Partage de documents sécurisé dans le Cloud Personnel

Paul Tran Van

université PARIS-SACLAY

Thèse de doctorat

UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES

université PARIS-SACLAY

Partage de documents sécurisé dans le Cloud Personnel

Thèse de doctorat de l'Université Paris-Saclay
préparée à l'Université de Versailles Saint-Quentin-en-Yvelines

École doctorale n° 580 : sciences et technologies de l'information et de la communication (STIC)

Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Versailles, le 03 avril 2018, par

Paul Tran-Van

Composition du Jury :

Mme. Célia Zolynski Professeure, UVSQ	Présidente
M. Cédric du Mouza Maître de conférences, CNAM	Rapporteur
M. David Gross-Amblard Professeur, Université de Rennes 1	Rapporteur
M. Benjamin André Président, Cozy Cloud	Examinateur
M. Philippe Lamarre Professeur, INSA Lyon	Examinateur
M. Philippe Pucheral Professeur, UVSQ et Inria Paris-Saclay	Directeur de thèse
M. Nicolas Anciaux Chargé de recherche, Inria Paris-Saclay	Co-Directeur de thèse

Remerciements

Mes premiers remerciements vont naturellement à mes encadrants, Philippe Pucheral, Nicolas Anciaux et Benjamin André. Chacun d'entre eux ont su me guider et m'apporter leurs visions tout au long de ces riches années. Je n'oublie pas la proposition de Philippe de passer un entretien pour rejoindre en stage l'équipe SMIS qui m'a par la suite donné envie de continuer dans la recherche et la rencontre avec Benjamin, puis Frank, qui m'ont fait découvrir Cozy et tout son potentiel. Cette thèse n'aurait pas été possible sans eux et je leur suis immensément reconnaissant !

Je remercie également chaleureusement les membres du jury de soutenance : Cédric du Mouza, David Gross-Amblard, Célia Zolynski et Philippe Lamarre qui ont accepté de libérer un peu de leur temps, malgré des agendas déjà chargés. Merci particulièrement à Cédric et David pour leur travail de rapporteur.

L'équipe SMIS, devenue Petrus, a une place toute particulière dans ces remerciements : je pensais y rester le temps d'un stage, j'y ai finalement passé cinq années, presque jour pour jour. J'ai pu y apprécier les discussions, techniques ou non, avec les ingénieurs qui se sont succédés sans jamais perdre en qualité : Alexeï, Quentin, Aydogan et Laurent. Merci et bravo aux permanents, présents et passés, de mener l'équipe avec une passion constante, sous la houlette efficace de Philippe puis Nicolas. J'ai été ravi de travailler avec des gens aussi brillants et enthousiastes et de partager avec certains d'entre eux des TP, belotes et boissons houblonnées. Merci donc à Luc, Iulian, Guillaume et Benjamin. Enfin, merci à nos assistantes, Laurence, Régine, Emmanuelle et Chantal, pour leur efficacité et leur patience.

Une équipe ne serait pas complète sans ses thésards, avec qui j'ai pu partager tant de bons moments, que ce soit en conférence, en soirée, ou tout simplement dans les bureaux. Merci donc à ceux qui étaient là avant moi, à Cuong et sa bonne humeur, Saliha, partenaire de

bureau, et Athanasia, avec qui j'ai partagé articles, projets, et activités en tout genre. Merci aussi à ceux qui sont arrivés après, a.k.a « *the darts guys* », Julien, Riad et Dimitris, dont le prosélytisme méditerranéen n'enlève rien à leur sympathie. Mention spéciale à Julien, demi, puis tiers-science, dont l'aide et les échanges sur l'implémentation du partage m'auront été plus que précieux et à Riad qui m'a sauvé la mise pour le papier EDBT.

L'équipe Cozy n'est également pas en reste ; ils sont à présent trop nombreux pour être tous cités, de neuf personnes à mon arrivée pour plus d'une trentaine à présent. De près ou de loin, beaucoup d'entre eux ont participés à cette thèse, de par leurs conseils techniques, leur vision produit, ou tout simplement leur bonne humeur. Le Cloud personnel grandit et je me sens chanceux de faire partie d'une équipe emplie de tant de compétences et de pétillance !

Un très grand merci aux relecteurs de la première et de la dernière heure qui auront aidés à rendre ce document le plus conforme possible à toutes les règles grammaticales en vigueur. Grâces soient ainsi rendues à Pauline, Simon, David et Sébastien. Un immense merci à Sylvie et Philippe pour leur travail de relecture titanesque, aussi exhaustif que rigoureux. Et merci quand même à tous ceux qui se sont proposés et ont manqué de temps ou de motivation, l'intention compte aussi !

Merci aussi à tous ceux qui ont été présents depuis le début de cette aventure et dont le soutien moral a été des plus précieux, dans les bons et mauvais moments ; nul doute qu'ils se reconnaitront. Une pensée particulière à Pauline, dont l'absence dans ces remerciements aurait été bien incongrue, tant elle a œuvré dans l'ombre (et parfois la lumière) et qui a su toujours être compréhensive lorsque ma présence se faisait des plus aléatoires, en particulier durant la dernière année.

Enfin, comme tout bon discours de remerciement, je remercie ma famille qui aura tenté jusqu'au bout de comprendre ce que je faisais de mes journées et dont le soutien n'aura pas failli !

Pour finir, je dédicace cette thèse à Nanou, qui n'oubliait jamais de me demander comment avançait mon travail – même si elle n'en comprenait pas un traitre mot – et s'exclamait de

surprise lorsque je lui montrais un aperçu des possibilités de l’informatique. Ces pages sont pour elle.

Table des matières

Liste des figures

Liste des tables

« Combien de fois, et suivant quel plan, la Police de la Pensée se branchait-elle sur une ligne individuelle quelconque, personne ne pouvait le savoir. On pouvait même imaginer qu'elle surveillait tout le monde, constamment. Mais de toute façon, elle pouvait mettre une prise sur votre ligne chaque fois qu'elle le désirait. On devait vivre, on vivait, car l'habitude devient indistincte, en admettant que tout son émis était entendu et que, sauf dans l'obscurité, tout mouvement était perçu. »

George Orwell, 1984

Chapitre 1 Introduction

Contexte

Le 6 juin 2013, le lanceur d'alerte Edward Snowden dévoile ce qui allait devenir un des plus grands scandales du 21ème siècle : **la surveillance massive** des individus par les agences gouvernementales américaines et leurs alliés [Snowden 2014] [Szoldra 2016]. Ces révélations furent le point de départ d'une prise de conscience mondiale autour des enjeux de **la vie privée sur Internet**.

Cette surveillance généralisée est notamment rendue possible grâce à la complicité – volontaire ou non – des géants d'Internet, que l'on résume parfois par les GAFAM, pour désigner Google, Amazon, Facebook, Apple et Microsoft. En effet, ces entreprises centralisent une volumétrie gigantesque de données provenant de leurs utilisateurs : historiques d'achats et de recherches Web, e-mails, discussions instantanées, photos, traces GPS, commandes vocales… Il suffit donc pour les agences gouvernementales d'espionner ces silos à données pour disposer d'une base d'information titanesque et prête à l'emploi. Pour donner une idée de cette omniprésence numérique : en 2016, Facebook compte 2 milliards d'utilisateurs actifs. Un dollar sur deux dépensé en ligne aux Etats-Unis l'est sur Amazon. Les systèmes d'exploitation d'Apple et Google équipent 90% des Smartphones, tandis que ceux de Microsoft sont installés sur 90% des ordinateurs personnels. Toujours en 2016, leur capitalisation boursière cumulée a atteint les 3278 milliards de dollars [NewCenter 2017], un montant supérieur au PIB de la France, estimé à 2629 milliards cette année-là [INSEE 2017]. Mais aussi puissantes soient-elles, ces entreprises ne sont pas seules dans la course aux données personnelles. Ces dernières années ont vu l'émergence asiatique des BATX, pour désigner Baidu, Alibaba, Tencent, Xiaomi [Greeven 2017] [Chelet 2016] et des NATU, Netflix, Airbnb, Tesla et Uber qui connaissent des croissances exponentielles [Haski 2015]. Sans oublier des géants de la donnée comme Acxiom, Oracle ou Palantir.

L'agrégation et l'analyse des données parfois extrêmement personnelles des individus permettent d'établir des **profils** de plus en plus précis, à des fins de **contrôle** pour les gouvernements ou **commerciales** pour les entreprises privées. En effet, les données personnelles sont devenues la ressource économique la plus profitable au monde [The Economist 2017], ce qui leur vaut l'appellation de « Pétrole du 21ème siècle ». Cette manne financière pousse les entreprises à recueillir le plus d'informations possibles sur les individus et à établir des profils toujours plus précis sur eux.

Google peut retracer tous les trajets effectués depuis des années[1], avec ou sans GPS activé [Liao 2017] et les croiser avec les données de ses nombreux services, agenda, contacts, e-mail, thermostat, vidéos, etc. Par défaut, le système d'exploitation Windows 10 récupère, entre autres, toutes les entrées clavier de l'utilisateur, ce qui a valu à Microsoft une mise en demeure de la CNIL [CNIL 2016]. Les dernières télévisions Samsung filment et écoutent les personnes en train de les regarder [McGoogan 2017], système dont aurait profité la CIA à des fins d'espionnage [Titcomb 2017]. Uber peut visualiser en temps réels les déplacements des utilisateurs de leur choix et s'amuse à débusquer des « Rides of Glory », version moderne des « Walk of Shame », c'est-à-dire des utilisateurs qui ont découché suite à une soirée [Uber 2014] [Wolfe 2016]. Leurs concurrents Chinois ne sont pas en reste : le navigateur de Baidu transmet toutes les données personnelles en clair [Knockel 2016], pendant que Tencent assume de livrer les données de ses utilisateurs au gouvernement Chinois, produites via son application WeChat, véritable couteau-suisse numérique [MoneyControl 2017]. Elle regroupe notamment des fonctionnalités de discussions instantanées, appels vidéo, géolocalisation, paiement, etc. Les données sont donc variées et sensibles, ce qui a notamment suscité la convoitise de Huawei [WSJ 2017].

Ces acteurs connaissent de mieux en mieux leurs utilisateurs et parfois même **mieux qu'eux-mêmes** : Facebook est par exemple capable de prédire des relations amoureuses sur son réseau [Diuk 2014] et d'anticiper leur rupture [Friggeri 2014]. De même, il est en mesure de détecter automatiquement des comportements suicidaires [Rosen 2017] ou terroristes [Bickert 2017], grâce à l'analyse des contenus aussi bien textuels (publications, conversations privées) que multimédias (photos, vidéos).

[1] https://www.google.com/maps/timeline?pb

Tous ces exemples, loin d'être exhaustifs, donnent un aperçu de l'intrusion dont sont capables ces entreprises. Il est intéressant de noter que la majorité d'entre eux sont communiqués par les entreprises elles-mêmes, qui voient l'analyse et l'exploitation des données personnelles comme une activité légitime [Johnson 2010] [Schmidt 2009]. Ce sur quoi elles ne communiquent pas est d'autant plus intéressant : dans un long rapport publié par l'organisation Cracked Labs [Christl 2017], on apprend l'existence de gigantesques bases de données, partagées et alimentées entre différents acteurs, où chaque individu dispose d'un identifiant unique, auquel est associé toutes sortes d'informations, comme les activités sur les réseaux sociaux, les transactions bancaires, les convictions politiques et religieuses, les données de santé, de localisation, d'humeur, etc. Ces bases de données permettent un **recoupement d'informations** sans précédent, phénomène qui s'accentue par les partenariats ou les rachats entre les géants d'Internet et des start-up spécialisées dans le profilage. Le pistage devient alors généralisé à l'ensemble d'Internet, mais aussi à la vie réelle : les bases clientèles d'enseignes traditionnelles sont incorporées, ainsi que les données qui proviennent de capteurs, de Smartphones et de divers autres objets connectés qui se multiplient, souvent proposés par ces mêmes entreprises qui y voient une nouvelle source de données exploitables.

A partir de ces données personnelles, des analyses comportementales sont menées afin **d'anticiper** ce que les individus vont faire ou tenter **d'influer** sur leur comportement, typiquement pour les inciter à acheter tel produit ou effectuer telle action. Dans une étude publiée avec des chercheurs des universités de Californie et de Cornell [Kramer 2014], Facebook a prouvé être en mesure d'influencer les humeurs de ses utilisateurs par les publications qu'il choisit de leur afficher. Une autre expérimentation menée sur 61 millions de personnes et publiée dans *Nature* démontre la capacité de Facebook à inciter significativement ses utilisateurs à aller voter [Bond 2012]. Cela fait écho au rôle du réseau social, certes indirect, dans l'élection de Donald Trump lors des élections américaines de 2016 [Entous 2017]. OkCupid, site de rencontres en ligne, est capable de manipuler les relations entre ses utilisateurs, par exemple en leur affichant de faux pourcentages de compatibilité [Rudder 2014].

Dans le même temps, de nombreux gouvernements cherchent à déceler tout comportement jugé déviant ou dangereux pour la société et mettent en place des arsenaux législatifs pour légaliser la **surveillance de masse** [Gellman 2005] [Investigatory Powers Act 2016] [LOI n° 2015-912 2015]. Certains pays, comme la France, externalisent même le traitement des

données à des sociétés privées qui font partie intégrante du système de pistage généralisé [Cohen-Grillet 2016].

Pour ne donner qu'un exemple de surveillance d'état, le gouvernement Chinois prévoit de mettre en place à l'horizon 2020 un score de citoyenneté, obtenu en croisant les informations fournies par des entreprises comme Alibaba et Tencent, avec leurs propres données, provenant par exemple de la vidéo-surveillance et des fichiers de police. Des algorithmes comportementaux seront régulièrement appliqués pour calculer le score de chaque personne en fonction de ses actions et de ses relations ; un individu clivant sera d'autant plus marginalisé que son score impactera négativement ceux de ses relations proches. Selon son score, un citoyen chinois aura par exemple plus ou moins de facilité à obtenir un prêt ou avoir des places dans une bonne école pour ses enfants [Botsman 2017]. Ce système est déjà mis en place sous une forme expérimentale, d'après différents témoignages [Hvistendahl 2017] [Rauhala 2018].

Alors que nos vies se numérisent de plus en plus et que le **volume des données personnelles explose** [Thirani 2017] [Marr 2015], il devient vital de pouvoir se protéger contre une surveillance et un contrôle qui se rapprochent dangereusement des dystopies de la littérature [Orwell 1949] [Elton 2008]. Au-delà du simple respect de la vie privée, auquel chaque individu peut et doit prétendre, les enjeux sous-jacents vont bien au-delà de la sphère individuelle et posent de vraies questions sur les impacts démocratiques et le libre-arbitre des individus.

Cette centralisation pose également de **graves problèmes de sécurité**. En effet, regrouper les données à un même endroit assure un gain important à un attaquant qui parviendrait à s'infiltrer dans le système. Le ratio coût / bénéfice d'une attaque est donc très intéressant, et il n'est pas surprenant de voir des exemples de fuites de données se multiplier ces dernières années. Pour n'en citer que certains : 68 millions de comptes Dropbox ont été touchés par une attaque en 2012 [Heim 2016], contre 117 millions pour LinkedIn et 360 millions pour MySpace en 2016 [Perez 2016]. Quant à la plateforme Yahoo!, l'intégralité des comptes ont été impactés par des attaques débutées en 2013, soit 3 milliards de comptes exposés [Newman 2017].

En 2015, le site de rencontres extra-conjugales Ashley Madison a vu sa base de données exposée publiquement sur le Web, avec des conséquences parfois dramatiques pour des personnes mariées ou homosexuelles [Segall 2015] [Cain 2015]. En 2017, l'agence

américaine de crédit Equifax annonce avoir été touchée par une attaque informatique qui a compromis plus de 145 millions de comptes. Les données fuitées incluent entre autres des adresses postales, numéros de sécurité sociale et informations bancaires. Cette même année, Rik Falkvinge, fondateur du Parti Pirate, annonce que le gouvernement Suédois a accidentellement fait fuiter une base de données regroupant des informations personnelles de ses concitoyens, ainsi que des données militaires extrêmement sensibles. Cela n'a pas été le fruit d'une attaque, mais d'une simple négligence : le ministère des transports a externalisé le traitement de ces données dans un Cloud d'IBM et communiqué les accès par e-mail en clair [Falkvinge 2017] [Chirgwin 2017].

Enfin, au-delà des négligences ou des attaques qui proviennent de tiers malveillants, des **abus** peuvent aussi se produire directement par les personnes en charge de **l'administration** du service. Des cas de harcèlement, *stalking* ou chantages par des employés de Google, Uber ou encore Apple ont notamment été reportés, et sans doute de nombreux autres furent passés sous silence [Hough 2010] [Hern 2016] [Gibbs 2017].

Pour faire face à cette situation, le World Economic Forum a formulé le besoin de voir émerger des plateformes de gestion de données personnelles permettant à chaque individu de collecter, gérer et **partager** ses données dans différents contextes, avec de réelles **garanties de sécurité et de protection de la vie privée** [WEF 2012]. Le principe du **Cloud personnel**, décrit dans [Abiteboul 2015], [Anciaux 2013] [de Montjoye 2012] et [Haddadi 2015], est précisément celui-là. Dans cette approche, chaque individu dispose de son propre espace personnel, sous son contrôle, sur lequel il peut stocker en toute confiance son patrimoine numérique et le gérer via des applications. Dans ce paradigme, le principe **d'*empowerment*** est essentiel. Traduit par « autonomisation » au Québec, il implique la responsabilisation des individus, afin qu'ils ne soient plus de simples utilisateurs dépendants du système, mais des acteurs à part entière de leur vie numérique.

Le Cloud personnel est **décentralisé** par nature, c'est-à-dire que l'utilisateur a le choix de l'hébergement et du fournisseur. C'est en opposition frontale aux approches traditionnelles, où le fournisseur de service impose son infrastructure centralisée à l'ensemble des individus qui n'ont alors plus aucun contrôle sur la façon dont leurs données sont stockées et utilisées, avec les risques d'abus que nous avons détaillés. La **gouvernance** change alors complètement : l'individu n'est plus prisonnier d'un service qui détient ses données et a la liberté d'en changer s'il n'en est pas satisfait : il peut aussi bien choisir un hébergeur commercial dans un *data-center* qu'un hébergement associatif qui met l'éthique au centre de

ses préoccupations, par exemple le collectif CHATONS[2]. En réalité, l'utilisateur peut même s'en priver complètement et décider d'héberger son patrimoine numérique à la maison, s'il en a les compétences. De plus, la décentralisation amène aussi une garantie supplémentaire de résilience : si un service centralisé devient inaccessible, suite à une panne, attaque, ou simplement une fermeture, l'ensemble des données de tous les utilisateurs deviennent injoignables et sont potentiellement perdues à jamais. Enfin, l'aspect décentralisé atténue grandement le risque d'être victime d'une attaque massive : le ratio coût / bénéfice d'une attaque est inversé par rapport à un système où les données de tout le monde sont regroupées à un même endroit.

Dans ce type d'environnement, le partage est particulièrement complexe à mettre en œuvre, car il est à l'intersection de nombreuses problématiques, dont nous parlerons au chapitre suivant : contrôle d'accès, identification et authentification des sujets, chiffrement, synchronisation, révocation, etc. L'enjeu est de pouvoir donner la possibilité aux individus de contrôler facilement quels tiers peuvent accéder à quelles données, ces tiers pouvant être aussi bien d'autres individus ou des services en ligne. A l'inverse des coffre-forts numériques qui se concentrent sur la sécurité des données au détriment des usages, le Cloud personnel fait la part belle aux applications et s'inscrit dans la dimension sociale et collaborative du Web : les individus cherchent constamment à se partager des ressources de toutes sortes, que ce soit dans leur sphère privée ou professionnelle. Or, cette dissémination ne va pas sans risques : les politiques de partage peuvent être contournées de diverses façons, voire être mal perçues par les usagers qui perdent alors le contrôle sur leurs propres données. Il est donc essentiel de disposer de moyens qui permettent de garantir la sécurité du partage des documents dans le Cloud personnel.

Ainsi, le but de cette thèse est de permettre aux propriétaires de redevenir maîtres de leurs données et de la façon dont elles sont disséminées sur Internet. Il s'agit d'un des principes fondateur de l'approche *Privacy-by-Design* [Cavoukian 2009] et va également dans le sens du nouveau règlement européen sur la protection des données qui entrera en application en mai 2018 [EU 2016]. Nous espérons que les travaux détaillés dans la suite de ce document contribueront à une gestion de données plus responsable et respectueuse de la vie privée, conduisant à une certaine forme **d'émancipation numérique** des individus.

2 https://chatons.org/

Contributions

Les contributions de cette thèse sont les suivantes :

1. Nous proposons une architecture *Privacy-by-Design*, dédiée au Cloud personnel, qui permet aux utilisateurs de partager leurs données avec des sujets en toute connaissance de cause et en disposant d'une sécurité tangible.

2. Nous définissons un modèle de partage adapté aux propriétés du Cloud personnel et qui ainsi met l'accent sur l'auto-administration, la gestion du dynamisme et le contrôle des effets du modèle par l'utilisateur du Cloud personnel.

3. Enfin, nous décrivons comment le modèle de partage a été implémenté dans la plateforme de Cloud personnel Cozy, comme une partie intégrante du protocole conçu pour disposer d'un système de partage complet.

Dans la première contribution, nous définissons une architecture de référence pour le Cloud personnel qui peut être déclinée en de multiples instances sécurisées. Des outils d'administration sont également détaillés afin que le propriétaire puisse avoir conscience, à tout moment, des effets concrets de sa politique de partage. Il peut ainsi facilement contrôler qu'elle est correctement appliquée. Dans la deuxième contribution, nous discutons d'un modèle de partage spécifique au Cloud personnel et ses propriétés. Nous discutons de la sémantique des différents opérateurs et montrons comment leur évaluation permet de matérialiser une politique de partage qui peut porter sur n'importe quel type de document ou sujet, tout en assurant leur contrôle par le propriétaire. Enfin, la dernière contribution présente la partie industrielle de cette thèse : la conception et l'implémentation d'un protocole de partage sécurisé dans la plateforme Cozy, dont le modèle décrit dans la deuxième contribution fait partie intégrante.

Les deux premières contributions ont chacune fait l'objet d'un article de conférence et d'une validation technique sous forme d'article de démonstration. Ainsi, l'architecture est présentée dans [Tran-Van, SWYSWYK: a privacy-by-design paradigm for personal information management systems. 2017] et validée dans [Tran-Van, SWYSWYK: A New Sharing Paradigm for the Personal Cloud. 2017]. La sémantique du modèle est quant à elle introduite dans [Tran-Van, A New Sharing Paradigm for the Personal Cloud. 2017] et validée dans

[Tran-Van, Reconciling Privacy and Data Sharing in a Smart and Connected Surrounding 2018]. La troisième contribution a été concrétisée par deux mises en production du protocole de partage dans le produit Cozy, sur deux versions différentes. La première a eu lieu au printemps 2016 et la seconde pendant l'été 2017. Le code source et la documentation sont accessibles en ligne, sous licence AGPL v3[3].

Plan

Le cœur de cette thèse est composé de trois parties principales.

Le Chapitre 2 introduit les travaux et concepts nécessaires à la compréhension des contributions de la thèse et à leur positionnement dans l'état de l'art. Nous dérivons également de cette analyse de l'existant la problématique qui servira de fil conducteur à cette thèse.

Les Chapitre 3 et Chapitre 4 exposent la contribution académique principale de la thèse : le modèle SWYSWYK (*Share What You can See with Whom You Know)* qui est introduit en deux parties. D'abord, la dimension architecturale du modèle, présentée au Chapitre 3, et ensuite, la sémantique du modèle, présentée au Chapitre 4. Notre plateforme expérimentale, sur laquelle des évaluations qualitatives et quantitatives ont été menées, y est également détaillée. De plus, deux démonstrateurs sont présentés, chacun servant à valider l'intérêt et la faisabilité technique de SWYSWYK pour permettre un partage sécurisé.

Le Chapitre 5 est consacré à la description du protocole de partage implémenté et mis en production dans la solution de Cloud personnel éditée par l'entreprise Cozy Cloud. Le protocole se veut suffisamment générique pour permettre différents types d'implémentation : nous détaillons à la fois les principes généraux et les choix technologiques. Nous présentons enfin trois cas d'usages qui exploitent le protocole et permettent un partage sécurisé de documents dans différents contextes du Cloud personnel.

Enfin, nous concluons dans le Chapitre 6, tout en explorant des perspectives futures prometteuses.

---

[3] https://www.gnu.org/licenses/agpl-3.0.html

Chapitre 2
Etat de l’art

Nous faisons ici un tour d’horizon des différentes méthodes et travaux liés au partage sécurisé dans le Cloud personnel. Nous nous concentrons en particulier sur le contrôle d’accès qui est une composante essentielle du partage. En effet, pouvoir exprimer quels sujets ont accès à quels documents revient à définir une politique de de contrôle d’accès. Nous verrons dans ce chapitre que de nombreux modèles de partage et d’administration existent, chacun avec leurs avantages et inconvénients.

Enfin, nous présenterons des projets concrets de Cloud personnels, dont l’un d’eux, Cozy, a fait l’objet d’une implémentation du système de partage présenté dans les chapitres suivants.

De ce tour d’horizon, nous tirerons des enseignements, afin de pouvoir en extraire une problématique qui sera notre fil conducteur tout au long de ce document.

Pour éviter toute confusion, nous appelons dans la suite de ce document *propriétaire* le possesseur d’un Cloud personnel, c'est-à-dire la personne unique qui peut se connecter sur son espace avec des droits complets : il peut ajouter des données, des applications, des appareils, etc. Nous appelons *sujets* les utilisateurs de Cloud personnels avec lesquels le propriétaire veut interagir et *objets* les ressources accédées.

2.1 Contrôle d'accès centralisé

Dans un système d'information, on considère traditionnellement la **confidentialité**, **l'intégrité** et la **disponibilité** des données comme les trois propriétés fondamentales à respecter pour en garantir la sécurité [Cherdantseva 2013]. On classifie ainsi les failles de sécurité sur les données en fonction de ces principes : l'observation non autorisée, la modification incorrecte et la non disponibilité [Bertino 2005].

Le contrôle d'accès désigne l'ensemble des techniques mises en œuvre pour restreindre l'accès à des ressources. Il se positionne donc à la fois sur le respect de la **confidentialité** et de l'**intégrité** des données.

L'application du contrôle d'accès, également appelée *enforcement*, est assurée par un module appelé **moniteur de référence** qui intercepte tous les accès aux données et détermine s'ils sont légitimes ou non. L'ensemble des droits et conditions appliqués par le moniteur de référence constitue la **politique de contrôle d'accès** qui peut être représentée et exprimée de multiples façons.

A noter que l'on distingue authentification et contrôle d'accès. Le moniteur de référence considère généralement qu'une demande d'accès est déjà correctement authentifiée lorsqu'il l'évalue [Sandhu 1994], bien que cela ne soit pas toujours vrai, typiquement dans les systèmes dits *credential-based*. Dans ces modèles, chaque utilisateur s'authentifie via un jeton qui contient les permissions.

Enfin, bien que le contrôle d'accès et l'authentification soient des composants essentiels, la sécurité d'un système d'information ne peut se résumer à cela. L'application de techniques de chiffrement, d'audit ou de protection contre les dénis de service [Sandhu 1994], [Bertino 2011] sont autant de moyens pour réduire les surfaces d'attaques. Cela vaut également pour le partage qui touche à de nombreux aspects de la sécurité du système d'information. Nous nous concentrons essentiellement ici sur la problématique du contrôle d'accès qui est le cœur de notre contribution, mais nous ne négligeons pas pour autant les autres composantes de la sécurité, sans y contribuer directement.

Nous détaillons ici quelques modèles de contrôle d'accès traditionnels, largement étudiés dans la littérature et utilisés à large échelle dans le monde industriel.

2.1.1 Contrôle d'accès discrétionnaire

Communément désigné par l'acronyme **DAC**, pour *Discretionary Access Control*, ce modèle permet d'associer l'identité d'un sujet à un ensemble d'autorisations sur des objets. Lorsqu'un sujet fait une requête sur un objet, l'action est accordée si et seulement si une autorisation le permet [Griffiths 1976].

La notion clé de DAC est l'aspect **discrétionnaire** qui se traduit par une délégation des droits : chaque utilisateur du système peut lui-même attribuer des droits à d'autres sujets sur les objets dont il possède des autorisations. Il ne peut en revanche pas attribuer de droits qu'il ne possède pas.

Dans la plupart des modèles DAC, les autorisations sont représentées sous forme de **listes de contrôle d'accès**, ou **ACL** pour *Access Control List*. Chaque élément d'une ACL représente une autorisation pour un sujet sur un objet, associé à un mode d'accès (lecture, écriture, etc). L'évaluation du contrôle d'accès peut alors se résumer à parcourir la ou les listes d'ACL et vérifier que l'action demandée a bien une autorisation enregistrée.

La grande majorité des systèmes d'exploitation (Unix, Windows, …) utilisent un modèle DAC pour leur contrôle d'accès : les utilisateurs peuvent ainsi transférer leurs droits à d'autres utilisateurs sur les données ou services qu'ils contrôlent sans avoir besoin d'une autorité centrale qui pilote le tout. Cet aspect discrétionnaire le rend flexible et adapté à de nombreux systèmes multi-utilisateurs. Cependant, il rend difficile le contrôle de la dissémination des données : une fois qu'un propriétaire accorde un accès pour un utilisateur sur un objet, il n'a plus aucun contrôle sur les futurs accès qui pourraient être créés. C'est précisément ce que les modèles de contrôle d'accès obligatoire cherchent à éviter.

2.1.2 Contrôle d'accès obligatoire

Le modèle *Mandatory Access Control* **(MAC)**, permet de définir des politiques d'accès par une classification prédéfinie des objets et des sujets. Avec le modèle DAC, il fait partie des deux grandes classes de contrôle d'accès et a inspiré de nombreux travaux.

Historiquement associé au gouvernement américain, ce modèle est apparu dans les années 1960 et avait pour objectif initial d'assurer la protection des données sensibles de l'armée, en

particulier contre des attaques par cheval de Troie[4], particulièrement dangereuses pour des systèmes discrétionnaires comme DAC où un poste infecté pourrait déléguer tous ses accès à un attaquant.

Dans le modèle [Bell 1976], à chaque objet est associé un niveau de sensibilité, et, à chaque sujet, un niveau d'autorisation. Les valeurs de ces niveaux sont définies dans des classes d'accès, constituées de deux éléments :

- Un label de sécurité, par exemple, 'Top Secret' (TS) ou 'Confidentiel' (C), où TS > C en termes de sécurité.
- Une catégorie, par exemple 'Aviation', ou 'Armée de terre'. Cela permet de connaître le contexte dans lequel s'exécute l'accès aux données et de définir des classes d'accès différentes selon les domaines.

Les deux propriétés centrales du contrôle d'accès sont : *no read up* et *no write down*. Cela signifie qu'un sujet ne peut lire un objet que si sa classe d'accès est supérieure à celle de l'objet et qu'il ne peut écrire un objet que si sa classe d'accès est inférieure.

Dans ce modèle très rigide, les utilisateurs ne peuvent en aucun cas attribuer des droits par eux-mêmes, même s'ils sont le « propriétaire » des données. Un administrateur central décide, à priori, de la politique d'accès et de la classification, ce qui évite que les utilisateurs aient la responsabilité du degré d'exposition des données auxquelles ils ont accès. Ce système est particulièrement justifié dans les organisations où l'accès aux données est extrêmement sensible, typiquement dans des contextes militaires ou des centrales nucléaires.

Il convient de préciser que l'approche MAC n'est pas forcément incompatible avec DAC. En effet, certains systèmes permettent de les combiner pour bénéficier du meilleur des deux mondes [IBM 2017] [Oracle 2017].

[4] https://fr.wikipedia.org/wiki/Cheval_de_Troie_(informatique)

2.1.3 Contrôle d'accès à base de rôles

Le coût de maintenance du contrôle d'accès peut rapidement devenir prohibitif dans les modèles précédemment cités. En particulier dans les systèmes industriels où l'accès aux objets peut se faire par des centaines ou milliers de sujets. La gestion de ces autorisations entraîne une explosion combinatoire qui ralentit l'évaluation du contrôle d'accès et rend complexe l'attribution ou la révocation de droits. De plus, ni la rigidité des modèles MAC, ni le manque de contrôle des modèles DAC ne sont réellement satisfaisants pour des systèmes industriels où les autorisations doivent pouvoir être déléguées, mais aussi contrôlées.

C'est en réponse à ces problématiques qu'est apparu le contrôle d'accès à base de rôles, appelé *Role-Based Access Control* **(RBAC)** [Sandhu 1996]. La Figure 1 illustre la mécanique RBAC : les sujets obtiennent des autorisations sur des ressources grâce à des rôles qui leur sont attribués, eux-mêmes associés à un ensemble de permissions. Comme il y a normalement nettement moins de rôles que d'utilisateurs et de ressources, cela simplifie grandement la gestion des autorisations. Différents utilisateurs peuvent jouer le même rôle, par exemple 'manager' et un même utilisateur peut jouer différents rôles, ce qui permet de mutualiser les autorisations en fonction des tâches assignées aux sujets. De plus, un système de hiérarchie entre rôles simplifie son administration : chaque rôle enfant hérite des permissions de ses parents, par exemple, le rôle de 'manager informatique' peut hériter du rôle 'manager' et de ses autorisations, tout en bénéficiant de permissions spécifiques à son rôle enfant. Cela évite d'avoir à redéfinir des autorisations pour chaque rôle et permet aux industriels d'avoir un modèle mental qui fait écho à leurs organisations hiérarchiques.

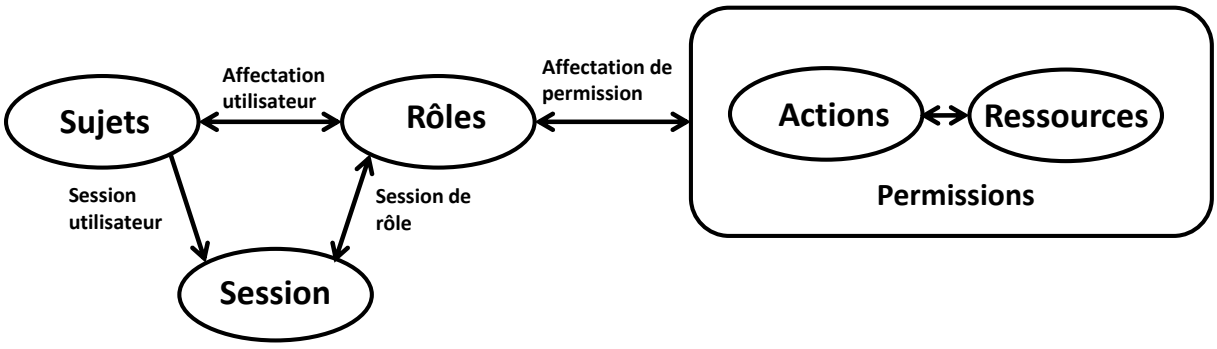


Figure 1. Modèle RBAC.

Ce mécanisme de contrôle d'accès se veut être une alternative crédible à DAC et MAC, en particulier pour les systèmes évolutifs de grande ampleur. Néanmoins, il est important de

signaler que RBAC ne s'oppose pas nécessairement à ces modèles. Grâce à sa flexibilité et au fait qu'il n'impose pas une unique façon d'appliquer le contrôle d'accès, il est possible d'implémenter DAC [Sandhu 1998] et simuler MAC [Osborn 2000] dans RBAC.

Enfin, bien qu'originellement conçu pour des systèmes à large échelle pouvant accueillir de nombreux utilisateurs, il est apparu que la maintenance du contrôle d'accès peut devenir extrêmement complexe pour gérer des centaines de milliers voire des millions d'utilisateurs. En particulier, l'assignation manuelle des rôles peut devenir cauchemardesque. Pour pallier ce problème d'échelle, des solutions ont été proposées pour automatiquement attribuer des rôles aux utilisateurs, grâce à l'utilisation de règles évaluées sur les attributs des utilisateurs [Al-Kahtani 2002] [Al-Kahtani 2003]. Cela a ouvert la voie à une nouvelle famille de contrôle d'accès qui allait pleinement tirer parti de l'émergence des services web et de leur besoin de souplesse et de dynamisme.

2.1.4 Contrôle d'accès à base d'attributs

Le contrôle d'accès à base d'attributs, ou A*ttribute-Based Access Control* **(ABAC)** est apparu dans les années 2000 [Yuan 2005] [Shen 2006] pour répondre à certaines limitations des contrôles d'accès précédemment cités, notamment vis-à-vis des services web. Ces derniers sont dynamiques par nature et ne sont pas adaptés à des systèmes comme MAC ou RBAC où les autorisations reposent sur des identités ou rôles prédéfinis. De plus, ils requièrent une sémantique plus puissante et plus fine afin d'exprimer l'accès aux données.

L'affectation des permissions dans ABAC repose sur les attributs qui peuvent être définis sur les sujets, les ressources ou l'environnement. Un sujet peut aussi bien être un utilisateur ou une application, et une ressource peut être aussi bien une donnée qu'un service web. L'environnement représente le contexte où s'applique le contrôle d'accès, et peut porter sur la date, la nature du réseau, etc.

Par exemple, la règle suivante permet d'exprimer l'accès à mon service en ligne de musique pour tout sujet qui fait partie de mon groupe d'ami :

can_access (s, r, e) ⟵ Group(s) = 'Friends' ∧ ServiceName(r) = 'MusicPlayer'

Ici, la règle porte à la fois sur les attributs des sujets *s* (par l'appartenance au groupe '*Friends'*) et sur l'attribut du nom du service qui est une ressource *r*. ABAC permet une grande souplesse d'expression qui n'existe pas dans RBAC et évite ainsi la définition et gestion de multiples rôles, ce qui devient inévitable lorsque les services et contextes se multiplient.

L'application du contrôle d'accès se fait dans une architecture représentée sur la Figure 2. Deux entités, appelées *Policy Decision Point* (PDP) et *Policy Enforcement Point* (PEP), sont en charge de l'application du contrôle d'accès et jouent le rôle de moniteur de référence. Lorsqu'un sujet cherche à accéder à une ressource, la requête est interceptée par le PEP qui est le point d'entrée incontournable du système. Il transmet alors la requête au PDP qui récupère les règles d'accès et les attributs concernés afin de déterminer si la demande d'accès est légitime. Il envoie sa décision au PEP qui accorde ou refuse l'accès à la ressource en fonction de la réponse.

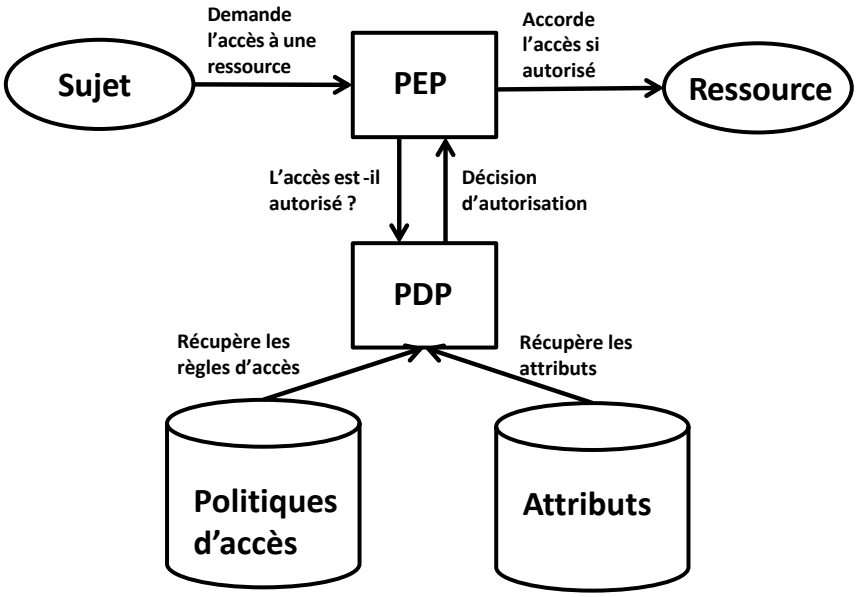


Figure 2. Architecture ABAC.

ABAC permet ainsi d'exprimer de nombreuses règles d'accès avec une granularité fine, au niveau des attributs. Il est néanmoins agnostique concernant le format des règles d'accès. Il est donc possible d'utiliser des langages de règles comme XACML [Anderson 2005] ou

EPAL [Ashley 2003] qui permettent d'exprimer une grande variété de politiques de contrôle d'accès, formatées en XML. Ces langages mettent l'accent sur l'interopérabilité, afin que des systèmes hétérogènes soient capables de se transmettre des politiques d'accès et d'éviter la redéfinition des formats de règles *ad-hoc* pour chaque usage. Bien que EPAL ait été défini en premier via une recommandation W3C, c'est XACML qui s'est finalement imposé, notamment dans l'industrie [Anderson 2005].

Bien que plus souple que les approches précédemment citées, ABAC peut néanmoins rendre la compréhension et l'administration du contrôle d'accès assez complexe : il est difficile, à partir d'un ensemble de règles plus ou moins élaborées, d'appréhender les implications concrètes d'une telle politique et de savoir exactement qui peut accéder à quoi, dans quelles conditions. L'évaluation à la volée des requêtes pour déterminer leur légitimité par rapport à un ensemble de prédicats sur des attributs est hors de portée de la compréhension de la plupart des individus et n'est ainsi pas compatible avec la propriété **d'*empowerment*** du Cloud personnel.

2.1.5 Conclusion

Tous les modèles cités ci-dessus sont largement étudiés dans la littérature et ont fait l'objet de nombreuses implémentations. Ils sont ainsi répandus dans les bases de données et pour certains nativement intégrés au langage de requête SQL [ISO 2016]. Et de fait, ils sont parfaitement adaptés à des environnements centralisés, mais partagent les caractéristiques suivantes qui les rendent peu adaptés pour l'usage du Cloud personnel :

1. L'administration du contrôle d'accès est une tâche centrale et complexe, généralement assurée par des experts en sécurité. La gestion des politiques d'accès, des utilisateurs, voire des rôles ou des attributs, n'est pas une mince affaire et hors de portée d'individus non experts, en particulier lorsque la politique repose sur un ensemble de règles exprimées dans un langage spécifique, et évaluées à la volée lorsque des requêtes arrivent.

2. La logique applicative, les utilisateurs et leurs rôles sont généralement identifiés à priori, c'est-à-dire lors de la conception même du système d'information. En effet, la politique de

contrôle d'accès fait souvent partie intégrante de la définition du schéma de la base de données et doit être pensée dès le début. Même ABAC, considéré comme plus flexible, nécessite d'avoir des attributs sur des schémas de données unifiées pour évaluer correctement le contrôle d'accès. Or, dans le Cloud personnel, des applications peuvent être ajoutées dynamiquement par le propriétaire pour répondre à des usages précis et non prévisibles. Chacune d'entre elles peut avoir ses propres structures et sa propre vision de comment les données doivent être représentées pour implémenter ses usages.

Cela rend ces modèles peu adaptés à l'usage du Cloud personnel où il est impossible de prédire quelles applications, modèles de données et utilisateurs seront définis. De plus, si une plateforme de Cloud personnel proposait des politiques prédéfinies que le propriétaire pourrait accepter ou refuser, cela irait à l'encontre du principe **d'*empowerment*** de l'utilisateur qui est un des concepts clés du Cloud personnel. Afin de répondre à ces limitations, des modèles de contrôle d'accès flexibles et centrés autour de l'utilisateur ont été spécifiquement conçus pour supporter la nature décentralisée du Web, et par extension, du Cloud personnel.

2.2 Contrôle d'accès décentralisé

Les modèles précédemment exposés sont adaptés aux structures centralisées où des administrateurs sont en charge de la supervision et de l'application correcte des politiques d'accès. Les serveurs où elles s'appliquent sont considérés comme étant de confiance, et le cycle de vie de la donnée doit y être maîtrisé de bout en bout. Or, l'émergence ces dernières années d'applications centrées sur l'utilisateur, ou *user-centric,* a nécessité le développement de nouveaux modèles où l'utilisateur redevient maître de ses données et de leur dissémination, tout en lui donnant des garanties sur le respect de sa privée. Le Cloud personnel s'inscrit dans ce mouvement.

Cependant, cette décentralisation apporte de nouveaux challenges, comme la gestion des sujets qui ne sont maintenant plus regroupés dans une base de données unique, mais dispersés entre serveurs personnels hétérogènes, chacun avec leur propre représentation des données. La découverte de ces sujets, leur identification et **authentification** sont des challenges majeurs. Le glissement des responsabilités vers l'utilisateur n'est également pas

sans contraintes : comment assurer au propriétaire des données que sa politique est **correctement appliquée**, et ce, même si son environnement d'exécution n'est pas forcément de confiance ? Car ici, les serveurs de Cloud utilisés ne sont à priori pas maîtrisés par l'utilisateur. Il ne peut donc avoir qu'une confiance très limitée en eux et s'expose à des attaques de **confidentialité** s'ils se révèlent malhonnêtes ou mal sécurisés. Le **chiffrement** devient donc obligatoire pour ne pas risquer d'avoir toute sa vie numérique exposée. Enfin, l'**administration** du contrôle d'accès est une tâche cruciale qui échoit à présent au propriétaire du Cloud personnel. Il faut donc lui donner les outils adéquats pour facilement pouvoir exprimer ses partages et les maintenir.

2.2.1 Authentification

OpenID

[Seong 2010] et [Yeung 2009] proposent des mécanismes de partage qui authentifient les sujets par **OpenID** [Recordon 2006]. Dans ce protocole, un utilisateur s'authentifie auprès de services tiers en utilisant une identité souscrite chez une autorité n'ayant potentiellement aucun lien avec les services accédés.

Nous illustrons ici l'authentification d'un utilisateur Jon souhaitant se connecter à son application web favorite *Knowthing*. Afin de ne pas alourdir la lecture, certains détails techniques sont volontairement éclipsés :

1. Jon va sur la page de connexion de *Knowthing* qui joue le rôle de *Relying Party* (RP), et entre son OpenID *jon.snow*, préalablement souscrit après d'un fournisseur d'identité (IdP).

2. L'identifiant OpenID est une URI qui pointe vers un document indiquant l'IdP de Jon Snow, *stark.corp*. Le RP redirige alors ce dernier vers l'IdP concerné.

3. L'IdP demande à Jon Snow de s'authentifier grâce au mot passe fournit lors de la création de son OpenID. S'il le connaît, l'authentification est réussie et Jon Snow est redirigé de nouveau vers le RP dont il peut à présent utiliser le service, grâce à un échange de jetons entre l'IdP et le RP.

Ce protocole a grandement contribué à populariser l'authentification décentralisé et a connu un intérêt croissant depuis sa première publication au milieu des années 2000 [Fitzpatrick 2005]. Malheureusement, la nécessité d'utiliser des autorités centrales de confiance, incarnées par les fournisseurs d'identités, laisse les utilisateurs vulnérables à des fermetures de services, symbolisées par celle de MyOpenID en 2014 [TheNextWeb 2014]. De plus, diverses faiblesses en termes de sécurité [Van Delft 2010] [Sun 2012] ont contribué à une perte de confiance progressive dans le protocole.

Web Of Trust

Des approches totalement décentralisées, c'est-à-dire ne reposant sur aucune autorité centrale, ont été étudiées dans la littérature. Beaucoup d'entre elles s'inspirent du principe de ***Web of Trust*** **(WoT),** introduit au début des années 90 par PGP [Zimmermann 1995]. Originellement, le WoT permet à des utilisateurs de s'échanger des messages chiffrés, tout en ayant des garanties sur leur intégrité et l'identité de l'émetteur, et ce, grâce à un réseau de confiance établi entre les participants.

On suppose ici que chaque individu possède un couple de clés privée et publique qui servent à chiffrer et signer les données. Le **chiffrement** sert à assurer la **confidentialité** des données, tandis que la signature assure leur **intégrité** et sert ici également à **authentifier** l'émetteur. La clé publique est, comme son nom l'indique, accessible à tous, et sert à chiffrer les données et vérifier les signatures. A l'inverse, la clé privée ne doit être connue que de son propriétaire et lui sert à déchiffrer et signer les données.

Supposons par exemple que l'utilisateur Jon souhaite communiquer un message privé à son amie Ygritte. Les étapes suivantes, illustrées par la Figure 3, ont lieu :

1. Jon génère une clé de session temporaire et la chiffre avec la clé publique d'Ygritte.

2. Jon chiffre son message avec la clé de session et en génère une empreinte avec une fonction de hachage. Puis, il chiffre l'empreinte avec sa clé privée pour obtenir la signature du message.

3. Jon réunit le message chiffré, la clé de session chiffrée et la signature du message, et envoie le tout à Ygritte, via un canal potentiellement non protégé.

4. Ygritte déchiffre la clé de session grâce à sa clé privée.

5. Ygritte déchiffre le message grâce à la clé de session obtenue.

6. Ygritte vérifie la validité de la signature en la déchiffrant avec la clé publique de Jon et compare le résultat avec l'empreinte du message déchiffré. Si elle obtient la même chose, tout s'est bien déroulé.

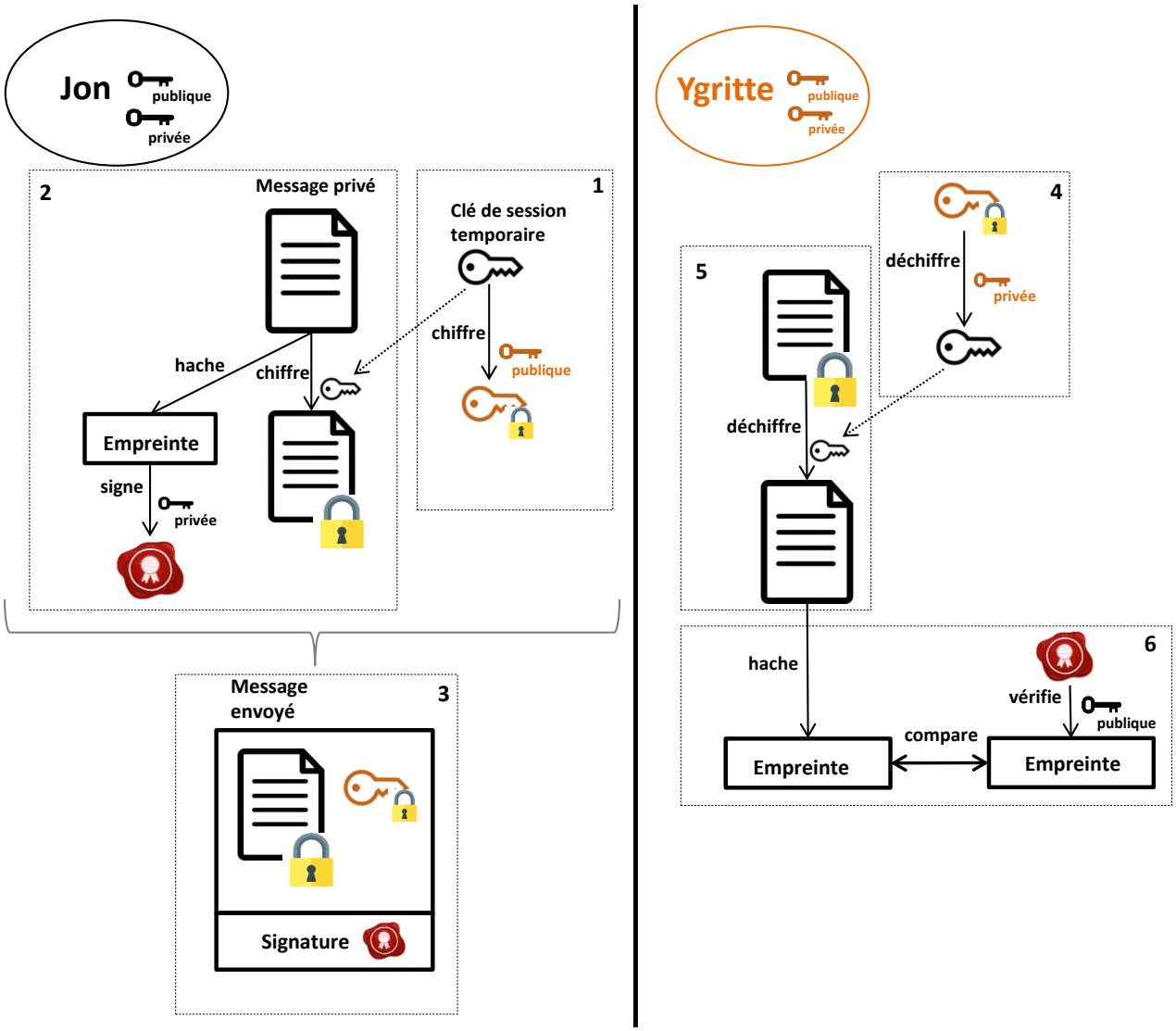


Figure 3. Envoi d'un message PGP de Jon à Ygritte.

Toute la problématique du WoT réside dans la **confiance** que peut accorder Ygritte vis-à-vis de l'authenticité de la clé publique de Jon, sans disposer d'une autorité centrale de confiance.

En effet, un utilisateur malveillant aurait pu se faire passer pour Jon depuis le début en publiant sa propre clé publique sous l'identité de ce dernier. Sans une vérification rigoureuse, Ygritte échangerait des messages secrets avec un usurpateur et n'y verrait que du feu. Une solution simple et robuste pour prévenir cela est que Jon et Ygritte échangent préalablement leurs clés publiques lors d'une rencontre physique. A chaque message reçu, la vérification de la signature leur assure la légitimité du message. Cependant, devoir rencontrer chaque personne physiquement est un procédé peu commode à l'échelle du Web.

Pour pallier cela, le *Web of Trust* introduit le concept de **confiance transitive**, illustrée dans la Figure 4. Dans ce système, les participants du réseau signent les clés publiques des personnes dont elles sont sûres de l'authenticité et leur attribuent un niveau de confiance dans leur propre habilité à signer des clés. Par exemple, supposons que Jon signe les clés de Ygritte, Tyrion et Ned qu'il a tous rencontrés en personne. Il a donc entièrement confiance dans l'authenticité de leurs clés publiques et peut communiquer avec eux sereinement. Il fait également entièrement confiance à Ygritte dans le fait qu'elle ne signe que les clés dont elle est sûre de l'authenticité. Comme cette dernière a signé la clé publique de Mance, Jon attribue transitivement sa confiance dans sa clé publique. En revanche, il n'a que moyennement confiance en Tyrion et Ned dans leur aptitude à signer des clés et à ne pas se laisser usurper. Il ne fait ainsi que peu confiance à la clé de Petyr, signée par Tyrion. Néanmoins, il décide de faire confiance à la clé de Robert, signée par Tyrion et Ned et par de nombreuses autres personnes que Jon ne connaît pas forcément. Cette popularité lui inspire suffisamment confiance pour valider sa clé, mais ne connaissant pas du tout Robert, il décide de n'accorder aucune confiance dans les clés signées par ce dernier.

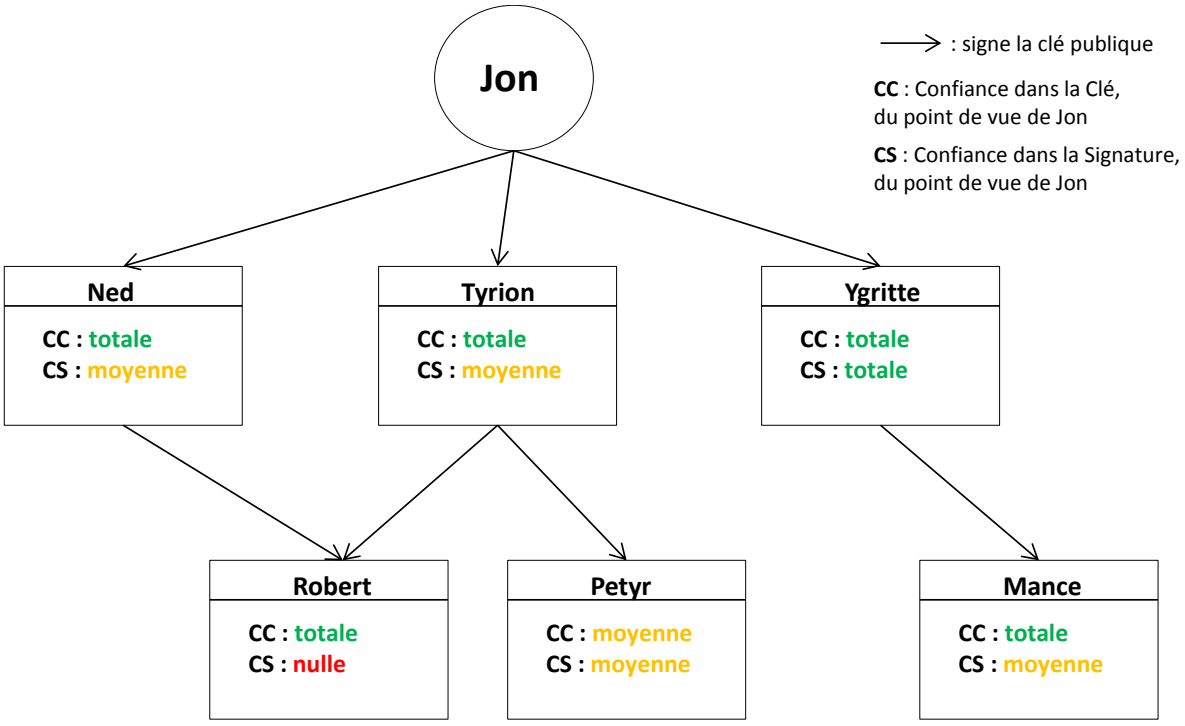


Figure 4. Graphe du *Web of Trust* de Jon.

La solidité du WoT réside ainsi dans la capacité des individus à signer les clés de leurs contacts et à déléguer leur confiance dans certains. Comme mentionné au début de cette section, de nombreux travaux s'inspirent des principes du WoT, permettant aux utilisateurs de s'authentifier entre eux sans autorité centralisée.

C'est le cas de Lockr, [Tootoonchian 2009] qui constate que partager des données via des plateformes centralisées et fermées rend l'utilisateur dépendant de leur contrôle d'accès et peut difficilement, ou pas du tout, partager avec ses contacts qui utilisent d'autres plateformes. Dans Lockr, les utilisateurs partagent leurs données en créant des ACL qui indiquent la relation nécessaire pour accéder à l'objet. Une relation, par exemple « famille » est composée d'un ensemble d'utilisateurs où chaque membre de la relation possède la clé publique des autres membres et se signe mutuellement leurs clés. Un lien entre deux membres est prouvé par une attestation sociale qui contient la clé publique, le libellé de la relation et une clé de relation partagée par tous les membres et la signature de l'attestation. Pour accéder à un objet partagé par Ned, Jon lui envoie son attestation chiffrée par la clé de relation « famille ». Ned la déchiffre, vérifie la signature et accorde l'accès si la relation et l'objet rentrent bien dans une ACL.

[Van Kleek 2012] présente WebBox, un système qui permet également aux utilisateurs de partager des données entre eux de façon décentralisée. Chaque utilisateur du système dispose d'un serveur personnel sur lequel il héberge ses données, ainsi qu'un profil public, accessible via une URI[5] sémantique, appelée **WebID** [Inkster 2017]. Outre des informations sur l'identité du propriétaire (nom, prénom, e-mail, etc), ce profil contient sa clé publique et la liste de ses contacts, au format FOAF[6]. Il n'y a ici pas besoin de signer la clé publique par d'autres tiers comme dans le WoT originel car elle est liée au profil public de l'utilisateur référencé par le WebID. Le propriétaire signe sa clé avec le contenu de son profil public et son WebID ; il devient dès lors beaucoup plus compliqué pour un tiers malveillant d'usurper une identité, tant que les contacts vérifient correctement la signature et l'URI. Dans WebBox, toutes les permissions sont représentées par des ACL, sous la forme de triplets RDF[7] <*sujet, prédicat, objet*>, où le sujet est représenté par son WebID, le prédicat correspond à l'action HTTP qu'il peut effectuer sur l'objet (GET, POST, PUT, etc), et l'objet est la ressource accordée, également sous la forme d'une URI sémantique. Toutes les données sont ainsi représentées en RDF et accessibles via le langage de requête SPARQL[8].

Outre l'aspect décentralisé, le point commun de ces travaux est qu'ils requièrent de définir manuellement les autorisations pour chaque sujet (ou relation) et chaque ressource, ce qui peut rapidement devenir fastidieux et propice aux erreurs au fur et à mesure que le nombre de partages grandit. Pire encore, un système certes sécurisé mais avec une mauvaise expérience utilisateur a de grandes chances de n'être que peu utilisé, voire pas du tout. Malheureusement, OpenID est aujourd'hui presque abandonné et les principes de *Web of Trust* sont difficilement compréhensibles pour les utilisateurs non techniques : la confiance transitive est une notion loin d'être intuitive, et les travaux qui en découlent supposent un niveau d'expertise élevé de la part du propriétaire qui a la charge de vérifier l'authenticité d'URI ou transmettre des secrets cryptographiques.

---

[5] https://www.ietf.org/rfc/rfc3986.txt

[6] http://xmlns.com/foaf/spec/

[7] https://www.w3.org/TR/PR-rdf-syntax/

[8] https://www.w3.org/TR/rdf-sparql-query/

2.2.2 Chiffrement

Dans les approches traditionnelles détaillées en section 2.1, l'*enforcement* est la responsabilité des administrateurs du système d'information qui disposent d'outils et de compétences spécifiques pour les aider dans cette tâche. Cela est beaucoup plus difficile à faire dans les approches décentralisées et centrées autour de l'utilisateur, vu que ce dernier n'a souvent pas la main sur le serveur de Cloud et n'a de toute façon pas la maîtrise nécessaire pour contrôler ce qui s'y passe. Or, les fournisseurs de service peuvent se révéler peu scrupuleux et les attaques de **confidentialités** y sont nombreuses[9,10]. Une façon de s'en protéger est de rendre l'application du contrôle d'accès indissociable du chiffrement.

[Ali 2017], [Thilakanathan 2014] et [Wang 2016] se concentrent sur la protection des données dans des Cloud centralisés où peu voire aucune confiance ne peut être accordée dans le serveur. Malgré le stockage centralisé des données, le contrôle d'accès reste décentralisé car il est défini et maintenu depuis le ou les appareils du propriétaire. Les données sont ainsi gardées chiffrées dans Cloud, tout en permettant à l'utilisateur de définir des règles de partage dessus sans les exposer au fournisseur.

[Thilakanathan 2014] et [Wang 2016] proposent d'utiliser du chiffrement *Attribute-Based Encryption* **(ABE)** [Sahai 2005] pour protéger les documents et y accéder sur la base d'un contrôle d'accès par attributs. Le propriétaire place des tags sur ses documents, par exemple *type='Medical'* ou *date='2017'*, et les utilise pour exprimer sa politique de partage. Typiquement, pour partager tous les documents cardiologiques d'avant 2017, une règle pourrait être exprimée comme ceci : *(type='Cardio' AND date < 2017)*. Pour chaque règle, une clé de déchiffrement est communiquée aux sujets concernés qui contient l'expression de la règle et leur permet de déchiffrer les documents qui rentrent dans son évaluation. Persona [Baden 2009] utilise également du chiffrement ABE, mais appliqué aux réseaux sociaux. Le propriétaire crée des groupes via des attributs ("famille", "rock band", etc) et leur distribue les

[9] http://breachlevelindex.com/

[10] https://haveibeenpwned.com/

clés de chiffrement associées, chiffrées avec leur clé publique. Chaque groupe peut accéder à certaines ressources, spécifiées par des ACL, produites et stockées dans des applications du réseau. Chaque application peut définir sa propre sémantique de contrôle d'accès, par exemple *publier* pour une application de micro-blogging ou *chatter* pour une application de messagerie. Chaque application dispose d'une *home* centralisée sur laquelle les données sont stockées chiffrées et les permissions sont appliquées. Les utilisateurs doivent donc avoir confiance dans leurs applications pour correctement appliquer leur politique d'accès.

Une difficulté classique d'ABE est la révocation. Si une clé doit être révoquée, toutes les données concernées doivent être rechiffrées et les nouvelles clés communiquées aux destinataires. Si cette opération est effectuée côté serveur, les données seront exposées en clair ; si elle est faite côté client la lourdeur du rechiffrement risque d'impacter négativement l'appareil de l'utilisateur. Le chiffrement homomorphique par clé [Boneh 2013] est une technique alternative proposée par [Thilakanathan 2014] pour effectuer le rechiffrement depuis le serveur sans rien exposer en clair. Cependant, le coût d'une telle approche peut s'avérer rédhibitoire, et ce, alors que le chiffrement ABE introduit un surcoût 100 à 1 000 fois supérieur à du chiffrement asymétrique plus traditionnel comme RSA[11] [Baden 2009].

Outre les performances, une faiblesse de ces travaux à base de chiffrement ABE est que les attributs ne sont pas protégés d'une attaque de **confidentialité** sur le serveur. Cela signifie qu'un observateur extérieur pourrait inférer les politiques de partage et déterminer ce qui est partagé, avec qui. Des pistes à base de chiffrement par prédicat [Katz 2008] pourraient adresser ce problème, mais elles impactent elles aussi considérablement les performances.

Dans [Ali 2017], la confiance est déportée dans un serveur cryptographique, en charge des opérations de chiffrement et de l'évaluation du contrôle d'accès. Contrairement au serveur de Cloud, le serveur cryptographique doit disposer de toute la confiance de l'utilisateur. Malheureusement, l'honnêteté et la robustesse du serveur n'est pas vérifiable par le propriétaire qui doit aveuglement faire confiance dans l'application correcte du chiffrement et du contrôle d'accès.

[11] https://fr.wikipedia.org/wiki/Chiffrement_RSA

Une autre façon possible pour partager des données chiffrées depuis un serveur de non-confiance est l'utilisation d'un proxy de rechiffrement, ou *proxy re-encryption* (PRE), discuté dans [Xu 2012] et [Canard 2016]. Là encore, les données sont gardées stockées sur un Cloud centralisé. Pour chaque partage avec un sujet, le propriétaire génère une clé de rechiffrement et l'envoie au proxy, afin qu'il rechiffre les données concernées, de façon à ce que le sujet puisse les déchiffrer, sans que le proxy n'ait jamais accès au clair. Contrairement aux approches ABE, le proxy ne dispose d'aucune information, si ce n'est savoir avec quels sujets le propriétaire communique.

Enfin, [Yuan 2015] et [Guha 2008] préfèrent utiliser des stratégies d'obfuscation, afin de pouvoir utiliser des plateformes de partage et de réseaux sociaux existantes, de façon transparente. Ces deux systèmes supposent que les clés sont préalablement échangées entre les propriétaires et leurs sujets par des canaux sécurisés. [Yuan 2015] implémente des algorithmes de brouillage de photos à des niveaux divers selon leur degré de sensibilité. La visualisation en clair n'est possible que si le destinataire dispose des clés de déchiffrement et peut n'être que partielle selon les préférences du propriétaire. Il peut par exemple décider qu'un sujet ne puisse voir en clair que son visage, sur une photo de groupe. [Guha 2008] utilise des techniques de substitution des données sur des plateformes de réseaux sociaux où le respect de la vie privée n'est pas garanti, comme Facebook. Or, ces plateformes interdisent généralement l'obfuscation dans leurs conditions d'utilisations et peuvent aller jusqu'au bannissement en cas d'infraction. Cela s'explique à la fois pour des raisons d'expérience utilisateur, pour éviter d'avoir du contenu illisible dans les actualités, mais surtout pour des raisons commerciales, vu que leur *business model* se repose en très grande partie sur la collecte des données et le *profiling* des utilisateurs. Dans [Guha 2008], les données privées d'un propriétaire sont ainsi aléatoirement remplacées par des données provenant d'autres personnes. Seuls les contacts autorisés peuvent visualiser les données réelles grâce à leurs clés de déchiffrement, ce qui est réalisé automatiquement grâce à une extension Web dédiée. Malheureusement, cela rend la mobilité compliquée car il devient impossible d'utiliser la plateforme depuis des appareils sans l'extension, en particulier sur mobile.

Dans tous ces travaux, les politiques de contrôle d'accès sont liées au chiffrement qui protège les données contre des attaques de **confidentialité,** mais réduit inévitablement les usages pouvant être effectués. De plus, ces approches nécessitent dans tous les cas que le

propriétaire définisse manuellement qui peut accéder à quelle donnée, au cas par cas. Or, les utilisateurs n'ont pas pour vocation de fournir un effort cognitif démesuré pour partager leurs documents avec leur famille, leurs amis ou collègues. Un outil simple mais peu sécurisé et/ou peu respectueux de la vie privée aura certainement de bien meilleures chances d'adoption qu'un outil sécurisé mais complexe à utiliser et comprendre. De même, les performances sont inévitablement dégradées, alors qu'elles sont également un point critique pour l'adoption ou non d'un service. Idéalement, la latence doit rester en dessous du seuil de perception humain [Nielsen 1993] pour que l'utilisateur ait une impression de fluidité.

Finalement, la lourdeur et la complexité du contrôle d'accès découragent les utilisateurs qui finissent par le voir comme un fardeau. Ils se retrouvent ainsi la plupart du temps à définir des politiques d'accès bien plus permissives que ce qu'ils ne le souhaiteraient, parfois même sans en avoir conscience [Liu 2011] [Mazurek 2010] et à utiliser des services connus pour n'avoir que peu de scrupules quant au respect de la vie privée. Afin que le Cloud personnel puisse s'imposer comme une alternative crédible, il est essentiel de remettre l'individu au centre du contrôle et de la décision, tout en lui fournissant les outils adéquats pour qu'il puisse s'acquitter de ces tâches le plus intuitivement possible, tout en préservant une expérience utilisateur fluide.

2.2.3 Administration centrée sur l'utilisateur

Comme dit en introduction, l'importance du contrôle d'accès est plus grande que jamais, alors que les usages numériques se diversifient et que le partage sur nos données se démocratise, que ce soit pour échanger ses photos de chats, transmettre ses derniers relevés de paie ou communiquer sa géolocalisation. Mais dans le même temps, **l'administration du contrôle d'accès** est une tâche qui rebute les utilisateurs, la plupart ne souhaitant y consacrer ni le temps, ni l'effort cognitif que requièrent habituellement des outils puissants mais non adaptés au grand public.

Dans cette section, nous faisons un tour d'horizon de travaux qui visent à simplifier l'expression, la maintenance et la compréhension du partage dans des systèmes décentralisés et respectueux de la vie privée. Comme vu précédemment, la décentralisation

amène un nouveau mode de gouvernance et de nouvelles problématiques sur l'administration des politiques de partage, tâche traditionnellement réservée à des experts sur des plateformes centralisées.

Administration des sujets

Lorsqu'un propriétaire souhaite partager ses données personnelles, il a généralement le choix entre les rendre publiques, c'est-à-dire partager sans discrimination avec le monde entier, ou limiter l'accès à certains sujets bien définis. Ce deuxième cas, probablement le plus courant, nécessite de pouvoir contrôler l'accès des individus autorisés mais aussi de pouvoir l'exprimer simplement. Typiquement, il est assez fastidieux de devoir entrer manuellement l'adresse e-mail de chaque destinataire, pour chaque partage. De plus, chaque service a tendance à disposer de sa propre définition des sujets, ce qui amène inévitablement une fragmentation des contacts entre ces différents silos non interopérables. Nous recensons ici des travaux qui cherchent à simplifier l'administration des sujets et éviter au propriétaire d'avoir à définir manuellement qui sont les destinataires de chaque partage, sur la base d'identités propres à chaque système.

Dans [Carminati 2006], les sujets se voient attribués des accès en fonction de la nature de leurs relations avec le propriétaire, représentées par un graphe social exprimé en FOAF. Les propriétés utilisées sont notamment le type de la relation, sa profondeur dans le graphe social et un indice de confiance calculé par le système. Le propriétaire définit des règles qui expriment quelles propriétés de relation doivent être satisfaites par un sujet pour qu'il puisse accéder à des données. Les accès ne sont donc pas figés dans le temps, le système étant capable de gérer le dynamisme du graphe social avec l'apparition de nouveaux sujets, le rapprochement ou l'éloignement de certains avec le propriétaire, etc. De même, dans [Cutillo 2009] et [Chard 2012] les utilisateurs du réseau définissent leurs contacts sur la base de relations réelles et leur attribuent des niveaux de confiance qui définissent leurs droits. Ainsi, un contact considéré comme proche à l'instant t ne le sera pas forcément à $t + \lambda$, et verra ses droits automatiquement mis à jour le cas échéant. Dans le même esprit, [Bellavista 2014] tente de fédérer les multiples identités sociales des utilisateurs et de classifier leurs relations en trois types (*awareness*, *symmetry* et *mutual acceptance*). Chaque type est

associé à un ensemble de permissions prédéfinies que le propriétaire peut modifier et adapter à sa guise.

Ces travaux visent à simplifier l'expression des politiques de partage et en particulier la gestion des sujets dont le statut est rarement figé au cours du temps. Les amis, collègues et même la famille évoluent constamment, tandis que les identités sont dispersées sur de multiples plateformes sociales. Exprimer des partages par la nature des relations plutôt que par l'identité permet de gérer cette instabilité sociale et s'abstraire du problème récurrent de la gestion des relations sur internet propre à chaque service qui pose bien souvent des casse-têtes aux individus pour savoir avec qui partager tel ou tel document. Cependant, cette approche peut présenter certains risques vis-à-vis du contrôle que peut avoir un utilisateur sur la dissémination de ses données personnelles : en effet, il est compliqué pour lui d'appréhender qui sont exactement ses destinataires, sur la base de graphes sociaux et de métriques plus ou moins opaques. De plus, les partages ne sont pas toujours fonction d'un type de relation, mais parfois plus contextuels ; un utilisateur peut être prêt à partager des documents médicaux très sensibles avec des médecins qu'il ne connaît pas, mais ne pas vouloir que ses amis proches y aient accès, par exemple.

Il convient donc de trouver un compromis entre simplicité d'utilisation et contrôle du propriétaire. Cela est vrai pour la définition des sujets, mais aussi des objets.

Administration des objets

Aujourd'hui, la plupart des plateformes qui offrent la possibilité à leurs utilisateurs de partager leurs données proposent une approche très manuelle et peu dynamique. Bien souvent, les utilisateurs sont obligés d'exprimer leurs partages au cas par cas, avec peu de possibilités d'automatisation et encore moins de **transversalité** : il est typiquement très compliqué d'exprimer le partage de tous les documents associés à un contexte particulier, quel que soit leur type. Par exemple il serait pratique de pouvoir exprimer le partage de tous les documents relatifs à un *road trip* entre amis (photos, dépenses, traces GPS, etc) ou pour un projet d'entreprise (entrées d'agenda, contact de chaque participant, compte-rendu de réunions, etc) sans avoir à refaire une action manuelle pour chaque nouveau document concerné par ce type de partage. Cette transversalité est propre au Cloud personnel qui agrège les données de tout type et n'a pas un modèle de données figé : la possibilité d'ajouter et supprimer des applications et de nouveaux usages amènent inévitablement une

versatilité qui n'existe pas dans les systèmes d'informations traditionnels. De plus, les données peuvent être ajoutées à un rythme élevé, par exemple une série temporelle de consommation électrique : dans ce cas, le partage ne peut être statique mais doit pouvoir se mettre à jour pour capter tout nouveau document ou mise à jour légitime à entrer dans les autorisations.

[Geambasu 2007] propose une solution en pair-à-pair pour facilement exprimer des partages transverses et dynamiques. Un langage fondé sur SQL permet aux applications de créer des vues sur les données du propriétaire et de transmettre des *capabilities* aux sujets concernés qui sont des jetons d'authentification donnant droit à accéder aux résultats d'une vue. Les sujets requêtent ainsi directement les vues en transmettant leurs *capabilities* pour récupérer les données chez le propriétaire. Les permissions sont donc associées aux résultats des vues, ce qui permet de gérer l'aspect dynamique des données personnelles. Si des données sont ajoutées, modifiées ou supprimées, et que le résultat d'une vue change, les sujets pourront relancer une requête pour récupérer les mises à jour, sans que le propriétaire n'ait à redéfinir des accès manuellement.

[Mazurek 2014] présente Penumbra, un système de fichiers distribué à base de tags qui vise à simplifier à la fois l'administration des fichiers et du partage pour des personnes non techniques. L'utilisation des tags est justifiée par le fait que les utilisateurs en auraient une compréhension et un usage plus intuitifs que les traditionnels systèmes de fichiers hiérarchiques [Seltzer 2009] [Klemperer 2012]. Cela donne également la possibilité d'avoir une vue transversale des données qui peuvent être indifféremment des fichiers, notes, agenda, etc. Le propriétaire d'un appareil classifie donc ses données en leur attribuant des tags qui peuvent être requêtés et combinés en des expressions logiques. Par exemple, *type=photo ∧ album=Iceland17* retourne l'album photos des vacances de l'été 2017. De la même façon, le propriétaire partage ses données en attribuant des accès sur ses tags. Comme les tags sont au cœur de la politique du contrôle d'accès, ils sont signés avec la clé privée du propriétaire afin de s'assurer qu'aucune application ou tiers non autorisé ne puisse les modifier. Les règles d'accès sont également signées et vérifiées par un système *logic-based* [Wobber 1994] qui permet de s'assurer que seul un sujet accrédité est capable de forger la preuve nécessaire pour exécuter sa requête.

[Riva 2011] définit une sémantique de règles adaptée au contexte du Cloud personnel, ici vu comme l'union des appareils d'un propriétaire. Ces règles permettent de répliquer les

données entre les appareils, selon leur type (photos, contacts, etc), ou des éléments contextuels (importance des données, espace mémoire de l'appareil, etc). Les règles de partage sont exprimées en Prolog[12], un langage de programmation logique. C'est aussi le cas de [Zaychik Moffitt 2015] qui permet d'exprimer des règles de contrôle d'accès pour des données distribuées, dans un langage inspiré de Datalog[13], lui-même fondé sur Prolog.

On constate que ces travaux utilisent des systèmes de règles de partage qui permettent au propriétaire de facilement exprimer des vues transverses sur ses données, que ce soit par des langages hérités de SQL [Geambasu 2007], de Prolog [Riva 2011] [Zaychik Moffitt 2015] ou par des tags [Mazurek 2014]. Dans tous les cas, cela évite au propriétaire d'avoir à exprimer manuellement chaque partage pour chaque type de données, comme c'est majoritairement le cas aujourd'hui. Ceci étant posé, on peut également se demander s'il n'est pas possible d'aller encore plus loin, en automatisant la création de règle.

Administration des règles

Créer des règles d'accès est une tâche qui peut s'avérer fastidieuse. Bien que les travaux précédemment cités allègent considérablement ce fardeau en permettant de simplifier l'expression des sujets et des objets, cela demande d'activer manuellement des règles pour représenter la politique de partage. Or, les progrès réalisés en intelligence artificielle laissent à penser que ces règles pourraient être directement inférées des habitudes et préférences de l'utilisateur. [Fang 2010] et [Squicciarini 2011] proposent ainsi d'utiliser des d'algorithmes de *machine learning*, pour déduire automatiquement des politiques de partage à partir des données des utilisateurs.

[Fang 2010] part de l'observation que les individus ont tendance à concevoir leurs préférences de vie privée sous la forme de règles implicites, en partageant certaines données sur les réseaux sociaux avec certains types de personnes. Afin de construire un modèle de partage adapté à chaque utilisateur, quelques données d'initialisation sont

[12] https://fr.wikipedia.org/wiki/Prolog

[13] https://fr.wikipedia.org/wiki/Datalog

demandées manuellement au propriétaire. Plus ce dernier fournit d'informations, plus le système de classification est précis. En complément, toutes les données sociales du propriétaire sont analysées afin d'en extraire des communautés d'individus avec qui le propriétaire a des préférences de dissémination communes. Chaque nouvelle donnée ajoutée par le propriétaire est ensuite classifiée et exposée à certaines communautés, en fonction de la construction du modèle. A noter que les utilisateurs avancés peuvent également visualiser leur modèle sous la forme d'un arbre de décision et les modifier à leur guise.

[Squicciarini 2011] utilise un modèle de classification d'images entraîné sur un large ensemble de photos, afin d'être capable d'inférer le degré d'exposition souhaité pour chaque image ajoutée par le propriétaire, à partir de ce qu'elle représente et de ses métadonnées. Il doit également spécifier ses préférences de vie privée pour que le classifieur puisse s'adapter à ses spécificités. Une sémantique de règles est fournie pour l'expression des préférences qui permet de combiner des prédicats simples.

Ces travaux affirment obtenir de bons résultats quant à la précision de leurs modèles, supérieure à 90%. Parallèlement, les algorithmes de *machine learning* sont de plus en plus puissants et voient fleurir de multiples applications, allant de la recommandation musicale à la création automatique de tag sur des albums photos. Néanmoins, appliquées au contrôle d'accès, les conséquences d'une mauvaise classification peuvent être bien plus importantes qu'une chanson de mauvais goût ou un tag imprécis : cela peut aboutir à des partages non voulus sur des données potentiellement sensibles. De plus, il n'est pas certain que les utilisateurs fassent confiance à des algorithmes sur lesquels ils n'ont aucune compréhension ni réel contrôle.

2.2.4 Conclusion

Nous avons vu que de nombreux travaux ont été réalisés afin de pouvoir partager en toute sécurité sur des systèmes décentralisés. Les approches *user-centric* requièrent inévitablement une nouvelle prise de responsabilité de la part de l'utilisateur qui doit être accompagné grâce à des outils facilitant le partage de sujets, d'objets et leur

administration. Les sujets ne peuvent être gérés de la même façon qu'un annuaire centralisé de type LDAP[14] ; exploiter les liens entre les individus est une piste intéressante pour gérer la versatilité des relations sociales. Mis en perspective avec le Cloud personnel, on pourrait extraire ces liens directement depuis les données des individus : un ami proche a toutes les chances d'apparaître dans des photos, contacts ou messages ; un collègue sera mentionné dans des entrées agenda, rapports, e-mails, etc. La diversité et **l'aspect dynamique** des données doivent aussi être prises en compte dans la capacité d'expression du partage des objets : un utilisateur doit pouvoir échanger n'importe quel type de document et ne pas avoir à effectuer d'action supplémentaire lorsque de nouveaux documents légitimes sont ajoutés ou modifiés.

De plus, des protocoles d'**authentification** spécifiques ont été proposés pour tenter de gérer les identités des sujets, fragmentées sur le Web ; chaque identité est alors associée à un ensemble de permissions. Malheureusement, ces protocoles induisent souvent une complexité technique qui rebute la plupart des utilisateurs qui ont du mal à savoir exactement ce qui est partagé et avec qui. Cela introduit la problématique de l'***enforcement*** des permissions**,** c'est-à-dire l'application correcte de la politique de partage. Des techniques de **chiffrement** permettent de l'assurer sur des plateformes centralisées non contrôlées par les individus et garantissent la **confidentialité** des données. Mais ces techniques impactent considérablement les performances et dégradent l'expérience utilisateur. De plus, la complexité de ces modèles rend difficile la compréhension générale de l'individu vis-à-vis de la façon dont sont appliquées ses politiques. N'ayant aucune garantie tangible, son ***empowerment*** se retrouve ainsi cassé et le propriétaire n'a alors d'autre choix que de placer son entière confiance dans le système, sans même avoir la possibilité de savoir si quelque chose se passe mal.

Finalement, aucun des travaux cités n'a réellement fait consensus dans le monde « réel ». Au-delà des désavantages cités, un modèle de partage doit pouvoir s'intégrer dans une plateforme de Cloud personnel existante afin de garantir son adoption et permettre un déploiement à l'échelle industrielle. Cela est loin d'être évident, car ces plateformes ont chacune leur propre architecture, applications et usages associés.

[14] https://fr.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol

2.3 Cloud personnel et partage

Alors que nos vies se numérisent et que l'on assiste à une explosion de la volumétrie des données personnelles, il devient de plus en plus important de disposer d'outils permettant de stocker, gérer et partager ses données de façon sécurisée et pérenne. Peu à peu, les traditionnels systèmes de fichiers ont laissé leur place aux solutions de Cloud, où les données ne sont plus stockées sur un disque dur de l'ordinateur familial, mais dans des infrastructures dédiées qui assurent une synchronisation des données entre les différents appareils du propriétaire et une sauvegarde automatique garantie.

Dans cette section, nous faisons un tour d'horizon de solutions de Cloud personnel et des fonctionnalités de partage qu'elles offrent. Le Cloud personnel est ici vu au sens large, c'est-à-dire, un espace personnel pour un individu, accessible en ligne, sur lequel il peut librement ajouter, modifier ou supprimer ses documents, tout en les synchronisant entre ses différents appareils.

2.3.1 Solutions centralisées

Comme dit précédemment, la centralisation des données pose de nombreux problèmes en termes de sécurité et de respect de vie privée. Cependant, la popularité des solutions centralisées actuelles les rend de fait incontournables. [Bocchi 2015] a analysé le trafic internet de plus de 20 000 foyers et a montré qu'environ 25% d'entre eux utilisaient un client Dropbox[15] ou Microsoft OneDrive[16] et 10% pour Google Drive[17].

Toutes ces solutions proposent un système classique de partage de fichiers, c'est-à-dire soit public qui donne accès à la ressource pointée par n'importe qui disposant de l'URL générée,

[15] https://www.dropbox.com/

[16] https://onedrive.live.com/

[17] https://www.google.com/drive/

soit restreint à des utilisateurs de la solution utilisée. Cela correspond à un partage DAC, précédemment décrit dans ce chapitre, qui manque de souplesse dans l'expression du partage. Cependant, la centralisation des données procure un avantage indéniable pour le partage : tout document partagé correspond à un accès créé pour un destinataire qui reçoit une version synchrone avec celle du propriétaire, vu qu'il s'agit de la même donnée. Cela permet donc un partage **collaboratif**, où toute mise à jour est propagée entre les participants d'un partage.

Pour ces services Cloud, la notion de *personnel* est toute relative. Premièrement, toutes les données stockées par l'individu se retrouvent dans un silo unique, détenu par l'entreprise qui édite le service. Il n'est pas possible pour un individu de choisir l'emplacement physique du stockage de ses données et il doit donc abandonner tout contrôle dessus et accorder une confiance aveugle aux administrateurs du service.

Deuxièmement, les conditions d'utilisation de ces services accordent de nombreux droits au gestionnaire du service, au détriment des propriétaires. Pour Google Drive et Microsoft OneDrive, les conditions sont les mêmes que pour leurs autres services, c'est-à-dire qu'ils s'arrogent le droit d'exploiter les données stockées [Google 2017] [Microsoft 2016]. iCloud[18], le service d'Apple, se réserve la possibilité de supprimer n'importe quelles données, à n'importe quel moment, sans avoir à prévenir l'utilisateur [Apple 2017]. Et même si les termes peuvent être acceptables à un moment donné, l'utilisateur reste sous la menace d'un changement inopiné qui peut se faire sans avoir besoin de son consentement, comme c'est le cas pour Dropbox [Dropbox 2017]. Ces conditions d'utilisation sont généralement longues, complexes et rébarbatives à appréhender ; la plupart des personnes ne les lisent ainsi simplement pas et se contentent de cocher la case d'acceptation lorsqu'ils souscrivent à un tel service. Heureusement, des initiatives comme [TOSDR 2017] résument l'impact sur la vie privée des services les plus populaires et aident l'utilisateur à mieux comprendre les engagements qu'il a pris, généralement à son insu.

Bien que très populaires, car bénéficiant d'un service fiable conjugué à une expérience utilisateur éprouvée, les solutions centralisées ne sont pas exemptes de risques et reproches

[18] https://www.icloud.com/

et s'inscrivent dans un processus de perte de contrôle des utilisateurs sur leurs propres données personnelles. Le partage proposé y est basique et n'est pas adapté à la transversalité du Cloud personnel. Le besoin d'alternatives décentralisées est donc réel et se concrétise progressivement ces dernières années par des outils qui visent à faire aussi bien, voire mieux que les géants du Web.

2.3.2 Solutions décentralisées

De nombreux outils existent qui permettent aux utilisateurs techniques de stocker et utiliser leurs données en garantissant leur vie privée. Par exemple, il est tout à fait possible de configurer un serveur (S)FTP[19] sur un ordinateur à la maison et utiliser des clients de synchronisation sur les terminaux, à base de scripts rsync[20] et cron[21]. Le partage peut s'effectuer en créant des utilisateurs distincts et en configurant manuellement leurs droits. Il ne faut également pas oublier d'attribuer une IP statique au serveur pour qu'il soit accessible de l'extérieur, lui attribuer un nom de domaine, et configurer une redirection de port NAT[22] depuis le routeur local, s'il n'est pas possible d'utiliser une IP publique.

Cela va sans dire que ce genre de mise en place n'est pas à la portée de tout le monde. Heureusement, des solutions existent pour permettre à des utilisateurs non techniques de stocker, synchroniser et partager leurs données, sans avoir besoin de connaissances techniques poussées.

Nous ne considérons ici que des solutions qui requièrent l'usage d'un serveur pour y stocker les données et utiliser des applications pour les manipuler et les partager. Il convient de préciser que d'autres approches, dites *serverless*, existent, qui permettent un stockage

[19] https://fr.wikipedia.org/wiki/File_Transfer_Protocol

[20] https://rsync.samba.org/

[21] http://www.linux-france.org/article/man-fr/man5/crontab-5.html

[22] https://fr.wikipedia.org/wiki/Network_address_translation

totalement distribué entre les pairs, comme SyncThing[23], MaidSafe[24] et Storj[25], mais limitent les usages qui peuvent être fait sur les données et se focalisent exclusivement sur les fichiers.

Des initiatives comme FreedomBox[26] ou YunoHost[27] promeuvent l'auto-hébergement, c'est-à-dire le fait d'héberger soi- même ses données, en alternative aux services centralisés. Cela garantit un contrôle total des données par l'utilisateur qui n'est alors plus tributaire des conditions d'utilisation et des pratiques commerciales opaques des solutions centralisées. De plus, ces initiatives défendent l'utilisation de logiciel libre[28] qui est une condition importante, sinon essentielle, à l'émancipation numérique de l'individu. Difficile en effet de faire confiance à un logiciel utilisant du code non accessible, ce qui revient à exécuter une boîte noire sur ses données personnelles. La transparence est une composante importante de l'auto-hébergement et de la confiance qui en découle; même si l'utilisateur final ne vérifie pas lui-même le code, il peut raisonnablement faire confiance en la communauté pour faire ce travail et assurer que le logiciel peut être considéré comme étant de confiance. Enfin, au-delà de la confiance que peuvent apporter les projets libres, la liberté de pouvoir récupérer le code, l'exécuter soi-même et même le modifier apporte des garanties sur la pérennité du système. En effet, les utilisateurs des solutions centralisées sont à la fois dépendants du logiciel et de l'infrastructure du service. Si l'entité derrière le service décide de le fermer, comme cela arrive régulièrement [Van der Sar 2012] [Eichner 2012] [Albers 2017] [Puel 2009] [Hermann 2009], il devient impossible de le réutiliser et toutes les données deviennent inaccessibles. A l'inverse, si les mainteneurs d'un projet libre et décentralisé décident de jeter l'éponge, il est toujours possible pour d'autres personnes de le récupérer. Et comme les données sont stockées dans un lieu sous le contrôle de l'individu, ce dernier n'a pas à

[23] https://syncthing.net/

[24] https://maidsafe.net/

[25] https://storj.io/

[26] http://freedomboxfoundation.org/

[27] https://yunohost.org/#/

[28] https://fr.wikipedia.org/wiki/Logiciel_libre

craindre une fermeture brutale qui le priverait à jamais de ses données. Il est cependant important de préciser que l'auto-hébergement n'est pas forcément synonyme de logiciel libre. Par exemple, Lima[29] est un projet non libre qui consiste à stocker toutes ses données sur un disque dur chez soi, relié à un boîtier connecté qui permet d'accéder au disque depuis n'importe quel appareil. De même, il ne faut pas confondre libre et open-source [Stallman 2016], le deuxième étant une notion plus laxiste du premier.

Même si des initiatives cherchent à le simplifier, l'auto-hébergement reste un sujet complexe qui ne peut être raisonnablement mené à bien qu'avec l'intervention d'un expert dans le domaine. Des projets comme ownCloud[30], nextCloud[31], Sandstorm[32], Cozy[33], Camlistore[34], Pydio[35], Seafile[36] ou [Haddadi 2015] ont une approche pragmatique : l'emplacement du stockage et le lieu de l'exécution du logiciel sont laissés libre à l'individu : cela signifie qu'il peut soit l'installer sur sa propre machine, s'il a les connaissances nécessaires, ou bien déléguer l'administration à un hébergeur spécialisé, éventuellement moyennant finances.

ownCloud est un pionnier dans le domaine du Cloud personnel. Crée en 2010, il avait pour vocation d'offrir les mêmes fonctionnalités de stockage et synchronisation de fichiers que les géants centralisés, sans faire de compromis sur la vie privée des individus. Suite à des désaccords internes [Karlitschek 2016] et grâce à la licence libre du projet, nextCloud a été lancé en 2016 comme un *fork* de ownCloud, c'est à dire une nouvelle branche logicielle fondée sur le même code source. Les deux projets utilisent une API commune [GEANT

[29] https://meetlima.com/

[30] https://owncloud.org/

[31] https://nextcloud.com/

[32] https://sandstorm.io/

[33] https://cozy.io/

[34] https://perkeep.org/

[35] https://pydio.com/

[36] https://www.seafile.com/

2017] pour le partage de fichiers, fondée sur WebDAV[37] et développée par une fédération de partenaires[38]. Cette spécification vise à permettre un partage interopérable entre différentes plateformes de Cloud personnel décentralisées, par exemple Pydio. En effet, pouvoir partager entre des systèmes hétérogènes est essentiel pour espérer rivaliser avec les géants centralisés et permet de mutualiser les efforts. Malheureusement, cette spécification, à l'origine conçue par ownCloud, est focalisée sur le partage de fichier et répertoire et n'adresse donc pas la problématique d'un modèle agnostique vis-à-vis des types de document.

Sandstorm permet d'exécuter des applications dans des conteneurs isolés les uns des autres, tout en leur laissant la possibilité de communiquer grâce à un protocole de communication de bas niveau, Capnproto[39], avec une authentification par *capabilities*. Grâce à leur système de conteneurs, il est théoriquement possible d'exécuter n'importe quelle application dans Sandstorm, développée dans n'importe quel langage, moyennant une configuration spécifique. Et toute application peut implémenter l'API de partage fournie par Sandstorm. Cette API repose sur le principe de délégation qui reprend le concept clé de DAC : tout sujet ayant des droits sur une application peut déléguer ces droits, ou un sous-ensemble, à d'autres sujets en leur communiquant la *capability* associée. Cette approche fonctionne particulièrement bien pour des contextes de collaboration qui est le parti pris de Sandstorm. Cependant, ce type de partage devient dangereux pour des contextes plus intimes, que ce soit pour des photos personnelles ou des documents sensibles, car il devient très compliqué pour le propriétaire de réguler ses accès et contrôler qui peut accéder à quoi.

Enfin Cozy, la solution éditée par l'entreprise Cozy Cloud, est une plateforme de Cloud personnel proche de Sandstorm, où les données ne sont pas que des fichiers à synchroniser, mais peuvent être de multiples natures, comme des évènements, des mémos, traces GPS, données bancaires, etc. Mais contrairement à Sandstorm, les applications installées sont essentiellement *client-side*, c'est-à-dire qu'elles s'exécutent dans le

[37] https://tools.ietf.org/html/rfc4918

[38] https://wiki.geant.org/display/OCM/Open+Cloud+Mesh

[39] https://capnproto.org/

navigateur, dans un sous-domaine dédié. Il n'est donc pas possible d'installer n'importe quel type d'application dans Cozy, qui met l'accent sur une expérience utilisateur uniformisée entre les applications grâce à des API et chartes graphiques partagées. Il est néanmoins possible d'avoir des applications qui s'exécutent dans des containeurs, comme c'est par exemple le cas avec l'application *Collect* qui permet de récupérer des données provenant de fournisseurs externes (factures d'énergie, remboursements de santé, etc). L'implémentation du partage y est décrite dans le Chapitre 5 et concrétise le modèle décrit au Chapitre 4.

2.3.3 Conclusion

Les solutions décentralisées libres permettent un vrai contrôle des individus sur leurs données. En ayant le choix de l'hébergement de la plateforme, combiné à la transparence du code exécuté, le propriétaire s'assure que ses données ne sont pas analysées à son insu et réduit le risque d'un piratage de grande ampleur. Cependant, cela ne résout pas un paradoxe fondamental : pour être complètement souverain sur ses données personnelles, le propriétaire doit être en mesure de contrôler le serveur qui les stocke et comprendre le code qui s'exécute dessus. Or, cela n'est réellement possible que si le propriétaire est un expert. De plus, la question du contrôle du partage demeure : comment le propriétaire peut-il simplement exprimer une politique de partage plus ou moins complexe tout en ayant des garanties sur son application effective ?

Une réponse appropriée à cette question est d'autant plus essentielle que les solutions centralisées sont de plus en plus omniprésentes, avec des fonctionnalités parfois basiques, mais éprouvées et simples à prendre à main, même pour l'utilisateur débutant. Les risques de violation de la sécurité et du non-respect de la vie privée étant des concepts non tangibles, voire abstraits, les utilisateurs font en grande majorité le choix de la simplicité. Alors que pour Montesquieu, *« Le mieux est l'ennemi du bien »,* nous affirmons ici que « Le simple est l'ennemi du bien ».

2.4 Problématique

De fait, le partage de documents et son administration peuvent être appréhendés de nombreuses façons, mais aucune des approches précédemment citées n'est réellement satisfaisante pour le contexte du Cloud personnel. Deux difficultés majeures ressortent de ces travaux :

- *Le propriétaire est le maillon faible de la sécurité*. Alors que le propriétaire est *de facto* l'administrateur de son Cloud personnel, il est illusoire d'attendre de lui qu'il sécurise son serveur contre toutes formes d'attaques ou bien qu'il utilise des protocoles cryptographiques complexes pour échanger des secrets avec ses destinataires. D'autre part, déléguer ces tâches à des tierces parties est en opposition frontale avec le principe fondamental d'*empowerment* de l'utilisateur. Cette délégation s'accompagne inévitablement d'une perte de souveraineté sur les données et contribue à une nouvelle forme de centralisation.

- *L'usage du Cloud personnel est versatile et volatile.* Tandis que les systèmes d'information traditionnels sont construits pour supporter des services bien identifiés invoqués par des sujets et applications bien définis, le Cloud personnel favorise les usages opportunistes et des interactions non prévisibles entre les utilisateurs. Les approches décentralisées échouent à s'imposer à large échelle à cause de leur manque de clarté et de souplesse, conjugués à une dégradation de l'expérience utilisateur.

Nous pouvons dériver de ce constat trois propriétés majeures à appliquer dans la quête d'un système de partage sécurisé, adapté au Cloud personnel :

1. ***Empowerment* éclairé :** le propriétaire doit être en mesure de percevoir et comprendre les effets concrets de toutes ses décisions relatives au partage, et ce, quel que soit le modèle utilisé. Pour cela, il doit avoir la possibilité de visualiser chaque document et chaque sujet concerné par un partage et être capable de révoquer ce partage si besoin. Ainsi, il reste maître de toutes les actions, tout en ayant la compréhension de chacune d'entre elles.

2. **Auto-administration :** l'administration des objets, sujets et règles est une tâche chronophage et complexe, et ce, d'autant plus dans le Cloud personnel où des données et applications sont constamment ajoutées. Il est illusoire d'attendre du propriétaire qu'il

trie ses documents selon une hiérarchie imposée par la plateforme de Cloud personnel utilisée, ou bien qu'il définisse manuellement chaque sujet avec qui il souhaite partager. L'administration doit donc être intuitive et quasi automatique, en exploitant la diversité et la versatilité du Cloud personnel : le contenu des documents peut être directement exploité pour en extraire des règles de partage et les sujets peuvent en être dérivés, le tout dynamiquement, c'est-à-dire à chaque ajout, mise à jour ou suppression de document.

3. **Sécurité par construction :** la logique du moniteur de référence qui est en charge de l'application de la politique du contrôle d'accès doit rester compréhensible par le propriétaire, pour qu'il soit en mesure d'appréhender les effets de sa politique de partage, typiquement savoir exactement ce qui est partagé et avec qui, et pourquoi. De plus, la plateforme qui exécute le moniteur de référence doit fournir des garanties fortes et **tangibles** de sécurité, afin que le propriétaire puisse avoir une entière confiance dans l'application correcte du contrôle d'accès et donc de sa politique de partage, et ce, de par la construction même de son architecture.

Nous ne prétendons pas que ces propriétés soient suffisantes à elles seules pour résoudre tous les problèmes identifiés dans ce chapitre. Néanmoins, nous croyons qu'elles peuvent rendre le partage bien plus accessible, puissant, et sécurisé pour les individus, afin qu'ils puissent bénéficier d'un système à la fois intuitif et respectueux de leur vie privée et accéder à l'*empowerment* cher au Cloud personnel.

Dans la suite de ce document, nous détaillons notre solution, SWYSWYK, pour répondre à ces trois propriétés. Nous détaillons dans un premier temps au Chapitre 3 une architecture sécurisée pour le Cloud personnel, ainsi que des outils aidant le propriétaire à avoir l'assurance que sa politique est correctement appliquée. Puis, au Chapitre 4, nous présentons un modèle de partage capable d'exploiter pleinement les propriétés du Cloud personnel et de produire des règles et des sujets directement depuis les documents. Enfin, au Chapitre 5 nous présentons comment SWYSWYK a été implémenté et mis en production dans la plateforme Cozy. Nous concluons au Chapitre 6 et discutons des résultats et perspectives futures.

Chapitre 3
Architecture SWYSWYK

Nous présentons ici une architecture *Privacy-by-Design* dédiée au Cloud personnel qui a fait l'objet d'une publication à la conférence ISD' 17 [Tran-Van, SWYSWYK: a privacy-by-design paradigm for personal information management systems. 2017]. L'objectif est d'avoir une approche suffisamment générique pour être agnostique vis-à-vis de la plateforme de Cloud personnel utilisée et de proposer un ensemble d'outils qui permettent d'administrer simplement le partage, tout en garantissant la sécurité de l'ensemble. L'architecture présentée peut ainsi être déclinée en plusieurs instances, avec différents types de matériel. Nous présentons une déclinaison possible, sous la forme de plateforme expérimentale qui nous a permis de valider notre approche à la fois quantitativement et qualitativement.

De plus, nous présentons dans ce chapitre un démonstrateur [Tran-Van, SWYSWYK: A New Sharing Paradigm for the Personal Cloud. 2017] qui utilise cette plateforme pour démontrer à la fois la faisabilité de l'architecture et son intérêt pour le respect de la vie privée des individus.

3.1 Introduction

Comme vu au Chapitre 2, de nombreuses approches peuvent être envisagées autour du partage de données et de son application sécurisée. [Tootoonchian 2009], [Van Kleek 2012], [Seong 2010] et [Yeung 2009] ont des approches décentralisées *user-centric* et laissent l'utilisateur en charge de la sécurité de sa plateforme. Certains travaux se concentrent spécifiquement sur un chiffrement par attributs ABE [Ali 2017] [Thilakanathan 2014] [Wang 2016] [Baden 2009], tandis que d'autres proposent des techniques d'offuscation des

données [Yuan 2015] [Guha 2008]. D'autres encore nécessitent un proxy central pour appliquer la politique de partage [Xu 2012] [Canard 2016].

Cependant, aucune de ces approches n'a remporté la bataille de l'adoption à grande échelle et toutes échouent à adresser les spécificités du Cloud personnel.

La première difficulté vient du fait que le moniteur de référence en charge du contrôle d'accès est soit exécuté sur le serveur en charge d'héberger le service, soit déporté sur les appareils de l'individu, si le serveur n'est pas de confiance. Mais dans tous les cas, ce moniteur est généralement trop complexe pour pouvoir être réellement compris par une personne non technique : l'exécution à la volée de requêtes qui peuvent porter sur des attributs ou des rôles permet une certaine flexibilité dans la définition du contrôle d'accès, mais complexifie la perception générale de la politique de partage et ses potentiels effets. Le propriétaire doit donc avoir une entière confiance à la fois dans le code exécuté et dans la plateforme qui le fait tourner. Or ces deux vecteurs de confiance ne sont pas toujours justifiés : d'une part, plus un système est complexe et plus les probabilités d'erreurs sont élevés ; et d'autre part, aucune garantie tangible n'est fournie vis-à-vis de l'environnement d'exécution du moniteur de référence, ce qui le rend potentiellement vulnérable à tout type d'attaques.

La seconde difficulté est liée au fait que les propriétaires se retrouvent démunis pour correctement administrer leurs partages, afin de vérifier quelles sont exactement les permissions accordées et s'assurer que leur modèle mental en terme de dissémination des données correspond bien à ce qui est réellement appliqué, ce qui est en réalité rarement le cas. La plupart du temps, les utilisateurs partagent plus que ce dont ils ont conscience [Mazurek 2010]. Le risque est donc grand de voir les propriétaires déléguer l'administration de leurs données à des fournisseurs de services centralisés pour se décharger de cette responsabilité. Cela brise le principe d'*empowerment* mais surtout aggrave le phénomène de centralisation des données, en fournissant potentiellement l'intégralité du patrimoine numérique.

Comme dit à la section 2.4, nous nous retrouvons ici avec deux faits contradictoires. D'une part le propriétaire du Cloud personnel est le seul administrateur légitime et doit donc être en charge de la sécurité, alors qu'il est parfaitement illusoire d'espérer qu'il puisse acquérir l'expertise et consacrer le temps requis pour s'en charger correctement. Il est de fait le

maillon faible de la sécurité. D'autre part, la délégation de ces tâches est en opposition frontale avec le principe d'*empowerment* du Cloud personnel et expose l'utilisateur à des risques de violation de sa vie privée.

L'objectif de ce chapitre est précisément d'adresser ce problème. Nous ne proposons pas ici un nouveau modèle de contrôle d'accès, mais un nouveau paradigme de partage dédié au Cloud personnel qui s'inscrit dans les principes de *Privacy-by-Design*. Nous faisons ici les contributions suivantes :

- Nous proposons un nouveau paradigme appelé **SWYSWYK**, pour *Share What You can See with Whom You Know*, qui aide les propriétaires de Cloud personnel à visualiser et valider les effets concrets de leurs politiques de partage sur leurs données, quel que soit le modèle de partage utilisé.

- Nous définissons une architecture de référence qui intègre le paradigme SWYSWYK et fournit au propriétaire des garanties tangibles concernant la sécurité et l'application correcte de ses partages. En particulier, la logique du moniteur de référence se veut triviale à comprendre, pour permettre au propriétaire de facilement appréhender le contrôle d'accès. Cette simplicité a pour effet de bord de pouvoir implémenter le moniteur dans un matériel dédié, avec de faibles ressources de calcul, mais de fortes garanties de sécurité.

- Nous présentons une évaluation de performances sur une instance de l'architecture de référence qui combine un Cloud personnel commercial et un matériel dit *tamper resistant*, c'est-à-dire protégé contre les attaques physiques. Les résultats de cette évaluation prouvent la faisabilité de cette approche. De plus, un démonstrateur réalisé sur cette plateforme présente une utilisation concrète de l'architecture du point de vue d'un propriétaire.

Ce chapitre contribue aux propriétés 1 et 3 que nous avons édictées à la section 2.4, à savoir : **l'*empowerment* éclairé** et **la sécurité par construction.** L'architecture et les outils présentés ici visent à permettre aux propriétaires d'être conscients des effets de leur politique de partage, tout en ayant des garanties fortes sur leur application correcte.

3.2 Principes de base de SWYSWYK

Comme dit précédemment, l'objectif ici n'est pas de présenter un nouveau modèle de contrôle d'accès. Nous faisons l'hypothèse que la politique de partage est définie par le modèle fourni par la plateforme de Cloud personnel utilisée. Ainsi, n'importe quel modèle cité dans le Chapitre 2 pourrait être utilisé, à condition de respecter les trois prérequis suivants :

1. **Visualisation des documents.** La granularité du partage est au niveau du document et n'importe quel document partagé doit être visualisable par le propriétaire. Cela implique qu'il existe une ou plusieurs applications de visualisation, appelées *Viewer,* capables d'interpréter chaque document partagé et de le rendre visualisable par un humain. Ainsi, même si un partage est le résultat d'un traitement complexe sur un ensemble de documents, ce traitement doit produire un document visualisable et compréhensible par le propriétaire. Par exemple, un calcul d'agrégation sur une série temporelle provenant d'un compteur électrique connecté pourra être rendu sous la forme d'une courbe de consommation. Nous n'envisageons pas une granularité plus fine que celle du document car cela pourrait rendre l'interprétation difficile pour les *Viewers*. Cette limitation peut néanmoins être facilement contournée, en générant un nouveau document depuis un sous-ensemble d'un ou plusieurs autres documents. Ceci, toujours à la condition de pouvoir le visualiser.

2. **Représentation des sujets.** Chaque sujet avec qui le propriétaire veut interagir doit également correspondre à un document visualisable qui représente le sujet de façon unique. Nous ne posons aucune limitation sur le type de ces documents qui peut être une fiche contact, un CV, une entrée de carnet d'adresse e-mail, etc. Le propriétaire doit pouvoir à tout moment identifier aisément les destinataires de ses partages.

3. **Matérialisation des permissions.** La politique de contrôle d'accès est matérialisée par un ensemble d'ACL (*Access Control List)*, qui représentent les permissions sous la forme de triplets < *s, d, a* >, où *s* et *d* désignent respectivement un sujet et un document stockés dans le Cloud personnel, et *a* l'action accordée au sujet *s* sur le document *d*. Le propriétaire a la possibilité de vérifier ces ACL et supprimer celles qu'il considère comme pouvant porter atteinte à sa vie privée. Il n'est pas attendu de sa part de vérifier toutes les ACL, ce qui serait beaucoup trop fastidieux, mais simplement de valider celles identifiées comme suspectes grâce à des outils d'administration décrits plus loin dans ce chapitre.

Matérialiser toutes les permissions, les rendre visualisables par le propriétaire et lui laisser la possibilité de filtrer les indésirables : voilà les principes fondamentaux de *Share What You See with Whom You Know* (SWYSWYK), littéralement : partage ce que tu vois avec qui tu connais.

Ce principe est en totale opposition avec les approches traditionnelles où les politiques de partage sont souvent définies par un ensemble potentiellement complexe de règles évaluées à la volée par un moniteur de référence opaque, pour accorder ou refuser l'accès aux documents. En effet, nous ne croyons pas qu'un individu lambda soit en mesure de comprendre le résultat d'un solveur complexe prenant en entrée un ensemble potentiellement conflictuel de règles de partage positives ou négatives. Comme dit en introduction de ce chapitre, les approches traditionnelles peuvent ainsi contribuer à l'opacité de la gestion du contrôle d'accès des Cloud personnels, casser le principe d'*empowerment* de l'utilisateur et finalement aboutir à l'exact opposé du but recherché, c'est-à-dire pousser les utilisateurs à définir des politiques de partage trop permissives ou les amener à déléguer le contrôle de leur plateforme à un tiers.

L'hypothèse de confiance faite par SWYSWYK peut être résumée de la façon suivante : *n'accordez pas une confiance aveugle dans les règles de partage, mais faites confiance à votre moniteur de référence pour examiner et ajuster les permissions produites, et les appliquer correctement.*

La logique du contrôle d'accès de SWYSWYK appliquée par le moniteur de référence se veut donc triviale et peut être comprise par n'importe qui : l'opération *a* sur *d* est accordée à *s* si et seulement si $(s,d,a) \in ACL$. Dit autrement, une demande d'accès n'est accordée que si elle apparaît explicitement dans les permissions. Cette logique contribue à la propriété d'***empowerment* éclairé** et permet au propriétaire de garder la maîtrise de sa politique de partage.

Les règles de partage ne peuvent être considérées comme totalement de confiance, car elles peuvent typiquement être fournies par des tiers (une entreprise, une association, des développeurs indépendants, etc) via des applications. La confiance accordée dans ces tiers est variable, et la qualité et l'honnêteté des règles également. Il est donc nécessaire de contrôler la production de ces règles, c'est-à-dire les permissions, potentiellement **suspectes**. Dans SWYSWYK, cela est réalisé via une phase de validation utilisateur et un

processus d'assainissement sur les ACL. A noter que cela ne compromet pas la solidité du modèle de partage du Cloud personnel. En effet, le modèle reste *cohérent* par construction (la décision est unique), *complet* (la décision existe toujours) et peut être évalué dans un *temps logarithmique* sur les ACL matérialisées. Ces trois principes étant généralement considérés comme les fondamentaux nécessaires pour tout modèle de contrôle d'accès [Bertino 2005].

Le point sensible du paradigme de SWYSWYK réside ainsi dans la **détection** et la **validation** des ACL suspectes. Le processus global de validation des ACL est le suivant, résumé par la Figure 5 :

1. La politique de partage est traduite en un ensemble d'ACL dites **candidates**, appelées ACL^{*}.

2. Depuis ACL^{*}, les ACL suspectées de ne pas respecter la politique de vie privée du propriétaire sont détectées et placées en **quarantaine** dans un ensemble appelé $ACL^{?}$. Elles sont alors dans l'attente d'une validation manuelle du propriétaire. Les ACL non suspectes sont placées dans un ensemble appelé ACL^{+}, l'unique ensemble pris en considération par le moniteur de référence pour accorder les accès aux documents. Aucune ACL suspecte n'est donc prise en compte dans l'évaluation du contrôle d'accès.

3. Le propriétaire assainit $ACL^{?}$, l'ensemble des ACL suspectes, au cas par cas. Pour cela, il tire parti de la propriété de visualisation de SWYSWYK, en vérifiant quel document pourrait être partagé avec qui. Pour chaque ACL suspecte, il décide soit de la valider s'il la considère légitime, soit de la refuser. L'ACL est alors déplacée dans ACL^{+} pour le premier cas et dans ACL^{-} pour le second. La matérialisation des ACL^{-} permet d'éviter de stocker dans $ACL^{?}$ des ACL dont la décision a déjà été prise par le passé, et ainsi éviter une vérification superflue du propriétaire. Cela lui permet également de se rétracter et de revalider une permission précédemment refusée.

Nous proposons deux mécanismes pour détecter automatiquement les ACL suspectes et remplir $ACL^{?}$ depuis le contenu d'ACL^{*}. Le premier mécanisme se repose sur un processus de recommandation, appelé *Advisor.* Il identifie les éléments d'ACL^{*} qui sont contradictoires avec des décisions passées, c'est-à-dire similaires à des ACL précédemment classées dans ACL^{-}. Ce mécanisme se fonde sur l'hypothèse que les utilisateurs ont tendance à avoir une

politique de partage relativement stable et cohérente au cours du temps, comme cela est montré par [Roth 2010] et [De Choudhury 2010] .

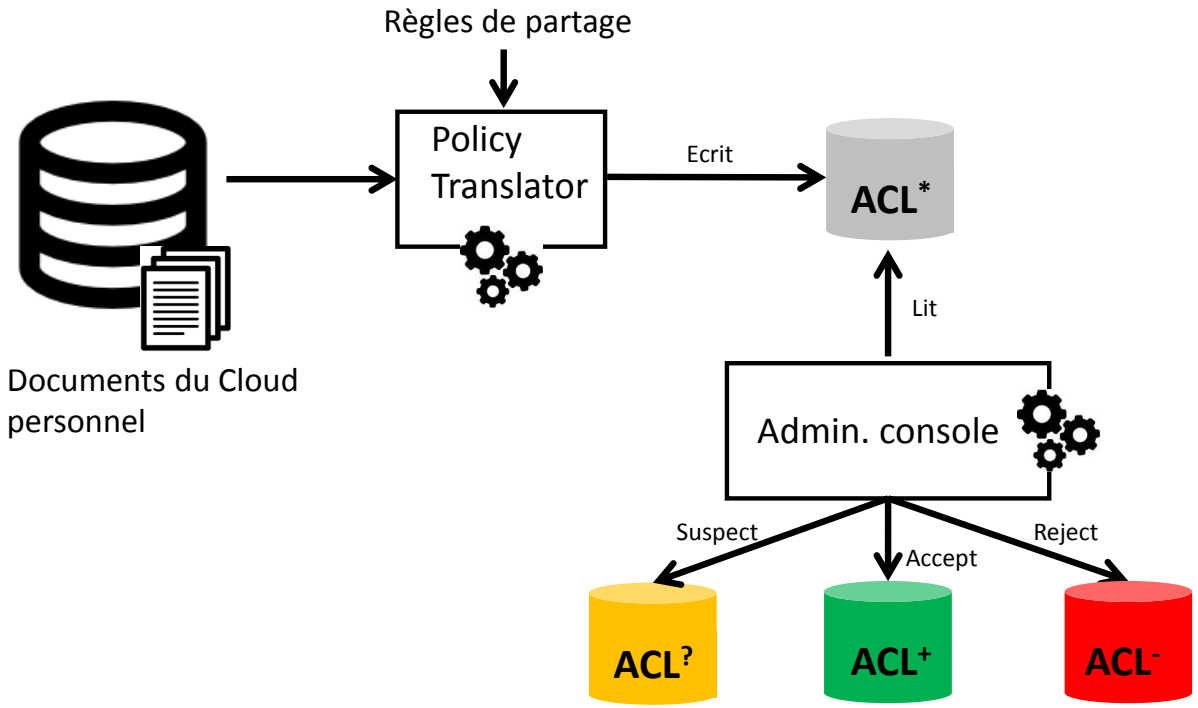


Figure 5. Production des ACL.

Le deuxième mécanisme permet au propriétaire de définir des garde-fous, appelés *watchdog triggers*, qui font ressortir des ACL sur lesquelles une attention particulière doit être portée, du fait de la sensibilité des sujets et/ou des documents impliqués. Dit autrement, ces *triggers* permettent de désigner des documents, sujets ou association des deux comme étant sensibles et de les placer préventivement dans $ACL^{?}$.

Advisor et les *watchdog triggers* sont complémentaires et permettent au propriétaire d'assainir ses ACL. Nous détaillons ces deux mécanismes ci-dessous.

3.3 Suspicion à partir des décisions passées

Le but de *l'Advisor* est de suggérer une décision, *Accept* ou *Reject,* au propriétaire, à partir de n'importe quelle ACL candidate $(s,d,a) \in ACL^*$.

Cette décision est notée $r_{(s,d,a)}$. La décision étant unique par construction, *(s,d,a)* est rejetée si elle existe déjà dans ACL^+ ou ACL^-. Autrement, l'*Advisor* calcule une distance entre l'ACL candidate *(s,d,a)* et les ACL précédemment résolues, c'est-à-dire les éléments de ACL^+ et ACL^-. Puis, une décision $r_{(s,d,a)}$ est suggérée, qui correspond à la plus proche ACL déjà résolue si la distance calculée est sous un certain seuil.

La façon dont les distances sont calculées est un problème intéressant. Une façon de faire serait de comparer les attributs des documents et des sujets, ou bien une comparaison d'historiques de décisions passées. Nous reportons le choix de la meilleure métrique pour des travaux futurs et validons simplement le concept à partir d'un calcul simple, dont l'avantage est de pouvoir être exécutable dans un environnement d'exécution sécurisé qui dispose de peu de ressources, typiquement une carte à puce.

Soit *S* l'ensemble des sujets identifiés dans le Cloud personnel du propriétaire, et $S' \subseteq S$ l'ensemble des sujets *s'* différents de *s* avec une décision résolue concernant *d* et *a*, tel que :

$$S' = \{ s' \in S, s' \neq s, \exists (s',d,a) \in ACL^+ \cup ACL^- \}$$

Soit h_s (respectivement. $h_{s'}$) l'historique de toutes les décisions résolues concernant *s* (respectivement *s'*), tel que :

$$h_s = \{ (s,d,a,r_{(s,d,a)}), \exists (s,d,a) \in ACL^+ \cup ACL^- \}$$

L'algorithme *Resolve(s,d,a),* présenté ci-dessous, retourne deux valeurs. La première est la décision conseillée $r_{(s,d,a)}$ qui se fonde sur le calcul d'une distance $Dist(h_{s'}, h_s)$ entre h_s et l'ensemble des $h_{s'}$. La deuxième valeur retournée est la confiance que peut accorder le propriétaire à la première valeur pour prendre sa décision finale. L'algorithme repose sur l'hypothèse que si *s* et *s'* partagent de fortes similitudes dans leur historique de partage avec le propriétaire, alors une décision prise sur *(d,a)* pour *s'* s'appliquera probablement pour *s* également. Ainsi, si *Resolve* suggère la décision *Accept* avec une confiance proche ou égale à 1, il n'y a pas de raison particulière de douter de sa légitimité et l'ACL correspondante peut être stockée dans ACL^+. A l'inverse, si la décision suggérée a une confiance inférieure à un

certain seuil, il est préférable de marquer l'ACL comme *Suspect* et la placer dans $ACL^{?}$.pour que le propriétaire la classifie manuellement plus tard. En effet, une confiance faible signifie que la distance entre cette ACL et l'historique est grande, et donc qu'elle sort des habitudes du partage.

De la même façon que pour les sujets, l'hypothèse pourrait être faite que si *d* et *d'* partagent de fortes similitudes dans l'historique de partage du propriétaire, une décision passée sur *(s,a)* pour *d'* s'appliquera probablement pour *d.* L'adaptation de l'algorithme ci-dessous, en Figure 6, est alors triviale.

Fonction Resolve

Entrée : $t = \langle s,d,a \rangle \in ACL^{*}$

Sortie : (r, p) avec r une décision binaire (Accept ou Reject) et p ∈ [0,1] la confiance dans cette décision.

1. if $t \in ACL^{+}$ **return** ('Accept', 1);
2. if $t \in ACL^{-}$ **return** ('Reject', 1);

3. /* construit h_s l'historique des décisions pour le sujet s */
   h_s = { (acl, 'Accept'), acl.s=t.s, acl ∈ ACL^{+} }
   ∪ { (acl, 'Reject'), acl.s=t.s, acl ∈ ACL^{-} };

4. /* construit H, l'ensemble des historiques des sujets autres que s, avec la décision a pour le document d */
5. H = { $h_{s'}$, s'≠s, ∃ acl∈ $h_{s'}$, acl.d = t.d, acl.a = t.a };
6. h_{close} ← the history with smallest Dist(h_s, $h_{s'}$), ∀ $h_{s'}$∈H ;
7. r ← the decision for (d,a) in h_{close};
8. p ← 1- Dist(h, h_{close}) ;

9. **return** (r, p);

Figure 6. Algorithme *Resolve*

Ce mécanisme permet donc de détecter automatiquement des permissions qui sortent des habitudes de partage du propriétaire et de les classifier automatiquement dans l'ensemble $ACL^{?}$. Pour chaque nouvelle ACL suspecte, la décision que prendra le propriétaire sera enregistrée dans l'historique et utilisée par la fonction *Resolve* à la prochaine ACL similaire. La précision de l'algorithme s'améliore donc au fil des décisions : plus l'ensemble des ACL^{+}

et ACL^{-} est grand, plus la probabilité d'avoir une confiance forte dans la décision conseillée augmente. La valeur du seuil à partir duquel on considère une permission comme suspecte est également déterminant et doit être adaptable selon les préférences des propriétaires qui peuvent être plus ou moins prudents dans leurs comportements.

La pertinence du calcul de distance faite par la fonction *Dist* détermine fortement l'utilité de *Resolve* pour le propriétaire. Dans la section 3.6.4, nous démontrons qu'utiliser un calcul de distance simple permet déjà d'obtenir des résultats satisfaisants, ouvrant ainsi la voie à des métriques plus riches.

Ainsi, le module d'*Advisor* permet une automatisation de l'administration des ACL du propriétaire, en minimisant les actions requises de sa part et en prenant garde à détecter les risques inhérents à sa vie privée. Cependant, cet outil n'est pas suffisant lorsque peu ou pas de décisions passées ont été prises et que la confiance accordée dans la décision n'est pas suffisante. De plus, le propriétaire pourrait vouloir bénéficier d'un outil plus déclaratif et personnalisable qu'un système uniquement fondé sur ses habitudes de partage.

3.4 Suspicion à partir de la sensibilité

Une ACL peut être considérée comme suspecte parce qu'elle implique un sujet sensible (par exemple, un supérieur hiérarchique), un document sensible (une photo compromettante ou mon dossier médical), ou bien parce l'association entre un sujet et un document est elle-même sensible, même s'ils ne le sont pas forcément intrinsèquement (je ne veux pas partager la date de ma soirée d'anniversaire avec mes collègues non invités). La notion de sensibilité des sujets, documents et associations est laissée à l'appréciation du propriétaire du Cloud personnel. Pour cela, il définit des requêtes appelées *watchdog triggers*, qui lui permettent de faire ressortir les ACL qui concernent ses données sensibles.

Nous faisons l'hypothèse que les documents du Cloud personnel sont liés à des métadonnées (par exemple, le type du document, la date, l'auteur, des tags, etc), qui peuvent être requêtées par un langage de prédicat spécifique à la plateforme.

Soit $Q_E(S)$ les requêtes exprimées sur les métadonnées des documents qui décrivent des sujets, et $Q_E(D)$ les requêtes exprimées sur les métadonnées des documents partagés, avec E le prédicat de la requête.

Trois types de *watchdog triggers* peuvent être exprimés :

- *What'sNewforS* $(E,A) \rightarrow \{(s,\{(d,a)\}) / (s,d,a) \in ACL^* \wedge s \in Q_E(S) \wedge a=A\}$

  Identifie, pour chaque sujet sensible *s*, le nouvel ensemble d'actions *a* qui lui est accordé sur les documents *d*. Par exemple : « Quels nouveaux documents peuvent être vus par mon supérieur ? ».

- *Who'sNewforD* $(E,A) \rightarrow \{(d,\{(s,a)\}) / (s,d,a) \in ACL^* \wedge d \in Q_E(D) \wedge a=A\}$

  Identifie, pour chaque document sensible *d*, le nouvel ensemble des sujets *s* ayant le droit *a* dessus. Par exemple : « Quels nouveaux sujets ont un droit de lecture sur mon dossier médical? »

- *WhichNewSD* $(E,E',A) \rightarrow \{(s,d,a) / (s,d,a) \in ACL^* \wedge s \in Q_E(S) \wedge d \in Q_{E'}(D) \wedge a=A\}$

  Identifie les nouvelles ACL combinant une sélection sensible de sujets et de documents. Par exemple : « Quelles nouvelles autorisations mes collègues ont-ils sur mes photos de famille ? ».

Une fois définies, les requêtes *watchdog triggers* s'exécutent à chaque nouvelle permission insérée dans ACL^*. Cela permet de faire ressortir toute nouvelle permission sensible et de la placer automatiquement en quarantaine dans l'ensemble $ACL^?$.

Contrairement au mécanisme complètement automatisé d'*Advisor*, ces requêtes doivent être manuellement exprimées par le propriétaire. Cependant, il n'est pas question ici de lui imposer l'apprentissage d'une syntaxe de *triggers* à la SQL. Un module d'administration, appelé *Administration GUI* doit permettre l'expression de ces requêtes via une interface simple et intuitive, par exemple en suggérant au propriétaire de définir les sujets ou documents sensibles comme étant ceux avec le tag *privé* ou *sensible*. Et ceci n'empêche en rien les utilisateurs expérimentés d'exprimer directement des *watchdog triggers* en mode console.

Au même titre que l'*Advisor*, les *watchdog triggers* visent à simplifier l'administration des ACL et aident le propriétaire à détecter des permissions suspectes afin d'éviter toute fuite de données, tout en minimisant les actions requises de sa part. La combinaison de ces deux mécanismes permet à la fois d'exploiter les habitudes de partage du propriétaire pour détecter des partages inhabituels et d'exprimer des sujets, documents ou associations sensibles pour effectuer un contrôle systématique des permissions les concernant. Ceci constitue la **console d'administration** qui a pour rôle de nettoyer périodiquement l'ensemble d'ACL^* et remplir les ensembles d'ACL^+, ACL^- et d'$ACL^?$.

3.5 Architecture de référence SWYSWYK

Cette section présente une architecture de référence pour le Cloud personnel et qui implémente la propriété de **sécurité par construction.**

L'objectif de cette architecture est de protéger par construction les données du propriétaire, sans que ce dernier n'ait d'action à effectuer, hormis la validation des ACL suspectes. En reprenant le principe de *Privacy-by-Design* de [Cavoukian 2009], cela signifie contribuer aux principes de *Privacy-by-Default* (respect de la vie privée par défaut) et de *End-to-End security* (sécurité de bout en bout).

Cette architecture, présentée en Figure 7, distingue trois parties principales, chacune ayant différentes hypothèses en termes de confiance et de sécurité :

1. Un environnement de non-confiance, appelé ***Untrusted Environment* (*UE*).** Aucune hypothèse de sécurité ne peut être faite sur cet environnement, ni sur le code exécuté, ni sur les données qui doivent donc être chiffrées préalablement avant d'y être stockées.

2. Un environnement isolé, appelé ***Isolated Environment* (*IE*).** Du code arbitraire peut y être exécuté, avec la garantie qu'aucune information ne peut fuiter, mais aucune garantie sur la fiabilité et l'honnêteté des résultats, le code pouvant être corrompu.

3. Un environnement d'exécution sécurisé, appelé ***Secure Execution Environment* (*SEE*).** Les garanties de sécurité y sont très fortes ; seul du code certifié peut y être

exécuté et aucune application tierce n'est autorisée. Les données et le code y sont protégés contre l'espionnage et les attaques physiques.

Nous reportons à la section 3.6 la discussion concernant des exemples concrets illustrant les environnements *UE*, *IE* et *SEE*, et considérons les hypothèses de sécurité décrites ci-dessus comme acquises. Dans les sous-sections suivantes, nous présentons les différents composants de l'architecture, qui apparaissent dans la Figure 7.

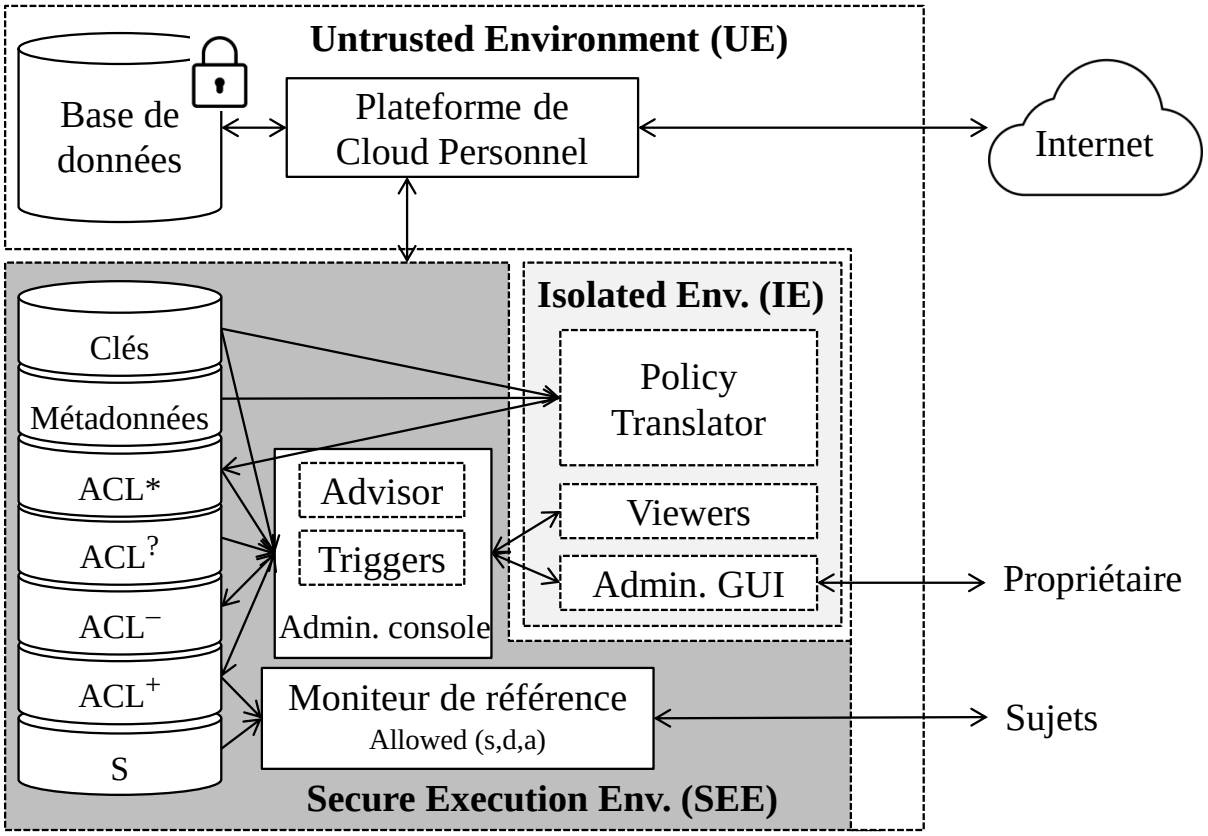


Figure 7. Architecture de référence.

3.5.1 Plateforme de Cloud personnel

Notre objectif est d'être agnostique vis-à-vis de la plateforme de Cloud personnel utilisée. Nous pouvons en effet difficilement justifier de choix architecturaux qui seraient uniquement motivés par la topologie de telle ou telle solution commerciale, alors qu'aucun consensus n'existe pour le moment, ni dans la littérature, ni sur le marché. Ainsi, aucune hypothèse ne peut être faite sur la sécurité intrinsèque de la plateforme de Cloud personnel : ouvert sur

Internet, il est possible d'y installer du code tiers via des applications que le propriétaire peut ajouter à souhait. C'est pourquoi on la considère comme faisant partie de l'environnement de non-confiance, l'*UE*. Tous les documents stockés dans la base de données de la plateforme doivent donc être chiffrés afin de les protéger contre des attaques de confidentialité.

La plateforme communique avec l'environnement sécurisé, le *SEE*, et lui transmet notamment tous les documents devant être chiffrés ou déchiffrés. Tous les accès des sujets aux documents doivent être transmis au moniteur de référence pour évaluer le contrôle d'accès.

3.5.2 Moniteur de référence

Le moniteur de référence fait partie du cœur de l'architecture et est particulièrement critique car c'est lui qui évalue si un accès et légitime ou non. Il doit donc être intégré à l'environnement sécurisé, le *SEE*, afin de garantir qu'il ne puisse être ni observé, ni contourné, ni corrompu par aucun acteur extérieur. Le moniteur de référence évalue la fonction *Allowed(s,d,a)* pour toute demande d'action *a* du sujet *s* sur le document *d*. La fonction répond *true* si et seulement si $<s,d,a> \in ACL^+$, et *false* sinon. Seul l'ensemble ACL^+ est pris en compte, qui correspond aux permissions validées par la console d'administration.

Le moniteur de référence agit comme un gardien incorruptible pour toute la plateforme de Cloud personnel. Chaque fois que *Allowed(s,d,a)* est évaluée à *true*, le document *d* est déchiffré dans le *SEE* avant d'être retourné au sujet *s.* Au vu de la simplicité de la fonction *Allowed*, le moniteur peut être embarqué dans de nombreux types de *SEE* résistant aux attaques physiques, qui ont la particularité d'être extrêmement contraints en termes de ressources, notamment mémoire. Par exemple une carte SIM, un *token* personnel sécurisé [Anciaux 2007], un *Trusted Platform Module*[40], etc.

L'authentification est également réalisée dans le moniteur de référence. Nous ne faisons pas d'hypothèse sur le mode utilisé : cela peut être un des systèmes décrits en section 2.2.1, par

[40] https://en.wikipedia.org/wiki/Trusted_Platform_Module

exemple une vérification de clé publique à la *Web of Trust* ou bien une simple vérification de mot de passe, une authentification biométrique, etc. Quel que soit le moyen utilisé, les authentifiants de chaque sujet sont stockés dans le *SEE*, dans l'ensemble *S.* Sa structure est détaillée dans la section 3.5.5.

Afin de pouvoir remplir l'ensemble ACL^{+}, la première étape est d'évaluer les règles de partage, ce qui est fait par un module appelé *Policy Translator.*

3.5.3 Policy translator

Le *Policy translator*, littéralement traducteur de politique, est en charge d'évaluer les règles du modèle de partage du Cloud personnel sur les documents et de matérialiser les ACL candidates dans *ACL**. Il s'exécute dans l'environnement isolé *IE*. En effet, il ne peut faire partie de l'*UE* car l'évaluation des règles peut potentiellement nécessiter de déchiffrer tous les documents de la base données et ainsi les exposer dans un environnement de non-confiance. Il ne peut faire partie du *SEE* non plus, à moins de casser le principe de généricité du modèle : en effet, la complexité du *Policy translator* est liée au pouvoir d'expression du modèle de partage et exécuter du code arbitraire potentiellement complexe dans le *SEE* peut s'avérer extrêmement difficile, voire dangereux, la complexité étant propice aux erreurs.

Exécuter le *Policy translator* dans l'*IE* ne compromet pas la sécurité, car par construction aucune donnée ne peut fuiter de cet environnement. Cette propriété est garantie en exécutant le code de l'*IE* dans des conteneurs isolés[41] disposant de droits restreints. Le *Policy translator* s'exécute à l'intérieur d'un conteneur avec un accès en lecture aux documents du Cloud personnel et leurs métadonnées, un accès en écriture aux ACL^{*} dans le *SEE,* et aucun privilège supplémentaire (en particulier, aucun accès réseau). Subséquemment, le *Policy translator* ne peut faire fuiter aucune donnée et ne peut pas être

[41] En pratique, des conteneurs isolés peuvent être implémentés en utilisant une plateforme matérielle dédiée (isolation physique), un hyperviseur ou un micro-kernel. Des travaux récents proposent un support matériel pour exécuter du code isolé, par ARM Trustzone [Alves 2004] ou des processeurs SGX [Costan 2016].

espionné par des processus concurrents, grâce aux propriétés des conteneurs isolés[42]. On ne peut en revanche pas garantir la légitimité des règles exécutées ni leur production, les ACL^{*}. D'où l'intérêt de la console d'administration et de ses mécanismes de vérification d'ACL décrits en sections 3.3 et 3.4.

3.5.4 Console d'administration

La console d'administration est utilisée dans le but d'aider le propriétaire du Cloud personnel à effectuer la validation des ACL. Comme détaillé dans les sections 3.3 et 3.4, cette console doit exécuter le processus d'*Advisor* et les *watchdog trigger* sur les ACL^{*} pour mettre les ACL suspectes en quarantaine dans $ACL^{?}$, les ACL légitimes dans ACL^{+} et les ACL refusées dans ACL^{-}. Cela nécessite un droit d'écriture sur ACL^{+}, ACL^{-}, ACL^{*} et un droit de lecture sur ACL^{*}.

Ces droits impliquent de devoir faire entièrement confiance à la console d'administration et donc l'exécuter dans le *SEE*. Elle ne peut cependant pas l'être entièrement ; en effet, des interactions avec le propriétaire sont nécessaires via une interface graphique, afin de lui permettre de visualiser le contenu des permissions, c'est-à-dire les documents et sujets concernés, via des applications de *Viewer*, et lui donner la possibilité de valider ou infirmer les ACL suspectes, via l'*Administration GUI*. Or les interfaces graphiques, même basiques, requièrent généralement une puissance de calcul et une mémoire dont ne disposent potentiellement pas les *SEE*, que l'on veut également préserver de tout code tiers ou complexe, ce qui les rend incompatibles pour des bibliothèques graphiques. C'est pourquoi ces modules doivent être exécutés en isolation dans l'*IE*, pour prévenir toute fuite de donnée.

[42] La véracité de cette information dépend toutefois de la robustesse des conteneurs utilisés, comme nous le rappellent les récentes failles Meltdown [Lipp 2017] et Spectre [Kocher 2017].

3.5.5 Structures de données internes

Les ensembles ACL^{*}, $ACL^{?}$, ACL^{+} et ACL^{-}, ainsi que les clés de chiffrement et les métadonnées des documents sont stockés dans le *SEE* pour des raisons évidentes de sécurité : ces ensembles sont particulièrement sensibles, tout le contrôle d'accès et donc la dissémination des données se reposant sur leur contenu. Les stocker dans l'*UE* serait techniquement possible tant qu'ils y restent chiffrés, mais cela impliquerait des coûts prohibitifs en déchiffrement et vérifications d'intégrité dans le *SEE*, durant l'exécution des modules *Allowed, Advisor* et *watchdog triggers.*

La contrepartie de la simplicité du moniteur de référence est le coût potentiel de l'exécution de la fonction *Allowed(s,d,a)* dans le cas où l'ensemble ACL^{+} serait extrêmement grand. Cette situation pourrait intervenir si une politique de partage associe un grand nombre de sujets à un grand nombre de documents et pourrait résulter en des millions d'ACL. Le stockage et l'indexation à la fois sur les *s* et les *d* d'une telle quantité d'ACL ne sont pas un problème dans un environnement traditionnel, mais peuvent devenir très compliqués dans un environnement sécurisé avec de très faibles ressources.

L'objectif est ici de trouver une représentation compacte de l'ensemble ACL^{+}. Il peut en réalité être représenté par un graphe bipartite $G = (S, D, A)$, où les ensembles de sommets *S* et *D* représentent respectivement les ensembles des sujets et des documents, et les arêtes *A* représentent les autorisations qui les lient, à travers des règles basiques ou réflexives. Chaque règle basique *BR* définit un sous-graphe bipartite complet G_{BR} de *G.* Ainsi, chaque G_{BR} peut être stocké de façon compressé comme un ensemble de sommets $S_{BR} \subseteq S$ de sujets et comme un ensemble de sommets $D_{BR} \subseteq D$ de documents; A_{BR} étant implicite. En revanche, trouver une décomposition des parties restantes de *G* correspondant aux règles réflexives en un ensemble de sous-graphes bipartites complets est complexe à construire et maintenir. De plus, le ratio de compression obtenu est incertain car il dépend de la distribution des ACL. En se reposant sur l'hypothèse que les arêtes communes sont prédominantes dans les règles basiques, nous suggérons de stocker *in extenso* toutes les arêtes de $G_{RR} \subseteq G$, l'union de tous les sous-graphes de *G* qui correspondent aux règles réflexives. De même, nous suggérons de stocker toutes les arêtes de $G_{except} = (S, D, A)$, qui représentent ACL^{-}, les exceptions étant assumées rares.

3.6 Validation de SWYSWYK

Pour valider notre approche, nous présentons ici une plateforme expérimentale qui est une implémentation de l'architecture de référence, illustrée sur la Figure 7. Cette plateforme est suffisamment générique pour en tirer des conclusions sur la faisabilité de la décliner en d'autres instances cibles.

Nous avons mené diverses évaluations de performances sur cette plateforme afin de prouver que le surcoût engendré reste raisonnable et offre des perspectives intéressantes. De plus, nous présentons une analyse qualitative qui vise à valider le concept du module d'*Advisor* sur des jeux de données réels.

3.6.1 Plateforme expérimentale

La plateforme expérimentale utilisée pour valider notre proposition se fonde sur la combinaison du Cloud personnel Cozy, présenté dans la section 2.3.2, et de PlugDB[43].

PlugDB est une plateforme sécurisée matérielle et logicielle développée dans l'équipe Petrus[44] de l'Inria Saclay. Elle combine les éléments suivants :

- Un *Secure Element* de type carte à puce, en charge des opérations cryptographiques et du stockage des secrets nécessaires.
- Un micro-contrôleur (MCU), qui exécute un moteur de base de données relationnelle requêtable en SQL et capable de gérer efficacement un grand volume de données [Anciaux 2007] avec une faible mémoire (quelques Ko).
- Une mémoire flash qui stocke la base de données chiffrée, pour se protéger des attaques actives et passives. Selon la déclinaison matérielle, cette mémoire peut être soit une puce soudée, soit une carte amovible de type microSD.

[43] https://project.inria.fr/plugdb/en/

[44] https://www.inria.fr/equipes/petrus

La combinaison de ces composants permet d'obtenir un matériel conçu pour résister aux attaques physiques et capable de gérer une grande volumétrie de données malgré des ressources très limitées, avec typiquement une centaine de Ko de RAM. Cette limitation en ressources s'explique à la fois pour des raisons financières – contrairement aux mémoires persistantes, la mémoire vive coûte cher – et pour des raisons de sécurité : plus la mémoire est grande et plus il est difficile de la protéger, or la mémoire vive est peu compacte.

PlugDB peut se décliner matériellement en diverses instances. Il est par exemple possible d'avoir des interfaces de communication USB, Bluetooth ou même Wi-Fi. De même, l'authentification peut se faire physiquement, par un code PIN ou un module d'empreinte digitale. Nous utilisons ici une version qui dispose de deux interfaces de communication, une USB et une Wi-Fi.

La plateforme de Cloud personnel utilisée est Cozy qui utilise une base de données CouchDB, un moteur NoSQL qui représente les données en JSON et fournit un langage de requêtes et d'indexation pour les manipuler. Nous utilisons un modèle de partage simple, implémenté dans la *stack* de Cozy, capable d'exprimer des partages sur des sélections de documents avec des sélections de sujets. Par exemple : « Partager l'album photo 'Road-trip post-thèse' avec le groupe 'Famille' ». Les règles de partage sont produites par des applications Cozy et évaluées par le *Policy Translator*.

La plateforme expérimentale, présentée en Figure 8, est une instance de l'architecture de référence détaillée dans la section 3.5. Chaque environnement d'exécution est distinct des autres, avec un matériel et du logiciel qui lui est propre :

1. L'environnement de non-confiance (*UE*) est représenté par un ordinateur personnel relié à internet, sur lequel une instance Cozy s'exécute. Le système d'exploitation est Ubuntu[45] 16.04, une suite GNU/Linux populaire basée sur Debian[46].

2. L'environnement isolé (*IE*) est joué par un Raspberry Pi[47], sur lequel tournent le *Policy translator*, *l'Administration GUI* et le *Viewer.* Aucune connexion réseau n'y est

[45] https://ubuntu-fr.org/

[46] https://www.debian.org/index.fr.html

disponible et la seule interface de communication possible est celle utilisée pour relier le *SEE*.

3. L'environnement sécurisé (*SEE*) est représenté par PlugDB, dans lequel le moniteur de référence et la console d'administration sont exécutés. Les données sont stockées sur une carte flash microSD et chiffrées depuis une carte SIM de type BasicCard[48]. Afin de pouvoir requêter et manipuler les données stockées, PlugDB embarque un moteur relationnel, MiloDB [Anciaux 2014], spécialement conçu pour gérer efficacement de grands volumes de données avec les contraintes des mémoires flash dans des environnements avec peu de ressources. PlugDB est branché en USB High Speed 2.0) au Raspberry Pi et communique en Wi-Fi IEEE 802.11n avec l'ordinateur personnel qui exécute le Cozy.

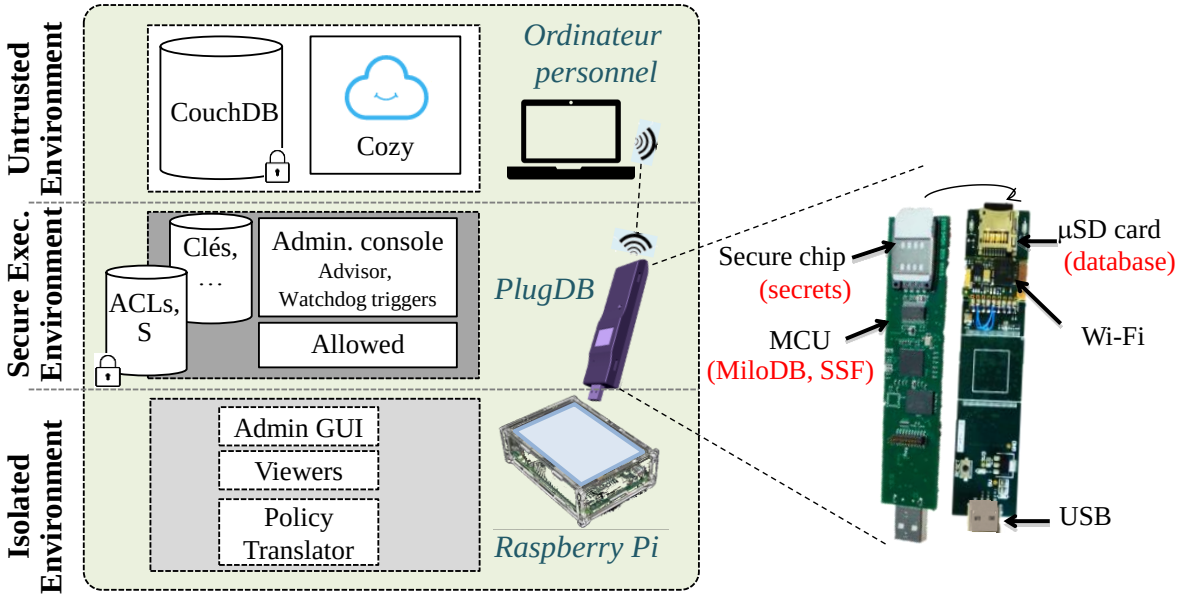


Figure 8. Plateforme expérimentale.

[47] https://www.raspberrypi.org/

[48] http://basiccard.com/

Pouvoir exécuter le moniteur de référence qui est le garant de l'application du contrôle d'accès, dans un environnement aussi contraint que celui de PlugDB est un gage de sa simplicité et de la possibilité de formellement prouver le code. Cela démontre également la possibilité d'embarquer un moteur SWYSWYK dans n'importe quel type de *SEE*, y compris provenant de l'Internet des Objets, ou *Internet of Things* (IoT), qui sont généralement de faible puissance.

Dans l'*IE*, la propriété d'isolation est *physiquement* garantie par le Raspberry Pi, sur lequel les ports réseaux ont été rendus inutilisables. Aucune communication autre que celles passant par le *SEE* ne sont donc possibles. Ainsi, un code frauduleux n'aurait que peu d'impact dans cet environnement et pourrait au pire générer des permissions suspectes, ou rendre inutilisable l'*IE*. Mais en aucun cas exposer des données personnelles du propriétaire.

D'autres instances d'architectures peuvent être envisagées à terme, plus simples à mettre en œuvre et plus adaptées à des scénarios industriels. Par exemple, l'*IE* pourrait être implémentée par un hyperviseur certifié qui s'exécuterait sur un processeur Intel SGX [Costan 2016], directement dans l'ordinateur personnel (Figure 9, partie gauche). De même, des appareils équipés d'une carte SIM, une mémoire flash et un processeur ARM Trustzone [Alves 2004] – comme la majorité des téléphones et tablettes aujourd'hui – sont d'autres options envisageables (Figure 9, milieu et partie droite). La différence majeure entre ces alternatives est dans la façon dont les environnements *UE, IE et SEE* communiquent. Dans l'évaluation de performance que nous présentons dans la section suivante, nous mesurons et isolons le coût de communication afin de pouvoir en tirer des conclusions générales sur l'architecture.

Dans nos expérimentations, l'ordinateur personnel qui implémente l'*UE* dispose d'un processeur Intel Xeon E5-1660 CPU cadencé à 3.0 GHz, 8 Go de mémoire RAM et d'un disque dur magnétique de 500 Go qui tourne à 10 000 rotations par minute. Le *SEE* incarné par PlugDB utilise un MCU de type STM32 F417GH6, qui embarque un micro-processeur ARM Cortex M4 à 168 MHz, 192 Ko de RAM et 1 Mo de stockage NOR. Enfin, l'*IE* est un Raspberry Pi 3, qui comprend un processeur ARMv8 de 1,2 GHz et 1 Go de RAM. Une carte microSD externe de type UHS-I de 16 Go est utilisée à la fois pour le *SEE* et l'*IE*.

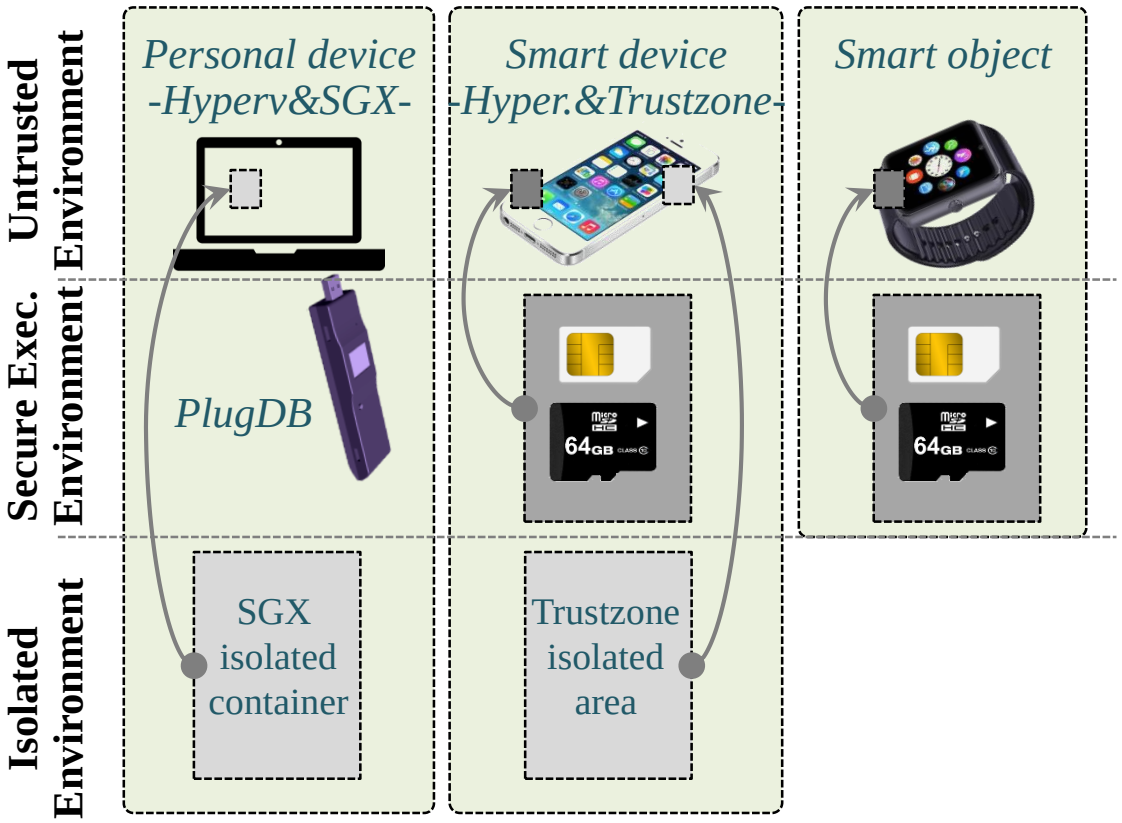


Figure 9. Exemples d'autres instances d'architecture.

3.6.2 Coûts d'environnement

Nous mesurons tout d'abord le coût total d'environnement lors de l'insertion d'un nouveau document dans le Cozy. La Figure 10 indique les quatre composantes de ce coût :

1. Le coût de transfert Wi-Fi d'un document entre l'*UE* et le *SEE*. A noter que ce coût comprend l'aller-retour, l'aller étant le document en clair envoyé par l'*UE* et le retour le chiffré renvoyé par le *SEE*.

2. Le coût de chiffrement du document par le *SEE.* Chaque document est chiffré par une clé AES 128 unique, stockée dans PlugDB. Ce type de chiffrement symétrique à le double avantage d'être efficace et extrêmement robuste [Nechvatal 2000].

3. L'insertion du document chiffré dans le Cozy. Cela correspond à une entrée/sortie dans le moteur de base de données CouchDB.

4. Le coût de transfert USB du document en clair du *SEE* au *IE*, afin qu'il soit évalué par le *Policy Translator* pour vérifier s'il rentre dans des règles de partage.

A noter que la 4ème étape peut être effectuée en parallèle du reste, ce qui explique pourquoi nous isolons le coût de transfert USB dans la Figure 10.

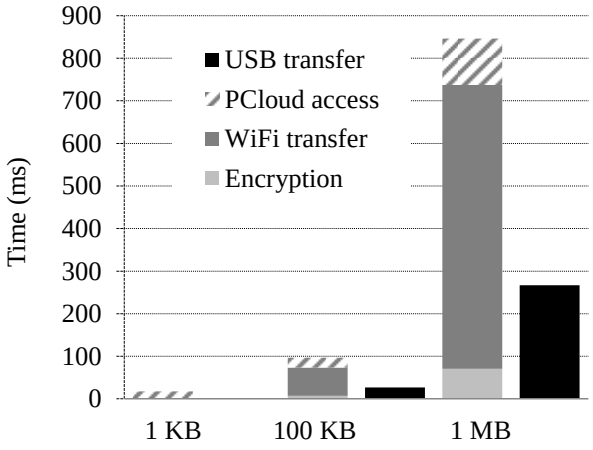


Figure 10. Coûts d'environnement pour l'insertion d'un document.

On constate que le coût d'environnement est négligeable pour les documents de petite taille, c'est-à-dire inférieur à 1 Mo. Pour des documents plus volumineux, le transfert WiFi domine assez largement le coût global, suivi par le transfert USB. Il faut néanmoins préciser que ces coûts de communications pourraient être grandement diminués en utilisant les dernières normes Wi-Fi et USB : le Wi-Fi 802.11ac et l'USB 3.1 offrent des débits respectivement 2,9 et 20,8 fois plus rapides que les normes 802.11n et USB 2.0. Des optimisations au niveau du *firmware*[49] utilisé dans PlugDB sont également des pistes à explorer pour améliorer les débits.

De plus, ces coûts sont grandement liés à la façon dont la plateforme expérimentale est implémentée, plutôt que liés au système SWYSWYK lui-même. En effet, les architectures alternatives présentées en Figure 9 réduiraient drastiquement ces coûts: les échanges Wi-Fi disparaîtraient, et l'USB ne serait présent que dans le cas d'un ordinateur relié à PlugDB.

[49] https://fr.wikipedia.org/wiki/Firmware

Enfin, il est aussi intéressant de noter que le coût du chiffrement reste faible, alors qu'il a lieu dans le *SEE*, qui n'a pourtant que peu de ressources. Cela s'explique par la conception même de PlugDB, qui utilise du matériel dédié et optimisé pour les opérations cryptographiques.

3.6.3 Impact du moniteur de référence

La fonction *Allowed* prend en entrée un triplet *(s,d,a)* et retourne *true* si le sujet *s* a l'autorisation *a* sur le document *d.* Dans tout autre cas, *false* est retourné. Comme expliqué dans la section 3.5.5, exécuter cette fonction se résume à vérifier l'existence du triplet dans l'ensemble ACL^{+}, qui est représenté de façon compacte par un graphe bipartite pour chaque règle de partage. Cela amène à stocker les ACL en tables relationnelles de la forme :

Rule (RuleId, A)

NodeS (RuleId, SubjectId)

NodeD (RuleId, DocId)

NodeS contient les ID des sujets tandis que *NodeD* stocke les ID des documents concernés. *Rule* stocke les actions pour chaque permission. A noter que ces ID correspondent à l'identifiant unique attribué à chaque document de CouchDB.

Ainsi, la function *Allowed* peut être évaluée par une simple requête SQL sur ces tables. *NodeS* et *NodeD* pouvant être potentiellement volumineux, nous indexons donc les attributs de ces tables pour s'assurer que la requête soit traitée rapidement. Cependant, le système d'indexation dans PlugDB n'est pas traditionnel : il a été spécialement conçu pour gérer des environnements avec de larges volumes de données stockés sur mémoire flash (plusieurs Go) et très peu de RAM (quelques Ko). Généralement, dans ce type d'environnement contraint, les données sont massivement indexées pour minimiser le besoin de stocker des données intermédiaires en mémoire et assurer un temps de réponse acceptable. Malheureusement, les structures d'index généralement utilisées [Sundaresan 1999] [Weininger 2002] génèrent de nombreuses écritures aléatoires (en opposition aux écritures séquentielles), qui sont très mal supportées par la mémoire flash [Bouganim 2009].

A la place, nous utilisons les index présentés dans [Anciaux 2007] qui ont fait l'objet d'une implémentation dans PlugDB, détaillée dans [Anciaux 2014] et appliqués dans le contexte

des objets connectés dans [Anciaux 2015] puis du Cloud personnel sécurisé dans [Le 2016]. En quelques mots, ces index permettent d'évaluer des requêtes en *pipeline*, avec une consommation mémoire minimale et en évitant autant que possible des écritures aléatoires. Ils combinent des filtres de Bloom [Bloom 1970], qui sont des structures probabilistes compressées, donc pouvant tenir en mémoire vive, avec des structures d'accès séquentielles. Les mises à jour et suppressions sur les données ne sont pas directement appliquées, mais stockées dans des logs dédiés qui font partie intégrante de l'évaluation d'une requête. Pour éviter que ces logs ne deviennent trop gros et impactent négativement les temps d'évaluation, un processus de réorganisation est effectué pour purger ces structures et appliquer les mises à jour séquentiellement. Comme cela est détaillé dans [Anciaux 2015], cette réorganisation peut s'avérer coûteuse. Cependant dans notre cas, les mises à jour et suppressions sont à priori rares et correspondent à des ajustements de la politique de partage par le propriétaire, ce qui minimise le besoin de réorganisation. Si cela devait néanmoins arriver, des stratégies opportunistes permettent de l'effectuer durant les périodes où PlugDB n'est pas sollicité, et de la mettre en pause si des requêtes arrivent.

Ici, nous utilisons des index de sélections, appelés *climbing index* (CI), dans les tables *NodeS* et *NodeD*, sur les attributs *SubjectId* et *DocId*. Chaque entrée de l'index est associée à une liste d'ID de la table parent, ici, *Rule*.

La requête SQL pour évaluer *Allowed(s,d,a)* est la suivante :

```
SELECT    r.RuleId
FROM      Rule r, NodeS ns, NodeD nd
WHERE     ns.SubjectId = s AND nd.DocId = d AND ns.RuleId = r.RuleId AND nd.RuleId =
            r.RuleId AND r.A = a;
```

Pour évaluer cette requête, les deux CI sont parcourus afin de trouver les occurrences du sujet *s* et du document d recherchés. Les index retournent chacun une liste de RuleId : une intersection est faite entre ces deux listes, qui retourne ainsi les autorisations concernant à la fois *s* et *d*. Un opérateur de sélection est appliqué sur ces tuples pour trouver celui qui correspond à l'action *a*. Enfin, une projection est faite pour retourner l'identifiant de la règle concernée. *Allowed* retourne *true* si la projection renvoie un résultat et *false* s'il n'y en a pas.

La Figure 11 montre le temps d'exécution de *Allowed* en fonction du nombre d'ACL total. On constate que les temps restent acceptables, c'est-à-dire moins de 1 seconde, même avec un million d'ACL, alors que ce cas est rendu peu probable grâce à la représentation compacte. Cette fonction est particulièrement critique en termes de performance, car elle doit être évaluée à chaque demande d'accès d'un sujet pour un document ; réussir à maintenir bas le coût de cette évaluation dans un matériel sécurisé et résistant aux attaques physiques démontre ainsi l'intérêt et l'efficacité de l'approche.

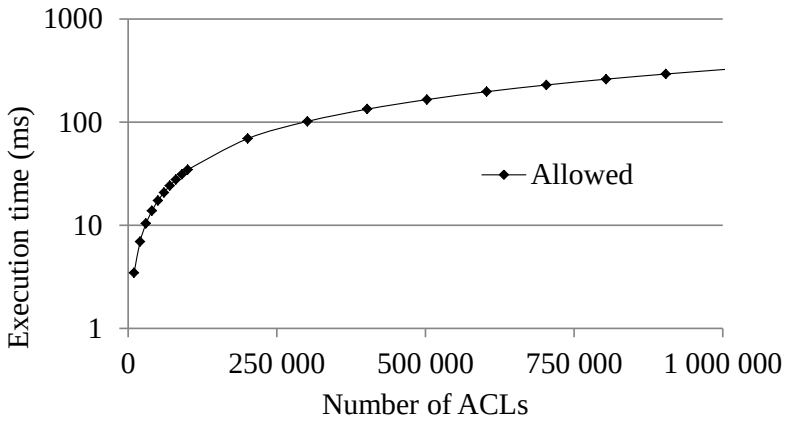


Figure 11. Temps d'exécution de *Allowed* dans le *SEE.*

Néanmoins, cette évaluation ne serait pas complète sans avoir la possibilité de valider la pertinence de la console d'administration et en particulier du module d'*Advisor*, qui est au cœur de l'automatisation de la validation des ACL et promet donc au propriétaire de n'avoir qu'un minimum d'actions manuelles à effectuer.

3.6.4 Analyse qualitative de la validation des ACL

Nous évaluons ici la pertinence de la fonction *Resolve*, décrite à la section 3.3, qui est centrale au module d'*Advisor*. Nous l'appliquons sur des jeux de données réels : en effet, évaluer qualitativement cette fonction sur des données synthétiques n'aurait que peu de sens car non représentative d'usages réels.

N'ayant pas connaissance de jeux de données publiques d'ACL, nous avons choisi des courriels personnels pour cette analyse. De célèbres comptes d'e-mails ont été rendus publics et utilisables par la communauté ; de plus, les courriels reflètent par certains aspects

la politique des utilisateurs en termes de dissémination de leurs données personnelles [Roth 2010].

Afin d'être le plus représentatif possible, nous utilisons ici quatre jeux de données provenant de sources distinctes : la base publique de l'entreprise Enron [Klimt 2004], les courriels d'Hillary Clinton, révélés par Wikileaks en 2016 [Clinton 2016], des courriels personnels de l'auteur de ce manuscrit (appelé *People1*), et ceux d'un membre de sa famille (appelé *People2*).

Dans cette expérimentation, nous considérons le corps du courriel comme étant un document personnel d_i donné en lecture par l'émetteur aux destinataires $\{s_j\}$. Les ACL^+ correspondent ici à l'ensemble {(s_j, d_i, '*read*')}. Les ACL^- ne sont pas prises en compte dans ce contexte.

Pour générer les ACL^*, nous avons choisi aléatoirement des éléments « légitimes », c'est-à-dire sélectionnés dans l'ensemble ACL^+, et des éléments « illégitimes », n'existant pas dans ACL^+. Chaque ACL légitime sélectionnée est ensuite retirée des ACL^+. Les éléments illégitimes sont générés en combinant des e-mails *d* de l'ensemble des $\{d_i\}$ avec des destinataires *s* de l'ensemble des $\{s_j\}$, n'étant pas des destinataires de l'e-mail *d*. Idéalement, la fonction *Resolve* devrait retourner *Accept* pour les ACL légitimes et *Suspect* pour les ACL illégitimes, afin qu'elles soient stockées dans $ACL^?$.

La fonction utilisée pour évaluer la distance entre les historiques de deux sujets se résume à compter le nombre de courriels où ces deux sujets apparaissent tous les deux comme destinataires, puis appliquer la fonction inverse. Ce calcul est suffisamment simple pour pouvoir être implémenté en SQL dans PlugDB en utilisant l'opérateur COUNT :

```
SELECT    COUNT(nd.DocId)
FROM      Rule r, NodeD nd
WHERE     r.RuleId= nd.RuleId and r.RuleId IN (
          SELECT r.RuleId FROM Rule, NodeS ns where r.RuleId= ns.RuleId AND ns.SubjectId
          = 's1'
          INTERSECT
          SELECT r.RuleId FROM Rule, NodeS ns where r.RuleId= ns.RuleId AND ns.SubjectId
          = 's2'
```

);

Pour chaque élément (d_i, s_j) de ACL^*, on calcule la distance entre s_j et les autres destinataires du courriel, c'est-à-dire les sujets ayant un accès à d_i dans ACL^+. Puis, on garde le sujet ayant la plus petite distance. Si cette distance est en-dessous d'un seuil minimum *t,* préalablement fixé, *Resolve* accepte l'ACL et la place dans ACL^+. Sinon, l'ACL est placée en quarantaine dans $ACL^?$.

Dans la Figure 12, nous montrons le résultat de l'évaluation avec 50 éléments dans ACL^*, la moitié étant légitime. Les courbes sont le résultat du processus répété 1 000 fois, en conservant la moyenne de toutes les exécutions. Nous montrons le résultat pour chaque jeu de données, en faisant varier le seuil *t.* Plus *t* est petit, plus un élément de ACL^* doit avoir une distance proche d'un élément de ACL^+ pour être accepté par *Resolve*. Cela explique les allures respectives des courbes *Accept* et *Suspect,* qui représentent le taux de succès d'un élément de ACL^* correctement catégorisé en ACL^+ ou $ACL^?$.

La valeur optimale de *t* correspond au point de croisement entre les deux courbes, où le taux de succès se situe entre 68 et 83 % selon les jeux de données. Cette valeur optimale est différente pour chaque utilisateur car elle est fonction de ses habitudes de partage. Typiquement, le seuil optimal de Clinton est grand, car son carnet d'adresse est gigantesque, ce qui entraine une vaste dispersion dans la distribution entre destinataires. En revanche, *People1* a un comportement beaucoup plus intime, avec des partages sur un nombre réduit de destinataire.

Ainsi, l'utilisation d'une métrique simple pour le calcul de distance produit des résultats prometteurs, tout en restant compatible avec un environnement aux ressources limitées. Afin d'améliorer le taux de succès, d'autres variables pourraient être intégrées dans le processus de décision, par exemple en y ajoutant une dimension temporelle : un partage avec un sujet avec qui l'on a échangé fréquemment ces derniers mois est probablement moins suspect que d'autres avec lesquels l'on n'a maintenant aucun contact depuis des années). De même, la proximité sémantique entre les documents et sujets peut être explorée : si l'on a tendance à partager les mêmes types de documents avec les mêmes personnes, un éloignement soudain dans ces habitudes pourrait entrer en considération.

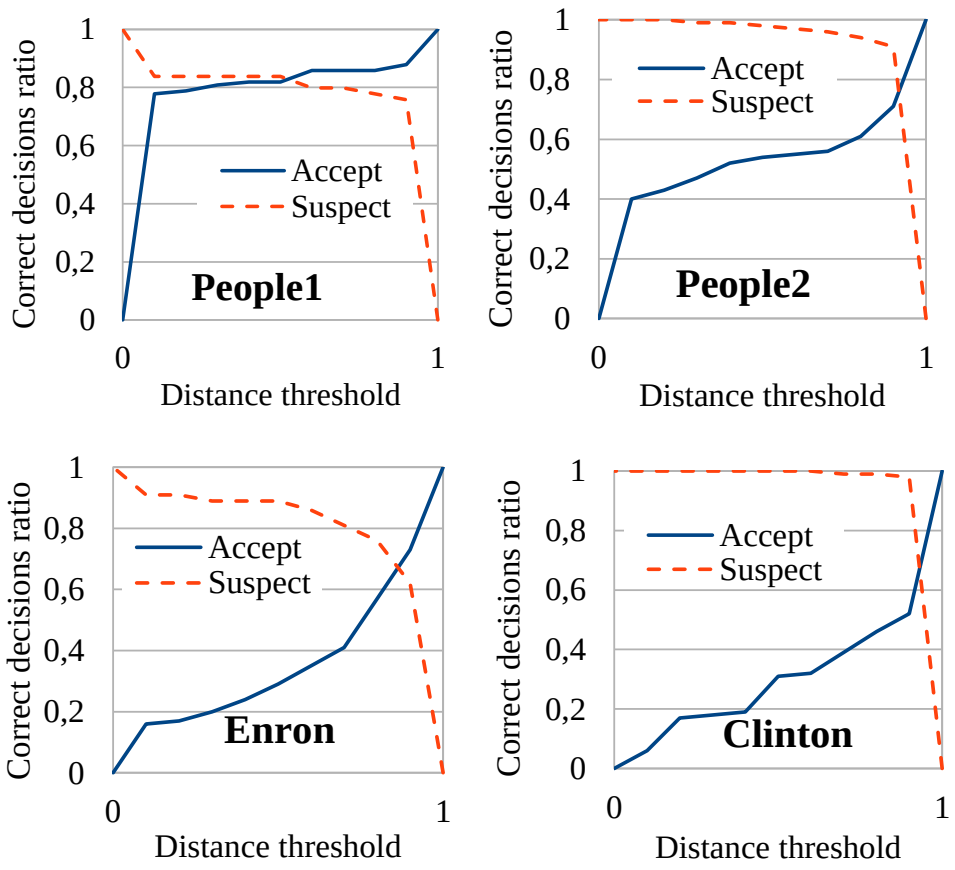


Figure 12. Taux de succès de la fonction *Resolve* pour Accept et Suspect

3.7 Démonstration

Afin de démontrer un cas d'usage concret de notre architecture, nous présentons ici un démonstrateur de la plateforme expérimentale décrite dans la section précédente dont nous donnons un aperçu en Figure 13. Le but est de démontrer les principes ***d'empowerment éclairé*** et de **sécurité par construction** en donnant un aperçu concret de l'architecture et de son fonctionnement général ; en particulier nous nous attardons sur l'utilisation de l'*Administration GUI*, qui permet au propriétaire de visualiser et manipuler ses ACL, et de faire ressortir des permissions suspectes grâce aux *watchdog triggers* et à l'*Advisor*. Ce démonstrateur a été présenté lors de la conférence ADMA, en novembre 2017 [Tran-Van, SWYSWYK: A New Sharing Paradigm for the Personal Cloud. 2017].

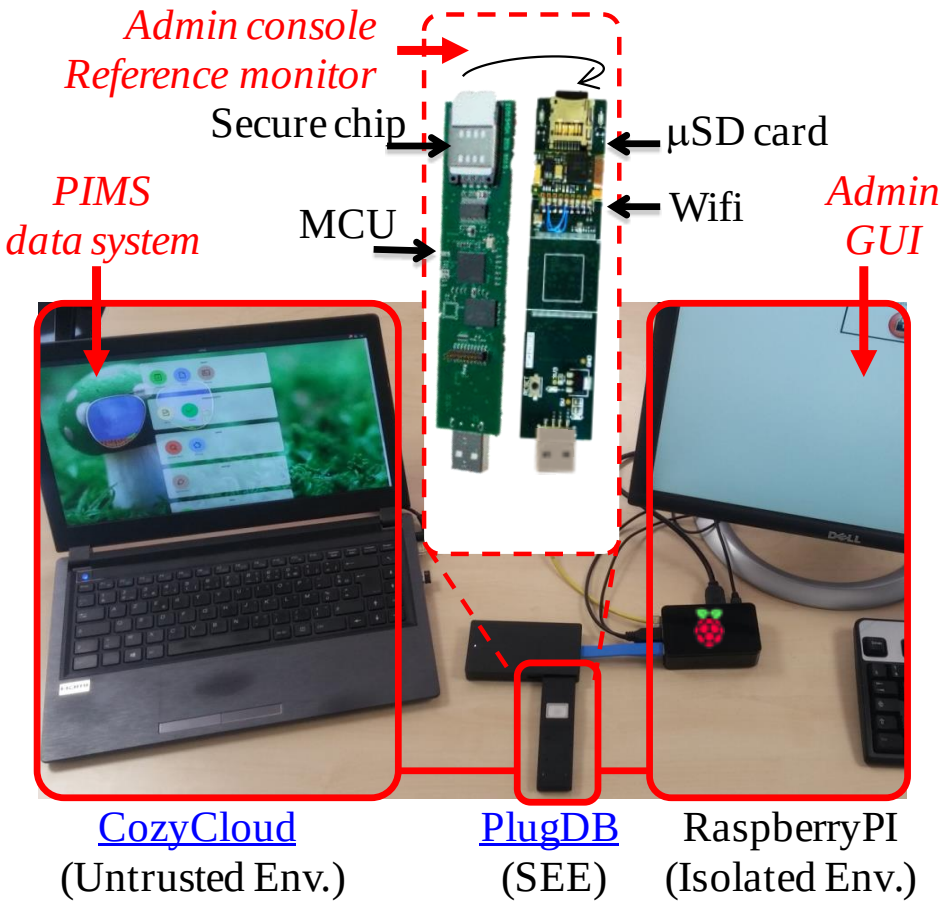


Figure 13. Plateforme de démonstration

3.7.1 Scénario de la démonstration

Pour les plus curieux, une vidéo du déroulement de la démonstration est disponible en ligne[50].

La première partie de la démonstration (étapes 1 à 5 de la Figure 14) montre l'utilité de SWYSWYK en termes de protection de la vie privée. Nous utilisons une instance de Cozy initialisée avec des applications, un ensemble de documents et des règles de partage.

[50] : http://wanda.inria.fr/demos/videos/swyswyk_archi.ogg

Lors de la démonstration, les participants peuvent s'authentifier en tant que Alice, propriétaire du Cozy. Une instance distante de Cozy, appartenant à Bob, contact d'Alice, est également installée. Les participants peuvent librement explorer le Cozy d'Alice. Depuis l'application *Contacts,* ils découvrent des fiches contacts avec des photos représentant les sujets ; parmi eux se trouve Bob qui est le supérieur hiérarchique d'Alice. Dans le Cozy se trouve également un ensemble d'images, certaines plus ou moins confidentielles, comme la photo de son ventre gonflé, destinée à un usage médical. Alice souhaite que (i) ses photos médicales ne soient partagées qu'avec ses médecins et (ii) seuls les documents anodins soient partagés avec Bob, son manager.

Les participants créent une nouvelle règle de partage depuis l'application *Files* de Cozy, via une interface graphique dédiée. Cette règle leur permet de partager un répertoire avec un ensemble de sujets. Les participants sont alors invités à vérifier la politique de partage d'Alice et à tenter de comprendre les droits accordés, en accédant directement aux règles de partage existantes ainsi qu'à celle nouvellement créée. Ceci se fait via une application dédiée qui joue le rôle d'*Adminnistration GUI*. Cette application s'exécute dans l'environnement isolé (*IE*) sur le Raspberry Pi. Les règles sont représentées dans un tableau, indiquant les prédicats nécessaires pour que des documents et des sujets soient légitimes. Bien que cela soit utile aux utilisateurs avancés, cela demande une expertise poussée, probablement hors de portée de la plupart des individus. De plus, l'augmentation du nombre de règles de partage dans le Cloud personnel rend ce travail de plus en plus fastidieux.

Heureusement, l'interface permet surtout de visualiser l'ensemble des ACL générées par les règles de partage. Grâce à cela, les participants découvrent alors que Bob a un accès à des photos compromettantes d'Alice – ce qui est confirmé après s'être authentifié sur le Cozy de Bob. Malheureusement, le grand nombre d'ACL rend une vérification manuelle exhaustive prohibitive en termes de temps ; Alice n'est donc pas parvenue à s'apercevoir de ce partage non voulu et ne peut être sûre qu'il n'y ait pas d'autres partages non souhaités.

Les participants utilisent alors le module d'*Advisor,* afin d'identifier les ACL suspectes. En complément à cela, ils créent également des *watchdog trigger,* pour détecter automatiquement des ACL concernant des sujets, documents ou associations sensibles. Pour simplifier ce processus, des *watchdog triggers* prédéfinis sont disponibles et simplement activables. Par exemple, exprimer que les photos médicales et le sujet Bob sont

respectivement des documents et sujet sensibles. Grâce à ces outils, les participants sont maintenant en mesure de détecter des ACL qui violent potentiellement la vie privée d'Alice, automatiquement mises en quarantaine. Enfin, pour chaque ACL suspecte, les participants peuvent choisir de l'accepter ou la refuser, ce qui a pour effet de la déplacer dans l'ensemble des ACL^+ ou ACL^-.

La deuxième partie de la démonstration se concentre sur les propriétés de sécurité et sur ce qui se passe concrètement lors de la création d'une règle de partage. Les participants jouent le rôle d'un attaquant et créent une règle de partage malveillante, destinée à violer la vie privée d'Alice. Un module de l'interface affiche ce qui est exécuté sur le Cozy (*UE*), le Raspberry Pi (*IE*) et PlugDB (*SEE*). Grâce à ce module, les participants peuvent comprendre où et quand les données sont déchiffrées, la console d'administration exécutée et les décisions de contrôle d'accès prises. Forts de cette compréhension, les plus hardis peuvent tenter d'élaborer des modèles d'attaque, qui seront – nous l'espérons – voués à l'échec.

La troisième et dernière partie de la démonstration se concentre sur les performances, le passage à l'échelle et la compatibilité avec des environnements d'exécution sécurisés à faibles ressources comme l'est PlugDB. Cette partie de la démonstration se déroule dans un module de l'interface qui permet de générer aléatoirement des ACL concernant des sujets et documents du Cozy. Les performances de la fonction *Allowed* et des *watchdog triggers* sont présentés sous forme de graphe avec des statistiques détaillées sur les temps d'exécution, la consommation mémoire et le coût des entrées / sorties.

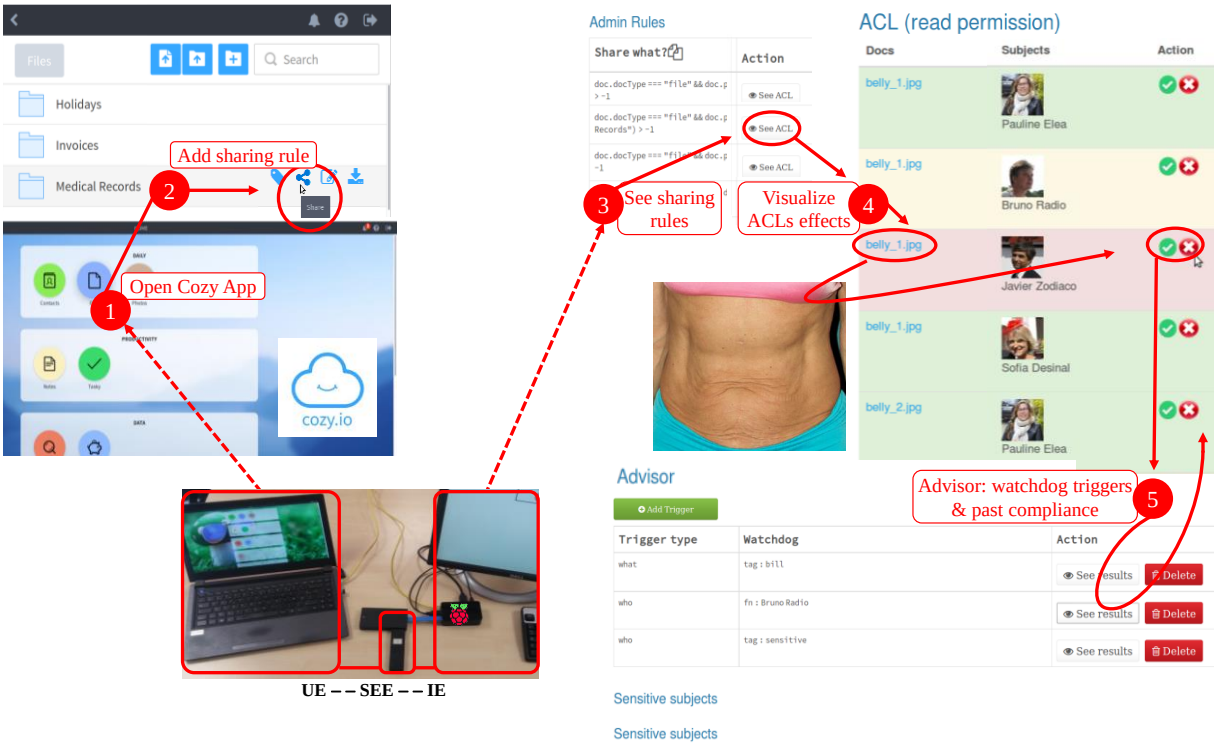


Figure 14. Scénario de démonstration

3.7.2. Bilan de la démonstration

La démonstration illustre les trois propriétés suivantes liées au paradigme SWYSWYK :

- **Simplicité d'utilisation :** le principe de la démonstration n'est pas de pouvoir exprimer des règles de partage complexes ; pour cela, une interface simple depuis une application Cozy est utilisée. Mais plutôt de valider et assainir les ACL générées par ces règles. La démonstration met en lumière l'efficacité des *watchdog triggers* et du processus d'*Advisor* afin d'aider le propriétaire à détecter et filtrer les ACL compromettantes, même sur de larges ensembles.

- **Sécurité par construction :** la combinaison des documents stockés en chiffrés dans l'*UE*, de l'isolation du code dans l'*IE*, et d'un moniteur de référence embarqué dans un *SEE* résistant aux attaques physiques apporte des garanties fortes au propriétaire contre les attaques de confidentialité dont sont régulièrement victimes les systèmes d'information traditionnels. De plus, la validation de l'approche dans un environnement physiquement contraint tel que PlugDB est une preuve de la simplicité

du moniteur de référence et des outils d'administrations associés. Par ailleurs, cette simplicité ouvre la voie à une validation formelle du code embarqué, qui reste suffisamment simple, comme discuté dans la section 3.6.

- **Performances :** nous avons mesuré que le temps d'exécution des requêtes Cozy sont impactées, à cause du chiffrement et surtout des coûts de communication entre *UE*, *IE* et *SEE*, comme cela a été discuté dans la section 3.6.2. Cependant, le démonstrateur prouve que ce surcoût reste acceptable pour un usage raisonnable de la plateforme, c'est-à-dire hors fichiers binaires de grande taille. Ceci valide la faisabilité de l'approche avec la plateforme expérimentale actuelle et ce, alors que d'importants gains de performances peuvent être espérés avec du matériel fournissant de meilleurs débits de communication et des *firmwares* plus effiaces.

Ces trois propriétés entrent parfaitement dans les principes ***d'empowerment éclairé*** et de **sécurité tangible** et permettent ainsi de s'assurer qu'une utilisation concrète de l'architecture SWYSWYK apporte de réels bénéfices au niveau de la protection de la vie privée de l'utilisateur, tout en permettant de garder un usage raisonnable de son Cloud personnel.

3.8 Conclusion

Dans ce chapitre, nous avons présenté le paradigme de *Privacy-by-Design* SWYSWYK, dédié au contexte du Cloud personnel et présenté à la conférence ISD en septembre 2017 [Tran-Van, SWYSWYK: a privacy-by-design paradigm for personal information management systems. 2017].

Ce paradigme permet à chaque propriétaire de visualiser les effets concrets de leurs règles de partage sur leur plateforme de Cloud personnel et de les corriger facilement si besoin, afin que leur politique de partage soit parfaitement en phase avec leur modèle mental. **L*'empowerment éclairé*** donne cette assurance aux individus, les rendant pleinement maîtres de leurs données et de leurs expositions sur Internet.

De plus, ce paradigme fournit des garanties **tangibles** quant à **l'application sécurisée** des politiques de partage, sans avoir besoin de l'intervention du propriétaire. L'utilisation d'un

environnement sécurisé dans lequel s'exécute un moniteur de référence à la logique triviale lui permet de s'assurer que son contrôle d'accès est correctement appliqué par construction et n'est pas dépendant de règles complexes impossibles à appréhender. La console d'administration lui permet également de s'assurer que le modèle de partage ne dévoile pas plus de documents que souhaité et de détecter les partages suspects, soit parce qu'ils englobent des documents, sujets ou associations sensibles, soit parce que le partage s'éloigne des habitudes du propriétaire.

Enfin, nous avons démontré que cette approche reste réaliste à travers une évaluation de performances réalisée sur une plateforme expérimentale intégrant un matériel sécurisé embarquant un moteur relationnel de base de données, PlugDB, et une plateforme de Cloud personnel, Cozy, éditée par l'entreprise Cozy Cloud. Cette plateforme a également été l'objet d'un démonstrateur [Tran-Van, SWYSWYK: A New Sharing Paradigm for the Personal Cloud. 2017] afin d'illustrer des cas concrets d'utilisation de SWYSWYK et de valider l'intérêt de l'architecture et de ses outils.

De nombreuses problématiques restent cependant à investiguer. Une première perspective est liée à l'analyse de sécurité de SWYSWYK, qui pourrait être conduit pour d'autres architectures, par exemples fondées sur des processeurs ARM Trustzone [Alves 2004] ou Intel SGX [Costan 2016], qui offrent de nouvelles perspectives au travers d'environnements d'exécution isolés, directement depuis le microcontrôleur de l'appareil utilisé.

Une autre perspective de recherche est la définition d'outils puissants et intuitifs pour aider le propriétaire à détecter des permissions suspectes. L'utilisation d'algorithmes évolués de *machine learning* distribués et respectueux de la vie privée pourrait être envisageable. Dans cette approche, chaque propriétaire de Cloud personnel pourrait contribuer à l'apprentissage de l'algorithme, de façon anonyme. Une fois suffisamment entraîné, il deviendrait capable de détecter automatiquement des règles suspectes, tout en s'adaptant intelligemment aux spécificités propres à chaque utilisateur. Ainsi, la combinaison de modèles communautaires et de préférences propres à chacun ouvrirait la voie à un système de détection de comportements suspects puissants et totalement respectueux de la vie privée.

Trouver de nouveaux moyens de partager les données personnelles des utilisateurs simplement, intuitivement, et de façon sécurisée est essentiel. Autrement, le risque est grand de voir les individus frustrés soit déléguer l'administration de leur Cloud personnel à des

fournisseurs de services centralisés, soit tout simplement se détourner de ces plateformes pour revenir (ou rester) dans les bras des GAFAM, avec tous les problèmes de vie privée et de sécurité que cela implique.

L'adoption de ces modèles passe par de nouveaux usages innovants et simples d'utilisation. Pour cela, la sécurité seule n'est pas suffisante, comme le disait déjà Bruce Schneier en 2000 [Schneier 2000] et les spécificités du Cloud personnel appellent à la définition d'un nouveau modèle de partage, capable de tirer pleinement partie de la richesse des données présentes. Cela, évidemment, tout en restant totalement respectueux de la vie privée des utilisateurs, c'est-à-dire en ne partageant pas plus que le strict nécessaire et en laissant le propriétaire toujours maître des décisions et informé des effets de sa politique de partage. Nous discutons dans le chapitre suivant de la partie modèle de SWYSWYK, complémentaire à l'architecture.

Chapitre 4
Modèle SWYSWYK

Dans ce chapitre, nous présentons la partie **modèle de partage** de SWYSWYK qui a été publié pour la conférence TrustBus' 17 [Tran-Van, A New Sharing Paradigm for the Personal Cloud. 2017]. Dédié au contexte du Cloud personnel, ce modèle se veut à la fois intuitif et suffisamment puissant pour créer de nouveaux usages autour du partage des données personnelles.

Ce modèle s'intègre nativement dans l'architecture présentée au chapitre précédent, la combinaison des deux formant l'essence de SWYSWYK. Cependant, il en reste indépendant et peut être implémenté sans considérations architecturales.

Nous présentons tout d'abord les bases du modèle et sa sémantique, puis nous discutons des aspects opérationnels. La validation du modèle est ensuite présentée via une évaluation faite sur la plateforme expérimentale, ainsi que par un démonstrateur destiné à présenter l'intérêt concret du modèle de partage, qui sera publié pour la conférence EDBT' 18 [Tran-Van, Reconciling Privacy and Data Sharing in a Smart and Connected Surrounding 2018].

4.1 Introduction

Nous avons présenté au chapitre précédent la partie architecturale de SWYSWYK. La combinaison des environnements d'exécution distincts et les outils d'administration spécifiques permettent une sécurité par construction pour le Cloud personnel. Elle reste agnostique vis-à-vis de la politique de contrôle d'accès utilisée, du moment que le modèle respecte les prérequis détaillés dans la section 3.2, et en particulier la matérialisation des permissions sous forme d'ACL représentée en triplets < *s, d, a* >.

Or, disposer d'un modèle dédié au Cloud personnel capable de tirer pleinement parti de ses caractéristiques est essentiel pour l'adoption de SWYSWYK. En effet, de nombreux exemples nous prouvent que la sécurité seule ne suffit pas [Schneier 2000] et doit le moins possible nuire aux usages [Kainda 2010]. Dans le monde du Web actuel, où tout un chacun partage quotidiennement des documents, il est essentiel que des fonctionnalités de partage à la fois poussées et intuitives soient utilisables par les utilisateurs.

Comme vu au Chapitre 2, de nombreuses approches peuvent être envisagées autour du partage de données. Cependant, aucun modèle actuel n'émerge dans le contexte du Cloud personnel ; la plupart étant soit extrêmement manuels et limités [Tootoonchian 2009], soit difficiles à contrôler du point de vue de la dissémination des données [Cutillo 2009] [Chard 2012] [Van Kleek 2012], soit opaques et difficilement compréhensibles pour des utilisateurs non avertis [Fang 2010] [Squicciarini 2011]. Certains travaux introduisent des sémantiques de règles pour pouvoir exprimer des partages transverses sur les données ; cependant, leurs cas d'usages sont soit différents du nôtre, par exemple [Riva 2011] permet d'exprimer des stratégies de réplication entre les appareils de l'individu, selon différents éléments de contexte, tandis que [Mazurek 2014] s'intègre dans une architecture *serverless* où les données sont distribuées sur les différents appareils des individus. Quant à [Geambasu 2007], sa sémantique repose exclusivement sur SQL, ce qui réduit considérablement la facilité d'intégration dans des environnements qui tendent de plus en plus vers le NoSQL, par exemple Sandstorm, Cozy ou Camlistore.

Comme expliqué en 2.4, le Cloud personnel est **volatile et versatile**. Un modèle de partage dédié doit être capable de capter les évolutions des données, quelle que soit leur structure. De même, le modèle doit être le plus simple et intuitif possible, afin que l'utilisateur ait le moins de tâches administratives rébarbatives à effectuer. Typiquement, la gestion des sujets peut être automatisée, alors que c'est une tâche qui est généralement très manuelle et répétée pour chaque système qui opère en silo vis-à-vis des autres.

Au chapitre précédent, nous avons vus comment l'architecture présentée permet d'appliquer deux principes majeurs de SWYSWYK, à savoir l'***empowerment* éclairé** et la **sécurité par construction.** Afin d'intégrer à SWYSWYK un modèle de partage respectueux de la vie privée et dédié au Cloud personnel, ces deux propriétés doivent être également être respectées ici. De plus, le modèle permet d'appliquer le troisième principe édicté en 2.4, à

savoir, **l'auto-administration** qui comme nous le verrons dans ce chapitre, tire parti de la généricité du Cloud personnel pour administrer efficacement les sujets.

Comme de nombreuses propositions existantes, SWYSWYK est un modèle qui permet d'exprimer des partages par des règles. Cependant, notre but n'est pas de proposer un nouveau modèle de contrôle d'accès avec plus d'expressivité. L'originalité de SWYSWYK réside dans quelques principes essentiels qui peuvent être résumés ainsi : **les documents sont des règles ; les sujets et les objets sont des documents.** Nous expliquons dans la section suivante ces principes.

4.2 Bases du modèle

Nous illustrons tout d'abord l'impact de **l'*empowerment* éclairé** dans la déclaration du contrôle d'accès.

Partager une photo avec les personnes qui apparaissent dessus, partager un compte-rendu de réunion avec les participants ou une entrée d'agenda avec les personnes invitées, sont autant d'actions simples à concevoir mais difficiles voire impossibles à exprimer avec les modèles actuels.

Pour chaque cas sus-cité, des permissions peuvent être induites du contenu du document à partager. Dit autrement, certains documents font apparaître naturellement des sujets qui sont implicitement légitimes pour être des cibles d'une action de partage. Ceci nous amène au premier principe fondamental de notre modèle : **les documents sont des règles.**

Afin de pouvoir exprimer ces permissions, les sujets directement concernés par ces documents, également appelés *identifiees* [Park 2004], doivent pouvoir être considérés dans la définition de la règle. Nous appelons *reflexives rules* ou règles réflexives, les règles qui se fondent sur ce principe. Un corollaire est que chaque sujet doit correspondre à un document du Cloud personnel, par exemple, une fiche contact, un CV, une carte d'identité, etc. Cela permet d'éviter d'imposer une structure unique aux sujets qui sera inévitablement limitée et non compatible avec différents types d'usages et plateformes. Cela donne corps à un autre principe clé du modèle : **les sujets sont des documents.**

De même, ce qui est partagé doit toujours correspondre à des documents visualisables, comme cela a été vu au Chapitre 3. Même si l'objet du partage est le résultat d'un calcul complexe sur un ensemble de documents, par exemple une série temporelle de consommation électrique, ce résultat doit pouvoir s'exprimer sous la forme d'un document intelligible pour le propriétaire. Cela correspond ici au dernier principe du modèle : **les objets sont des documents**.

La combinaison de ces principes clés, les documents sont des règles, les sujets et les objets sont des documents, donne corps au paradigme du « *Share What You can See with Whom You Know »,* du point de vue du modèle de partage.

L'impact de **l'auto-administration** est également essentiel car il permet l'automatisation de la déclaration et la maintenance des sujets. Par exemple, le partage d'un compte-rendu avec les participants d'une réunion pourrait provoquer la création ou mise à jour automatique des sujets concernés, sans aucun besoin d'intervention de la part du propriétaire.

Nous montrons dans la section suivante comment ces principes peuvent être intégrés dans un modèle de partage simple, en laissant ouverte la discussion sur leur intégration dans des modèles plus expressifs ou traditionnels.

Ainsi, nous faisons un ensemble d'hypothèses simplificatrices :

1. Notre modèle repose sur une politique dite fermée : toute action non explicitement accordée est refusée.

2. Les actions acceptées sont de type *Create, Read, Update, Delete* (CRUD), sur les documents du Cloud personnel.

3. Le modèle ne supporte que des autorisations, c'est-à-dire des règles positives.

4. Le propriétaire peut éditer à sa guise les ACL produites, représentant les permissions, afin de pouvoir gérer des exceptions dans les règles.

De fait, il y a une correspondance directe entre les règles de partage et les ensembles d'ACL. Comme vu au Chapitre 3, une action *a* est accordée au sujet *s* sur le document d si et seulement si $(s,d,a) \in ACL$.

Notre modèle rentre donc bien dans les critères édictés au chapitre précédent, le rendant *cohérent* (la décision est unique), *complet* (la décision existe toujours) et peut être évalué dans un *temps logarithmique*.

4.3 Sémantique du modèle de partage

Nous posons ici les notations relatives au modèle et donnons la définition des deux types de règles de partage SWYSWYK ; les règles réflexives et basiques.

4.3.1 Notations

Ensembles

D : ensemble de tous les documents du Cloud personnel. Par extension, ensemble de tous les *DocId,* identifiants uniques de chaque document.

S : ensemble de tous les sujets du Cloud personnel. Par extension, ensemble de tous les *SubjectId,* identifiants uniques de chaque sujet.

A : ensemble des actions pouvant être appliquées sur les éléments de *D*.

Q : ensemble des qualifications pouvant être exprimées sur les éléments de *D*, avec le langage propre à la plateforme de Cloud personnel. Nous ne faisons aucune hypothèse sur la nature de ce langage.

IT : ensemble de traits d'identification associés de façon unique à chaque élément de *S*. Par exemple, <prénom, nom de famille>, pseudo, numéro de téléphone, *DocId* d'une fiche contact, etc.

Relations et fonctions

Nous considérons les relations et fonctions suivantes :

$ACL \subseteq S \times D \times A$: ensemble des listes de contrôle d'accès.

Allowed : $S \times D \times A \rightarrow \{true, false\}$: caractérise le contrôle d'accès.

Allowed (s, d, a) = *true* ssi $(< s, d, a > \in ACL)$

= *false* autrement

SI : $S \rightarrow IT$: donne tous les traits d'identification connus pour n'importe quel élément de S.

MatchS : $\mathcal{P}(IT) \times \mathcal{P}(IT) \rightarrow \{true, false\}$ où $\mathcal{P}$ est l'ensemble des parties d'un ensemble. *MatchS* évalue la comparaison par paires entre les sujets, en comparant les *IT*.

MatchS$(ident1, ident2)$ = *true* ssi $(ident1 \cap ident2 \neq \varnothing)$

= *false* autrement

IsS : $D \rightarrow S \cup \Phi$: donne l'unique élément de *S* caractérisé par un document ou Φ (absence de valeur) si le document ne correspond à aucun sujet. *IsS* est surjectif : $\forall s \in S, \exists d \in D$ / *IsS*$(d) = s$, ce qui signifie que chaque sujet est représenté par un document (visualisable) existant dans le Cloud personnel.

Fonctions utilisateurs

Deux autres fonctions sont indépendantes de la logique du contrôle d'accès et peuvent être fournies par exemple par des développeurs tiers faisant partie de la communauté du Cloud personnel ou des fournisseurs de service. Pour des besoins de généricité, ces fonctions sont considérées comme des *user-defined functions* (UDF) dans le modèle.

DI : $D \rightarrow \mathcal{P}(IT) \cup \Phi$: donne les traits d'identification, potentiellement de plusieurs sujets, contenus dans un document, ou Φ. Par exemple, un *DI* pourrait être une fonction de reconnaissance faciale sur des photos ou une fonction qui extrait des traits d'identification depuis un document texte.

Filter : $D \times Q \rightarrow \{true, false\}$: évalue si $d \in D$ satisfait une qualification *Q*. Les *Filters* sont utilisés pour former des sous-ensembles de documents qui satisfont le ou les critères donnés. Ils sont dépendants de la plateforme de Cloud personnel et supposés sélectionner

des documents soit sur la base de leur contenu, soit sur leurs métadonnées, par exemple le type, la date, l'auteur, les tags, etc.

4.3.2 Règles de partage réflexives

Dans notre modèle, les règles de partage dites réflexives, ou *reflexive rules*, sont à la base de l'application du principe **les documents sont des règles** et doivent donc être considérées comme des *first-class citizen*, c'est-à dire au cœur du modèle. Ces règles expriment le partage de documents avec des sujets directement concernés par ces documents.

Grâce aux notations présentées ci-dessus, les règles de partage réflexives peuvent être exprimées de la façon suivante :

$$\text{ACL} \leftarrow \{ (s,d,a) \in S \times D \times A \,/\, Filter_1(d,Q) \wedge MatchS(DI(d), SI(s)) \}$$

Nous donnons ici quelques exemples de règles réflexives, avec à chaque fois l'expression des *Q* et *DI* en pseudo-langage.

Exemple 1. *Partager des compte-rendus de réunions avec leurs participants* :

Q : docPath = 'Files/Meetings/minutes-.doc'*

DI : fonction qui extrait des noms des participants depuis le compte-rendu.

Exemple 2. *Partager l'album photo 'Anniversaire 2017' avec les personnes qui apparaissent sur ces photos :*

Q : docType = 'album photo' ∧ tagGallery = 'Anniversaire 2017'

DI : fonction de reconnaissance faciale prenant en entrée les photos de l'album.

Le concepteur d'une règle réflexive pourrait vouloir appliquer des restrictions supplémentaires sur les sujets identifiés dans les documents. Afin de supporter cela, nous introduisons une définition plus complète des règles réflexives :

$$\text{ACL} \leftarrow \{(s,d,a) \in S \times D \times A \,/\, \text{Filter}_1(d,Q) \wedge \text{MatchS}(DI(d), SI(s))$$
$$\wedge \exists d' \in D,\ \text{Filter}_2(d',Q') \wedge (\text{IsS}(d')=s) \}$$

Exemple 3. *Partager le compte-rendu d'une réunion qui a eu lieu à Paris le 6 juin 2017 avec les participants qui font partie de mon équipe :*

Q : docPath = /Files/Meetings/minutes-Paris-060617.doc

Q' : docType = 'contact' $\wedge$ *tagStatus = 'Team member'*

DI : fonction qui extrait les noms des participants depuis le compte-rendu.

4.3.3 Règles de partage basiques

Le modèle doit également pouvoir supporter des règles plus classiques, qu'on appellera *basic rules* ou règles basiques. On désigne de la sorte les règles qui permettent de partager une sélection de documents ($Filter_2$) avec une sélection de sujets ($Filter_1$). Elles sont exprimées de la façon suivante :

$$ACL \leftarrow \{(s,d,a) \in S \times D \times A \,/\, \exists d' \in D,\ Filter_2(d',Q') \wedge IsS(d')=s \wedge Filter_1(d,Q)\}$$

Nous donnons ici quelques exemples de règles basiques, avec à chaque fois l'expression des *Q* et *Q'* en pseudo-langage.

Exemple 4. *Partager le document qui contient les enregistrements de mon rythme cardiaque provenant de mon bracelet connecté, avec mes contacts médecins :*

Q : docType = hearbeat

Q' : docType = 'contact' $\wedge$ *job = 'medecin'*

Exemple 5. *Partager l'album photo 'Peintures psychédéliques' avec mes amis :*

Q : docType = 'album photo' $\wedge$ *tagGallery = 'Peintures psychédéliques'*

Q' : docType = 'contact' $\wedge$ *relationship = 'Friend'*

Exemple 6. *Partager mon suivi calorique des 3 derniers mois avec mon nutritionniste* :

Q: tagContent = 'calorie' $\wedge$ *Date* $\geq$ *'CurrentDate* $-$ *3 month'*

Q': docType = 'vcf' $\wedge$ *job = 'Nutritioniste'*

4.4 Modèle d'administration

L'administration des sujets est une tâche bien souvent perçue comme pénible, chronophage, voire incompréhensible par leurs utilisateurs. La déclaration des sujets et leur maintenance

doit donc être le plus automatisée possible pour garantir une adoption sans effort du modèle. Cela ne doit cependant pas nuire au respect de la vie privée, le propriétaire devant toujours être capable d'affiner sa propre politique en intervenant à souhait sur les permissions générées qu'il doit pouvoir visualiser.

Nous commençons par présenter la notion de ***rule consistency*** qui concrétise le fait que la production de chaque règle de partage peut être visualisée. Puis, nous montrons comment un système d'exceptions permet de supporter la politique de vie privée spécifique du propriétaire, sans introduire de complexité. Enfin, nous expliquons comment l'administration des sujets peut être réalisée de façon transparente, comme partie intégrante des règles de partage.

4.4.1 ***Rules consistency***

Une règle SWYSWYK est dite bien formée si et seulement si elle ne produit que des ACL impliquant des documents visualisables, partagés avec des sujets indentifiables. De nouvelles notations sont ici introduites, liées à l'administration SWYSWYK.

$DV \subseteq D$: le sous-ensemble des documents visualisables, dans le sens interprétable, par le propriétaire. De façon plus formelle, $DV = \{d \in D \,/\, \exists Viewer(d)\}$, avec *Viewer* une application en laquelle le propriétaire a pleine confiance. Elle pourrait typiquement être certifiée par une autorité de confiance ou une communauté d'utilisateurs. Cette application doit être capable d'afficher une vue interprétable du document au propriétaire, par exemple, transformer un binaire en une image.

$DS \subseteq DV$: ensemble des documents interprétables caractérisant un sujet unique. Par exemple, une fiche contact, un CV, une carte d'identité, etc.

$DI \subseteq D$: ensemble des documents contenant des traits d'identification de sujets (ou *identifiees*). Par exemple une entrée d'agenda, une photo de groupe, un compte-rendu de réunion, etc.

SR : ensemble des règles de partage activées par le propriétaire.

A partir de ces notations, on définit qu'une règle SWYSWYK est bien formée, si et seulement si :

$$\forall sr \in SR,\ \forall acl \in ACL,\ acl.d \in DV \wedge acl.s \in DS$$

N'importe quelle *acl*$\in$ACL qui ne satisfait pas à cette condition sera automatiquement filtrée.

4.4.2 Exceptions

Dans ce chapitre, nous avons acté comme hypothèse que l'on ne considérait qu'une politique fermée, avec uniquement des règles positives, c'est-à-dire qui génèrent des autorisations.

Nous faisons le choix d'exclure les règles négatives, c'est-à-dire qui génèrent des interdictions. En effet, nous pensons que l'utilisateur lambda ne peut pas appréhender aisément une politique mêlant des autorisations et des interdictions, parfois en conflit, ce qui peut devenir un enfer, même pour des administrateurs aguerris [Bertino 1993].

Appliqué à notre contexte, cette complexité introduirait un coût cognitif préjudiciable au principe **d'*empowerment* éclairé** car l'utilisateur ne peut alors plus appréhender les effets concrets du modèle. Enfin, cela casserait également le principe de **sécurité par construction**. En effet, comme discuté au chapitre précédent, la logique du moniteur de référence doit être également compréhensible par le propriétaire. Cela est rendu possible par la simplicité du contrôle d'accès qui se résume à vérifier l'existence d'un sujet, document, action dans l'ensemble des ACL. L'introduction de règles négatives complexifierait de façon répréhensible cette logique et pourrait potentiellement empêcher l'implémentation du moniteur de référence dans du matériel sécurisé, comme ceux discutés dans la section 3.6.

Ainsi, plutôt que d'enrichir la sémantique des règles pour tenter de capturer tous les cas de figures possibles, nous introduisons le principe d'**exceptions** qui permettent au propriétaire de filtrer certaines permissions qui lui sembleraient inadéquates avec sa politique de partage.

Afin de supporter des règles à base de prédicats pouvant souffrir d'exceptions, mais sans introduire de négation, nous utilisons l'interface d'administration, l'*Administration GUI*, présentée avec l'architecture qui permet au propriétaire d'interagir avec l'ensemble des ACL considérées comme suspectes et filtrer les indésirables, considérées alors comme des exceptions et stockées dans ACL^{-}.

Pour rappel, seul l'ensemble ACL^+ est pris en compte lors de l'évaluation du contrôle d'accès par la fonction *Allowed*. La matérialisation de l'ensemble des ACL^- permet à la fois au propriétaire de visualiser les exceptions qu'il a établies et à la fonction *Resolve* de l'utiliser dans le calcul de distance. A noter que le propriétaire peut à tout moment annuler une exception pour la faire passer en ACL^+, la rendant ainsi active pour le contrôle d'accès.

4.4.3 Administration des sujets

La versatilité des sujets rend leur administration ardue, et nécessite des mises à jour régulières, des ajouts, des suppressions, etc. L'objectif ici est de rendre cette tâche la plus transparente possible en extrayant la définition des sujets directement depuis les documents du Cloud personnel, et ce via les règles de partage.

Tout d'abord, nous détaillons la structure de *S* qui peut être représentée sous forme de table, avec le schéma suivant :

Sid: S, Did: ℘(DS), It: ℘(IT), [Ct: CONTACT], [Auth: {(AUTH, credential)}]

Sid est l'identifiant unique du sujet. *Did* est l'ensemble des identifiants qui référencent des documents représentant de façon unique ce sujet. *It* est l'ensemble des traits d'identification du sujet. *Ct* et *Auth* sont optionnels et dépendant de la plateforme de Cloud personnel. Ils permettent respectivement de notifier et authentifier le sujet. Typiquement, la notification peut se faire par e-mail qui est le moyen interopérable le plus répandu pour contacter des sujets et l'authentification peut être réalisée par un des protocoles décrits dans la section 2.2.1.

La fonction *IsS(d)*, brièvement évoquée dans la section 4.3.1 est utilisée pour enregistrer de nouveaux sujets dans *S*. En réalité, il s'agit d'un effet de bord de la fonction, dont le rôle premier est de retourner le sujet $s \in S$ associé au document $d \in D$. Nous présentons l'algorithme de *IsS* ci-dessous, en Figure 15.

Fonction *IsS*

Entrée: $d \in D$

Sortie: $s \in S$ si d correspond à un sujet reconnu, Φ *sinon*

```
1. if  ∃ s ∈ S / MatchS(DI(d), SI(s)) then
2.    s.It ← s.It ∪ DI(d)

3. else if RegistrationAgreement() then
4.    s ← NewSid()
5.    S ← < s, d, DI(d), [Ct(s)], [Auth(s)] >
6.    DS ← d

7. else s ← Φ

8. return s
```

Figure 15. Fonction *IsS*

IsS est automatiquement appelée (1) chaque fois qu'un document $d \in DS$ est créé ou mis à jour dans le Cloud personnel, par exemple une fiche contact, et (2) chaque fois qu'une règle de partage est créée. Les documents considérés comme faisant partie de l'ensemble *DS* sont à la discrétion de la plateforme de Cloud personnel.

Si au moins un trait d'identification présent dans un document *d* correspond à un sujet *s* existant, *s* est retourné et *s.It* potentiellement enrichi avec les autres traits d'identification présents dans *d.* Si aucune correspondance n'est trouvée, le propriétaire peut être notifié pour accepter l'enregistrement du nouveau sujet à partir du contenu de *d.* Autrement, *IsS* échoue et retourne Φ.

Ainsi, l'ensemble des sujets grossit automatiquement au fil des insertions des documents et des déclarations de règles, en minimisant autant que possible les interactions avec le propriétaire.

Ce dernier pourrait cependant vouloir désambiguïser de temps en temps le contenu de *S* à travers l'interface d'administration. Par exemple, fusionner des situations telles que *'John Doe'∈s.It* et *'john@doe.io'∈s'.It*. C'est un problème fréquemment rencontré dans les applications de contacts et résolu de façon semi-automatique dans la plupart d'entre elles par des heuristiques simples. Il est par exemple possible de calculer des similarités textuelles par la méthodes de Jaccard [Tan 2006] ou la distance de Levenshtein [Navarro 2001]. Ce n'est donc à priori pas un challenge de l'intégrer dans l'*Administrative GUI*.

En complément de cette mécanique, des systèmes plus poussés pourraient être utilisés pour créer et mettre à jour automatiquement les sujets, se reposant par exemple sur l'ontologie FOAF et l'utilisation de profils publics ou privés. Dans ce cas de figure, chaque sujet est

responsable de la création et la maintenance de ses propres attributs d'identification depuis sa page personnelle. Chaque Cloud personnel peut alors automatiquement requêter cette page pour récupérer les attributs accessibles. L'intégration d'un tel système, présenté dans [Van Kleek 2012] est remis à des travaux futurs.

4.5 Sémantique opérationnelle

Nous discutons ici de la sémantique opérationnelle de SWYSWYK, en présentant la mécanique générale du modèle, depuis la création des règles de partage à l'évaluation du contrôle d'accès. Nous expliquons également comment sont évaluées et maintenues les règles via le séquencement des différents opérateurs sur les deux types de règles du modèle, basiques et réflexives, et leur intégration dans l'architecture.

4.5.1 Bases opérationnelles de SWYSWYK

Dans la section 3.2, nous avons énuméré les différentes étapes de validation des ACL du point de vue architectural. Nous décrivons ici le processus complet de création, maintenance et évaluation d'un ensemble de permissions dans SWYSWYK :

1. Le propriétaire crée des règles de partage, via une interface fournie par la plateforme de Cloud personnel. Parallèlement, il peut définir des *watchdog triggers* pour s'assurer que les données sensibles ne seront pas partagées par inadvertance.

2. Le module *Policy translator* évalue la règle de partage sur l'ensemble des documents du Cloud personnel, et produit un ensemble d'ACL candidates, stockées dans ACL^{*}.

3. La console d'administration exécute les modules *Advisor* et les *watchdog triggers* pour assainir les ACL^{*} en plaçant les ACL suspectes dans $ACL^{?}$ et les autres dans ACL^{+} ou ACL^{-}.

4. Le propriétaire vérifie les ACL produites et accepte ou rejette les ACL suspectes en utilisant l'interface d'administration.

5. Le moniteur de référence authentifie les sujets et évalue les appels *Allowed(s,d,a)*. Si *true* est retourné, le document *d* est déchiffré pour *s*.

6. De nouveaux sujets sont automatiquement ajoutés lorsque des documents nouvellement insérés sont évalués par le *Policy translator* et des traits d'identifications détectés.

Toutes ces étapes forment le contrôle d'accès SWYSWYK. Les étapes 3, 4 et 6 sont inhabituelles dans la gestion du contrôle d'accès. L'étape 6 contribue à la propriété **d'auto-administration** tandis que l'étape 4 donne corps à la propriété ***d'empowerment* éclairé**, en poussant le propriétaire à vérifier les effets de ses règles de partage et en particulier les permissions suspectes. Contrairement aux moniteurs de référence dits *rule-based*, comme ceux présentés en 2.1.4, l'évaluation du contrôle d'accès se fait sur un ensemble matérialisé de permissions, ACL^{+}, produites en amont des demandes d'accès. Ce qui permet au propriétaire de surveiller à tout moment les effets de sa politique de partage, alors même qu'elle découle de règles de partage qui sont traditionnellement complexes à appréhender, puisqu'évaluées à la volée pour chaque demande d'accès.

4.5.2 Construction et gestion des règles de partage

Ainsi, SWYSWYK se fonde sur la matérialisation de toutes les ACL, ce qui garantit une évaluation triviale de la fonction *Allowed* et permet au propriétaire de visualiser et filtrer les effets de ses règles de partage. Nous détaillons ici comment les règles réflexives (RR) et les règles basiques (BR) sont créées et maintenues.

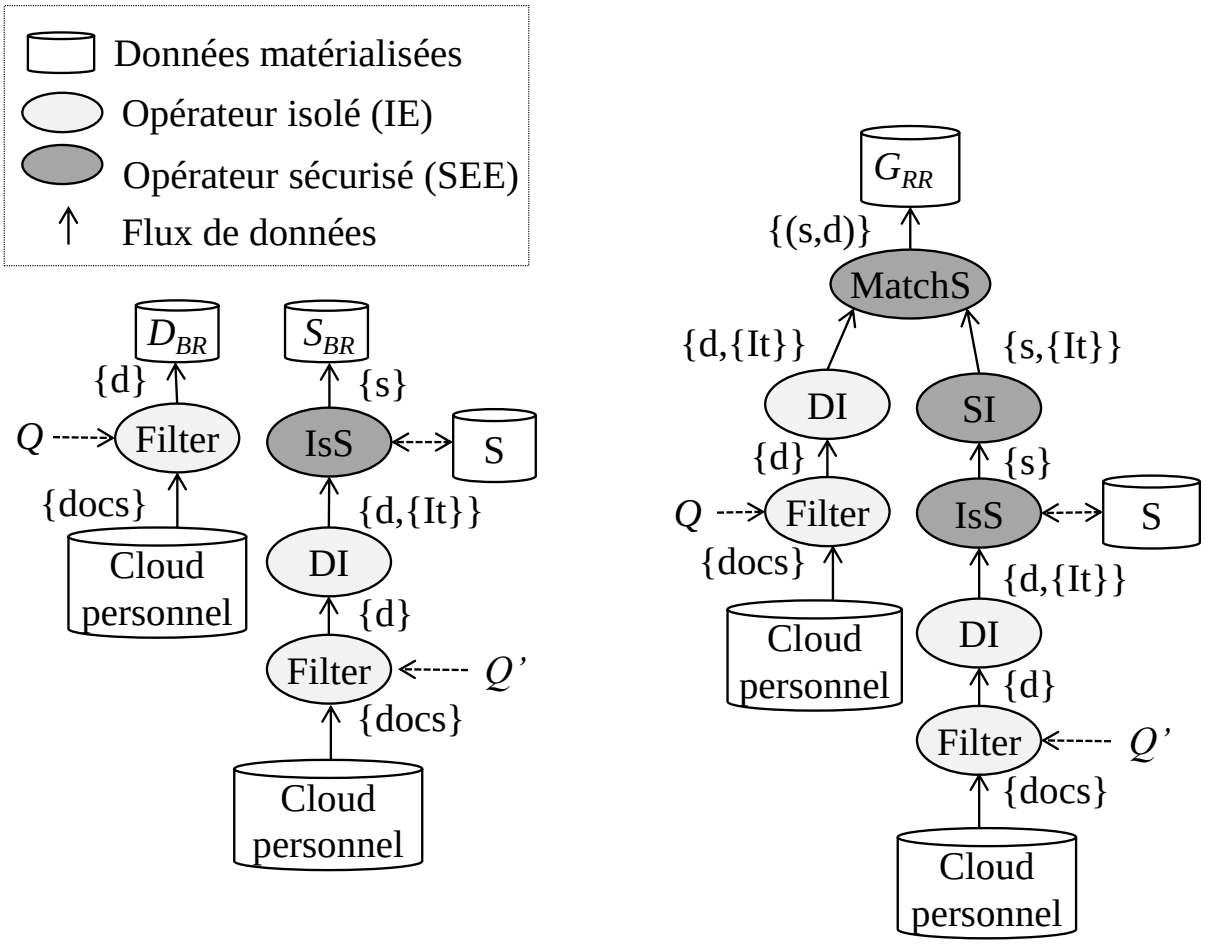
Cinq opérateurs sont nécessaires pour exprimer n'importe quelle règle basique ou réflexive, à savoir *Filter, DI, SI, IsS et MatchS*. La sémantique de ces opérateurs est équivalente à leurs homologues fonctionnels présentés en section 4.3 et n'est donc pas rappelée ici. La principale particularité réside dans le fait que chaque opérateur implémente la fonction correspondante de façon ensembliste. Par exemple, l'opérateur *Filter* s'applique à tous les documents de *D* et retourne le sous-ensemble des documents satisfaisant la condition Q. La séquence de données consommées et produite par les opérateurs est présentée en Figure 16, pour la traduction de règles basiques et réflexives en ACL.

Lors de la création d'une règle, le *Policy translator* doit l'appliquer sur tous les documents du Cloud personnel. Pour une règle basique, un opérateur *Filter* est appliqué aux feuilles de chaque branche de l'arbre d'évaluation, ce qui correspond à une sélection de sujets sur la branche de droite et une sélection de documents sur la branche de gauche. Puis, l'opérateur *DI* extrait la liste des *IT* depuis les documents des sujets ciblés et les transmet à *IsS* qui tente de faire correspondre ces *IT* à ceux des sujets déjà enregistrés dans *S.* Comme dit en 4.4.3, l'opérateur *IsS* peut avoir pour effet de bord la mise à jour dynamique de *S* quand des sujets inconnus sont rencontrés. Enfin, pour une règle basique BR, la branche de droite alimente la structure S_{BR} tandis que la branche de gauche alimente la structure D_{BR}, enregistrant ainsi les ACL (candidates) produites.

L'arbre d'opérateur des règles réflexives suit la même logique, à l'exception du fait que les branches de gauche et de droite doivent être jointes sur les *IT* des sujets et produire des ACL dans la structure G_{RR}. Cette jointure est réalisée par l'opérateur *MatchS* qui prend en entrée les *IT* retournés par les opérateurs *DI*, pour les documents, et *SI* pour les sujets.

A chaque fois qu'un document *d* est supprimé du Cloud personnel, toute entrée qui le référence dans S_{BR}, D_{BR} et G_{RR} doit l'être également. A l'inverse, pour tout nouveau document *d* inséré dans le Cloud personnel, les *Filters* de chaque branche de l'ensemble des règles sont réévalués sur *d* afin de vérifier si de nouvelles ACL candidates peuvent être produites.

IsS, MatchS et *SI*, coloriés en gris foncé sur la Figure 16, sont des opérateurs sécurisés qui s'exécutent directement dans le *SEE* de l'architecture. A l'inverse, les *Filters* et *DI,* coloriés en gris clair, sont des UDF, dans lesquels la confiance est relative car pouvant provenir de sources variées. Ils sont donc exécutés en isolation par le *Policy Translator* pour prévenir toute fuite de données provoquée par un code malveillant ou mal conçu.



(a) Règle basique (b) Règle réflexive

Figure 16. Production des ACL

4.6 Plateforme expérimentale

Afin d'évaluer l'intégration du modèle dans l'architecture SWYSWYK, nous utilisons la même plateforme expérimentale que celle décrite en section 3.6.1.

4.6.1 Règles et jeux de données

Nous indiquons dans le Tableau 1 les règles et jeux de données utilisés dans nos expérimentations pour mesurer les performances au moment de la création d'une règle. Nous définissons quatre règles distinctes représentatives des règles basiques et réflexives,

chacune ayant une déclinaison *Big* et *Small* qui correspondent à des tailles différentes en termes de production d'ACL. Par exemple, *Big BR*, la grande règle basique, qualifie 1 000 documents sur le prédicat *type='health community',* et produit 10 000 ACL.

Règle	***Filtre sur D***	***Filtre sur S***	***Nombre de D, S, ACL***
Small BR	type = 'directory' && name = 'team'	type = 'contact' && group = 'team'	10, 5, 50
Big BR	type = 'cardio'	type = 'health community'	1000, 10, 10000
Small RR	type = 'note'	type = 'contact' && group = 'lab'	10, 30, 50
Big RR	type = 'album' && tag = 'holidays'	type = 'contact' && group = 'friends'	1000, 200, 5000

Tableau 1. Règles de partage utilisées pour l'évaluation

4.6.2 Evaluation des règles

Dans Cozy, les documents sont formatés en JSON[51] avec un ensemble de clé-valeurs et potentiellement associés à un ou plusieurs fichiers binaires.

Une implémentation simple de *Filter* et *DI* consiste à vérifier un ensemble de conditions qui portent sur les attributs JSON des documents. Des implémentations plus complexes peuvent se faire sur des fichiers binaires, par exemple un algorithme de reconnaissance faciale. Nous discutons de ce type de *DI* dans le démonstrateur présenté en 4.7 et utilisons l'implémentation simple pour cette évaluation.

Pour rappel, l'architecture SWYSWYK, illustrée en Figure 7, nécessite la matérialisation d'un certain nombre de données, comme les différents ensembles d'ACL. Comme discuté dans la section précédente et illustré par la Figure 16, les permissions sont ici représentées par les structures S_{BR} et D_{BR} pour les règles basiques et G_{RR} pour les règles réflexives. Nous

[51] https://www.json.org/json-fr.html

implémentons ces structures sous la forme de tables relationnelles dans le moteur MiloDB [Anciaux 2014], embarqué dans PlugDB.

Plus précisément, les tables *BRD* et *BRS* matérialisent l'union de S_{BR} et D_{BR} pour toutes les règles basiques BR, avec *Rid* l'identifiant unique d'une règle basique donnant l'autorisation *A* sur le document *Did* au sujet *Sid*. Les tables *RR* et *Except* matérialisent respectivement G_{RR} pour les règles réflexives et G_{Except} pour les exceptions qui correspondent à ACL^{-}.

L'ensemble $ACL^{?}$ est représenté par trois tables $BRD^{?}$, $BRS^{?}$ et $RR^{?}$, avec les mêmes schémas respectifs que *BRD, BRS* et *RR*.

Enfin, les sujets et leurs traits d'identification sont stockés dans la table *SIT.*

Les opérateurs *SI, IsS* et *MatchS*, ainsi que les fonctions *Allowed* et *Who, What, Which* sont implémentés sous la forme de simples requêtes SQL sur ces tables.

4.6.3 Initialisation

Lors de leur création, chaque règle de partage doit être évaluée sur l'ensemble des documents du Cloud personnel. Cela génère des coûts d'environnement pour chaque document, en plus du temps pour évaluer les *Filters* de la règle et les autres opérateurs de l'arbre d'évaluation si le document est qualifié. Ce coût d'environnement a déjà été détaillé au chapitre précédent via la Figure 10, nous ne revenons donc pas dessus.

La Figure 17 montre le temps pris pour exécuter le processus d'initialisation pour chaque règle du Tableau 1, en fonction du nombre de documents à considérer. Les conclusions sont les suivantes : (i) bien que considérable, le coût d'initialisation d'une règle reste acceptable en considérant que c'est une tâche asynchrone, (ii) le coût environnemental représente la moitié du coût total et est principalement dû aux communications entre l'*UE*, *IE* et le *SEE* qui pourraient être grandement réduites avec une autre instance architecturale, comme discuté dans 3.6.23.6.3 et (iii) mis à part les coûts environnementaux, le coût d'évaluation des *Filters* est prédominant à cause du nombre d'itérations (1 évaluation par document et par règle) mais pas à cause d'un filtre en tant que tel qui ne prend que quelques millisecondes individuellement. Un filtre plus élaboré, typiquement d'analyse d'image, provoquerait un surcoût plus élevé mais pourrait être atténué en pré-filtrant les documents sur leurs

métadonnées ; par exemple dans un partage d'album avec les personnes apparaissant dessus, seules les photos faisant effectivement partie de l'album seraient analysées.

4.6.4 Maintenance

Comme vu en 4.5.2, chaque fois qu'un document *d* est inséré, les *Filters* de toutes les règles sont appliqués dessus afin de vérifier si de nouvelles ACL peuvent être produites. Cela n'est pas un problème pour des règles basiques mais peut mener à une réévaluation complète sur tous les documents du Cloud personnel par l'opérateur *MatchS* d'une règle réflexive si les associations entre les documents et les sujets ne sont pas maintenues. Par exemple, un sujet *s* inséré au temps t_2 pourrait satisfaire une règle concernant le document *d* inséré au temps $t_1 < t_2$, alors que cette association n'existait pas au temps t_1. Pour répondre à ce problème, nous ajoutons des structures additionnelles qui enregistrent les associations réflexives en attente, entre les sujets et les documents :

RD (Rid int, Did char(32), IT varchar)

HR (Rid int, Sid int)

RD sert à enregistrer les documents rentrés dans une règle réflexive avec les traits d'identifications qui ne correspondent pas à un sujet existant au moment présent. De même, *HR* enregistre les identifiants des sujets qualifiés par cette règle réflexive, dans le but de détecter une association avec un futur document qui référencerait un de ces sujets. Grâce à ces structures, il est possible de maintenir efficacement une règle du type « Partager tous les compte-rendus de réunion du projet X avec toutes les personnes qui apparaissent sur l'un d'entre eux ». A noter que le *Did*, l'identifiant unique du document, est codé sur 32 octets car cela correspond au format par défaut des *UUID* de CouchDB[52].

A partir de l'allure de l'arbre d'opérateur de la Figure 16, nous pouvons en déduire que le coût de maintenance est déterminé par (i) l'évaluation des *Filters* pour toutes les règles actives et (ii) le coût de *IsS* quand un document est qualifié par un *Filter*. En effet, le coût de *IsS* dépend de la cardinalité de *S* et du nombre d'*IT* que *S* maintient. La Figure 18 indique le nombre de sujets et d'*IT* par sujets qu'il est possible de maintenir, tout en maintenant

[52] https://wiki.apache.org/couchdb/HttpGetUuids

l'insertion d'un document en moins d'une seconde, pour un nombre donné de règles. Pour faire varier le nombre de règles actives jusqu'à 100 nous en avons généré ayant les mêmes caractéristiques que celles indiquées dans le Tableau 1. A noter que le coût de maintenance des règles *BR* reste toujours inférieur à 2.5 ms par document, et n'est pas reporté ici, car il est indépendant du nombre de sujets.

Au vu des résultats montrés par la Figure 18, il n'y a pas de problème de performance lié à la maintenance des ACL. Avec 100 règles de type RR et sans utilisation d'index, 1000 sujets avec 7 *IT* chacun peuvent être gérés avec un coût de maintenance inférieur à 1 seconde par document inséré.

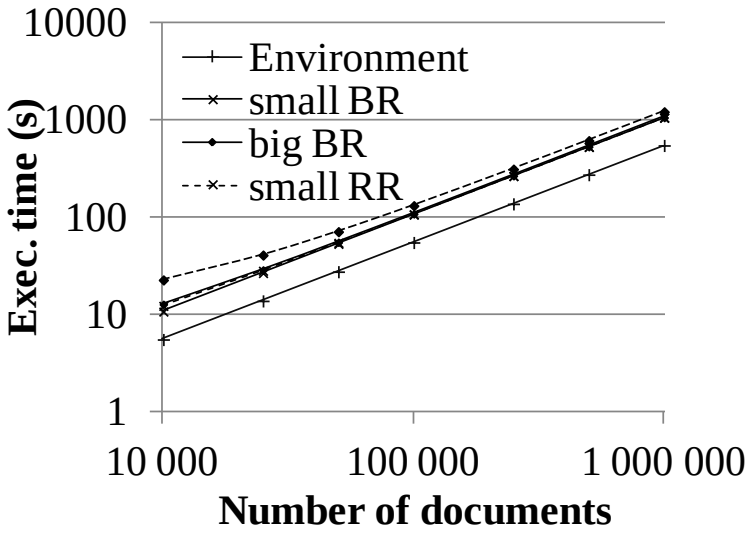


Figure 17. Coûts d'initialisation d'une règle

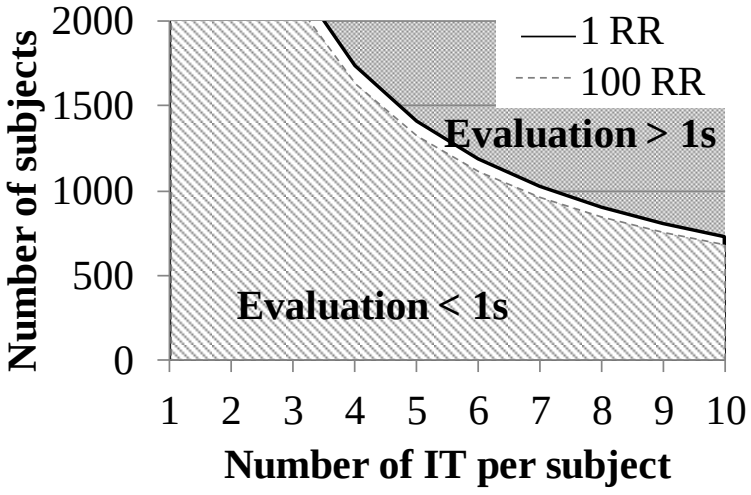


Figure 18. Coûts de maintenance d'une règle en fonction de la cardinalité des sujets

Ainsi, cette évaluation montre la faisabilité d'intégrer le modèle dans l'architecture SWYSWYK, en implémentant les structures adéquates dans l'environnement sécurisé PlugDB.

Le principal point de friction se situe à l'initialisation des règles, comme montré par la Figure 17 qui implique de déchiffrer et évaluer tous les documents du Cloud personnel dans l'environnement isolé. Ce coût est moins critique que celui relatif au contrôle d'accès lui-même car il s'exécute de façon asynchrone, sans impacter l'utilisation de la plateforme de Cloud personnel. Mais il peut devenir rédhibitoire pour le propriétaire s'il dispose d'une base de données avec des millions de documents et qu'il crée plusieurs règles successives : la production de ses ACL prendrait alors un temps conséquent.

Une solution possible pour réduire ces coûts serait de stocker les métadonnées des documents directement dans le *SEE.* Il suffirait alors de les requêter pour directement obtenir les documents concernés par une règle, sans avoir à les déchiffrer. Un mécanisme d'index inversé tel que celui présenté dans [Lallali 2015] pourrait être utilisé pour indexer efficacement l'ensemble des termes, tout en prenant en compte les spécificités de l'environnement (grande mémoire flash et peu de mémoire vive). Dans ce cas de figure, un langage de requête serait alors fourni à la plateforme de Cloud personnel afin de pouvoir exprimer des *Filters* et *DI* capables de s'exécuter dans le *SEE* sans en compromettre la sécurité. De plus, afin de ne pas compromettre la généricité du modèle, le principe d'UDF pourrait être conservé, spécifiquement pour les règles ne pouvant pas être exécutées en embarqué, typiquement si elles portent sur le contenu même du document. Ces règles seraient alors toujours exécutées dans l'*IE*, mais avec un impact grandement réduit (seuls les documents concernés seraient déchiffrés, plutôt que potentiellement toute la base de données).

Selon les cas d'usages et la puissance du *SEE*, il serait même possible de fournir un *framework* de règles capables de s'exécuter dans l'environnement sécurisé. Par exemple, dans le cas de la reconnaissance d'image, des techniques de *transfer learning* comme celles exposées dans [Ng 2015] peuvent être utilisées pour effectuer des prétraitements qui minimiseraient les opérations à effectuer dans le *SEE*.

4.7 Démonstration

Nous présentons ici un cas concret de l'utilisation du modèle SWYSWYK via un démonstrateur implémenté sur le Cloud personnel Cozy et qui sera présenté lors de la conférence EDBT' 18 [Tran-Van, Reconciling Privacy and Data Sharing in a Smart and Connected Surrounding 2018]. L'objectif ici, outre de démontrer la faisabilité technique de notre approche, est de montrer comment les propriétés **d'*empowerment* éclairé** et **d'auto-administration** peuvent être concrètement mises en œuvre.

4.7.1 Problématique

Quand Paul part faire de la randonnée dans les Pyrénées avec ses amis, il est équipé d'une montre connectée pour mesurer ses efforts, il prend des photos avec son Smartphone et il utilise une application mobile pour enregistrer ses déplacements grâce à une puce GPS.

Mais comment Paul peut-il obtenir un aperçu complet des données personnelles générées, et comment peut-il en partager une partie avec ses amis de façon simple et intuitive ?

4.7.2 Plateforme de démonstration

La plateforme de démonstration consiste en un Smartphone Android[53], une montre connectée Withings[54] et une instance Cozy locale, installée sur un ordinateur personnel. De plus, plusieurs instances distantes Cozy, spécialement préparées pour la démonstration, s'exécutent sur un serveur distant.

[53] https://www.android.com/

[54] https://health.nokia.com/fr/fr/go

Les traces GPS et les photos sont synchronisées avec l'instance Cozy locale grâce à l'application mobile Cozy. Les données d'activité de la montre connectée sont récupérées grâce à l'API Withings[55] et stockées dans le Cozy.

Les applications Cozy qui permettent d'afficher et manipuler les documents sont uniquement clientes, c'est-à-dire qu'elles s'exécutent exclusivement en JavaScript dans le navigateur et communiquent avec le serveur Cozy grâce à une API dédiée[56]. Le système de partage fait partie intégrante de la stack Cozy dont la documentation est accessible en ligne[57], ainsi que le code source et les détails d'implémentation[58]. Nous renvoyons au 0 pour une description détaillée du protocole de partage implémenté et mis en production dans Cozy.

4.7.3 Scénario

La démonstration illustre l'intérêt du modèle SWYSWYK dans un environnement entièrement connecté. Le scénario est décomposé en quatre étapes, résumées dans la Figure 19. Le déroulement complet de la démonstration est également visible dans une vidéo disponible en ligne[59].

Etape 1. Recueil des données. Cette étape illustre comment la plateforme Cozy peut être utilisée pour collecter facilement des données produites par des appareils connectés et les utiliser. Une instance Cozy est initialisée avec des documents et progressivement complétée par les données produites par le Smartphone et la montre connectée de Paul. Une application, appelée *Sharowalky,* installée sur le Cozy et développée spécifiquement pour cette démonstration, permet d'afficher les photos prises lors de randonnées, visualiser les données GPS sur une carte, et représenter l'activité physique sous formes de graphes, en fonction des données podométriques recueillies. Lors de la démonstration, les participants

[55] https://developer.health.nokia.com/api

[56] https://cozy.github.io/cozy-client-js

[57] https://cozy.github.io/cozy-stack/sharing

[58] https://cozy.github.io/cozy-stack

[59] http://wanda.inria.fr/demos/videos/swyswyk_model.avi

sont invités à se connecter au Cozy de Paul, considéré comme le propriétaire, ouvrir l'application *Sharowalky*, et parcourir les journées de randonnées pour visualiser les données associées.

Etape 2. Définition du partage. L'application *Sharowalky* propose aux participants de partager les photos d'une randonnée avec les amis de Paul qui apparaissent sur ces photos, par exemple Riad. L'interface permet de visualiser la sémantique de la règle de partage qui est une règle réflexive représentée sous la forme de prédicats logiques sur les métadonnées des photos. L'interface permet de vérifier que Riad obtient bien un droit d'accès sur certaines photos de la randonnée, ce qui est confirmé après s'être connecté sur le Cozy de Riad. Les participants peuvent également partager les traces GPS et les données d'activité très facilement. Cette étape permet de souligner la simplicité d'usage de SWYSWYK, malgré l'apparente complexité des règles de partage.

Etape 3. Administration du partage. L'interface d'administration permet aux participants de visualiser et contrôler les effets de la politique de partage. Toutes les permissions créées sont affichées en tant qu'ACL visualisables, c'est-à-dire des triplets <*sujet, object, action*> où chaque sujet et objet est un document interprétable du Cloud personnel. Dans la démonstration, l'interface met en lumière une permission suspecte que les participants peuvent supprimer ou accepter.

Etape 4. Gestion du dynamisme. Finalement, le présentateur de la démonstration affiche des photos prises avant la démonstration qui montrent des groupes de personnes présentes à la conférence. Il prend alors un *selfie* avec l'un des participants qui le souhaite. La photo est automatiquement stockée dans le Cozy et il crée la fiche contact du participant en utilisant le *selfie* comme photo de profil. Cela déclenche l'enregistrement automatique du participant en tant que sujet et lui attribue les droits sur toutes les photos – présentes et futures - sur lesquels il apparaît, à minima le *selfie.*

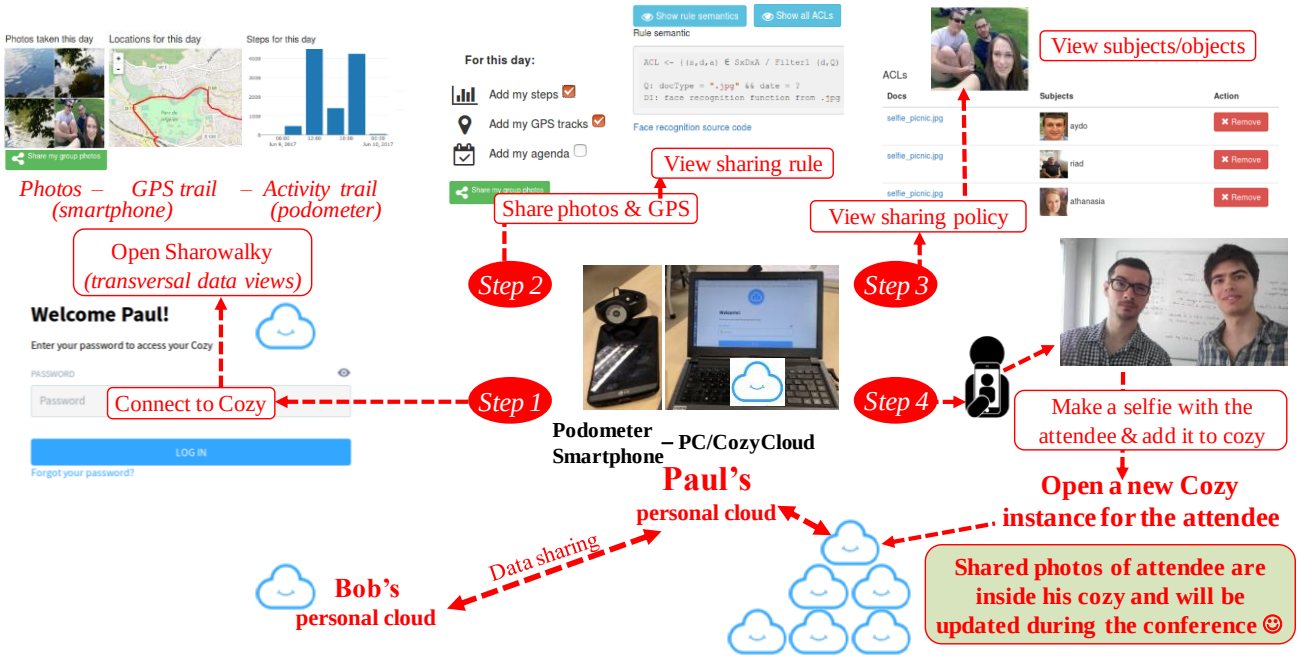


Figure 19. Scénario de démonstration

4.7.4 Bilan de la démonstration

Le modèle de partage SWYSWYK s'intègre particulièrement bien dans un environnement multi-connecté où des données sensibles sont produites à une fréquence importante depuis des objets connectés. Des données de géolocalisation, santé, ou de *quantified-self* sont de plus en plus omniprésentes et généralement dispersées dans des services cloisonnés avec des pratiques opaques quant à la dissémination de ces données, pourtant critiques. Regrouper ces données dans un Cloud personnel permet la création de nouveaux usages transverses et de grande valeur pour les individus. Les données de tous types peuvent alors être exploitées, sans que les propriétaires soient restreints par le fournisseur de l'objet connecté. Ceci, tout en leur garantissant un bien meilleur respect de leur vie privée en évitant à la fois que leurs données se retrouvent sur des serveurs centralisés et en leur permettant de spécifier et contrôler finalement leurs partages.

Ce démonstrateur s'évertue à prouver que SWYSWYK, s'exécutant dans une instance Cozy, répond à ce problème, en permettant le partage de données de nature diverses via la création de règles de partage classiques ou plus élaborées via un algorithme de reconnaissance faciale. Les propriétés d'***empowerment* éclairé** et **d'auto-administration** y sont démontrées, au travers de la facilité d'utilisation, de la visualisation des données, des

sujets et des permissions associées, ainsi que du contrôle exercé par le propriétaire qui peut facilement révoquer des permissions.

4.8 Conclusion

Dans ce chapitre, nous avons présenté un nouveau paradigme de partage pour le Cloud personnel qui s'inscrit dans l'approche SWYSWYK et donc le principe clé est : **les documents sont des règles ; les sujets et les objets sont des documents.**

Les modèles de partage existants sont généralement soit très manuels, nécessitant de définir à la main le partage de chaque document avec chaque sujet, soit à base de règles plus ou moins complexes qui rendent difficile le contrôle du partage. Ils ne sont guère adaptés au contexte du Cloud personnel qui nécessite de pouvoir partager n'importe quel type de document, dans un environnement très dynamique où des mises à jour constantes sont effectuées, provenant de multiples sources (données d'activités d'objets connectés, écritures bancaires, photos de Smartphones, etc). De plus, nous considérons que l'utilisateur doit être capable de parfaitement appréhender sa politique de partage et ses effets, quelle que soit la complexité des règles utilisées pour donner des accès sur ses données, cela afin de leur rendre le contrôle sur la dissémination de leurs documents, dans une optique d'***empowerment***.

Notre modèle est précisément conçu pour supporter la **versatilité** de ce type de plateforme, tout en mettant l'accent sur la simplicité d'utilisation que nous nous sommes évertués de prouver par un démonstrateur. Il permet l'expression de règles de partage intuitives, en particulier réflexives, de visualiser leurs effets et de les ajuster si besoin. Les sujets sont **auto-administrés** via l'évaluation des règles, capable d'inférer des créations et mises à jour de sujets depuis les documents du Cloud personnel. Le fait d'extraire des règles et des sujets depuis le contenu même des documents permet une administration quasi-automatique du partage et de représenter au mieux la politique de dissémination des propriétaires.

Le modèle s'intègre naturellement dans l'architecture présentée au Chapitre 3. Une évaluation des performances a été conduite sur la plateforme expérimentale utilisant PlugDB et Cozy afin d'évaluer l'impact du modèle sur les performances et de valider sa faisabilité. Il

apparaît que le surcoût engendré reste raisonnable, à l'exception de la création des règles qui peuvent prendre un temps conséquent si la base de données contient des millions de documents. Ce coût n'impacte néanmoins pas l'utilisation du Cloud personnel car il est réalisé dans l'environnement isolé de l'architecture. De plus, des techniques d'indexation sur les métadonnées des documents stockés dans le *SEE* pourraient grandement diminuer ces coûts, comme proposé dans [Lallali 2015].

Une implémentation de ce modèle de partage a été intégrée dans la plateforme Cozy et mise en production. La description de cette implémentation, du protocole mis en place et des cas d'usages industriels auxquels elle répond est l'objet du chapitre suivant.

Chapitre 5
Implémentation industrielle

Dans ce chapitre, nous présentons la partie industrielle de la thèse qui consiste en la conception et implémentation d'un protocole de partage décentralisé entre serveurs personnels. Nous décrivons ici comment le protocole a été élaboré et les choix d'implémentation réalisés pour l'intégrer dans la plateforme Cozy. Enfin, nous présentons trois cas d'usages qui concrétisent l'utilisation de ce protocole de partage, mis en production sur deux versions différentes de Cozy.

5.1 Introduction

Les objectifs académiques et industriels sont bien souvent antinomiques et difficiles à concilier. Cependant, il arrive qu'ils se rejoignent et se nourrissent mutuellement. C'est le cas de SWYSWYK dont la partie modèle a été implémentée dans la plateforme éditée par Cozy Cloud et mis en production courant 2017.

Bien que partielle, cette implémentation fait partie intégrante d'un protocole de partage conçu pour répondre à une problématique récurrente dans le Cloud personnel : le partage synchronisé en pair à pair. En effet, comme cela a été vu dans la section 2.3, dans une architecture centralisée où toutes les données résident à un même endroit, physique ou virtuel, gérer le partage peut se résumer à implanter un mécanisme de contrôle d'accès assurant aux seuls destinataires du partage l'accès à la donnée partagée.

Cela n'est pas suffisant dans une architecture décentralisée comme celle du Cloud personnel où chaque nœud peut se trouver sur un serveur physiquement distinct et où les données de chaque utilisateur sont cloisonnées vis-à-vis des autres.

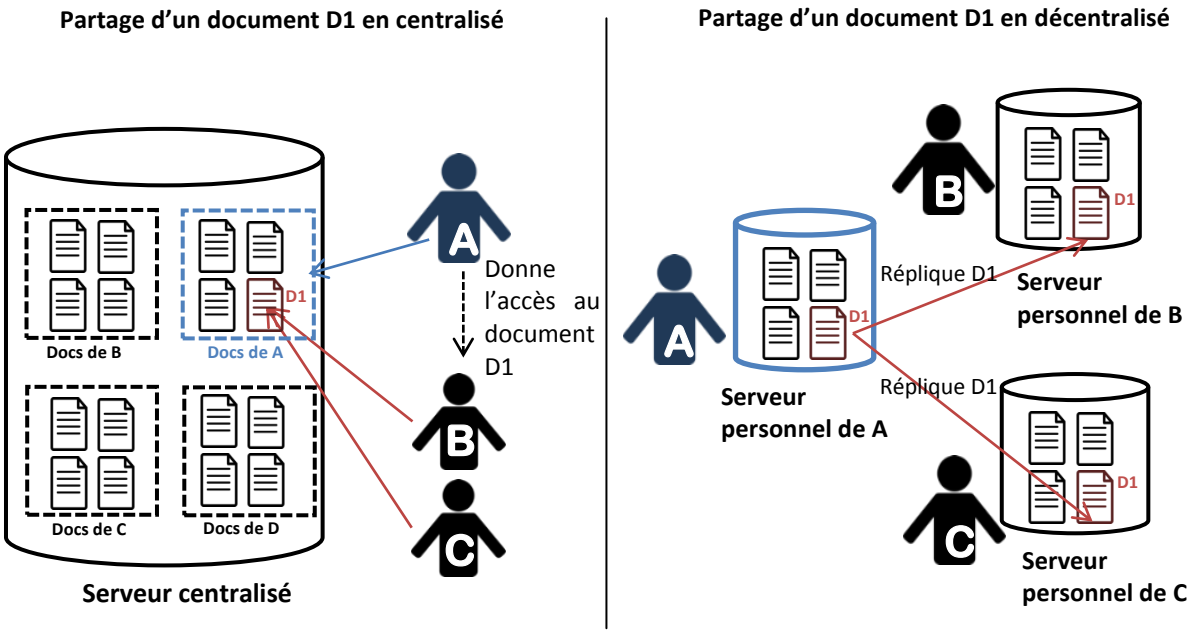


Figure 20. Partage centralisé vs décentralisé

Pour que les destinataires puissent récupérer ce qui leur est partagé et y accéder de façon aussi transparente que dans des plateformes centralisées, une étape supplémentaire doit être effectuée : la réplication des données, comme cela est illustré sur la Figure 20. Lorsqu'elle ne consiste pas en une simple copie, cette réplication doit s'accompagner d'une synchronisation: chaque écriture d'un des tiers du partage doit être propagée aux autres participants.

Nous pouvons rapprocher ce problème au théorème CAP. Enoncé comme une conjecture en 2000 et formalisé en théorème en 2002 [Gilbert 2002], il énonce que dans tout système distribué, il est impossible d'obtenir simultanément les trois propriétés suivantes :

1. **Cohérence** (ou *Consistency*) : garantie que toute donnée accédée sur un nœud correspond à la dernière version écrite sur le réseau.

2. **Disponibilité** (ou *Availability*) *:* garantie que toute requête envoyée à un nœud du système doit retourner une réponse.

3. **Tolérance au partitionnement** (ou *Partition tolerance*) *:* garantie que le réseau peut supporter la perte d'un nombre arbitraire de messages entre les nœuds sans affecter son fonctionnement.

Nous faisons ici le choix de tolérer une cohérence plus lâche : dans le contexte du Cloud personnel, il est acceptable qu'un document partagé ne soit pas dans sa dernière version à un instant *t,* s'il finit par l'être au temps $t + \lambda$. C'est le principe d'*eventual consistency*, implémenté par le moteur CouchDB et utilisé dans Cozy.

Bien que complexe, le problème de la réplication des données en pair à pair, connu de longue date [Son 1988] [Martins. 2007], n'est pas considéré comme un challenge académique ici ; de même que les problématiques liées à l'identité, l'authentification et la résilience du réseau sont adressées en utilisant des technologies existantes et éprouvées qui seront décrites dans ce chapitre.

Dans la suite de ce chapitre, nous détaillons au niveau fonctionnel le protocole conçu pour permettre le partage synchronisé de documents entre instances Cozy. Puis, nous discutons des détails d'implémentation dans le système Cozy et des choix technologiques. Enfin, nous présentons trois cas d'usages concrets qui répondent à des réelles problématiques métier chez Cozy Cloud.

5.2 Protocole de partage

Nous présentons ici le protocole de partage du point de vue fonctionnel, c'est-à-dire sans entrer dans les considérations techniques. Le paradigme SWYSWYK se veut agnostique à la plateforme de Cloud personnel. L'implémentation spécifique du protocole de partage dans Cozy et les choix technologiques qui en découlent sont discutés dans la section suivante. A noter qu'il ne s'agit pas d'un protocole à proprement parler, mais plus d'un ensemble de spécifications qui décrivent les échanges et actions devant avoir lieu pour permettre un partage sécurisé et synchronisé entre différents tiers. Nous utilisons néanmoins ici le terme « protocole » par abus de langage.

Afin de ne pas alourdir ce document, nous ne détaillons pas toutes les étapes du protocole ici, mais les diagrammes de séquences complets sont disponibles en annexes. Nous discutons ici des principes généraux dans un partage entre un émetteur « Alice » et un destinataire « Bob ».

Différents éléments sont volontairement laissés libres pour permettre différentes implémentations coexistant sur différentes plateformes de Cloud personnel. Les choix d'implémentation spécifiques à la plateforme Cozy sont décrits dans la section 5.3.

5.2.1 Création d'un partage

Pour créer un partage, Alice spécifie les documents et les destinataires via une règle de partage, activable par une interface dédiée. Elle indique aussi le type de synchronisation souhaité. Trois modes sont possibles :

- **One-Shot :** partage sans synchronisation. Alice envoie les documents une seule fois à Bob et aucune modification ultérieure ne sera propagée.
- **Master-Slave :** synchronisation unilatérale des documents. Tout nouveau document qui rentre dans la règle de partage ou toute mise à jour d'un document partagé sera propagé chez Bob.
- **Master-Master** : synchronisation bilatérale des documents. Les modifications et ajouts chez Bob sont également propagés chez Alice.

A noter un cas particulier lié au cas *master-master* : si des documents sont déjà partagés dans ce mode, on préfère bloquer un nouveau partage les concernant afin d'éviter un phénomène de propagation transitive entre destinataires. Supposons qu'un même document *D1* soit partagé avec les sujets *S1* et *S2*, respectivement via les partages *P1* et *P2*. Si *S1* modifie *D1*, cela sera propagé chez l'émetteur et synchronisé avec *S2* via le partage *P2*. Ainsi, *S1* et *S2* se synchronisent entre eux, bien qu'ils n'aient pas conscience l'un de l'autre. Cela casse le principe ***d'empowerment* éclairé** de SWYSWYK ; nous préférons donc bloquer cette possibilité.

Le propriétaire peut générer des règles basiques ou réflexives. La section 5.4 présente trois cas d'usages, qui implémentent deux exemples de règles basiques et un cas de règle réflexive.

Une fois le partage créé chez Alice, une demande de partage est envoyée à Bob afin qu'il puisse l'accepter dans son Cloud personnel.

5.2.2 Notification et réponse des destinataires

Afin que Bob soit notifié du partage, la plateforme d'Alice doit générer une demande de partage et la lui envoyer. Cette demande contient une description du partage, ainsi que toutes les permissions requises. En effet, comme les documents sont envoyés dans le Cloud personnel du destinataire, ce dernier doit accorder à minima un droit d'écriture afin de pouvoir les recevoir. Si le destinataire accepte le partage, un échange d'authentifiant se met en place afin que les différentes parties puissent se reconnaître et s'assurer que les requêtes sont légitimes. Dans le cas d'un partage de type *master-master*, la liste de tous les destinataires doit être envoyée dans la demande de partage, vu que chaque destinataire peut impacter et être impacté par les autres acteurs et doit donc être conscient des autres, cela toujours conformément au principe ***d'empowerment* éclairé.**

Le protocole ne fait pas d'hypothèse sur le moyen de notifier le destinataire. Cela peut être fait via l'envoi d'un e-mail, SMS, ou tout moyen de communication spécifique à la plateforme de Cloud personnel utilisée. Dans tous les cas, il faut que le destinataire soit capable de visualiser la description du partage, l'émetteur, et les permissions associées.

Si le destinataire accepte le partage, il doit envoyer une réponse à l'émetteur, ce qui peut se faire en cliquant sur un bouton d'interface, par une réponse à un e-mail, etc. Un échange d'authentifiant doit se mettre en place pour s'assurer que tous les futurs échanges entre Alice et Bob soient bien réalisés par leur plateforme respective. Le protocole d'authentification utilisé est laissé libre. Nous détaillons le choix fait dans notre implémentation en 5.3, qui utilise le système OAuth.

A noter que l'authenticité de la notification et de la réponse doit également être vérifiée, pour s'assurer que la demande de partage provient bien d'Alice et que l'acceptation a bien été faite par Bob. Là encore, la façon de faire n'est pas imposée, car elle dépend du moyen de communication utilisé.

5.2.3 Partage des documents

Une fois la réponse de Bob reçue chez Alice, le Cloud personnel de cette dernière vérifie sa validité. Si le partage est refusé, Bob est supprimé de la liste des destinataires et le partage révoqué s'il en était le seul destinataire. Si le partage est accepté, le Cloud personnel d'Alice récupère les documents concernés par le partage et les envoie à Bob, en utilisant la méthode d'authentification spécifiée, par exemple en incluant un *token* dans chaque message échangé.

Si le partage est de type *one-shot,* il est révoqué chez Alice et Bob une fois les documents partagés, afin de supprimer tous les accès qui n'ont plus lieu d'être. En revanche, s'il s'agit d'un partage *master-**, la plateforme doit être capable de gérer un système d'évènements afin de pouvoir détecter toute nouveauté sur des documents qui entreraient dans le périmètre du partage, afin de les propager.

Nous ne faisons pas d'hypothèse sur le système utilisé pour transmettre et synchroniser les documents entre les pairs. Il peut s'agir d'un protocole sur une couche de transport dédiée, comme XMPP [Saint-Andre 2011], ou via HTTP, par exemple le protocole de réplication CouchDB [CouchDB Replication Protocol 2017]. Quel qu'il soit, le système doit être capable de supporter une **authentification** entre les pairs, le **chiffrement** des documents échangés et leur **synchronisation** : toute nouveauté sur des documents qui entreraient dans le périmètre du partage doit être propagée.

Bien que le système d'authentification soit laissé libre, nous imposons l'utilisation de jetons d'authentification, ou *tokens* : la façon de générer et vérifier ces jetons est laissée libre à la plateforme. Imposer l'utilisation de *token* est suffisamment générique pour permettre l'utilisation de différents protocoles d'authentification. Néanmoins, il est à préciser que le protocole utilisé doit être capable d'associer un *token* à un ensemble de permissions. Ainsi, chaque requête authentifiée doit pouvoir être associée à une action accordée par les

permissions du *token*. Là encore, nous détaillerons les choix inhérents à la plateforme Cozy dans la section suivante.

5.2.4 Révocation

La révocation implique la suppression des accès pour un partage donné. Plus aucune action précédemment accordée dans le contexte du partage n'est alors possible.

Trois scénarios de révocation sont supportés :

1. Révocation d'un destinataire par l'émetteur.
2. Révocation d'un partage par l'émetteur.
3. Révocation d'un partage par un destinataire.

Dans le 1er scénario, le Cloud personnel de l'émetteur indique que le destinataire est révoqué pour un partage donné, afin qu'il ne soit plus pris en compte dans le système de synchronisation. Il notifie également la plateforme du destinataire pour que ses accès liés à ce partage, devenus caduques, soient supprimés. S'il s'agit d'un partage *master-master*, les actions sont symétriques pour l'émetteur et le destinataire.

Le 2ème scénario est similaire au premier, à la différence que tous les destinataires sont notifiés pour qu'ils entreprennent les actions nécessaires.

Pour le 3ème scénario, le destinataire supprime les accès de l'émetteur et notifie ce dernier afin qu'il le considère comme révoqué et le supprime du système de synchronisation. Là encore, un partage *master-master* implique des actions supplémentaires pour que le destinataire supprime la synchronisation et que l'émetteur supprime les accès.

Enfin, il est à noter que dans le cas d'un partage *one-shot*, une révocation automatique du partage par l'émetteur a lieu lorsque tous les documents ont été partagés.

Dans tous les cas de figure, une révocation n'entraîne pas la suppression des documents partagés. Cela serait difficile à justifier du point de vue de l'expérience utilisateur, avec des destinataires qui pourraient voir des documents qu'ils considèrent comme acquis soudainement disparaître, ce qui serait particulièrement dommageable s'ils avaient eux-

mêmes ajouté ou modifié des documents. L'émetteur doit donc être conscient que tout en restant le propriétaire du partage, les destinataires obtiennent un droit sur ses données qui ne pourra être totalement révoqué. Cela n'impacte pas réellement la sécurité de l'approche car quel que soit le paradigme du partage utilisé, un sujet est toujours capable de procéder à une copie des données accédées, bien que des travaux autour du contrôle d'usage explorent des pistes pour l'empêcher ou au moins le complexifier [Katsouraki 2016].

Nous avons détaillé ici le protocole de partage du point de vue fonctionnel, avec les actions et échanges à effectuer pour permettre la création d'un partage sécurisé dans le Cloud personnel, qui s'inscrit dans la logique SWYSWYK. De nombreux aspects sont laissés suffisamment libres afin de permettre à différentes implémentations de cohabiter, plutôt que d'imposer des choix technologiques potentiellement incompatibles avec des plateformes de Cloud personnel déjà existantes. Dans la section suivante, nous détaillons les choix effectués pour l'implémentation de ce protocole dans la plateforme Cozy.

5.3 Implémentation du protocole

Nous décrivons ici comment le protocole qui vient d'être présenté a été implémenté dans la plateforme Cozy, sur deux moutures distinctes. Nous détaillons les choix technologiques qui ont été faits sur la dernière version et les implications en termes de performance, sécurité et simplicité d'usage.

5.3.1 Contexte

Le protocole de partage a été implémenté dans deux versions de Cozy radicalement différentes. En effet, courant 2016, l'entreprise Cozy Cloud a entamé une réflexion sur la faisabilité d'un passage à l'échelle de la plateforme pour accueillir de potentiels millions d'utilisateurs. Il est apparu qu'il devenait très difficile voire impossible de continuer avec les technologies jusqu'alors utilisées et en particulier la partie serveur écrite avec le *framework*

Node.js[60]. Les performances et la consommation mémoire rendaient économiquement irréaliste une solution d'hébergement de grande ampleur comme celle envisagée par Cozy Cloud.

Une refonte totale de la plateforme a alors été décidée. En particulier, toutes les applications sont devenues *client-side*, c'est-à-dire sans aucune partie serveur, afin de réduire au maximum l'impact mémoire. Tous les traitements serveur devant dorénavant passer par une API unifiée contrôlée par la nouvelle *stack* Cozy, entièrement réécrite en Go[61]. Ce langage, apparu en 2009 chez Google, est relativement bas niveau avec une syntaxe proche du C[62,] mais s'inspire également de concepts venant de langages plus récents comme le Python[63]. Langage compilé et typé, le Go, outre un impact mémoire faible, est particulièrement efficace pour gérer la concurrence entre les processus. Exactement ce qui était recherché par Cozy Cloud afin de pouvoir développer une offre commerciale qui permettrait d'héberger de multiples instances Cozy sur une même infrastructure. Cela répond aussi à une demande récurrente de la communauté, à savoir pouvoir gérer plusieurs utilisateurs sur un même serveur, ce qui était difficilement réalisable jusqu'alors, chaque instance nécessitant environ 2 Go de RAM.

La refonte de Cozy a nécessité une réécriture complète du protocole de partage en Go, à partir de la version Node.js. Nous ne présentons ici que les détails d'implémentation de la dernière mouture. Une documentation complète des API du partage Cozy est disponible en ligne[64]. A destination des développeurs, elle décrit les formats de données et les différentes requêtes possibles pour créer, révoquer et maintenir un partage. Le format des requêtes est de type REST [Fielding 2000] et se fait via des routes de type https://moncozy/sharing/maroute. Cette documentation faisant foi et étant destinée à être mise à jour plus souvent que ce document, nous ne détaillons pas les routes et formats de

[60] https://nodejs.org/

[61] https://golang.org/

[62] https://fr.wikipedia.org/wiki/C_(langage)

[63] https://www.python.org/

[64] https://cozy.github.io/cozy-stack/sharing.html

données ici. Au lieu de cela, nous discutons des aspects les plus importants et des choix technologiques qui ont été faits.

5.3.2 Déclaration d'un partage

La création du partage se fait par l'envoi d'une requête POST sur la route /sharings/ du Cozy du propriétaire. Cette route attend un certain nombre de paramètres, encodés en JSON. La requête doit également être authentifiée et avoir les permissions nécessaires pour effectuer cette action.

Un certain nombre de paramètres sont attendus, tels quel la description du partage, son type (*one-shot, master-slave* ou *master-master*), ainsi que l'expression des documents et des objets, autrement dit les *Filters* de SWYSWYK. Nous donnons en Figure 21 un exemple de document de partage produit suite à une requête réussie.

Permissions

L'expression des documents à partager correspond également aux permissions que doit accepter le destinataire du partage. En effet, la règle exprimée permet de sélectionner un ensemble de documents avec des actions *Create, Read, Update, Delete* associés. Cela correspond donc aux permissions dont l'émetteur doit disposer pour partager les documents avec le Cozy du destinataire. Une syntaxe spécifique a été développée pour exprimer des permissions sur n'importe quel type de données ; nous la détaillons en 5.3.4 et donnons des exemples dans la section 5.4, qui traduisent des cas d'usages implémentés.

Destinataires

Les sujets du partage sont représentés ici comme un tableau de *recipients* où chaque élément est une référence vers le document qui contient les informations du destinataire, à minima son email ou l'URL de son Cozy. Dans l'exemple donné Figure 21, un *recipient* est représenté par une fiche contact dans le Cozy. Cela reprend le principe des *sujets sont des documents* de SWYSWYK où un sujet est représenté par un document visualisable, ici par une application de gestion des contacts.

A chaque destinataire du partage est associé un statut qui peut être par exemple *accepted* si le destinataire a accepté, *rejected* dans le cas contraire, *pending* s'il n'a pas encore répondu ou *revoked* s'il est révoqué. Ce champ est également utilisé pour indiquer les éventuelles erreurs lors du déroulement du protocole.

Si tous les paramètres sont correctement renseignés, un document de type partage est créé. Un exemple est donné en Figure 21 de type partage *master-slave* avec deux destinataires. Ce document est donné à titre informatif et ne contient pas toutes les informations liées au partage, afin de ne pas alourdir inutilement sa structure.

```
{
    "_id": "xxx",
    "_rev": "yyy",
    "type": "io.cozy.sharings",
    "sharing_type": "master-slave",
    "desc": "This is a human-readable description",
    "sharing_id": "zzz",
    "owner": true,

    "permissions": {
        "doctype1": {
            "description": "doctype1 description",
            "type": "io.cozy.doctype1",
            "values": ["docid1", "docid2"],
            "selector": "calendar-id",
            "verbs": ["GET","POST", "PUT"]
        }
    },
    "recipients": [
        {
            "recipient": {"id": "mycontactid1", "type":
"io.cozy.contacts"},
            "status": "accepted",
        },
        {
            "recipient": {"id": "mycontactid2", "type":
"io.cozy.contacts"},
            "status": "pending"
        }
    ]
}
```

Figure 21. Exemple de document de partage.

5.3.3 Notification d'un destinataire

Lorsqu'un partage est créé, chaque destinataire doit être notifié afin de pouvoir récupérer les documents auxquels il a droit. Dans le Web actuel où les moyens de communications sont légions et disparates, le courriel apparaît comme le moyen le plus universel et interopérable pour contacter un destinataire. Nous l'utilisons donc comme système de base pour notifier le sujet d'un partage. Cela implique donc que le propriétaire d'un Cloud personnel connaisse à minima l'adresse e-mail d'un sujet pour pouvoir partager avec lui. Si c'est bien le cas, un e-mail est automatiquement généré et contient une URL forgée à partir des métadonnées du partage qui devra être cliquée par le destinataire pour qu'il puisse visualiser le partage. Deux cas de figure peuvent se produire :

- **Cas 1 : l'émetteur du partage connaît l'adresse du Cozy du destinataire.** L'e-mail contient alors une URL forgée à partir de cette adresse et qui redirige le destinataire vers sa propre plateforme, afin qu'il puisse visualiser les permissions demandées et accepter le partage.

- **Cas 2 : L'émetteur du partage ne connaît pas l'adresse du Cozy du destinataire**. L'e-mail contient une URL qui redirige le destinataire vers la plateforme de l'émetteur. Il lui est alors demandé de rentrer l'adresse de son Cozy pour recevoir le partage. Une fois cela fait, le destinataire est redirigé vers son Cloud personnel pour visualiser les permissions. Parallèlement, le Cozy de l'émetteur complète automatiquement la fiche contact du destinataire avec l'URL fournie, dans le respect du principe **d'auto-administration**.

Ce système de notification et de visualisation des permissions est indispensable pour s'assurer du principe **d'*empowerment* éclairé,** mais cette fois du point de vue du destinataire. Comme ce dernier accorde des droits en écriture sur sa plateforme afin de recevoir les documents partagés, il doit s'assurer que ce qui lui est demandé soit bien légitime. La Figure 22 montre un exemple de visualisation des permissions pour un partage de répertoire.

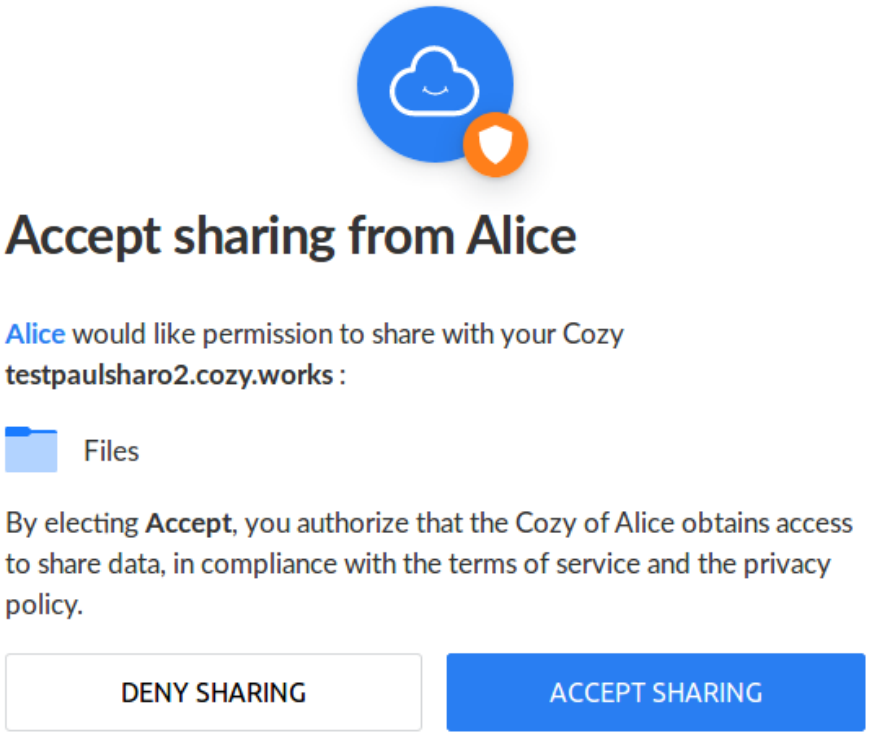


Figure 22. Pages des permissions

Ici, la connaissance d'un destinataire est conditionnée par la connaissance de son e-mail. Le problème de l'identité sur internet est un sujet intéressant mais néanmoins complexe, et de nombreux travaux existent pour tenter de le résoudre, au moins partiellement. Citons par exemple WebID [Inkster 2017], présenté en 2.2.1, qui propose d'utiliser des URI sémantiques comme moyen d'échanger des identités et d'en découvrir de nouvelles grâce à graphe social exprimé en FOAF. De même, XDI [Reed 2004] recommande un identifiant unique sur le Web pour chaque individu auquel chacun peut associer des traits d'identité publics ou privés. Des registrars sont alors en charge d'assurer la correspondance entre identifiant et trait d'identité, de la même façon qu'un DNS traduit un nom de domaine en IP.

Malheureusement, ces travaux n'ont pas encore fait consensus et sont pour le moment difficilement exploitables dans un contexte industriel comme celui de Cozy Cloud. Cependant, bien qu'extrêmement populaire, le courriel souffre d'un problème natif de sécurité ; le contenu des e-mails est transmis en clair sur le réseau, ce qui le rend vulnérable à des attaques de confidentialité. Un attaquant qui écoute le réseau serait potentiellement capable d'intercepter un e-mail de partage, avec les conséquences décrites ci-dessous.

S'il s'agit du **cas 1**, où l'URL contenu dans l'e-mail redirige vers le Cozy du destinataire, l'attaquant ne pourra rien faire de plus qu'apprendre l'existence du partage entre les deux pairs. Il sera incapable de connaître le contenu des documents, ni d'impacter le déroulement du partage. Une fois établi, tous les échanges du protocole sont chiffrés via TLS[65]. L'attaquant ne pourra donc rien intercepter de plus que ce qui est dévoilé par ce protocole, c'est-à-dire les métadonnées des échanges, comme les adresses IP de l'émetteur et du destinataire.

En revanche, s'il s'agit du **cas 2**, l'URL redirige vers une page du Cozy de l'émetteur qui n'est pas protégée, car le destinataire ne dispose alors d'aucun secret partagé avec l'émetteur. L'attaquant serait donc capable d'entrer l'adresse de son propre Cloud personnel pour recevoir le partage à la place du destinataire légitime et potentiellement tous les futurs partages, si l'émetteur ne se rend pas compte de la supercherie.

Plusieurs solutions peuvent être envisagées pour se protéger de ce genre d'attaque. PGP [Zimmermann 1995], détaillé en 2.2.1, permet de chiffrer le contenu des e-mails de bout en bout et d'authentifier l'émetteur. Mais il est loin d'être massivement utilisé, notamment à cause de sa difficulté d'utilisation et la complexité liée à la signature des clés et de la confiance qui peut leur être accordée. De plus, seul le contenu est chiffré, les en-têtes contenant les métadonnées restent en clair.

Le protocole OTR [Borisov 2004] est une autre piste envisageable. Il permet de réaliser des échanges asynchrones chiffrés en évitant certaines faiblesses de PGP, en particulier le manque de *forward secrecy*, qui se traduit par le fait que la compromission d'une clé PGP permet de déchiffrer toutes les communications passées. La répudiation est une autre propriété intéressant de OTR, qui permet à un tiers de nier avoir eu des communications passées avec un sujet.

D'autres protocoles préfèrent l'utilisation du numéro de téléphone comme identifiant universel [Ermoshina 2016]. Ce système se révèle efficace pour les communications mobiles chiffrées [Cohn-Gordon 2016], mais requiert l'installation d'une application dédiée sur un

---

https://tools.ietf.org/html/rfc5246

appareil du destinataire, typiquement son Smartphone, ce qui n'est pas vraiment adapté à l'usage du Cloud personnel.

Bien que des solutions existent, aucune ne s'impose réellement et ce problème reste ouvert. L'implémentation actuelle utilise les e-mails pour notifier les destinataires, via STARTTLS qui impose l'utilisation du chiffrement TLS durant le transport, donc de protéger contre un attaquant qui espionnerait le réseau. Dans le futur, nous souhaiterions utiliser des systèmes plus robustes, tout en préservant une expérience utilisateur fluide. La simplicité de la découverte de nouveaux contacts et la façon de les notifier est un point crucial pour la facilité d'utilisation du partage et de son adoption. Nous espérons donc que le modèle actuel servira de base à l'utilisation de moyens plus évolués et sécurisés pour atteindre ces objectifs.

5.3.4 Authentification et autorisations

Une fois qu'un destinataire accepte un partage, un jeton d'authentification est généré par sa plateforme et envoyé à l'émetteur. Cet échange repose sur les spécifications OAuth 2.0[66] et permet d'associer le jeton généré aux permissions accordées. OAuth est un standard qui spécifie un ensemble de recommandations pour construire un protocole d'autorisation et d'authentification robuste et interopérable entre différentes plateformes. Dans Cozy, OAuth est utilisé pour authentifier les requêtes des applications, des appareils distants et des partages. On distingue quatre acteurs aux rôles spécifiques dans OAuth :

- **Le propriétaire des ressources**, autrement dit l'utilisateur physique qui accorde des accès.
- **Le client,** l'entité qui souhaite accéder aux ressources. Cela peut typiquement être une application ou un appareil connecté.
- **Le serveur d'autorisation** qui délivre les jetons d'authentification.
- **Le serveur de ressources**, là où les documents sont stockés.

[66] https://oauth.net/2/

Nous illustrons ici le rôle de chaque acteur lors de l'installation d'une application sur un Cozy afin de faire un parallèle avec le partage.

Le propriétaire du Cozy est le **propriétaire des ressources.** L'application joue ici le rôle du **client** et commence par transmettre une requête d'enregistrement au serveur Cozy qui est le **serveur d'autorisation**. Cette requête doit contenir les informations de base concernant l'application, par exemple son nom, numéro de version, etc. Le serveur répond en lui renvoyant notamment un identifiant appelé *client_id*. Le client envoie alors une requête d'autorisation au serveur, en spécifiant son *client_id*, accompagné d'un *scope,* qui décrit l'ensemble des documents et actions dont l'application estime avoir besoin. Cela correspond aux permissions demandées. La *stack* Cozy génère alors une page de permissions, similaire à celle montrée en Figure 22. Si le propriétaire accepte les permissions en cliquant sur le bouton dédié, le serveur génère un jeton d'authentification qu'il transmet à l'application, ce qui lui donne les droits sur les permissions demandées, pour une durée fixée. L'application pourra alors utiliser ce jeton pour requêter des documents via l'API de la *stack,* qui sera alors le **serveur de ressources**.

Cette mécanique est la même pour les applications mobiles et *desktop.* Pour le partage, la logique reste la même, bien que les acteurs diffèrent. Le **client** est le Cozy de l'émetteur du partage, le **propriétaire des ressources** est le destinataire, tandis que son Cozy est le **serveur d'autorisation et de ressources**. En effet, c'est le destinataire qui accorde un accès à sa plateforme pour que des documents y soient répliqués et potentiellement synchronisés. Dans le cas d'un partage *master-master*, tous les rôles sont joués de façon symétrique pour permettre au destinataire d'avoir également un accès chez l'émetteur. Seule la partie d'acceptation des permissions chez l'émetteur n'a pas lieu, car on la considère comme implicite.

Afin de permettre l'expression de permissions plus ou moins complexes, une syntaxe inspirée du langage de requêtes du moteur NoSQL MongoDB[67] a été définie. Il est par exemple possible de donner un droit d'écriture pour un unique document, pour l'ensemble des documents d'un certain type, etc.

[67] https://www.mongodb.com

Une permission est représentée par un objet JSON qui contient plusieurs champs :

- **Type** : caractérise le type des documents sur lesquels la permission s'exerce, en utilisant la nomenclature Cozy[68]. Par exemple, les fichiers sont représentés par le type *io.cozy.files*, les contacts par *io.cozy.contacts*, etc.

- **Verbs** : l'action accordée sur les documents. Les verbes supportés sont ceux définis par la spécification HTTP[69] et permettent de définir des droits en lecture, écriture, mise à jour et suppression. A noter qu'un verbe « ALL » permet d'accorder tous les droits.

- **Values** : tableau d'identifiants qui correspondent aux documents sur lesquels s'exercent les permissions.

- **Selector** : permet de spécifier un champ arbitraire pour sélectionner un ensemble de documents. Par exemple, si l'on a un calendrier dont chaque évènement contient une référence 'calendar-id', l'utilisation d'un sélecteur sur ce champ permettrait de sélectionner tous les évènements. Le champ *values* contiendrait alors l'identifiant du calendrier.

Il est possible de combiner plusieurs permissions, par exemple pour exprimer un partage qui porte sur différents types de documents. Plusieurs exemples de permissions sont donnés dans la section suivante sur les cas d'usages et une documentation complète est disponible en ligne[70].

5.3.5 Réplication des documents

Comme dit précédemment, la réplication et la synchronisation des données en pair à pair est un vaste sujet qui a fait l'objet d'importants travaux depuis de nombreuses années. Le

[68] https://github.com/cozy/cozy-doctypes

[69] https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html

[70] https://cozy.github.io/cozy-stack/permissions.html

challenge ici n'est pas académique, mais bien industriel ; la plupart des protocoles de synchronisation actuels étant centralisés et fermés.

Le protocole de réplication CouchDB [CouchDB Replication Protocol 2017] possède les avantages d'être libre, décentralisé, documenté et robuste. De plus, son intégration native dans Cozy le pose comme un candidat particulièrement crédible pour la réplication des données entre serveurs Cozy. De fait, la première implémentation du partage en Node.js l'utilise. Cela garantit des performances et une fiabilité générale grâce à la reprise sur panne de la synchronisation, de la gestion de l'historique des versions, etc.

Cependant, la dernière version de Cozy faite en Go n'utilise pas cette réplication dans sa première implémentation. En effet, suite à des tests de montée en charge, il est apparu que la consommation mémoire de CouchDB augmentait considérablement en fonction du nombre de partages et de documents, comme montré en Figure 23. Ce comportement n'est néanmoins pas totalement déterministe et difficile à appréhender sans rentrer dans les détails d'implémentation de CouchDB, codé en Erlang[71]. Conjugué au fait que l'implémentation du protocole de réplication n'est pas une tâche triviale, cela rendait la livraison du partage incertaine au vu des délais et contraintes de Cozy Cloud.

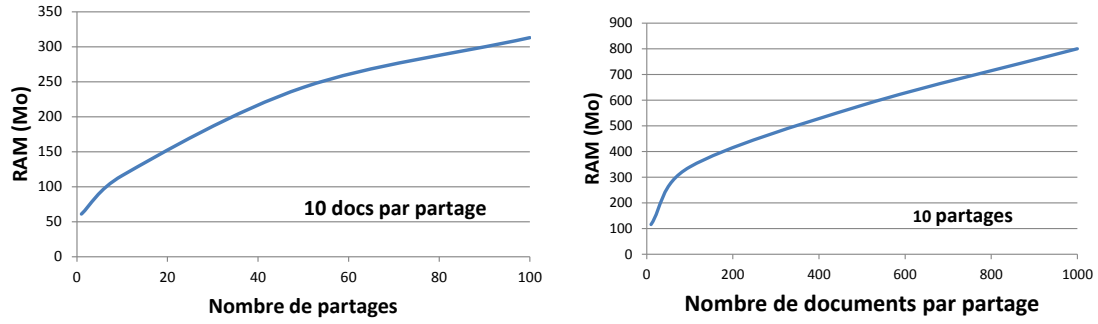


Figure 23. Consommation de RAM en fonction du nombre de partages et documents

Le choix a donc été fait de se passer de la réplication CouchDB dans un premier temps, sans que cela ne soit définitif. Il est notamment envisagé d'utiliser la réplication CouchDB pour les utilisateurs auto-hébergés, c'est-à-dire qui ont installé Cozy sur leurs propres

[71] https://www.erlang.org/

machines. Dans ce cas de figure, les contraintes d'utilisation mémoire sont généralement moins fortes que dans une architecture mutualisée, tandis que le besoin de résilience l'est d'autant plus, la stabilité du réseau étant moins garantie que sur des infrastructures spécialisées. De plus, les dernières versions de CouchDB laissent entrevoir un impact mémoire moindre grâce à une mutualisation des processus de réplication.

A la place, nous avons implémenté un système de réplication et de synchronisation qui combine des appels HTTP et un système de *triggers,* dont la documentation est accessible en ligne[72], pour répliquer les documents et maintenir le partage à jour dans les partages *master-**. Chaque document créé, mis à jour ou supprimé est évalué afin de déterminer s'il rentre dans un partage, suivant le principe de l'évaluation des *Filters* de SWYSWYK. S'il est correctement évalué, un processus appelé *worker* est déclenché par le *trigger*, qui propage la modification chez les pairs. Afin de s'assurer que seules les nouvelles modifications sont propagées, un système de gestion de versions simple a été implémenté : avant chaque propagation, le *worker* compare les *hash* des documents pour s'assurer qu'il y a bien des différences. Outre le fait que cela évite des communications réseaux inutiles, cette vérification évite de se retrouver dans un phénomène de boucle infinie pour un partage *master-master* : si le document partagé D1 est modifié chez Alice, il rentre dans un *trigger,* et est propagé chez Bob. Or, cela serait considéré chez lui comme une nouvelle mise à jour, et un *worker* chercherait de nouveau à le propager chez Alice qui irait le re-propager, etc.

Enfin, dans le cas où Alice et Bob feraient chacun des mises à jours simultanées sur le même document chacun de leur côté, cela crée un conflit : il est impossible de savoir quelle version est la plus récente et une fusion automatique n'est pas toujours possible. Dans ce cas, une nouvelle version du document est créée, avec le tag *conflict* dans son nom ainsi que la date et l'heure de la mise à jour qui a provoqué le conflit. A charge d'Alice et Bob de fusionner manuellement les documents en conflits.

Ce système est une part importante de l'implémentation du partage dans Cozy, dont le code est accessible en ligne[73], mais de nombreux points restent perfectibles. Par exemple, les

[72] https://cozy.github.io/cozy-stack/jobs.html

[73] https://github.com/cozy/cozy-stack/tree/master/pkg/workers/sharings

mises à jour devant être propagées pourraient être regroupées dans une mémoire tampon si elles sont très fréquentes, pour optimiser les coûts réseaux. Ces écritures pourraient aussi être persistées dans un document de la base de données, afin de pouvoir assurer une reprise sur panne en cas de défaillance. De plus, le processus de synchronisation est asynchrone et pourrait être exécuté en isolation du reste de la *stack*, comme c'est déjà le cas pour certaines tâches. Par exemple les connecteurs, en charge de se connecter à des sites externes pour importer les données de l'utilisateur (factures, historique bancaire, données *IoT*, etc) sont isolés via nsjail[74]. Cela peut être vu comme une implémentation logicielle de l'environnement isolé de SWYSWYK. Enfin, comme dit précédemment, tous les échanges se font via des requêtes HTTP protégées par TLS. Cela ne protège cependant pas les métadonnées du partage (on sait qu'Alice communique avec Bob et à quelle fréquence, mais sans connaître le contenu). Ce problème pourrait être évité en explorant des solutions utilisant TOR[75] où les destinataires et émetteurs des requêtes sont anonymisés, au risque d'une dégradation des performances.

Ainsi, à l'heure où ces lignes sont écrites, des efforts d'implémentations et des tests sont toujours en cours pour garantir la fiabilité et la robustesse du protocole.

5.4 Cas d'usage

Nous présentons ici trois cas d'usage du partage dans Cozy, dans l'ordre chronologique de leur implémentation. Chaque cas d'usage repose sur l'utilisation du protocole de partage décrit dans les précédentes sections et sur des interfaces utilisateurs spécifiques qui appellent les API du partage de Cozy.

[74] http://nsjail.com/

[75] https://www.torproject.org/

5.4.1 Evènements de calendrier

Le partage d'évènement a été le premier cas d'usage implémenté, dans la version de Cozy en Node.js. Cette implémentation valide une première utilisation du protocole de partage avec une structure de données simple : l'objet du partage est un unique document JSON dans lequel sont indiqués les invités de l'évènement. Du point de vue SWYSWYK, il s'agit donc d'une règle de partage réflexive.

Dans ce scénario, le propriétaire du Cozy sélectionne ou crée un évènement à partager et indique s'il souhaite ajouter des invités. La base de contacts du Cozy est alors requêtée, ce qui permet au propriétaire d'obtenir une auto-complétion sur les noms, e-mails et URL. Dans le cas où une URL existe pour un contact, le propriétaire a la possibilité de partager avec celui-ci l'évènement directement dans son Cozy[76]. Le destinataire est alors notifié de la demande de partage, qu'il peut accepter ou refuser. S'il accepte, l'évènement est automatiquement ajouté à un calendrier partagé et synchronisé avec les appareils du destinataire, s'il a activé cette option.

Toute modification ultérieure de l'évènement par le propriétaire (changement de date, notes complémentaires, etc.) est instantanément propagée chez les destinataires qui sont avertis du changement grâce à un système de notification propre à Cozy.

[76] L'implémentation utilisée pour ce cas d'usage n'est pas tout à fait celle décrite en 5.3, car réalisée sur la précédente version de Cozy. Notamment, le système de notification utilise une API interne qui permet de commmuniquer directement entre serveurs Cozy et se passer de l'e-mail.

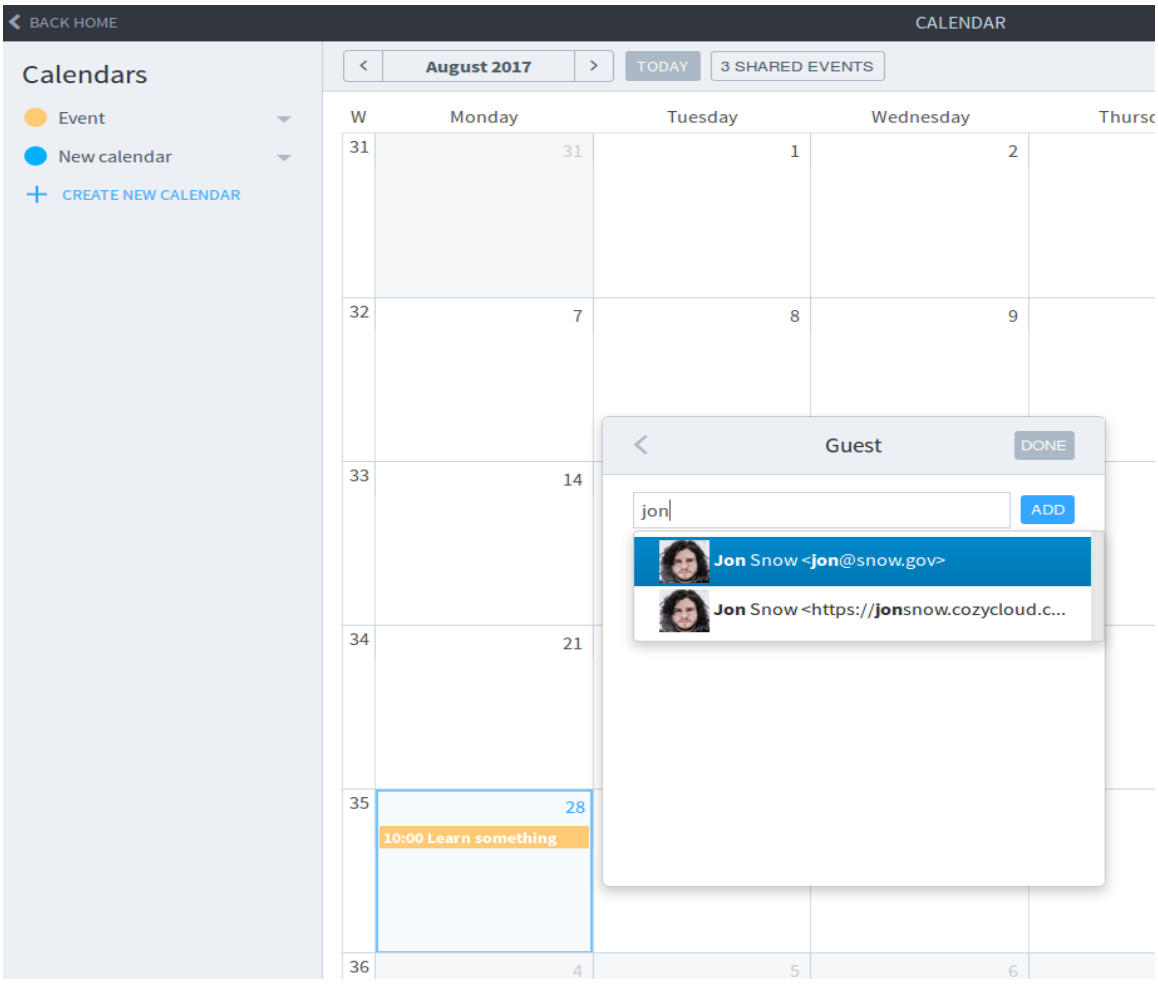


Figure 24. Partage d'un évènement

Ce cas d'usage qui concrétise des travaux à la fois sur l'interface et sur la partie serveur**,** fut l'occasion d'une annonce dans le forum Cozy[77] et d'une présentation aux utilisateurs lors d'un *meetup* Cozy Cloud[78]. La Figure 24 montre l'interface utilisateur pour ajouter un destinataire depuis un évènement.

Quelques mois plus tard, dans un souci de passage à l'échelle pour de potentiels millions d'utilisateurs, Cozy Cloud décida de la refonte complète de la plateforme, entraînant une nouvelle implémentation du protocole de partage et de nouveaux cas d'usages.

[77] https://forum.cozy.io/t/share-events-between-cozy/2881

[78] https://www.meetup.com/fr-FR/Meetup-des-utilisateurs-de-Cozy-Cloud-en-France/

5.4.2 Album Photos

La refonte de Cozy s'est également accompagnée d'un recentrage des développements autour des applications *Files* et *Photos*. De fait, le partage d'album a rapidement été identifié comme le premier cas d'usage à implémenter dans la nouvelle mouture de Cozy. Répondant à une vraie problématique commerciale, ce partage est plus complexe que celui des évènements. En effet, il faut ici pouvoir gérer une collection dynamique de fichiers binaires et supporter des ajouts et suppressions, contrairement au partage d'évènement qui consiste en un unique document.

En base de données, un album photo est représenté par un document de type *io.cozy.photos.albums,* qui contient les métadonnées de l'album, comme le nom ou la date. Parallèlement, chaque photo est représentée par un document de type *io.cozy.files* qui contient entre autres choses le chemin du fichier binaire et l'identifiant du document représentant l'album, donné dans un champ *referenced_by*. Ce champ est indexé, pour récupérer rapidement toutes les photos d'un album donné à partir de son identifiant.

Pour exprimer le partage d'album photo, une syntaxe spécifique a été utilisée, décrite dans la section 5.3.4, dont l'expression complète donnée en Figure 25 pour ce cas d'usage. Deux types de permissions sont nécessaires, pour exprimer à la fois le partage du document qui décrit l'album photos, et son contenu, c'est-à-dire toutes les photos référencées par des documents *io.cozy.files*.

```
"permissions": {
    "photos": {
        "description": "Holidays album",
        "type": "io.cozy.albums.photos",
        "values": ["albumdocid"],
        "verbs": ["ALL"]
    },
    "files": {
        "description": "Holidays photos",
        "type": "io.cozy.files",
        "values": ["io.cozy.albums.photos/albumdocid"],
        "selector": "referenced_by",
        "verbs": ["ALL"]
    }
}
```

Figure 25. Déclaration d'un partage d'album photo

Une fois le partage exprimé, un *trigger* est créé, qui écoute en permanence les ajouts, modifications et suppression des fichiers référencés par l'album photo, ainsi que sur l'objet album photo lui-même.

La Figure 26 montre l'interface utilisateur pour partager un album photo. A noter qu'il est possible de définir le partage en lecture seule (« *Can View »,* partage *master-slave*) ou avec des droits d'écriture (« *Can Write* », partage *master-master*).

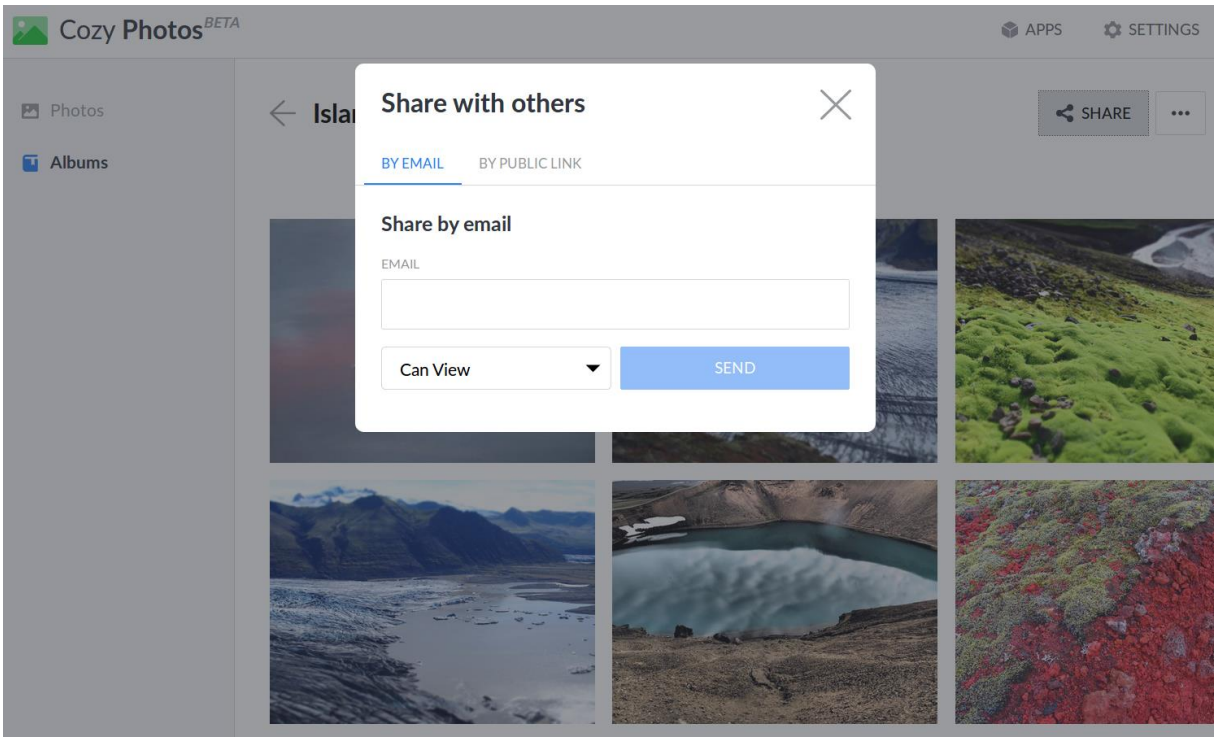


Figure 26. Partage d'un album photo

Ce cas valide le partage dans la nouvelle version de Cozy. Les efforts furent ensuite mis sur le partage de répertoires.

5.4.3 Répertoires

Le partage de répertoires est un des cas d'usage les plus fréquents pour les services de gestion de fichiers. C'est une fonctionnalité incontournable aussi bien dans les solutions centralisées (Dropbox, Google Drive, etc) que décentralisées (nextCloud, Seafile, etc).

Contrairement à l'album photos qui est une structure « à plat », les répertoires ont une structure hiérarchique et peuvent contenir des sous-répertoires, eux-mêmes pouvant avoir des fichiers et sous-répertoires, etc.

Un système de fichiers virtuel, ou *Virtual File System* (VFS) est utilisé dans Cozy pour représenter cette hiérarchie et fournir les API nécessaires pour la manipuler[79]. Cela fournit l'abstraction nécessaire pour utiliser des couches de stockage qui peuvent varier selon

[79] https://cozy.github.io/cozy-stack/files.html

l'hébergement. Par exemple, Swift[80] pour un hébergement mutualisé, ou le système de fichiers du système d'exploitation du serveur en cas d'auto-hébergement. Au niveau de la base de données, chaque répertoire et fichier est représenté par un document JSON qui contient entre autres un champ *dir_id* qui indique l'identifiant du répertoire parent. Il est laissé vide pour le répertoire racine. Ce champ est à la base de l'indexation pour calculer la hiérarchie et pour en assurer les divers contrôles d'intégrité, par exemple s'assurer qu'un enfant n'a qu'un seul parent.

La Figure 27 montre un exemple d'expression de partage de répertoire, qui peut se résumer à indiquer l'identifiant du dossier à partager dans le champ *values*. Toute la hiérarchie est automatiquement gérée par un *trigger* dédié. Ce dernier parcourt récursivement la hiérarchie et écoute toute modification pouvant avoir lieu sur le dossier racine ou un de ses descendants de n'importe quel niveau afin de mettre à jour le partage.

```
"permissions": {
    "files": {
        "description": "Dear Hectory",
        "type": "io.cozy.files",
        "values": ["dirid"],
        "verbs": ["ALL"]
    }
},
```

Figure 27. Déclaration d'un partage de répertoire

L'interface pour partager un répertoire est la même que pour un album photo ; en effet le code *front-end* a été mutualisé afin d'obtenir une interface cohérente pour l'utilisateur entre les applications et pour faciliter le développement de nouveaux cas d'usages en disposant d'une librairie clé en main pour le partage.

[80] https://wiki.openstack.org/wiki/Swift

A noter qu'il est aussi possible de partager un simple fichier : l'expression du partage reste la même, et le fait que ce soit un fichier sera automatiquement détecté par l'API du VFS. Ce cas peut être vu comme un sous-ensemble du partage de répertoire, sans aucune gestion de hiérarchie.

Ainsi, ce dernier cas d'usage concrétise la volonté de Cozy d'obtenir un *Minimum Valuable Product* (MVP), recentré autour des applications permettant de gérer des fichiers, en les important depuis des sources externes (application *Collect*) ou de les visualiser en albums photos (application *Photos*). Les fonctionnalités de partage sont essentielles pour un produit comme Cozy, qui vise à devenir le domicile numérique des individus et donc devenir le point d'entrée pour des fonctionnalités sociales et décentralisées.

De nombreux autres cas d'usages intéressants peuvent être envisagés dans le futur. Aux classiques partages de calendrier ou notes, des scénarios plus précis pourraient être explorés. Par exemple, le partage en lecture seule d'un compte bancaire par un mineur ou une personne âgée avec des membres de sa famille. Ces derniers pourraient alors superviser les comptes depuis leur propre Cloud personnel, sans pouvoir intervenir sans l'aval du propriétaire du partage. Pour rester dans le domaine financier, on peut également imaginer pouvoir créer un compte commun sur une crypto-monnaie comme Bitcoin [Nakamoto 2008], via un partage en écriture sur le portefeuille électronique. Enfin, pour exploiter toute la puissance d'expression du partage, des règles sur des données transverses peuvent être envisagées, par exemple l'ensemble des documents relatifs à un projet, qu'ils soient des rapports, des entrées de calendrier, des factures, etc. Ou encore partager les PDF de papiers de recherche d'un sujet précis, avec les notes correspondantes prises dans une application séparée. Ce seraient autant de cas d'usages intéressants, et qui s'inscrivent parfaitement dans la logique de SWYSWYK.

5.5 Conclusion

Nous avons présenté ici l'implémentation industrielle du modèle SWYSWYK dans la plateforme de Cloud personnel Cozy. Bien que partielle, cette implémentation permet l'expression de règles de partage basiques et réflexives dont l'évaluation via un système de

triggers permet de générer de nouvelles permissions et de gérer dynamiquement les évolutions des documents impactant les partages. Trois cas d'usages ont été présentés, chacun ayant été l'objet d'un déploiement en production et utilisable par les utilisateurs Cozy Cloud.

Le protocole respecte la propriété **d'*empowerment* éclairé** en permettant d'une part à l'émetteur de pouvoir à tout moment visualiser les documents partagés et leurs destinataires, et d'autre part en générant une page de permissions afin que chaque destinataire soit conscient de ce qui lui est partagé et des droits associés, pour qu'il l'accepte ou refuse en toute conscience.

La propriété d'**auto-administration** des sujets est partiellement réalisée par l'utilisation des fiches contacts du Cozy pour représenter les destinataires du partage et en laissant la possibilité à ces derniers de compléter eux-mêmes l'adresse de leur Cloud personnel afin que leur fiche contact chez l'émetteur soit automatiquement complétée.

En revanche, la propriété de **sécurité par construction** n'est pas respectée ici. Garantie par l'utilisation de l'architecture, présentée au Chapitre 3, le respect de cette propriété impose de nombreuses contraintes afin de garantir qu'aucune donnée ne puisse être divulguée à l'insu du propriétaire. Elle nécessite notamment l'utilisation d'un environnement sécurisé, par exemple le matériel sécurisé PlugDB, et l'utilisation d'interfaces d'administration physiquement isolées. Or, l'objectif initial pour Cozy Cloud était de pouvoir disposer d'un système de partage simple, performant, avec une sécurité acceptable et pouvant être déployée à la fois en auto-hébergement, c'est-à-dire un serveur physiquement directement géré par un utilisateur, et dans une infrastructure mutualisée gérée par un hébergeur professionnel. La faisabilité technique de l'intégration de PlugDB dans Cozy a été prouvée par des démonstrateurs [Tran-Van, SWYSWYK: A New Sharing Paradigm for the Personal Cloud. 2017] [Tran-Van, Partage de documents sécurisé entre Cloud personnels 2015] et son intérêt motivé dans [Andre 2016]. Cependant, cette intégration implique un accès physique aux serveurs du propriétaire, ce qui est le cas pour un hébergement à la maison, mais n'est pas possible avec des infrastructures mutualisées, sur lesquelles repose l'offre commerciale Cozy Cloud.

La sécurité de l'implémentation actuelle, certes réduite, n'est néanmoins pas nulle, grâce à l'utilisation du chiffrement de la base de données et des échanges entre les pairs, au

protocole d'authentification fondé sur le standard OAuth2 et du système de permission. L'utilisation de systèmes d'isolations tels que nsjail ou des enclaves sécurisées de type Intel SGX [Costan 2016] ou ARM TrustZone [Alves 2004] sont des pistes envisageables pour disposer d'une sécurité accrue, même sur des solutions d'hébergement mutualisé. Mais leur incorporation dans l'offre commerciale Cozy Cloud reste à faire et doit répondre à des problématiques qui vont au-delà de la seule dimension technique.

Chapitre 6
Conclusion et Perspectives

Alors que nos vies se numérisent et que le volume des données personnelles explose, un pistage généralisé de plus en plus puissant et intrusif se met en place, au mépris de la vie privée des individus. Malgré des réglementations qui cherchent à atténuer ce phénomène, de nombreuses entreprises et gouvernements tirent parti de ce système, extrêmement profitable économiquement et qui apporte la promesse d'un contrôle accru des individus.

Fermement opposées à ce glissement progressif vers des sociétés dystopiques, des alternatives se mettent en place, afin de remettre l'individu au centre des décisions et de la sécurité. Le Cloud personnel fait partie de ce mouvement et contribue à **l'émancipation numérique** des individus en leur permettant de stocker, manipuler et partager leurs données en toute confiance, dans une logique de **responsabilisation et de reprise du contrôle**.

Dans cette thèse, nous avons adressé la problématique du **partage de documents sécurisé dans le Cloud personnel**. En particulier, nous nous sommes penchés sur une architecture *Privacy-by-Design* adaptée à notre contexte, ainsi qu'un modèle de partage dédié au Cloud personnel. Enfin, un protocole de partage a été développé dans la plateforme Cozy pour permettre son utilisation par des utilisateurs finaux.

Ce chapitre conclut cette thèse : les travaux présentés tout au long de ce document y sont résumés et des perspectives futures en sont extraites.

6.1 Résumé

Comme dit au Chapitre 1, cette thèse se fait dans le contexte du Cloud personnel, qui est un paradigme mettant l'accent sur l'*empowerment* de l'utilisateur final. En particulier, nous nous concentrons sur le partage sécurisé dans ce type d'environnement, afin de donner à l'utilisateur des garanties fortes sur le respect de sa vie privée vis-à-vis de la dissémination de ses données, tout en mettant l'accent sur la simplicité d'utilisation et la diversité des usages.

Le Chapitre 2 fait un tour d'horizon des différents travaux autour du partage sécurisé et met en lumière les limitations des modèles actuels. En particulier, le fait que la **responsabilisation** de l'utilisateur prônée par le Cloud personnel est en contradiction avec le fait qu'il est généralement incapable d'assumer son **administration**. De plus, la **versatilité** du Cloud personnel le rend peu adapté aux modèles généralement conçus pour des systèmes aux formats de données et aux sujets bien définis. De ce constat, nous avons dérivé trois propriétés à respecter pour obtenir un système de partage sécurisé : (i) **l'*empowerment* éclairé,** (ii) **l'auto-administration** et (iii) **la sécurité par construction.**

Afin d'y répondre, nous avons conçu SWYSWYK, pour *Share What You can See with Whom You Know*, qui est constitué d'une partie architecturale et d'une partie modèle.

La partie architecturale de SWYSWYK, présentée au Chapitre 3, se concentre sur les propriétés d'*empowerment* éclairé et de sécurité par construction. Des outils d'administration y sont détaillés, afin que le propriétaire puisse aisément contrôler et appréhender les effets concrets de sa politique de partage. De plus, une architecture de référence y est proposée, à partir de laquelle diverses instances peuvent être dérivées en fonction du contexte d'exécution recherché. Afin de valider la faisabilité de cette approche, nous avons conçu une plateforme expérimentale et mené une évaluation de performances à la fois qualitative et quantitative. Les résultats sont prometteurs et laissent envisager une utilisation efficace sur des instances sans coût de communication. De plus, un démonstrateur réalisé sur la base de cette plateforme permet de donner un aperçu concret des outils d'administration, tout en ayant une perception tangible de la sécurité.

Le Chapitre 4 détaille la partie modèle de SWYSWYK. Nous y exposons un modèle de partage dédié au Cloud personnel, capable de gérer sa versatilité et sa volatilité, tout en

respectant les principes architecturaux de base. Ce modèle permet de respecter les propriétés d'*empowerment* éclairé et d'auto-administration en exploitant le contenu même du Cloud personnel, pour en extraire directement des permissions et des sujets. Une analyse de performance a également été menée pour évaluer l'impact du modèle, qui est grandement dépendant du nombre de sujets et reste donc viable pour un usage personnel. Un démonstrateur permet de donner des exemples concrets de règles de partage transverses qui permettent au propriétaire de développer de nouveaux usages autour de ses données personnelles tout en contrôlant leur dissémination.

Enfin, le Chapitre 5 présente un protocole de partage conçu pour permettre à des propriétaires de Cozy, le Cloud personnel édité par l'entreprise Cozy Cloud, de partager certains documents avec leurs contacts. Trois cas d'usage ont été implémentés : le partage d'évènement de calendrier, de répertoire et d'album photo. Sans respecter tous ses principes, le protocole intègre de nombreux concepts de SWYSWYK et laisse ouverte la discussion pour permettre l'intégration de l'architecture sécurisée dans une offre commerciale.

6.2 Perspectives

Nous espérons que les travaux présentés dans cette thèse ouvriront de nouvelles perspectives excitantes autour du Cloud personnel sécurisé et respectueux de la vie privée. Diverses directions peuvent être explorées sur la base de nos travaux, avec des problématiques à la fois scientifiques et industrielles, mais aussi sociétales.

Outils d'administrations. Le paradigme du Cloud personnel impose une responsabilisation du propriétaire qui va de pair avec l'administration de son espace. Il est donc essentiel que des outils adéquats soient développés afin de permettre aux propriétaires de gérer et disséminer correctement leurs données. Idéalement cette administration doit se faire de façon transparente, pour que l'utilisateur n'ait aucun effort cognitif à faire. Comme nous l'avons vu au Chapitre 3, des algorithmes simples donnent des résultats prometteurs pour détecter des permissions suspectes : l'exploration de métriques plus complexes pourrait donner de meilleurs résultats, comme par exemple la proximité sociale entre les sujets ou

des niveaux de sensibilité des documents. Il est également probable que parmi l'ensemble des propriétaires de Cloud personnel, certains aient des politiques similaires ; l'utilisation d'algorithmes distribués de *machine learning* respectueux de la vie privée permettrait de bénéficier de classifications d'utilisateurs aux politiques proches, sans rien dévoiler d'eux.

Architectures sécurisées. Nous avons proposé une architecture de référence pour un Cloud personnel sécurisé par construction. Il reste néanmoins des challenges non résolus, notamment sur la question des environnements sécurisés et isolés. Il serait intéressant de pouvoir les embarquer directement dans des enclaves sécurisées au niveau du processeur, avec des technologiques comme SGX ou TrustZone. Cela permettrait de pouvoir disposer d'instances sécurisées dans des Smartphone, objets connectés ou sur des serveurs centralisés, hébergés par des fournisseurs de Cloud personnel. La gestion de données sur ce type d'environnements est un problème ouvert qui pourrait amener de nombreux travaux intéressants.

De plus, la définition formelle de l'environnement permettrait d'avoir une preuve de la robustesse du code. Cela impose de minimiser le plus possible la taille et la complexité du code et d'identifier précisément le rôle et les actions de chaque module du cœur.

Intéropérabilité. Les solutions décentralisées de Cloud personnel fleurissent, chacune disposant de son architecture propre, avec des modèles de données spécifiques. Ces environnements sont généralement hétérogènes entre eux et ne peuvent communiquer nativement, ce qui réduit considérablement leur attractivité par rapport à des mastodontes centralisés, capables d'acquérir rapidement une masse critique d'utilisateurs. La fédération entre plateformes décentralisées est donc essentielle pour pouvoir espérer un usage social à large échelle.

Un groupe de travail W3C*, Decentralized Sharing Community Group* [DSCG 2015], a été créé en ce sens en avril 2015 avec différents partenaires (Cozy Cloud, ownCloud, Known, IndieHoster, etc). Le but de ce groupe était de trouver un consensus sur des façons de partager des documents entre différentes plateformes hétérogènes. Plusieurs preuves de concepts ont été développées[81] , et une soumission à l'IETF a été faite [de Jong 2015], afin

[81] https://github.com/Decentralized-Sharing-Working-Group/

de proposer un ensemble de bonnes pratiques à respecter pour disposer d'un système de partage interopérable. Une autre initiative, menée notamment par ownCloud, a été de créer un nouveau standard de partage [GEANT 2017], implémentée notamment dans ownCloud, nextCloud et Pydio. Cette approche a permis une fédération entre différents partenaires, mais a le désavantage de créer une nouvelle norme, forcément incompatible avec celles déjà existantes.

Il est donc nécessaire de trouver un consensus, afin que ces plateformes puissent communiquer entre elles et s'assurer un réel avantage compétitif par rapport aux silos centralisés, qui n'ont aucun intérêt commercial à être plus ouverts.

Partage par requêtes distribuées. Les partages sont généralement conditionnés par une connaissance préalable du propriétaire vis-à-vis du contenu et du potentiel intérêt des sujets. Or, il serait intéressant d'être capable d'exécuter des requêtes distribuées respectueuses de la vie privée, afin d'être capable de partager sur la base de calculs portant sur les données d'autres utilisateurs.

Par exemple, une requête distribuée pourrait calculer des similarités entre les *playlist* musicales d'un cercle de sujet plus ou moins vaste, et automatiquement partager des pistes entre utilisateurs ayant les mêmes goûts musicaux. Un propriétaire pourrait alors à la fois diffuser plus largement ses trouvailles et en obtenir lui-même de nouvelles, provenant potentiellement de sujets dont il n'aurait pas soupçonné un si bon goût musical.

Le partage de disponibilité est un autre cas d'usage intéressant : afin de planifier une prochaine réunion, une requête distribuée est exécutée sur les agendas des Cloud personnels de chaque participant. Cette requête calcule l'intersection des différentes disponibilités et ne retourne que les créneaux possibles pour tout le monde, sans dévoiler leurs agendas respectifs.

Il serait également possible de faire un partage anonyme de géolocalisation : un utilisateur accepte de partager sa position, mais uniquement avec ses contacts eux-mêmes situés au même moment dans un rayon inférieur à une limite fixée. La donnée GPS brute et l'identité des utilisateurs ne sont partagées que dans ce cas, avec leur consentement explicite, par exemple pour indiquer dans quel bar des amis géographiquement proches peuvent se retrouver prendre un verre.

Chapitre 7
Bibliographie

Abiteboul, S., André, B., & Kaplan, D. «Managing your digital life.» *Communications of the ACM* 58, n° 5 [2015]: 32-35.

Albers, M. «One Last Away Message.» *tumblr.com.* 06 Octobre 2017. https://aimemories.tumblr.com/.

Ali, M., Dhamotharan, R., Khan, E., Khan, S. U., Vasilakos, A. V., Li, K., et Zomaya, A. Y. «SeDaSC: secure data sharing in clouds.» *IEEE Systems Journal* 11, n° 2 [2017]: 395-404.

Al-Kahtani, M., & Sandhu, R. «A model for attribute-based user-role assignment.» *Computer Security Applications Conference.* IEEE, 2002. 353-362.

—. «Induced role hierarchies with attribute-based RBAC.» *ACM symposium on Access control models and technologies.* ACM, 2003. 142-148.

Alves, T. «Trustzone: Integrated hardware and software security.» White Paper, 2004.

Anciaux, N., Benzine, M., Bouganim, L., Pucheral, P., et Shasha, D. «GhostDB: querying visible and hidden data without leaks.» *SIGMOD international conference on Management of data.* ACM, 2007. 677-688.

Anciaux, N., Bouganim, L., Nquyen, B., Popa, I. S., Pucheral, P., & Bonnet, P. «Trusted cells: A sea change for personal data services.» *CIDR*, 2013.

Anciaux, N., Bouganim, L., Pucheral, P., Guo, Y., Le Folgoc, L., et Yin, S. «MILo-DB: a personal, secure and portable database machine.» *Distributed and Parallel Databases* 32, n° 1 [2014]: 37-63.

Anciaux, N., Lallali, S., Sandu Popa, I.,& Pucheral P. «A Scalable Search Engine for Mass Storage Smart Objects.» *PVLDB* 8, n° 9 [2015]: 910-921.

Anderson, A. *A comparison of two privacy policy languages: EPAL and XACML.* Sun Microsystems, 2005.

Anderson, A. *Core and hierarchical role based access control (RBAC) profile of XACML v2. 0.* OASIS Standard, 2005.

Andre, B., Anciaux, N., Pucheral, P., & Tran-Van, P. «A Root of Trust for the Personal Cloud.» *ERCIM News*, Juillet 2016.

Apple. «iCloud Terms and Conditions.» *apple.com.* 19 Septembre 2017. https://www.apple.com/ca/legal/internet-services/icloud/en/terms.html.

Ashley, P., Hada, S., Karjoth, G., Powers, C., et Schunter, M. *Enterprise privacy authorization language (EPAL).* IBM, 2003.

Baden, R., Bender, A., Spring, N., Bhattacharjee, B., et Starin, D. «Persona: an online social network with user-defined privacy.» *ACM SIGCOMM Computer Communication Review* 39, n° 4 [2009]: 135-146.

Bayer, R. «Binary B-trees for virtual memory.» *SIGFIDET (now SIGMOD) Workshop on Data Description.* ACM, 1971. 219-235.

Bell, D.E. et LaPadula, L.J. *Secure computer systems: Unified exposition and multics interpretation (Technical Report ESD-TR-73-306).* The MITRE Corporation, 1976.

Bellavista, P., Giannelli, C., Iannario, L., Goix, L. W., et Venezia, C. «Peer-to-peer content sharing based on social identities and relationships.» *IEEE Internet Computing* 18, n° 3 [2014]: 55-63.

Bertino, E. & Sandhu, R. «Database security-concepts, approaches, and challenges.» *IEEE Transactions on Dependable and secure computing* 2, n° 1 [2005]: 2-19.

Bertino, E., Gabriel G., & Ashish K. «Access control for databases: Concepts and systems.» *Foundations and Trends® in Databases*, 2011: 1-148.

Bertino, E., Pierangela S., & Sushil J. «Authorizations in relational database management systems.» *Computer and communications security.* ACM, 1993. 130-139.

Bickert, M. & Fishman, B. *Newsroom.* 28 Novembre 2017. https://newsroom.fb.com/news/2017/11/hard-questions-are-we-winning-the-war-on-terrorism-online/.

Bloom, B. H. «Space/time trade-offs in hash coding with allowable errors.» *Communications of the ACM* [ACM] 13, n° 7 [1970]: 422-426.

Bocchi, E., Drago, I., et Mellia, M. «Personal cloud storage: Usage, performance and impact of terminals.» *Cloud Networking (CloudNet).* IEEE, 2015. 106-111.

Bond, R. M., Fariss, C. J., Jones, J. J., Kramer, A. D., Marlow, C., Settle, J. E., & Fowler, J. H. «A 61-million-person experiment in social influence and political mobilization.» *Nature* 489, n° 7415 [2012]: 295-298.

Boneh, D., Lewi, K., Montgomery, H., et Raghunathan, A. «Key homomorphic PRFs and their applications. .» *Advances in Cryptology–CRYPTO*, 2013: 410-428.

Borisov, N., Goldberg, I., & Brewer, E. «Off-the-record communication, or, why not to use PGP.» *Privacy in the electronic society.* ACM, 2004. 77-84.

Botsman, R. «Big data meets Big Brother as China moves to rate its citizens.» *Wired.* 21 Octobre 2017. http://www.wired.co.uk/article/chinese-government-social-credit-score-privacy-invasion.

Bouganim, L., Jónsson, B., & Bonnet, P. «uFLIP: Understanding flash IO patterns.» *The Biennial Conference on Innovative Data systems Research (CIDR).* 2009.

Cain, P. «Where 1,296 gay Ashley Madison users face prison, flogging, execution.» *GlobalNews.* 02 Septembre 2015. https://globalnews.ca/news/2186587/where-1296-gay-ashley-madison-users-face-prison-flogging-execution/.

Canard, S., et Devigne, J. «Highly privacy-protecting data sharing in a tree structure.» *Future Generation Computer Systems*, 2016: 119-127.

Carminati, B., Ferrari, E., et Perego, A. «Rule-based access control for social networks.» *On the Move to Meaningful Internet Systems.* Springer, 2006. 1734-1744.

Cavoukian, A. «Privacy by design. Take the challenge. .» *nformation and privacy commissioner of Ontario*, 2009.

Chard, K., Bubendorfer, K., Caton, S., et Rana, O. F. «Social cloud computing: A vision for socially motivated resource sharing.» *IEEE Transactions on Services Computing* 5, n° 4 [2012]: 551-563.

Chelet, J. «Baidu, Alibaba, Tencent et Xiaomi : ces incroyables «GAFA» chinois.» *Capital.fr.* 17 Juin 2016. https://www.capital.fr/entreprises-marches/baidu-alibaba-tencent-et-xiaomi-ces-incroyables-gafa-chinois-1139115.

Cherdantseva, Yulia, et Jeremy Hilton. «A reference model of information assurance & security.» *Availability, Reliability and Security (ARES).* IEEE, 2013. 546-555.

Chirgwin, R. «Sweden leaked every car owners' details last year, then tried to hush it up.» *TheRegister.* 23 Juillet 2017. https://www.theregister.co.uk/2017/07/23/sweden_leaked_every_car_owners_details_last_year_then_tried_to_hush_it_up/.

Christl, W. «Corporate Surveillance in Everyday Life.» *Cracked Labs.* Juin 2017. http://crackedlabs.org/en/corporate-surveillance.

Clinton. *Hillary Clinton's Emails.* 2016. https://www.kaggle.com/kaggle/hillary-clinton-emails.

CNIL. 26 07 2016. https://www.cnil.fr/fr/windows-10-la-cnil-met-publiquement-en-demeure-microsoft-corporation-de-se-conformer-dans-un-delai.

Cohen-Grillet, P. «La CIA appelée au secours par l'antiterrorisme français.» *Paris Match.* 07 Decembre 2016. http://www.parismatch.com/Actu/International/La-CIA-appelee-au-secours-par-l-antiterrorisme-francais-1138268.

Cohn-Gordon, K., Cremers, C., Dowling, B., Garratt, L., & Stebila, D. «A Formal Security Analysis of the Signal Messaging Protocol.» *IACR Cryptology ePrint Archive*, 2016: 1013.

Costan, V., et Devadas, S. «Intel SGX Explained.» *IACR Cryptology ePrint Archive*, 2016: 86.

«CouchDB Replication Protocol.» 2017. http://docs.couchdb.org/en/2.0.0/replication/protocol.html.

Cutillo, L. A., Molva, R., et Strufe, T. «Safebook: A privacy-preserving online social network leveraging on real-life trust.» *IEEE Communications Magazine* 47, n° 12 [2009].

De Choudhury, M., Mason, W. A., Hofman, J. M., & Watts, D. J. «Inferring relevant social networks from interpersonal communication.» *World wide web.* ACM, 2010. 301-310.

de Jong, M., & Tran-Van, P. «Decentralized Sharing.» *IETF.* 29 Mai 2015. https://tools.ietf.org/html/draft-dejong-decentralized-sharing-00.

de Montjoye, Y. A., Wang, S. S., Pentland, A., Anh, D. T. T., & Datta, A. «On the Trusted Use of Large-Scale Personal Data.» *IEEE Data Engineering* 35, n° 4 [2012]: 5-8.

Diuk, C. «The Formation of Love.» *facebook.* 14 02 2014. https://www.facebook.com/notes/facebook-data-science/the-formation-of-love/10152064609253859/?fref=mentions.

Dropbox. «Politique de confidentialité de Dropbox.» *dropbox.com.* 12 Avril 2017. https://www.dropbox.com/fr/privacy.

DSCG. *Decentralized Sharing Community Group.* 2015. https://www.w3.org/community/decsharing/.

Eichner, M. *Spring cleaning in Summer.* 03 Juillet 2012. https://googleblog.blogspot.fr/2012/07/spring-cleaning-in-summer.html.

Elton, B. *Blind Faith.* Transworld Publisher, 2008.

Entous, A., Timberg, C., Dwoskin, E. «Russian operatives used Facebook ads to exploit America's racial and religious divisions.» *The Washington Post.* 25 Septembre 2017. https://www.washingtonpost.com/business/technology/russian-operatives-used-facebook-ads-to-exploit-divisions-over-black-political-activism-and-muslims/2017/09/25/4a011242-a21b-11e7-ade1-76d061d56efa_story.html?utm_term=.2ff8b5bfc824.

Ermoshina, K., Musiani, F., & Halpin, H. «End-to-end encrypted messaging protocols: An overview.» *International Conference on Internet Science.* Springer, 2016. 244-254.

EU. *GDPR.* 2016. https://www.eugdpr.org/.

Facebook. 2014. https://www.facebook.com/data/posts/10152217010993415.

Falkvinge, R. "Worst known governmental leak ever is slowly coming to light: Agency moved nation's secret data to "The Cloud"." *PrivacyNewsOnline.* Juillet 21, 2017. https://www.privateinternetaccess.com/blog/2017/07/swedish-transport-agency-worst-known-governmental-leak-ever-is-slowly-coming-to-light/.

Fang, L., et LeFevre, K. «Privacy wizards for social networking sites.» *World wide web.* ACM, 2010. 351-360.

Ferraiolo, D. F., Sandhu, R., Gavrila, S., Kuhn, D. R., et Chandramouli, R. «Proposed NIST standard for role-based access control.» *ACM Transactions on Information and System Security (TISSEC)*, 2001: 224-274.

Fielding, T. *Architectural Styles and the Design of Network-based Software Architectures.* PhD Thesis, University of California, 2000.

Fitzpatrick, B. *Distributed Identity: Yadis.* 2005. https://web.archive.org/web/20060504054201/http://community.livejournal.com/lj_dev/683939.html.

Friggeri, A. «When Love Goes Awry.» *facebook.* 15 02 2014. https://www.facebook.com/notes/facebook-data-science/when-love-goes-awry/10152066701893859/?fref=mentions.

Geambasu, R., Balazinska, M., Gribble, S. D., et Levy, H. M. «Homeviews: peer-to-peer middleware for personal data sharing applications.» *SIGMOD international conference on Management of data.* ACM, 2007. 235-246.

GEANT. «Open Cloud Mesh API.» 2017. https://rawgit.com/GEANT/OCM-API/v1/docs.html.

Gellman, B. «The FBI's Secret Scrutiny.» *Washington Post.* 6 Novembre 2005. http://www.washingtonpost.com/wp-dyn/content/article/2005/11/05/AR2005110501366.html.

Gibbs, S. «Criminal gang arrested for selling Apple users' private data in China .» *The Guardian.* 09 Juin 2017. https://www.theguardian.com/technology/2017/jun/09/apple-employees-arrested-selling-private-user-data-china-criminal.

Gilbert, S., & Lynch, N. «Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services.» *Sigact News.* ACM , 2002. 51-59.

Google. «Bienvenue dans les règles de confidentialité de Google.» *google.com.* 18 Decembre 2017. https://www.google.com/policies/privacy/.

Greeven, M., & Wei, W. «The rise of new technology giants from China.» *chinadaily.com.cn.* 10 Octobre 2017. http://www.chinadaily.com.cn/opinion/2017-10/10/content_33071873.htm.

Griffiths, Patricia P., et Bradford W. Wade. «An authorization mechanism for a relational database system.» *ACM Transactions on Database Systems (TODS)*, 1976: 242-255.

Guha, S., Tang, K., et Francis, P. «NOYB: Privacy in online social networks.» *Online social networks.* ACM, 2008. 49-54.

Haddadi, H., Howard, H., Chaudhry, A., Crowcroft, J., Madhavapeddy, A., & Mortier, R. «Personal data: thinking inside the box.» *eprint arXiv:1501.04737*, 2015.

Haski, P. «Après les Gafa, les nouveaux maîtres du monde sont les Natu.» *Rue89.* 02 08 2015. https://tempsreel.nouvelobs.com/rue89/20150802.RUE3739/apres-les-gafa-les-nouveaux-maitres-du-monde-sont-les-natu.html.

Heim, P. «Resetting passwords to keep your files safe.» *Dropbox.* 25 Août 2016. https://blogs.dropbox.com/dropbox/2016/08/resetting-passwords-to-keep-your-files-safe/.

Hermann, V. «Lycos annonce la fermeture de Caramail pour le 15 février.» *nextinpact.com.* 14 Janvier 2009. https://www.nextinpact.com/archive/48412-lycos-caramail-fermeture.htm.

Hern, A. «Uber employees 'spied on ex-partners, politicians and Beyoncé' .» *The Guardian.* 13 Decembre 2016. Uber employees 'spied on ex-partners, politicians and Beyoncé' .

Hough, A. «Google engineer fired for privacy breach after 'stalking and harrassing teenagers'.» *The Telegraph.* 15 Septembre 2010. http://www.telegraph.co.uk/technology/google/8003925/Google-engineer-fired-for-privacy-breach-after-stalking-and-harrassing-teenagers.html.

Hvistendahl, M. «Inside China's Vast New Experiment in Social Ranking.» *Wired.* 17 Decembre 2017. https://www.wired.com/story/age-of-social-credit/.

IBM. «IBM Informix 12.10.» *ibm.com.* 2017. https://www.ibm.com/support/knowledgecenter/en/SSGU8G_12.1.0/com.ibm.sec.doc/ids_lb_002.htm [accès le 11 03, 2017].

Inkster, T., Story, H., et Harbulot, B. *WebID Authentication over TLS.* 2017. https://dvcs.w3.org/hg/WebID/raw-file/tip/spec/tls-respec.html#.

INSEE. *Comptes nationaux annuels - Révisions des principaux agrégats - année 2016.* 16 Mai 2017. https://www.insee.fr/fr/statistiques/2850642.

«Investigatory Powers Act.» *United Kingdom Parliament.* Novembre 2016. https://publications.parliament.uk/pa/bills/lbill/2016-2017/0066/17066.pdf.

ISO. «ISO/IEC JTC 1/SC 32.» *iso.org.* 2016. https://www.iso.org/committee/45342/x/catalogue/p/1/u/0/w/0/d/0 [accès le 12 06, 2017].

Johnson, B. «Privacy no longer a social norm, says Facebook founder .» *The Guardian.* 11 Janvier 2010. https://www.theguardian.com/technology/2010/jan/11/facebook-privacy.

Kainda, R., Flechais, I., & Roscoe, A. W. «Security and usability: Analysis and evaluation.» *ARES.* IEEE, 2010. 275-282.

Karlitschek, F. 2016. http://karlitschek.de/2016/04/big-changes-i-am-leaving-owncloud-inc-today/.

Katsouraki, A. *Sharing and Usage Control of Personal Information.* PhD Thesis, Versailles: University of Paris-Saclay, 2016.

Katz, J., Sahai, A., et Waters, B. «Katz, J., Sahai, A., & Waters, B. (2008). Predicate encryption supporting disjunctions, polynomial equations, and inner products.» *Advances in Cryptology–EUROCRYPT*, 2008: 146-162.

Klemperer, P., Liang, Y., Mazurek, M., Sleeper, M., Ur, B., Bauer, L., et al. «Tag, you can see it!: Using tags for access control in photo sharing.» *SIGCHI Conference on Human Factors in Computing Systems.* ACM, 2012. 377-386.

Klimt, B., et Yang, Y. «The enron corpus: A new dataset for email classification research.» *Machine learning: ECML* [Machine learning: ECML], 2004: 217-226.

Knockel, J., McKune, S., & Senft, A. «Baidu's and Don'ts.» *TheCitizenLab.* 23 02 2016. https://citizenlab.ca/2016/02/privacy-security-issues-baidu-browser/.

Kocher, P., Genkin, D., Gruss, D., Haas, W, Hamburg M. et al. «Spectre Attacks: Exploiting Speculative Execution.» Janvier 2017.

Kramer, A. D., Guillory, J. E., & Hancock, J. T. (. «Experimental evidence of massive-scale emotional contagion through social networks.» *National Academy of Sciences* 111, n° 24 [2014]: 8788-8790.

Lallali, S., Anciaux, N., Sandu Popa, I.& Pucheral, P. «A Secure Search Engine for the Personal Cloud. .» *SIGMOD.* 2015. 1445-1450.

Le, T. B. T., Anciaux, N., Gilloton, S., Lallali, S., Pucheral, P., Popa, I. S., & Chen, C. «Distributed Secure Search in the Personal Cloud.» *International Conference on Extending Database Technology (EDBT).* 2016. 652-655.

Liao, S. «Google admits it tracked user location data even when the setting was turned off.» *The Verge.* 21 Novembre 2017. https://www.theverge.com/2017/11/21/16684818/google-location-tracking-cell-tower-data-android-os-firebase-privacy.

Lipp, M., Schwarz, M., Gruss, D., Prescher T., Haas W., et al. «Meltdown.» 2017.

Liu, Y., Gummadi, K. P., Krishnamurthy, B., & Mislove, A. «Analyzing facebook privacy settings: user expectations vs. reality.» *SIGCOMM conference on Internet measurement.* ACM, 2011. 61-70.

«LOI n° 2015-912.» *Journal Officiel.* 26 Juillet 2015. https://www.legifrance.gouv.fr/affichTexte.do?cidTexte=JORFTEXT000030931899&categorieLien=id.

Marr, B. «Big Data: 20 Mind-Boggling Facts Everyone Must Read.» *Forbes.* 30 09 2015. https://www.forbes.com/sites/bernardmarr/2015/09/30/big-data-20-mind-boggling-facts-everyone-must-read/#726ab49417b1.

Martins., V. *Data Replication in P2P Systems.* PhD Thesis, Université de Nantes, Faculté des Sciences et Techniques, 2007.

Mazurek, M. L., Arsenault, J. P., Bresee, J., Gupta, N., Ion, I., Johns, C., Lee, D., Liang Y., Olsen J., Salmon B., Shay R., Vaniea K., Bauer L., Faith Cranor L., Ganger, G.R., et Reiter, M.K. «Access control for home data sharing: Attitudes, needs and practices. .» *SIGCHI Conference on Human Factors in Computing Systems .* ACM, 2010. 645-654.

Mazurek, M. L., Liang, Y., Melicher, W., Sleeper, M., Bauer, L., Ganger, G. R., Gupta, N., et Reiter, M. K. «Toward strong, usable access control for shared distributed data.» *File and Storage Technologies .* USENIX, 2014. 89-103.

McGoogan, C. *Why your smart TV is the perfect way to spy on you .* 8 Mars 2017. http://www.telegraph.co.uk/technology/2017/03/08/smart-tv-perfect-way-spy/.

Microsoft. «Contrat de Services Microsoft.» *microsoft.com.* 15 Juillet 2016. https://www.microsoft.com/fr-fr/servicesagreement/.

MoneyControl. «WeChat confirms that it makes all private user data available to the Chinese government.» *MoneyControl.* 19 07 2017. http://www.moneycontrol.com/news/business/companies/wechat-confirms-that-it-makes-all-private-user-data-available-to-the-chinese-government-2391847.html.

Nakamoto, S. «Bitcoin: A Peer-to-Peer Electronic Cash System.» White paper, 2008.

Navarro, G. «A guided tour to approximate string matching.» *ACM computing surveys (CSUR)* 33, n° 1 [2001]: 31-88.

Nechvatal, J., Barker, E., Bassham, L., Burr, W., Dworkin, M., Foti, J., et Roback, E. *Report on the Development of the Advanced Encryption Standard (AES).* National Institute of Standards and Technology, 2000.

NewCenter. «Challenging the Titans of Technology.» *newcenter.org.* 2017. http://newcenter.org/ideas-to-re-center-america/challenging-big-tech/ [accès le Decembre 02, 2017].

Newman, L. H. «Yahoo's 2013 Email Hack Actually Compromised Three Billion Accounts.» *Wired.* 10 Mars 2017. https://www.wired.com/story/yahoo-breach-three-billion-accounts/.

Ng, H. W., Nguyen, V. D., Vonikakis, V., & Winkler, S. «Deep learning for emotion recognition on small datasets using transfer learning.» *International conference on multimodal interaction.* ACM, 2015. 443-449.

Nielsen, J. *Usability Engineering.* Elsevier, 1993.

Oracle. «Porting Guide for Release 6.» *docs.oracle.com.* 2017. https://docs.oracle.com/cd/E37670_01/E52461/html/section_jsf_zpm_wm.html [accès le 11 06, 2017].

Orwell, G. *1984.* Gallimard, 1949.

Osborn, S., Sandhu, R., & Munawer, Q. «Configuring role-based access control to enforce mandatory and discretionary access control policies.» *ACM Transactions on Information and System Security (TISSEC)*, 2000: 85-106.

Park, J., & Sandhu, R. «The UCON ABC usage control model.» *Transactions on Information and System Security (TISSEC)* 7, n° 1 [2004]: 128-174.

Perez, S. «117 million LinkedIn emails and passwords from a 2012 hack just got posted online.» *TechCrunch.* 18 Mai 2016. https://techcrunch.com/2016/05/18/117-million-linkedin-emails-and-passwords-from-a-2012-hack-just-got-posted-online/.

—. «Recently confirmed Myspace hack could be the largest yet.» *TechCrunch.* 31 Mai 2016. https://techcrunch.com/2016/05/31/recently-confirmed-myspace-hack-could-be-the-largest-yet/.

Puel, H. «Microsoft clôt le chapitre Encarta.» *01net.com.* 31 Mars 2009. http://www.01net.com/actualites/microsoft-clot-le-chapitre-encarta-500525.html.

Rauhala, E. *https://twitter.com/.* 03 Janvier 2018. https://twitter.com/emilyrauhala/status/948499438141628416 [accès le Janvier 04, 2018].

Recordon, D., et Reed, D. «OpenID 2.0: a platform for user-centric identity management.» *Digital identity management.* ACM, 2006. 11-16.

Reed, D., & Strongin, G. *The Dataweb: An Introduction to XDI.* White Paper, OASIS, 2004.

Riva, O., Yin, Q., Juric, D., Ucan, E., et Roscoe, T. «Policy expressivity in the Anzere personal cloud. .» *Symposium on Cloud Computing.* ACM, 2011.

Rosen, G. *Newsroom.* 27 Novembre 2017. https://newsroom.fb.com/news/2017/11/getting-our-community-help-in-real-time/.

Roth, M., Ben-David, A., Deutscher, D., Flysher, G., Horn, I., Leichtberg, A., et al. «Suggesting friends using the implicit social graph.» *SIGKDD international conference on Knowledge discovery and data mining.* ACM, 2010. 233-242.

Rudder, C. «We Experiment On Human Beings!» *OkCupid.* 27 Juillet 2014. https://theblog.okcupid.com/we-experiment-on-human-beings-5dd9fe280cd5.

Sahai, A., et Waters, B. «Fuzzy identity-based encryption.» *Eurocrypt*, 2005: 457-473.

Saint-Andre, P. *Extensible Messaging and Presence Protocol (XMPP): Core.* RFC 6120, IETF, 2011.

Sandhu, R. S., Coyne, E. J., Feinstein, H. L., & Youman, C. E. «Role-based access control models.» *Computer*, 1996: 38-47.

Sandhu, R. S., Ferraiolo, D. et Kuhn, R. «ANSI Standard - Role Based Access Control.» 2004. http://profsandhu.com/journals/tissec/ANSI+INCITS+359-2004.pdf.

Sandhu, R.S., & Munawer, Q. «How to do discretionary access control using roles.» *ACM workshop on Role-based access control.* 1998. 47-54.

Sandhu, R.S., & Pierangela S. «Access control: principle and practice.» *IEEE communications magazine*, 1994: 40-48.

Schmidt, Eric, interviewer par CNBC. *Inside the Mind of Google* [02 Decembre 2009].

Schneier, B. *Computer Security: Will We Ever Learn?* 2000. https://www.schneier.com/crypto-gram/archives/2000/0515.html.

Segall, L. «Pastor outed on Ashley Madison commits suicide.» *CNN.* 08 Septembre 2015. http://money.cnn.com/2015/09/08/technology/ashley-madison-suicide/index.html.

Seltzer, M. I., et Murphy, N. «Hierarchical File Systems Are Dead.» *HotOS*, 2009.

Seong, S. W., Seo, J., Nasielski, M., Sengupta, D., Hangal, S., Teh, S. K., Chu, R., Dodson, B., et Lam, M.S. «PrPl: a decentralized social networking infrastructure.» *Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond.* ACM, 2010.

Shen, H. B., et Hong, F. « An attribute-based access control model for web services.» *Parallel and Distributed Computing, Applications and Technologies.* IEEE, 2006. 74-79.

Citizenfour. Réalisé par L. Poitras. Interprété par E, Greenwald, G., & Binney W. Snowden. 2014.

Son, S. H. «Replicated data management in distributed database systems.» *ACM SIGMOD Record* 17, n° 4 [1988]: 62-69.

Squicciarini, A. C., Sundareswaran, S., Lin, D., et Wede, J. «A3p: adaptive policy prediction for shared images over popular content sharing sites.» *Hypertext and hypermedia.* ACM, 2011. 261-270.

Stallman, Richard. 2016. https://www.gnu.org/philosophy/open-source-misses-the-point.html.

Sun, S. T., Hawkey, K., & Beznosov, K. «Systematically breaking and fixing OpenID security: Formal analysis, semi-automated empirical evaluation, and practical countermeasures.» *Computers & Security* 31, n° 4 [2012]: 465-483.

Sundaresan, P. Generalized key indexes. Washington, DC: U.S. Brevet 5,870,747. 1999.

Szoldra, P. «This is everything Edward Snowden revealed in one year of unprecedented top-secret leaks.» *Business Insider.* 06 Septembre 2016. http://www.businessinsider.fr/us/snowden-leaks-timeline-2016-9/.

Tan, P. N. *Introduction to data mining.* 2006.

«The Economist.» *The world's most valuable resource is no longer oil, but data.* 06 05 2017. https://www.economist.com/news/leaders/21721656-data-economy-demands-new-approach-antitrust-rules-worlds-most-valuable-resource.

TheNextWeb. *MyOpenID to shut down.* 2014. https://thenextweb.com/insider/2013/09/04/myopenid-to-shut-down/.

Thilakanathan, D., Chen, S., Nepal, S., et Calvo, R. A. «Secure data sharing in the Cloud.» *Security, Privacy and Trust in Cloud Systems*, 2014: 45-72.

Thirani, V., & Gupta, A. «The value of data.» *World Economic Forum.* 22 Septembre 2017. https://www.weforum.org/agenda/2017/09/the-value-of-data/.

«Timeline.» *google.fr.* s.d. https://www.google.com/maps/timeline?pb.

Titcomb, J. & McGoogan C. «Wikileaks: What has been revealed and can the CIA hack my TV? .» *Telegraph.co.uk.* 7 Mars 2017. http://www.telegraph.co.uk/technology/2017/03/07/wikileaks-vault-7-has-revealed-can-cia-hack-phone/.

Tootoonchian, A., Saroiu, S., Ganjali, Y., et Wolman, A. «Lockr: better privacy for social networks.» *Emerging networking experiments and technologies.* ACM, 2009. 169-180.

TOSDR. *Terms of Service; Didn't Read .* 2017. https://phoenix.tosdr.org/.

Tran-Van, P., Anciaux, N., & Pucheral, P. «A New Sharing Paradigm for the Personal Cloud.» *Trust and Privacy in Digital Business (TrustBus).* Springer, 2017. 180-196.

—. «Partage de documents sécurisé entre Cloud personnels.» *APVP'15 - 6e Atelier sur la Protection de la Vie Privée.* 2015.

—. «Reconciling Privacy and Data Sharing in a Smart and Connected Surrounding.» *To appear in Extending Database Technology (EDBT).* 2018.

—. «SWYSWYK: A New Sharing Paradigm for the Personal Cloud. .» *International Conference on Advanced Data Mining and Applications (ADMA).* Springer, 2017. 839-845.

—. «SWYSWYK: a privacy-by-design paradigm for personal information management systems.» *International Conference on Information Systems Development (ISD).* 2017.

Uber. «Uber's Deleted "Rides of Glory" Blog Post.» *whosdrivingyou.org.* 2014. http://www.whosdrivingyou.org/blog/ubers-deleted-rides-of-glory-blog-post.

Van Delft, B., et Oostdijk, M. «A security analysis of OpenID.» *Policies and Research in Identity Management*, 2010: 73-84.

Van der Sar, E. *MegaUpload Shut Down by the Feds, Founder Arrested.* 19 Janvier 2012. https://torrentfreak.com/megaupload-shut-down-120119/.

Van Kleek, M., Smith, D. A., et Shadbolt, N. « A decentralized architecture for consolidating personal information ecosystems: The WebBox.» *PIM.* 2012.

Wang, F., Mickens, J., Zeldovich, N., et Vaikuntanathan, V. «Sieve: Cryptographically Enforced Access Control for User Data in Untrusted Clouds.» *NSDI*, 2016: 611-626.

WEF. *Rethinking Personal Data: Strengthening Trust.* World Economic Forum, 2012.

Weininger, A. «Efficient execution of joins in a star schema.» *SIGMOD international conference on Management of dat.* ACM, 2002. 542-545.

Wetzel, T. «Houston man charged with child porn possession after Google cyber-tip.» *Khou.* 2014. http://www.khou.com/news/crime/houston-man-charged-with-child-porn-possession-after-google-cyber-tip/412278753.

Wobber, E., Abadi, M., Burrows, M., et Lampson, B. «Authentication in the Taos operating system.» *ACM Transactions on Computer Systems (TOCS)* 12, n° 1 [1994]: 3-32.

Wolfe, B.M. «Uber's 'God View' is Alive and Well, Say Former Employees.» *AppAdvice.* 13 Decembre 2016. https://appadvice.com/post/ubers-god-view/731803.

WSJ. «Two China Tech Titans Wrestle Over User Data.» *TheWallStreetJournal.* 03 Août 2017. https://www.wsj.com/articles/two-china-tech-titans-wrestle-over-user-data-1501757738.

Xu, L., Wu, X., et Zhang, X. «CL-PRE: a certificateless proxy re-encryption scheme for secure data sharing with public cloud.» *Symposium on Information, Computer and Communications Security.* ACM, 2012. 87-88.

Yeung, C. M. A., Kagal, L., Gibbins, N., et Shadbolt, N. «Providing Access Control to Online Photo Albums Based on Tags and Linked Data.» *AAAI Spring Symposium: Social Semantic Web: Where Web 2.0 Meets Web 3.0.* 2009. 9-14.

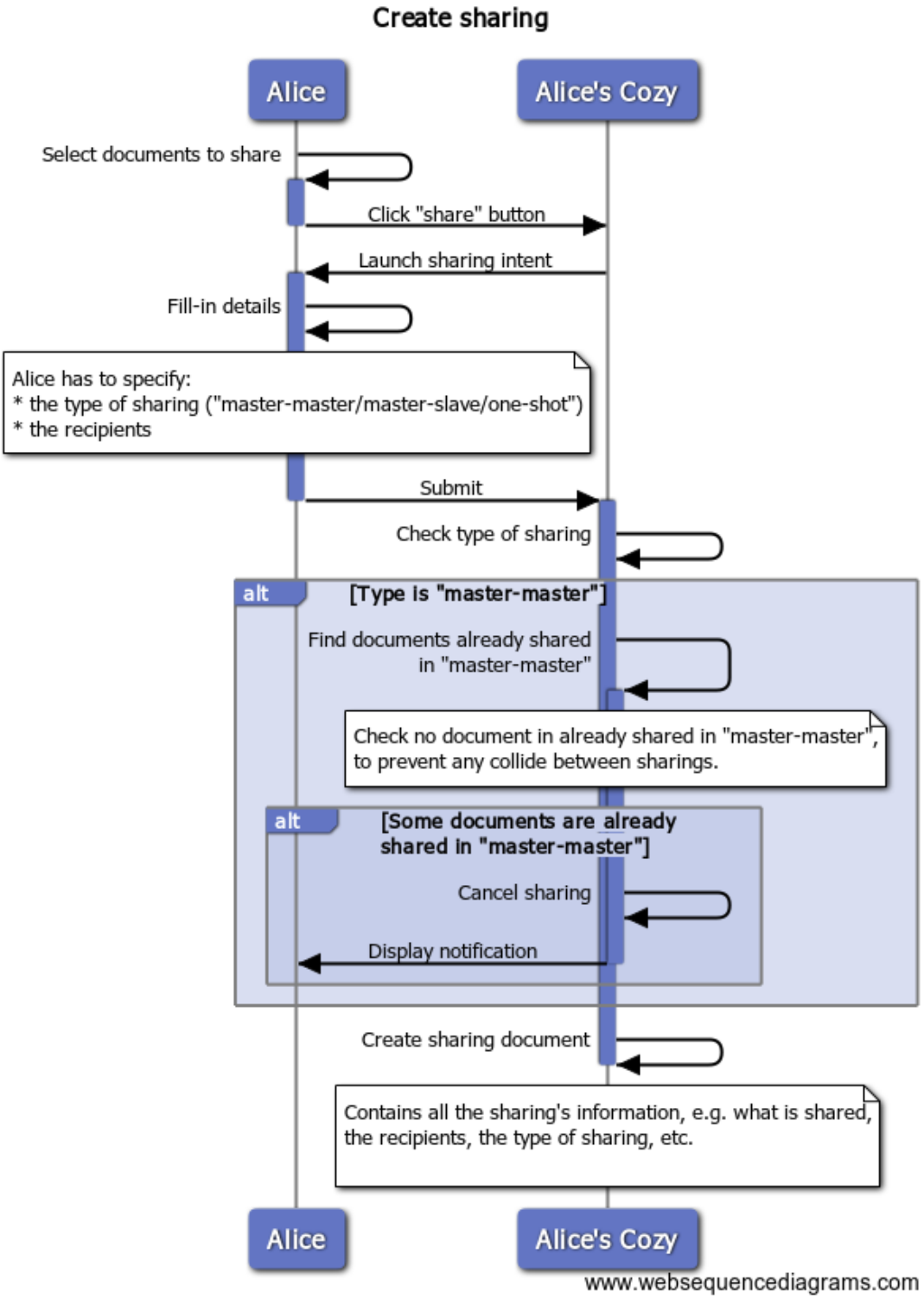
Yuan, E., et Tong, J. «Attributed based access control (ABAC) for web services.» *ICWS.* IEEE, 2005.

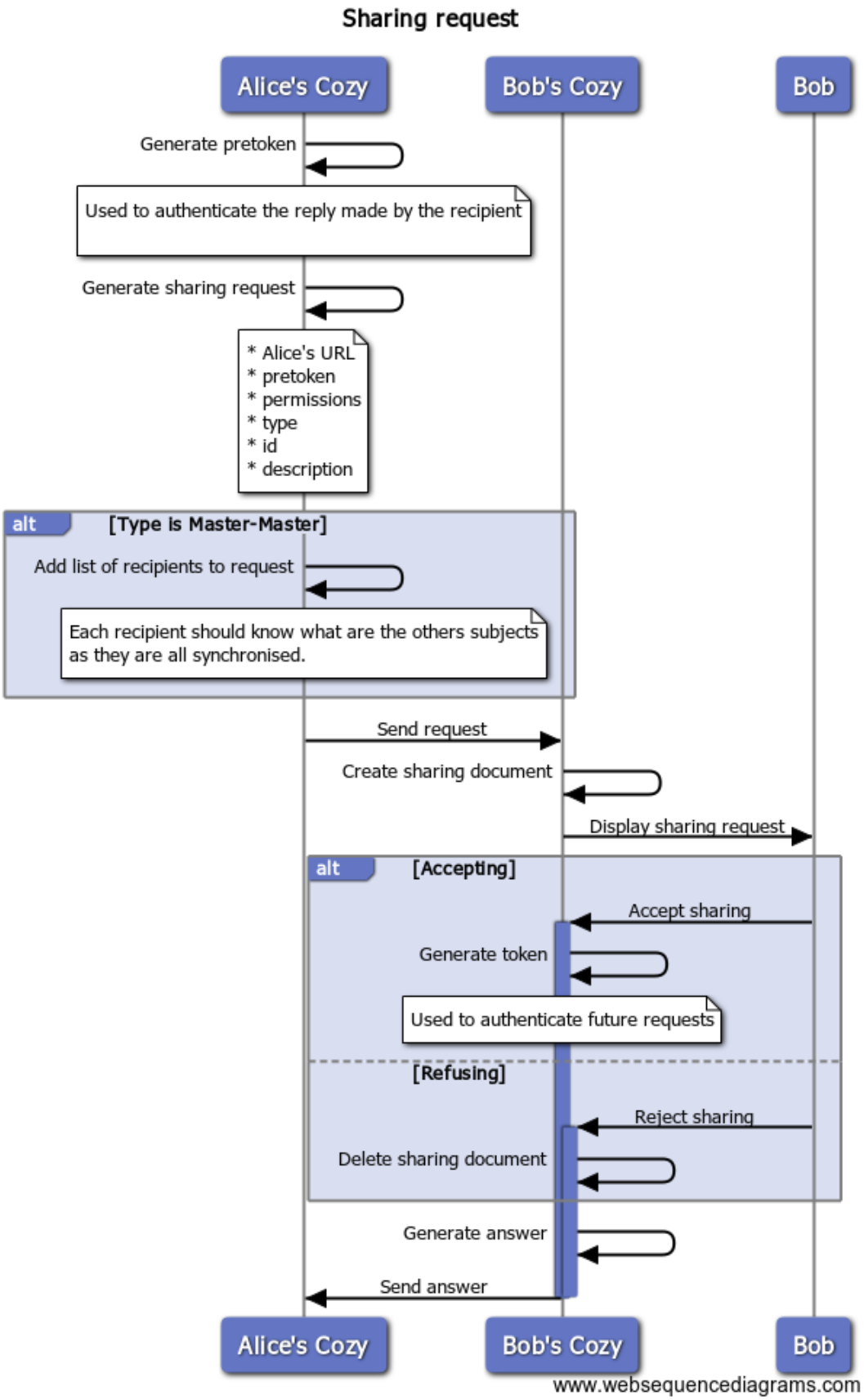
Yuan, L., Korshunov, P., et Ebrahimi, T. «Privacy-preserving photo sharing based on a secure JPEG.» *Computer Communications Workshops (INFOCOM WKSHPS).* IEEE, 2015. 185-190.

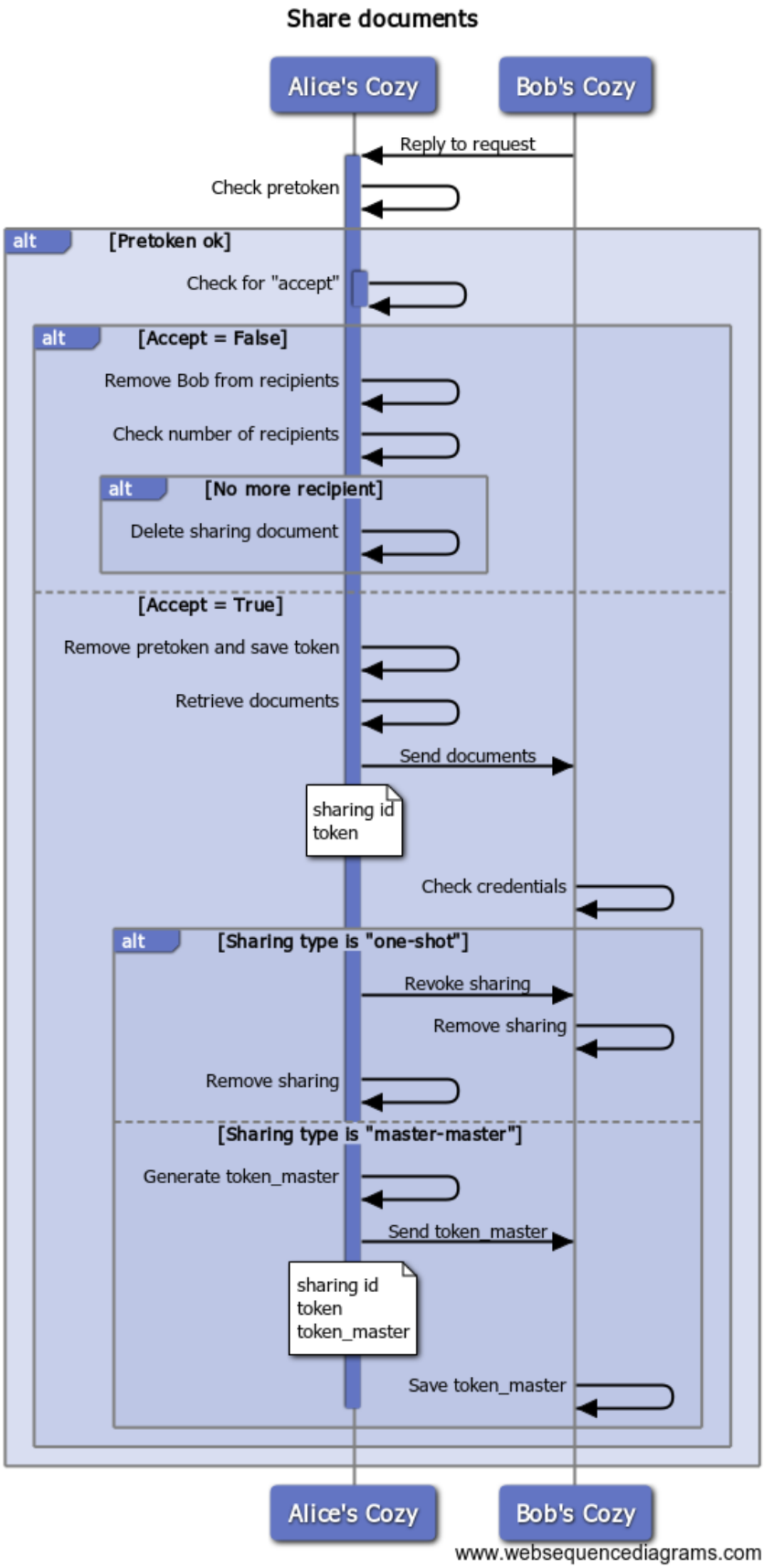
Zaychik Moffitt, V., Stoyanovich, J., Abiteboul, S., et Miklau, G. «Collaborative access control in webdamlog.» *SIGMOD International Conference on Management of Data.* ACM, 2015. 197-211.

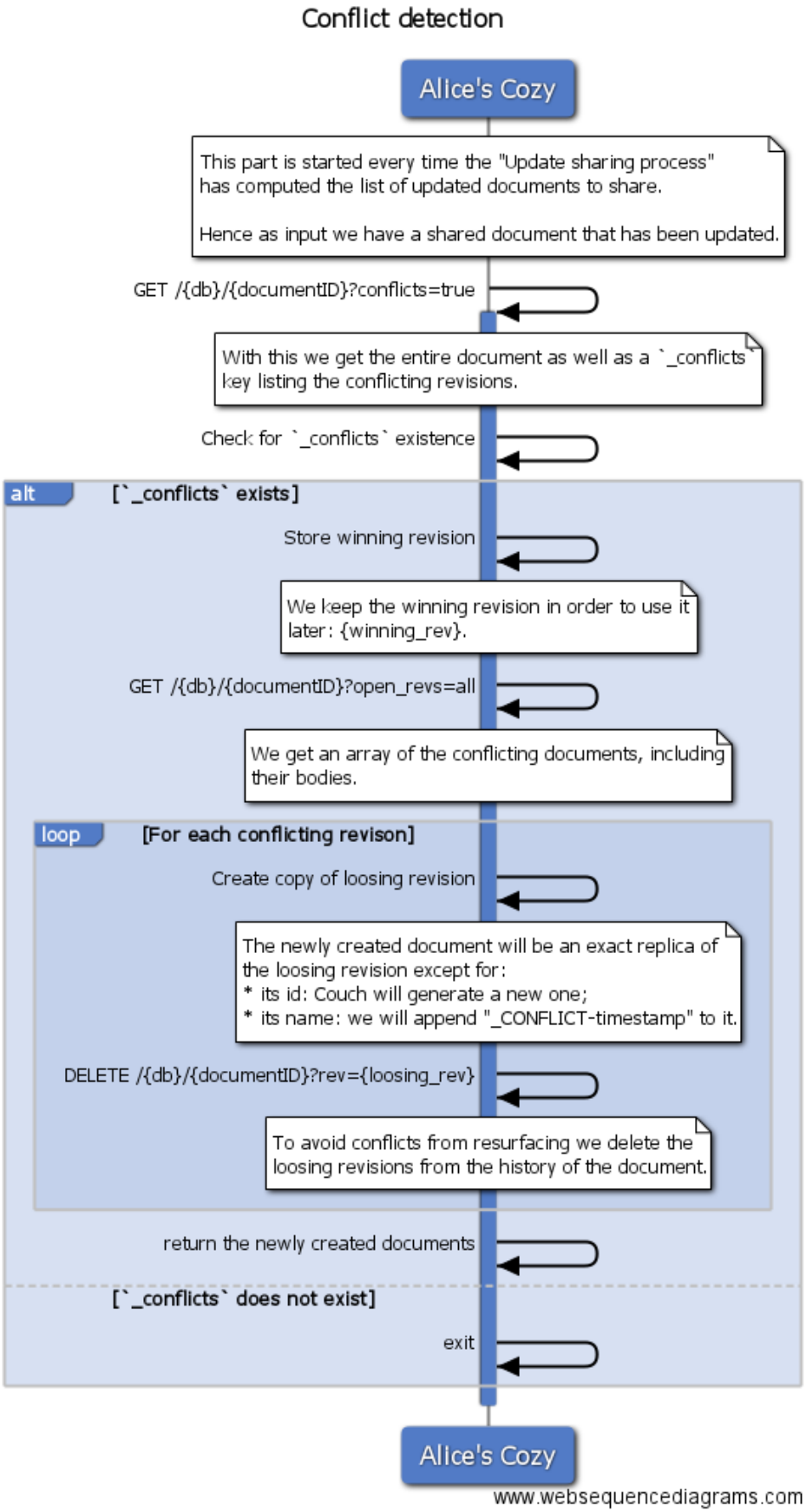
Zimmermann, P. R. «The official PGP user's guide.» *MIT Press*, 1995.

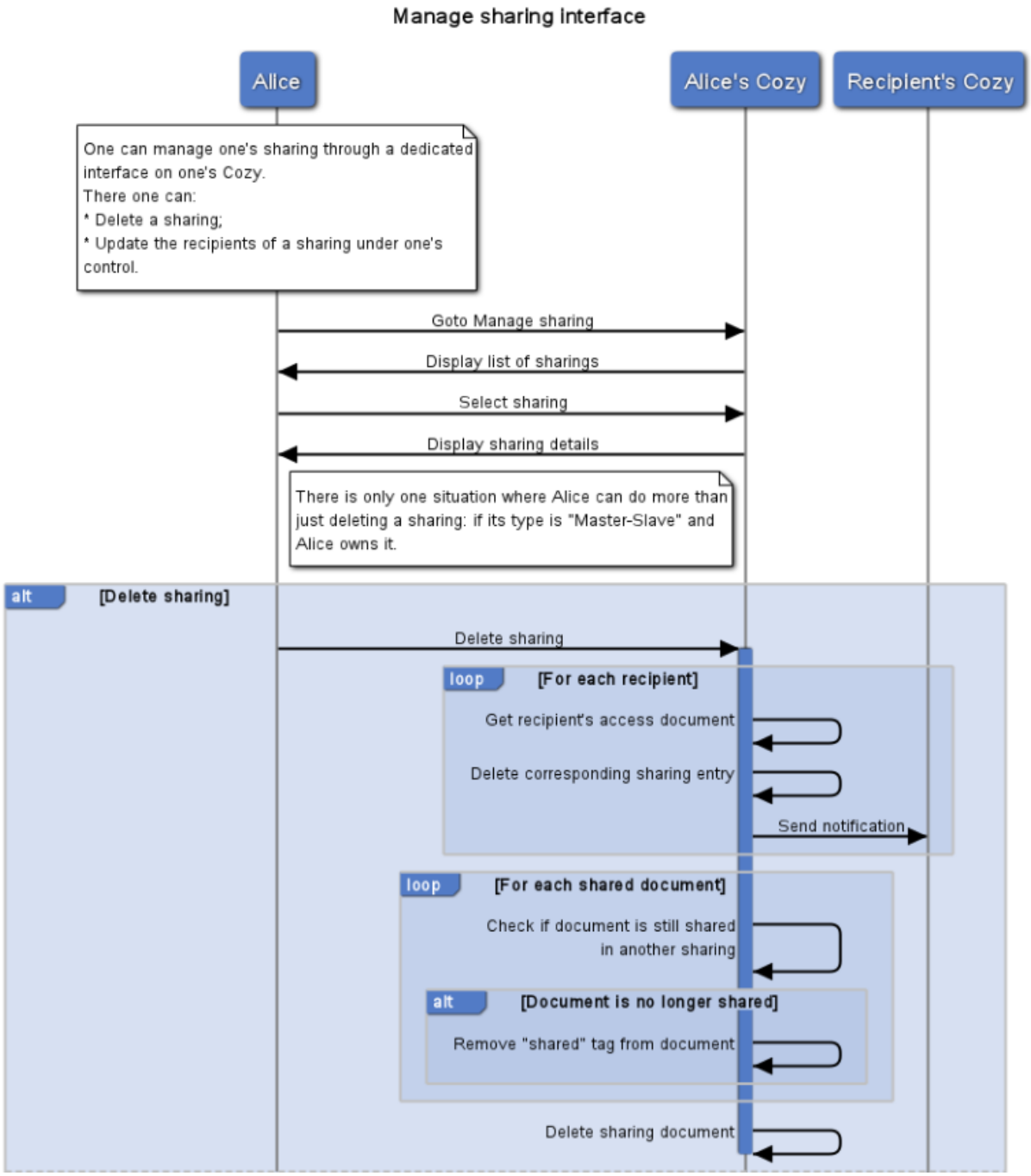
ANNEXES

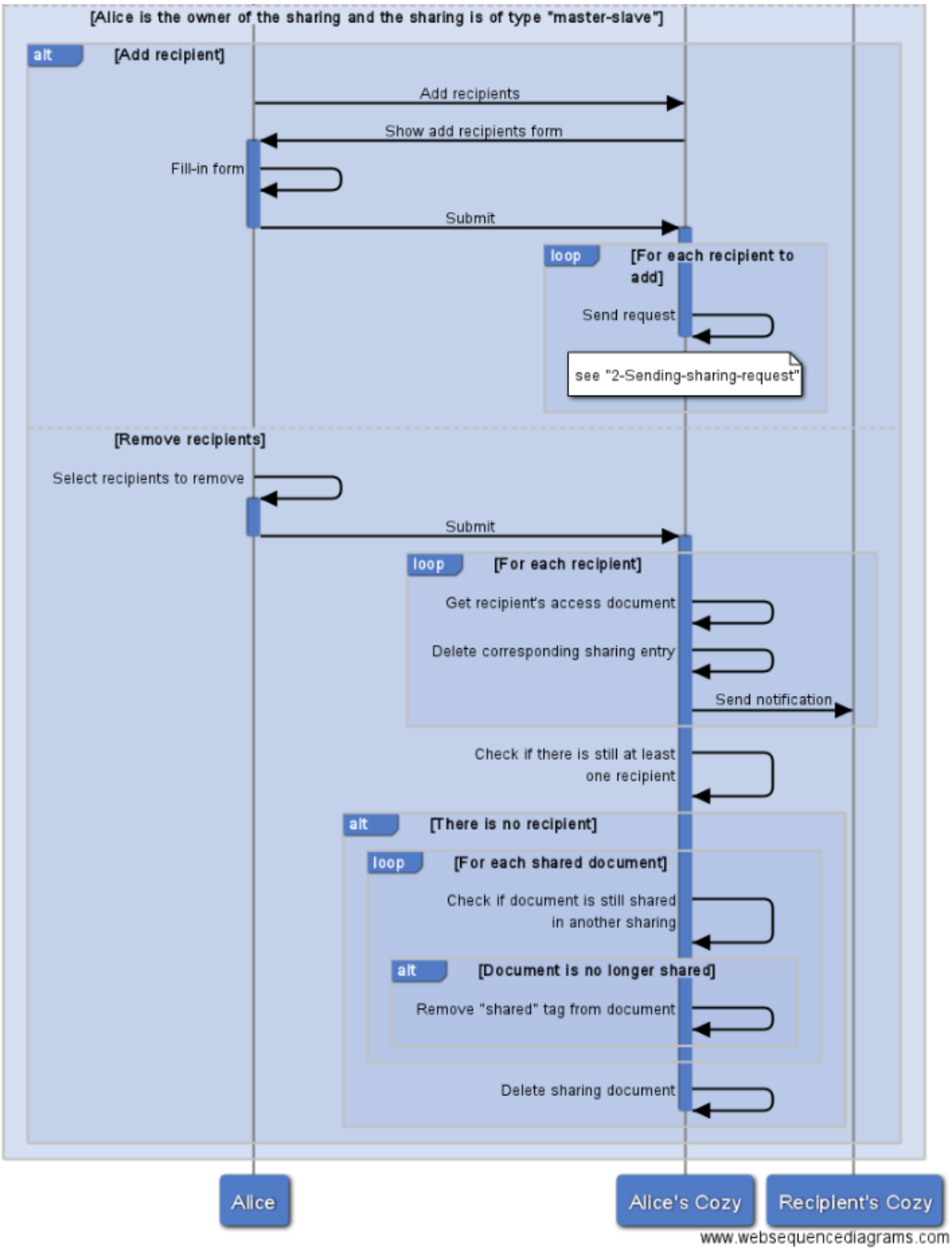

Create sharing
Alice
Alice's Cozy
Select documents to share
Click "share" button
Launch sharing intent
Fill-in details
Alice has to specify:
* the type of sharing ("master-master/master-slave/one-shot")
* the recipients
Submit
Check type of sharing
alt
[Type is "master-master"]
Find documents already shared in "master-master"
Check no document in already shared in "master-master", to prevent any collide between sharings.
alt
[Some documents are already shared in "master-master"]
Cancel sharing
Display notification
Create sharing document
Contains all the sharing's information, e.g. what is shared, the recipients, the type of sharing, etc.
Alice
Alice's Cozy
www.websequencediagrams.com

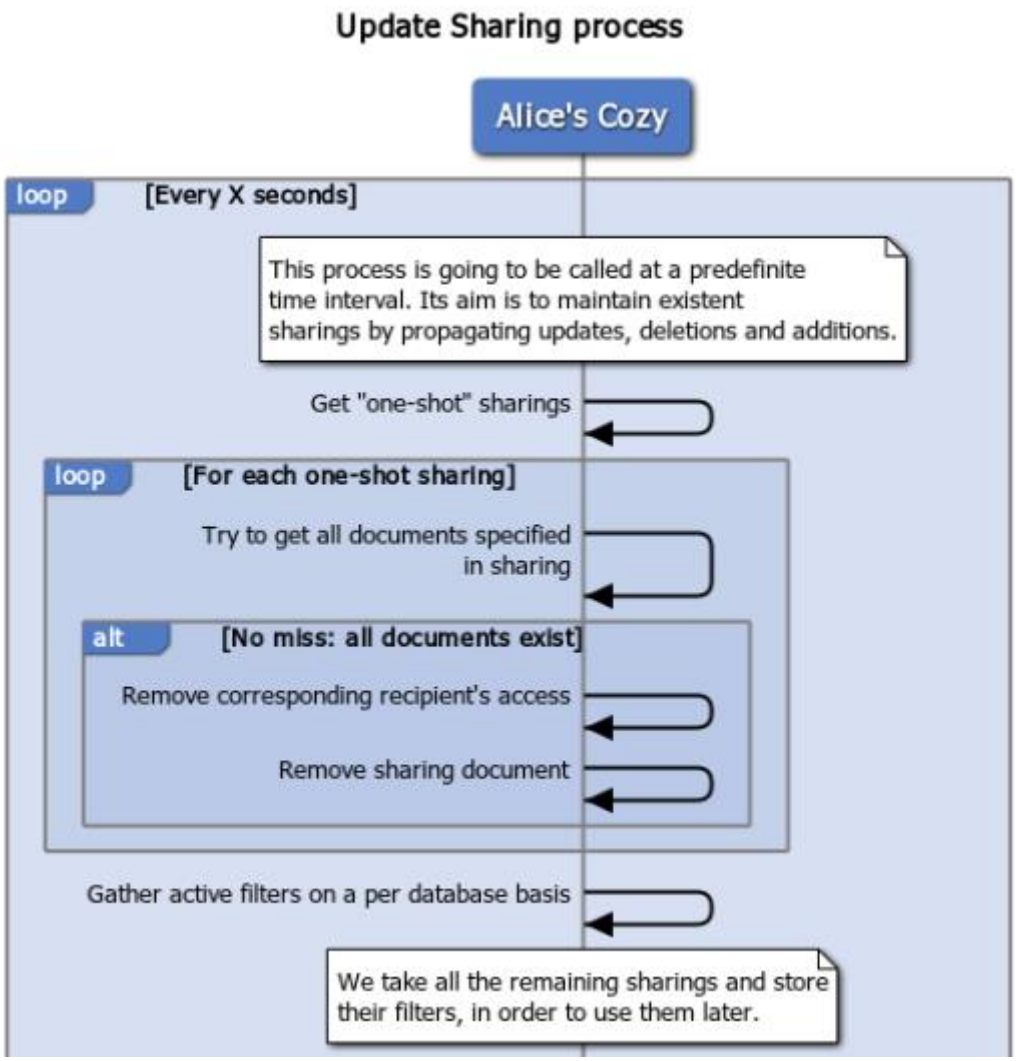

Sharing request
Alice's Cozy
Bob's Cozy
Bob
Generate pretoken
Used to authenticate the reply made by the recipient
Generate sharing request
* Alice's URL
* pretoken
* permissions
* type
* id
* description
alt
[Type is Master-Master]
Add list of recipients to request
Each recipient should know what are the others subjects as they are all synchronised.
Send request
Create sharing document
Display sharing request
alt
[Accepting]
Accept sharing
Generate token
Used to authenticate future requests
[Refusing]
Reject sharing
Delete sharing document
Generate answer
Send answer
Alice's Cozy
Bob's Cozy
Bob
www.websequencediagrams.com

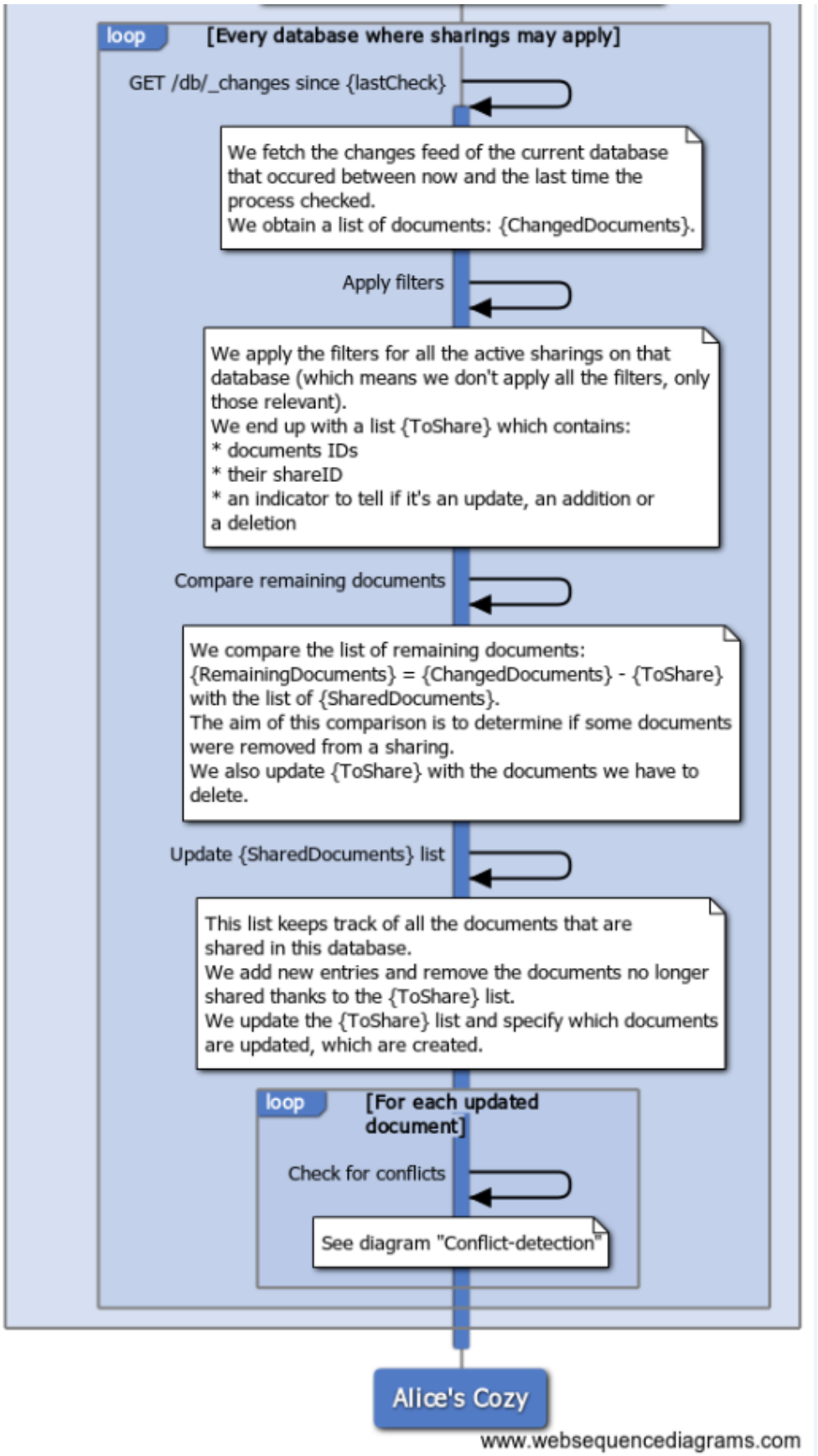

Share documents
Alice's Cozy
Bob's Cozy
Reply to request
Check pretoken
alt
[Pretoken ok]
Check for "accept"
alt
[Accept = False]
Remove Bob from recipients
Check number of recipients
alt
[No more recipient]
Delete sharing document
[Accept = True]
Remove pretoken and save token
Retrieve documents
Send documents
sharing id
token
Check credentials
alt
[Sharing type is "one-shot"]
Revoke sharing
Remove sharing
Remove sharing
[Sharing type is "master-master"]
Generate token_master
Send token_master
sharing id
token
token_master
Save token_master
Alice's Cozy
Bob's Cozy
www.websequencediagrams.com


Conflict detection
Alice's Cozy
This part is started every time the "Update sharing process" has computed the list of updated documents to share.
Hence as input we have a shared document that has been updated.
GET /{db}/{documentID}?conflicts=true
With this we get the entire document as well as a `_conflicts` key listing the conflicting revisions.
Check for `_conflicts` existence
alt
[`_conflicts` exists]
Store winning revision
We keep the winning revision in order to use it later: {winning_rev}.
GET /{db}/{documentID}?open_revs=all
We get an array of the conflicting documents, including their bodies.
loop
[For each conflicting revison]
Create copy of loosing revision
The newly created document will be an exact replica of the loosing revision except for:
* its id: Couch will generate a new one;
* its name: we will append "_CONFLICT-timestamp" to it.
DELETE /{db}/{documentID}?rev={loosing_rev}
To avoid conflicts from resurfacing we delete the loosing revisions from the history of the document.
return the newly created documents
[`_conflicts` does not exist]
exit
Alice's Cozy
www.websequencediagrams.com


Manage sharing interface
Alice
Alice's Cozy
Recipient's Cozy
One can manage one's sharing through a dedicated interface on one's Cozy.
There one can:
* Delete a sharing;
* Update the recipients of a sharing under one's control.
Goto Manage sharing
Display list of sharings
Select sharing
Display sharing details
There is only one situation where Alice can do more than just deleting a sharing: if its type is "Master-Slave" and Alice owns it.
alt
[Delete sharing]
Delete sharing
loop
[For each recipient]
Get recipient's access document
Delete corresponding sharing entry
Send notification
loop
[For each shared document]
Check if document is still shared in another sharing
alt
[Document is no longer shared]
Remove "shared" tag from document
Delete sharing document


[Alice is the owner of the sharing and the sharing is of type "master-slave"]
alt
[Add recipient]
Add recipients
Show add recipients form
Fill-in form
Submit
loop
[For each recipient to add]
Send request
see "2-Sending-sharing-request"
[Remove recipients]
Select recipients to remove
Submit
loop
[For each recipient]
Get recipient's access document
Delete corresponding sharing entry
Send notification
Check if there is still at least one recipient
alt
[There is no recipient]
loop
[For each shared document]
Check if document is still shared in another sharing
alt
[Document is no longer shared]
Remove "shared" tag from document
Delete sharing document
Alice
Alice's Cozy
Recipient's Cozy
www.websequencediagrams.com


Update Sharing process
Alice's Cozy
loop
[Every X seconds]
This process is going to be called at a predefinite time interval. Its aim is to maintain existent sharings by propagating updates, deletions and additions.
Get "one-shot" sharings
loop
[For each one-shot sharing]
Try to get all documents specified in sharing
alt
[No miss: all documents exist]
Remove corresponding recipient's access
Remove sharing document
Gather active filters on a per database basis
We take all the remaining sharings and store their filters, in order to use them later.


loop
[Every database where sharings may apply]
GET /db/_changes since {lastCheck}
We fetch the changes feed of the current database that occured between now and the last time the process checked.
We obtain a list of documents: {ChangedDocuments}.
Apply filters
We apply the filters for all the active sharings on that database (which means we don't apply all the filters, only those relevant).
We end up with a list {ToShare} which contains:
* documents IDs
* their shareID
* an indicator to tell if it's an update, an addition or a deletion
Compare remaining documents
We compare the list of remaining documents: {RemainingDocuments} = {ChangedDocuments} - {ToShare} with the list of {SharedDocuments}.
The aim of this comparison is to determine if some documents were removed from a sharing.
We also update {ToShare} with the documents we have to delete.
Update {SharedDocuments} list
This list keeps track of all the documents that are shared in this database.
We add new entries and remove the documents no longer shared thanks to the {ToShare} list.
We update the {ToShare} list and specify which documents are updated, which are created.
loop
[For each updated document]
Check for conflicts
See diagram "Conflict-detection"
Alice's Cozy
www.websequencediagrams.com

université PARIS-SACLAY | ÉCOLE DOCTORALE Sciences et technologies de l'information et de la communication (STIC)

Titre : Partage de documents sécurisé dans le Cloud Personnel

Mots clés : Partage, Sécurité, Informatique dans les nuages, Contrôle d'Accès, Vie privée

Résumé : Ces dernières années ont été marquées par une numérisation croissante de nos vies, conjuguée à une explosion du volume des données personnelles sur Internet. Cela a favorisé l'émergence d'un marché focalisé sur leur analyse, afin d'établir des profils de plus en plus poussés et intrusifs des individus, généralement à leur insu. Parallèlement, des surveillances d'états se mettent en place qui font craindre un glissement progressif vers des dystopies jusqu'ici réservées à la littérature. Afin de répondre à cette situation, le paradigme du Cloud personnel s'est développé : chaque utilisateur a désormais la possibilité de stocker et gérer l'intégralité de son patrimoine numérique dans un unique espace de confiance dont il est le seul responsable. Cette responsabilisation entraîne cependant un changement de gouvernance sur les données, dont la sécurité et l'administration reposent désormais sur les épaules des individus. En particulier lorsqu'ils souhaitent partager leurs documents et donc les exposer à des personnes ou services tiers. Cette thèse propose ainsi un nouveau paradigme dans la façon de partager dans le Cloud personnel qui met l'accent sur la sécurité, mais aussi sur le contrôle et la simplicité d'utilisation par les individus. Trois contributions sont faites en ce sens : (i) une architecture *Privacy-by-Design*, dédiée au Cloud personnel, (ii) un modèle de partage adapté aux propriétés du Cloud personnel et (iii) un protocole de partage implémenté dans la plateforme Cozy.

Title : Secure document sharing in the Personal Cloud

Keywords : Sharing, Security, Personal Cloud, Access Control, Privacy

Abstract: These past years have witnessed a growing digitalization of our lives, combined with an explosion of personal data quantity on the Internet. This has opened the way to a data-driven market focused on their analysis for profiling purposes, increasingly intrusive and most of the time performed without the user acknowledgement. At the same time, states surveillances are being established, raising concerns about potential dystopias, until now confined in the literature. To tackle this situation, the Personal Cloud paradigm has risen: each user has now the possibility to store and manage all her digital life in a trusted space in which she is the sole responsible. However, this empowerment leads to a governance switch. The user is now in charge of the security and the administration of their data. In particular in the sharing context, where the data is exposed to people or third-parties. Therefore, this thesis proposes a new paradigm in the way the sharing is performed in the Personal Cloud. It focuses on security, control and on a better simplicity of use for the users. Three contributions are made in this direction: (i) a Privacy-by-Design architecture, dedicated for the Personal Cloud context, (ii) a sharing model suited for the Personal Cloud properties and (iii) a sharing protocol implemented in the Cozy platform.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France