple (C_1, C_2), hence he first calculates the shared secret $s = {C_1}^x$ and then $M\prime = C_2 \cdot s^{-1}$ which can be converted back to the original message M. In such a scheme, the use of the random y improves security since allows to generate different ciphertexts for the same plaintext. Most important, y is not needed by the recipient for decrypting a ciphertext.

On the other hand, a classic example of a pure deterministic encryption algorithm is AES in ECB [49] mode. Indeed, when AES, which is a block cipher, is used in ECB mode, each block of the plaintext is encrypted independently of the other blocks. Despite the block cipher algorithm being recognized as secure, such a configuration raises the critical issue of allowing for statistical frequency-based attacks due to the limited size of each block, namely 16 bytes in AES.

Fortunately, in symmetric encryption algorithms that are based on block ciphers, a similar property can be achieved by employing a widespread technique known as block chaining. More precisely, some sort of pseudo-randomness can be introduced by re-using the output of the encryption of block $i-1$ as input for the encryption of the block i. As an example of this principle, AES when used in Cipher Block Chaining (CBC) [49] mode uses block chaining in order to make the encryption more secure, as shown in Fig. 2.3. In practice, each ciphertext C_i is calculated as the result of the following operation $E_K(P_i \oplus C_{i-1})$, except C_0 which is the result of $E_K(P_0 \oplus IV)$, where IV is a random initialization vector. In other words, each block of the plaintext is first XORed with the encryption of the previous block

and then encrypted using the encryption key. This way, the same block will yield different ciphertexts depending on its location, resulting in pseudo-randomness being achieved.

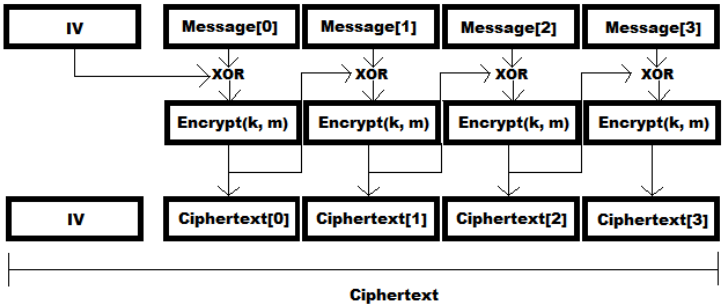

FIGURE 2.3: Diagram showing how AES works in CBC mode

2.4 Data Compression

In both network and storage applications, performance is often improved by employing data compression techniques. Such techniques are based on the objective of encoding a given data segment with a lower number of bits than the original representation. As a straightforward example, by doing that an application can speed up the transfer of a file over the network. Depending on the strategy adopted, compression can be lossy or lossless.

A compression technique is defined as lossy when some information, that are considered unimportant, are actually removed from the original data segment in a way that the final quality of the resulting compressed data is not affected. As an example, this kind of compression techniques is very common in multimedia scenarios such as audio and video encoding formats (e.g. MP3 [50], DVD [51]) where the compression algorithm is based on the assumption that human beings' senses are not developed enough to perceive all details.

On the other hand, a compression technique is defined as lossless when the process is fully reversible, meaning that no information is lost during the compression phase. Therefore, upon a decompression, that is the inverse operation of compression, the original data segment can be fully recovered. This family of compression techniques exploit the high presence of redundancy in real data in order to represent information more concisely by eliminating redundancy. These techniques

have become very common in a variety of daily life applications. As an example, the widespread ZIP method is based on the compression algorithms family known as LZ (Lempel-Ziv) [52]. Also, similar methods are used in other popular data formats such as PNG [53] and GIF [54].

2.4.1 Data Deduplication

Data deduplication can be seen as a special kind of lossless compression technique applied with a higher level of granularity. Instead of individually compressing the data segment itself by trying to represent it with a lower number of bits, in a cloud storage environment redundancy may likely be found globally, meaning across all users' files. Therefore, a significant amount of storage space may be saved by simply storing only unique data segments and, upon an upload or during a postponed batch process, checking whether a given data segment has already been stored in the past.

The global redundancy assumption can be easily verified in the reality. Indeed, in a scenario where many, potentially thousands of users store their files in the Cloud, chances are good that the same file, or a very similar one, will be uploaded by two or more users. In an era where companies and, more in general, users want to enjoy all the benefits of Cloud Storage along with low costs, great performance and usability, deduplication has become a de-facto standard for the majority of the Cloud Storage providers. We will now describe all possible variants of deduplication.

2.4.1.1 Source-based vs target-based Deduplication

The first differentiator is represented by where deduplication takes place. Indeed, deduplication can be performed either locally, meaning on the device of the user, or remotely, meaning on the Cloud Storage servers. For instance, the latter is often the case for personal Cloud Storage. Deduplicating data locally brings the useful advantage of being able to save not only storage space but also the consumption of network bandwidth. However, as explained in [55], such a system would easily allow a malicious user to play with the storage system in order to learn whether a file was already stored, which may represent a serious confidentiality threat. Indeed, in some contexts, a malicious user may exploit such a weakness in order

to obtain very sensitive and confidential information. Such an attack would not be easy to detect since it does not incur any violation of the underlying protocol. Data can also be deduplicated remotely, meaning that users will upload their data as usual while the Cloud Storage provider will take care of performing the actual deduplication operation, which consists of looking for an identical copy of the data segment and decide whether it needs to be physically stored or not.

2.4.1.2 Inline vs Post-processing Deduplication

Depending on the moment at which deduplication is performed, it can be either inline or post-processing. When the inline approach is adopted, data is deduplicated immediately, meaning that the storage server checks whether a data segment is already stored as soon as it is uploaded by the user. This approach is quite common in most of the file storage systems, whereby saving bandwidth in addition to storage space is crucial from a performance standpoint. On the other hand, when post-processing deduplication is chosen, data deduplication takes place after the upload. Usually, post-processing deduplication takes place at regular intervals (e.g. once a day). Therefore, this approach lends itself very well to backup applications, whereby snapshots are submitted at regular intervals and can be processed in batch mode after being submitted.

2.4.1.3 Single-user vs Cross-user Deduplication

Data deduplication solutions can also be classified according to their scope, meaning the set of users whose files can be deduplicated. In real scenarios, cross-user deduplication, that is deduplication across multiple users, certainly is the most popular configuration. Indeed, enabling deduplication across all users of the system greatly increases the chances of finding redundant data, hence achieve significant storage space savings. However, making cross-user deduplication secure raises the issue of putting in place a mechanism to enable users to use the same encryption key for a given data segment without compromising confidentiality, that is the problem we tried to solve in ClouDedup and PerfectDedup. Moreover, cross-user deduplication may occur across users belonging to the same organization (e.g. employees of a company) or a much larger set of unrelated users such as all clients of a public cloud storage service. On the other hand, when deduplication is performed only on the data of a single user (single-user deduplication), encryption

can be safely added without negatively affecting deduplication effectiveness, as long as the same key is used to encrypt all data segments. Nowadays, single-user deduplication is far from being widespread because of its poor effectiveness, which depends on the fact that the degree of redundancy within files of a single user is usually low.

2.4.1.4 File-level vs Block-level Deduplication

Finally, based on the data segment granularity, deduplication can be either file-level or block-level. As the same suggests, block-level deduplication is more fine-grained with respect to file-level deduplication. Indeed, in block-level deduplication a file is first split into blocks and then each block is deduplicated individually. The size of a block can be either fixed or variable, depending on the chunking method that is adopted (see section 2.5 for further details). According to recent studies [12], block-level deduplication achieves higher storage space savings, therefore it is recommended in order to maximize the benefits of deduplication. However, the drawback of this technique resides in the fact that handling multiple blocks per file (potentially thousands) may introduce a significant overhead in terms of metadata storage. Indeed, the deduplication server has to keep track of the links between logical blocks and the corresponding physical copies as well as the structure of each file, so that when a file is requested by its owner, the file can be correctly reconstructed by merging its blocks. Also, upon checking whether a block was already stored, the deduplication server should compare a block (or its ID) with all blocks previously stored, which may lead to a non-negligible computational overhead. For these reasons, a system meant for block-level deduplication should be carefully designed with scalability as a primary design goal.

2.4.1.5 State-of-the-art in the Industry

As stated above and shown in Table 2.1, deduplication has become a de-facto standard in the industry: most of the major vendors of large-scale storage systems employ deduplication in order to improve performance and save significant amounts of storage space. From our study, as a confirmation of our statements, we found out that the vast majority of the commercial systems currently available make use of inline source-based block-level deduplication, which clearly is the

configuration that brings the best results in terms of storage space savings. Unfortunately, as introduced above, such a configuration may present a few security issues that need to be addressed in order to guarantee data confidentiality.

Enterprise	Product Name	In-line / Post-process	Target-based / Source-based	Block level / File Level	Variable length / Fixed Length	Cross-site	Deep / ID Comparison	Enterprise Backup	VM Backup	Personal use	Mean space saving	Platform
Nine Technology	-	In-line	Source-based	Block level	-	-	-	Yes	No	Yes	-	-
Opendedup	-	In-line	Both	Block level	Fixed	No	-	-	Yes	Yes	90-95%	Windows and Linux
DELL	DR4000	In-line	Both	Both	Variable	Yes	-	Yes	Yes	-	-	Intel-based servers
Napatech	-	In-line	Source-based	Block-level	-	Yes	-	Yes	-	No	-	-
ASIGRA	Asigra Cloud Backup	In-line	Source-based	Block-level	-	Yes	-	Yes	-	Yes	-	-
AVAMAR	-	In-line	Source-based	Block-level	Variable	No	-	Yes	Yes	No	-	-
NETAPP	-	-	-	-	-	-	-	-	-	-	-	-
Oracle	ZFS Deduplication	Both	Both	Both	-	No	Just Hash	-	-	Yes	?	ZFS volumes
CommVault	Simpana Software	In-line	Source-based	-	-	Yes	-	Yes	-	No	90,00%	-
HP	HP StoreOnce	In-line	Source-based	Block-level	-	Yes	-	Yes	Yes	-	-	HP Cloud
Microsoft	Windows Server 2012	Post-process	Source-based	Block-level	Variable	No	-	-	-	Yes	-	Windows Server
QUANTUM	DXi disk-based deduplication	In-line	Source-based	Block-level	Variable	-	-	Yes	-	-	-	-
StarWind	iSCSI SAN storage	In-line	Source-based	Block-level	-	No	-	Yes	Yes	-	-	Windows
VMWare	vSphere	Post-process	Target-based / Source-based	Block-level	-	Yes	-	Yes	Yes	-	-	VMWare
EMC	EMC VNX	In-line	Source-based	Block-level	Variable	-	-	Yes	Yes	No	-	EMC Products
IBM	IBM System Storage N series Deduplication	In-line	Source-based	Block-level	Variable	Yes	Deep	Yes	Yes	No	-	IBM Backup
EXAGRID	-	Post-process	Target-based	-	-	-	-	Yes	Yes	-	-	Several backup softwares
SYNCSORT	Data Protection for Reduction and Deduplication	In-line	Source-based	Block-level	Variable	-	-	Yes	-	No	-	-
SYMANTEC	NetBackup and Backup Exec	In-line	Both	Block-level	Variable	Yes	-	Yes	Yes	-	-	Symantec Backup Products

TABLE 2.1: List of existing commercial solutions for data deduplication

2.5 Data-chunking Techniques

As stated above, there has been concrete evidence that block-level deduplication outperforms file-level deduplication, meaning that it achieves higher storage space savings. However, when dealing with block-level deduplication, it is crucial to decide at which points a file can be split into chunks, since a good or bad choice of a block boundary may have a high impact on the ability of detecting duplicate blocks, hence on the final deduplication ratio. For instance, splitting a file into fixed-size blocks has proved to achieve low savings due to a phenomenon known as "boundary shifting": if a file F is modified by inserting even a single byte in the middle of a sequence of blocks, all block boundaries will be shifted, resulting in a new sequence of different blocks. For this reason, the algorithm used for the detection of block boundaries plays an important role, hence it needs to be properly chosen and then tuned.

Content-based data-chunking algorithms Nowadays, all major data-chunking algorithms share the following objective: the boundary shifting problem can be avoided by making chunk boundaries self-adapting and dependent on the content of the block, instead of its position. This can be achieved by employing a principle known as "sliding window", which requires to scan the entire file by moving a window by one byte at a time. At each iteration of the scan, the algorithm computes a fingerprint (or a checksum or a hash) of the current window and, if the fingerprint meets a given property, the current block boundary is determined and the algorithm starts looking for next block boundary.

More precisely, algorithms based on the sliding window principle work as follows: a pre-determined integer I is defined, a sliding window of a pre-defined width is moved across the file, and at every position in the file, the fingerprint of the bytes within this window is calculated. A position k is declared to be a chunk boundary if the fingerprint computed on that position matches with the integer I at position k. Such an algorithm assures that a chunk boundary is selected according to some property no matter where the chunk is located in the file.

For efficiency reasons, fingerprints are usually calculated with a particular family of hash functions which goes by the name of "Rolling Hashes": the main advantage of this family of hash functions is the ability of computing the hash value of a sequence of bytes (sliding window) in a very efficient way, which consists of reusing the previously calculated hashes. This is possible thanks to a special property: a

rolling hash function can quickly compute the hash value of the new window by taking as input the old hash value, the byte removed from the window and the byte that has just been added.

It is worth pointing out that the solutions we propose in this manuscript are completely independent from the underlying data-chunking algorithm, meaning that they are fully compatible with any algorithm. Therefore, this problem is out of the scope of our work.

2.5.1 Rabin Fingerprinting

To the best of our knowledge, a commonly used and very efficient rolling hash algorithm for generating variable-size blocks is Rabin Fingerprinting [56], that was introduced by Michael O. Rabin in 1981. This scheme is based on polynomials over a finite field. More precisely, given a message of n bits $m_0, ..., m_{n-1}$, it is represented as a polynomial of degree $n-1$ over the finite field $GF(2)$ [57].

$$f(x) = m_0 + m_1 x + ... + m_{n-1} x^{n-1} \tag{2.1}$$

A random irreducible polynomial $p(x)$ of degree k over $GF(2)$ is picked, and the fingerprint of the message m is defined as the remainder $r(x)$ obtained after the division of $f(x)$ by $p(x)$ over $GF(2)$, which is a message of k bits, with $k \leq n$.

In practice, depending on how the algorithm is configured, Rabin Fingerprinting makes use of a 48 byte sliding window which is moved over the file in order to compute the Rabin fingerprint of the current window. Given the fingerprint, the algorithm computes the modulo (the remainder $r(x)$) between the fingerprint ($f(x)$) and the expected average size of a block ($p(x)$, set to 8 KB by default), which is a constant value. If the result is equal to a pre-defined prime number (set to 3 by default), then a block boundary is determined and a new block begins. Also, a maximum block size is defined so that a block boundary is determined even when no match has been found. Since the output of Rabin fingerprints are pseudo-random, the probability of any given chunk of 48 bytes hitting a block boundary is very low, therefore this has the effect of producing variable size blocks which are resistant to the aforementioned boundary shifting phenomenon.

An accurate C implementation of the Rabin Fingerprint scheme is publicly available and released as open-source code [58].

2.6 Convergent Encryption

As mentioned above, Convergent Encryption (CE) is a deterministic and symmetric encryption technique that has been proposed long ago [4] as a simple but effective solution to cope with the conflict between deduplication and encryption. The novelty of CE resides in the way the encryption key is generated: instead of using a random key or a user's password-dependent key, the encryption key is deterministically derived from the data segment that is being encrypted (the plaintext). The key is usually obtained by calculating the unkeyed secure hash of the data, optionally concatenated to some additional information. This feature enables all users to easily generate the same encryption key for the same data segment, resulting in the same ciphertext. Therefore, an untrusted Cloud Storage provider will be able to compare two encrypted data segments and determine whether they are identical or not, without having access to the original content of the data segment or requiring any extra information. Moreover, users do not need to put in place any coordination in order to agree on the key to be used for the encryption.

More formally, CE can be described as follows: Alice wants to encrypt her message M, hence derives a key $K = H(M)$ and then encrypts the message as $C = E(K, M) = E(H(M), M)$, where H is an unkeyed secure hash function and E is a symmetric block cipher. C is the ciphertext that will be uploaded to the Cloud and Alice retains the decryption key K. Bellare et al. [5] have formalized this concept and named it as Message-Locked Encryption (MLE). They also introduced a number of variants of MLE, which differ from each other in the way the encryption key K is generated and how the decryption process takes place.

Because of its simplicity and effectiveness, CE has been widely adopted in the industry and exploited in several commercial products [26]. For instance, Bitcasa [59] not long ago launched an offer based on an enhanced version of CE, called zero-knowledge encryption, and unlimited storage space. However, after CE has been massively adopted in the industry, researchers have conducted in-depth studies [5]

aimed at demystifying CE and bringing to light the existing weaknesses of this approach.

The common versions of CE are vulnerable to attacks that can compromise data integrity. As an example, an adversary A may run a duplicate-faking attack by uploading random fake ciphertexts to the Cloud. Whenever a user U will upload a valid ciphertext C, generated from a message M, such that its hash $H(C)$ matches the hash of one of the random fake ciphertexts previously uploaded by A, the valid ciphertext C will be deduplicated and thus will not be stored. Upon retrieving the ciphertext C, the user U will decrypt it and obtain an error or a message different than M, resulting in data integrity being compromised. As a solution, the user U may pre-compute a tag T and upload it along with the ciphertext C as follows: $C||T$, where $||$ denotes the concatenation operator. This way, upon retrieving the ciphertext C, the user U will be able to verify whether the decrypted data segment corresponds to the retrieved tag. If not, the user U will be aware of the duplicate-faking attack attempt and raise an error. However, such a solution is not complete since it does not protect against erasure attacks, meaning that duplicate-faking attacks can be successfully detected but not prevented. Indeed, achieving protection against erasure attacks is far from being straightforward.

Most importantly, the authors in [5] give a formal proof of the fact that all CE scheme cannot achieve semantic security [60] for all messages. In short, if the message M belongs to a finite domain S of size s, then an adversary A given a ciphertext C can always recover M in at most s trials. This is possible thanks to the fact that both the encryption and the key are fully deterministic. Therefore, a CE scheme can only achieve semantic security for unpredictable messages (high entropy), meaning that the encryptions of two unpredictable messages must be indistinguishable. On the other hand, this implies that if an adversary A has sufficient knowledge to narrow down the set of possible values of M to a small enough subset of S (predictable messages), semantic-security cannot be guaranteed anymore. This crucial limitation is due to the close relation between the data and the encryption key, which makes CE vulnerable to a category of attacks known as "dictionary attacks". As an example, a malicious Cloud Storage provider (the adversary) without any knowledge on the original content of users' data may try to perpetrate a dictionary attack whose steps can be summarized as follows:

- given a target file F, guess its content either randomly or by using a template;

- if block-level deduplication is used, split F into blocks using the same method used by clients;
- encrypt data segments with CE in order to obtain the ciphertexts;
- compare each ciphertext with all encrypted data segments in storage (by using their hash as unique identifier or by performing a deep comparison);
- if there is a match for all data segments, then the attack has been perpetrated with success, hence the content of the file has been revealed; if not, repeat the previous steps by re-using the data segments for which a match has already been found.

Such a simple attack is also known as "confirmation-of-a-file" (COF) attack and can allow an adversary who has direct access to the storage, such as the Cloud Storage provider or a malicious insider, to easily check whether a file has been stored or not. Also, such an attack may be extremely effective in those scenarios where the adversary has some pre-knowledge on victim's data. In other words, the adversary may know most of the content of the file (e.g. the template) and try to guess the missing parts (e.g. the security code in a letter from a bank). This can be done by running a brute-force attack which, although it would be costly and time-consuming, will eventually reveal the content of the whole file. Such an attack is known as "Learn-the-remaining-information" (LRI). These two weaknesses are both strictly related to the nature of CE itself, which by definition links the encryption key to the data that is being encrypted. This proves that CE cannot be used as-is in order to protect users' data and guarantee full confidentiality.

A simple solution to overcome this severe weakness of CE would consist of introducing a secret, known only by a set of users, upon generating the encryption key for a given data segment. For instance, this may be achieved by using a secure keyed hash function with a key shared across all users willing to enable deduplication of their files. However, such a solution would present the severe drawback of greatly decreasing deduplication effectiveness, which would be possible only across the files of those users that share the secret.

In addition to the security issues introduced by the encryption layer, deduplication may introduce additional security issues that may further weaken data confidentiality, no matter what encryption technique is used. More precisely, the fact that

Solution	CE-based	Client-side	Popularity Distinction	Block-level	No Secret	TTP
[4]	✓	✓	X	X	✓	X
[26]	✓	✓	X	X	X	X
DupLESS	✓	✓	X	X	X	✓
iMLE	X	✓	X	X	✓	X
[7]	✓	✓	✓	X	✓	Semi-trusted
PAKE	X	✓	X	X	✓	X
ClouDedup	✓	X	X	✓	X	✓
PerfectDedup	✓	✓	✓	✓	✓	Semi-trusted

TABLE 2.2: Summary of the relevant related work

the Cloud Storage provider knows the structure of a file, its owners, the access patterns and the frequency of each data segment may give him the ability of inferring some information on the file and revealing, at least in part, its original content, even when data segments are securely encrypted. We studied this problem and show our results in next chapter.

2.7 Existing Approaches for Secure Data Deduplication

In this section, we review the most relevant research contributions targeting the problem of secure data deduplication. For each them, we summarize the main characteristics and point out the drawbacks which prevent it from being a suitable solution and motivated us to propose ClouDedup and PerfectDedup. Also, in Table 2.2 we summarize the main features of each solution in order to make comparison easier.

2.7.1 Convergent Encryption

As mentioned earlier in this manuscript, Convergent Encryption seemed to be a good candidate to achieve both confidentiality and deduplication thanks to its ease of implementation and high practicality. However, researchers showed that it suffers from critical weaknesses which do not ensure protection of predictable files against dictionary attacks [14]. In a scenario where sensitive data must be protected (e.g. payslips, emails and letters with passwords, etc.), such a weakness is too severe to be tolerated.

In order to overcome this issue in a simple way and without losing the advantages of CE, Warner and Pertula [14] have proposed a simple workaround which consists of adding a secret value S to the encryption key. Deduplication will thus be applied only to the files of those users that share the secret. The new definition of the encryption key is $K = H(S|M)$ where $|$ denotes an operation between S and M. However, this solution overcomes the weaknesses of convergent encryption at the cost of dramatically limiting deduplication effectiveness, which is known to become more effective when expanding its scope. Most importantly, confidentiality may be broken by just learning the secret, which would compromise the security of the system.

As opposed to this workaround, the work behind ClouDedup and PerfectDedup, and more generally behind the whole thesis, has been conducted with more advanced goals in mind. Our main goal was to come up with a secure yet practical solution capable of providing both full confidentiality and the highest deduplication ratios. Indeed, any solution that partially satisfies these two requirements or fully satisfies just one of them, would far from being suitable for real scenarios, therefore its impact and relevance would be rather limited.

In both ClouDedup and PerfectDedup, we propose two different approaches to achieve full confidentiality of sensitive data without decreasing the deduplication ratios that are normally achievable. Indeed, we keep enjoying the advantages of Convergent Encryption while protecting sensitive data by adding a deterministic encryption layer which is not vulnerable to dictionary attacks, hence cannot be broken by malicious entities.

2.7.2 DupLESS

A relevant contribution on secure deduplication is DupLESS [25] which, similarly to ClouDedup and PerfectDedup, makes use of deterministic encryption. More precisely, DupLESS uses a slightly modified version of Convergent Encryption in which the novelty is introduced in the key generation process. In DupLESS, the encryption key is still generated as the result of a deterministic function calculated on the data segment to be encrypted, however this is done with the aid of a trusted key server storing a secret value. The role of this key server is to run a privacy-preserving protocol with the user who wishes to upload a file. The privacy-preserving protocol allows the user to compute a pseudo-random function using as

input the hash of the file being uploaded and the secret value stored at the trusted key server. However, the computation is carried out in a way that both inputs are not disclosed to the other party. In practice, during the computation the user cannot learn the secret stored at the key server and the key server cannot learn the hash of the file being uploaded.

Thanks to such a protocol a user can generate a deterministic and data-dependent encryption key without disclosing any information on his file, so that all users will encrypt the same file with the same key, resulting in the same ciphertext that can be easily deduplicated by the untrusted Cloud Storage Provider. However, this scheme presents a few drawbacks. First, if an attacker learns the secret stored at the key server, confidentiality can no longer be guaranteed. Also, if in addition to discovering the secret the attacker has also access to the storage (e.g. the CSP), all stored files are subject to the same attacks to which CE is vulnerable, that is dictionary attacks. Second, DupLESS suffers from a severe scalability issue. Indeed, since the generation of the encryption key requires to run an interactive protocol, consisting of multiple round-trips between the user and the key server, extending the scheme to block-level deduplication would introduce a severe performance issue due to the high latency introduced by the need to run the protocol once per block, meaning that uploading a file may potentially require to run the protocol thousands of times.

As opposed to DupLESS, both ClouDedup and PerfectDedup aim at secure yet efficient cross-user block-level deduplication. Indeed, the solutions we introduced in this manuscript have been designed with the support to block-level deduplication as a design goal. Also, PerfectDedup does not rely on any trusted component with respect to the storage of any secret, resulting in a scheme much more robust against compromised components.

2.7.3 iMLE (Interactive message-locked encryption and secure deduplication)

Another relevant and recent work in this field is iMLE [27], which proposes an elegant scheme for secure data deduplication based on an interactive protocol running between the user an the untrusted cloud storage provider, in which the former is able to detect whether a file is a duplicate (it has already been stored

in the past) while the latter does not learn anything about the content of the file. The interaction occurring between the user and the cloud storage provider is a fundamental ingredient to achieve full confidentiality for all files, including the predictable and correlated ones. Differently from traditional approaches, the main novelty that has been introduced by this scheme is the fact that the user can determine whether a file can be deduplicated based on the result of an interactive protocol, meaning that the Cloud Storage Provider does not need to have access to the file in order to compare it with the other files currently in storage. This idea is similar to the one behind PerfectDedup, where the user determines the popularity of a data block through the execution of a secure lookup protocol based on Perfect Hashing.

Interestingly, an additional novelty introduced by this scheme is incremental updates. As opposed to any other scheme that has been proposed so far, iMLE allows to update a previously uploaded and encrypted file without re-encrypting and re-uploading the entire file from scratch, meaning that the cost (in terms of computational and network overhead) of an update is proportional to the distance between the previous version of the file and the current one.

However, as acknowledged by the authors themselves, this scheme suffers from a critical drawback in terms of computational and network overhead. In other words, this scheme is purely theoretical and its performance is far from being practical, hence this scheme cannot be adopted yet in real scenarios. The main reason behind this limitation is the extensive use of fully homomorphic encryption [28] in all phases of the interactive protocol, which is well-known to be still unpractical due to the prohibitive overhead in terms of computation and increase of data size.

As opposed to iMLE, both ClouDedup and PerfectDedup have been designed with efficiency and practicality in mind. Moreover, the performance analysis that have been conducted on their prototypes have clearly shown storage, network and computational overhead are affordable and not far from real production systems. This is also due to the use of symmetric cryptography and the absence of any encryption technique based on asymmetric cryptography.

2.7.4 Popularity-based Encryption

Recently, authors in [7] proposed a scheme which is based on the novel idea of differentiating data protection depending on its popularity degree. More precisely, the authors claim that in standard scenarios data belonging to several users is very unlikely to contain sensitive data, while unpopular data, which belong to a low number of users, might contain sensitive data. Starting from this simple yet realistic assumption, authors also observe that popular data are those data that we wish to deduplicate since they can bring higher storage space savings, while unpopular data, for which there is a non-negligible probability of also being sensitive, should rather be protected in order to keep them confidential. This being said, a straightforward yet effective solution consists of encrypting popular data with convergent encryption and unpopular data with a stronger encryption technique (e.g. semantically-secure encryption). Such a solution would bring the advantage of achieving almost optimal deduplication ratios (popular data can be easily deduplicated) without decreasing the level of confidentiality for unpopular and thus potentially sensitive data. The only price to pay is to lower the confidentiality of popular data. However, in most scenarios such a risk is acceptable since popular data hardly contain sensitive information, therefore even in the unfortunate case of an attack there would be no serious data leak.

Authors in [7] tried to implement the aforementioned idea by proposing a scheme that makes use of a mixed cryptosystem combining convergent encryption and a threshold encryption scheme [61]. More precisely, a trusted server takes care of keeping track of data popularity (based on updates sent by users) and notifying users as soon as a file becomes popular and thus needs to be re-encrypted with convergent encryption in order to enable cross-user deduplication. At the beginning of the system life, every file is considered unpopular and encrypted with two encryption layers: the first one is convergent encryption while the second one is an ad-hoc threshold cryptosystem based on asymmetric cryptography. As soon as a file reaches the popularity threshold (a pre-defined number of users uploaded the same file), the Cloud Storage Provider has enough information (keys) to remove the threshold cryptosystem layer and obtain the file encrypted with convergent encryption.

Unfortunately, this scheme suffers from a few drawbacks that have been addressed in PerfectDedup, which is also based on the popularity distinction idea. First,

the system suffers from a significant storage and bandwidth overhead. Indeed, for each unpopular file the user uploads two encrypted copies, one encrypted with a random symmetric key and one encrypted with the mixed encryption scheme. In scenarios with a high percentage of unpopular files, the storage overhead will be significant and nullify the savings achieved thanks to deduplication. Second, the system proposed in [7] relies on a trusted component which provides an indexing service for all data, both popular and unpopular. Third, because of the aforementioned reasons, the effectiveness of the system proposed in [7] is limited to file-level deduplication, which is known to achieve lower space savings than block-level deduplication. Fourth, both the client and the CSP have to perform complex cryptographic operations based on threshold cryptography on potentially very large data. Fifth, the scheme proposed in [7] requires an initialization phase in order to setup and distribute key shares among users.

As opposed to [7], although PerfectDedup also differentiates the encryption technique based on the popularity of the data segment, our scheme proved to achieve the same advantages in terms of data confidentiality with a much better performance. First, at any point of the system life only one version of each data segment is stored, resulting in a negligible (almost null) storage overhead. Second, PerfectDedup also makes use of a trusted component for indexing data segments, however the required amount of storage is much lower than [7]. Third, PerfectDedup has been designed to be efficient when implementing cross-user block-level deduplication which is known to achieve the highest deduplication ratios. Finally, the computational overhead for all components of the system is extremely low, even when performing the cryptographic operations, which are all based on symmetric cryptography.

2.7.5 PAKE

To the best of our knowledge, one of the most recent and relevant works in the field of secure data deduplication is PAKE [6], in which the authors claim to propose the first scheme that achieves secure deduplication without the aid of any additional independent servers. In [6], authors introduce a novel oblivious key-sharing protocol called "PAKE", which aims at allowing users to collaboratively perform client-side cross-user deduplication without relying on the support of any trusted entity.

Similarly to iMLE and PerfectDedup, in this scheme the Cloud Storage Provider is not responsible for determining whether or not a file can be deduplicated, hence it does not need to compare the new encrypted file with the ones already in storage. Rather, this operation is entirely handled by users and the Cloud Storage Provider is left with raw storage and coordination responsibilities. To give more details, before uploading a file the user collaboratively runs the protocol with other users in order to autonomously discover whether a file is a duplicate and, if this is the case, securely obtain the encryption key with which it has been previously encrypted by the first user who uploaded it. This way, the user can encrypt the file using the same key and thus get the same ciphertext (assuming that a deterministic encryption algorithm is used), which finally enables him to successfully perform client-side deduplication without exposing the original file to the potentially untrusted Cloud Storage Provider.

Although the application of the PAKE protocol in the field of secure data deduplication can be considered an important novelty, the algorithms behind the protocol itself cannot be considered as such. Indeed, PAKE stands for Password Authenticated Key Exchange and corresponds to a family of protocols [62] that is well-known in the literature. In short, the purpose of such protocols is to provide an interactive and secure method to allow two or more parties to establish a cryptographic key based on the knowledge of a password, or more generally a secret, without disclosing the secret to the other party in case of mismatch. The idea behind PAKE is thus to employ a revised version of such a protocol to let a user collaboratively retrieve (with the aid of the other users in the system and the Cloud Storage Provider) the encryption key that has been previously used to encrypt a popular file, using the hash of the file as password. Of course, this is the case whenever the file is actually a duplicate. If it is not, the protocol returns a random key. This means that if a file is a duplicate then it will be successfully deduplicated since it will be encrypted with the same encryption key that has just been retrieved through PAKE, otherwise it will be encrypted with a random key and stored normally.

It is worth pointing out that in order to reduce the number of users involved in the protocol, the authors introduce a relaxed security definition by associating a short hash to each file. Thanks to this short hash, the user can select the subset of users with whom to run the protocol, as there is a non-negligible chance that one them uploaded the same file in the past. Most importantly, such a leak does not weaken

confidentiality since the hash is short enough to provide no useful information to the Cloud Storage Provider due to huge number of collisions.

Although the authors implemented and evaluated a proof-of-concept prototype which proved to be both effective and efficient, the scheme presents a few limitations which makes its adoption in real scenarios challenging. First, the scheme requires a sufficient number of users to be simultaneously online in order to run the protocol, otherwise a potentially duplicate may not be detected, resulting in a decrease of the deduplication ratio. In order to prove that this does not severely impact deduplication ratio, the authors assume that users stay online long enough and their status (offline/online) is uniformly distributed over the time. However, it is well-known that in real scenarios users have more irregular and hardly predictable behaviors [63], therefore it is unrealistic to make such an assumption on the distribution of users' statuses and predict that a given number of users will be online at a given time. Second, once again, since the user needs to run an instance of protocol for each file and each user, extending this scheme to block-level deduplication would be unpractical.

Similarly to PAKE, PerfectDedup also leverages a secure client-side protocol in order to enable the users to determine whether or not a data segment can be deduplicated. Also, although in a very different way, PerfectDedup relies on collision rates in order to guarantee confidentiality. More precisely, PerfectDedup uses collision rates in order to assure that lookup queries issued by users do not reveal the content of users' data blocks. However, thanks to the use of a client-server architecture, PerfectDedup proved to be much more scalable and easy to integrate with real production environments. Indeed, PerfectDedup can easily support block-level deduplication.

Chapter 3

Study on Deduplication

3.1 Introduction

Conflict between Encryption and Deduplication The main reason behind the growing interest for secure data deduplication among security researchers is the challenging research question due to the difficulty of solving the inherent conflict between encryption and deduplication. Indeed, encryption and deduplication are two techniques that are based on different foundations and do not share the same goals. Encryption has the primary goal of making a ciphertext unlinkable to the plaintext that originated id, especially when the same plaintext is encrypted by different users. By contrast, deduplication requires a potentially untrusted entity such as the Cloud Storage Provider to be able to compare data segments and detect whether a given data segment is already in storage. When dealing with encrypted data segments, perhaps encrypted by different users at different points in time, this requirement implies that the Cloud Storage Provider must be able to detect whether they were originated from the same plaintext.

Deterministic Encryption as a main building block Any practical solution achieving secure deduplication of encrypted data requires deterministic encryption as a main building block. Such a strict requirement is raised by the necessity of generating identical ciphertexts for identical plaintexts, even when the encryption is performed by different users. If such a requirement is not met, deduplication becomes unfeasible due to the inability of the Cloud Storage Provider to detect that two different ciphertexts are the result of the encryption of the same plaintext.

It is worth pointing out that the use of deterministic encryption does not only include Convergent Encryption. Indeed, deterministic encryption includes all those solutions that are based on Convergent Encryption (which serves as a first encryption layer) and on top of which one or more deterministic encryption layers are added. This is the case for many existing solutions including ClouDedup, which is presented in next chapter.

However, we observed that even when data confidentiality is apparently guaranteed by a strong form of deterministic encryption, the main requirement for secure data deduplication, that is allowing the Cloud Storage provider to detect whether or not two encrypted data segments are identical, may introduce a new potential weakness which may lead to new kinds of attacks, such as statistical attacks, from the Cloud Storage provider.

More precisely, this new weakness arises from the necessity of always encrypting the same data segment using the same key material and the same input no matter where (what file and what position in that file) the data segment is located. Such a restriction is applied in order to always generate the same ciphertext and thus make data segments deduplicable.

In practice, this scenario can be seen as a generalization of the well-known phenomenon that occurs in symmetric block ciphers: to avoid statistical attacks, block ciphers usually apply some kind of chaining between blocks in order to introduce entropy even in those files with a potentially high level of redundancy. For instance, this is the case of AES in Cipher Block Chaining (CBC) mode where a random initialization vector is XORed with the first plaintext block and then every encrypted block is XORed with the next plaintext block. Unfortunately, applying a similar technique in the context of the deduplication would negatively and severely affect the deduplication ratio, therefore such an approach is not suitable.

Statistical Attacks Unfortunately, applying deterministic encryption without any improvements may pave the way for a potential confidentiality threat that needs to be assessed. Indeed, an adversary with direct access to the storage such as the Cloud Storage Provider, may perpetrate an attack thanks to the knowledge of a number of information that he can get without needing to decrypt users' data.

More precisely, we can consider this kind of attack as a particular instance of a practical ciphertext-only attack, where the attacker, that in this case may be the Cloud Storage provider or a malicious insider, does not know how to break the

encryption algorithm, hence obtain the plaintext given a ciphertext, but he has full access to the ciphertexts and an a priori knowledge of the plaintexts such as their distribution, their size or their format. Thanks to these information, an attacker may be able run statistical attacks, mainly based on frequency analysis, on the ciphertexts in order to link them back to the plaintexts and thus break the protection provided by the encryption layer.

In the context of block-level deduplication, since a fully deterministic encryption technique is required, such an attack may allow the attacker to exploit the presence of static data (e.g. predefined headers) in the plaintexts in order to efficiently revert encryption for part of the encrypted data and try to guess the remaining part through a brute-force dictionary attack. Moreover, since the feasibility of frequency-based attacks is related to the underlying distribution of data segments, there might be a direct relation between the data segment size and its distribution, meaning that a low data segment size may make statistical attacks easier. If such a relation does exist, we aim at determining whether increasing the block size can alleviate this problem and, if this is the case, what block size assures an optimal tradeoff between confidentiality, which is our primary goal, and deduplication ratio, which should remain high enough not to lose the advantages of block-level deduplication.

We aim at investigating this relation and the feasibility of frequency-based attacks in real scenarios. The outcome of this study may have a crucial impact on the security of the existing and future solutions. Indeed, in the worst case, this study may reveal that deterministic encryption in block-level deduplication enables an attacker with access to the storage to perpetrate frequency-based attacks, hence introduces a potentially severe confidentiality threat affecting most of the solutions based on deterministic encryption. On the other hand, in the best case, this study may assess that frequency-based attacks are not feasible. Also, there might be a tradeoff between deduplication savings and confidentiality, hence in this case we aim at finding an operational point at which deduplication savings are still significant and confidentiality is guaranteed.

Our Study Given these considerations regarding the likelihood of a statistical attack, we decided to perform an in-depth study in order to assess whether the information leaked because of the use of deterministic encryption in conjunction with deduplication may actually put confidentiality at risk.

In order to do so and make the study even more valuable, we first ran a series of experiments with two main goals in mind: first, we wanted to have a tangible proof of the effectiveness of deduplication in real scenarios; second, we wanted to conduct our security study using a realistic deduplication setup, including the average block size for block-level deduplication. The latter is a fundamental starting point for the security study since it affects the block frequency distribution and, as a consequence, the likelihood of a statistical attack.

In addition to that, we also observed that none of the studies currently available in the literature was comprehensive enough to answer our questions. This lack was mainly due to the fact that all of them were conducted on experimental datasets or particular datasets containing only specific kinds of data, which cannot be considered as good candidates for representing real use-cases.

Experiments and Results For the above-mentioned reasons, we ran a number of experiments on several real datasets belonging to the students and the staff of the EURECOM research institute, whereby each dataset corresponds to a different Cloud Storage use-case. The goal of these experiments was twofold. First, we wanted to verify the space savings that are achievable by putting in place a data deduplication solution. In particular, we wanted to find out what kinds of files are more likely to contain duplicate segments and which data-chunking technique achieves higher space savings. Also, we performed the same analysis with different data-chunking techniques and varying data segment size and granularity.

The results provided the needed starting point for the security study, that is the most effective data-chunking technique and the realistic average block-sizes. In this regard, variable-size block-level deduplication with an average block size of 4, 8 or 16 KB proved to be the most effective strategy. Also, they provided interesting hints and may be used as an additional informative support when evaluating the right deduplication strategy to be used in a given context. A comprehensive analysis along with an explanatory discussion can be found in next sections.

We then used this setup to analyze the distribution of data segments popularity, measure the entropy of the whole dataset and observe its evolution across different average block sizes. Starting from that, we investigated the existence of the potential confidentiality threats introduced by deduplication and verified whether and when the information possessed by the Cloud Storage provider are sufficient

to perform a statistical attack based on frequency analysis, which would leak sensitive information on users' data. In short, the outcome of this analysis proved that frequency-based statistical attacks are not feasible in such a context. More details are provided in next sections.

To the best of our knowledge, this is the first study that has taken into consideration statistical attacks on deduplicated data. Based on the findings of this study, we will then be able to design schemes for secure data deduplication.

3.2 Datasets

Deduplication may bring different storage space savings depending on the composition of the dataset on which it is performed. Indeed, as our study shows, some particular kinds of files are more likely to contain duplicates, whereby other kinds of files usually contain a lower percentage of duplicate blocks. Also, an important factor that may lead to lower or higher storage space savings is the data-chunking technique used for splitting a file into blocks. The first goal of our experiments was to clarify these two aspects and give useful recommendations with respect to the proper strategy to adopt in order to maximize the storage space savings for a given dataset. The datasets we analyzed are the following:

- Dataset 1: a POP Email Server containing all emails of 1078 users;
- Dataset 2: an IMAP Email Server containing all emails of 1571 users;
- Dataset 3: the home folders of 117 users containing more than 2M files;
- Dataset 4: a repository of data related to the research activity containing more than 4,5M files;
- Dataset 5: a repository of data related to the teaching activity containing more than 1,6M files;
- Dataset 6: a set of 7 Linux VM images.

Table 3.1 provides further details about the characteristics of each dataset. More importantly, we also analyzed the composition of each dataset, meaning the most common file types present in each of them. This information is shown in Figures

	Dataset 1	**Dataset 2**	**Dataset 3**	**Dataset 4**	**Dataset 5**
Number of users	1078	1571	117	-	-
Number of files	-	-	2074955	4516911	1615286
Percentage of identical files	0	12,15	49,98	17,24	45,77
Mean file size (in MB)	8,36	0,08	0,01	0,09	0,33
Total size (in GB)	9,35	153,52	209,87	391,82	519,19

TABLE 3.1: Description of datasets used for this analysis

3.1, 3.2 and 3.3 and is crucial in order to determine whether a dataset, given his composition, is likely to have a high rate of duplicate content, hence is a good candidate for performing data deduplication. Our analysis also includes the mean file size per dataset which confirms a widespread and well-known trend: the majority of files are very small (a few KB) and most of the storage space is consumed by a few very large files. As a confirmation of this statement, the cumulative density function of file sizes shows that in all datasets, except Dataset 1, more than 98% of files are smaller than 1MB. At a first glance, one may think that very small files are unlikely to contain duplicate content. However, the results illustrated in this chapter show that despite this trend the storage space savings are still very high, thanks to the frequent redundancy in larger files.

3.2.1 Dataset 1 (Emails POP)

This dataset consisted of one single file (aggregating all messages) per user. Therefore, we may consider it as a set of text files. This dataset represents the common use-case of POP server backup, which consists of storing a backup copy of all messages of an internal email server. This use-case is extremely interesting since emails are likely to contain high amounts of redundant data due to attachments, therefore they are one of the most suitable candidates for data deduplication.

3.2.2 Dataset 2 (Email IMAP)

This dataset consisted of approximately 1000 files per user. All of them were text files containing one or more messages. As above, deduplicating the backup of an email server can lead to high storage space savings.

3.2.3 Dataset 3 (Users Homes)

This dataset includes file system contents from a large set of users composed by students, professors, researchers, engineers and other members of the EURECOM staff. The vast majority of these file systems were based on either Linux distributions such as Fedora, Debian and Ubuntu or Windows 7. As shown in Figure 3.1, the set of file types was quite diverse, resulting in a highly fragmented dataset. However, a significant fraction of this dataset was composed by source-code (research and student projects), SVN repositories and images. Thanks to its very diverse composition, this dataset is suitable for approximating the kind of data that are often stored in the Cloud by standard users. Therefore, this dataset may be seen as a good approximation of the scenario corresponding to major Cloud Storage Providers such as Dropbox, Google Drive, etc.

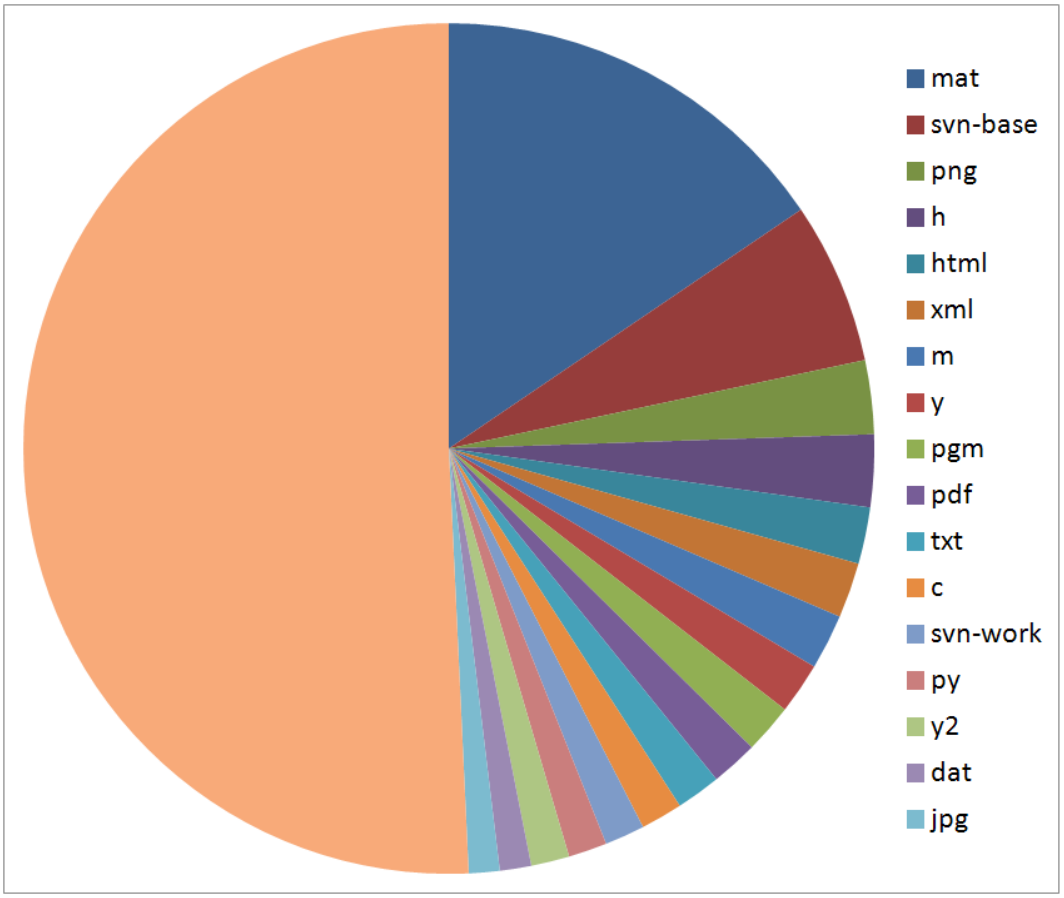

FIGURE 3.1: Composition of Dataset 3 (Users Homes)

3.2.4 Dataset 4 (Research)

This dataset consisted of a set of files used for backup, collaboration and development in research projects. Once again, the set of file types was quite diverse, as shown in Figure 3.2. This dataset can be appropriate to represent the use-case of a company that needs a shared storage space as a support for internal projects.

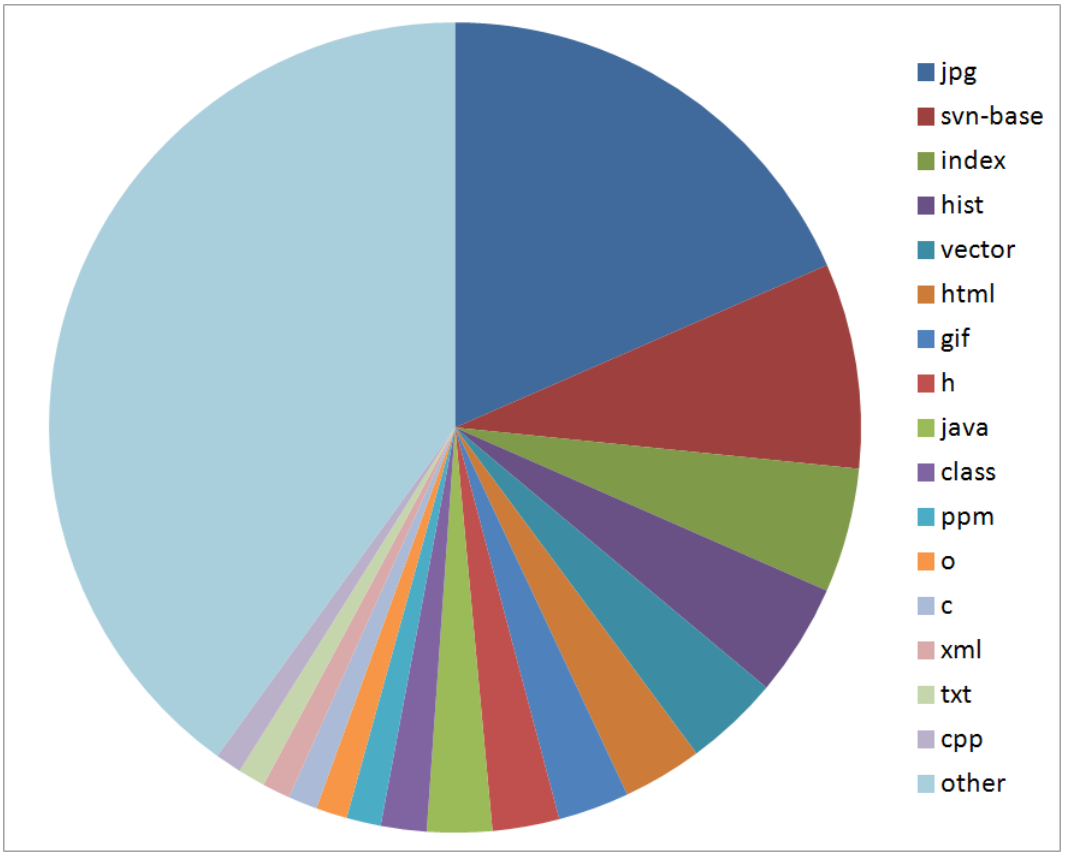

FIGURE 3.2: Composition of Dataset 4 (Research)

3.2.5 Dataset 5 (Teaching)

This dataset consisted of a set of files used as a support for the teaching activity within the EURECOM institute across several years. The composition of this dataset is shown in Figure 3.3. Since the class material does not always change from one year to another, this dataset can effectively represent the use-case of a company with many employees that often store high amounts of redundant data (documents).

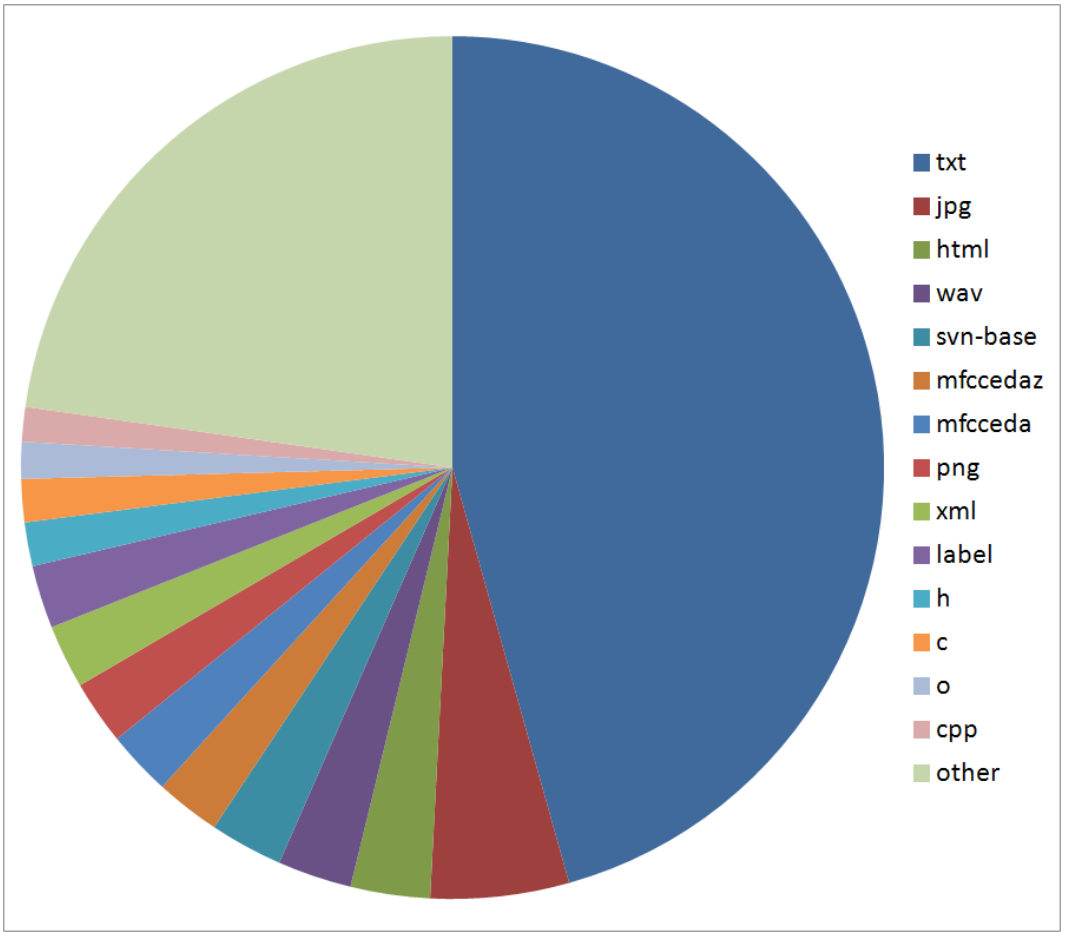

FIGURE 3.3: Composition of Dataset 5 (Teaching)

3.2.6 Dataset 6 (Linux VM images)

This dataset consisted of 7 different Linux VM images, with a size varying from 10GB to 30GB. All VM images were in vmdk format. VM images are usually likely to contain a high fraction of redundant data, especially when they are based on the same operating system. This dataset is perfectly suited for proving data deduplication effectiveness in the widespread use-case of backup and the storage of VM images.

3.3 Technical Environment

Deduplicating data at the level of the blocks requires to store the identifier of each block and also keep track of the structure of each file. In order to perform this

kind of analysis on such large datasets containing millions of files, we needed to employ extremely efficient tools through which we could store massive amounts of metadata in a very short period of time. Also, block-level deduplication raises the requirement of being able to quickly check whether a block was already stored. This quick lookup operation is done by using the unique block identifier, which is usually obtained by calculating an unkeyed secure hash of the block content. Given these two critical requirements, we identified a lightweight NoSQL database as the right software component to fulfill the aforementioned requirements. In particular, we decided to use REDIS [9], a well-known and widely used key-value store which provides the following features "out of the box":

- efficient (constant time) lookup of complex data structures through a unique key thanks to the internal data structure based on an hash table;
- storage of complex data structures such as lists, hash tables, sets, etc.;
- great performance when storing massive amounts of data in a very short period of time;
- in-memory storage with optional backup copy on disk.

These features enabled us to quickly scan all files in the aforementioned datasets, process them as shown in Figure 3.4 and finally collect the results shown in next section. In particular, for each dataset, we used REDIS to store the set of block IDs composing each file and also keep track of the popularity (number of copies) of each block in the current dataset. Thanks to these information, we could easily compute the amount of deduplicated data blocks, hence the percentage of storage space saved thanks to deduplication. REDIS allowed us to efficiently detect whether a given block was already stored by checking whether or not a given key was stored in REDIS. Indeed, REDIS maintains an internal hash table in memory, hence it is possible to retrieve or check the existence of a data structure (through its unique key) in a very efficient way.

We analyzed the datasets using different data-chunking techniques. In particular, we measured the storage space savings both in case of block-level deduplication and file-level deduplication. Block-level deduplication was implemented with two different data-chunking techniques: fixed-size and variable-size blocks. The latter was tested against three different configurations with an expected average block

size of 4KB, 8KB and 16KB respectively. The technique we adopted to determine block boundaries, that is the point at which the current block is cut and next block starts, was based on the rolling-hash algorithm Rabin fingerprint [56], which is widely used and known to be both efficient and effective.

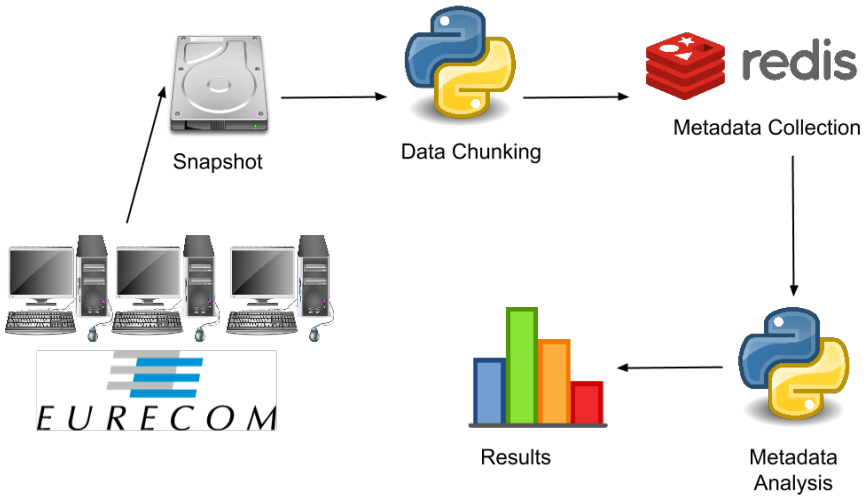

FIGURE 3.4: Diagram summarizing the main steps of the data analysis

It is worth pointing out that as opposed to the study conducted in [12], our study was conducted offline on a single snapshot of the file system contents, hence we measured the amount of data that may be deduplicated at a given point in time. Of course, performing the same kind of analysis on a series of consecutive snapshots of the same file systems, which corresponds to a backup scenario, would have given even better results. Indeed, backup is one of the most attractive scenarios for deduplication, due to the likelihood of finding duplicate content between consecutive snapshots. More precisely, backup, which can be either full or incremental, is traditionally accomplished by periodically storing a snapshot of file system contents: since file systems do not tend to significantly change from one snapshot to another, two consecutive snapshots are very likely to contain a large amount of unchanged and thus identical data.

3.3.1 Performance Overhead

Intuitively, the lower the average block size is, the higher the obtained storage space savings are. However, reducing too much the mean block size introduces the

drawback of increasing the number of blocks stored in the system, hence the overhead due to the fragmentation of a file and the storage of the resulting metadata. For instance, when dealing with very large datasets (millions of files), the storage of the metadata may require tens of GBs: if the deduplication system does not have enough resources, the overall performance may be negatively affected.

In our experiments, we tried to minimize the amount of metadata by only storing those information that were strictly required to effectively calculate storage space savings, individual block popularity and block popularity distribution per file type. Of course, in a real storage system for production environments, more fine-grained data would be needed.

Furthermore, we used SHA256 as hash function to calculate global and unique block identifiers. Since REDIS, as many other NoSQL databases, requires the key to be a UTF8-encoded string, the output (a raw sequence of 32 bytes) produced by the hash function was encoded into a Base64 string. As opposed to to the traditional hexadecimal encoding, the efficiency of this encoding method allowed us to save a non-negligible amount of memory, namely 33.5%. The decision of using SHA256 as block identifier function was not motivated by a need for security, rather, it was pondered taking into consideration the high number of different blocks in each dataset: using a hash function with a shorter output or a less uniform distribution could negatively affect the precision of the results due to the presence of a non-negligible number of false matches, meaning different blocks being mapped to the same ID. In addition to that, by using such a reliable hash function we could greatly improve the performance of the analysis thanks to the elimination of the need for a deep comparison, meaning the byte-by-byte comparison of blocks.

Regarding the metadata storage overhead and the lack of resources, in the case of REDIS, when the currently available memory is not sufficient, part of the dataset is stored on disk, which causes a considerable performance deterioration due to the rise of the time needed to read the internal hash table, resulting in a very poor performance of the deduplication operation. Generally, other kinds of metadata storage technologies such as relational databases suffer from similar performance issues when dealing with huge datasets without having been properly configured.

Therefore, the recommended practice is to set the mean block size to a value such that there is a good compromise between the deduplication ratio, which should be

as high as possible, and the amount of metadata to be stored, which should not exceed the available resources.

3.4 Storage Space Savings

File-level vs Block-level Figure 3.5 shows the percentage of storage space savings obtained on each dataset by using different data-chunking techniques. As shown in Figure 3.5, although datasets were quite different, all of them contained a non-negligible percentage of redundant data blocks, hence for all of them block-level deduplication proved to be beneficial. In particular, in our experiments, an average block size of 8KB appeared to provide both high deduplication ratios and an affordable and scalable metadata management. On the other hand, as expected, file-level deduplication proved to be way less effective than block-level deduplication and completely ineffective when applied on some datasets, leading to storage space savings lower than 1%.

The VM images use-case As an example of the ineffectiveness of file-level deduplication, performing it on VM images is likely to bring no storage space savings, whereas in our experiments block-level deduplication was able to detect a high percentage of duplicate blocks (more than 70%). The reason for the high presence of redundant data in that dataset is twofold: first, all VM images were based on a Linux distribution, hence they have the same kernel; second, VM images are very likely to contain a high percentage of identical "zero" blocks, which are blocks that have only zeros written into them or sequences of identical bytes, hence, all zero blocks can be easily deduplicated by storing only one copy of them.

Another trend that can be easily observed is that the amount of deduplicated data seems to be directly proportional to the percentage of duplicate files: this is due to the simple fact that, even in the worst case, block-level deduplication always achieves a deduplication ratio which is not lower than the one achieved with file-level deduplication. Indeed, if two files are identical, block-level deduplication, which is a deterministic technique, will definitely split them into the same set of blocks and all of them will be successfully deduplicated.

Dataset diversity Moreover, it is worth pointing out why the teaching dataset, despite a similar percentage of identical files, contains many more duplicate blocks

compared to the users homes dataset. This remarkable difference is due to the degree of diversity of the dataset: as discussed and shown in Figures 3.1, 3.2 and 3.3, the teaching dataset has a more homogeneous composition, meaning that there a few file types that represent the vast majority of the dataset, whereas the users homes dataset contains files belonging to many users and thus has much more diverse composition. In addition to that, as we pointed out earlier, the teaching dataset includes teaching material of several years, which is unlikely to significantly variate from one year to another.

Fixed-size vs Variable-size As expected, the results prove that splitting files into variable-size blocks (using Rabin fingerprinting) allows to achieve higher storage space savings with respect to fixed-size blocks. Also, the results confirm an expected trend: the smaller the block is, the higher is the chance of finding duplicate blocks, thus increasing the space savings.

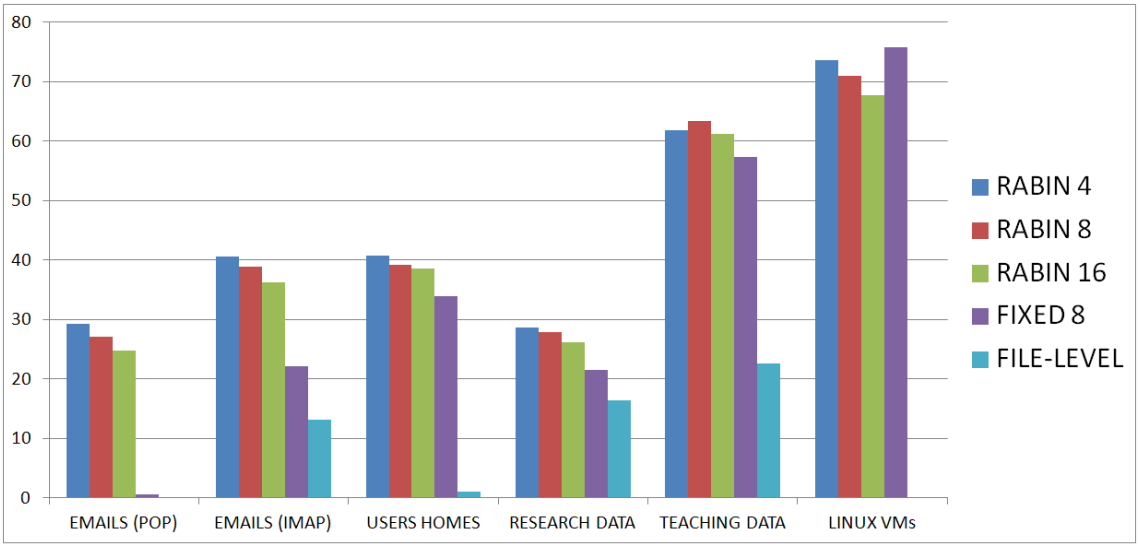

FIGURE 3.5: Storage space savings achieved with different data-chunking techniques

Redundancy based on file type Figure 3.6 shows the distribution by file extension of duplicate blocks detected through the Rabin fingerprinting algorithm with a mean block size of 8KB, where duplicate blocks are those blocks that are encountered at least twice, even in the same file. As Figure Figure 3.6 shows, duplicate blocks are quite fragmented, meaning that they are distributed across a very diverse set of file types. However, it is easy to see that the majority of duplicate blocks belong to VM images (vmdk), uncompressed media files (mpg, ppm, bmp), repositories (svn-base) and documents (pdf, txt, doc, ppt). What

all these file types have in common is the high presence of redundant content. Therefore, these results give an important hint: block-level deduplication can be extremely effective not only in those datasets containing identical copies of the same files shared between multiple users, but also in those datasets containing a high percentage of uncompressed data. Surprisingly, a big fraction of all duplicate blocks is due to compressed archives, namely zip, jar and tar files. Although this may sound counterintuitive since compression is usually employed to remove redundant content, we speculate that archives may contain predefined headers and other static metadata which are common to all of them. Hence, the Rabin fingerprinting algorithm may be able to find these blocks and deduplicate them.

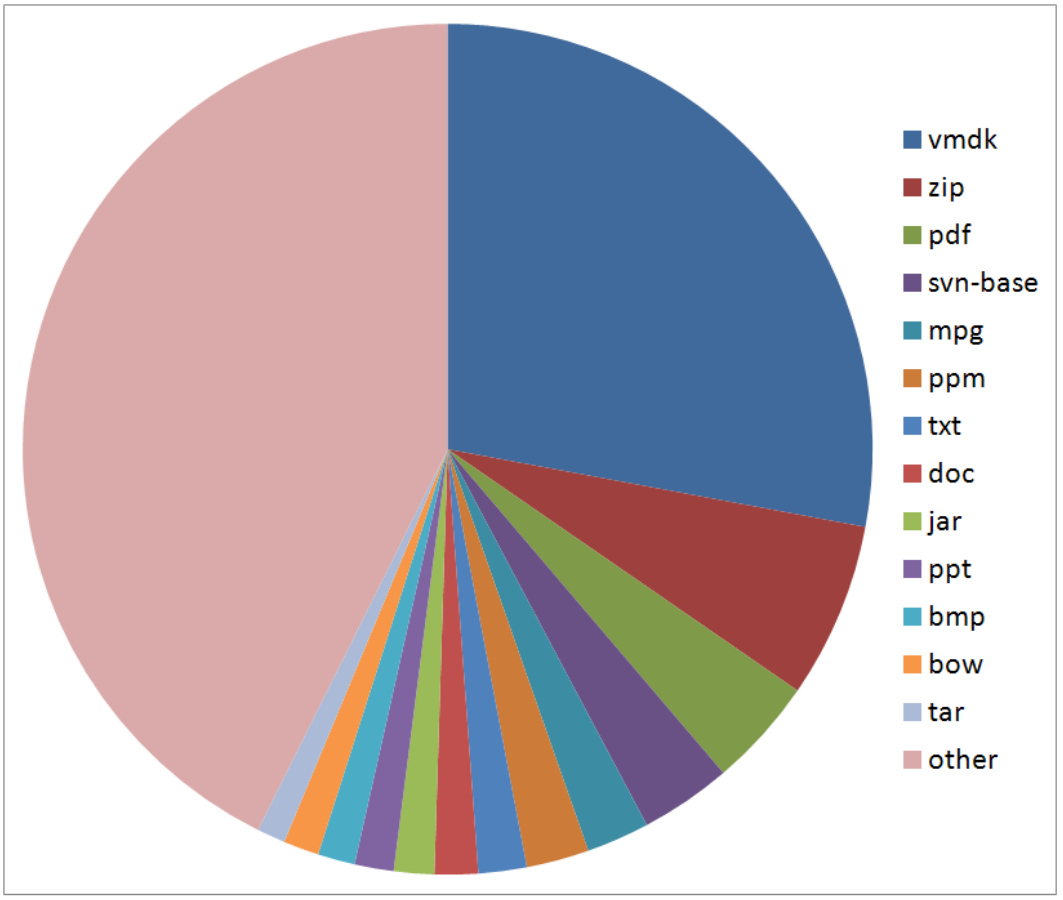

FIGURE 3.6: Distribution of duplicate blocks by file type

3.5 Statistical Attacks

In order to assess the feasibility of statistical attacks, we conducted a number of experiments in order to collect information on the number and the frequency duplicate blocks and, more generally, the evolution of the set of data segments when varying the average block size. Based on the findings of the study on storage space savings, all experiments were conducted using three different data-chunking configurations, namely Rabin fingerprinting with three average block size values: 4KB, 8KB and 16KB, which proved to provide an affordable tradeoff between metadata storage overhead and deduplication ratios. As stated above, these experiments were aimed at assessing whether block-level deduplication combined with deterministic encryption may pose a threat with respect to confidentiality. More precisely, the goal of these experiments was to assess whether a malicious Cloud Storage provider, aiming at discovering (part of) the original content of users' encrypted data, may be able to perpetrate a successful and practical ciphertext-only attack based on the analysis of the ciphertext frequency. Also, we wanted to investigate the relationship between the data segment size and the feasibility of such an attack. Indeed, intuitively, a low average block size reduces the number of possible blocks, hence may increase the chance of associating a ciphertext to a plaintext by analyzing its frequency.

Block Frequency Distribution and Entropy The first finding worth pointing out concerns the distribution of duplicate blocks. The frequency of each block was determined by dividing the number of occurrences of that block across all stored files (including multiple occurrences within the same file) by the total number of blocks in all stored files (without filtering out non-unique blocks).

As pointed out in previous works such as [6], we found out that the frequency distribution of duplicate blocks is far from being uniform, meaning that there are a few blocks that are much more frequent than the others. More generally, this confirms that data segments are not uniformly distributed. We also noticed that apart from the few very frequent blocks, all other duplicate blocks were somewhat uniformly distributed and had a very low and similar frequency, namely lower than 0.01%.

As previously stated in this chapter, we aim at assessing whether block-level deduplication can somehow weaken data confidentiality when used in conjunction with deterministic encryption. Therefore, in order to have a more relevant evaluation of

the overall distribution of blocks, we decided to compute the global entropy of the whole dataset (after performing block-level deduplication) for each data segment size. The global entropy E was computed using the following formula:

$$E = -\sum p_i \cdot \log_2 p_i$$

where p_i is the frequency of the block i across the whole dataset, that is the number of its occurrences divided by the total number of blocks stored in the system.

Measuring the entropy and its evolution when decreasing the average block size allowed us to determine the impact in terms of confidentiality of block-level deduplication across different data-chunking scenarios. More intuitively, entropy is traditionally used to measure the degree of uncertainty of a dataset. In our scenario, entropy could be used to assess whether or not guessing becomes easier when decreasing the block size. Therefore, observing the variation of the entropy does not give us a measure of the level of confidentiality in absolute terms, rather it provides an indication on how the likelihood of a statistical attack changes when varying the average block size. For instance, if by decreasing the block size the global entropy significantly decreases, it means that the chances of a successful statistical attack are increasing accordingly. On the other hand, if decreasing the block size does not have any significant impact of the entropy, it means that the risk of a successful statistical attack is not affected by the smaller block size.

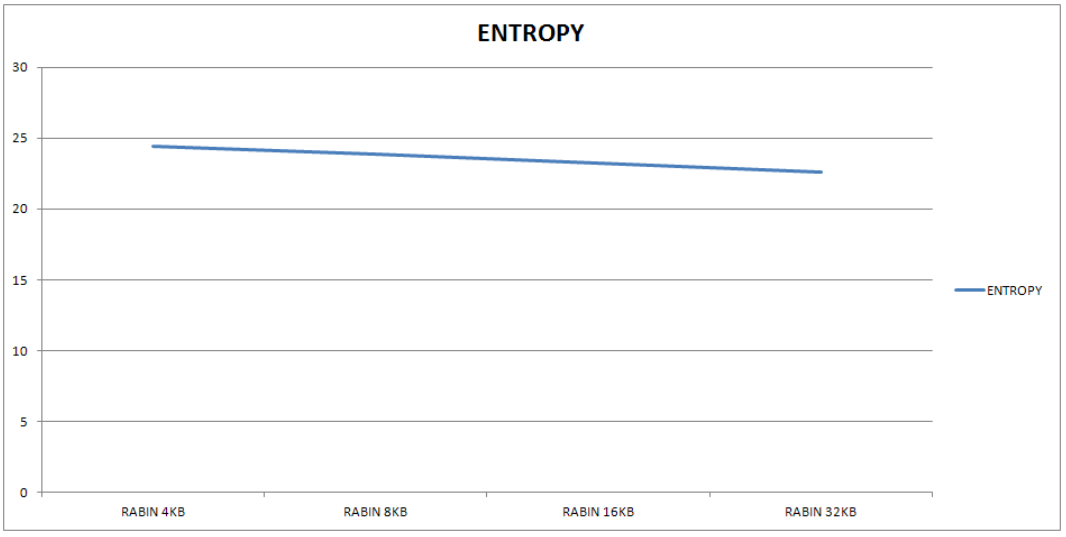

FIGURE 3.7: Global entropy with varying average block size

Surprisingly, as shown in Fig. 3.7, the experiments conducted with different average block sizes led to very similar results. Regardless of the configuration being used, the entropy values were very close and the distributions of duplicate blocks were almost identical. More precisely, decreasing the average block size up to 4KB, which is already a very low value in real systems, slightly increases the entropy instead of decreasing it, as we supposed when describing our motivations behind this study. Such a behavior is apparently counterintuitive. Indeed, intuitively, decreasing the block size should cause an increase in the level of redundancy that can be found and thus reduce the number of all possible values that a data segment can take. Instead, in our experiments the inverse phenomenon occurred. In fact, we observed that due to the large increase of the total number of data segments in the system, the resulting entropy also increases. Of course, by further decreasing the block size (e.g. lower than 1KB) the entropy will certainly drop at some point, however we decided to only focus on realistic configurations, therefore we did not take into account these extreme cases which are out of the scope of our study.

Content of popular blocks Starting from the distribution of duplicate blocks, we decided to analyze the content of the most frequent blocks in order to assess whether their presence may leak some sensitive information to the Cloud Storage provider. We found out that independently of the data-chunking technique being used, the most frequent data block is always the so-called "zero" block, as we introduced in the previous sections. Indeed, this result is a consequence of the fact that VM image files are the ones with the highest percentage of duplicate content. Moreover, we analyzed the content of other duplicate blocks in order of frequency, and we found out that the vast majority of duplicate blocks included headers, metadata (in particular SVN repositories) and other kinds of blank blocks, that are those unused blocks filled with identical sequences of bytes.

Feasibility of statistical attacks Based on these two findings, that are the distribution of duplicate blocks and the variation of the global entropy, we can state that there is no concrete evidence of a correlation between the data segment size used in block-level deduplication and the likelihood of a successful ciphertext-only attack. Indeed, contrarily to our initial hypothesis, decreasing the block size does not seem to negatively impact the global entropy and the distribution of duplicate blocks. Of course, this conclusion remains true only when using realistic block sizes: if a too small block size is mistakenly chosen (e.g. a few tens of bytes), such a statement may not be true anymore.

Therefore, assuming that the deterministic encryption layer is strong enough not to be broken and the average block size is not extremely small (at least 4KB), these results show that ciphertext-only attacks based on content frequency are unlikely to be successful in the context of block-level deduplication and thus can hardly lead to the leakage of sensitive information, even when using a very small average block size such as 4KB.

Indeed, a frequency-based attack in order to be successful would require a more characteristic distribution and a low entropy. The first requirement is raised by the necessity of being able to precisely identify a block starting from its frequency. The second requirement implies that the degree of variety in the dataset is low, hence an attacker may be able to run more targeted attacks on a lower number of blocks. Fortunately, according to our studies, both requirements seem to be missing when using a realistic average block size.

However, as introduced by the intuition behind this study, the Cloud Storage provider has access to a non-negligible knowledge base including file size, file access patterns and file owners which, together with our findings on duplicate blocks frequency, may disclose the type of content stored in a given encrypted file. Of course, this information alone is not sufficient to be considered as a serious threat to the confidentiality of users' data, even though it may be helpful for the attacker. For instance, if the attacker knew that a given encrypted file is likely to be a SVN repository, he may run a brute-force attack trying to generate content in a known format, instead of merely generating random content: this is what researchers mean by "predictable files" [5], for which CE cannot guarantee confidentiality. However, we assume that a deterministic encryption mechanism stronger than CE is employed. Therefore, although such a knowledge on the victim's data may help the attacker to reduce the complexity of brute-force attacks in some particular scenarios, it would only allow the attacker to concentrate his energies on a specific format, which does not eliminate the protection provided by the encryption layer and thus still makes such an attack very difficult if not unfeasible.

Recommended block size As an additional result provided by this study, it is worth pointing out that since there is no visible correlation between the average block size and the increase of the risk for ciphertext-only attacks, using a low block size such as 4KB seems to be the recommended approach since it allows to achieve the best storage space savings without apparently compromising data confidentiality. Besides the security considerations, since reducing the block size

raises the overhead due to metadata management, performance and scalability are also very important factors that should be taken into account.

3.6 Popularity

In addition to the aforementioned findings on storage space savings and statistical attacks, we decided to take advantage of such a comprehensive analysis on real datasets in order to dig into another important aspect of deduplication schemes, that is popularity. Indeed, some secure data deduplication schemes, such as PerfectDedup and [7], apply a different treatment to data segments based on their popularity. More precisely, data segments are labeled as either popular or unpopular and based on their label a different encryption mechanism is used, namely convergent encryption for popular data segments and a semantically-secure encryption for the others. Besides the security considerations on the encryption mechanisms being used, another crucial question arises: how can we determine whether a data block is popular? In more practical terms, if we assume that popularity of a data segment is given by the number of occurrences or users owning it, the question becomes: what is the best popularity threshold?

In order to answer this question, which is far from being trivial, we analyzed in depth the frequency of duplicate blocks with a precise goal in mind: finding a popularity threshold allowing to achieve a good tradeoff between storage space savings and confidentiality. Ideally, the popularity threshold should be chosen in a way that storage space savings remain high while there is no confidential data segment that is considered popular and encrypted with a weaker encryption mechanism. In other words, if the popularity threshold was set to a too high value, the storage space savings would drastically drop. making the whole system unbeneficial. On the other hand, setting it to a too low value would bring higher storage space savings at the cost of potentially increasing the risk of labeling a data segment containing sensitive information as popular and thus protecting it with a weaker encryption mechanism.

Of course, a straightforward strategy would consist of setting the popularity threshold to the lowest value possible, that is 2, and thus deduplicating every data segment belonging to more than one user. Unfortunately, in some particular cases, this may have the unfortunate effect of opening a confidentiality breach. For

instance, if a confidential file was shared between multiple users, it would be considered popular, hence encrypted with convergent encryption, which would make it vulnerable to dictionary attacks.

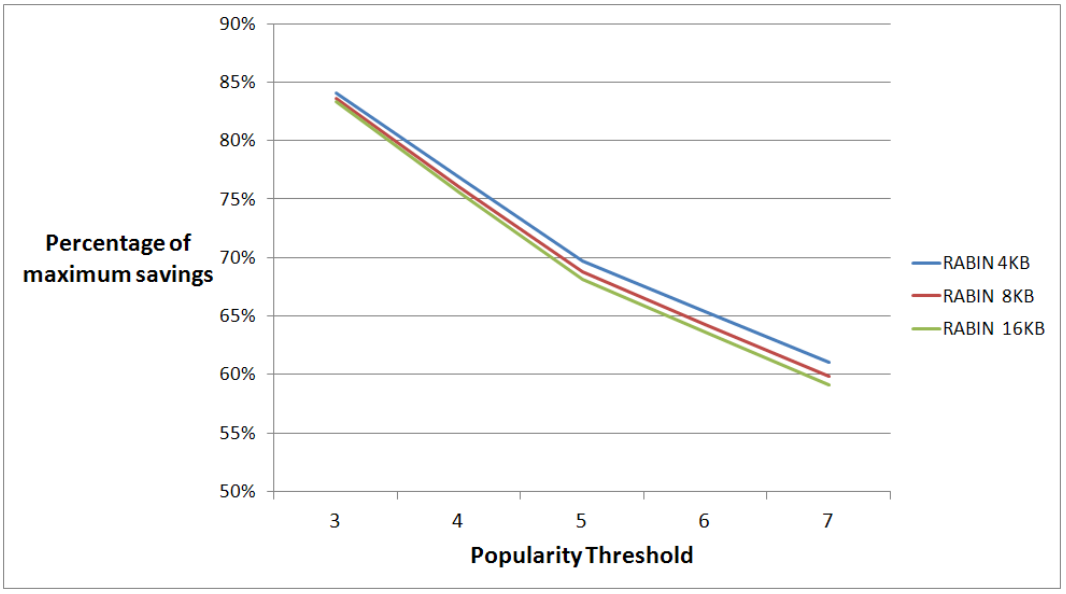

FIGURE 3.8: Decrease of storage space savings when increasing the popularity threshold

In order to have a clearer idea of how negatively the increase of the popularity threshold affects the storage space savings, we decided to measure the percentage of storage space savings that can be achieved with an increasing popularity threshold value going from 3 to 7 and a varying average block size going from 4KB to 16KB. As shown in Fig. 3.8, a small increase of the popularity threshold corresponds to a remarkable decrease in the amount of storage space savings. As an example, no matter what average block size is being used, slightly increasing the popularity threshold from 2, which is the minimum, to 3, causes a decrease of more than 15% in the total amount of storage space savings. Also, this trend seems to be independent of the average block size. Furthermore, the storage space savings keep decreasing when further increasing the popularity threshold to higher values. For instance, when setting the popularity threshold to 7, the storage space savings drop by almost 40%.

These measurements confirm the intuition that the popularity threshold should remain low enough not to lose the advantages of block-level deduplication. Also, to answer our initial question, that is the popularity threshold value that assures a good compromise between savings and confidentiality, it is hard to give a general

answer since it strictly depends on the scenario and the composition of the dataset. For instance, considering our dataset, in a scenario where storage space savings have higher priority, even a slight decrease of 15% would not be acceptable. On the other hand, in a scenario where very confidential files are frequently stored, a decrease up to 30% in storage space savings may be accepted in favor of better security, therefore the popularity threshold may be set to 5.

3.7 Conclusions

The reason behind this comprehensive study on practical deduplication was twofold. First, we wanted to make sure that deduplication is effective and beneficial in real scenarios and figure out the best setup in terms of storage space savings. Also, we wanted to determine what type of content is more likely to contain redundancy, hence we collected a number of real datasets and performed an extensive analysis comparing the storage space savings across different data-chunking techniques. Second, we wanted to assess whether performing block-level deduplication could realistically pose a threat with respect to data confidentiality. More precisely, we were interested in verifying whether deterministic encryption, which is a requirement for deduplication, could pave the way for ciphertext-only attacks based on the knowledge of duplicate blocks frequency. As mentioned earlier, the answer to this question is crucial for any scheme aiming at providing secure data deduplication.

Storage space savings Our experimental results prove that deduplication, especially when performed at level of blocks and with a variable block size, can bring significant storage space savings in real scenarios. Depending on the scenario, that is the nature of data stored by users, storage space savings may be greater than 70%. Moreover, we point out that in backup scenarios, where redundancy is extremely high, storage space savings may be even higher.

Statistical attacks We studied the distribution of duplicate blocks and measured the global entropy in order to assess whether statistical attacks such as frequency analysis are likely to be successful, hence enable a malicious Cloud Storage provider to (partially) discover the content of encrypted data stored by users. Our experimental results show that the number of duplicate blocks is extremely high and their distribution, apart from a few very popular blocks, is not characteristic enough to

allow for a statistical attack based on frequency analysis. Also, the global entropy does not decrease when decreasing the average block size, which means that using relatively low average block sizes (e.g. 4KB) does not negatively affect the degree of confidentiality provided by the scheme.

However, as we pointed out above, the presence of one of the very popular blocks together with the knowledge of file size, file owners, access patterns and file structure may leak the type of content stored in the file, which is far from being a sensitive information and alone is not sufficient to break or weaken the protection provided by the encryption and put data confidentiality at risk. From a research standpoint, these results are very promising since they show that the combination of block-level deduplication with deterministic encryption does not pave the way for statistical attacks. Therefore, schemes based on deterministic encryption can effectively address secure block-level data deduplication and do not suffer from this hypothesized inherent weakness that was one of the main motivations behind our study.

Chapter 4

ClouDedup

4.1 Introduction

Thanks to our study on the security of block-level deduplication schemes based on deterministic encryption, we can now start from the assumption that deterministic encryption itself does not introduce any confidentiality threat. Therefore, as long as the encryption algorithm is secure, a scheme aiming at secure block-level deduplication can safely employ deterministic encryption as a main building block. It is worth pointing out that in this scenario deterministic encryption refers to the use of the latter on top of convergent encryption.

In this chapter, we cope with the inherent conflict between encryption and deduplication by designing a novel architecture, which goes by the name of ClouDedup, where files are protected by two layers of deterministic encryption: the first layer is convergent encryption and is applied by users before uploading the file; the second layer is applied by a trusted component which makes use of secret keying material stored locally and never stored with any other entity. Such an architecture also aims at delegating different tasks to different components in a way that any component alone does not have sufficient knowledge to compromise data confidentiality. In other words, this scheme achieves a principle known as "no single point of failure", which in this context means that the compromise of a single component cannot completely undermine the security provided by the whole system.

The security of ClouDedup thus relies on its new architecture whereby in addition to the basic storage provider, a metadata manager and an additional encryption

component, called gateway, are defined: the gateway prevents well-known attacks against convergent encryption and thus protect the confidentiality of the data; on the other hand, the metadata manager is responsible of the key management task since block-level deduplication requires the memorization of a huge number of keys. Therefore, the underlying deduplication is performed at block-level and we define an efficient key management mechanism to avoid users to store one key per block.

To summarize the advantages of ClouDedup:

- ClouDedup assures **block-level deduplication** and **data confidentiality**. Block-level deduplication renders the system more **flexible and efficient**;
- ClouDedup preserves **confidentiality and privacy even against potentially malicious cloud storage providers** thanks to an additional layer of deterministic encryption;
- ClouDedup offers an efficient key management solution through the metadata manager;
- The new architecture defines several different components and a **single component cannot compromise** the whole system without colluding with other components;
- ClouDedup works **transparently** with existing cloud storage providers. As a consequence, ClouDedup is fully compatible with standard storage APIs and any cloud storage provider can be easily integrated in our architecture.

4.2 The Idea

As discussed in previous chapters, using a deterministic encryption layer on top of convergent encryption provides full confidentiality for all data while allowing for deduplication by an untrusted party. The idea we develop in this chapter consists of delegating a trusted component to perform the additional deterministic encryption right after clients have already encrypted data blocks using convergent encryption. This way, data blocks encrypted with a weaker encryption mechanism, that is convergent encryption, will not be exposed to untrusted and potentially malicious components such as the Cloud Storage Provider.

The scheme proposed in this chapter aims at deduplication at the level of blocks of files encrypted with convergent encryption while guaranteeing confidentiality against an untrusted Cloud Storage Provider. The scheme consists of two basic components: an encryption gateway that is also in charge of access control and that achieves the main protection against COF and LRI attacks; another component, named as metadata manager (MM), is in charge of the actual deduplication and key management operations.

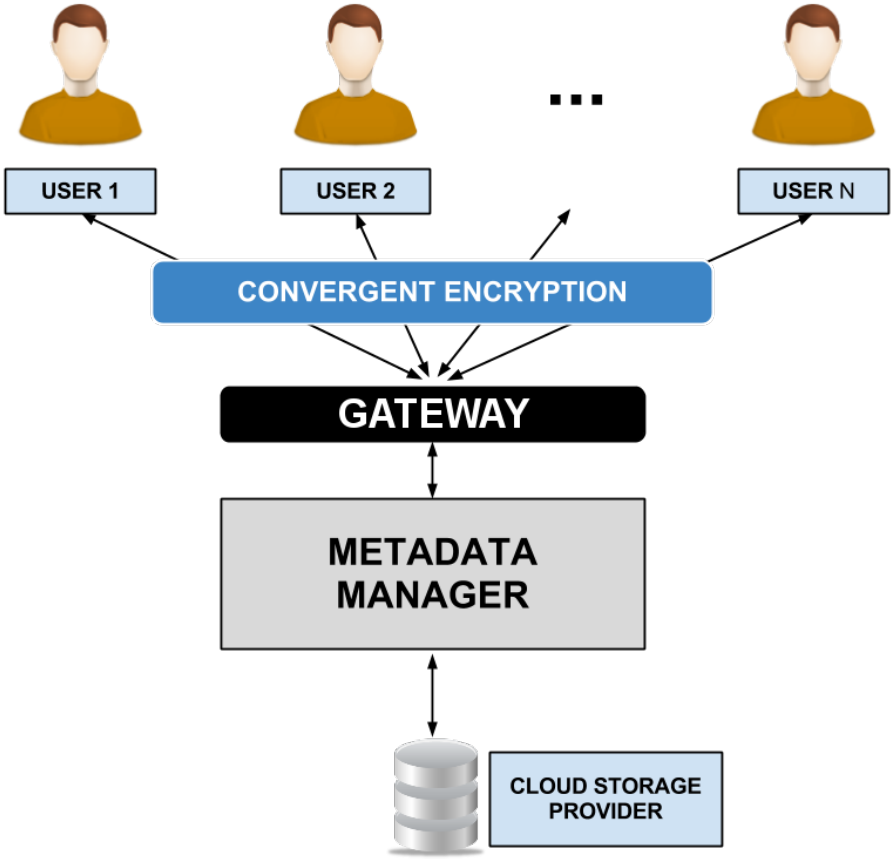

FIGURE 4.1: High-level view of ClouDedup

4.2.1 The Gateway

An effective and practical solution to prevent the attacks against convergent encryption (CE) consists of encrypting the ciphertexts resulting from CE with another encryption algorithm using the same keying material for all input. This solution is compatible with the deduplication requirement since identical ciphertexts resulting from CE would yield identical outputs even after the additional

encryption operation. Yet, this solution will not suffer anymore from the attacks targeting CE such as COF and LRI.

We suggest to combine the access control function with the mechanism that achieves the protection against CE through an additional encryption operation. Indeed, access control is an inherent function of any storage system with reasonable security assurance. Enhancing the trusted component of the storage system, that implements access control, with the new mechanism against COF and LRI attacks, seems to be the most straightforward approach. The core component of ClouDedup is thus an encryption gateway that implements the additional encryption operation to cope with the weaknesses of CE, together with a user authentication and an access control mechanism embedded in the data protection mechanism. Each data segment is thus encrypted by the gateway in addition to the convergent encryption operation performed by the user. As to the data access control, each encrypted data segment is linked with a signature generated by its owner and verified upon data retrieval requests. The gateway relies on the signature of each segment to properly identify the recipient.

4.2.2 Block-level Deduplication and Key Management

Even though the mechanisms of the encryption gateway cope with the security weaknesses of CE, the requirement for deduplication at block-level further raises an issue with respect to key management. As an inherent feature of CE, the fact that encryption keys are derived from the data itself does not eliminate the need for the user to memorize the value of the key for each encrypted data segment. Unlike file-level deduplication, in case of block-level deduplication, the requirement to memorize and retrieve CE keys for each block in a secure way, calls for a fully-fledged key management solution. We thus suggest to include a new component, the metadata manager (MM), in the new ClouDedup system in order to implement the key management for each block together with the actual deduplication operation.

4.2.3 Threat Model

The goal of the system is to guarantee data confidentiality without losing the advantage of deduplication. Confidentiality must be guaranteed for all files, including the predictable ones. The security of the whole system should not rely on the security of a single component (single point of failure), and the security level should not collapse when a single component is compromised. We consider the encryption gateway as a trusted component with respect to user authentication, access control and additional encryption. The gateway is not trusted with respect to the confidentiality of data stored at the cloud storage provider. Therefore, the gateway is not able to perform offline dictionary attacks. Anyone who has access to the storage is considered as a potential attacker, including employees at the cloud storage provider and the cloud storage provider itself. In our threat model, the cloud storage provider is honest but curious, meaning that it carries out its tasks but might attempt to decrypt data stored by users. We do not take into account cloud storage providers that can choose to delete or modify files and, more generally, behave in a byzantine way and not perform the operations requested by users. The motivation behind this choice is related to the economic model of Cloud Computing: a cloud provider which does not provide a sufficient quality of service would likely lose all customers and be out of the market very soon, hence we do not focus on this kind of edge cases. Our scheme might be extended with additional features such as data integrity [16] and proofs of retrievability [15].

Among the potential threats, we identify also external attackers. An external attacker does not have access to the storage and operates outside the system. This type of attacker attempts to compromise the system by intercepting messages between different components or compromising a user's account. External attackers have a limited access to the system and can be effectively neutralized by putting in place strong authentication mechanisms and secure communication channels.

4.2.4 Security

In the proposed scheme, only one component, that is the gateway, is trusted with respect to a limited set of operations, therefore we call it semi-trusted. Once the gateway has applied the additional encryption, data are no longer vulnerable to CE weaknesses. Indeed, without possessing the keying material used for the additional

encryption, no component can perform dictionary attacks on data stored at the cloud storage provider. The gateway is a simple semi-trusted component that is deployed on the user's premises and is in charge of performing user authentication, access control and additional symmetric encryption. The primary role of the gateway is to securely retain the secret key used for the additional encryption. In a real scenario, this goal can be effectively accomplished by using a hardware security module (HSM) [64], thanks to which secret keys can be generated in a hardware-dependent way by the device itself, which will not share them with anyone else. In order to guarantee data confidentiality even in the unfortunate case the server is compromised, an additional symmetric encryption can be applied by the MM before uploading blocks to the Cloud. When data are retrieved by a user, the gateway plays another important role. Before sending data to a given recipient, the gateway must verify if block signatures correspond to the public key of that recipient. The metadata manager (MM) and the cloud storage provider are not trusted with respect to data confidentiality, indeed, they are not able to decrypt data stored at the cloud storage provider. We do not take into account components that can spontaneously misbehave and do not accomplish the tasks they have been assigned.

4.3 Components

In this section we describe the role of each component.

4.3.1 User

The role of the user is limited to splitting files into blocks, encrypting them with the convergent encryption technique, signing the resulting encrypted blocks and creating the storage request. In addition, the user also encrypts each key derived from the corresponding block with the previous one and his secret key in order to outsource the keying material as well and thus only store the key derived from the first block and the file identifier. For each file, this key will be used to decrypt and re-build the file when it will be retrieved. Instead, the file identifier is necessary to univocally identify a file over the whole system. Finally, the user also signs each block with a special signature scheme. During the storage phase, the user computes the signature of the hash of the first block: $S_0 = \sigma_{PK_u}(H(B_0))$. In

order not to apply costly signature operations for all blocks of the file, for all the following blocks, a hash is computed over the hash of the previous block and the block itself: $S_i = H(B_i|S_{i-1})$. The high-level view of ClouDedup's architecture is illustrated in Figure 4.1.

4.3.2 Gateway

The gateway has three main roles: authenticating users during the storage/retrieval request, performing access control by verifying block signatures embedded in the data, encrypting/decrypting data traveling from users to the cloud and viceversa. The gateway takes care of adding an additional layer of encryption to the data (blocks, keys and signatures) uploaded by users. Before being forwarded to MM, data are further encrypted in order to prevent MM and any other component from performing dictionary attacks and exploiting the well-known weaknesses of convergent encryption. During file retrieval, blocks are decrypted and the gateway verifies the signature of each block with the user's public key. If the verification process fails, blocks are not delivered to the requesting user.

4.3.3 Metadata Manager (MM)

MM is the component responsible for storing metadata, which include encrypted keys and block signatures, and handling deduplication. Indeed, MM maintains a small database in order to keep track of file ownerships, file composition and avoid the storage of multiple copies of the same data segments. The data structures used for this purpose contain information on files, pointers and signatures. With respect to file structure and blocks, from a logical point of view, the data structures stored at the MM can be seen as a linked list that is structured as follows:

- Each node in the linked list represents a data block. The identifier of each node is obtained by hashing the encrypted data block received from the server.
- If there is a link between two nodes X and Y, it means that X is the predecessor of Y in a given file. A link between two nodes X and Y corresponds to the file identifier and the encryption of the key to decrypt the data block Y.

The other data structures used by MM are structured as follows:

- **File.** The file data structure contains the file id, file name, user id and the id of the first data block.
- **Pointer.** The pointer data structure contains the block id and the id of the block stored at the cloud storage provider.
- **Signature.** The signature data structure contains the block id, the file id and the signature.

In addition to the access control mechanism performed by the gateway, when users ask to retrieve a file, MM further checks if the requesting user is authorized to retrieve that file. This way, MM makes sure that the user is not trying to access someone else's data. This operation can be considered as an additional access control verification, since an access control verification already takes place at the gateway. Another important role of MM is to communicate with cloud storage provider (SP) in order to actually store and retrieve the data blocks and get a pointer to the actual location of each data block.

4.3.4 Cloud Storage Provider (SP)

SP is the most simple component of the system. The only role of SP is to physically store encrypted data blocks. SP is not aware of the deduplication and ignores any existing relation between two or more blocks. Indeed, SP does not know which file(s) a block is part of or if two blocks are part of the same file. This means that even if SP is malicious, meaning that he wants to access users' data in clear, it has no way to infer the original content of an encrypted data block and rebuild the files uploaded by the users. It is worth pointing out that potentially any cloud storage provider would be able to operate as SP. Indeed, ClouDedup is completely transparent from SP's perspective, which does not collaborate with MM for deduplication. The only role of SP is to store data blocks coming from MM, which can be considered as files, or objects in the case of object storage, of small size. Therefore, it is possible to make use of well-known and widespread cloud storage providers or technologies such as Amazon S3 [20], OpenStack SWIFT [40] and Microsoft Azure [39].

4.4 Protocol

In this section we give a high-level overview of the protocol regulating the two main operations of ClouDedup: storage and retrieval. The description of other operations such as removal, modification and search are out of the scope of this chapter.

Notation	
E_K	encryption function with key K
H	hash function
B_i	i^{th} block of a file
B'_i	i^{th} block of a file after convergent encryption
B''_i	i^{th} block of a file after encryption at the server
K_i	key generated from the i^{th} block of a file
K'_i	K_i after encryption at the server
K_A	secret key of server
K_{U_j}	secret key of user j
PK_{U_j}	private key of the certificate of user j
S_i	signature of i^{th} block of a file with PK_{U_j}

4.4.1 Storage

During the storage procedure, a user uploads a file to the system. As an example, we describe a scenario in which a user U_j wants to upload the file F1.

USER User U_j splits F1 into several blocks. For each block B_i, U_j generates a key by hashing the block and uses this key to encrypt the block itself. Therefore $B'_i = E_{K_i}(B_i)$ where $K_i = H(B_i)$. U_j stores K_1 and encrypts each following key with the key corresponding to the previous block: $E_{K_{i-1}}(K_i)$. U_j further encrypts each key (except K_1) with his own secret key K_{U_j}: $E_{K_{U_j}}(E_{K_{i-1}}(K_i))$. U_j computes the block signatures as described in 4.3.1. U_j sends a request to the server in order to upload file F1. The request is composed by:

- U_j's id : ID_{U_j};
- the encrypted file name;

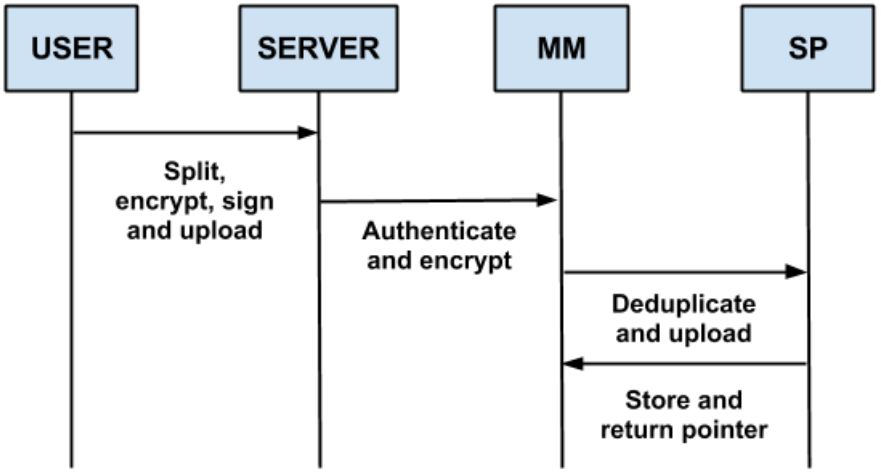

FIGURE 4.2: Storage Protocol

- file identifier : F_{id1};
- first data block : $E_{K_1}(B_1)$;
- for each following data block B_i ($i \geq 2$): key to decrypt block B_i, that is $E_{K_{U_j}}(E_{K_{i-1}}(K_i))$; signature of block B_i, that is S_i; data block B'_i : $E_{K_i}(B_i)$;

In order to improve the level of privacy and reveal as little information as possible, U_j encrypts the file name with his own secret key. File identifiers are generated by hashing the concatenation of user ID and file name $H(user\ ID \mid file\ name)$.

GATEWAY The gateway receives a request from user U_j and runs SSL in order to securely communicate with U_j. Each key, signature and block are encrypted under K_A (gateway's secret key): $B''_i = E_{K_A}(E_{K_i}(B_i))$, $K'_i = E_{K_A}(E_{K_{U_j}}(E_{K_{i-1}}(K_i)))$, $S'_i = E_{K_A}(S_i)$. The only parts of the request which are not encrypted are user's id, the file name and the file identifier. The gateway forwards the new encrypted request to MM.

MM MM receives the request from the gateway and for each block B''_i contained in the request, MM checks if that block has already been stored by computing its hash value and comparing it to the ones already stored. If the block has not been stored in the past, MM creates a new node in the linked list, the identifier of the node is equal to $H(B''_i)$. MM updates the data structure by linking each node (block) of file F1 to its successor. A link from block B''_{i-1} to block B''_i contains the following information: $\{F_{id1}, E_{K_A}(E_{K_{U_j}}(E_{K_{i-1}}(K_i)))\}$. It is worth pointing out

that each key is encrypted with the key of the previous block and users retain the key of the first block, which is required to start the decryption process. This way, a chaining mechanism is put in place and the key retained by the user is the starting point to decrypt all the keys. Moreover, MM stores the signature of each block in the signature table, which associates each block of each user to one signature. For each block B_i'' not already stored, MM sends a storage request to SP which will store the block and return a pointer. Pointers are stored in the pointer table, which associates one pointer to each block identifier.

SP SP receives a request to store a block. After storing it, SP returns the pointer to the block.

MM MM receives the pointer from SP and stores it in the pointer table.

4.4.2 Retrieval

During the retrieval procedure, a user asks to download a file from the system. As an example, we describe a scenario in which a user U_j wants to download the file F1.

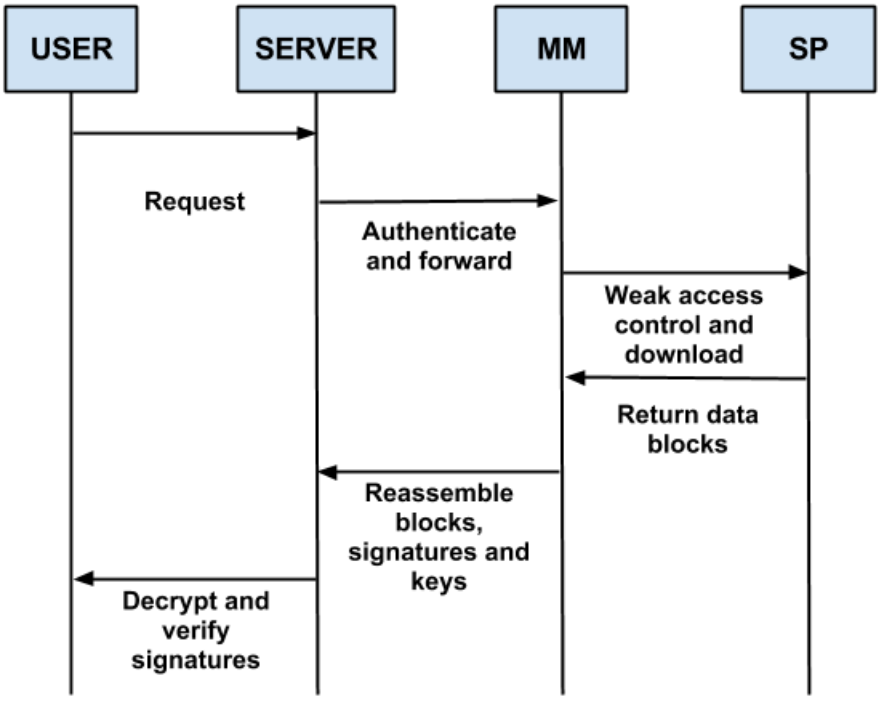

FIGURE 4.3: Retrieval Protocol

USER User U_j sends a retrieval request to the gateway in order to retrieve file F1. The request is composed by the user's id ID_{U_j}, the file identifier F_{id1} and his certificate.

GATEWAY The gateway receives the request, authenticates U_j and if the authentication does not fail, the gateway forwards the request to MM without performing any encryption.

MM MM receives the request from the server and analyzes it in order to check if U_j is authorized to access F_{id1} (U_j is the owner of the file). If the user is authorized, MM looks up the file identifier in the file table in order to get the pointer to the first block of the file. Then, MM visits the linked list in order to retrieve all the blocks that compose the file. For each of these blocks, MM retrieves the pointer from the pointer table and sends a request to SP.

SP SP returns the content of the encrypted blocks to MM. $B_i'' = E_{K_A}(E_{K_i}(B_i))$.

MM MM builds a response which contains all the blocks, keys and signatures of file F1. Signatures are retrieved from the signature table. The response is structured as follows:

- file identifier: F_{id1};
- first data block : $E_{K_A}(E_{K_1}(B_1))$;
- for each following data block $B_i (i \geq 2)$: key to decrypt block B_i, that is $E_{K_A}(E_{K_{U_j}}(E_{K_{i-1}}(K_i)))$; signature of block B_i, that is $E_{K_A}(S_i)$; data block B_i'' : $E_{K_A}(E_{K_i}(B_i))$;

MM sends the response to the gateway.

GATEWAY The gateway decrypts blocks, signatures and keys with K_A. If the signature verification does not fail, the gateway sends a response to U_j. Each key-block pair received by the user, will be structured as follows: $\{E_{K_{U_j}}(E_{K_{i-1}}(K_i)), E_{K_i}(B_i)\}$.

USER U_j can finally decrypt blocks and keys. U_j already knows the key corresponding to the block B_1. For each data block B_i, U_j decrypts block B_i' using K_i and K_{i+1} using K_{U_j} and K_i. U_j can finally rebuild the original file F1.

4.5 Prototype Implementation

This section gives a comprehensive overview of the prototype architecture and presents the implementation choices that were made for each component. Moreover, we describe the technical challenges we faced and explain the solutions we adopted and their corresponding motivations. Also, we point out the current known limitations of the prototype and discuss possible solutions to overcome them in the future.

The prototype aims at providing a comprehensive block-level deduplication system respecting the design goals and the specifications introduced in the previous sections. The prototype consists of three components: the client, the gateway and the metadata manager. Intentionally, the prototype does not include all features described in this chapter. In fact, we focused on those features that we considered crucial and necessary for a cloud storage system, leaving the rest as future work. The implementation language is Python.

4.5.1 Client

The most important software component of the client is a Windows process taking care of the synchronization of a local folder, a functionality similar to the one provided by the Dropbox desktop application [32]. Synchronization stands for the background and "real-time" replication process of all files in the local folder towards the Cloud Storage provider. More precisely, in order to optimize the synchronization process and not to replicate temporary changes to the files, a change is reflected to the remote side after a given period of inactivity, which we set to 2 seconds. The remote side also internally reflects the file system structure of each synchronization folder, so that upon a download operation the files are placed in the correct folders. To improve user experience, all files in the synchronization folder are decorated with an overlay icon representing the current synchronization status of a given file. For example, when a new file is added to the folder, it is decorated with a blue synchronization icon. After the background upload of the new file is complete, the icon is changed to a green ok sign. Interestingly, the actual operation that occurs upon file modification is that the entire file is uploaded to the MM, but most of the blocks are deduplicated, resulting in the generation of a very low amount traffic towards the cloud.

The client also keeps a local plaintext copy of all files that is accessible by the current user at any time. This may be seen as a security risk in the sense that anyone that potentially has access to someone else's computer, may easily have access to the files. On the other hand, not storing a local copy requires the system to download and decrypt files on the fly upon access. In systems intended to provide access to documents and other kinds of personal files, usability is usually considered as a higher priority goal. As a result, the files are stored in the synchronization folder in clear and made available to the user without any particular constraint.

4.5.2 Gateway

The gateway has been implemented as a plug-in for the Squid open-source web proxy. In practice, the gateway is a reverse proxy which works as an intermediate entity between clients and the Metadata Manager, so that users only have to communicate with the local and unique gateway which takes care of handling the communication with the remote components, namely the MM. Thanks to this setup, users can enjoy the functionalities provided by the system without having to deal with the complexity of the architecture. The role of the gateway is to receive requests from clients and forwarding them to the Metadata Manager after encrypting users' data when necessary. Similarly, the gateway is responsible for receiving responses from the Metadata Manager and forwarding them to clients after decrypting users' data. In order to assure data confidentiality, the gateway keeps one secret key, which is randomly generated and securely stored upon the first execution of the gateway.

4.5.3 Metadata Manager

The Metadata Manager runs a web service offering storage functionalities to clients. The Metadata Manager receives the encrypted blocks and the corresponding keys from the gateway and uploads only unique blocks into the cloud storage. Internally, the Metadata Manager consists of three separate entities. To communicate with the gateway, the metadata manager runs a Tornado web server [65]. The core application that performs deduplication is based on REDIS [9], which is

Symbol	Dataset index	Key structure	Value structure
F	0	F:user_id:file_id *	{'name':fname} *
K	0	K:file_id *	[key1, key2,...]
FB	0	FB:file_id *	[b_id0,b_id1,...]
B	0	B:block_id	{'storage_container':p.container.name 'storage-object':pointer.name 'counter':number_of_owners}
U	1	U:uid	{'password':pwd}
U	1	U:uid:session_key	session_key with expiration

Table 4.1: Metadata structure in REDIS

an in-memory key-value store providing excellent performance along with flexibility and robustness. To communicate with different cloud storage providers, the Metadata Manager relies on an internal component based on Libcloud [66], which is a multi-cloud open-source library.

The structure of the REDIS database is presented in Table 4.1.

For each file, the filename (that is the full file path, relative to the path of the local synchronization folder), the block keys and block identifiers are stored. The filename contains the path information so that the file system structure can be safely constructed at the client side. The block identifiers are 256-bit hashes of the blocks and are stored in order to know which blocks belong to the file. For each block, the storage information is stored so that the block can be retrieved from the cloud. A block is considered as a duplicate, hence not stored, if it has an identifier that already exists in the system. In that case the reference counter, which counts number of occurrences of the block across all files in the system, is increased by one. Correspondingly, when a block is deleted from a file, the corresponding counter is decreased, and the block is actually removed from the cloud only if the counter value reaches 0.

4.5.4 Access Control

The previously introduced security features provide confidentiality and deduplication by design. In other words, they prevent anyone other than users, including a curious Cloud Storage provider, from accessing the files in the system by decrypting them.

This section presents how access control has been enforced thanks to the introduction of authentication and authorization features, like credential-based authentication. As opposed to data confidentiality by encryption, access control mechanisms aim at verifying the identity of a user (who the user is) in order to determine

whether he is authorized to access a given resource, such as a file or an encryption key. ClouDedup's scheme proposes a particular and original authentication method, which consists of appending signatures to each block and checking them upon data retrieval. According to the design, this operation is performed by the gateway in order to verify ownership. However, for the sake of simplicity, we decided not to implement this feature in the first version of the prototype. Indeed, as long as the Metadata Manager correctly implements its access control mechanism, this feature is not crucial.

Considering the prototype, the goal was to build a robust yet efficient access control mechanism without negatively affecting performance or restricting the set of desired features.

4.5.4.1 Client Access Control

The synchronization folder client only allows a single user at a time. It is not possible for multiple users to operate on the same desktop client. The access control is enforced by encrypting the block encryption keys with the user's secret key, which is then protected by symmetrically encrypting it using the hash of the user's password as a key. The hash is securely computed by using the well-known SCRYPT key derivation function [67], which has been designed to be resistant against brute-force attacks. This way, traditional brute-force attacks aimed at obtaining the value of the secret key by simply trying all possible values of user's password will not be feasible.

4.5.4.2 Gateway Access Control

The prototype provides no built-in authentication between the clients and the gateway. However, if the component is located in a company's premises, it can be easily configured to accept requests issued only from the local area network. Furthermore, SQUID already provides an extensive set of authentication modules supporting the most common interfaces. Therefore, the organization can easily configure the authentication at the level of the gateway to reflect their policies by properly tuning SQUID configuration.

4.5.4.3 Metadata Manager Access Control

The Metadata Manager implements a simple yet robust credential-based authentication. Upon user registration, the hash of the new user's password is stored among the metadata. When a user logs in from his device, an authentication request is made by submitting the username and the hash of the password computed with SCRYPT. As a response, the user receives a session key (authentication token) which expires after a pre-defined period of time. The Metadata Manager keeps track of active session keys in order to validate any user request: if the session key provided by the user is wrong or expired, the request is not performed and an authentication error is returned, inviting the user to re-enter his credentials.

4.5.5 Prototype Credential Management

This section presents the solutions concerning the secure management of the secret keys and the credentials of the users stored at client-side.

4.5.5.1 Key Management

A recommended and widely accepted practice for key management consists of delegating this task to external and independent components. In practice, this implies that users, or an organization on their behalf, should take care of safely and securely storing their keys and making them available when necessary. In real scenarios, such a solution is quite common since IT companies are already used to putting in place centralized systems for storing passwords, hard-drive encryption keys and other sensitive information. Hence, integrating the backup of the user's secret key with the existing systems seems to be a reasonable solution that presents the desirable advantage of being compliant with organization's policies. Furthermore, in the case of ClouDedup, it is crucial to make sure that the backup system with which encryption keys are stored is reliable. Indeed, accidentally losing the secret key would mean for all users to definitely lost their files too, without any chance of getting them back.

4.5.5.2 Credentials and Key Rotation

The prototype allows the user to change his password from the client. In order to do so, the user selects a new password, and sends a request containing both the old and the new secure password hashes to the metadata manager. If the request is performed successfully by the Metadata Manager, the secret key is re-encrypted with the new password and rewritten on disk. In addition to that, the prototype allows the administrator of the Metadata Manager to force a user to change the password. If password change is forced, the user is prompted for a new password as a response to an authentication request. Such a mechanism is required in order to be compliant with the security policies of most of the companies, which usually encourage users to often change their password (e.g. every 6 months).

Regarding the secret key rotation, the first version of the prototype does not handle this directly. Rather, this task is delegated to users. In order to change their secret key, users are required to remove their files from the synchronization folder, generate a new key and put the files back. The performance cost of lacking a specific key rotation feature is that the users need to re-upload also the data, instead of just uploading the re-encrypted block keys. However, such a feature would be beneficial from the security point of view, therefore it has been planned as near future work.

4.5.6 Technical Challenges

4.5.6.1 Fast Upload of Large Files

When dealing with cloud storage solutions, depending on the use-case, one of the most important requirements from a user standpoint is great performance. In practice, a system is considered to perform well when users experience low latencies when uploading or downloading very large files, even simultaneously. In a first proof-of-concept implementation, our prototype suffered from high latencies, especially when uploading very large files (tens of MBs). This issue was caused by the way the upload of blocks was handled: instead of processing a data block as soon as it was received, the Metadata Manager waited for the whole request to be received before processing any data block. This suboptimal behavior was due to the default behavior of the HTTP servers: indeed, by default a request can

be handled only when it has been fully received. However, the HTTP protocol optionally provides the possibility of treating the request as a set of separate blocks, called "chunks", which can be processed separately. We enabled this option in our system in order to allow both the Gateway and the Metadata Manager to process data blocks individually in a streaming fashion and thus greatly improve the overall performance.

4.5.6.2 Disaster Recovery

In order to preserve the integrity of users' data, in ClouDedup it is fundamental to safely protect metadata stored at the Metadata Manager. Without these information, it would be impossible to recover users' files from the encrypted data blocks stored at the Cloud Storage Provider. Therefore, a mechanism aimed at protecting the Metadata Manager against any potential event that may cause the loss of metadata and all users' data is needed. Such a mechanism should also guarantee the availability of the Metadata Manager service, meaning that in case of an unfortunate event such as a crash, users should not hopefully notice any downtime.

The solution we adopted makes use of the built-in master-slaves architecture available in REDIS. Thanks to this architecture, the main Metadata Manager (the master) is replicated in real-time over one or more replicas (the slaves) which store an exact copy of the same database. If for any reason the current master becomes unavailable (e.g. because of a crash or a network problem), all sentinels, that are special processes running on all replicas, start a "voting" process aimed at electing the new master. As soon as the majority quorum is reached, a new master is elected and the Metadata Manager service is fully functioning again. This process happens very quickly, namely in a couple of seconds at most, hence users are unlikely to notice any downtime. In the unlikely case a user sends a request during the downtime period, the request will be rejected and automatically re-issued after a few seconds, meaning that no data will be lost and the impact on the quality of service will be minimal. In order to make the Gateway aware of the new master, an additional sentinel process runs on the Gateway as well, so that it can be notified as soon as the address of the current Metadata Manager changes. This notification is necessary due to the fact that the Gateway operates

as a reverse proxy between users and the Metadata Manager, therefore the Gateway needs to be aware of the identity (IP address) of the current master in order to correctly forward requests and responses.

Of course, this solution incurs a non-negligible network overhead due to the need of communicating all metadata changes to each replica. However, we point out that metadata do not contain raw data blocks but only IDs and encrypted keys, which take a few KBs per file. Also, we assume that the link between the master and its slaves is fast and reliable enough to allow for instantaneous metadata replication.

4.5.6.3 Upload Buffer

This section presents a solution that was added to reduce the latency of upload requests experienced by clients when the metadata manager's upload link to the cloud is the throughput bottleneck. The decrease in latency is achieved by storing the received data blocks in a local buffer stored at the Metadata Manager and sending a response immediately after all data blocks have been received, as opposed to responding after all data blocks have been uploaded to the Cloud Storage Provider. Sending the confirmation to the client before all data blocks have finished uploading leaves a possibility of losing some blocks in the unfortunate case the Metadata Manager stops functioning (e.g. crash) before the buffer is flushed. Different approaches to buffer implementation are presented, along with corresponding pros and cons in terms of disaster recovery. Finally, a known security weakness of the buffer solution is discussed.

The upload link from Metadata Manager to the Cloud Storage Provider is the throughput bottleneck of the system, so if the blocks are buffered and uploaded in the background, it is possible to respond to the client faster than if one would have to wait until every block has been uploaded. From a security standpoint, as an additional bonus, such operation would make the upload response time depend on the data buffering time instead of the actual upload time, which would remove the client's ability to detect deduplication by timing upload requests. Furthermore, in case of an unavailability period of the Cloud Storage Provider, the buffer could be used a temporary storage replacement, within its storage limits. This would result in higher availability, but only until the buffer becomes full. However, buffering the blocks introduces a replication problem in case the buffer is a local database, or a memory problem, if the blocks are kept in memory.

Considering the architecture of ClouDedup, the most straightforward solution would be to buffer the blocks in REDIS, meaning that blocks will be stored in memory. This has the advantage of being disaster-safe without any additional implementation effort, as REDIS already provides a master-slave replication mechanism, as explained above. However, storing the blocks in memory, especially in multiple replicas, is memory-expensive and incurs a significant network overhead. Indeed, depending on the size of the buffer, it would require potentially gigabytes of additional memory in every replica. Also, the current master would need to send blocks to every replica, in addition to the Cloud Storage Provider.

An alternative solution consists of storing blocks in a local database. This way, the blocks are stored on disk instead of memory, which guarantees their persistence and protection against crashes. Of course, such a solution requires the disk to be faster than the Metadata Manager's upload link, which is a realistic assumption. Otherwise the disk may become the bottleneck of the system. Besides providing the ability to preserve data blocks upon a crash, the local database solution introduces a problem with respect to disaster recovery: if the upload confirmation has been sent to the user and then the entire machine is destroyed, the blocks in the local database will be lost, while the user will assume that his file is safely stored in the system. The most straightforward way of addressing the disaster recovery problem is to replicate the local database to the slaves. Despite the fact that the cost of this operation is actually the same than the cost of uploading the data directly to the cloud, the solution is viable if the master has a faster connection to the slaves than to the Cloud Storage Provider. Obviously, uploading the database to the slaves is not a good option if it is not much more efficient than uploading the blocks directly to the Cloud Storage Provider. A possible way of decreasing the cost of the local database replication would consist of putting a place a mechanism to incrementally update the database on all replicas. However, this optimization would still require the master to send new blocks to all replicas.

In the current implementation of ClouDedup, the Metadata Manager uses a local SQLITE database. In order to address the replication problem, the Metadata Manager may replicate the database to a slave in the same network, and then send the positive response to the client. As a result, the system would have been disaster-safe, but the response time would have been extended by the replication time. Currently, we decided not to replicate the database. The reason for this decision is that losing the blocks stored in the buffer is in our case very unlikely

to happen, as the synchronization folder at the client always keeps a local copy of users' files. In the event of Metadata Manager crashes or is destroyed, one of the slaves takes over. The new master will not be aware of the pending files in the previous master's local buffer, however this is not an issue, as the client will start a synchronization process with the new master. The synchronization process will make sure that any files that are only present in the client synchronization folder will be pushed to the Metadata Manager and then to the Cloud Storage Provider. In other words, user's files are only lost if the client's computer and the Metadata Manager are both simultaneously destroyed. We considered that risk very unlikely to happen, but moderately severe. As a final result, we decided that the reductions in internal network load and user-visible response latency are worth taking the risk, and did not replicate the local database. As a result, the system imposes only very low latency to the users as long as the buffer is not full. Also, the solution allows the maximum buffer size to be very large, as it is only bound by the available disk space on the Metadata Manager machine. In addition to that, this solution makes the system resistant to timing-based deduplication detection at the clients, denying them the possibility to perform COF attacks.

In a buffered system, a file upload latency remains low as long as the buffer does not become full. This attribute can be exploited with a certain accuracy in order to perform a timing-based attack aimed at detecting whether a file has been stored. In order to do so, an adversary is required to measure the time a packet spends in the buffer. This information may be obtained by uploading data to the Metadata Manager and measuring the response time when the buffer becomes full. Once the adversary knows how long a packet stays in the buffer, he can detect duplicates by measuring the difference between response time and the time spent in the buffer. Depending on the size of the buffer and the variance of the upload rate of the metadata manager upload link, the measurement may be fairly accurate. Protection against timing-based detection of deduplication in buffered systems can be improved by adding random variance to the upload rate of the metadata manager. Such addition would randomize the time the packet spends in the buffer. However, this may not be sufficient as the adversary might succeed by repeating the experiments multiple times and measuring the average. In the prototype, this is a known weakness which has not been addressed yet.

4.6 Evaluation

In this section we show and discuss the results of the evaluation of the proposed scheme, from both the performance and security point of view. Regarding the performance, we analyze both the theoretical complexity of the proposed scheme and the actual performance of the implemented prototype. In particular, we focus on the analysis of the storage (upload) and retrieval (download) operations, which are the most common operations in cloud storage. Regarding the security, we consider all most likely scenarios and evaluate ClouDedup's resilience against potential attacks.

4.6.1 Complexity

We analyze the computational complexity of the two most important operations: storage and retrieval. N is the mean number of blocks per file and M the total number of blocks in the system.

	Storage	**Retrieval**
Encryption	$O(N)$	$O(N)$
Hash	$O(N)$	$O(N)$
Lookup in data structures	$O(N)$	$O(N)$
Other	$O(N)$	$O(N)$

4.6.1.1 Storage

The first step of the storage protocol requires the gateway to encrypt B_i, K_i and S_i. As the encryption is symmetric, the cost of each encryption can be considered constant, so for N blocks the total cost is $O(N)$. The second step of the protocol requires the metadata manager to hash each block in order to compare it with the ones already stored. As for symmetric encryption, the total cost is $O(N)$. In order to perform deduplication, MM has to check if a block has already been stored. In order to do so, he searches for a given hash in a hash table containing the hash of all stored blocks. As calculating the candidate position of a hash and checking whether it is stored can be done in constant time, the cost of this operation is constant, that is $O(1)$, and it is performed for each block. The cost of the update of the data structures can be considered constant. The last (optional) step of the

protocol is the additional encryption performed by the MM, which symmetrically encrypts at most N blocks. The total cost of the storage operation is linear for the encryption operations and almost linear for the lookup in data structures, therefore the metadata management is scalable.

4.6.1.2 Retrieval

The first step of the retrieval protocol requires the metadata manager to compute a hash of the concatenation of user id and file name. The cost of this operation can be considered constant. Even the lookup in the file data structure, in order to get the pointer to the first block of the file, has a constant cost. Visiting the linked list, searching in the data structures and sending a request to the cloud storage provider, have a constant cost and are repeated N times. Once again, the cost of the symmetric decryptions is constant, hence the complexity remains linear. The signature verification process requires the gateway to verify one signature and compute $N - 1$ hashes, hence the cost of this operation is linear. The total cost of the retrieval operation is linear, therefore the system is scalable for very large datasets.

4.6.2 Performance and Overhead

4.6.2.1 Throughput

In a cloud storage solution, from a performance standpoint the highest priority is to achieve a very high upload throughput. The objective in throughput optimization is that the actual throughput should be bounded only by the user's or the Metadata Manager's upload links. In other words, there should not be any software component causing a non-negligible decrease of the final throughput. As an example, an inefficient encryption mechanism at the client or the Gateway would negatively impact the overall performance and decrease the final throughput. Similarly, at the Metadata Manager the performance would not be optimal if the deduplication mechanism built on top of REDIS would incur a significant computational overhead, resulting in a decrease of the upload throughput. On the client-side, we can safely assume that unless the client has a very slow CPU and an exceptional upload bandwidth, the symmetric encryption and data-chunking operations are fast enough not to cause any bottleneck.

The theoretical throughput bottleneck for upload is the upload link capacity between the Metadata Manager and the Cloud Storage Provider. In order to prove that, we measured the actual time required to upload data blocks to the Cloud. As expected, the result was that the upload link is not only the main bottleneck, it also severely handicaps the upload performance. Indeed, the results showed that Libcloud, which is the library we used for handling the communication with Cloud Storage Providers APIs, incurs a significant overhead to the overall upload time of each file. The explanation of this issue is simple: when uploading single data blocks, Libcloud does not seem to re-use the same connection, hence establishing a new connection for the upload of every single block introduces the observed severe delay.

As a solution to this serious performance issue, we explored the possibility of taking advantage of parallelization. In order to do so, we conducted a series of experiments with simultaneous file upload requests to the Amazon S3 Cloud Storage Provider. The results of these experiments are shown in next section.

4.6.2.2 Libcloud Upload Performance

The efficiency of making parallel upload requests with Libcloud was measured by uploading a 1 MB file with different number of parallel requests. In the experiments, the file was uploaded when using a number of threads from 1 to 10. Each experiment was executed 10 times and the average block upload time and the total upload time were calculated. The final results were the average of those two values in the 10 different executions. Double standard deviation of the different round averages was used as error margin for both values. The machine used for the tests was an Amazon EC2 M3 (large) instance with 2 CPUs, 7.5 GB of RAM and a Gigabit upload link. Figure 4.4 presents the results of our experiments which show the efficiency of Libcloud when uploading a 1 MB file, split into 99 blocks.

Surprisingly, the results indicate that increasing the number of parallel block upload requests also increases the time that each of the requests takes to complete. This behavior is due to the internal implementation of Python, which does not handle native parallel threads. As a conclusion, increasing the number of threads beyond 5 does not improve performance significantly. To confirm this result, the same experiment was performed with 100 parallel threads. The total time measured was 3.53 seconds with an error margin of 0.23 seconds. Thus, the results

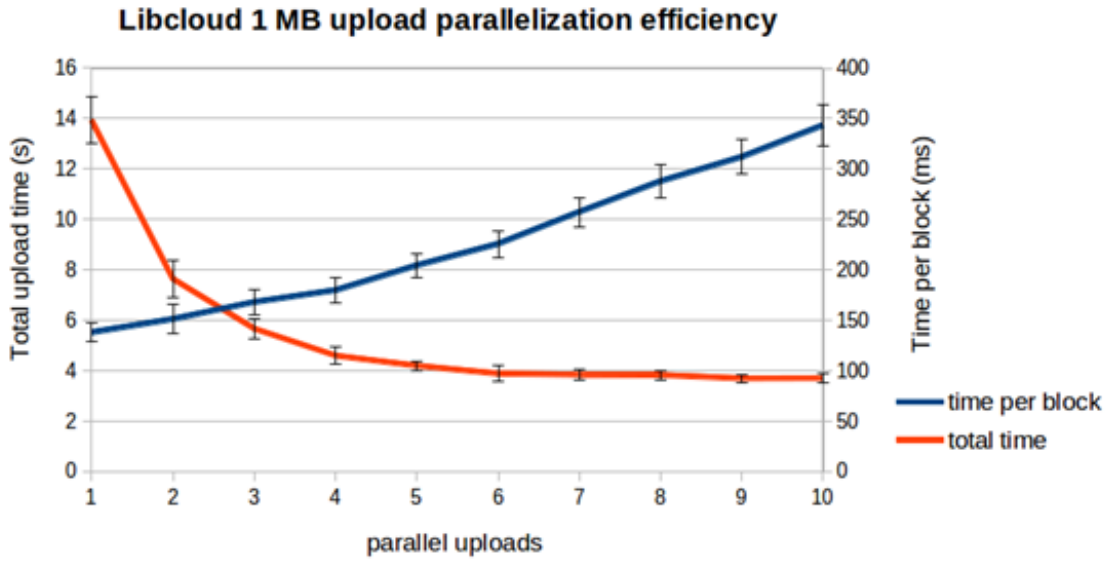

FIGURE 4.4: Results of Libcloud parallel uploads experiments

indicate that even though all data blocks of a given file are uploaded in parallel, the upload rate will be limited to 2.27 Mbps, that in this case is about 0.2% of the available bandwidth.

In order to overcome this bottleneck, several steps may be taken. In order to find out whether the cause is Libcloud or Amazon S3, an alternative interface can be built from scratch in order to directly communicate with the Amazon S3 API. Both are very efficient when uploading one single file, but when uploading many small blocks as different files the performance seems to be significantly lower. By implementing our own interface we could directly measure the performance when directly using the Amazon S3 API and other providers' APIs. Also, in order to get the most out of parallelization, a different language or framework may be used for creating parallel upload threads or processes. This way, the Metadata Manager may take advantage of truly parallel threads and maximize the bandwidth usage. If none of these approaches succeeds, the ultimate solution would be to make use of a local Cloud Storage Provider such as a local OpenStack SWIFT appliance.

4.6.2.3 Upload Throughput vs File Size

Ideally, an optimal cloud storage solution should not let the file size affect the throughput of the system. In other words, the upload time should grow linearly

with respect to the size of the file being uploaded. In order to verify whether our current implementation of ClouDedup is compliant with this principle, upload, download and deduplication times were measured for 1, 5, 10, 20 and 50 MB files. In particular, the deduplication time is the time required to upload a file that already exists in the system since it has already been uploaded in the past. Of course, the deduplication time is expected to much lower than the upload time. During our experiments, the client machine was a Amazon EC2 M3 medium instance. The Metadata Manager machine was the same as in the upload parallelization experiments. Each experiment was executed 10 times, and average upload, download and deduplication times were measured. The error margins, counted as double standard deviations of the values during the 10 executions, are not presented, as they were all less than a second. In terms of parallelization the system was configured to 100 simultaneous block uploads and downloads. The results are presented in Figure 4.5.

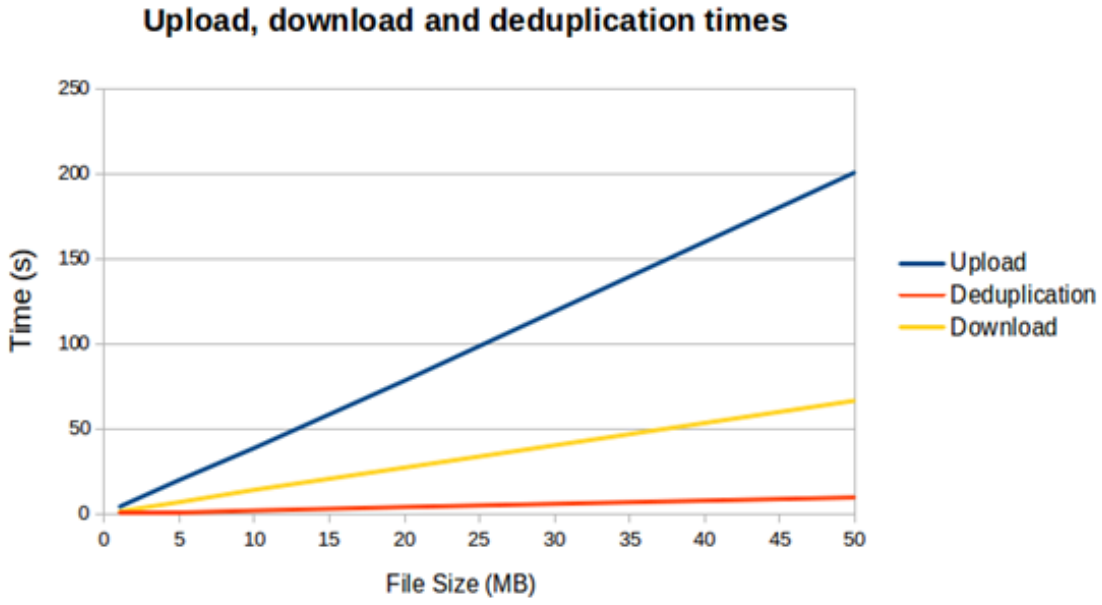

FIGURE 4.5: Results of Upload Throughput vs File Size experiments

The results confirm that the upload, download and deduplication times increase linearly as file size increases. In other words, the transfer rate is independent of the file size. For the 50 MB file, the upload, download and deduplication rates were 2.0 Mbps, 43.2 Mbps and 6.2 Mbps respectively. These values define the system performance with sufficient accuracy.

4.6.2.4 Upload Buffer vs Response Time

As described in the implementation of the prototype, the upload latency visible by the client can be greatly decreased by putting in place a buffering solution at the Metadata Manager. In practice, when a client uploads a file, a positive acknowledgment will be returned to the client as soon as all data blocks have been successfully received and stored in the buffer. If the buffer is temporarily full or there is not enough space to store all data blocks, the client has to wait until enough blocks are uploaded to the Cloud Storage Provider. To confirm that the response time decreases as a function of the proportion of the file that immediately fits in the buffer, experiments were conducted. The upload response time of a 1 MB file was measured with different buffer sizes. The buffer size is presented as the percentage of the file that fits into the buffer. For client and the Metadata Manager, the same machines were used as in the previous experiments. Parallelization was configured to allow for the execution of at most 10 simultaneous threads, which corresponds to about 10% of the file being simultaneously uploaded, as the total number of blocks in the file was 99. The experiments were executed 10 times and the values presented in this section are the averages of the response times in different executions. The error margins are doubled standard deviations between the values of different executions. Figure 4.6 presents the results.

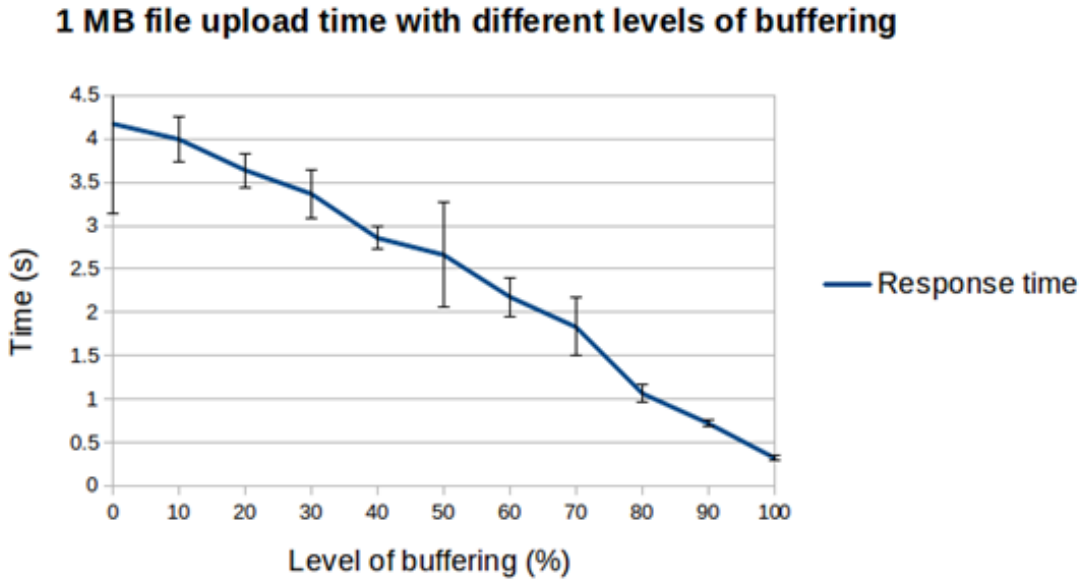

FIGURE 4.6: Results of Upload Buffer vs Response Time experiments

As expected, the results indicate that the response time indeed decreases linearly as the level of space in the buffer increases. The average response time for 0% buffering (no buffering) was 4.2 s and for 100% buffering 0.3 s. The times correspond to the upload rates of 1.9 Mbps without buffering and 24.7 Mbps with complete buffering. The round trip time between the client and the Metadata Manager was ignored in the rate calculations, as it was less than 20 ms.

4.6.2.5 Network Overhead

The network overhead caused by the system was measured by monitoring the amount of data transferred between the client and the Metadata Manager, and dividing that amount by the actual data that needed to be uploaded to the Cloud. In this case the actual data means the encrypted data blocks, including padding due to symmetric encryption, but excluding encoding overhead and all the metadata such as encrypted symmetric keys. As shown in the results, most of the network overhead is due to base64 encoding of the encrypted binary data, which cannot be transferred in raw format over HTTP(S). As the base64 algorithm encodes every 3 characters as 4, and occasionally applies one or two characters of padding, the overhead for an encrypted block is between 33.3% and 33.4%. The overhead is slightly increased by the transferred encrypted filename and the encrypted block keys, which are also base64-encoded. The impact of filename is negligible, as it takes only a few bytes and is transferred only once per file. The encrypted block keys, however, impose an overhead of 64 bytes per block. As the mean block size for the rabin fingerprint data-chunking method has been configured to 8 KB, the mean overhead caused by the encrypted block keys is about 0.8%. Moreover, a fraction of the overhead depends on the fact that we decided to pack every encrypted block along with its corresponding attributes (e.g. encrypted key) into a JSON object, which is the chunk that is actually sent to the Metadata Manager. However, JSON proved to be very space efficient, therefore its overall impact on the network overhead is negligible. The network overhead between the client and the Metadata Manager was measured for three files of 10 KB, 1 MB and 5 MB respectively. The results are presented in Figure 4.7.

The total overhead was between 33.4% and 33.6% for all the files. As expected, this confirms that unless the file is extremely small, the amount of overhead is linear with respect to the file size. Also, we point out that since the main source

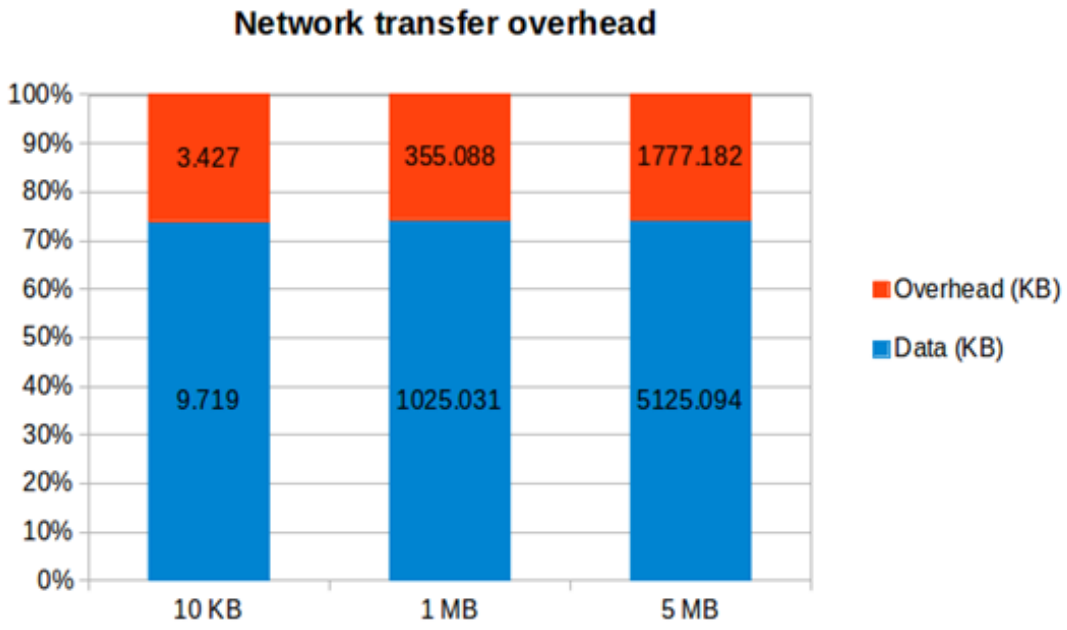

FIGURE 4.7: Results of the Network Overhead Experiments

of overhead is the method used for encoding the encrypted binary data, we plan to investigate the possibility of using a more efficient encoding method such as Z85 which introduces an overhead of 25%, meaning that it would reduce the network overhead by more than 8%. In addition to that, we point out that for the sake of simplicity and to speed-up the development we decided to use HTTP(S) as protocol, which introduces a space efficiency problem due to the nature of HTTP itself. Indeed, HTTP was designed as a text protocol, meaning that even though it can be used to send binary files, a binary protocol would always be way more space efficient than HTTP. Therefore, we also plan to investigate the possibility of using a more efficient protocol, namely a binary protocol, for uploading encrypted data blocks. Also, as we mentioned earlier, we used JSON as data serialization format in order to pack encoded encrypted blocks and related attributes into single textual chunks. Although JSON is a de-facto standard and known to be very efficient, there are a few alternatives such as MessagePack [68] that may be even more efficient. Finally, since encrypted data blocks are encoded into long ASCII strings, we plan to test the effect of gzip compression in order to further reduce the size of upload and download requests and responses.

4.6.2.6 Metadata Storage Overhead

The metadata storage overhead consists of the amount of memory required to store all metadata, which consist of the information needed to keep track of all blocks, files and users. This quantity was measured by monitoring the memory usage of REDIS, which is the software component used for storing all metadata, with different amounts of data in the system. The results are presented in Figure 4.8.

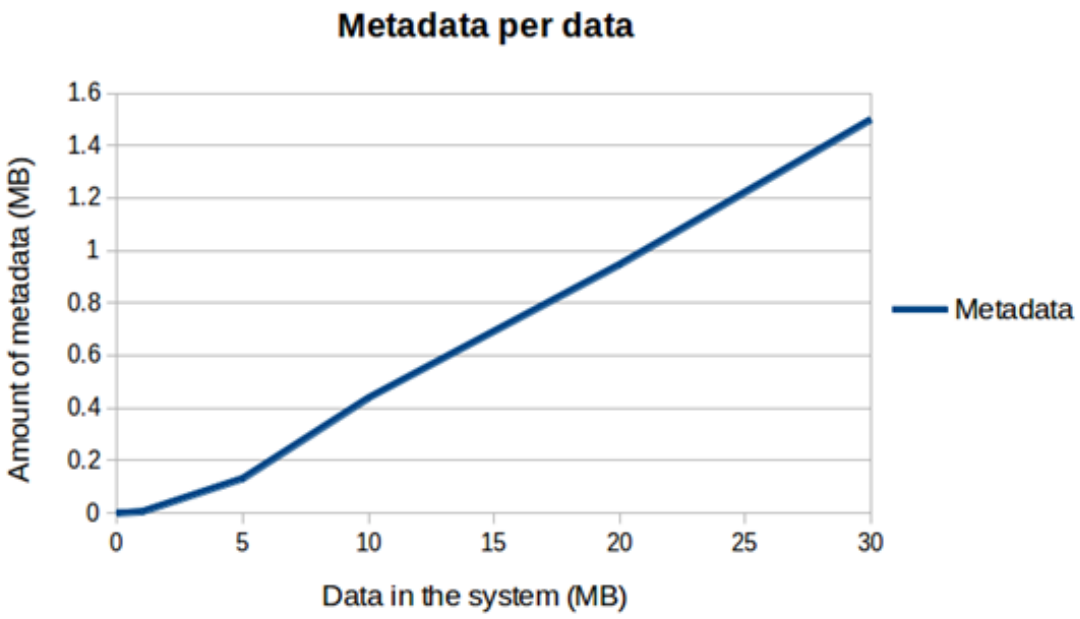

FIGURE 4.8: Metadata Storage Overhead

The results clearly indicate that the memory required for metadata is about 5% of the data stored in the system, which can be considered as a reasonable and affordable overhead. However, we remind that our implementation is not yet production ready, therefore we investigated the possibility of further reducing the metadata footprint by removing redundant information.

We started from the observation that the Metadata Manager currently requires more memory than what would be needed to store the encrypted block keys. Therefore, this paves the way to further optimizing the memory usage at the Metadata Manager.

Key	**Value**
F:file_id	'user_id':user_id,'filename':filename,'block_keys': [k0, k1,...]
B_block_id	'storage_info':block_storage_information,'owner_counter': number_of_owners

TABLE 4.2: A memory optimized metadata structure

As shown in Table 4.1, the current implementation stores the metadata as partially redundant key-value entries in REDIS. In this context, partially redundant means that some pieces of information appear multiple times either as a key or as a value. Ideally, a fully optimized solution would contain each unique piece of information only once, either as a key, as a value or as a part of a value. However, as a block may appear in multiple files, a distinction between per-block and per-file information is necessary. A memory optimized metadata structure is presented in Table 4.2.

Unfortunately, REDIS does not allow complex types like lists as values of a hash table. A possible workaround to this limitation would consist of serializing complex values that cannot be directly handled by REDIS and store them as simple strings, which would bring the additional overhead of deserializing these complex values before accessing them, resulting in an increase of the access time. However, this overhead would not seriously impact the performance of the system since some entries are only searched when a client is downloading a file. As the download occurs rarely, the priority of the access time is low for certain entries, like the encrypted block keys or the cloud storage coordinates.

As a more complex alternative, the memory consumption at the metadata manager could be further reduced by encrypting all the metadata that do not need to be accessed often and outsourcing it to the cloud. As an example, the encrypted block keys are data bound, as they need to be accessed only upon a download or a file update. More precisely, they are subject to change only when a file modification or deletion occurs. Therefore, encrypted block keys could be safely outsourced to the Cloud Storage Provider. Also, from a security standpoint, this solution would not introduce any additional threat with compared to the current solution, considering that the encrypted block keys would still be protected against a malicious Cloud Storage Provider unless the Gateway is compromised. In order to store the encrypted block keys in the cloud, each file entry would require to contain the necessary storage information to retrieve them. However, the storage info is of constant size, as opposed to a block keys that depend on the number of blocks in the file. Also, if the block keys are not further encrypted by the Metadata

Manager, or if the Metadata Manager already performs an extra encryption for data blocks, this operation does not impose any additional cost with respect to key management. Otherwise the additional cost would be to store one symmetric encryption key at the Metadata Manager.

4.6.2.7 Data Storage Overhead

In this section we discuss the storage overhead caused by ClouDedup at the Cloud Storage Provider. We remind that in our current implementation the only data that is stored at the Cloud Storage Provider is the encrypted blocks, which are about the same size as the original unencrypted files. Indeed, the base64 encoding method is only applied when transferring data from one component to another. More precisely, the only overhead introduced at this point is due to symmetric encryption padding. However, even in the worst case the client applies a padding of 31 bytes which, assuming a mean block size of 8KB, is about the 0.4% of the original data.

4.6.3 Deduplication Rate

Our proposed solution aims at providing a robust security layer which provides confidentiality and privacy without impacting the underlying deduplication technique. Each file is split into blocks by the client, who applies the best possible chunking algorithm. When encrypted data blocks are received by the MM, a hash of each block is calculated in order to compare them to the ones already stored. This task is completely independent from the chunking technique used by clients. Also, all the encryptions performed in the system do not affect the deduplication effectiveness since the encryption is fully deterministic. Therefore, ClouDedup provides additional security properties without having an impact on the deduplication rate. Users can thus benefit from the additional security provided by the system without affecting the deduplication rate.

4.6.4 Security

We explained the main security benefits of our solution in section 4.2.4. We now focus on potential attack scenarios and possible issues that might arise.

Curious Cloud Storage Provider As stated in the threat model section, we assume that an attacker, like the malicious storage provider, has full access to the storage. If the attacker has only access to the storage, he cannot get any information. Indeed, files are split into blocks and each block is first encrypted with convergent encryption and then further encrypted with one or more secret keys using a deterministic encryption mechanism. As discussed in Chapter 3, deterministic encryption can effectively provide full confidentiality. Moreover, no metadata (file owner, file name, file size, etc.) are stored at the cloud storage provider. Clearly, thanks to this setup, the attacker cannot perform any dictionary attack on predictable files.

Compromised Metadata Manager A worse scenario is the one in which the attacker manages to compromise the metadata manager and thus has access to data, metadata and encrypted keys. In this case, confidentiality and privacy would still be guaranteed since block keys are encrypted with users' secret keys and the gateway's secret key. The only information the attacker can get are data similarity and relationships between files, users and blocks. However, as file names are encrypted by users, these information would be of no use for the attacker, unless he manages to find a correspondence with a predictable file according to its size and popularity. Also, as discussed in Chapter 3, deterministic encryption assures confidentiality even when used in conjunction with block-level deduplication. Indeed, ciphertext-only attacks based on the analysis of block frequency do not seem to be feasible in real scenarios.

Compromised Gateway The system must guarantee confidentiality and privacy even in the unlikely event where the gateway is compromised. An additional encryption performed by the metadata manager before sending data to the storage provider will then enforce data protection since it also offers another encryption layer; therefore confidentiality is still guaranteed and offline dictionary attacks are not possible. On the other hand, if the attacker compromises the gateway, only online attacks would be possible since this component directly communicates with users. The effect of such a breach is limited since data uploaded by users are encrypted with convergent encryption, which achieves confidentiality for unpredictable files [5]. Furthermore, a rate limiting strategy put in place by the metadata manager can limit online brute-force attacks performed by the gateway.

Compromised Gateway and Metadata Manager In the worst scenario, the attacker manages to obtain all secret keys by compromising both the gateway

and the metadata manager. In this case, the attacker will be able to remove the two additional layers of encryption and perform offline dictionary attacks on predictable files. However, since data are encrypted with convergent encryption by users, confidentiality for unpredictable files is still guaranteed.

Malicious Users colluding with Metadata Manager Another interesting scenario that we discuss is the case in which one or more users collude with the metadata manager in order to circumvent the gateway and compromise confidentiality. In such a scenario a dictionary attack would work as follows: the malicious user generates a plaintext, encrypts it with convergent encryption and uploads it as usual. The gateway receives the upload request and encrypts all data blocks with its secret key. At this point, the metadata manager can easily check whether the file has already been stored, as it would do for deduplication, and send a feedback to the user. If the file exists, then the user knows that a given file has already been uploaded by another user. Such a simple attack may prove to be extremely effective since it may be used for discovering confidential information such as the pin code in a letter from a bank or a password in an email. However, we argue that this kind of attacks would not have a severe impact in real scenarios, the reason is twofold. First, such an attack would be perpetrated online, which means that the attack rate would be limited by the upload capacity of the user. Moreover, similarly to the metadata manager, the gateway may easily prevent these attack by putting in place a rate limiting strategy. Second, in a real scenario all users would belong to the same organization and be authenticated using an internal and trusted strong authentication mechanism. In addition to that, due to the high number of upload requests, such an attack would likely leave traces and be detected promptly, therefore a potentially malicious user would be strongly discouraged.

External Attacker Finally, we analyze the impact of an attacker who attempts to compromise users and have no access to the storage. If an attacker only compromises one or more users, he can attempt to perform online dictionary attacks. As the gateway and the metadata manager are not compromised, the attacker will only retrieve data belonging to the compromised user thanks to the access control mechanism. Furthermore, as mentioned above, the gateway can limit such attacks by setting a maximum threshold for the rate with which users can send requests.

Chapter 5

PerfectDedup

5.1 Introduction

ClouDedup achieves secure block-level deduplication at the cost of requiring a complex architecture where the most crucial encryption operation is delegated to a trusted component. Also, as discussed in the security analysis section, a Metadata Manager colluding with one or more users may easily circumvent the protection guaranteed by the additional encryption layer and successfully perform COF and LRI attacks.

Starting from these two drawbacks, we aim at designing a scheme, called PerfectDedup, with a simpler architecture where users could autonomously assess whether a data block can be deduplicated by running a privacy-preserving protocol with an untrusted Cloud Storage Provider. Such an approach would have the additional and non-negligible benefit of allowing for client-side (source-based) deduplication, which brings bandwidth savings in addition to storage space savings. Thanks to the privacy-preserving protocol, PerfectDedup securely and efficiently combines client-side cross-user block-level deduplication and confidentiality against potentially malicious (curious) cloud storage providers without relying on a trusted entity with respect to the encryption operation. Unlike ClouDedup, this scheme also allows for client-side deduplication, meaning that a client can securely check whether a block is a duplicate before uploading and encrypting it.

Data Popularity In PerfectDedup, we propose to counter the weaknesses due to convergent encryption by taking into account the popularity [7] of the data

segments. Data segments stored by several users, that is, popular ones, are only protected under the weak CE mechanism whereas unpopular data segments that are unique in storage are protected under semantically-secure encryption. This declination of encryption mechanisms lends itself perfectly to efficient deduplication since popular data segments that are encrypted under CE are also the ones that need to be deduplicated. This scheme also assures proper security of stored data since sensitive thus unpopular data segments enjoy the strong protection thanks to the semantically-secure encryption whereas the popular data segments do not actually suffer from the weaknesses of CE since the former are much less sensitive because they are shared by several users. Nevertheless, this approach raises a new challenge: the users need to decide about the popularity of each data segment before storing it and the mechanism through which the decision is taken paves the way for a series of exposures very similar to the ones with CE. The focus of schemes based on popularity then becomes the design of a secure mechanism to determine the popularity of data segments.

We suggest a new scheme for the secure deduplication of encrypted data, based on the aforementioned popularity principle. The main building block of this scheme is an original mechanism for detecting the popularity of data segments in a perfectly secure way. Users can lookup for data segments in a list of popular segments stored by the Cloud Storage Provider (CSP) based on data segment identifiers computed with a Perfect Hash Function (PHF). Thanks to this technique, there is no information leakage about unpopular data segments and popular data segments are very efficiently identified. Based on this new popularity detection technique, our scheme achieves deduplication of encrypted data at block level in a perfectly secure manner.

The advantages of our scheme can be summarized as follows:

- PerfectDedup allows for storage size reduction by deduplication of popular data;
- PerfectDedup relies on symmetric encryption algorithms, which are known to be very efficient even when dealing with large data;
- PerfectDedup achieves deduplication at the level of blocks, which leads to higher storage space savings compared to file-level deduplication [12];
- PerfectDedup does not require any coordination or initialization among users;

- PerfectDedup does not incur any storage overhead for unpopular data blocks;

5.2 Secure Deduplication Based on Popularity

Given the inherent incompatibility between encryption and deduplication, existing solutions suffer from different drawbacks. CE was considered to be the most convenient solution for secure deduplication but it has been proved that is is vulnerable to various types of attacks [14]. Hence, CE cannot be employed to protect data confidentiality and thus stronger encryption mechanisms are required.

We point out that data may need different levels of protection depending on its popularity [7] a data segment becomes "popular" whenever it belongs to more than t users (where t is the popularity threshold). The "popularity" of a block is viewed as a trigger for its deduplication. Similarly, a data segment is considered to be unpopular if it belongs to less than t users. This is the case for all highly sensitive data, which are likely to be unique and thus unlikely to be duplicated.

Given this simple distinction, we observe that popular data do not require the same level of protection as unpopular data and therefore propose different forms of encryption for popular and unpopular data. For instance, if a file is easily accessible by anyone on the Internet, then it is reasonable to consider a less secure protection. On the other hand, a confidential file containing sensitive information, such as a list of usernames and passwords, needs much stronger protection. Popular data can be protected with CE in order to enable source-based deduplication, whereas unpopular data must be protected with a stronger encryption. Whenever an unpopular data segment becomes popular, that is, the threshold t is reached, the encrypted data segment is converted to its convergent encrypted form in order to enable deduplication.

However, this approach, which consists of using popularity as an indicator for determining whether or not a file is confidential, may fail in some extreme cases. For instance, if a confidential file is distributed to a number of users higher than the current threshold t, it will be encrypted with CE, resulting in a severe exposure to the well-known weaknesses of CE. As a simple solution to this issue, we point out that when the user wants to keep a file confidential even when it becomes popular, he may encrypt the file with a standard encryption solution and then upload it. This way, even though the file will reach the threshold and thus become

popular, CE will be applied on top of the existing user encryption layer, resulting in confidentiality still being guaranteed. Alternatively, the system may provide an option for allowing the user to define a file as non-deduplicable and never get it encrypted with CE. As opposed to simply increasing the popularity threshold, both approaches would guarantee full confidentiality of the most sensitive data without negatively affecting storage space savings. Indeed, as shown in the study of Chapter 3, increasing the global popularity threshold for all files would likely imply the loss of the ability of detecting a non-negligible fraction of the redundant blocks, resulting in a performance degradation of the whole system.

We propose to encrypt unique and thus unpopular data blocks (which cannot be deduplicated) with a symmetric encryption scheme using a random key, which provides the highest level of protection while improving the computational cost at the client. Whenever a client wishes to upload a data segment, we propose that she should first discover its popularity degree in order to perform the appropriate encryption operation. The client may first lookup for a convergent encrypted version of the data stored at the CSP. If such data segment already exists, then the client discovers that this data segment is popular and hence can be deduplicated. If such data segment does not exist, the client will encrypt it with a symmetric encryption scheme. Such a solution would greatly optimize the encryption cost and the upload cost at the client. However, a standard lookup solution for the convergent encrypted data segment would reveal the convergent encrypted data segment ID, that is the digest of the data computed under an unkeyed hash function like SHA-2, which would be a serious breach. Secure lookup for a data segment is thus a delicate problem since the ID used as the input to the lookup query can lead to severe data leakage as explained in [5] and [14]. Therefore, in such a scenario the main challenge becomes how to enable the client to securely determine the popularity of a data segment without leaking any exploitable information to the CSP. Also, the client needs to securely handle the "popularity transition", that is the phase triggered by a data segment that has just reached the popularity threshold t. More formally, the popularity detection problem can be defined as follows: given a data segment D and its ID ID_D, the client wants to determine whether ID_D belongs to the set P of popular data segment IDs stored at an untrusted CSP. It is crucial that if $ID_D \notin P$, no information must be leaked to the CSP. More generally, this problem can be seen as an instance of the Private Set Intersection (PSI) problem [69]. However, existing solutions are known to be costly in terms of computation and communication, especially when dealing with

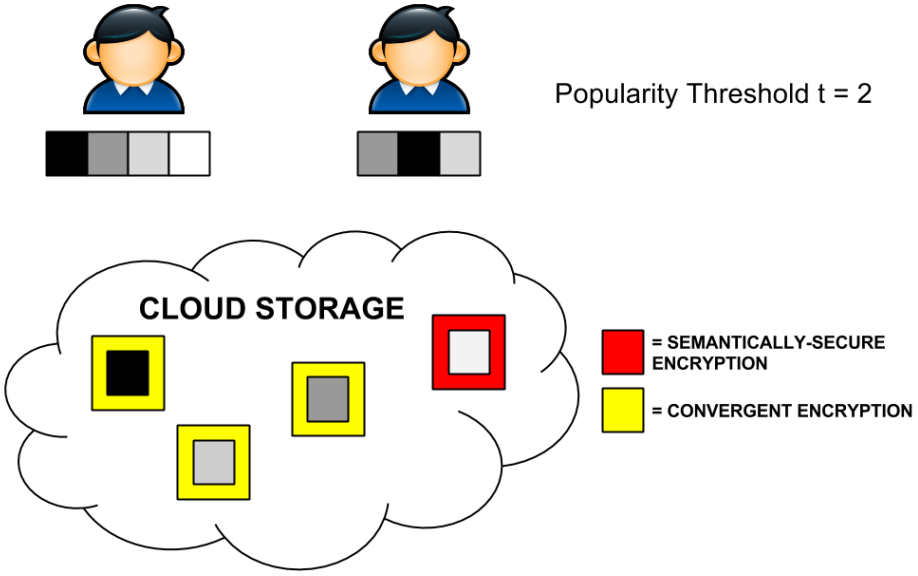

FIGURE 5.1: Our approach: popular data are protected with CE whereas unpopular data are protected with a stronger encryption

very large sets. Private Information Retrieval (PIR) [70] may also be a solution to this problem. However, using PIR raises two main issues: first, it would incur a significant communication overhead; second, PIR is designed to retrieve a single element per query, whereas an efficient protocol for the popularity check should allow to check the existence of multiple data segment IDs at once. Hence instead of complex cryptographic primitives like PSI and PIR we suggest a secure mechanism for popularity detection based on a lightweight building block called Perfect Hashing [11]. We aim at solving this problem by designing a novel secure lookup protocol, which is defined in next section, based on Perfect Hashing [11].

5.3 Basic Idea: Popularity Detection Based on Perfect Hashing

The popularity detection solution we propose makes use of the Perfect Hashing process which, given an input set of n data segments, finds a collision-free hash function, called the perfect hash function (PHF), that maps the input set to a set of m integers (m being larger than n by a given load factor). The CSP can run this process in order to generate the PHF matching the IDs of the convergent encrypted popular blocks that are currently stored at the CSP. The resulting PHF can be efficiently encoded into a file and sent to the client. Using the PHF received

from the CSP, the client can lookup for new blocks in the set of encrypted popular block IDs stored at the CSP, as illustrated in Figure 5.2. For each new block D, the client first encrypts the block to get $CE(D)$, he then computes the ID thereof using an unkeyed hash function h like SHA-2. Finally, by evaluating the PHF over ID, the client gets the lookup index i for the new block. The integer i will be the input of the lookup query issued by the client. Once the CSP has received the lookup query containing i, he will return to the client the convergent encrypted popular block ID stored under i. At this point, the client can easily detect the popularity of his data segment by comparing the ID he computed with the one received from the CSP: if the two IDs match, then D is popular. As mentioned above, it is a crucial requirement to prevent the CSP from discovering the content of the block D when it is yet unpopular. We achieve so by introducing an enhanced and secure version of Perfect Hashing, which makes the generated PHF one-way, meaning that the CSP cannot efficiently derive the input of the PHF from its output i. This also implies that the PHF must yield well-distributed collisions for unpopular blocks.

However, even though the client is now able to securely detect the popularity of a block, he still needs to handle the popularity transition, that is the phase in which a block reaches the threshold t and the convergent encrypted block needs to be uploaded to the CSP. Since the client cannot be aware of other copies of the same block previously uploaded by other users, a mechanism to keep track of the unpopular data blocks is needed. Clearly, the client cannot rely on the CSP for this task, as the CSP is not a trusted component. Therefore, we propose to introduce a semi-trusted component called Index Service (IS), which is responsible for keeping track of unpopular blocks. If the result of a popularity check is negative, then the client updates the IS accordingly by sending the popular convergent encrypted block ID and the ID of the symmetrically encrypted block. As soon as a block becomes popular, that is reaches the threshold t, the popularity transition is triggered and the client is notified in order to let him upload the convergent encrypted block, which from now on will be deduplicated by the CSP. Upon a popularity transition, the IS will delete from its storage any information related to the newly popular block. Regarding the popularity threshold, we point out that users do not have to be aware of its value, since the popularity transition is entirely managed by the IS, that is responsible for determining the current value for t. For instance, the value of t may be either static or dynamic, as proposed

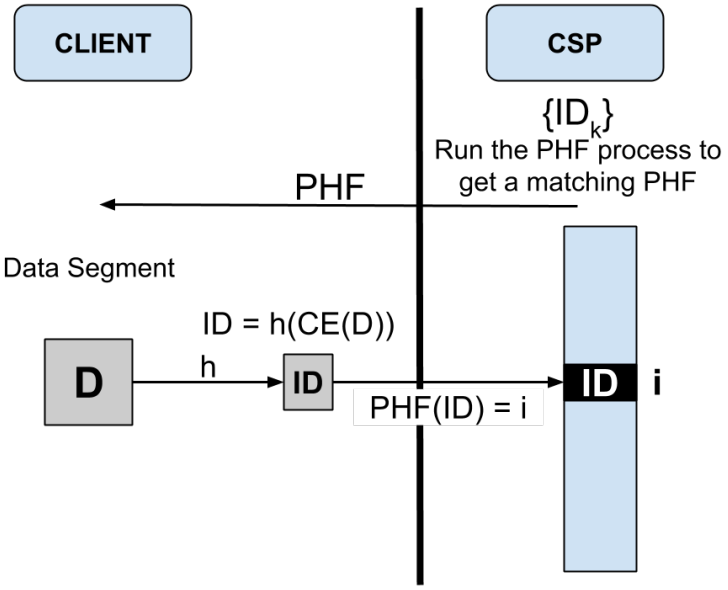

FIGURE 5.2: The secure PHF allows users to detect popular blocks while preventing the CSP from discovering unpopular blocks

in [7]. Indeed, our scheme is completely independent of the strategy used for determining the value of the popularity threshold.

5.4 Background

5.4.1 Perfect Hashing

A Perfect Hash Function (PHF) maps a set of arbitrary entries into a set of integers without collisions. Authors in [11] proposed a new algorithm that allows finding a perfect mapping for very large sets in a very efficient way. This algorithm, which is called CHD (Compress, Hash and Displace), has been designed with scalability as a main design goal. Indeed, even when dealing with datasets of millions of elements, it achieves linear space and computational complexities (with respect to the size of the set). The main idea behind this algorithm is to split the input set into several buckets (subsets), each containing a few elements, and find a collision-free mapping for each of these buckets separately. This approach has proved to be much more scalable than previous approaches. The mean number of elements per bucket is a parameter that can be tuned upon executing the generation algorithm. CHD also allows choosing a load factor, which is the fraction of non-empty positions in the hash table.

Although perfect hashing is widely adopted for efficient indexing in the field of relational databases [71], it has some desirable properties which make it an appropriate building block for our scheme. First, the computational complexity to build the PHF is linear and the PHF can be evaluated in constant time. Thanks to these properties, the system is scalable since the PHF generation remains feasible when dealing with very large datasets. In addition to that, the main computational load is outsourced to the CSP, while the client only has to perform very simple and lightweight operations such as evaluating the PHF on block IDs and symmetrically encrypting data blocks. Second, thanks to a special encoding and compression mechanism, the size of the PHF file is small and therefore it can easily be transferred to the client. Therefore, the performance impact is minimal and this approach can easily scale up to sets of millions of elements. Third, the resulting hash table is collision-free with respect to the elements of the input set (popular block IDs), meaning that any index is associated to at most one element of the input set. On the other hand, if the PHF is evaluated over the rest of the domain (unpopular block IDs) then collisions are well-distributed. This property is an important starting point to build our secure lookup protocol which must guarantee that an attacker is not able to determine on what input the PHF has been evaluated. Indeed, while an index in the hash table corresponds to a unique popular block ID, many unpopular block IDs are mapped to the same index. Therefore, given an index in the hash table, the CSP cannot determine the corresponding block ID. In our solution we propose to extend the existing PHF by replacing the underlying hash function with a one-way secure hash function such as SHA-2 [13]. Indeed, for the security of the scheme, it is crucial that the hash function used by the algorithm is one-way, meaning that it is easy to compute on a given input, but hard to invert given the image of a random input.

5.4.1.1 CHD Algorithm

In this section we explain in more detail how the CHD algorithm works and why it is currently considered as the fastest and most scalable perfect hashing algorithm. To start with, it is worth pointing out that CHD requires to configure two parameters, the load factor and the mean bucket size, which need to be chosen carefully since they can significantly affect the tradeoff between the generation time and the size of the file representing the PHF. The load factor is a value between 0 to 1 (in practice it is set between 0.5 to 0.99) which represents the fraction of empty

positions in the final hash table. Intuitively, the lower this value is, the easier (faster) will be to find a perfect mapping for all the elements of the input set. On the other hand, the mean bucket size also plays a crucial role in the performance of the generation process and the size of the PHF file. More precisely, as mentioned earlier, CHD splits the input set into multiple buckets, where the number of buckets is chosen as the result of the size of the input set divided by the mean bucket size. The smaller a bucket is, the easier is to find a mapping (unallocated positions in the hash table) for all elements in the bucket. However, since the PHF file stores information for each bucket, a low mean bucket size will increase the number of buckets and thus the size of the PHF file.

We now have a closer look at the algorithm in order to understand the idea behind it and the structure of the resulting PHF, which can be encoded in a file and decoded by users. The data structures used to represent the PHF are two: the first data structure uses a hash function h in order to map each element x of the input set S to a bucket B_i; the second data structure associates a hash function g_i to each bucket B_i. For the sake of simplicity, we can assume to have a family of hash functions which can be enumerated and from which we can pick a hash function based on its index. In practice, the authors in [11] suggest that this can be efficiently achieved by using two hash functions and a pair of displacement factors. As an example, a hash function g_i can be built on top of two hash functions as follows: $g_i)(x) = h_1(x)+a*h_2(x)+b$. Given the input set S and the two parameters mentioned above, the goal of the CHD algorithm is to find for each bucket B_i a hash function g_i such that all elements are mapped to a unique and unallocated position in the final hash table. More formally, as a result of this split operation we have r buckets such that

$$B_i = \{x \in S s.t. h(x) = i\} \tag{5.1}$$

For each bucket B_i we have a second hash function g_i that is found thanks to the algorithm. Therefore, the mapping for an element x is defined as follows:

$$f(x) = g_{h(x)}(x) \tag{5.2}$$

that is perfect over the whole input set S. Given these definitions, the main steps of the algorithm can be described as follows

CHD Algorithm

```
Split S into r buckets Bi using h
Initialize array T[0, ..., m1] with 0's;
for all i from 0 to r do
   for l = 1,2, ... repeat forming Ki = {gl(x) | x in Bi}
   until |Ki| = |Bi| and INTERSECTION(Ki, {j | T[j] = 1}) = {}
   let gi = the successful l
   for all j in Ki let T[j] = 1
```

where m is the size of the resulting hash table that is calculated by dividing the size of the input set by the load factor. As we can see, the result of this algorithm is a sequence of indices where each index corresponds to the hash function mapped to the corresponding bucket. In order to get this result, the algorithm iterates over each bucket and for each of them aims at finding a hash function that maps all elements in the bucket to empty positions in the hash table. The authors in [11] prove that the indices do not diverge, therefore both the generation time and the size of the PHF file are linear with respect to the cardinality of the input set. Also, since the indices are bounded they can be represented with a lower number of bits. In the current implementation, the resulting sequence of indices is further compressed by using the method proposed in [72] which allows to retrieve an element from the list in constant time. Thanks to this compression mechanism and other low-level optimizations, with a load factor of 0.81, CHD is able to represent a perfect hash function using only 1.40 bits per element of the input set.

5.5 The system

5.5.1 Overview

We consider a scenario where users want to store their data (files) on a potentially untrusted Cloud Storage Provider (CSP) while taking advantage of source-based block-level deduplication and protecting the confidentiality of their data at the same time. Users run a client C which is a lightweight component with respect to both storage and computational capacity. CSP is assumed to be honest-but-curious and thus correctly stores users' data while trying to disclose the content

thereof. Prior to uploading its data, C runs a secure lookup protocol to check whether the data are popular. The CSP is responsible for the generation of the PHF over the popular blocks and the storage of the resulting collision-free hash table. The proposed protocol introduces a trusted third party called Index Service (IS) which helps the client to discover the actual number of copies of a yet unpopular block. We stress the fact that IS only stores information on unpopular blocks and once a block becomes popular, all corresponding information are removed from its database, hence this component does not need to have a significant storage capacity.

The proposed solution is described under three different scenarios:

- Unpopular data upload (Scenario 1): if C finds out that the data is yet unpopular, it performs the upload to the CSP and updates the IS;
- Popularity transition (Scenario 2): if C finds out that the popularity degree of the data is $t-1$ (where t is the popularity threshold), then it performs the appropriate operations to upload the newly popular data. IS removes all information with respect to this specific data and CSP deletes all the encrypted copies previously stored;
- Popular data upload (Scenario 3): C only uploads metadata since it has detected that the requested data is popular, therefore deduplication can take place.

CSP stores a hash table for popular block IDs which is constructed with the previously introduced PHF. Each element of the hash table is defined by the couple $(PHF(h(CE(b_i))), h(CE(b_i)))$ where $h(CE(b_i))$ is the unkeyed secure hash of the convergent encrypted block. Before any operation, given the current set of popular blocks, CSP creates a corresponding secure PHF. This PHF is updated only when CSP needs to store new popular blocks. In the next sections, we first present the popularity check phase which is common to all three scenarios and then explain the following phases.

5.5.2 Popularity Check (Scenarios 1, 2 and 3)

Before uploading a file F, C splits F into blocks $F = \{b_i\}$, encrypts each of them with CE and computes their IDs. We point out that our scheme is completely

independent of the underlying data-chunking strategy used for determining block boundaries, which is a problem that is out of the scope of this work. The client fetches the PHF from the CSP and evaluates it over $\{h(CE(b_i))\}$. The result of this operation is a set of indices $I = \{PHF(h(CE(b_i)))\}$, where each index represents the position of the potentially popular block ID in the hash table stored at the CSP. These indices can be used to perform the popularity check without revealing the content of the blocks to the CSP. Indeed, given a set of indices obtained as above, the client can retrieve the corresponding block IDs stored in the hash table and then compare them with his own block IDs. Any block b_i such that $h(CE(b_i))$ is equal to the popular block ID retrieved from the CSP, is considered as popular, hence will be deduplicated. The index does not reveal any exploitable information on the block.

5.5.3 Popularity Transition (Scenarios 1 and 2)

If the popularity check reveals that a block is not popular, C needs to check whether it is going to trigger a popularity transition. A block becomes popular as soon as it has been uploaded by t users. In order to enable C to be aware of the change of the popularity status and perform the transition, C sends an update to the IS whenever the popularity check has returned a negative result for a given block ID. IS stores a list of block IDs and owners corresponding to each encrypted copy of the yet unpopular block. When the number of data owners for a particular block reaches t, the popularity transition protocol is triggered and IS returns to C the list of block IDs. In order to complete this transition phase, CSP stores the convergent-encrypted copy, removes the corresponding encrypted copies and updates the PHF. From now on, the block will be considered popular, therefore it will be deduplicated. We point out that this operation is totally transparent to the other users who uploaded the same block as unpopular. Indeed, during their upload phase, users also keep encrypted information about the convergent encryption key. This allows them decrypting the block when it becomes popular.

5.5.4 Data Upload (Scenarios 1, 2 and 3)

Once the client has determined the popularity of each block, he can send the actual upload request. The content of the request varies depending on the block

status. If the block is unpopular, C uploads the block symmetrically encrypted with a random key. If the block is popular, C only uploads the block ID, so that the CSP can update his data structures. Optionally, in order to avoid to manage the storage of the encryption keys, C may rely on the CSP for the storage of the random encryption key and the convergent encryption key, both encrypted with a secret key known only by the client.

5.6 Security Analysis

In this section, we analyze the security of the proposed scheme, the CSP being considered the main adversary. The CSP is "honest-but-curious", meaning that it correctly performs all operations but it may try to discover the original content of unpopular data. We do not consider scenarios where the CSP behaves in a byzantine way. We assume that CSP cannot collude with the IS since this component is trusted. Since the goal of the malicious CSP is to discover the content of unpopular blocks, we analyze in detail whether (and how) confidentiality is guaranteed for unpopular data in all phases of the protocol. Finally, we also analyze some attacks that may be perpetrated by users themselves and propose simple countermeasures against them.

Security of blocks stored at the CSP By definition, an unpopular block is encrypted using a semantically-secure symmetric encryption. The confidentiality of unpopular data segments thus is guaranteed thanks to the security of the underlying encryption mechanism.

Security during Popularity Check The information exchanged during the Popularity Check must not reveal any information that may leak the identity of an unpopular block owned by the user. The identity of an unpopular block is protected thanks to the one-wayness of the secure PHF: the query generated by the client does not include the actual unpopular block ID but an integer i that is calculated by evaluating the secure PHF on the block ID. Simple guessing by exploring the results of the secure hash function embedded in the PHF is not feasible thanks to the one-wayness of the underlying secure hash function (SHA-2 [13]). In addition to that, when the PHF is evaluated over an unpopular block ID, there is definitely a collision between the ID of the unpopular block and the ID of a popular block stored at the CSP. These collisions serve as the main countermeasure

to the disclosure of the unpopular block ID sent to the CSP during the lookup. With a reasonable assumption, we can also consider that the output of the underlying secure hash function (SHA-2) is random. In case of a collision between an unpopular block ID and the ID of a popular block stored at the CSP, thanks to the randomness of the underlying secure hash function, the output of a PHF based on such a hash function is uniformly distributed between 0 and m. In the case of such a collision, the probability that the CSP guesses the unpopular block ID used as input to the PHF by the client thus is:

$$\frac{m}{\left|\bar{P}\right|} = \frac{|P|}{\left|\bar{P}\right| * \alpha} \tag{5.3}$$

where P is the set of popular block IDs stored at the CSP, $\bar{P}$ is the rest of the block ID domain including all possible unpopular block IDs, α is the load factor of the PHF such that $m = \frac{|P|}{\alpha}$.

Assuming that the cardinality of the entire domain is much larger than the cardinality of the set of popular block IDs (which is the case if popular block IDs are the result of a secure hash function), we can state that the number of collisions per index is large enough to prevent a malicious CSP from inferring the actual block ID used as input to the PHF. In a typical scenario using a PHF based on a secure hash function like SHA-2, whereby the complexity of a collision attack would be 2^{256}, and a popular block ID set with 10^9 elements, this probability will be ($\alpha = 0.81$):

$$\frac{10^9}{(2^{256} - 10^9) * 0.81} \approx 1.06 * 10^{-68} \tag{5.4}$$

Hence collisions can effectively hide the identity of unpopular blocks from an untrusted cloud provider while keeping the lookup protocol extremely efficient and lightweight for the users.

Security against potential protocol vulnerabilities We now consider a few additional attacks that may be perpetrated by the CSP. For each of them, we propose simple but effective countermeasures, which are easy to implement and do not significantly increase the computational and network overhead. First, we consider that the CSP may pre-build a PHF based on some specific data (derived for example from a dictionary) which have not been yet uploaded by users. Within

such a scenario, clients would detect their requested block to be popular although it has never actually been uploaded by any user; such a block will then be stored with a lower level of protection. As a countermeasure to such an attack, we propose that the IS attaches a signature to each popular block ID upon the Popularity Transition. Therefore, the IS will sign popular block IDs before being stored at the CSP, enabling clients to verify the authenticity of these blocks when running the popularity check. Such a countermeasure would have a minimal impact on the performance of the system. Another attack we consider is related to the confirmation-of-file attack to which convergent encryption is also vulnerable [14]. Indeed, upon a Popularity Check, the CSP may compare the sequence of indices sent by the client with the sequence produced by a given popular file F. If the two sequences match, then there is a chance that the client is actually uploading F. In order to hide this information from the CSP, the client may add a number of random indices to the list of indices being sent upon the Popularity Check. Thanks to the resulting noise included in the index list, the identification of the target file by the CSP will be prevented. This countermeasure also prevents the CSP from running the learn-the-remaining-information attack. Moreover, the overhead due to this countermeasure is negligible both in terms of bandwidth and computation.

Security against users Users may force a popularity transition by repeatedly uploading random or targeted blocks. As a countermeasure, the popularity threshold may be set to a value $t' = t + u$, where u is the expectation of the maximum number of malicious users. As opposed to the proposal of [7], the threshold can be dynamically updated at any time of the system life. Indeed, this parameter is transparent to both users and the CSP, hence the Index Service can update it depending on the security needs. For instance, in a scenario where all users are trusted ($u = 0$) and thus no collisions will ever occur, t' may be set to 2. Moreover, we assume that users can only have a unique authenticated identity and hence cannot impersonate other existing users or create fake identities with the aim of increasing the popularity of a block artificially.

However, as we discussed in Chapter 3, globally increasing the popularity threshold can cause a significant decrease in the storage space savings. Fortunately, since in PerfectDedup the popularity threshold can be dynamic and transparent to users, we can propose a more advanced and fine-grained solution. For instance, we may put in place a strategy aimed at using a different popularity threshold per file, based on the likelihood of presence of confidential content. As an example, upon

the upload of a file, the user may be asked to assign a level of confidentiality. Based on this attribute, the IS can transparently set a different threshold for each file: the higher the confidentiality requested for a file, the higher the popularity threshold applied to its blocks.

Users may also perpetrate a DoS attack by deleting random blocks stored at the cloud. This may happen upon a popularity transition: the client is asked to attach a list of block IDs that may not be the actual encrypted copies of the block being uploaded. We suggest making the Index Service sign the list of block IDs to be deleted so that the cloud can verify whether the request is authentic. This signature does not significantly increase the overhead since several schemes for short signatures [73] have been proposed in the literature.

5.7 Performance Evaluation

5.7.1 Prototype Implementation

In order to prove the feasibility of our approach, we implemented a proof-of-concept prototype consisting of the three main components, namely, the Client, the IS and the CSP. All components have been implemented in Python. Cryptographic functions have been implemented using the pycrypto library [74]. Both the Client and the IS run on an Ubuntu VM hosted on our OpenStack platform, while the CSP runs on an Ubuntu VM hosted on Amazon EC2 (EU Region). Additionally, the Index Service runs a server based on the tornado web framework [65], and uses REDIS [9] in order to efficiently store the information on unpopular blocks, which are encoded as lists. The CSP also runs a server based on tornado. Metadata (block IDs, file IDs, files structures, encrypted keys) are stored in a MySQL database, which is considered as one of the most reliable software components for storing large scale metadata. Perfect Hashing has been implemented using the CMPH library [75] at both the Client and the CSP. In order to achieve one-wayness, we customized CMPH by replacing the internal hash function with SHA256 [13]. We stress the fact that this is a proof-of-concept implementation, therefore for the sake of simplicity the CSP has been deployed on a VM where data blocks are stored locally. In a production environment, the CSP service may be deployed on a larger scale and any storage provider such as Amazon S3 [20] may be employed to physically store blocks.

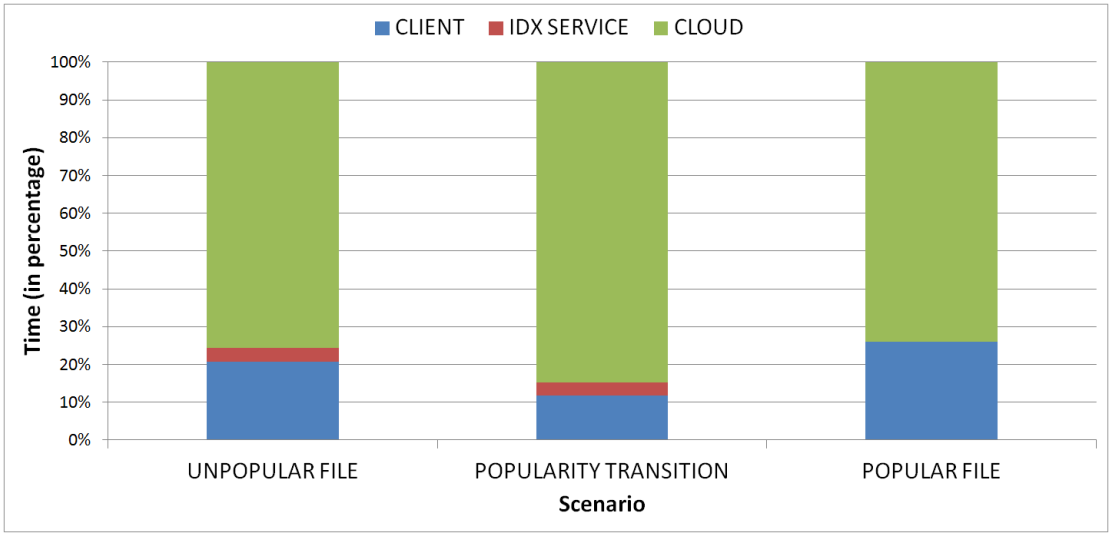

FIGURE 5.3: Portion of the total computation time spent at each component in each scenario

We consider a scenario where the client uploads a 10MB file to the CSP pre-filled with 10^6 random blocks. We propose to first evaluate the computational overhead of each single component and measure the total time a client needs to wait during each phase until the data upload has been completed. We then analyze the network overhead of the proposed solution. Our analysis considers the three previously described scenarios:

- Scenario 1 (Unpopular File): the file to be uploaded is still unpopular;
- Scenario 2 (Popularity Transition): the file has triggered a popularity transition hence is going to become popular;
- Scenario 3 (Popular File): the file to be uploaded is already popular.

For the sake of consistency, all experiments have been repeated 10 times and the results shown in this section are mean values.

5.7.2 Computational Overhead

In this section we present our measurements of the computational overhead at each component and then show the total time a client takes to upload a file. Figure 5.3 shows an aggregate measure of all computation-intensive operations each component performs. The results prove that, as expected, the computational overhead

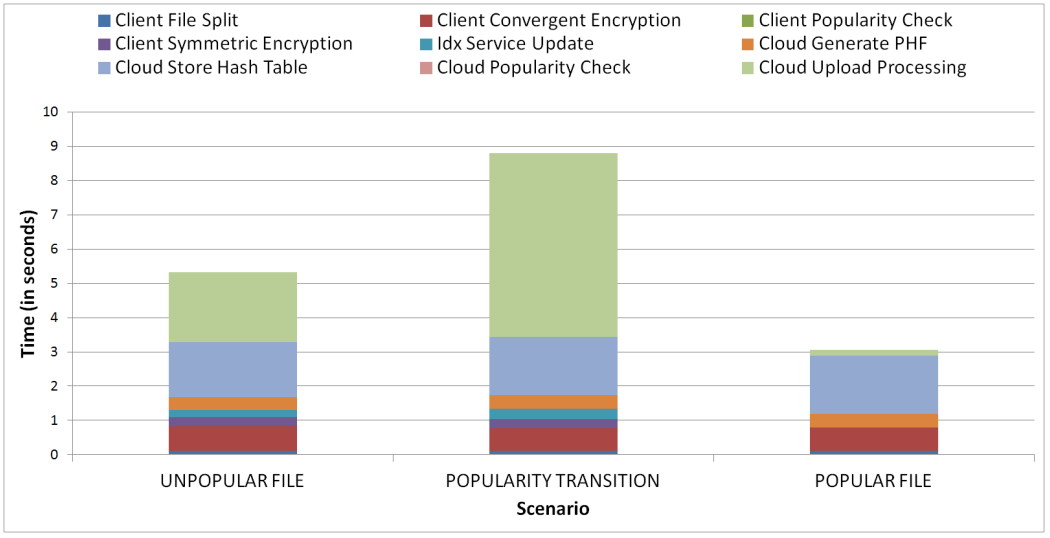

FIGURE 5.4: Total time spent during each phase of the protocol in each scenario

introduced in the CSP is much higher than the one affecting the client. Also, since the operations performed by the IS are extremely simple, its computational overhead is negligible.

Figure 5.4 shows more detailed results by highlighting which operations introduce a higher computational overhead. The results prove that:

- Symmetric encryption introduces a negligible computational overhead, hence it does not affect the system performance;
- The client-side Popularity Check is extremely lightweight and thus introduces a negligible computational overhead;
- The most computation-intensive operations (PHF generation, hash table storage, upload processing) are performed by the CSP, hence a big fraction of the computational overhead is outsourced to the CSP.

Figures 5.5 and 5.6 show the results of an in-depth study on the performance of the Perfect Hashing algorithm, both in terms of storage space and computation time for the generation of the PHF. The generation time also includes the time needed to store the hash table. We measured these quantities on a dataset of 10^6 random block IDs while varying the load factor and the bucket size. The former is a coefficient indicating the fraction of non-empty positions in the final collision-free hash table; the latter is the mean number of elements in each subset of the input set (see [11] for further details). As we can observe from Figures 5.5 and

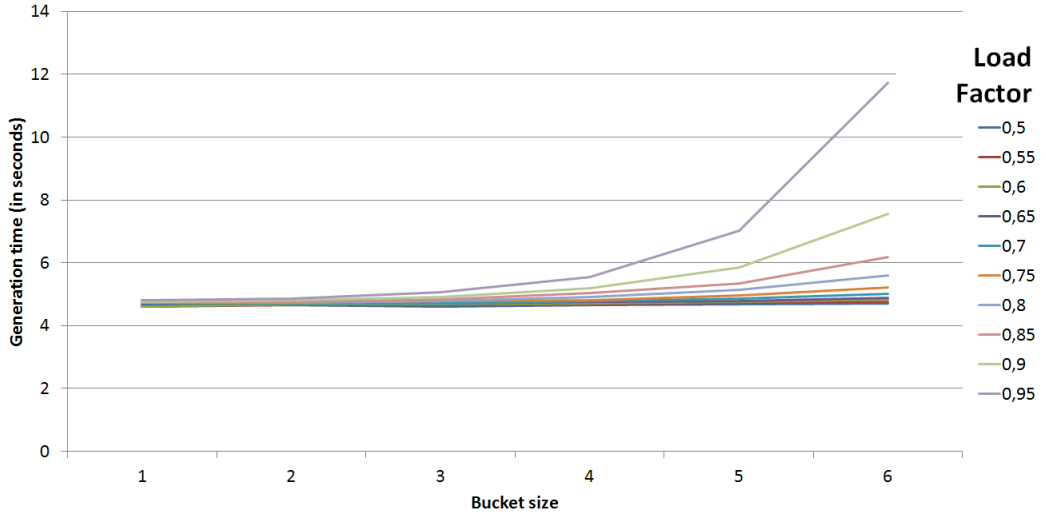

FIGURE 5.5: Analysis of PHF generation time with varying parameters for a set containing 10^6 elements

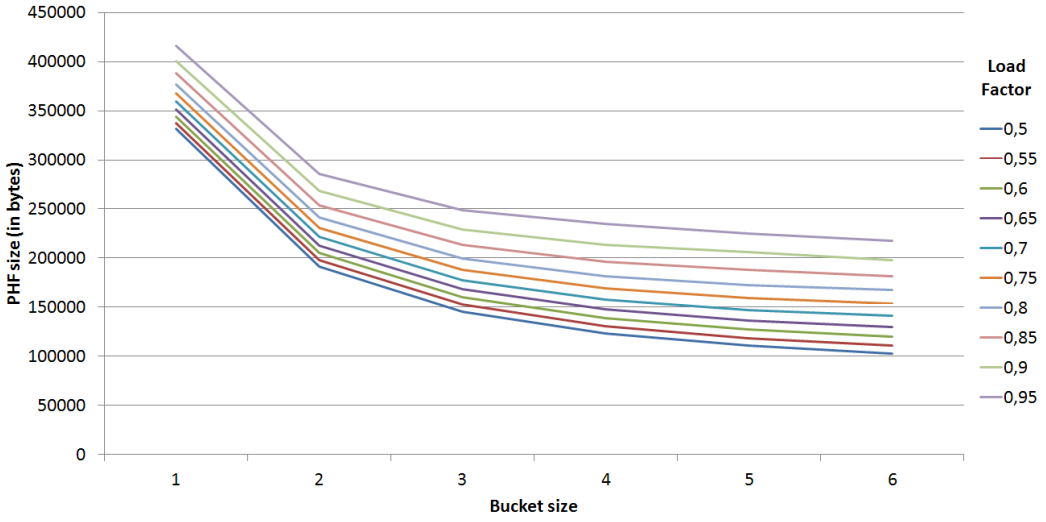

FIGURE 5.6: Analysis of PHF size with varying parameters for a set containing 10^6 elements

5.6, the optimal bucket size is between 3 and 4 and the load factor should not be greater than 0.8. These parameters can be tuned depending on the scenario (e.g. bandwidth) in order to achieve the best performance.

Furthermore, as mentioned earlier, in order to improve the security of our novel lookup protocol, we replaced the default hash function employed by the CMPH library (Jenkins [76])) with a secure hash function such as SHA-2. This improvement is required for the following reason: using a non-secure hash function would allow an adversary such as the CSP to easily enumerate all block IDs mapped to a given index of the hash table. Such a threat may compromise the security

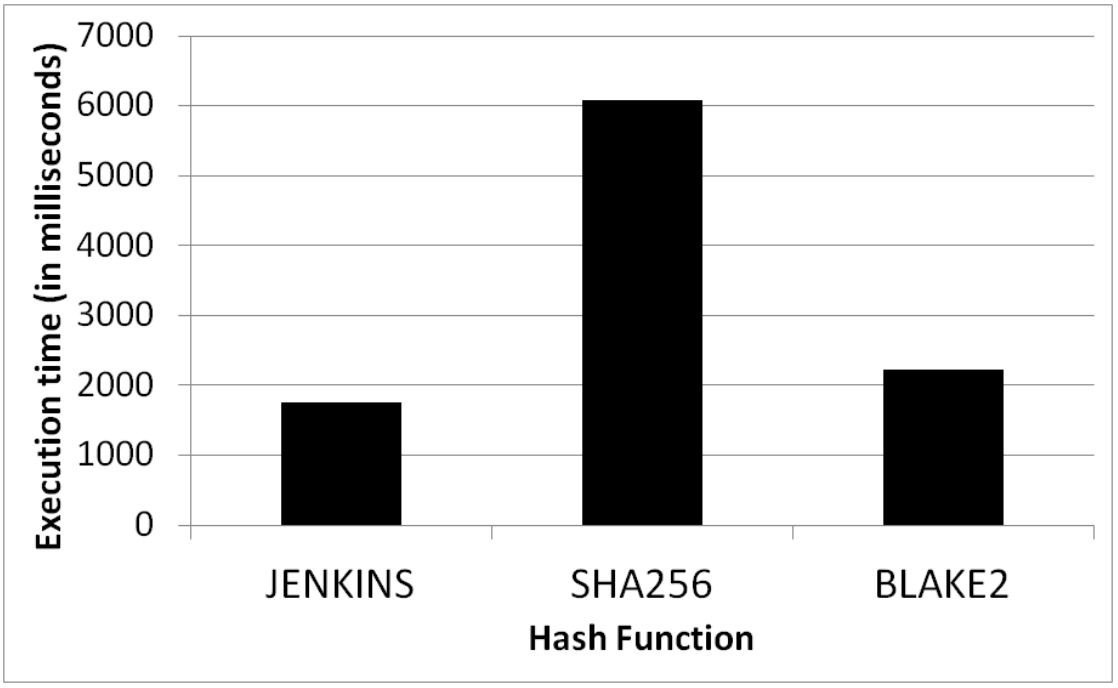

FIGURE 5.7: Performance comparison of Jenkins, SHA-2 (SHA256) and Blake2 hash functions

of the whole system and make the popularity check protocol insecure. In Figure 5.7 we show the result of the comparison of the performance of three hash functions: Jenkins (insecure), SHA-2 (secure) and Blake2 [77] (secure and optimized). We decided to include Blake2 in this analysis in order to prove that an efficient yet secure hash function does not dramatically increase the computational overhead compared to the original implementation. As an example, Blake2 achieves performance close to Jenkins thanks to the use of parallelization and hardware acceleration.

5.7.2.1 Conclusion

Figure 5.8 summarizes all measurements by showing the total time spent during each phase of the upload protocol within the three scenarios. These results show that despite the delay introduced by the Popularity Check phase, the user achieves a throughput of approximately 1MB per second even when a file does not contain any popular block.

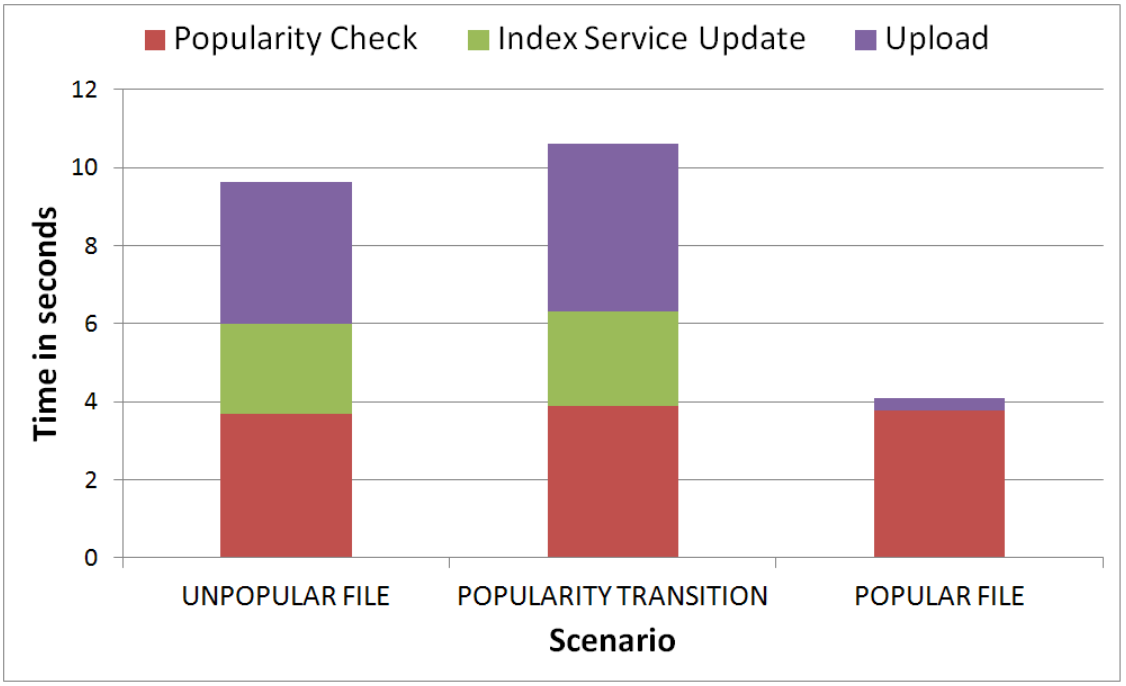

FIGURE 5.8: Total time spent by all components when uploading a file (including Popularity Check) in each scenario

5.7.3 Communication Overhead

In this section we analyze the communication overhead of our scheme considering the same scenarios. The upload has been split into multiple sub-operations: PHF Download, Popularity Check, Index Service Update (not performed in Scenario 2) and the Upload. For each of these operations we analyze the size of all messages exchanged (both requests and responses). Table 5.1 regroups all the results expressed in MB. The PHF Download response size is linear with respect to the set of popular block IDs. The larger the set, the larger the response will be. However, as shown in [11], the size of PHF file is about 1.4 bits per popular block ID; hence this operation does not introduce a significant delay even when dealing with very large datasets. We point out that the PHF file does not have to be downloaded at every request, since the user can cache it. Furthermore, the size of the Popularity Check request and response is linear with respect to the number of blocks in the file that is being uploaded. The Popularity Check request contains a list of indices (one integer per block), while the response contains a list of block IDs (one per index) of 32 bytes each. The Index Service Update request is only sent for unpopular blocks. The request consists of two block IDs (32 bytes each) per block. The response size varies depending on whether the popularity transition occurs. If the file has triggered a popularity transition, then the response includes

a list of block IDs, otherwise it is empty. As we can see from Table 5.1, requests and responses of the Popularity Check and the Index Service Update operations have a negligible size with respect to the file size. Finally, the size of the Upload request varies depending on the block status. If a block is popular, the request only consists of the block ID and one key (32 bytes). If a block is not popular, the request contains the encrypted data, two keys (32 bytes each) and a few fields: the file ID (32 bytes), the user ID and the block status (1 byte). As shown in Table 5.1, the overhead introduced by the Upload is minimal and mainly depends on the encoding method used to transfer the encrypted binary data. For simplicity, we used JSON objects to pack encrypted blocks and keys and Base64 to encode binary data, which increases the size of the data by 1/3. To summarize, the preliminary operations performed in our scheme before the Upload introduce a negligible communication overhead. In addition, the scheme does not affect the gains in terms of storage space and bandwidth achieved thanks to deduplication.

	SCENARIO 1	**SCENARIO 2**	**SCENARIO 3**
PHF DOWNLOAD IN	0.67	0.67	0.67
POPULARITY CHECK REQUEST	0.004	0.004	0.004
POPULARITY CHECK RESPONSE	0.02	0.02	0.02
INDEX SERVICE UPDATE REQUEST	0.1	0.1	-
INDEX SERVICE UPDATE RESPONSE	0.009	0.04	-
UPLOAD REQUEST	13.51	13.47	0.09

Table 5.1: Communication overhead (in MB) introduced by each operation

Chapter 6

Conclusions and Future Work

6.1 Study on Deduplication

First of all, based on the observation that deduplication raises a requirement for deterministic encryption, we assessed whether and in which scenarios block-level deduplication may pave the way for statistical attacks. In order to answer this question, we conducted a comprehensive analysis on a real and representative dataset which led to three interesting results:

- After comparing the most common data-chunking techniques on several datasets, the results clearly show that block-level deduplication always achieves the best deduplication ratios. Furthermore, the overhead due to the metadata generated by block-level deduplication does not nullify the gain in terms storage space savings, which is remarkable.
- When using realistic block sizes (going from 4KB to 32KB), the global entropy does not seem to decrease when decreasing the average block size, therefore there is no concrete evidence that a statistical attack based on frequency analysis can be successfully carried out. Also, the total number of blocks is such that a statistical attack becomes extremely difficult.
- As an additional result, we also analyzed the popularity of each block within the dataset in order to learn more about the popularity distribution and find a value such that storage space savings remain high while there is no risk for confidentiality. This is particularly useful for those systems such

as PerfectDedup that are based on a popularity threshold. As expected, the popularity distribution is far from being uniform. Indeed, slightly increasing the popularity threshold from 2 to 3 causes a drop in the storage space savings of more than 15%, regardless of the data-chunking technique being used. Therefore, based on these data, we suggest that the popularity threshold should not be higher than 5.

Thanks to this study, we also learned that redundant data blocks in real datasets are distributed in a way that is clearly non-uniform. This property may be used, in both industry and research, as an important criteria when designing or tuning a solution for storage or network optimization. For instance, when designing a caching algorithm, being able to make some assumptions on the distribution of redundant data blocks may help achieve higher cache hit rates. Also, if an organization wants to put in place a solution aimed at optimizing storage performance through caching, knowing that redundant data is not distributed uniformly can help build more effective caching policies and mechanisms.

In addition to that, from a practical point of view, we also learned about how challenging and demanding it is to build a system that is aimed at storing metadata and can scale up to very large datasets of thousands of Gigabytes. We proved that relying on key-value storage is a very efficient solution. However, additional countermeasures have to be taken in order to be able to handle large amounts of metadata that cannot fit entirely in memory.

6.2 ClouDedup

ClouDedup was our first attempt to provide a secure yet practical solution to achieve confidentiality together with block-level deduplication. We designed a system which makes use of a number of additional deterministic and symmetric encryption layers in order to cope with the weaknesses of Convergent Encryption. Additional layers of encryption are added by the Gateway and, optionally, by the Metadata Manager. As the additional encryptions are symmetric, the impact on performance is negligible. In addition to that, we showed that it is worth performing block-level deduplication instead of file-level deduplication since the gains in terms of storage space are not affected by the overhead of metadata management, which is minimal.

We also showed that our design, in which no component is completely trusted, prevents any single component from compromising the security of the whole system. Our solution also prevents curious cloud storage providers from inferring the original content of stored data by observing access patterns or accessing metadata. Furthermore, we implemented a full prototype, which showed that our solution can be easily and efficiently implemented with existing and widespread technologies. Also, our solution is fully compatible with standard cloud storage APIs and transparent for the cloud storage provider, which does not have to be aware of the running deduplication system. Therefore, any potentially untrusted cloud storage provider such as Amazon S3 may be a good candidate to play the role of storage provider.

Finally, we performed a comprehensive performance analysis on the implemented prototype aimed at measuring the current performance of the system, especially when uploading a file, and pointing out any existing performance issue. In addition to that, for each component we measured the theoretical and practical overhead in terms of computation, network and storage. The results of the analysis show that although there is a potential bottleneck between the Metadata Manager and the Cloud Storage Provider, thanks to our improvements the current implementation achieves great performance along with affordable network and storage overheads. Also, we pointed out a few practical optimizations that may further decrease storage and network overheads.

As part of future work, ClouDedup may be extended with more security features such as proofs of retrievability [15], data integrity checking [16], search over encrypted data [17], [18] and secure file sharing [19], which is a very valued feature in cloud storage. In the near future we aim at completing the implementation of the current prototype in order to make it a production-ready cloud storage system and start deploying it in real scenarios. Furthermore, we will work on finding possible optimizations in terms of bandwidth, storage space and computation.

6.3 PerfectDedup

Based on the expertise earned during the design and development of ClouDedup, we decided to work on a new scheme aimed at solving the same problem while coping with the existing shortcomings, among which there is:

- **No source-based deduplication:** in ClouDedup, clients upload all data to the Gateway which encrypts and forwards it to the Metadata Manager, where finally deduplication takes place. This means that, the scheme does not allow for bandwidth savings.
- **Complex architecture:** ClouDedup requires to deploy a complex architecture in which a component, that is the Gateway, needs to be trusted with respect to the encryption operation. This means that if the Gateway is compromised confidentiality is no longer guaranteed.

Based on these shortcomings, we designed a system which guarantees full confidentiality for confidential files while enabling source-based block-level deduplication for popular files. The main building block of our system is a novel secure lookup protocol built on top of an enhanced version of Perfect Hashing. To the best of our knowledge, this is the first work that uses Perfect Hashing for a different purpose other than database indexing. A semi-trusted component is employed for the purpose of storing metadata concerning unpopular data and providing a support for detecting popularity transitions, meaning that a data block has just reached the popularity threshold.

Once again, we implemented a prototype of the proposed solution. Our measurements show that the storage, network and computational overhead is affordable and does not affect the advantage of deduplication. Also, we showed that most of the computational overhead is moved to the CSP, while the client only has to perform very lightweight operations.

This work allowed us to explore an unusual yet interesting approach consisting of achieving confidentiality by leveraging collisions. More precisely, although collisions are usually seen as an event that should be avoided, in our case collisions are useful since they provide the main protection against attacks from a curious Cloud Storage Provider and allow to achieve confidentiality for users. Therefore, this contribution may pave the way for new approaches based on the use of collisions as a mean for protecting confidential information.

As part of future work, PerfectDedup may be optimized in order to reduce the overhead due to the PHF generation and transmission. In particular, in order to reduce the generation time when dealing with very large datasets, the PHF should be dynamic, meaning that a block ID can be inserted or deleted without

necessarily generating the PHF from scratch. In addition to that, it would be beneficial to have a solution that allows a user to incrementally update the PHF without having to download it entirely at every update.

6.4 Future Work

At the time of the writing, designing a fully-featured scheme for secure and efficient storage remains an open challenge. We suggest that extending one of our schemes with new mechanisms aimed at providing missing security features can be a fruitful research direction.

For instance, given a secure deduplication scheme such as ClouDedup and PerfectDedup, it would be beneficial for users to integrate a number of additional security features such as such as proofs of retrievability (PoR) [15], data integrity checking [16], search over encrypted data [17], [18] and secure file sharing [19]. However, such an integration may present several challenges due to the use of deterministic encryption. Indeed, most of the approaches that achieve the aforementioned security features make use of semantically-secure encryption techniques, which are known to be incompatible with deduplication. Also, although there exist some schemes [18] based on deterministic encryption, they rely on asymmetric cryptography, which is known to be inefficient when encrypting large files.

As a more explanatory example, in the case of PoR, many of the existing approaches require users to modify the original file by injecting additional pieces of information and then encrypt the whole file with a semantically-secure encryption. These extra blocks, which go by the name of secure tags, erasure codes or sentinels, allow users to remotely and securely query the Cloud Storage Provider, make it run some computation on the file and return a compact (much smaller than the file itself) value which can be finally used by the user to run a verification process and determine whether or not the file has been altered. On the other hand, the extra encryption step is required in order to make the extra blocks indistinguishable from the the rest of the file. Unfortunately, this kind of approach, which is widespread in the literature, will certainly cause a drop in deduplication ratios since it requires to change the original content of files through encryption, which lowers redundancy and increases entropy, hence undermines the ability of detecting duplicate blocks.

Furthermore, due to practical constraints including storage limitations, our data study was conducted on a single storage snapshot taken at a given point of time. However, it would be interesting to study the evolution of both confidentiality and storage space savings over time. More precisely, one could take multiple storage snapshots at different times (e.g. one snapshot per week) and compare their entropy and storage space savings. This would allow us to observe whether and how entropy and deduplication ratios change over time. For instance, since users tend to consume a constantly increasing amount of storage space, a potential outcome of such a study might lead to the conclusion that the dataset eventually reaches a steady state where most of the newly stored blocks are duplicates. As a consequence, the global entropy may decrease and the deduplication ratios may increase. Moreover, measuring deduplication ratios across consecutive storage snapshots would give useful indications on the impact of deduplication in a very popular scenario in cloud storage, that is backup.

Last but not least, another challenging option for future research is to study whether and how existing secure deduplication approaches can be applied in the context of relational databases, which are often employed in widespread applications. In such a context, the granularity and the size of data segments change, since in databases deduplication should be performed on records and fields instead of blocks. This might have serious implications with respect to confidentiality due to the fact that data segments will probably be much smaller, namely a few bytes instead of a few kilobytes. Also, in order to preserve standard operations that are usually performed in database queries (e.g. comparisons), the underlying encryption technique should allow for such operations to be carried out on encrypted data, in addition to being deterministic.

Publications

1. **PerfectDedup: Secure Data Deduplication**
   Pasquale Puzio (SecludIT EURECOM), Refik Molva (EURECOM), Melek Önen, (EURECOM), Sergio Loureiro (SecludIT)
   10th DPM International Workshop on Data Privacy Management

2. **ClouDedup: Secure Deduplication with Encrypted Data for Cloud Storage**
   Pasquale Puzio (SecludIT & EURECOM), Refik Molva (EURECOM), Melek Önen, (EURECOM), Sergio Loureiro (SecludIT)
   Delivery and Adoption of Cloud Computing Services in Contemporary Organizations, IGI Global

3. **Block-level De-duplication with Encrypted Data**
   Pasquale Puzio (SecludIT & EURECOM), Refik Molva (EURECOM), Melek Önen, (EURECOM), Sergio Loureiro (SecludIT)
   OJCC 2014, 1(1), Pages 10-18

4. **ClouDedup: Secure Deduplication with Encrypted Data for Cloud Storage**
   Pasquale Puzio (SecludIT & EURECOM), Refik Molva (EURECOM), Melek Önen, (EURECOM), Sergio Loureiro (SecludIT)
   IEEE CloudCom 2013

5. **Elastic SIEM: Elastic Detector integrated with OSSIM**
   Pasquale Puzio (SecludIT & EURECOM), Sergio Loureiro (SecludIT)
   RaSIEM Workshop, ARES 2013

Bibliography

[1] http://www.acronis.com/fr-fr/backup-recovery/deduplication-roi-calculator.html.

[2] http://opendedup.org/.

[3] Idilio Drago, Marco Mellia, Maurizio M Munafo, Anna Sperotto, Ramin Sadre, and Aiko Pras. Inside dropbox: understanding personal cloud storage services. In *Proceedings of the 2012 ACM conference on Internet measurement conference*, pages 481–494. ACM, 2012.

[4] John R Douceur, Atul Adya, William J Bolosky, Dan Simon, and Marvin Theimer. Reclaiming space from duplicate files in a serverless distributed file system. In *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, pages 617–624. IEEE, 2002.

[5] Mihir Bellare, Sriram Keelveedhi, and Thomas Ristenpart. Message-locked encryption and secure deduplication. In *Advances in Cryptology–EUROCRYPT 2013*, pages 296–312. Springer, 2013.

[6] Jian Liu, N Asokan, and Benny Pinkas. Secure deduplication of encrypted data without additional independent servers.

[7] Jan Stanek, Alessandro Sorniotti, Elli Androulaki, and Lukas Kencl. A secure data deduplication scheme for cloud storage. In *Financial Cryptography and Data Security*, pages 99–118. Springer, 2014.

[8] Pasquale Puzio, Refik Molva, Melek Onen, and Sergio Loureiro. Cloudedup: secure deduplication with encrypted data for cloud storage. In *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, volume 1, pages 363–370. IEEE, 2013.

[9] http://redis.io/.

[10] Pasquale Puzio, Refik Molva, Melek Onen, and Sergio Loureiro. Perfectdedup: Secure data deduplication.

[11] Djamal Belazzougui, Fabiano C Botelho, and Martin Dietzfelbinger. Hash, displace, and compress. In *Algorithms-ESA 2009*, pages 682–693. Springer, 2009.

[12] Dutch T Meyer and William J Bolosky. A study of practical deduplication. *ACM Transactions on Storage (TOS)*, 7(4):14, 2012.

[13] PUB FIPS. 180-4. *Federal Information Processing Standards Publication, Secure Hash*, 2012.

[14] `https://www.tahoe-lafs.org/hacktahoelafs/drew_perttula.html`.

[15] Ari Juels and Burton S. Kaliski, Jr. Pors: proofs of retrievability for large files. In *Proceedings of the 14th ACM conference on Computer and communications security*, CCS '07, pages 584–597, New York, NY, USA, 2007. ACM.

[16] 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*, CCS '09, pages 187–198, New York, NY, USA, 2009. ACM.

[17] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on*, pages 44–55. IEEE, 2000.

[18] Mihir Bellare, Alexandra Boldyreva, and Adam ONeill. Deterministic and efficiently searchable encryption. In *Advances in Cryptology-CRYPTO 2007*, pages 535–552. Springer, 2007.

[19] Mahesh Kallahalla, Erik Riedel, Ram Swaminathan, Qian Wang, and Kevin Fu. Plutus: Scalable secure file sharing on untrusted storage. In *Fast*, volume 3, pages 29–42, 2003.

[20] `https://aws.amazon.com/s3/`.

[21] Roundup of cloud computing forecasts and market estimates q3 update, 2015.

[22] Peter Mell and Tim Grance. The nist definition of cloud computing. 2011.

[23] http://www.informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks/.

[24] http://www.bbc.com/news/world-us-canada-23123964.

[25] Mihir Bellare, Sriram Keelveedhi, and Thomas Ristenpart. Dupless: Server-aided encryption for deduplicated storage. In *Proceedings of the 22nd USENIX conference on security*, pages 179–194. USENIX Association, 2013.

[26] Zooko Wilcox-O'Hearn and Brian Warner. Tahoe: the least-authority filesystem. In *Proceedings of the 4th ACM international workshop on Storage security and survivability*, pages 21–26. ACM, 2008.

[27] Mihir Bellare and Sriram Keelveedhi. Interactive message-locked encryption and secure deduplication. In *Public-Key Cryptography–PKC 2015*, pages 516–538. Springer, 2015.

[28] Craig Gentry et al. Fully homomorphic encryption using ideal lattices. In *STOC*, volume 9, pages 169–178, 2009.

[29] http://www.cloudindex.fr/.

[30] http://www.rightscale.com/lp/2015-state-of-the-cloud-report.

[31] Jon Brodkin. Gartner: Seven cloud-computing security risks. *Infoworld*, 2008:1–3, 2008.

[32] https://dropbox.com/.

[33] https://box.com/.

[34] https://drive.google.com/.

[35] https://onedrive.com/.

[36] https://www.amazon.com/clouddrive/home.

[37] https://cloud.google.com/.

[38] https://rackspace.com/.

[39] https://azure.microsoft.com/.

[40] http://swift.openstack.org/.

[41] `https://tresorit.com/`.

[42] `https://spideroak.com/`.

[43] `https://www.mcafee.com/consumer/en-us/store/m0/catalog/mls_430/mcafee-livesafe.html?pkgid=430`.

[44] Claude E Shannon. Communication theory of secrecy systems*. *Bell system technical journal*, 28(4):656–715, 1949.

[45] Whitfield Diffie and Martin E Hellman. New directions in cryptography. *Information Theory, IEEE Transactions on*, 22(6):644–654, 1976.

[46] Ronald L Rivest, Adi Shamir, and Leonard M Adleman. Cryptographic communications system and method, September 20 1983. US Patent 4,405,829.

[47] PUB FIPS. 197, advanced encryption standard (aes), national institute of standards and technology, us department of commerce (november 2001). *Link in: http://csrc. nist. gov/publications/fips/fips197/fips-197. pdf.*

[48] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in cryptology*, pages 10–18. Springer, 1985.

[49] Morris Dworkin. Recommendation for block cipher modes of operation. methods and techniques. Technical report, DTIC Document, 2001.

[50] Davis Pan. A tutorial on mpeg/audio compression. *IEEE multimedia*, (2):60–74, 1995.

[51] Barry G Haskell, Atul Puri, and Arun N Netravali. *Digital Video: An Introduction to MPEG-2: An Introduction to MPEG-2*. Springer Science & Business Media, 1997.

[52] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on information theory*, 23(3):337–343, 1977.

[53] Greg Roelofs and Richard Koman. *PNG: the definitive guide*. O'Reilly & Associates, Inc., 1999.

[54] David Jeff Jackson and Sidney Joel Hannah. Comparative analysis of image compression techniques. In *System Theory, 1993. Proceedings SSST'93., Twenty-Fifth Southeastern Symposium on*, pages 513–517. IEEE, 1993.

[55] Danny Harnik, Benny Pinkas, and Alexandra Shulman-Peleg. Side channels in cloud services: Deduplication in cloud storage. *Security & Privacy, IEEE*, 8(6):40–47, 2010.

[56] Michael O Rabin et al. *Fingerprinting by random polynomials*. Center for Research in Computing Techn., Aiken Computation Laboratory, Univ., 1981.

[57] Leonard Carlitz. The arithmetic of polynomials in a galois field. *American Journal of Mathematics*, pages 39–50, 1932.

[58] `https://github.com/joeltucci/rabin-fingerprint-c`.

[59] `https://www.bitcasa.com/`.

[60] Silvio Micali, Charles Rackoff, and Bob Sloan. The notion of security for probabilistic cryptosystems. *SIAM Journal on Computing*, 17(2):412–426, 1988.

[61] Yvo G Desmedt. Threshold cryptography. *European Transactions on Telecommunications*, 5(4):449–458, 1994.

[62] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In *Advances in CryptologyEurocrypt 2000*, pages 139–155. Springer, 2000.

[63] Anand Balachandran, Geoffrey M Voelker, Paramvir Bahl, and P Venkat Rangan. Characterizing user behavior and network performance in a public wireless lan. In *ACM SIGMETRICS Performance Evaluation Review*, volume 30, pages 195–205. ACM, 2002.

[64] `www.safenet-inc.com/data-encryption/hardware-security-modules-hsms/`.

[65] `http://tornado.readthedocs.org/en/latest/web.html`.

[66] `https://libcloud.apache.org/`.

[67] Colin Percival and Simon Josefsson. The scrypt password-based key derivation function. 2015.

[68] `http://msgpack.org/`.

[69] Michael J Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In *Advances in Cryptology-EUROCRYPT 2004*, pages 1–19. Springer, 2004.

[70] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *Journal of the ACM (JACM)*, 45(6):965–981, 1998.

[71] Edward A Fox, Lenwood S Heath, Qi Fan Chen, and Amjad M Daoud. Practical minimal perfect hash functions for large databases. *Communications of the ACM*, 35(1):105–121, 1992.

[72] Kunihiko Sadakane and Roberto Grossi. Squeezing succinct data structures into entropy bounds. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 1230–1239. Society for Industrial and Applied Mathematics, 2006.

[73] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *Advances in CryptologyASIACRYPT 2001*, pages 514–532. Springer, 2001.

[74] `https://pypi.python.org/pypi/pycrypto`.

[75] `http://cmph.sourceforge.net/`.

[76] `http://www.burtleburtle.net/bob/hash/index.html`.

[77] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-OHearn, and Christian Winnerlein. Blake2: simpler, smaller, fast as md5. In *Applied Cryptography and Network Security*, pages 119–135. Springer, 2013.