Dégradation des données par publication éphémère

Simon Bouget [*] Sébastien Gambs [†] Guillaume Piolle [‡]

Résumé — Le respect de la vie privée est un droit fondamental mais difficile à garantir sur Internet. Malgré les recommandations des experts, le principe de *minimisation des données* — qui exige notamment que les données soient effacées aussitôt qu'elles ne sont plus strictement nécessaires (*rétention minimale*) — est rarement appliqué, en raison d'intérêts divergents entre les hébergeurs des données personnelles et les utilisateurs. Pour diminuer l'impact de ces divergences, une alternative à la rétention minimale a été proposée, la *dégradation des données* : au lieu d'être complètement effacée, une donnée peut être progressivement dégradée vers des états moins précis qui protègent mieux la vie privée de l'utilisateur tout en conservant (partiellement) l'utilité de la donnée initiale. Cependant, aucune architecture n'existe à l'heure actuelle qui garantisse la dégradation des données sans un gestionnaire de base de données de confiance. En utilisant un système de publication éphémère (une donnée ne peut plus être accédée après une date d'expiration fixée à l'avance), nous proposons dans ce travail une architecture décentralisée qui garantit la dégradation des données sans aucun acteur de confiance dans le système.

1 Introduction

Le respect de la vie privée est un droit fondamental mais malheureusement difficile à protéger en ligne. Dans un effort de systématisation, la recherche dans ce domaine a dégagé un grand principe qui, s'il est mis en pratique, permet de limiter les risques pour les utilisateurs : le principe de *minimisation des données*, qui consiste notamment à limiter la collecte et la rétention des données personnelles : un utilisateur ne doit révéler que les données nécessaires pour réaliser le service souhaité et rien de plus et elles ne doivent pas être conservées plus longtemps que la durée nécessaire pour remplir ce service. Pourtant, ce principe est rarement implémenté dans les technologies existantes. En particulier, les délais de rétention sont quasi-systématiquement surévalués, voire non explicitement déclarés, sous prétexte de constituer un profil client, de fournir des services personnalisés plus performants, etc. En effet, c'est grâce à la monétisation des données de leurs utilisateurs (via la publicité par exemple) que la plupart des services en ligne gagnent de l'argent. Il y a donc une contradiction importante entre les intérêts du contrôleur des données, qui cherche à maximiser son profit à partir des données personnelles collectées, et l'utilisateur, qui souhaite le respect de sa vie privée.

Pour minimiser ce conflit, un nouveau mécanisme a été proposé [1] qui est une variante de la suppression exigée par la rétention minimale : la *dégradation des données*. Au lieu d'une situation binaire — soit conservée soit effacée —, une donnée peut exister dans différents états, différents niveaux de précision, qui protègent de manière plus ou moins forte la vie privée de l'utilisateur mais, inversement, permettent de pouvoir réaliser plus ou moins de services.

[*] ENS Cachan - Antenne de Bretagne : simon.bouget@ens-cachan.fr
[†] Université de Rennes 1 / Inria : sebastien.gambs@irisa.fr
[‡] SUPELEC : guillaume.piolle@supelec.fr

Toutefois, à l'heure actuelle, les seuls systèmes de dégradation des données qui existent en pratique sont mis en œuvre par un SGBD [1] spécialement adapté pour cette tâche. L'utilisateur doit donc faire confiance au contrôleur des données et le croire sur parole lorsqu'il déclare appliquer la dégradation.

Étant donné notre motivation de départ — réduire l'impact des intérêts divergents entre utilisateur et contrôleur —, cette hypothèse nous paraît difficile à admettre, et nous proposons ici une architecture qui offre des garanties techniques sur l'application de la dégradation des données, sans nécessiter l'existence d'un acteur de confiance dans le système, et fondée sur un système de publication éphémère décentralisé qui assure la dégradation des données.

Par la suite, nous présentons les travaux existants dans les domaines de la publication éphémère et de la dégradation des données (Section 2), puis nous détaillons l'architecture proposée à partir de ces deux mécanismes (Section 3). Enfin, nous développons un exemple utilisant cette architecture pour illustrer nos propos (Section 4), avant de conclure.

2 État de l'art

Notre travail repose sur la combinaison de deux aspects : la dégradation des données et la publication éphémère, pour lesquels nous détaillons les travaux existant dans ce qui suit.

Les systèmes de publication éphémère permettent de publier des fichiers tout en garantissant qu'ils ne seront plus accessibles au delà d'une date d'expiration T. L'hypothèse importante commune à tous ces systèmes est qu'aucun utilisateur mal intentionné ne doit accéder à un fichier *avant* la date T. Sinon, il peut en faire une copie et rediffuser cette copie ou y accéder après l'expiration, et la publication éphémère perd tout son intérêt. Les systèmes de publication éphémères supposent donc que des mécanismes de contrôle d'accès plus traditionnels sont mis en place pour protéger le fichier avant son expiration.

L'article fondateur du domaine est celui de Radia Perlman en 2005, où elle introduit Ephemerizer [7]. L'idée centrale est que pour avoir des messages éphémères, il suffit de diffuser des messages chiffrés et d'assurer l'effacement des clés à la date d'expiration. Cependant, cette gestion des clés peut être difficile à mettre en place si chaque utilisateur doit gérer ses propres clés et de nombreuses dates d'expiration. Perlman propose donc de concentrer les tâches de gestion dans un service centralisé, nommé *ephemerizer*, ce qui permet d'amortir les coûts sur l'ensemble des utilisateurs et sur un nombre plus important de messages.

Depuis, diverses extensions et améliorations ont été développées pour Ephemerizer : détection puis correction de failles à l'aide de nouveaux schémas de communication [6, 9], granularité plus fine de la date d'expiration [6] et même spécification d'une fenêtre de disponibilité au lieu d'une simple date d'expiration [9]. Toutefois, l'*ephemerizer* est un nœud central sur lequel repose tout le système. Même si ce n'est plus l'hébergeur, c'est encore un acteur auquel il est nécessaire de faire confiance. En conséquence, cela revient seulement à déplacer le problème.

Avec la volonté de supprimer ce besoin de confiance dans une partie cruciale du système, Geambasu, Kohno *et al.* ont cherché à décentraliser Ephemerizer. À partir de 2009, ils ont proposé Vanish [4], où l'*ephemerizer* central est remplacé par une THD [2] et les clés de chiffrement sont stockées en plusieurs fragments sur différents nœuds à l'aide d'un partage de secret à la Shamir [8]. De cette manière, aucun nœud du réseau ne possède seul la capacité d'accéder aux données, et il n'y a plus besoin d'acteur de confiance.

Comme pour Ephemerizer, de nombreuses améliorations ont été proposées pour Vanish. On notera en particulier Cascade [3], un *framework* générique qui permet de répartir les parts du secret dans n'importe quel système de stockage (et même dans plusieurs systèmes simultanément), et plus seulement dans une THD ; Tide [3] et EphPub [2], qui permettent respectivement de stocker les parts dans des serveur Apache et DNS.

En ce qui concerne le principe de dégradation, son intérêt en tant qu'alternative à la simple suppression a été postulé par Anciaux, van Heerde *et al.* à partir de 2008 [1, 12] et principalement exploré dans la thèse

1. Système de Gestion de la Base de Données
2. Table de Hashage Distribuée.

de van Heerde [11], publiée en 2010. Comme eux, nous pensons que la dégradation des données permet d'augmenter la valeur totale[3] d'un ensemble de données personnelles, comparé à la simple suppression. Cependant, la dégradation y est présentée comme une mesure que le contrôleur des données (ici, le gestionnaire de la base) peut mettre en place pour limiter les conséquences d'une fuite involontaire tout en préservant l'utilité des données. De notre point de vue, le problème abordé n'est pas la fuite involontaire des données, mais le manque de confiance entre l'utilisateur et le contrôleur. En conséquence, la mise en place de la dégradation directement par le contrôleur n'est pas satisfaisante.

3 Description du système

Dans cette section, nous allons présenter en détails l'architecture que nous proposons à partir des deux techniques détaillées ci-dessus.

3.1 Acteurs en présence

Nous considérons des systèmes qui mettent en relation deux types d'acteurs : les utilisateurs (ou *sujet*) et les hébergeurs (ou *contrôleur*). Ces systèmes peuvent être de purs systèmes d'hébergement (*e.g.* Dropbox[4]) ou des réseaux sociaux (*e.g.* Facebook[5], Foursquare[6]) qui hébergent les données relatives à l'utilisation du système (liste des pages aimées par l'utilisateur, des lieux visités, etc.).

En réalité, le rôle de l'hébergeur est souvent double : en plus d'être contrôleur des données, il en est aussi le *processeur*, c'est-à-dire celui qui effectue des traitements sur les données pour les valoriser. C'est particulièrement le cas dans les réseaux sociaux, où les données que l'utilisateur publie permettent de lui construire un profil que le réseau social pourra ensuite monétiser (auprès d'une régie publicitaire par exemple). Par la suite, nous prendrons soin de distinguer précisément ces deux rôles de contrôleur et de processeur.

3.2 Architecture proposée

L'interaction entre les différents acteurs, résumée dans la Figure 1 (p. suivante), s'articule de la manière suivante : lorsqu'il souhaite publier une donnée, un utilisateur doit d'abord effectuer localement les étapes de dégradations vers chaque niveau de précision. Il envoie ensuite une copie chiffrée de chaque état à l'hébergeur. Évidemment, chaque état doit être chiffré avec une clé différente puisqu'il a potentiellement sa propre date d'expiration. Les clés de chiffrements sont ensuite stockées dans une structure de données décentralisée — non contrôlée par l'hébergeur —, à une position déterminée à partir de clés de localisation aléatoires. Dans un système à la Vanish, la structure utilisée est une THD, mais il peut également s'agir de serveurs DNS, Apache, etc. Enfin, l'utilisateur envoie à chaque processeur la clé de localisation correspondant au niveau de précision qu'il est autorisé à consulter.

De cette manière, l'hébergeur ne possède que des copies chiffrées, et c'est le système décentralisé — résistant aux collusions et à de nombreux types d'attaques — qui assure la dégradation des données, en oubliant les clés de chiffrement au fur et à mesure que le temps passe. La rétention minimale de chaque niveau de précision est ainsi assurée. Seuls les processeurs explicitement autorisés par l'utilisateur (via l'envoi de la clé de localisation) peuvent accéder aux données, et il peut avoir globalement confiance que la dégradation est bien appliquée

Il faut noter que si l'hébergeur est *uniquement* hébergeur, il n'a plus aucun intérêt à fournir le service. En général, il sera donc aussi un processeur autorisé à un faible niveau de précision, dont la rétention au-delà de la date d'expiration ne prêtera pas à conséquence au cas où l'hébergeur serait malveillant.

3.3 Détail des sous-systèmes

L'architecture proposée est constitué de deux sous-ensembles : le système d'hébergement proprement dit et le système de gestion des clés. Dans cette section nous allons détailler leur fonctionnement.

3. Valeur totale au sens de bien commun, valeur cumulée pour l'utilisateur et le contrôleur.
4. `https://www.dropbox.com/`
5. `https://www.facebook.com/`
6. `https://foursquare.com/`

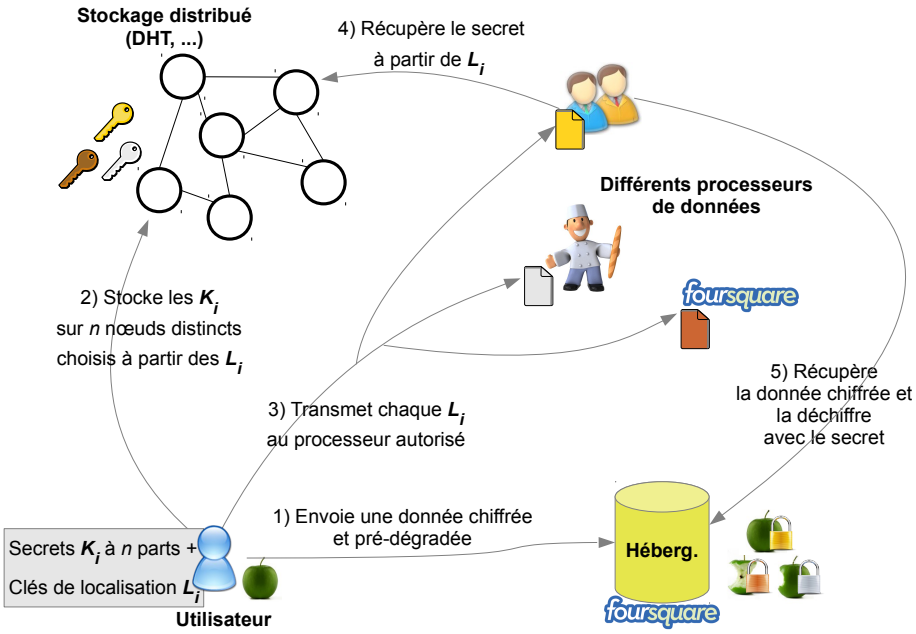

FIGURE 1: Schéma simplifié de notre architecture, avec les différentes étapes de publication d'une donnée.

Hébergement Lorsqu'un utilisateur veut ajouter une entrée dans la base de données, il doit dégrader les données avant de les transmettre à l'hébergeur. Dans notre contexte, on ne peut pas parler du niveau global de précision d'une entrée, il faut traiter les données attribut par attribut. Avoir différents niveaux de précision pour un attribut revient à dédoubler la colonne correspondante de la base, ce qui augmente l'espace de stockage nécessaire pour chaque entrée. Toutefois, les données dont la date d'expiration est passée peuvent être supprimées, ce qui permet de récupérer de l'espace de stockage le cas échéant.

Cependant, le SGBD n'a pas connaissance de l'expiration des données. Il faut donc introduire un nouveau mécanisme pour lui permettre de supprimer les données obsolètes. Nous envisageons deux solutions :

- le SGBD considère une durée de vie *a priori* pour chaque colonne, ce qui n'empêche pas l'utilisateur de spécifier une durée de vie plus courte. Par contre, s'il spécifie une durée de vie plus longue, la clé sera présente dans le système de gestion des clés mais la donnée aura déjà été effacée de la base ;
- l'utilisateur spécifie au moment où il ajoute une entrée la date d'expiration de chaque attribut. Cette solution augmente légèrement la consommation de bande passante, mais offre une plus grande flexibilité au système.

Gestion des clés Pour illustrer notre proposition, nous décrivons ici le fonctionnement d'un système de type Vanish, mais n'importe quel autre système de publication éphémère conviendrait également. Le système de gestion des clés est le suivant : pour chaque clé de chiffrement utilisée, l'utilisateur effectue un partage de secret. Le partage de secret, ou schéma à seuil (k, n), est une technique cryptographique consistant à découper un secret en n parts avec les propriétés suivantes :

- n'importe quel sous-ensemble de k parts permet de reconstruire l'intégralité du secret ;
- tout sous-ensemble d'au plus $(k-1)$ parts ne donne aucune information sur le secret.

L'utilisateur choisit ensuite aléatoirement une clé de localisation L qui détermine à quelle adresse seront stockées les parts dans la THD, puis il transmet aux nœuds correspondants la date d'expiration souhaitée et une des n parts du secret. Enfin, il envoie à chaque processeur la ou les clés de localisation qui lui permettront de déchiffrer les attributs auxquels il a le droit d'accéder.

3.4 Hypothèses et limitations

Notre architecture reposant sur un système de publication éphémère, elle doit vérifier les mêmes hypothèses que le système en question. Pour Vanish, il faut en particulier les propriétés suivantes :

- le sujet est capable d'établir un canal de communication asynchrone et sûr avec les processeurs, pour leur transmettre les clés de localisation. C'est possible par courrier électronique avec chiffrement asymétrique, par exemple, mais cela constitue un point d'attaque supplémentaire sur le système. De plus, cela peut complexifier l'interface du système et avoir un impact sur l'adoption par le grand public [7] ;
- un attaquant ne peut pas de manière réaliste prendre le contrôle d'une part significative de la THD et collecter préventivement assez de parts du secret pour déchiffrer les données après leur date d'expiration. Cela dépend de la THD utilisée, et la validité de cette hypothèse a été contestée [13], mais les différentes corrections et extensions réalisées [3, 5] offrent des garanties raisonnables sur ce point.

De plus, quel que soit le système de publication éphémère utilisé, seule l'expiration de l'exemplaire original de la donnée est garantie. Si un acteur accède (de manière légitime) à une donnée en clair avant la date d'expiration et en fait une copie, cette copie ne sera pas protégée, il est donc nécessaire que chaque acteur prenne un soin particulier à éviter les copies intempestives, comme cela peut se produire avec la mise en cache de pages web ou l'archivage de requêtes sur une base de données, par exemple.

Enfin, la base de données étant chiffrée, de nombreuses recherches ou agrégations de données deviennent impossibles. Ce n'est pas toujours gênant, notamment dans le cas d'interactions entre utilisateurs d'un réseau social (accès direct à une entrée donnée, sans recherche), mais cela peut empêcher certaines tâches de fouilles de données par exemple. On peut supposer que les acteurs qui réalisent ces tâches sont des processeurs autorisés à un faible niveau d'accès, qui feront une copie locale en clair des données traitées, qu'ils auront la charge de protéger convenablement contre les accès illégitimes.

4 Vers un réseau géo-social avec dégradation des données.

L'architecture présentée dans la section précédente est générique et peut potentiellement être appliquée à n'importe quelle base de donnée. Cependant, de nombreux détails sont spécifiques à chaque application particulière : nombre d'étapes de dégradation, choix du processus de dégradation (généralisation, ajout de bruit, ...), durée de rétention pour chaque niveau de précision, etc. Pour illustrer plus précisément notre propos, nous décrivons dans cette section un exemple fictif mettant en application ce qui précède.

Pour notre exemple, nous avons choisi un réseau géo-social de type Foursquare car les données géo-localisées se prêtent relativement bien au principe de dégradation : on peut généraliser des coordonnées GPS par troncature, un lieu avec un arbre ontologique, etc.

Tout d'abord, définissons les éléments du système.
Les acteurs en présence sont : (a) l'entreprise qui gère le réseau géo-social ; (b) un utilisateur-témoin, du point de vue duquel les exemples seront détaillés ; (c) ses relations dans le réseau ; (d) des fournisseurs d'applications tierces.
Les données stockées par le réseau sont : (i) lieux visités ; (ii) coordonnées visitées ; (ii) date des visites ; (iii) conseils sur un lieu. Les lieux sont dégradés par généralisation en utilisant une ontologie (supposée partagée entre tous les utilisateurs). Les coordonnées sont dégradées par transformation en intervalle. Les conseils sont dégradés par suppression du nom de l'auteur.

Enfin, déroulons quelques cas d'usage :

- une relation de l'utilisateur-témoin veut le retrouver : c'est un processeur autorisé avec un niveau de précision maximale sur les coordonnées visitées, mais avec un délai de rétention très faible (quelques heures) ;
- un commerçant local (application tierce) veut envoyer une promotion à l'utilisateur lorsqu'il est à proximité : c'est un processeur autorisé pour les coordonnées visitées avec un niveau de précision "quartier" pour un délai de rétention de quelques heures ;

7. L'article original sur Vanish proposait un plug-in Firefox pour envoyer des mails éphémères via GMail, et les auteurs avaient particulièrement veillé à ce que son utilisation soit la plus transparente possible.

– l'entreprise gestionnaire veut établir un profil publicitaire : c'est un processeur autorisé pour les lieux visités, mais seulement au niveau de précision "catégorie de lieu", avec un délai de un an. Ces réglages sont suffisants pour définir des habitudes de consommation, sans révéler la localisation ou les détails de l'emploi du temps de l'utilisateur.

Ces exemples mettent en évidence que le choix des étapes de dégradation et des délais de rétentions pour chaque état ne doit pas être fait arbitrairement, mais en cohérence avec les services souhaités. L'ensemble des choix possibles peut être complexe à gérer pour un utilisateur non-avertis, mais on peut raisonnablement espérer que des moteurs automatisés de gestions des politiques de confidentialité comme le PPL Engine [10] pourront à terme intégrer ce paramétrage dans leurs fonctionnalités.

Conclusion

Dans ce travail, nous défendons l'intérêt de la dégradations des données en tant qu'alternative à leur suppression, pour augmenter la valeur totale d'un ensemble de données et minimiser l'impact des intérêts divergents entre utilisateurs et hébergeurs. De plus, ce nouveau mécanisme ne pourra être largement adopté que s'il permet une confiance accrue entre les différents acteurs. Nous affirmons donc que l'implémentation de la dégradation ne doit pas être laissée aux mains des hébergeurs et nous présentons une architecture décentralisée, fondée sur un système de publication éphémère, qui permet des garanties techniques sur la mise en pratique de ce mécanisme.

Nous terminons avec un exemple fictif qui permet d'illustrer nos propos et détaille certains points importants de la conception d'un système de dégradation par publication éphémère. Un prototype de démonstration de cet exemple, utilisant un système existant de publication éphémère et des données réelles est en cours de développement.

Références

[1] Nicolas Anciaux, Luc Bouganim, Harold Van Heerde, Philippe Pucheral, and Peter M.G. Apers. Data Degradation : Making Private Data Less Sensitive Over Time. In *Proceedings of the 17th ACM Int. Conf. on Information and Knowledge Management (CIKM)*, pages 1401–1402, 2008.

[2] Claude Castelluccia, Emiliano De Cristofaro, Aurelien Francillon, and Mohamed-Ali Kafaar. EphPub : Toward robust ephemeral publishing*. *Proceedings of IEEE ICNP*, pages 1–18, 2011.

[3] Roxana Geambasu, Tadayoshi Kohno, Arvind Krishnamurthy, Amit Levy, Henry Levy, Paul Gardner, and Vinnie Moscaritolo. New directions for self-destructing data systems. Technical report, 2011.

[4] Roxana Geambasu, Tadayoshi Kohno, Amit A. Levy, and Henry M. Levy. Vanish : Increasing data privacy with self-destructing data. *Procs 18th USENIX Security*, 2009.

[5] Roxana Geambasu, Amit Levy, Yoshi Kohno, Arvind Krishnamurthy, and Hank Levy. Comet : an active distributed key-value store. *Procs 9th OSDI*, 2010.

[6] Srijith K. Nair, Mohammad T. Dashti, Bruno Crispo, and Andrew S. Tanenbaum. A hybrid PKI-IBC based ephemerizer system. *Procs 22nd IFIP SEC*, 2007.

[7] Radia Perlman. The Ephemerizer : Making data disappear. *Journal of Information System Security (JISSec)*, 2005.

[8] Adi Shamir. How to share a secret. *Communications of the ACM*, November 1979.

[9] Qiang Tang. From Ephemerizer to Timed-Ephemerizer : Achieve assured lifecycle enforcement for sensitive data (extended version). *initially EUROPKI*, 2009.

[10] Slim Trabelsi, Akram Njeh, Laurent Bussard, and Gregory Neven. PPL engine : A symmetric architecture for privacy policy handling. *W3C Workshop on Privacy and data usage control*, October :5, 2010.

[11] Harold van Heerde. *Privacy-aware data management by means of data degradation : making private data less sensitive over time.* PhD thesis, University of Twente, Enschede, The Netherlands, June 2010.

[12] Harold van Heerde, Maarten Fokkinga, and Nicolas Anciaux. A framework to balance privacy and data usability using data degradation. In *Procs 12th CSE*. Ieee, 2009.

[13] Scott Wolchok, Owen S. Hofmann, Nadia Heninger, Edward W. Felten, J. Alex Halderman, Christopher J. Rossbach, Brent Waters, and Emmett Witchel. Defeating Vanish with low-cost Sybil attacks against large DHTs. In *Procs 17th NDSS*, 2010.