2018-ENST-046

EDITE - ED 130

Doctorat ParisTech

T H È S E

pour obtenir le grade de docteur délivré par

TELECOM ParisTech

Spécialité « Informatique et Réseaux »

présentée et soutenue publiquement par

Cédric VAN ROMPAY

le 4 octobre 2018

Protocoles Multi-Utilisateurs de Recherche sur Bases de Données Chiffrées

Multi-User Searchable Encryption

TELECOM ParisTech

école de l'Institut Télécom - membre de ParisTech

T H È S E

Multi-User Searchable Encryption

Cédric Van Rompay

Quelques fleurs sur la terre et toutes les étoiles dans le ciel.
—Victor Hugo, Les Misérables.

Remerciements

Je voudrais remercier particulièrement mon directeur de thèse Refik Molva pour m'avoir convaincu de faire une thèse sur la cryptographie avec lui. Quand je suis venu le voir dans son bureau parce que je me posais des questions sur la recherche, je n'avais aucune idée des conséquences que cette discussion allait avoir sur ma vie, et je lui en serais toujours redevable. Merci aussi à Davide Balzarotti et Aurélien Francillon qui m'ont également reçu dans leur bureaux respectifs à la même période.

Ensuite, mes remerciements vont évidemment à mes collègues et en particulier mes co-auteurs, aux autres chercheurs d'EURECOM et au personnel, à mes proches et à mes amis. Je me vois mal faire une liste des noms de toutes les personnes concernées, avec des annectodes, des détails rétrospectifs et des *private jokes*, comme on voit dans beaucoup de thèses. Par pudeur sans doute, mais aussi parce que ce sont des choses personnelles que je préfère dire personnellement.

Il y a en revanche des personnes que je n'aurais jamais l'occasion de remercier personellement mais qui m'ont beaucoup apporté pour cette thèse. Et notamment il y a la communauté des chercheurs en cryptographie et en sécurité en général sur Internet. C'est tout de même une époque fantastique, où il suffit de brancher un cable Ethernet dans un mur pour avoir accès à plus de ressources qu'on ne pourra jamais en lire et à une communauté prête à discuter et aider sans contrepartie. Je voulais donc profiter de cette page pour dire, à tous ceux qui ont rendu possible des lieux comme `https://eprint.iacr.org`, `https://dblp.org` ou `https://crypto.stackexchange.com`, ou des projets open source qui m'ont aidé, et à tous ceux qui ont bien voulu discuter avec moi,
merci.

Résumé en Français

Augmenter la Sécurité des serveurs « dans les nuages »

Le modèle dit « client-serveur », où un ordinateur (le serveur) effectue des tâches pour le compte d'une autre machine (le client), est au moins aussi vieux que l'Internet[1]. Cependant au début des années 2000, des progrès significatifs dans les domaines des réseaux, de la virtualisation matérielle et de l'algorithmique distribuée ont permis ce qui s'apparente à un changement d'échelle de ce modèle client-serveur : La fourniture même de serveurs devient un service. Un utilisateur peut, sur simple demande, « louer » momentanément une certaine quantité de puissance de calcul, de bande passante et de capacités de stockage de données. Cette « location » a lieu de façon complètement dématérialisée, via le réseau Internet, et pour une quantité de ressources et une durée aussi grandes ou aussi petites que nécessaire. C'est l'émergence du paradigme du *cloud computing*, ou « informatique dans les nuages » en français. La majorité des systèmes d'information, allant des téléphones grand public aux départements d'informatique de grandes entreprises, se mettent alors à utiliser cette nouvelle infrastructure flexible et bon marché pour externaliser une grande partie de leurs tâches qui autrefois auraient été traitées localement.

Les avantages que peut offrir cette nouvelle façon d'utiliser l'outil informatique, sous forme de coûts réduits, de flexibilité, de simplicité d'utilisation et de connectivité, sont tels que les entreprises ainsi que les concepteurs de solutions informatiques semblent obligés d'opter pour l'utilisation du « nuage », sous peine de perdre leur part de marché. Mais cette externalisation massive des tâches vers un tiers-parti (le nuage) crée d'importants problèmes de confiance et de sécurité : non seulement les nuages publics sont des cibles privilégiées pour les attaquants de part la quantité de valeur numérique qu'ils concentrent, mais l'utilisation massive de nuages par une entité augmente aussi la « surface d'attaque » d'une entité (le nombre de moyens par lesquels l'attaquant peut entrer dans le système) de part l'utilisation massive du réseau Internet au lieu d'un réseau privé, en même temps qu'elle peut réduire les compétences techniques de ladite entité en la poussant à réduire son investissement dans les compétences numériques.

Ce constat, principale inquiétude envers l'informatique dans les nuages et principal frein à une adoption encore plus massive, a donné lieu à la recherche de nouvelles techniques pour augmenter la sécurité dans ce paradigme. Parmi ces techniques, on trouve un certain nombre de protocoles cryptographiques qui sont censés apporter des garanties aux utilisateurs de nuages mêmes dans le cas où un nuage ne serait pas digne de confiance, que ce soit un attaquant externe qui obtienne un contrôle partiel ou total du nuage ou une menace venant de l'administration même du nuage. Les protocoles de recherche sur BDC (Bases de Données Chiffrées, en anglais *Searchable Encryption*)

[1]le mot *server* (« serveur » en anglais) est présent par exemple dans le document RFC №5 (daté de 1969) décrivant le fonctionnement du réseau ARPANET, ancêtre de l'Internet.

sont un exemple de ces solutions : leur objectif est de permettre d'effectuer des recherches dans une base de données stockée sur un serveur distant, tout en limitant la quantité d'information que ledit serveur puisse obtenir sur la donnée stockée et les requêtes qui sont faites, au cas où ce serveur soit corrompu.

Le premier papier de recherche sur les BDC date de l'an 2000 [SWP00]. Ce sujet de recherche est depuis devenu particulièrement actif, comme on peut le voir dans les synthèses de Bösch et al. [Bös+14] et de Fuller et al. [Ful+17]. Les principaux progrès qui ont été réalisés ont été dans un premier temps sur la façon de définir la notion de confidentialité dans le contexte d'une base de donnée chiffrée, jusqu'au papier historique de Curtmola et al. [Cur+06] qui propose la première définition à faire consensus. Ce papier présente également le premier protocole dont le temps d'exécution d'une recherche est linéaire par rapport au nombre d'entrées de la base de données qui correspondent à la requête exécutée, au lieu d'être linéaire par rapport au nombre total d'entrées de la base de données comme cela était le cas avec les protocoles précédents. Après le papier de Curtmola et al. [Cur+06], des progrès ont continué d'être faits sur le plan de la performance, mais aussi la complexité des requêtes, et enfin sur la prise en comptes de contextes avec plusieurs utilisateurs : d'une part des contextes ou n'importe qui peut ajouter des entrées chiffrées dans la base de données mais un seul utilisateur peut effectuer des recherches (protocoles « PEKS »), et d'autre part des protocoles où un seul utilisateur peut ajouter des entrées dans la base de données mais peut ensuite donner l'autorisation à d'autres utilisateurs de chercher dans cette base de données.

Contexte Multi-Utilisateurs et Nécéssité de Modéliser des Utilisateurs Corrompus

L'objet de cette thèse est l'étude d'une famille de BDC où l'accès en lecture (c'est-à-dire le droit d'effectuer des recherches) *et* l'accès en écriture sont tous les deux répartis entre une multitude d'utilisateurs. On utilise le terme de *BDC multi-lecteurs-multi-auteurs*, ou plus simplement de *BDC multi-utilisateurs*, abrégé BDCMU, pour nommer cette famille de protocoles (en anglais, *Multi-User Searchable Encryption (MUSE) protocols*). Ce type de BDC est naturellement plus difficile à concevoir et il faut s'attendre à ce que les solutions auxquelles on parviennne aient des performances moindres que des protocoles de BDC ne prenant en compte qu'un unique utilisateur ; mais le modèle de BDCMU, contrairement au modèle à utilisateur unique, convient à des scénarios où une large base de donnée est principalement un agrégat d'un grand nombre de contributions, toutes modestes et venant d'un ayant-droit différent. Ce genre de scénarios devient de plus en plus pertinent avec l'ère de la collecte massive de données personnelles et l'apparition de normes comme le RGPD européen [GDPR] qui demande à assurer le droit des individus à un contrôle de l'utilisation de leur données personnelles.

Concrètement une BDCMU, illustrée Figure 1, compte un grand nombre d'utilisateurs répartis en deux groupes, les *auteurs* et les *lecteurs*. Chaque auteur possède de la donnée qu'il peut ajouter dans la BDC sous forme d'entrées, et peut décider quels lecteurs sont autorisés à effectuer des recherches sur ces entrées. Un exemple typique d'entrée pour les protocoles qui sont présentés dans cette thèse sont des ensembles de mots-clés correspondant à des fichiers hébergés sur un autre système quelconque (typiquement un système de stockage dans le nuage), dans le but de décider

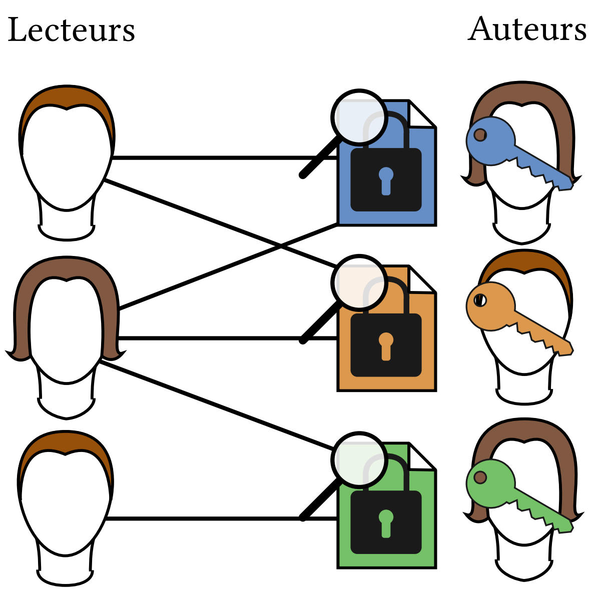


FIGURE 1 : Une illustration du contexte multi-utilisateurs

quels fichiers parmis un large ensemble doivent être téléchargés pour une tâche spécifique. Les propriétés de sécurité d'un système de BDCMU sont censées assurer à un auteur que ses entrées dans la BDC ne soient pas accessibles à un potentiel attaquant.

Plus formellement, on définit la syntaxe d'une BDCMU de la façon suivante :

- R est l'ensemble des lecteurs.
- Chaque entrée de la base de données est associé à un identifiant unique d et l'entrée est notée soit W_d, soit $\boldsymbol{W}[d]$ où $\boldsymbol{W}$ est l'ensemble de toutes les entrées dans le système.
- On note $q_{r,s}$ la requête numéro s du lecteur $r \in R$. On peut aussi utiliser la notation $\boldsymbol{q}[r][s]$ où $\boldsymbol{q}$ est la liste bi-dimensionelle de toutes les requêtes qui sont envoyées. On utilise parfois le notation "$\forall q_{r,s} \in \boldsymbol{q}$" comme un synonyme de $\forall(r,s) : r \in R \wedge s \in [|\boldsymbol{q}[r]|]$".

Un système BDCMU est alors défini comme suit :

- $\mathsf{Reader.KeyGen}(1^\lambda) \rightarrow \rho_r$: cet algorithme est exécuté par chaque lecteur r pour générer sa clé de lecteur.
- $\mathsf{Writer.KeyGen}(1^\lambda) \rightarrow \gamma_d$: Cet algorithme est exécuté par l'auteur possédant l'entrée W_d pour générer la clée de chiffrement pour cette entrée.
- $\mathsf{Writer.Encrypt}(W_d, \gamma_d) \rightarrow \overline{W}_d$: Cet algorithme est exécuté par un auteur pour chiffrer l'entrée W_d avec la clée γ_d. Le résulat, appelé l'**entrée chiffrée** $\overline{W}_d$ est envoyée au serveur. Noter que parler d'*« entrée chiffrée »* ou de *« mots-clé chiffrés »* est souvent un abus de language, étant donnée que dans la plupart des protocoles BDC les mots-clé chiffrés ne sont pas déchiffrable. Certains auteurs parlent d'"empreinte" au lieu de "mot-clé chiffré", mais on choisit de

garder le terme d'"entrée/mot-clé chiffré" parce qu'il a été utilisé dans une large part de la littérature sur les BDC

- Writer.Delegate$(r, \gamma_d) \rightarrow \Delta_{r,d}$: Dans certains protocoles BDCMU, cet algorithme est exécuté par un auteur pour autoriser un lecteur r à chercher dans l'entrée W_d qui a été chiffrée avec la clé γ_d. Le résultat, noté $\Delta_{r,d}$, est appelé un **Delta** et est envoyé au serveur. On définit également la fonction $Auth$ qui à un lecteur r associe les identifiants des entrées dans lesquelles que ce lecteur est autorisé à chercher. Autrement dit, $d \in Auth(r)$ signifie que r a accès à l'entrée W_d.

- Reader.Trapdoor$(q_{r,s}, \rho_r) \rightarrow t_{r,s}$: Cet algorithme est exécuté par le lecteur r avec comme entrées la requête $q_{r,s}$ et la clé ρ_r. Le résultat, noté $t_{r,s}$, est la **trappe** correspondant à $q_{r,s}$ et est envoyée au serveur. Une trappe peut être vue comme une "requête chiffrée" (ce qui là encore est un abus de language).

- Server.Search$(t_{r,s}, \boldsymbol{\Delta}, \overline{\boldsymbol{W}}) \rightarrow p_{r,s}$: Cet algorithme est exécuté par le serveur avec comme entrées la trappe $t_{r,s}$, l'ensemble $\boldsymbol{\Delta}$ de tous les deltas, et l'ensemble $\overline{\boldsymbol{W}}$ de toutes les entrées chiffrées. Le résultat, noté $p_{r,s}$, est la **réponse serveur** correspondant à la requête $q_{r,s}$ et est envoyée au lecteur r.

- Reader.Open$(p_{r,s}) \rightarrow a_{r,s}$: Cet algorithme est exécuté par le lecteur r avec comme entrées la réponse serveur $p_{r,s}$ (et dans certains protocoles, certaines clés possédées par r). Le résulat, noté $a_{r,s}$, est le **résultat de requête** correspondant à la requête $q_{r,s}$.

Cependant il reste à définir comment modéliser l'attaquant. Le fait qu'un lecteur dans un BDCMU ne puisse pas avoir accès à une entrée tant que son auteur n'en ait pas donné son accord laisse entendre que les utilisateurs ne se font pas confiance entre eux *a priori* : dit autrement, si l'autorisation d'un lecteur par un auteur exprime une certaine confiance en ce lecteur, alors l'abscence d'autorisation exprime une certaine méfiance. Cette observation sur la nature même des BDCMU, à laquelle s'ajoute l'évidence qu'un attaquant a bien plus de chances de prendre le contrôle d'un des utilisateurs du système que du serveur qui héberge la BDC, nous pousse à conclure qu'un modèle naturel d'adversaire pour une BDCMU doit inclure les utilisateurs dans les menaces potentielles. Autrement dit, un utilisateur d'un BDCMU ne se défend pas seulement contre un serveur potentiellement corrompu, mais aussi contre tous les autres utilisateurs, ou au moins ceux qu'il n'a pas explicitement autorisé à accéder à sa donnée.

On remarque que les premiers protocoles de BDCMU présentés dans la litérature scientifique, ne sont étudiés au contraire que dans un modèle de sécurité où seul le serveur est considéré comme une menace. D'une certaine façon, il est naturel dans l'étude d'un nouveau problème de commencer par l'étude d'une version simplifiée de celui-ci, en ayant recours à des hypothèses supplémentaires (ici, n'étudier la sécurité que face au serveur revient à faire l'hypothèse que tous les utilisateurs sont dignes de confiance) qui mènent à des cas particuliers. D'une part ces cas particuliers peuvent correspondre à des situations réelles, et d'autre part les techniques développées dans la résolution du cas particulier peuvent aider à résoudre le problème de façon générale, en une sorte d' « apprentissage ».

La première contribution de cette thèse est la remise en cause de l'idée que l'étude des BDCMU avec le serveur comme seul adversaire reflète un quelconque problème concret. Le scénario pratique qui est invoqué dans les papiers de recherche utilisant ce modèle simplifié d'adversaire est celui d'une entreprise qui externalise une base de donnée « dans le nuage » et où les utilisateurs sont les employés de cette entreprise. Comme l'entreprise a un certain contrôle sur ses employés, il est raisonable de penser que les utilisateurs de la BDC ne soient pas corrompus et que seul le nuage soit une source d'inquiétude. Or il apparaît que dans ce scénario, et probablement dans tout scénario similaire où une unique entitée a un contrôle fort sur la totalité des utilisateurs, la mise en place d'un protocole de BDC *à utilisateur unique* est possible. Mieux, au vu des performances naturellement supérieures des BDC à utilisateur unique par rapport à leur pendant multi-utilisateurs, une telle option n'est pas seulement possible, elle est préférable. Il y a donc de fortes chances pour que n'importe quel scénario de BDCMU qui justifie de ne considérer que le serveur comme une menace soit en fait un scénario où la solution adaptée se révèle être une BDC à utilisateur unique (couplée à un système centralisé de contrôle des utilisateurs).

L'incorporation des utilisateurs dans le modèle d'adversaire ne semble donc pas un « mieux » dans l'étude des BDCMU, mais une *nécessité*. Une publication de Popa et Zeldovich en 2013 [PZ13] et l'impact important qu'elle a eu dans la litérature vient appuyer cette conclusion en proposant le premier BDCMU censé assurer la confidentialité des données et des requêtes face à un adversaire contrôlant à la fois le serveur et certains utilisateurs.

Vulnérabilité Intrinsèque des BDCMU Existantes Face aux Utilisateurs Corrompus

La deuxième contribution de cette thèse est la démonstration que l'adoption d'un modèle de sécurité où seul le serveur est une menace, au lieu de donner naissance à des techniques qui se révèlent utiles dans un modèle plus réaliste, a poussé au contraire à l'adoption de techniques qui rendent les protocoles naturellement faibles face à des utilisateurs corrompus.

Cette vulnérabilité est causée par la façon dont le serveur, dans les protocoles de BDCMU, applique les requêtes (chiffrées) sur la base de donnée (chiffrée elle aussi) afin de trouver quelles sont les entrées qui correspondent à la requête. Dans ces protocoles, chaque entrée de la BDC a la forme d'un ensemble de mots-clés chiffrés, et le serveur applique la requête chiffrée à chacun de ces mots-clés chiffrés pour décider si l'entrée correspond ou non à la requête. Bien que chaque protocole BDCMU existant soit différent, tous suivent cette structure algorithmique consistant à appliquer un certain « test » (c'est à dire une fonction renvoyant « vrai » ou « faux ») sur chacun des mots-clés encryptés. La différence entre les différents protocoles consiste principalement en ce en quoi consiste ce test qui est effectué. Nous nommons cette structure *test itéré* (en anglais *iterative testing*), décrite par l'algorithme 1, et nous montrons que cette structure condamne les protocoles qui l'implémentent à être vulnérables face à un adversaire qui contrôle une proportion même très faible des utilisateurs du système (en plus de controler le serveur lui-même, hypothèse centrale de toutes les BDC).

En effet, la structure dite de « test itéré » révèle beaucoup d'information au serveur : non seulement le serveur voit quelles sont les entrées qui correspondent à la requête (même si dans l'immédiat

Algorithm 1 : La structure algorithmique dite de « test itéré »

```
Algorithm : Search
Input : $t_{r,s}, \mathbf{\Delta}, \overline{\boldsymbol{W}}$
Output : $p_{r,s}$
Initialize $p_{r,s}$ as an empty set;
for $d \in Auth(r)$ do
    for $\overline{w} \in \overline{\boldsymbol{W}}[d]$ do
        if $\mathsf{Test}(t_{r,s}, \overline{w}) = \mathit{True}$ then
            $p_{r,s} \leftarrow p_{r,s} \cup \{d\}$;
```

il ne voit ni le contenu de l'entrée ni celui de la requête), mais quand une telle correpondance est trouvée, le serveur voit quel mot-clé encrypté à causé un test positif. On pourrait penser en premier lieu qu'un telle fuite d'information soit relativement bénigne, pour les raisons suivantes : d'une part il est très courant qu'un protocole de BDC laisse le serveur voir quelles entrées correspondent à une requête ; cette fuite d'information, nommé *access pattern* en anglais (que l'on pourrait traduire par *motif d'accès* en français), est plus ou moins admise comme acceptable dans la litérature sur les BDC au nom de la performance, même si de récentes attaques exploitant cette fuite d'information on été publiées qui remettent cette considération en cause. D'autre part, l'information supplémentaire que le test itératif révèle au serveur semble minime : au lieu de voir que *« telle entrée correspond à telle requête »* (le « motif d'accès » mentionné prédédemment), le serveur va voir par exemple que *« le troisième mot-clé encrypté de telle entrée correspond à telle requête »*. L'ordre des mots-clés encryptés dans les entrées étant arbitraire dans les protocoles concernés, quelle information le serveur pourrait-il bien tirer de cette information ? En réalité, ce « petit » surplus d'information permet au serveur de voir si deux requêtes concernent le même mot-clé, *même si ces deux requêtes proviennent de lecteur différents*, à condition toutefois que ces deux lecteurs aient tous deux accès à une même entrée qui s'avère contenir ce mot-clé. Ce phénomène est illustré Figure 2.

Dans des travaux dont nous avons publié les résultats à la conférence PETS 2017, nous expliquons pourquoi cette information suplémentaire en apparence « bénigne » est en fait la cause d'une importante vulnérabilité des protocoles concernés (c'est à dire de tous les protocoles BDCMU existants avant cette thèse, et même de certains publiés après) face à des utilisateurs corrompus. Notamment nous donnons des résultats quantitatifs issus de simulations utilisant des corpus de documents réels montrant qu'un adversaire contrôlant le serveur ainsi qu'un nombre minime d'utilisateurs, après que le serveur ait traité un nombre conséquent mais réaliste de requêtes et ce dans le respect total des protocoles concernés, cet adversaire était capable de retrouver une très grande partie du contenu de la base donnée et des requêtes qui ont été envoyées. Nous insistons sur le fait que les mots-clés reconstruits de cette manière par l'adversaire ne sont pas dans des entrées auxquelles les utilisateurs corrompus ont accès. Ainsi du point de vue d'un utilisateur quelconque de ces protocoles BDCMU, la corruption d'un quelconque autre utilisateur du système, sans que cet autre utilisateur ait reçu d'autorisation du premier, sans même qu'il lui soit nécéssairement connu, a de grandes chances de mener à une compromission substantielle des données du premier utilisateur. Ceci va clairement à l'encontre de ce qu'on attend d'un protocole comportant un système

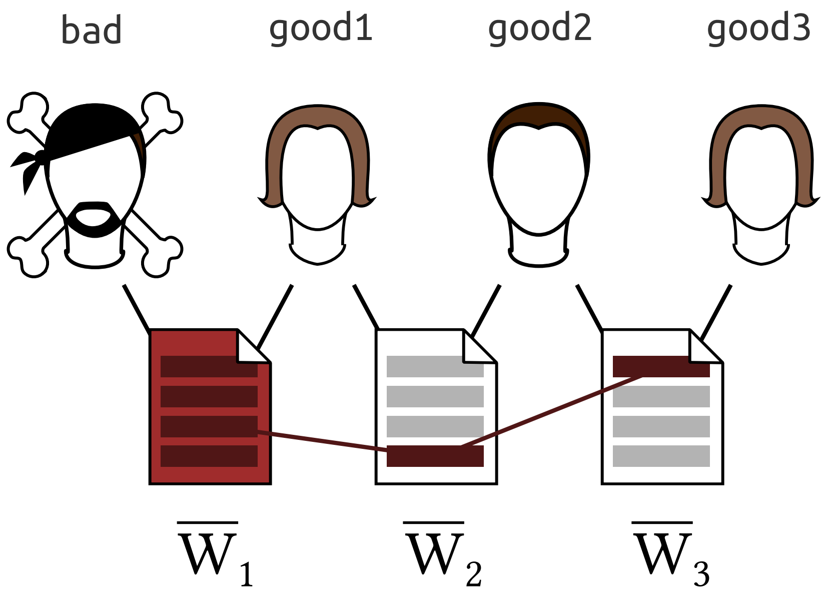


FIGURE 2 : Une illustration du phénomène à l'origine de la vulnérabilité de la structure en test itéré

L'adversaire a accès à l'entièreté de l'entrée W_1 (ici sa forme chiffrée $\overline{W}_1$ est représentée) via à l'utilisateur « bad » qui est corrompu. Puis à cause de l'information révélée par la structure en test itéré, les requêtes envoyées par les utilisateurs –non-corrompus, eux– « good1 » et « good2 » révèle à l'adversaire le contenu d'autres entrées, en violation du système d'autorisation. Voir Figure 4.1.

d'autorisations.

Ces premiers résultats montrent déjà que les protocoles BDCMU conçus dans un modèle ne prenant pas en compte des utilisateurs corrompus ne sont pas adaptés à un modèle plus réaliste, et que la cause de cela est liée à la structure fondamentale de ces protocoles, indiquant que la prise en compte d'utilisateurs corrompus va demander de « repartir de zéro » quand à la façon de concevoir ces protocoles. Mais cela va plus loin : il apparaît que le protocole de Popa et Zeldovich [PZ13], nommé MKSE, implémente lui aussi la structure de test itératif, et à ce titre est lui aussi très vulnérable face à des utilisateurs corrompus, bien que ce genre de menace soit exactement celui contre lequel ce protocole était censé protéger. Cette vulnérabilité suprenante de MKSE, combiné à l'exposition que ce protocole a eu depuis sa publication, indique que le danger de la structure de test itératif en présence d'utilisateurs corrompus, et la nécessité d'y trouver une alternative, semble ne pas avoir été remarqué ou avoir été mal comprise, ce qui donne d'autant plus d'importance à notre publication sur le sujet. De plus, cette vulnérabilité démontre la limitation des méthodes avec lesquelles la sécurité de MKSE (et ses dérivés) a été étudiée, ces méthodes n'ayant pas été à même de mettre à jour cette importante vulnérabilité qui se situe clairement dans le cadre dans lequel le papier se place, c'est à dire de la sécurité face à des utilisateurs corrompus. Pour appuyer nos affirmation sur la vulnérabilité de MKSE, nous présentons une attaque d'une implémentation de ce protocole fourni par le logiciel « Mylar » [Pop+14] provenant des concepteurs même de MKSE. La

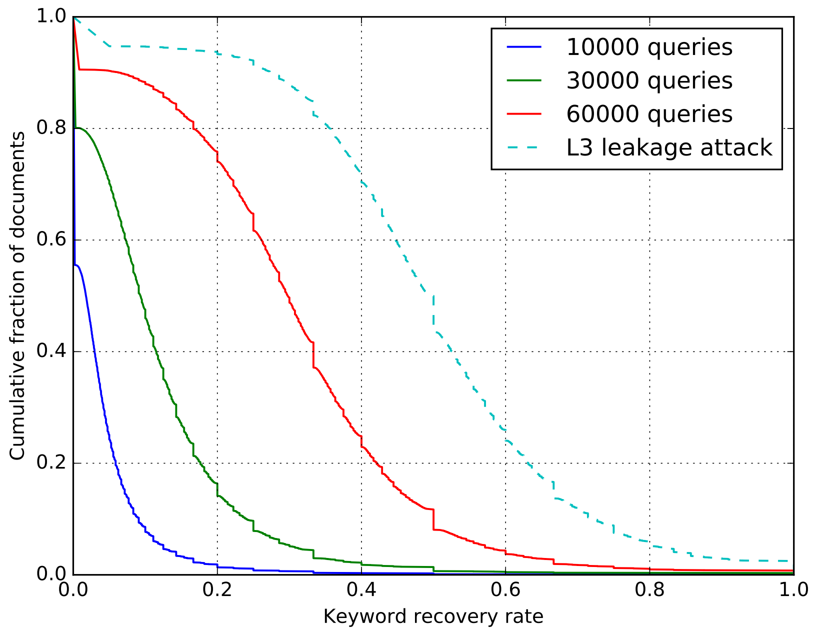


FIGURE 3 : Taux de décryption dans une de nos simulations, à nombre de requêtes observées variable.

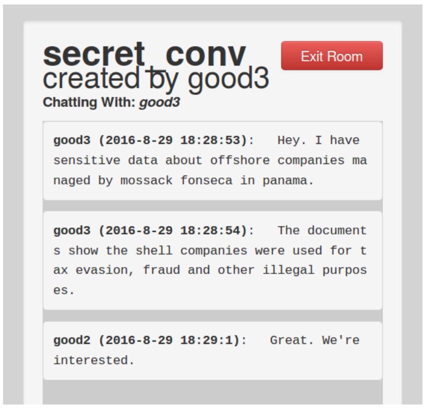
Ici le serveur n'a accès via corruption d'utilisateurs qu'à 0.02% des entrées de la base données, et on peut lire par exemple qu'avec 60 000 requêtes observées, la moitié des entrées ont au moins 30% de leur contenu décrypté.

Figure 4 présente le résultat de l'application de notre attaque sur Mylar. Enfin, nous proposont une étude de l'analyse de sécurité de MKSE présente dans [PZ13].

Une Première Génération de Protocoles BDCMU à l'Épreuve de la Corruption d'Utilisateurs

Nous espérons que les résultats de nos travaux cités précédement puisse influencer la recherche dans le domaine des BDCMU en poussant à considérer systématiquement les utilisateurs comme une menace potentielle et à chercher une voie alternative à l'utilisation de la structure en test itéré. La suite de nos travaux a pour but d'initier ces changements, en essayant de concevoir des protocoles BDCMU qui ne souffrent pas des défauts que l'on a identifiés dans l'état de l'art. Notre démarche part de ce qu'on identifie comme étant la cause de la vulnérabilité de la structure en test itératif : le fait que le serveur puisse voir quand deux requêtes venant de lecteurs différents contiennent le même mot-clé dès lors qu'une même entrée correspond à ces deux requêtes.

On remarque qu'un mécanisme introduit dans le protocole MKSE peut être ré-utilisé pour apporter un début de solution. On rappelle tout d'abord le fonctionnement de MKSE : L'objectif du



```
good3 (2016-8-29 18:28:53):   Hey. I have sensitive data about offshore companies managed by mossack fonseca in panama.

good3 (2016-8-29 18:28:54):   The documents show the shell companies were used for tax evasion, fraud and other illegal purposes.

good2 (2016-8-29 18:29:1):   Great. We're interested.
```

```
### REBUILT CONVERSATION ###
### CHATROOM: 'secret_conv' ###
good3 (2016-08-29 18:28:53): [fonseca, data,
                              mossack, panama]
good3 (2016-08-29 18:28:54): [evasion, tax]
```

FIGURE 4 : Une capture d'écran d'une conversation dans une application de messagerie protégée par le programme Mylar, et le résultat de notre programme appliquant notre attaque contre Mylar.

On voit que des informations cruciales sont récupérées par le programme d'attaque, alors que la conversation n'est pas censé être accessible à l'adversaire suivant les objectifs de sécurité fixé par le protocole MKSE que Mylar implémente

protocole MKSE était de rompre avec le fonctionnement des protocoles précédents où chaque utilisateur avait sa propre clé pour chiffrer entrées et requêtes mais cette clé était générée par un tiers-parti qui générait aussi une « clé complémentaire » donnée au serveur qui permettait de transformer les requêtes et entrées chiffrées afin qu'elles soient chiffrées sous une clé dite « maîtresse », connue de personne excepté le tiers-parti (supposé incorruptible). Le serveur, après cette opération de transformation, gérait en fait une base donnée équivalente à la BDC d'un utilisateur unique virtuel qui utiliserait la clé maîtresse. Ce fonctionnement causant de très graves fuites de données en cas de corruption même d'un seul utilisateur, l'idée d'une clé maîtresse est abandonnée dans MKSE. Au lieu de cela, les entrées dans MKSE ne sont pas transformées et restent chiffrées sous la clé de leur auteur ; les requêtes, elles, sont transformées en une copie distincte pour chaque entrée à laquelle le lecteur à l'origine de la requête a accès, de telle manière que chaque copie soit applicable à l'entrée correspondante, et uniquement à celle-ci. Bien que cela requiert une surcharge de travail de la part du serveur qui doit maintenant effectuer de nombreuses transformations à chaque requête, cette surcharge paraît acceptable vu qu'elle n'est pas à la charge de l'utilisateur, d'autant plus si elle permet d'obtenir un protocole réellement sécurisé.

Nous proposons d'appliquer cette même logique de « copie + transformation » aux entrées : Pour chaque entrée dans la base de donnée, pour chaque lecteur ayant accès à cette entrée, une copie de l'entrée est créée et transformée et le résultat de la transformation est l'entrée chiffrée sous une clé correspondant au couple lecteur-auteur concerné. Les requêtes venant du lecteur concerné sont transformées vers cette même clé lecteur-auteur et appliquées sur cette copie de l'entrée. La Figure 5 illustre la différence entre cette nouvelle structure et celles existant précédemment (MKSE et protocoles précedents).

À nouveau cette technique augmente la charge de travail du serveur qui doit maintenant trans-

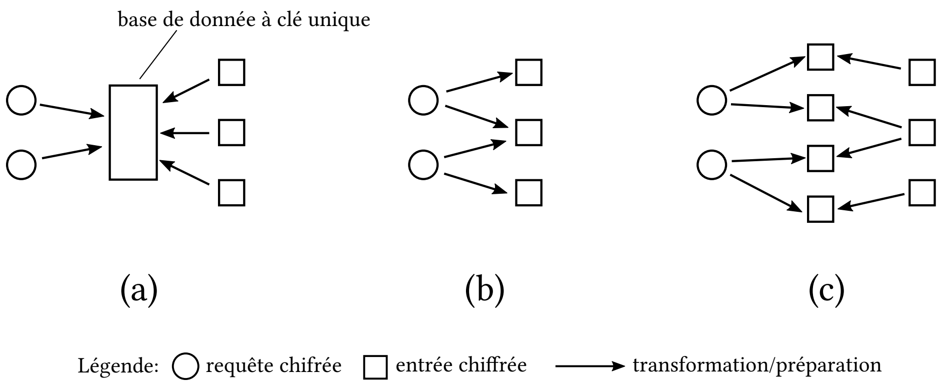

base de donnée à clé unique
(a)
(b)
(c)
Légende: ◯ requête chifrée □ entrée chiffrée → transformation/préparation

Figure 5 : Une illustration de la différence entre une structure à clé unique (a) telle qu'employé avant MKSE, la structure à clé multiple introduite par MKSE (b), et la structure avec préparation que nous proposons (c).

former chaque entrée de la base de données, et ce autant de fois qu'il y a de lecteurs ayant accès à cette entrée. Il y a aussi un certain coût pour le serveur à stocker toutes ces entrées transformées. Cette transformation ne doit cependant être faite que rarement (voire une seule fois si les lecteurs acceptent que le serveur voit quand un même lecteur envoie deux fois la même requête), alors que la transformation des requêtes introduite dans MKSE doit être faite à chaque requête. Mais surtout, le même argument qui tenait pour MKSE tient également pour cette modification que l'on présente : si cela nous permet de mettre au point les premiers protocoles BDCMU à l'épreuve d'utilisateurs corrompus, cette différence de coût, substantielle mais pas démesurée, en vaut clairement la peine. Des travaux récents de Hamlin et al. [Ham+18a] nous renforcent dans notre opinion que mettre au point un protocole de BDCMU véritablement sécurisé sans avoir recours à une duplication des entrées est particulièrement difficile.

Cette duplication et transformation des entrées ne suffisent pas à elles seules à guarantir la sécurité face à des utilisateurs corrompus, elles ne sont qu'un pas dans cette direction. En effet désormais deux requêtes venant de deux lecteurs différents ne seront plus appliquées sur une même entrée, mais sur deux copies différentes. Parce que ces deux copies sont chiffrées sous deux clés différentes on peut montrer que, même si le serveur voit quel mot-clé chiffré correspond à une requête dans chaque entrée, cette information ne suffit pas à dire si les deux requêtes correspondent au même mot-clé ou non. Mais le serveur, tel qu'on le considère jusqu'alors, a accès à plus d'information que cela : notamment, il connait aussi le lien entre les diverses entrées transformées vu qu'il est celui qui a effectué cette transformation. On peut montrer facilement que cette information supplémentaire permet au serveur d'extraire autant d'information que dans un protocole fonctionnant par test itéré.

Mais une nouvelle modification vient résoudre ce problème et nous amène au premier protocole BDCMU avec des propriétés de sécurité nous semblant satisfaisantes. Cette modification demande à introduire un deuxième serveur, potentiellement corrompu tout comme le premier, mais dont on fait l'hypothèse qu'il ne collabore pas avec le premier serveur. Nous appelons ce deuxième serveur le *proxy*. L'idée est que le serveur s'occupe de la transformation des entrées pendant que proxy s'occupe de la transformation des requêtes et de leur application sur les entrées. Ainsi le proxy voit la location des mots-clés chiffrés correspondant aux requêtes mais pas les relation entre ces mots-clés chiffrés car il n'est pas celui qui a effectué la transformation des entrées, et le serveur connait lesdites relations mais ne voit jamais les requêtes car seul le proxy y a accès. La combinaison de ces techniques, illustrée Figure 6, donnent lieu au premier protocole que l'on présente dans cette thèse, nommé DH-AP-MUSE. Non seulement DH-AP-MUSE est beaucoup plus sécurisé que tous les protocoles BDCMU existants auparavant car nous démontrons qu'il est à l'épreuve d'utilisateurs corrompus, en utilisant une technique de preuve plus proche de ce qui est habituellement utilisé pour les protocoles BDC (preuve par simulation), et beaucoup moins sujette à notre sens à des défauts tels que ceux que l'on a identifié dans l'analyse de sécurité originale de MKSE. Mais DH-AP-MUSE est également plus rapide dans le traitement des requêtes que MKSE : un méchanisme présent dans MKSE oblige le serveur, quand il veut appliquer une requête sur une entrée, à l'appliquer sur chacun des mots-clés chiffrés de cette entrée. Ce méchanisme peut être enlevé, mais la sécurité de MKSE en serait (encore d'avantage) dégradée. Dans DH-AP-MUSE au contraire, l'application d'une requête sur une entrée se fait en un temps très court et indépendant du nombre de mots-clés dans l'entrée. Aussi, DH-AP-MUSE n'utilise pas de couplages, qui sont très coûteux, contairement à MKSE. Nos travaux sur DH-AP-MUSE vont être présentés à la conférence ICICS 2018 [RMÖ18a].

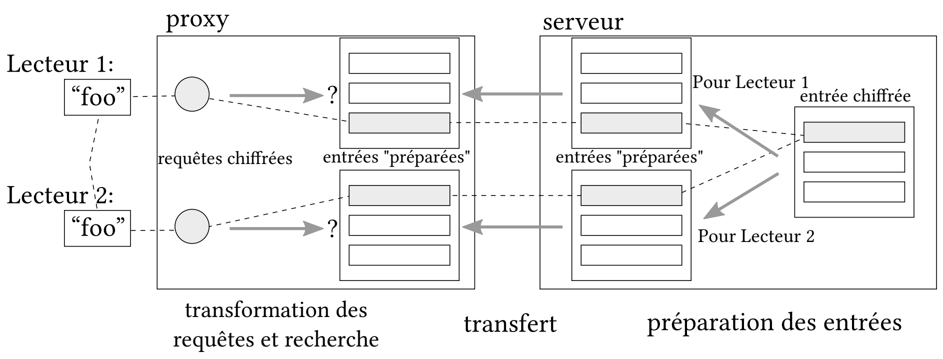


FIGURE 6 : Une illustration de notre structure à deux serveurs.

Ici, deux requêtes chiffrées correspondant au même mot-clé sont envoyées par deux lecteurs différents. Le proxy voit quels mot-clés dans les entrées « préparées » correspondent à ces requêtes, mais il ne voit pas le fait que ces « mot-clés préparés » viennent en fait du même mot-clé encrypté. Le serveur quand à lui voit cette origine commune des deux « mots clés préparés » mais ne voit pas les requêtes ni le résultat de leur application.

En récapitulant, nous avons donc mis au point le premier protocole BDCMU qui protège les données et les requêtes même en présence d'utilisateurs corrompus. Pour atteindre cet objectif nous avons du faire quelques concessions, notamment augmenter la charge de travail côté serveur mais de façon modérée, et dépendre de l'existence de deux serveurs qui, même s'ils sont tous les deux considérés comme potentiellement corrompus, sont supposés ne pas collaborer dans leur effort pour extraire des informations. Cette « supposition » affaibli le modèle de sécurité en y ajoutant une hypothèse, et il peut paraître surprenant de supposer une absence de collaboration entre adversaires alors qu'on tentait de mettre au point un protocol BDCMU spécifiquement pour des situations d'adversaires collaborant entre eux (utilisateurs et serveur). En fait, le type de collaboration est différent et le modèle de sécurité dans lequel DH-AP-MUSE est étudié, même avec cette hypothèse de non-collaboration entre deux serveurs, est beaucoup plus réaliste qu'un modèle où aucun utilisateur n'est supposé collaborer avec le serveur. Nous donnons trois raisons principales à cela : d'une part un fournisseur de nuage est habituellement beaucoup plus difficile à attaquer qu'un utilisateur isolé (ce que l'on a déjà mentionné plus haut), ce qui fait qu'un attaquant quel qu'il soit a des chances très faibles de prendre le contrôle de *deux* fournisseurs de nuages indépendants ; d'autre part il est tout à fait envisageable que les utilisateurs se cotisent pour un service d'audit et de contrôle des fournisseurs de nuage, assurant l'absence d'échange d'information non-autorisée entre les deux, alors que les utilisateurs peuvent être des entités éphémères et anonymes qu'on peut avoir du mal à identifier, et encore d'avantage à contrôler ; enfin le nombre très grand d'utilisateurs fini de rendre impossible leur contrôle, alors qu'il n'y a qu'un seul couple de serveurs à contrôler dans DH-AP-MUSE.

Mais les propriétés de sécurité de DH-AP-MUSE peuvent toujours paraître insuffisantes. En effet DH-AP-MUSE laisse l'adversaire obtenir certaines informations, sous la forme de ce que l'on appelle le *motif d'accès*, ou *access pattern* en anglais, dont on a déjà parlé plus haut. Pour mémoire, le motif d'accès consiste en l'information de quelle entrée correspond à quelle requête. La quasi-totalité des protocoles BDC à utilisateurs uniques laisse voir le motif d'accès et cela a longtemps été considéré comme une nécessité au nom de la performance ; quand aux protocoles multi-utilisateurs précédents, ils laissent en fait voir beaucoup plus que ce motif d'accès, comme nous l'avons déjà expliqué. Cependant depuis récemment, on remarque un grand nombre de travaux de recherche montrant des attaques toujours plus efficaces contre les protocoles BDC à utilisateur uniques utilisant le fait qu'ils révèlent le motif d'accès. On peut donc craindre que ce qui était considéré comme acceptable jusqu'alors en terme de fuite d'information ne le soit bientôt plus, d'autant plus que les conséquences en terme de sécurité semblent être généralement plus grandes dans un contexte multi-utilisateurs.

Il paraît alors nécessaire de travailler à concevoir des protocoles BDCMU qui, toujours dans un modèle de sécurité prenant en compte des utilisateurs corrompus, ne laisse pas l'adversaire voir le motif d'accès, possiblement au prix d'une complexité accrue et d'une performance moindre du protocole. Nous présentons un autre protocol de BDCMU, nommé RMÖ15, qui est construit à partir de DH-AP-MUSE auquel on applique une modification très simple. Cette modification assure une fuite d'information quasi-nulle mais au prix d'une charge de travail accrue pour les lecteurs dans le protocole.

Dans DH-AP-MUSE le serveur envoie les entrées qu'il a transformés au proxy, encodés sous forme de *filtres de Bloom* pour des raisons d'économie de mémoire et de bande passante. Le proxy, après avoir reçu et transformé une requête, applique cette requête à l'entrée correspondante en effectuant une recherche dans le filtre de Bloom correspondant à cette entrée. C'est en voyant le résultat de cette recherche que le proxy a accès au motif d'accès, l'information que l'on veut désormais protéger. Dans RMÖ15, le serveur chiffre les filtres de Bloom avant de les envoyer au proxy de façon à ce que le proxy ne puisse extraire aucune information de ces filtres. Le lecteur à la source de la requête, en revanche, connaît la clé utilisée dans le chiffrement des filtres et est donc capable d'effectuer cette recherche. Le proxy pourrait alors simplement envoyer chaque requête transformée et son filtre (chiffré) correspondant au lecteur qui effectuerait la recherche. Au lieu de cela et pour gagner en efficacité, le chiffrement des filtres de Bloom est fait d'une telle manière que le proxy soit capable de localiser et extraire les parties du filtre qui seront nécéssaire à la recherche. La quantité de données envoyées au lecteur pour chaque entrée est donc très petite, et le temps nécessaire pour le lecteur de finaliser la recherche dans cette entrée est quasi-nulle.

Nous montrons, toujours avec la même technique de preuve que pour DH-AP-MUSE, que RMÖ15 empêche les deux serveurs d'avoir accès au motif d'accès (seul le proxy y avait accès dans DH-AP-MUSE). À vrai dire, l'information révélée dans RMÖ15 est extrêmement limitée, et on montre qu'elle se résume à de l'information que l'on acceptait de révéler dès la définition de la notion de BDCMU (taille et nombre des entrées, autorisations…). On a donc bien atteint avec RMÖ15 l'objectif de réduire la fuite d'information présente dans DH-AP-MUSE. Mais cela est fait au prix de la capacité du protocole à supporter de large bases de données : car la charge de travail d'un lecteur est désormais proportionelle au nombre d'entrées sur lesquelles sont appliquées la requête. Même si le coût par entrée est extrêmement faible, la principale motivation à utiliser des protocoles BDCMU est la volonté de pouvoir chercher dans un très grand nombre d'entrées avec des machines utilisateurs aux ressources limitées. Ainsi cette concession du point de vue l'efficacité à grande échelle, même si elle permet d'atteindre des guaranties de sécurité très grandes, limite le protocole RMÖ15 à une certaine catégorie de situations et ne peut pas être une solution générale au problème des BDCMU. Nous avons présenté le protocole RMÖ15 à la conférence ISC 2015 [RMÖ15].

Un premier compromis entre protection du motif d'accès et efficacité à grande échelle

Nous présentons alors un troisième protocole BDCMU, nommé RMÖ18, qui est le premier protocole qui protège le motif d'accès tout en assurant aux lecteurs une charge de travail très faible, quelque soit le nombre d'entrées dans lesquelles ils veulent faire leur recherche. RMÖ18 a un niveau de sécurité presque égal à celui de RMÖ15, avec une charge de travail utilisateur presque égale à celle de DH-AP-MUSE. Ceci ce fait toutefois au prix d'une complexité accrue du protocole côté serveur, à la fois en terme de complexité théorique qui rend l'étude de sécurité plus ardue, et en terme de charge de travail.

Au départ de la conception de RMÖ18, il y a un apparent paradoxe : On veut que le proxy soit capable de filtrer les réponses négatives –les entrées ne correspondant pas à la requête– et de n'envoyer au lecteur que celles qui sont positives afin que la charge de travail du lecteur soit uniquement proportionelle au nombre d'entrées correspondant à la requête, comme c'est le cas dans DH-AP-MUSE et dans n'importe qu'elle base de données non chiffrée ; et en même temps, on ne veut pas que le proxy sache quelles entrées donnent lieu à des réponses positives car on veut garder le motif d'accès secret.

RMÖ18 résoud cet apparend paradoxe de la manière suivante : Dans RMÖ18, les filtres de bloom ne sont jamais envoyés au proxy ; le serveur les garde après les avoir construits. Le proxy va alors effectuer la procédure de recherche dans les filtres de Bloom à distance, via un protocole exécuté entre le proxy et le serveur qui limite l'information à laquelle le proxy a accès. Avant de pouvoir donner d'avantage d'explications, on doit revenir sur le fonctionnement des filtres de Bloom, et en particulier sur la façon dont on cherche un élément dans un ensemble encodé dans un filtre de Bloom : l'élément cherché est passé par une suite de fonctions de hachage, et le résultat de chacune de ces fonctions est interprété comme une position dans le filtre qui peut être vu comme une liste de valeurs binaires. Les valeurs stockées dans le filtre à ces positions sont lues et si elles sont toutes égales à un, on en déduit que l'élément recherché est dans l'ensemble encodé avec une grande probabilité. Si en revanche une des valeurs au moins est un zéro, on est certain que l'élément recherché n'est pas dans l'ensemble.

Dans RMÖ18, le proxy accède au filtres de Blooms hébergés chez le serveur via un protocole de *transfert opaque* (*Oblivious Transfer protocol* en anglais) qui guaranti que le proxy ne voit pas le contenu des valeurs du filtre autres que celles nécessaires à cette procédure de recherche. De plus, on utilise une variante des filtres de Bloom appelé *filtres de Bloom opaques* (*Garbled Bloom Filters* en anglais), inventés par Dong et al. [DCW13], qui assure qu'avec ces seules valeurs le proxy ne puisse rien savoir sur les autres éléments présents dans le filtre. Enfin on modifie légèrement la construction des filtres de Bloom opaques et on adapte le protocole de transfert opaque utilisé pour arriver à notre objectif : le proxy reçoit un ensemble de réponses dont il peut savoir si chacune est positive, auquel cas il la transmet au lecteur, ou négative auquel cas il la supprimme, mais le proxy ne peut rien dire d'autre sur une quelconque réponse, et en particulier il ne peut pas savoir à laquelle des entrées examinées elle correpond. La Figure 7 illustre ce fonctionnement.

On ne rentre pas ici dans les détails des modifications que l'on a du appliquer au solutions existantes de protocoles de transfert opaque et de filtres de Bloom opaques, mais la difficulté réside dans le fait de les modifier pour les rendre compatible avec cette propriété très spécifique, que ne satisfont pas les constructions originales, sans perdre les propriétés premières de ces constructions qui restent nécessaires à la sécurité de notre protocole.

Nous démontrons les propriétés de sécurité de RMÖ18 de la même manière que pour les précedents protocoles. Nous montrons que, en plus de l'information qualifiée de « bénigne » que nous acceptons de révéler dès la définition de la notion de BDCMU (et qui est révélée par tous les protocoles BDC existants), RMÖ18 ne révèle que le *nombre* d'entrées correspondant à chaque requête, en gardant secret l'identité de ces entrées. Ceci suffit à empêcher toutes le attaques existantes basées sur le motif d'accès. Du point de vue de la charge de travail et de la performance côté serveur, l'essentiel des coûts vient du protocole de transfert opaque. Nous montrons que des progrès récents dans ce domaine devraient nettement augmenter les performances de RMÖ18, et l'implémentation de RMÖ18 avec ces nouvelles solutions serait un sujet d'étude intéressant pour le futur.

Le protocole RMÖ18 a été présenté au Workshop SCC à AsiaCCS 2018 [RMÖ18b].

Correction de l'Analyse de Sécurité des Filtres de Bloom Opaques

Les filtres de Bloom opaques, dont notre protocole RMÖ18 dépend, sont une construction récente qui a eu un impact considérable dans la construction de protocoles pour la sécurisation de l'informatique dans les nuages. Or nous remarquons qu'un des arguments dans l'analyse de sécurité originale de cette construction semble douteux. Puis nous montrons que cet argument est faux, et nous en dérivons une large classe de contre-exemples qui infirment la preuve donnée dans l'analyse.

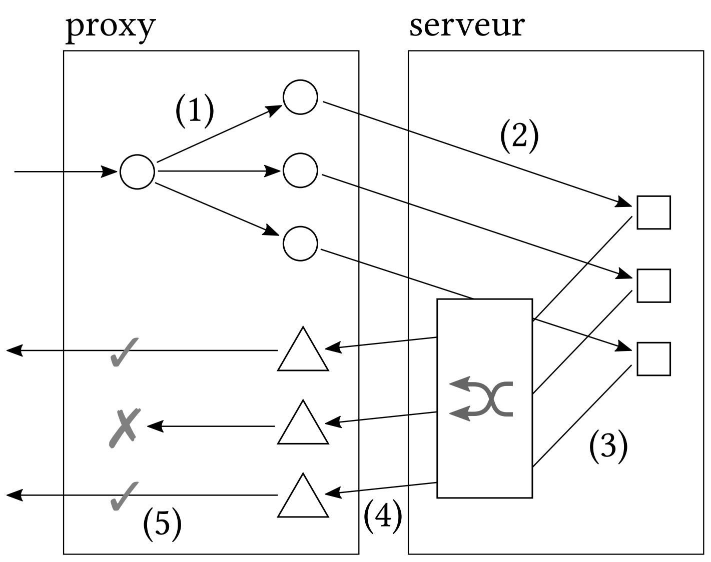


Figure 7 : Une illustration du fonctionnement de RMÖ18

Après la transformation des requêtes encryptées (1) et leur transfer vers le serveur pour leur application sur les entrées préparées (2) Les réponses correspondantes sont envoyées dans un ordre aléatoire (3 et 4, où le bloc entre 3 et 4 représente le ré-ordonnement aléatoire). Grâce à la façon dont RMÖ18 est construit, le proxy ne peut pas savoir quelle réponse correspond à quelle requête ou quelle entrée. Il est seulement capable de filtrer les réponses négatives et de transférer les réponses positives (5).

Ceci met en danger la sécurité d'un certain nombre de protocoles. Nous montrons également que le même défaut se retrouve dans l'analyse de sécurité d'autres constructions de filtres de Bloom opaques. Heureusement une des constructions alternative de filtre de Bloom opaque, due à Rindal et Rosulek [RR17], ne semble pas affectée par ce problème ; toutefois cette construction alternative ne convient qu'à certains cas, et en particulier dans le cas de notre protocole RMÖ18 ne peut pas être utilisée pour remplacer la construction originale. Heureusement nous sommes à même de fournir une nouvelle preuve que la construction originale des filtres de Bloom, celle dont RMÖ18 dépend, satisfait bien les propritétés de sécurités revendiquées. La sécurité de RMÖ18 n'est donc pas remise en question.

Nos travaux sur la sécurité des filtres de Bloom opaques ont été présentés à la conférence DBSec 2018 [RÖ18].

Conclusion

Concevoir des protocoles BDC qui sont à la fois performants et sécurisés s'est révélé une tâche difficile depuis l'apparition de ce sujet de recherche en 2000. Étendre la notion de BDC à un contexte multi-utilisateur ne peut l'être que plus. C'est ce que nos travaux au cours de cette thèse ont contribuer à montrer en démontrant que la voie dans laquelle les travaux précédents sur le sujet se sont engagés ne semble pas pouvoir mener à une solution satisfaisante, et qu'une telle solution demanderait de reprendre la travail de conception presque du début.

Cependant nous montrons aussi que d'autres voies sont possibles, et que l'on peut atteindre une guarantie de sécurité raisonable –principalement, un certain niveau de sécurité en présence d'utilisateurs corrompus– sans nécessairement faire de trop grands sacrifices sur le domaine de la performance et du coût. Ainsi on présente trois protocoles de BDCMU qui sont les trois premiers protocoles de ce type dont les propriétés de sécurité restent valides en présence d'utilisateurs corrompus. Ces trois protocoles présentent trois compromis différents entre sécurité et performance : DH-AP-MUSE est le plus performant, sans doute plus rapide même que certaines solutions de l'état de l'art. Cependant, même si la quantité d'information qu'il révèle est moindre que les protocoles de l'état de l'art, de récents progrès en maitère d'attaques sur les protocoles BDC poussent à vouloir réduire encore l'information révélée. Le protocole RMÖ15 est celui qui révèle le moins d'information, mais cela se fait au prix d'une lourde charge de travail pour les utilisateurs dans le cas de larges bases de données. RMÖ15 ne conviendrait donc qu'à des situations où la base de donnée est d'une taille limitée, ou avec des utilisateurs ayant des ressources assez conséquentes, et une très forte demande de sécurité. Enfin, RMÖ18 est à la fois immunisé contre les attaques basées sur le motif d'accès en révélant presque aussi peu d'information que RMÖ15, tout en ayant une charge de travail utilisateur proche de celle de DH-AP-MUSE. Malgré une certaine complexité et lenteur du protocole côté serveur, qui devrait se réduire grâce à de récents progrès, nous voyons RMÖ18 comme le meilleur candidat actuel pour des solutions de BDCMU. La Figure 8 résume ceci en situant tous les protocoles BDCMU existants, regroupant à la fois ceux conçu durant cette thèse et ceux déjà existants, au regard du compromis entre sécurité et performance qu'ils offrent.

Nos travaux sur les protocoles BDCMU ont aussi été l'occasion de contribuer à l'étude et l'amélioration d'autres constructions cryptographiques comme les filtres de Bloom opaques.

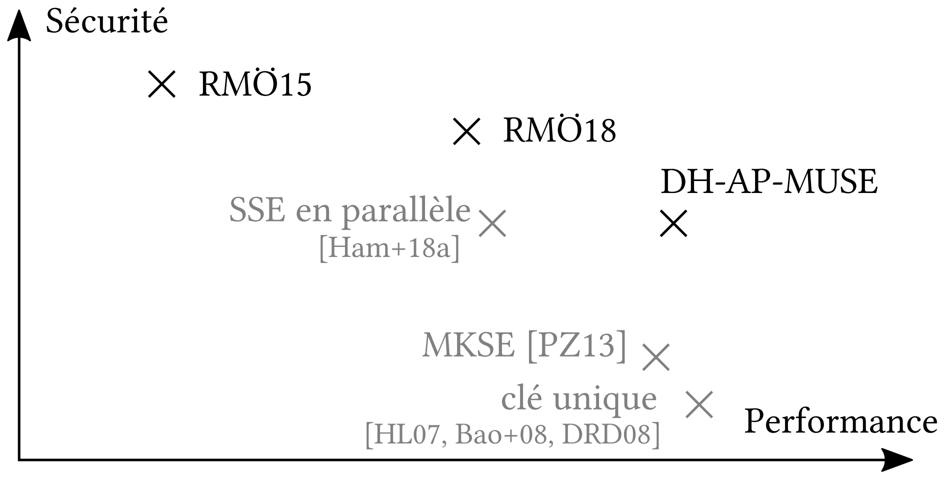


Figure 8 : Un graphe situant tous les protocoles BDCMU à la fin de cette thèse au regard du compromis entre sécurité et performance qu'ils offrent.

Contents

Remerciements iii

Résumé en Français v

1 Introduction 3

2 Preliminaries 11
- 2.1 General Notations 11
- 2.2 Hard Problems and security proofs 12
- 2.3 Cryptographic Primitives 13

3 Multi-Reader Multi-Writer Searchable Encryption 17
- 3.1 Motivations 17
- 3.2 Definitions 19
- 3.3 State of the Art 24
- 3.4 Leakage Abuse Attacks against Searchable Encryption 30

4 The Dilemma of MUSE: On the difficulty of trading privacy for efficiency 35
- 4.1 The Case for User-Server Collusions 35
- 4.2 Insecurity of Iterative Testing Against User-Server Collusions 36
- 4.3 Leakage-Abuse Attacks Applied to MUSE 39
- 4.4 Lessons from the MKSE Protocol 43
- 4.5 The Dilemma of MUSE 48

5 A First Generation of Secure MUSE protocols: choosing between access pattern leakage and scalability issues 51
- 5.1 Record Preparation and the use of Two Non-Colluding Servers 51
- 5.2 The DH-AP-MUSE protocol: fast MUSE with Access Pattern Leakage 55
- 5.3 The RMÖ15 protocol: Very low leakage at the price of scalability 63
- 5.4 Conclusion 69

6 A Secure yet Scalable MUSE protocol 73
- 6.1 Result Filtering, Result Length Leakage and Response Unlinkability 73
- 6.2 New primitives for response unlinkability 74
- 6.3 The RMÖ18 MUSE Protocol 82
- 6.4 Further Remarks and Implementation 91
- 6.5 Conclusion 91

7 On the security of Garbled Bloom Filters **95**
7.1 Original proof of security by Dong et al. [DCW13] 95
7.2 Invalidation of the proof in [DCW13] 96
7.3 Other GBF constructions 99
7.4 New Proof of Security 100
7.5 Related Work 102
7.6 Conclusion 103

8 Conclusion **105**

Bibliography **107**

List of Symbols Used **117**

List of Figures **119**

List of Tables **121**

List of Algorithms **123**

1 Introduction

Multi-User Searchable Encryption: Privacy-Preserving Search on Aggregated Data The cloud computing paradigm, where users outsource a large portion of computation and data storage tasks to powerful third parties called Cloud Service Providers (CSPs), became one of the major revolutions in IT during the last decade and is now ubiquitous: from smartphones and consumer computers to large enterprises, most of today's IT relies at least partly on a cloud infrastructure. Despite all the benefits cloud computing brings in terms of flexibility, availability and cost efficiency, the rise of the cloud computing paradigm brings the problem of protecting users against a potentially compromised CSP.

Many new technological solutions appeared that aim at increasing the robustness of CSPs against attackers, such as encryption at rest and process isolation mechanisms. But cryptographic protocols can allow a stronger level of security guarantee by protecting data and/or programs against an adversary having access to all the information of the CSP, or even an adversary controlling the CSP entirely.

Searchable Encryption (SE) is such a family of cryptographic protocols. SE enables searching on data hosted on a remote server, in the fashion of a database system, while protecting the privacy of the data and the queries that are executed against the potentially malicious server. More precisely, in a SE system a user encrypts a database with a key that is only known to him, then uploads the encrypted database to some potentially malicious server. The user can then perform various database operations on the outsourced data, from simple keyword search operations for the simplest SE systems (such as *"which database records contain the keyword 'urgent' ?"*) to queries evaluating complex range and boolean combination operations for the most evolved ones. The security properties of a SE protocol are meant to ensure that the server cannot recover the content of the database or the queries during the whole lifetime of the system, which comprises the upload phase but also the period where the database is being used.

This does not mean that the server cannot get any information about the data or the queries, as it is now accepted that no SE system can leak no information at all and be efficient at the same time. Nevertheless a SE system must guarantee that, under the expected conditions of operations and the expected capacities of the adversary, the amount of information the adversary can obtain even after a long period of activity of the system is sufficiently small.

Research on SE began with the paper of Song et al. [SWP00]. Then, several papers were published which main contributions were to propose definitions of security for SE protocols, until the seminal work of Curtmola et al. [Cur+06] that shows the limitations of previous definitions and presents a definition that is still considered as the reference. The paper of Curtmola et al. [Cur+06] also presented the first SE system where the running time of a search operation is linear with the number of matching database records instead of being linear with the total number of records in the database. From then, SE systems were improved in many different directions such as efficiency and scalability, but also query expressiveness —allowing range queries and boolean combinations—. Some SE schemes were also proposed that allow several users to add records to the database (such as the "Public-key Encryption with Keyword Search" (PEKS) family of protocols), or where several users are able to search it (sometimes called *Delegated Word Search* protocols).

A natural direction for the evolution of SE protocols is actually settings where there are many users, some having data to upload to the server and some willing to search this data after authorization of the rightful owners. One reason is the fact that very large databases —one of the main focus of research on SE— often consist in aggregated data coming from many different owners; another reason is the regulations (typically the US HIPAA and the European GDPR) that force companies to enforce the right of individuals to control their information. This motivates research on Multi-Reader-Multi-Writer Searchable Encryption protocols, which we call Multi-User Searchable Encryption (MUSE) protocols for short in this manuscript. A MUSE system comprises many users divided in two categories, the readers and the writers. Each writer has some data segments (called *records*) that she wants to store at the (untrusted) server, and each writer choses which reader should be able to search which of her records.

The first SE protocol that can be considered as a MUSE protocol was presented by Hwang and Lee in 2007 [HL07], but the protocols of Bao et al. [Bao+08] and Dong et al. [DRD08] are the first ontes to really match our definition of MUSE. Then, a major evolution took place in MUSE as a research topic with the first paper presenting a threat model where users are considered as a threat as well, by Popa and Zeldovich [PZ13]. Such threat model departs from the ones previously used where only the server was considered as a potential threat. All other MUSE protocols designed before this PhD study are built on top of the ones we just mentioned (as will be explained in Section 3.3 on the state of the art) with only minor differences.

Weakness of Existing protocols against corrupted users Considering the presence of some corrupted users colluding with the server in the threat model of MUSE was a necessary step and must become the standard in research on MUSE. Several facts make it quite unnatural to consider the server as the only threat: the large number of users, the fact that readers do not have access to data until the owner gives them an explicit authorization, but also the common wisdom that end users are often the weakest point of a system regarding security. Moreover, in situations where one can assume that users will not misbehave, such as the case of a company where users are the employees, it is likely that a single-user SE system can be used instead. Because single-user SE protocols are inherently more efficient than MUSE ones, and because this difference is even stronger in practice (state-of-the-art single-user SE protocols are extremely fast), such situations are not a proper use case for MUSE protocols. Hence it appears necessary to evaluate the security of MUSE protocols in a threat model that takes into account some users being under the control of the adversary.

Unfortunately, the large majority of existing MUSE protocols cannot protect the privacy of data and queries in the face of even a small number of corrupted users. Interestingly, we discovered a design pattern that is shared among all MUSE protocols existing before the beginning of this PhD study which, alone, makes all these protocols inherently weak against user-server collusions. This design pattern does not seem to be avoidable with just minor modifications. It it thus necessary to completely re-think how MUSE protocols are being built in order to achieve privacy in a realistic threat model.

Furthermore, the MUSE protocol proposed by Popa and Zeldovich in [PZ13], named MKSE for "Multi-Key Searchable Encryption", follows this same design pattern. As a result it has the same privacy issue as the other MUSE protocols, even though the aim of this protocol was explicitely to achieve privacy against user-server collusions. Hence, while the need to address collusions in MUSE was already acknowledged in the research community at the beginning of this PhD study, no constructions existed that reached this privacy goal.

Our Contributions Our first contribution is the discovery and the study of this design pattern in MUSE protocols, which we call the *iterative testing structure*, and its consequences on the privacy properties of these protocols in the face of collusions. After explaining in Section 4.1 why considering users as part of the threat appears to be a necessity in MUSE, we describe in detail in Section 4.2 the *iterative testing structure* and how it prevents MUSE protocols following it to provide any privacy against user-server collusions. Intuitively, in iterative testing the server applies a query on an encrypted record by testing encrypted keywords one by one. The server is thus able to see when two queries originating from different readers correspond to the same keyword if these two queries match the same record. The server does not see the plain text immediately, but rather relations between various queries and records. Then, after the server processed sufficiently many queries, the information from even a single corrupted user is enough, due to these relations known to the server, to recover large amounts of plaintext across the whole database, as well as to recover a large number of queries. While a small amount of information being revealed to the server during the processing of queries is common among SE protocols, we show that the amount of information leaked in the affected MUSE protocols, and the consequences of such leakage on privacy, are much more serious than what is considered as "normal" in SE.

In Section 4.3.3 we show how existing work on a type of attack on SE protocols named *leakage-abuse attacks* and only applied to the single-user setting so far can be extended to the multi-user setting to better understand and detect this type of issues in MUSE protocols. In Section 4.4 we prove that the issue we identified does affects the MKSE protocol —despite it being designed specifically against user-server collusions— by implementing a practical attack against a software named Mylar that implements MKSE. We also study the reasons for the security analysis of [PZ13] not to have helped noticing this issue. Finally, we show how our results apply to the arguments Popa and Zeldovich gave in response to critics from other researchers against MKSE.

The work we present in Chapter 4 was published at the 2017 PET Symposium in [RMÖ17], and was cited by at least one further paper on MUSE in a major conference [Ham+18a] as a motivation for their work.

We then design new MUSE protocols in Chapters 5 and 6 which we show protect the privacy of outsourced data and search queries against any number of users colluding with the CSP. We present three MUSE protocols (most having been improved incrementally in several "versions"), each having a different combination of privacy and efficiency level, suiting different situations. A core technique that is used in all the protocols we present is the use of two servers that are assumed not to collude. As we explain in Section 5.1.2, the slight weakening of the threat model caused by such assumption seems entirely worth it if it enables the design of MUSE protocols that are secure against user-server collusions. Another technique that we introduce is the notion of *record preparation* where for each reader being authorized to search a record, a copy of the record is created by the server that will solely be used for queries coming from this reader. This copy is sent to the second server (called the *proxy*) which is in charge of query application. Recent work by Hamlin et al. [Ham+18a] show that there may be solutions to avoid the storage overhead caused by duplicating records for every authorization, but such solutions are still far from being practical as the one of Hamlin et al. depend on primitives that are still only theoretical (namely, cryptographic obfusaction). We then show how the techniques we present can be used to build MUSE protocols that are truly secure in a threat model that includes users.

The first protocol that we present, named DH-AP-MUSE (Section 5.2), is the most efficient of all three protocols designed in this thesis. It is also the one that leaks the greatest amount of information, although the amount of information it leaks is much lower than the one of any MUSE protocol in the prior art, and corresponds to a profile of leakage which consequences on privacy are much better understood for being studied extensively in the single-user setting. DH-AP-MUSE is also the simplest of all three protocols and, interestingly enough, the one that was designed last. In DH-AP-MUSE, a writer encrypts her records with some secret key only known to her at first but which she sends to the proxy (which, recall, is the name of the second server). In a symmetric fashion, a reader will use some secret key for encrypting her queries but this secret key is sent to the server (which is different from the proxy and does not collude with it). The server will use the reader keys it receives to create “transformed” copies of the encrypted records (we call this step *record preparation*) which the reader sends to the proxy; in a symmetric fashion again, the proxy will apply an analogous transformation to the encrypted queries it receives using the record keys it received. As a result, the proxy obtains transformed queries that it can search for in the prepared records that were sent by the server. In this protocol that strongly resembles a Diffie-Hellman Key Exchange protocol, the fact that two queries from different readers will never be applied on the same (prepared) record, together with the assumption that the server and the proxy do not collude, ensure that the issue we identified in other MUSE protocols does not appear. Finally we formally show that the DH-AP-MUSE protocol provides strong privacy guarantees in presence of an arbitrary number of corrupted users by using the “real/ideal” methodology, also called “simulation technique”, which is the standard technique for security analyses in SE. To demonstrate the simplicity and the efficiency of the DH-AP-MUSE protocol, we implement its algorithms in less than 100 lines of C code and we report on performance measurements on this implementation.

The second protocol, named RMÖ15 (Section 5.3), adds a few mechanisms on top of DH-AP-MUSE that dramatically reduce the amount of information leaked by the protocol. The prepared records received by the proxy have a second layer of encryption that makes the proxy unable to complete the search procedure. The proxy then sends the querying reader a (small) encrypted response for each record that was searched. The reader is able to decrypt it, while the proxy gets no information about the result of the search. This however is at the price of scalability issues, as the workload of a reader in RMÖ15 grows linearly with the size of the number of records searched. The RMÖ15 protocol is then at the opposite end of the privacy-efficiency dilemma compared to DH-AP-MUSE, having the smallest possible leakage while having scalability issues. We presented the RMÖ15 protocol at the 2015 ISC conference in [RMÖ15] and realized a proof-of-concept implementation of it for the "User Centric Networking" European research project [UCN].

We then present in Chapter 6 a third protocol, named RMÖ18, that tries to get the best of both previous protocols: RMÖ18 leaks much less information than DH-AP-MUSE, almost reaching the leakage profile of RMÖ15; but it does not have the scalability issues of RMÖ15. More precisely, regarding scalability, user workload is almost the same in RMÖ18 as in DH-AP-MUSE (recall, DH-AP-MUSE is the most efficient of all our protocols); and however the server workload is higher in RMÖ18 than in DH-AP-MUSE, we give many techniques that help reduce this workload, and recent new improvements in Private Information Retrieval (PIR) protocols should make the server-side cost of RMÖ18 even lower. Intuitively, in RMÖ18 prepared records are kept by the server. To search these prepared records, the proxy uses some privacy-preserving protocol that we designed. This protocol assures that the proxy gets no information beyond the number of records that matched the query, while the server gets no information at all. We presented the RMÖ18 protocol as the 2018 SCC workshop at AsiaCCS in [RMÖ18b], and provided a proof-of-concept implementation of it to the TREDISEC European research project [TREDISEC].

Along the chapters of this manuscript, we use a graph that locates MUSE protocols regarding their efficiency and privacy levels which we fill progressively: first with the protocols in the state of the art, then with the protocols that we design. The final version of this graph is reproduced hereafter in Figure 1.1 (the protocols that we designed are in black, the others are in gray). We see this graph as a contribution in its own and hope that it can help give a better overview of the state of the art as it is after this thesis. Note that the evaluation of the privacy and efficiency level of schemes is more qualitative than quantitative in this graph.

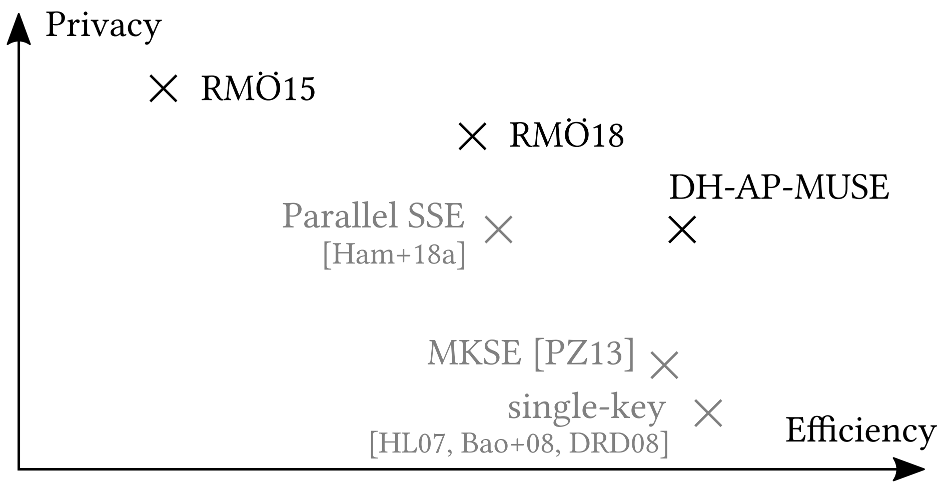


Figure 1.1: A graph locating all MUSE protocols as of the end of this thesis regarding the privacy-efficiency trade-off they achieve.

The design of the protocol run between the proxy and the server in RMÖ18 required new primitives, or rather the modification of existing primitives to give them specific security properties, and these modifications could be of independent interest. Also while studying these primitives, we noticed a flaw in the security proof of one of them named Garbled Bloom Filters (GBFs). This flaw was threatening the security of our entire protocol, but could have more widespread consequence due to the popularity of this recent data structure in research on cryptographic protocols for cloud security. Thankfully we were able to come up with a new proof for the security property of GBFs, which also applies to the variant we designed for the needs of RMÖ18. Our results regarding both the flaw and the new proof are presented in Chapter 7, and were published at the 2018 DBSec conference in [RÖ18].

Summing up, we think our work showed the need to depart from the previous techniques and designs for MUSE protocols in order to achieve privacy in a more realistic threat model, and we think that it paved the way for this revolution in MUSE by building the first MUSE protocols reaching this new level of privacy.

2 Preliminaries

In this chapter we introduce the fundamental cryptographic concepts that we use in this thesis. Following the principles of *provable cryptography* (see Section 1.4 of [KL14]), security properties in this thesis are shown by building a complexity reduction from some problem that is widely believed to be hard to the problem consisting in breaking the said property.

After setting some general mathematical notations, we present the only fundamental computationally hard problem we need in the security proofs of thesis, namely the Decisional Diffie-Hellman (DDH) problem. We also present the proof technique that is being used in this thesis, often called the *simulation-based technique*, that we use to define security in MUSE, and the *honest-but-curious* model in simulation-based proofs.

Finally we present the fundamental cryptographic building blocks that are used for the design of cryptographic protocols in this thesis.

2.1 General Notations

With n a positive integer, we will use the notation $[n]$ to represent the sequence $(1, 2, \ldots, n)$. When A is an algorithm, $y \leftarrow A(x)$ means that y is assigned the output of A run on input x. When X is a (finite) set, $x \overset{\$}{\leftarrow} X$ means that x is sampled randomly from X. Unless otherwise stated, the sampling is done following a uniform distribution.

A bit is an element of $\{0, 1\}$, and a bit string of length n is a sequence of n bits. The set of all bit strings on length n is noted $\{0, 1\}^n$ and the set of all bit strings of finite length is noted $\{0, 1\}^*$.

We will denote by λ the security parameter, which is used to bound the running time of an adversary. All algorithms are programs for probabilistic Turing machines which we require to run in time polynomial in the security parameter. We call such machines Polynomial-time Probabilistic Turing machines or PPT. Cryptographic algorithms must be efficient, meaning that they must run in a time that is a polynomial of the security parameter, and their security properties are defined as tasks that cannot be performed by any adversary in the threat model running in time polynomial in the security parameter. All algorithms are run by probabilistic Turing machines.

We recall the definition of a negligible function, which is used throughout this manuscript:

Definition 2.1.1 (Negligible Function)**.** A function $f : \mathbb{N} \to \mathbb{R}$ is *negligible* if for every positive integer c there exists an integer N_c such that for all $x > N_c$,

$$|f(x)| < \frac{1}{x^c}$$

2.2 Hard Problems and security proofs

Hard Problems The security properties of the protocols presented in this manuscript are shown using algorithmic reductions to computational problems that are widely believed to be hard or to properties of other cryptographic protocols and primitives. The only fundamental hard problem used in this manuscript is the *Decisional Diffie-Hellman* (DDH) problem, which we describe hereafter.

Definition 2.2.1 (Decisional Diffie-Hellman problem). Let $\mathbb{G}$ be a cyclic group of prime order ζ generated by $g \in \mathbb{G}$. The DDH problem consists, for an adversary $\mathcal{A}$ and for $(a, b, c) \overset{\$}{\leftarrow} \mathbb{Z}_\zeta^3$, to distinguish between (g^a, g^b, g^{ab}) and (g^a, g^b, g^c). Formally, we define the advantage of $\mathcal{A}$ as:

$$\mathsf{Adv}^{\mathsf{DDH}}_{\mathbb{G},\mathcal{A}}(\lambda) = \left| Pr[(a,b) \overset{\$}{\leftarrow} \mathbb{Z}_\zeta^2 : \mathcal{A}(g^a, g^b, g^{ab}) = 1] - Pr[(a,b,c) \overset{\$}{\leftarrow} \mathbb{Z}_\zeta^3 : \mathcal{A}(g^a, g^b, g^c) = 1] \right|$$

The hardness of the DDH problem on some group implies the hardness of the Computational Diffie Hellman (CDH) problem on the same group, where the adversary given (g^a, g^b) must compute g^{ab}, which in turn implies the hardness of the discrete logarithm problem (DLP) on the same group, where the adversary given g^a must compute a. Many groups believed to be DDH-hard are in wide use nowadays, some based on integers and some on elliptic curves.

Computational Indistinguishability We will use the notion of *computational indistinguishability* of two probability ensembles. We take the definition from Definition 7.30 of [KL14].

Definition 2.2.2 (computational indistinguishability). Two probability ensembles $\mathcal{X} = \{X_n\}_{n \in \mathbb{N}}$ and $\mathcal{Y} = \{Y_n\}_{n \in \mathbb{N}}$ are *computationally indistinguishable*, denoted $\mathcal{X} \overset{c}{\equiv} \mathcal{Y}$, if for every probabilistic polynomial-time distinguisher D there exists a negligible function negl such that:

$$\left| Pr_{x \overset{\$}{\leftarrow} X_n}[D(1^n, x) = 1] - Pr_{y \overset{\$}{\leftarrow} Y_n}[D(1^n, y) = 1] \right| \leq \mathsf{negl}(n)$$

Unless explicitly stated otherwise, by *indistinguishable* we mean *computationally indistinguishable* in the rest of this manuscript. Also and as it is usually done in the literature, we will often speak of indistinguishability of *probability distributions* rather than of *probability ensembles*, since the indexing of the probability distributions by the security parameter will be obvious. Said differently, the security parameter λ will take the role of the n variable in the definition.

Simulation-based Proofs Most security properties in this manuscript are shown using a technique called the "real/ideal model" or the "simulation technique". For a detailed description of this technique, see [Lin17]. Hereafter we give an overview of its functioning. The simulation technique, like most proofs in cryptography, is based on (theoretical) experiments where we bound the chance of some Turing machine to perform some task successfully. In the simulation technique, this tasks consist in deciding whether some data is the real view of the adversary during the execution of the studied protocol, or the output of some Turing machine called the "simulator". If one can show that these two situations are indeed indistinguishable, one can then conclude that the amount of information the adversary can recover in a real execution is no greater than the information that was received by the simulator. For a concrete example of how security is defined through a simulation experiment, see Definition 3.2.3.

Honest-but-curious Adversaries In this thesis we model the adversaries as honest-but-curious, also called "semi-honest" adversaries, as defined in Section 4 of [Lin17]. Intuitively, a honest-but-curious adversary will not deviate from the protocol specification, but will still try to get as much information as it can. Technically, it means that algorithms can be run on behalf of the adversary during the simulation. We justify the use of a honest-but-curious adversary in our definition of security in Section 3.2.2.

Hybrid Argument In the proofs we will sometimes refer to the *hybrid argument* or mention the use of a sequence of *hybrid distributions*. Simply put, the hybrid argument is a technique for proving indistinguishability that consists in defining a polynomial-length sequence of distributions called "hybrid" that bridge between two "extreme" distributions, then to show that distinguishing *any* two consecutive hybrid implies breaking some hard problem. This result in the two extreme distributions being indistinguishable. See [KL14] for more details.

2.3 Cryptographic Primitives

In this section we mention the basic cryptographic building blocks that are used in this manuscript.

Cryptographic Hash Functions and the Random Oracle Model Hash functions are defined in Chapter 5 of [KL14] as *"functions that take input of some length and compress them into short, fixed-length outputs"*. Their primary requirement is to avoid *collisions* where two different inputs are mapped to a same output. A usual property of cryptographic hash functions is collision resistance consisting in the hardness of finding (x, y) such that $x \neq y$ and $H(x) = H(y)$, H being the hash function. Weaker properties exist such as *preimage resistance* and *second-preimage resistance* (see Section 5.1.2 of [KL14]). In all the security analyses in this thesis, we use a methodology called the *Random Oracle Model* or ROM. We use the overview of the ROM given by Bost in [Bos18] and we refer to Section 5.5 of [KL14] for more details on the topic:

> The Random Oracle Model (or ROM), formally introduced by Bellare and Rogaway in [BR93], is a computational model where all parties have access to a (public) random oracle. As its name indicates, a random oracle outputs a random string for every new input it is given.
>
> To prove the security of some schemes in the ROM, we often use an additional feature, called programmability. This feature allows the games for pre-programming the output of the random oracle on some inputs, in a way that the programmed random oracle is indistinguishable from a regular random oracle.
>
> The ROM is a useful tool to show the security of some schemes. However, in practice, random oracles cannot exist (they would require an infinite description), and are often instantiated using hash functions. Actually, there is much debate among cryptographers on the quality of the ROM as an abstraction to analyze the security of cryptosystems [KM15]. Yet, for applied and real-world cryptography, it is a widely accepted and widely used model, as there is no convincing evidence that ROM-protocols have non-theoretical security weaknesses.

Cryptographic Bilinear Pairings None of the new protocols we present in this manuscript use bilinear pairings, but earlier versions of these protocols did make use of them. Pairings are also used in many other MUSE protocols. Let $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ be three cyclic groups of order ζ and let g_1, g_2 and g_T be their respective generators. Then $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a *bilinear pairing* if it is:

- efficiently computable;
- non-degenerate: if x_1 generates $\mathbb{G}_1$ and x_2 generates $\mathbb{G}_2$ then $e(x_1, x_2)$ generates $\mathbb{G}_T$;
- bilinear: $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}\ \forall (a, b) \in \mathbb{Z}_\zeta^2$

Throughout the manuscript, we will write h_e a cryptographic hash function to $\mathbb{G}_1$.

IND-CPA Encryption Another fundamental cryptographic building block that is used in this thesis is an encryption scheme that satisfies the *INDistinguishabilty under Chosen-Plaintext Attack* (IND-CPA) property. A symmetric encryption scheme is a triple of algorithms $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ such that KeyGen takes as input 1^λ (denoting the security parameter in unary form) and outputs a key k, Enc takes as input a key k and a message $m \in \{0, 1\}^*$ and outputs a ciphertext c, and Dec takes as input a key k and a ciphertext c and outputs a message m or an error. Such scheme is *correct* if for every λ, for every k output by $\mathsf{KeyGen}(1^\lambda)$ and for every $m \in \{0, 1\}^*$, it holds that $\mathsf{Dec}_k(\mathsf{Enc}_k(m)) = m$

The IND-CPA security property for symmetric encryption schemes is defined by the following experiment (See Definition 3.22 of [KL14]):

- A key k is generated by running $\mathsf{KeyGen}(1^\lambda)$
- The adversary $\mathcal{A}$ is given input 1^λ and oracle access to $\mathsf{Enc}_k(\cdot)$, and outputs a pair of messages m_0, m_1 of the same length.
- A uniform bit $b \in \{0, 1\}$ is chosen, and then a ciphertext $c \leftarrow \mathsf{Enc}_k(m_b)$ is computed and given to $\mathcal{A}$.
- The adversary $\mathcal{A}$ continues to have oracle access to $\mathsf{Enc}_k(\cdot)$, and outputs a bit b'.
- The output of the experiment is defined to be 1 if $b' = b$ and 0 otherwise. In the former case, we say that $\mathcal{A}$ *succeeds.*

A symmetric encryption scheme is said to be IND-CPA-secure if the probability of success of the adversary is negligibly close to $1/2$.

3 Multi-Reader Multi-Writer Searchable Encryption

In this chapter we present the problem that is studied in this thesis, *Multi-User Searchable Encryption* (MUSE). After giving the motivations for its study (motivating both study on Searchable Encryption and on its extension to the multi-user setting), we give a syntax for MUSE systems and we give definitions for their desired properties. We then review the state of the art of research on searchable encryption (both in the multi-user setting and in the original single-user one) and on a class of attacks against searchable encryption that will be of importance in the thesis.

3.1 Motivations

Progress in networking, hardware virtualization and distributed algorithms led to the advent of *cloud computing*. Amazon, the company that started the rise of cloud computing, defines it on its website [AWS] as *the on-demand delivery of compute power, database storage, applications, and other IT resources through a cloud services platform via the internet with pay-as-you-go pricing*. A date that is often cited as the beginning of the cloud computing era is the launch of Amazon EC2 in 2006. Nowadays, the cloud computing paradigm is ubiquitous, from companies of all sizes that outsource most of their Information Technology (IT) infrastructure, to smartphones and other small *Internet of Things* devices that use the cloud to compensate for their limited capacities. [Dav+17] provides an overview of the cloud computing market and some predictions on it.

The rise of the cloud computing paradigm can be explained by the numerous and strong benefits it provides: economies of scale, flexibility, availability and the opportunity to focus on one's core business. However cloud computing also brings novel security and privacy issues. While Cloud Service Providers (CSPs) have, in general, better expertise on cyber security than most of their clients thanks to concentration and specialization, the fear of security and privacy issues is still one of the major, if not the main obstacle to an even wider adoption of cloud computing. The main reasons we see for this fear are the lack of control on the infrastructure and the lack of accountability of the CSP, the increased attack surface caused by the massive use of the public Internet, and the concentration of value at CSPs making them very attractive targets for attackers.

These growing cyber-security threats coming from the wide adoption of cloud computing motivated research on cryptographic protocols that would reduce the risk of a partial or total corruption of the CSP. Among the many objectives that are interesting for cryptosystems for the cloud, the one this thesis is concerned with is privacy. The most cited motivation for studying privacy in cloud computing is the growing use of IT and data in health, with the typical use case of a hospital wanting to outsource as much of its IT operations as possible while having to comply with policies like the Health Insurance Portability and Accountability Act [HIPAA] and the European GDPR

[GDPR]. Other objectives that are actively researched in cloud security are verifiability and proof of retrievability.

A very simple way to ensure privacy against an untrusted CSP is to encrypt data before sending it to the CSP. This is often referred to in the folklore as *user-side encryption*. However, using a traditional encryption procedure like AES [AES] or ChaCha20 [ChaCha20], such technique would make the CSP unable to perform any computation with the encrypted data. This would result in a degraded service, since a lot of the added value of a CSP comes from its ability to process the outsourced data.

This problem, which has been studied even before the advent of cloud computing, led to many research topics that are still very active as of today. The most famous one is probably Fully Homomorphic Encryption (FHE), that allows an entity to evaluate arbitrary circuits on encrypted data without being able to decrypt the data. While FHE has many exciting applications and many improvements in FHE were made since the ground-breaking work of Gentry [Gen09], there is still a lot of progress to be done before it becomes a fully practical solution that can be used as a generic technique for secure cloud computing.

Searchable Encryption (SE) is a more focused research topic where instead of enabling arbitrary computation, one only tries to run search operations on the outsourced data, in order to realize some sort of "encrypted database" or, following the term of Fuller et al. [Ful+17], to enable *cryptographically protected database search*. Such functionality is particularly interesting because it enables arbitrary data processing task that only requires a small part of the whole dataset: the cloud user can use search operations to locate the required data, then download it and process it locally. Research on SE started in 2000 with the paper of Song, Wagner and Perrig [SWP00] (see Section 3.3.1 of this manuscript for more details on the history of SE). Since then, significant progress was made in many directions including security definitions and performance but also query expressiveness, database dynamism, and most importantly for the specific topic of this thesis, in the support for multi-user settings.

One of the trends in research on SE is the ability of protocols to process ever larger databases, because the amount of data being produced and stored has been growing at an extreme pace and because more naive protocols are often sufficient, if not better for small databases. For instance in [Cas+14], Cash et al. show that their SE protocol can efficiently process *"above terabyte-scale databases"*. However a large dataset often consists of a large number of small datasets, each having a different legitimate owner. This typically is the case in the hospital example that we mentioned previously, and will likely become even more relevant in the coming years with the application of regulations like the European GDPR [GDPR]. In this scenarios, using a model with a single user owning the whole dataset requires that every legitimate owner of a segment of this dataset gives up any control on their data. In particular if this single SE user — the hospital in our example — gets compromised, the data of every legitimate owner — the patients — is exposed. As a result the problem that was solved by SE, where users can outsource their data to a third party without trusting it, is raised again between the multiple owners of data segments and the entity that manages them. Hence, Single-user SE (SSE, also called Symmetric SE in the literature) is not suited for datasets that consist of several segments owned by different parties.

Multi-User Searchable Encryption (MUSE) typically addresses this scenario, by considering a large number of users that can be divided in two categories: readers and writers. Each writer has his own data and uploads it to the cloud. Then, each writer can choose which readers should be allowed to search her data. MUSE is then a suitable model for large databases composed of many segments each owned by a different user, as it allows to study how each of these users can get privacy guarantees on her data without having to put too much trust in entities she does not control.

3.2 Definitions

3.2.1 System and Functionality

We consider a system composed of a **server** and a number of **users**. Users can be of type either **reader** or **writer**. A writer owns **records** and uploads them to the server (in an encrypted form). For any record, the writer owning the record can authorize any reader to search it. A reader can search the records for which she got the authorization to do so by sending a **query** to the server (again, probably in an encrypted form which we will call **trapdoor** later on). A query is a function that takes a record as input and returns either the boolean value *"true"*, in which case we say that the record **matches** the query, or *"false"*. At the end of the search procedure, the server sends a **response** back to the querying reader (again, possibly in encrypted form) who outputs the ids of records that match the query among the records this reader was authorized to search.

Multi-User and Non-Multi-User Settings A system with several readers and writers is called **multi-reader-multi-writer**. For the sake of simplicity we will use the term **multi-user** (and the acronym MUSE) as a synonym, unless explicitly mentioned otherwise. A system that only supports a single reader and a single writer (often being the same physical entity) is called **single-user**. We sometimes use the acronym SSE for Single-user Searchable Encryption. Some SE protocols support multiple writers but only one reader or conversely, one writer with multiple readers. we will call them **single-reader-multi-writer** (resp. **multi-reader-single-writer**) or more simply **partially multi-user** protocols. By contrast, multi-reader-multi-writer protocols can also be called **fully multi-user**. Note that in some papers (for instance [Cur+06]), the term *multi-user* is used to qualify protocols that are only partially multi-user according to our definitions. Finally we will use the term **non-multi-user** for any protocol which is not fully multi-user.

Formally, we introduce the following syntax:

- R is the set of readers.
- Each record is identified by a unique id d and the record is denoted either as W_d or as $\boldsymbol{W}[d]$ where $\boldsymbol{W}$ is the set of all records in the system.
- We note $q_{r,s}$ the s-th query of reader $r \in R$. We may also use the notation $\boldsymbol{q}[r][s]$ where $\boldsymbol{q}$ is the two-dimension array of all queries that are sent. We will sometimes use the notation "$\forall q_{r,s} \in \boldsymbol{q}$" as a synonym of "$\forall (r,s) : r \in R \wedge s \in [|\boldsymbol{q}[r]|]$"

We define a MUSE system as comprising the following algorithms:

- $\mathsf{Reader.KeyGen}(1^\lambda) \to \rho_r$: This algorithm is executed by reader r to generate his reader keys.
- $\mathsf{Writer.KeyGen}(1^\lambda) \to \gamma_d$: This algorithm is executed by the writer owning record W_d to generate the encryption key for this record.
- $\mathsf{Writer.Encrypt}(W_d, \gamma_d) \to \overline{W}_d$: This algorithm is executed by a writer to encrypt a record W_d with key γ_d. The resulting **encrypted record** $\overline{W}_d$ is sent to the server. Note that speaking of *"encrypted record"* or *"encrypted keywords"* is often a language abuse as in most SE schemes the encrypted record is not decryptable. Some authors speak of "hash" instead of "encrypted keyword", but we keep the term of "encrypted keyword/record" because it has been widely adopted in the SE literature (see e.g. Section 1.2 of [Bös+14]).
- $\mathsf{Writer.Delegate}(r, \gamma_d) \to \Delta_{r,d}$: In some MUSE protocols, this algorithm is executed by a writer to authorize reader r to search record W_d that was encrypted with key γ_d. The output, noted $\Delta_{r,d}$, is called a **delta** and is sent to the server. We also define the function $Auth$ which maps a reader id r to the ids of records this reader is authorized to search. Said differently, $d \in Auth(r)$ means that r has access to record W_d.
- $\mathsf{Reader.Trapdoor}(q_{r,s}, \rho_r) \to t_{r,s}$: This algorithm is executed by reader r on input the query $q_{r,s}$ and the key ρ_r. The output, noted $t_{r,s}$, is the **trapdoor** corresponding to $q_{r,s}$ and is sent to the server. A trapdoor can be seen as an "encrypted query" (again, this is a slight language abuse).
- $\mathsf{Server.Search}(t_{r,s}, \boldsymbol{\Delta}, \overline{\boldsymbol{W}}) \to p_{r,s}$: This algorithm is executed by the server on input a trapdoor $t_{r,s}$, the set $\boldsymbol{\Delta}$ of all deltas, and the set $\overline{\boldsymbol{W}}$ of all encrypted indexes. The output, noted $p_{r,s}$ is the **server response** corresponding to query $q_{r,s}$ and is sent to reader r.
- $\mathsf{Reader.Open}(p_{r,s}) \to a_{r,s}$: This algorithm is executed by reader r on input response $p_{r,s}$ (and potentially some keys known to r). The output, noted $a_{r,s}$ is the **query result** corresponding to query $q_{r,s}$.

Definition 3.2.1 (Correctness of a MUSE protocol)**.** A MUSE protocol is **correct** if query results actually contain the id of records the query reader had access to, that is, if the following holds except with negligible probability:

$$a_{r,s} = \{d \in Auth(r) : q_{r,s}(W_d) = \text{True}\}$$

Types of Search Functionality In most protocols, a record is a set of bit strings called **keywords**, that is for every record W we have:

$$W \subset \{0,1\}^*$$

Such protocols are said to provide **keyword search**. Some SE protocols are presented as providing *conjunctive keyword search* or even *boolean keyword search*. **Boolean keyword search** denotes that queries are boolean combinations over the presence of keywords. For instance one can search records *containing either the keyword "urgent" or containing both keywords "cloud" and "security"*. Formally in boolean keyword search, a query q consists of a tuple of keywords $(w_1, w_2, \dots)$ and a boolean function ϕ such that for any record W:

$$q(W) = \phi(w_1 \in W, w_2 \in W, \dots)$$

Conjunctive keyword search is a special case of boolean keyword search where the boolean function is a conjunction, that is, only contains *and* operators. In conjunctive keyword search a query can then be characterized only as a tuple of keywords. Finally **single-keyword search** is a special case of conjunctive keyword search with a single keyword, where a query can be characterized simply by its keyword.

The notions of boolean keyword search is important, but is not really about functionality as it can be implemented over any protocol providing only single keyword search: the reader would simply send several separate queries that would be processed independently, and either the server would apply the desired combination on the results (if the server is allowed to see the results and the combination), or the reader would do it himself. As a consequence, the qualification of a SE protocol as providing boolean or conjunctive keyword search should instead be based on privacy and/or efficiency properties that should improve over the naive solution we just described.

Database Dynamism Database dynamism denotes the possibility to add/modify/delete records present in the system after the database has already been created. Again, this can always be done in a trivial but inefficient and potentially insecure way by dumping the existing system and setting up a whole new one with the modified database. A SE scheme is thus said to provide some database dynamism (sometimes referred to as **Dynamic SE**) when it has better performance or security than the trivial solution regarding database updates.

3.2.2 Security

Regarding security, the purpose of SE is to protect the privacy of the records. Additionally in most SE protocols, query privacy is considered as an objective in itself instead of something that is simply required by record privacy.

It is widely accepted that SE protocols cannot be efficient and at the same time let the server get no information at all about records and queries. As a result all SE protocols "leak" some information, meaning that they let the server get some information about records and queries. It is then important to identify the information being leaked by each protocol.

Most papers in SE characterize the leakage of protocols following the methodology introduced by the seminal paper of Curtmola et al. [Cur+06]. This methodology consists in defining the **history** representing *"the instantiation of a protocol execution"*, and the **leakage** (or **trace**), a function of the history representing *"exactly the information we are willing to leak about the history and nothing else"*. One then proves that the view of some adversary can be simulated in an indistinguishable way given only the leakage, and this proves that the adversary cannot recover any information about the history that is not in the trace.

Curtmola et al. define the history of a SE protocol as the set of all records in the system and the sequence of all queries that were sent. This definition however was made for the single-user setting. For the case of MUSE the history should also include the authorizations, materialized by the function $Auth$ defined in the previous section.

Definition 3.2.2 (History of a MUSE protocol)**.** The history of a MUSE protocol, noted $\mathcal{H}$, is defined as:

$$\mathcal{H} := (\boldsymbol{W}, \boldsymbol{q}, Auth)$$

We then adapt the definition of **non-adaptive semantic security** given by Curtmola et al. (Definition 4.8 of [Cur+06]) to MUSE. Intuitively, the definition expresses the idea that the view of the adversary can be simulated given only the information from the leakage so that any efficient algorithm will behave similarly whether it is given the real or the simulated view. As a result, whatever a real-world adversary can compute from its view must be computable from the leakage only, and we have the desired security property. Formally, the definition of security introduces three algorithms which must run in time polynomial in the security parameter: $\mathcal{D}_1$ is the "first half" of the distinguisher and tries to come up with a history $\mathcal{H}$ that will ease the task of the second half; $\mathcal{S}$ is the simulator which using only the information from the leakage must output a view that is indistinguishable from the real view; and $\mathcal{D}_2$ is the "second half" of the distinguisher which tries to tell whether it was given the real view or a simulated one. State information is passed from $\mathcal{D}_1$ to $\mathcal{D}_2$ through the variable $st_{\mathcal{D}}$.

Definition 3.2.3 (Non-adaptive semantic security)**.** Let MUSE be a Multi-User SE protocol. Let $\mathcal{V}_{\mathsf{MUSE},\mathcal{A}}$ be an algorithm which takes a MUSE history, runs protocol MUSE on this history, and outputs the view of adversary $\mathcal{A}$ during this execution. Let $\mathcal{S}$ be a simulator and λ be the security parameter.

We say that MUSE is semantically secure with leakage $\mathcal{L}$ with respect to $\mathcal{A}$ (or simply that it has leakage $\mathcal{L}$ w.r.t $\mathcal{A}$) if for all PPT $\mathcal{D}_1$, there exists a PPT simulator $\mathcal{S}$ such that for all PPT $\mathcal{D}_2$,

$$\begin{aligned}
&\mid Pr[\mathcal{D}_2(st_{\mathcal{D}}, \mathcal{V}(\lambda, \mathcal{H})) = 1; (st_{\mathcal{D}}, \mathcal{H}) \leftarrow \mathcal{D}_1(1^\lambda)] \\
&- Pr[\mathcal{D}_2(st_{\mathcal{D}}, \mathcal{S}(\lambda, \mathcal{L}(\mathcal{H}))) = 1; (st_{\mathcal{D}}, \mathcal{H}) \leftarrow \mathcal{D}_1(1^\lambda)] \mid \\
&\leq negl(\lambda)
\end{aligned}$$

Because the "view" algorithm $\mathcal{V}$ is not controlled by the distinguisher, this definition of security only covers **honest-but-curious** adversaries. While proving security properties against a fully malicious adversary, meaning an adversary that can send maliciously-crafted messages, would be preferable, it would also be much more difficult. The honest-but-curious model is still relevant for searchable encryption.

Indeed modifying the behaviour of the CSP, i.e. modifying the software running on it, seems very difficult to do without a great number of employees, managers and executives working at the CSP noticing the changes. It seems much easier for an attacker, be it from inside or outside the company operating the CSP, to simply "monitor" whatever messages are sent and received by the CSP and to try to derive as much information as possible from this. This last situation corresponds exactly to the honest-but-curious model.

The leakage function will be defined separately for each protocol, but we formally define several concepts that will be useful to describe leakages. One is the notion of **access pattern**, that comes from the literature on Oblivious RAMs (see [Gol87]). In SE, the access pattern simply consists in the result of queries.

Definition 3.2.4 (Access pattern of a MUSE history)**.** The access pattern of a MUSE history is the result of the queries

$$\mathsf{AccPatern}(\mathcal{H}) := \{\ ((r,s),\ \{d \in Auth(r) : \boldsymbol{q}[r][s] \in \boldsymbol{W}[d]\})\, \forall q_{r,s} \in \boldsymbol{q}\ \}$$

We then define the notion of **result length** that is simply the length of a query result

Definition 3.2.5 (Result Length)**.** The result length of a MUSE history consists in the following

$$\mathsf{RsltLength}(\mathcal{H}) := \{\ ((r,s),\ |\{d \in Auth(r) : \boldsymbol{q}[r][s] \in \boldsymbol{W}[d]\}|)\, \forall q_{r,s} \in \boldsymbol{q}\ \}$$

In this manuscript informations such as the length of each record, the number of queries sent by each reader and the authorizations will be considered as non-sensitive. We regroup this information in what we call the **benign leakage**.

Definition 3.2.6 (Benign leakage)**.** The benign leakage of a MUSE history consists in the following:

$$\mathsf{Benign}(\mathcal{H}) := (|W|\ \forall W \in \boldsymbol{W}, |\boldsymbol{q}[r]|\ \forall r \in R, Auth)$$

Finally we define the notion of **revealed content** that represents the queries and records the adversary has a "legitimate" access to through the users it controls. It regroups not only the queries of corrupted readers and the records owned by corrupted writers, but also the records corrupted reader are authorized to search.

Definition 3.2.7 (Revealed Content)**.** The revealed content of a MUSE history given colluding readers $R' \in R$ and colluding writers owning $\boldsymbol{W}' \subset \boldsymbol{W}$ is:

$$\mathsf{Revealed}(\mathcal{H}, R', \boldsymbol{W}') := (\{\boldsymbol{q}[r]\ \forall r \in R'\}, \boldsymbol{W}' \cup \{W_q\ \forall r \in R'\ \forall q \in Auth(r)\})$$

In all the MUSE protocols we study, including the ones we design, the adversary has access to the benign leakage and the revealed content; and while in theory one could study ways to hide the information in the benign leakage or to mitigate the privacy implications of a user corruption on the records this user has access to, we do not have such goals in this thesis. As a result when describing the leakage profiles of some MUSE schemes we will sometimes omit to mention these two parts of the leakeg profile. For instance, a protocol leaking the access pattern, the benign leakage and the revealed content could be described simply as "leaking the access pattern".

Finally we will speak of **strict leakage profile** to denote the fact that a protocol leaks *no more than* some information. For instance, a protocol that leaks the access pattern also leaks the result length since the latter can be computed from the former. A protocol that would leak *no more than* the result lenght (additionally to the benign leakage and revealed content, as always) can be said to have *strict result length leakage* in order to differentiate it from a protocol which leakage would *more than* the result length. We will see later in the thesis (Section 4.4.3) that such confusions led to mistakes in previous papers on MUSE.

3.3 State of the Art

We start by a short review of the state of art in SE in the non-multi-reader-multi-writer setting. Strictly speaking, these are a special case of MUSE with a number of readers and/or writers equal to one, and as a result we should be able, in theory, to cover the topic of MUSE without presenting non-multi-user settings at all. Historically however, the problem of SE was first studied in the single-user setting. As a result, it seems important to give an overview of non-multi-user SE to introduce the main concepts of searchable encryption at their inception.

We then give a complete review of the state of the art regarding fully multi-user SE protocols.

3.3.1 Non Multi-Reader-Multi-Writer SE Protocols

Because non multi-reader-multi-writer SE protocols are not the main focus of this thesis, we only cover the contributions that are historically important and the ones that are the currently most significant. For a more complete review of the subject, we refer the reader to the 2014 review of Bösch et al. [Bös+14]. and the 2017 paper of Fuller et al. [Ful+17].

Research on Searchable Encryption started with the paper of Song, Wagner and Perrig [SWP00] that studies the problem of *"searching on encrypted data"*. The main component of the protocol is an encryption scheme where a trapdoor can be generated for a given message, and this trapdoor reveals which ciphertexts are an encryption of this message. Nothing is revealed about other ciphertexts, except that they are not encryptions of the message. The protocol is proved to satisfy IND-CPA security, but this notion of security that was developed in the context of "traditional" encryption proved unadapted for searchable encryption.

The second major landmark in the history of research on SE is the paper of Curtmola et al. [Cur+06]. After showing the limitations of previous definitions of security for SE (including but not limited to the definition used by Song et al.), they give a new definition (which we mention in Section 3.2.2) which is still considered as the reference as of today. They present a SE protocol that satisfies this new definition of security and is the first SE protocol that achieves sub-linear search, in that the time complexity of the search procedure is linear with the number of records that match the query and not with the total number of records in the system.

Among other significant contributions we can cite a sequence of papers [Jar+13; Cas+13; Cas+14] starting with [Cas+13] that presents a protocol named "Oblivious Cross-Tag" (OXT). The OXT protocol and further improvements of it are characterized by a high query expressiveness (boolean keyword search) with a quite reduced leakage profile (access pattern of only one chosen keyword in the query) and a very high efficiency (linear in the number of records matching this chosen keyword). Another contribution providing high query expressiveness is the "Blind Seer" protocol by Pappas et al. [Pap+14].

Some protocols introducing efficient database updates are [Cas+14] and [KPR12] but these works fall short of meeting the privacy requirements raised by database dynamism and reveal sensitive information during update operations. Focusing on these privacy issues, further research came up with **forward-private** SE schemes [SPS14; Bos16] where update operations reveal no information, and **backward-private** protocols [BMO17] where the server cannot apply new queries on data that was deleted. In a recent paper, Kamara and Moataz [KM17] present a new SSE protocol named IEX that combines the efficiency of OXT (improves it, actually) with forward privacy properties.

The first paper exploring the (partial) multi-user setting is by Boneh et al. [Bon+04] and introduces a protocol called "Public key Encryption with Keyword Search" (PEKS). The PEKS protocol allows anyone to encrypt keywords using a public key, but only the owner of the corresponding secret key can create trapdoors that can reveal whether some ciphertext corresponds to some given keyword. The intended usage for PEKS is an email server which must learn nothing about the keywords associated to an email except the presence of very specific keywords chosen by the user owning the secret key. Typically, this user would issue a trapdoor for the keyword "urgent" and give it to the server so that the server can identify urgent emails and process them accordingly, while learning nothing about the other keywords attached to the emails it receives. Note that in PEKS the server knows both the content of the trapdoors and the result of their application. This is impossible to prevent due to the public nature of the encryption procedure: a curious server could always get information about the content of a trapdoor by encrypting arbitrary keywords itself and applying the trapdoor on them.

A small number of papers address the multi-reader-single-writer setting. In most cases the protocols try to tackle the problem of reader revocation, that is, of canceling existing authorizations in a secure and efficient manner. The seminal paper of Curtmola et al. [Cur+06] contains some notes about building such protocol using a standard SSE protocol and a Broadcast Encryption (BE) protocol. We can also note the protocol of Elkhiyaoui et al. [EÖM14] that uses Private Information Retrieval (PIR), Cuckoo Hashing, Attribute-Based Encryption (ABE) and Oblivious Pseudo-Random Functions (OPRF).

3.3.2 Multi-Reader-Multi-Writer SE Protocols

Hwang and Lee 2007 One of the first papers to consider a fully multi-user setting in SE is by Hwang and Lee [HL07]. The multi-user protocol is not the main contribution of the paper but rather an extension to the multi-user setting of the "Public key Encryption with Conjunctive Keyword Search" (PECKS) protocol they present. The multi-user variant, named mPECKS, is presented in Section 5 of [HL07]. Their multi-user protocol is very close to the simple juxtaposition of one single-reader-multiple-writer protocol for each reader. Indeed, while the cost of encrypting a record is lower than with several parallel systems, the size of an encrypted record is still linear with the number of readers authorized to search it. The notion of "record" is a bit different in [HL07] than what is used in this manuscript, as they consider a record as a tuple of distinct keywords. Note that viewing the records as sets of keywords is more general and that tuples of keywords can be implemented on top of sets by having keywords of the form `field_name=value`. Hwang and Lee define their own security definition for MUSE, derived from the one they use in their non-multi-user protocol. Informally, it consists in an indistinguishability game where the adversary, representing the server, tries to distinguish between the encryption of the record it submitted and the encryption of a random record, given access to a trapdoor oracle, with the restriction that it cannot query the oracle on the record that is submitted for the challenge. There is no real justification for such a security definition except its similarity with the one for the non-multi-user protocol.

Bao et al. 2008 One important paper in the history of MUSE is the one from Bao et al. [Bao+08]. In this paper, the authors present a protocol with a Trusted Third Party (TTP) that holds some master key k and performs key generation. For each user u, the TTP creates a user key k_u and a **delta token** Δ_u that is a cryptographic value "materializing" the authorization of user u to access the database (the term of "delta token" is actually from a more recent paper [PZ13]). k_u is given to the user which can use it both for searching and for writing, that is, $\rho_u = \gamma_u = k_u$. The authorization is given to the server.

What happens in the protocol of Bao et al. is that users create trapdoors and encrypted records by encrypting keywords with their secret key and the server uses the authorizations to transform these encrypted keywords so that they are encrypted under the master key.

More precisely, users use the following function to encrypt keyword w:

$$\mathsf{Encrypt}(w, k_u) = h(w)^{k_u}$$

and the server uses the following function to "transform" trapdoors and encrypted keywords (here noted t) that it receives using the delta Δ_u corresponding to the user:

$$\mathsf{Transform}(t, \Delta_u) = e(t, \Delta_u)$$

Chaining these two functions allows encrypting keywords under the master key k without user u nor the server knowing this master key:

$$\begin{aligned} &\mathsf{Transform}(\mathsf{Encrypt}(w, k_u), \Delta_u) \\ &= e(h_e(w)^{k_u}, g_2^{k/k_u}) \\ &= e(h_e(w)^{k}, g_2) \end{aligned}$$

As a result, after trapdoors and encrypted record are transformed by the server, it is as if a single user was interacting with the database using master key k. This type of architecture, where all records and trapdoors end up being encrypted under a common key, is denoted as **single-key architecture**. A consequence of a single-key architecture is that any trapdoor sent by any user can be applied on any encrypted record after transformation, and as a result the protocol does not provide a "true" per-user authorization mechanism beyond simply relying on the server for properly assuring access control.

Because the encryption procedure is deterministic, Bao et al. introduce an additional mechanism to prevent the server from noticing similar encrypted keywords across records: an encrypted keyword $e(h(w)^k, g_2)$ is stored as $(r, Enc_K(r))$ where r is a random value, Enc is a traditional symmetric encryption procedure like AES, and K is a symmetric key derived from $e(h(w)^k, g_2)$. This mechanism however does not prevent the server from seeing which encrypted keywords have the same plaintext as soon as a trapdoor for this keyword is issued.

Regarding security, Bao et al. define *query privacy*, *query unforgeability* that implies that generating a legitimate query for some user key cannot be done without knowledge of this user key, and *revocability*.

- **Query privacy** is defined using the simulation technique. The server alone is considered as the adversary and the security property implies that the server cannot learn anything beyond the access pattern of each query and the benign leakage as previously defined.
- For **query unforgeability**, two separate adversaries are considered: the server alone, and a collusion of users. This notion is defined as a computational problem, which is shown to be equivalent to forging a BLS short signature [BLS04].
- For **revocability**, the adversary is a user that is enrolled, searches the database several times, then is revoked. This notion is defined with an indistinguishability game where the revoked user must distinguish the encryption of two keywords it did not previously searched for.

As a follow-up of [Bao+08], a paper by Yang et al. [YLW11] introduces extensions to the protocol but restricts it to a single centralized writer, making it less relevant regarding the scope of this thesis.

Dong, Russello and Dulay 2008 A paper by Dong, Rusello and Dulay [DRD08] presents a protocol that is very similar to the one of [Bao+08], but is based on the RSA encryption scheme [RSA78] instead of bilinear pairings. In this protocol, encryption consists in plain RSA (see Section 11.5.1 of [KL14]) where the encryption exponent e is the master key:

$$\mathsf{Encrypt}(w, e) = h(w)^e \bmod n$$

with n being the RSA modulus and h a secure hash function into the proper group. A user key and authorization pair consists of (e_1, e_2) such that $e_1 e_2 \equiv e \bmod \phi(n)$. User encryption consists then in RSA encryption using the user key, and the Transform function consists of a re-encryption using the delta as the RSA key. This results in a keyword encrypted under the master key. Finally and as in [Bao+08], a technique is used to randomize the encrypted keywords when they are stored in order to prevent the server from noticing identical keywords as long as the keywords are not queried.

Dong et al. give a definition of security that uses the simulation technique based on the methodology of Curtmola et al. [Cur+06]. The adversary is the server alone and the leakage consists of the access pattern of queries and the benign leakage.

A paper by Asghar et al. [Asg+13] presents a protocol that runs on top [DRD08] to allow more complex queries.

The MKSE protocol In [PZ13], Popa and Zeldovich present a MUSE protocol named “Multi-Key Searchable Encryption” (MKSE). This protocol introduces radical changes from the previously existing MUSE protocols in order to address a much more challenging threat model where some users may be colluding with the server. As a result MKSE implements per-user authorizations unlike the protocols of [Bao+08] and [DRD08]. Another consequence is that MKSE does not follow the “single-key” structure where all encrypted keywords and all trapdoors end up being encrypted under a common key independently of which user they originate from. The very name of the protocol expresses this major difference with [DRD08] and [Bao+08]. Also, there is no trusted third party in MKSE.

MKSE uses some form of transformation in a way that is very similar to [Bao+08]. In particular it uses bilinear pairings as well. However in MKSE encrypted keywords in records are never transformed: they are stored encrypted under the key of the user that created them. A delta in MKSE works similarly as a delta in [Bao+08] but it allows the transformation from a user key to another user key, instead of to a master key. There is no notion of master key in MKSE. Let k_A and $k_B \in \mathbb{Z}_\zeta$ be the (secret) user key of users Alice and Bob respectively, Alice can authorize Bob to search her records by generating $\Delta_{B,A} = g_2^{k_A/k_B}$. We then have the following algorithms that allow Bob to generate a trapdoor using his key and the server to transform it into something that can be used to search Alice's record.

$$\mathsf{Encrypt}(w,k) = e(h_e(w), g_2^k)$$
$$\mathsf{Trapdoor}(w,k) = h_e(w)^k$$
$$\mathsf{Transform}(t,\Delta) = e(t,\Delta)$$

For the same reason as in [Bao+08], we have:

$$\begin{aligned} &\mathsf{Transform}(\mathsf{Trapdoor}(w,k_B), \Delta_{B,A}) \\ &= e(h_e(w)^{k_B}, g_2^{k_A/k_B}) \\ &= \mathsf{Encrypt}(w,k_A) \end{aligned}$$

Popa and Zeldovich suggest two ways for creating deltas in MKSE: either one of the users sends his secret key to the other one; or the user being granted search rights, that is, Bob in our example, makes the value g_2^{1/k_B} public. Thanks to the hardness of the Discrete Logarithm Problem in pairings groups, the latter solution does not reveal the value of k_B. However the server is now able to pair trapdoors coming from Bob to get a keyword encrypted under some kind of "neutral key" equal to one, which allows dictionary attacks:

$$\begin{aligned} &\mathsf{Transform}(\mathsf{Trapdoor}(w,k_B), g_2^{1/k_B}) \\ &= e(h_e(w)^{k_B}, g_2^{1/k_B}) \\ &= e(h_e(w), g_2) \end{aligned}$$

Finally, Popa and Zeldovich make use of the same mechanism as in [Bao+08] to randomize encrypted records as they are stored in the server, meaning the encrypted keyword is not stored directly but instead is used to derive a symmetric key that is used to encrypt a random value.

Regarding security, [PZ13] defines two properties, namely *data hiding* and *token hiding*, meant to model record and query privacy, respectively. Both definitions are modeled by indistinguishability games and consider the adversary as the server with the ability to obtain the secret keys of some users, in order to model user-server collusions.

The MKSE protocol had a quite important impact. [PZ13] has been cited by a number of papers [Yan+14; Yan+15; Tan14], most of them using it as a base and suggesting improvements and extensions to it. Also, the MKSE protocol is at the core of the Mylar platform, presented in [Pop+14], that aims at facilitating the development of secure web applications. Mylar in turn was cited several times in the literature and received broad press coverage[1].

[1] `http://bgr.com/2014/03/27/mylar-website-encryption-technology/`

Kiayias et al. [Kia+16] The protocol presented by Kiayias et al. in [Kia+16] only consider a single data owner and thus could seem partially multi-user. However the authors claim that *"the scenario where there are multiple data owners can be solved similarly"*. Also, the fact that they give a lengthy description of the MKSE protocol and that they cite one of the protocols from this thesis [RMÖ15] makes it part of the MUSE literature. Note that [Kia+16] was published *after* the beginning of this thesis.

The protocol, based on pairings, is rather complex. Because none of the techniques used in this protocol have an impact on the rest of this thesis, we refer the reader to [Kia+16] for more details on this construction.

Security is defined through some indistinguishability games in which the adversary choses the data it wants to be challenged on, then the challenger perform the system setup and give the public keys to the adversary, and finally the adversary must perform some distinguishing task after having access to some oracles. The adversary seems to model only the server but none of the users.

Hamlin et al. [Ham+18a]: theoretical improvements on collusion-safe MUSE Towards the end of this PhD study a paper was published at PKC 2018 by Hamlin et al. [Ham+18a], presenting three new MUSE protocols. We will call these three protocols HSWW18-I, HSWW18-II and HSWW18-III, respectively. The three protocols have a characteristic in common, being that a reader must download and process every single record she is given access to, decrypt these records, re-encrypt them with her own key, and finally she must upload the re-encrypted records to the server. This goes against our objective to have a low workload for users even with a large database, but as we explain further below the true aim of Hamlin et al. in [Ham+18a] seems to solve the problem of record duplication, not the one of user workload.

The authors of [Ham+18a] define security through a unique indistinguishability game that models corrupted users.

The HSWW18-I protocol, called "MKSE with Fast Search" by Hamlin et al. (Section 4 of [Ham+18a]), is only based on symmetric encryption, which makes each of its algorithms very fast. Its functionning is very similar to SSE protocols. Hamlin et al. note that having to store one re-encrypted record at the server for each authorized reader results in a large storage requirement at the server. It is this problem of record duplication that is addressed by protocols HSWW18-II and HSWW18-III. These protocols use recent *cryptographic obfuscation* techniques (See the introduction of [BSW16] for background and references) in order to lower the amount of data stored at the server. Intuitively, in these protocols the reader uploads an obfuscated program that decrypts the reader's query and the targeted encrypted record, and performs the search. Thanks to obfuscation, this program can be given to the server without the server being able to recover the reader key or record key.

It must be noted that obfuscation is still as of today a very theoretical primitive, and is still far from being usable in practice. For instance, protocol HSWW18-II is based on *differing-input obfuscation* (diO) (again, see the introduction of [BSW16]) while Hamlin et al. themselves admit that *"there is evidence that diO for general circuits might not exist"*. In fact, HSWW18-II rather is a

http://motherboard.vice.com/read/want-to-keep-data-private-encrypt-it-before-it-even-reaches-a-server

http://www.ibm.com/developerworks/cloud/library/cl-always-on-data-encryption-for-cloud-security

http://spectrum.ieee.org/computing/software/how-to-compute-with-data-you-cant-see "stepping stone" whose purpose is to ease the presentation of HSWW18-III. The latter is based on a variant of diO called *public-coin differing-input obfuscation* (pc-diO), a new notion in obfuscation presented in [IPS15]. Hamlin et al. only state that *"no such implausibility results are known for pc-diO"*. These obfuscation-based protocols are important new contributions to research in MUSE, because replication of encrypted records is an important problem in MUSE (we discuss it in Section 5.1.1 of this manuscript). However the use of obfuscation makes these protocols only theoretical for now, and as a result we do not compare them with the other MUSE protocols.

The HSWW18-I protocol, however, is very practical because it only depends on Pseudo-Random Functions (PRFs). Nevertheless, we cannot find any advantage of using the HSWW18-I protocol over the following protocol:

- For each record owned by a writer, the writer picks a symmetric record key, encrypts the record and uploads it to the server;
- For each reader allowed to search a record, the owner of the record sends the record key to the reader via some secure channel;
- Each reader downloads every record she was given the key for and decrypts them;
- Finally, each reader uses whatever system she wants for managing the records she received; In particular, she can use any state-of-the-art SSE protocol.

The first three steps of this protocol corresponds exactly to the beginning of HSWW18-I. The difference between HSWW18-I and the protocol we propose is that instead of using an already existing SSE protocol, HSWW18-I seems to define its own SSE protocol, essentially encrypting keyword w as $F_{F_{K^u}(w)}(r)$ where F is a PRF, K^u is the reader key and r is a random value.

HSWW18-I could then be seen as a simple juxtaposition of one parallel SSE system for each reader. We insist on that the presence of HSWW18-I in the paper of Hamlin et al. [Ham+18a] is still entirely justified, as such juxtaposition is probably the best example of the problem of record replication in MUSE, which is the true subject of [Ham+18a].

Contacted, the authors of [Ham+18a] agreed with our remarks, insisting however that the SSE protocol used must be dynamic. They updated the eprint version of their paper [Ham+18b] accordingly (remark 4.2 of [Ham+18b])

Also, even if such juxtaposition may seem "trivial", one must keep in mind that in some situations this "trivial" protocol will actually be the best choice. More precisely, in any situation where readers are able to bear the cost of decrypting and inserting every record they are given access to, this kind of protocol provides search operations that can only be faster than in all the protocols presented in this manuscript. Moreover, if readers use a state-of-the-art SSE as we suggested, one can expect additional properties as high query expressiveness or forward-secrecy (see Section 3.3.1 on the state-of-the-art SSE protocols).

3.4 Leakage Abuse Attacks against Searchable Encryption

We now turn to a category of attacks against SE protocols called **leakage-abuse attacks**, that recently attracted a significant amount of attention in the research community.

In [IKK12], Islam et al. present an attack affecting any SSE protocol that leaks the access pattern. The attack applies in a special setting that falls out of the scope of the usual security proofs: it requires that the server has information about the content of the records. More precisely the server is given the co-occurrence probability of any two keywords, that is, the probability for any two keywords to be present together in a record. Islam et al. invoke use cases where the records are derived from some publicly-known dataset to justify this assumption. The attack presented in [IKK12] allows the server to recover the content of most queries using a technique named *Simulated Annealing*.

The term *Leakage-Abuse Attacks* (LAAs) was coined later by Cash et al. [Cas+15] who define it as *"attacks that exploit the leakage rather than any particular construction"*. They identify 4 typical leakages of SSE protocols and present attacks against each of them, in an effort to estimate the resilience of each type of leakage against LAAs. As in the attack of Islam et al., all LAAs presented by Cash et al. require that the server has some additional information that is not obtained through the protocol execution and not considered in the usual security proofs for SSE protocols. This additional information is called **prior knowledge** by Cash et al. In passive attacks, the prior knowledge consists in some information about the records. In active attacks the server is able to add arbitrary records to the database.

The leakage profiles defined by Cash et al. are given in the following definitions, where $(w_k)_{k\in[N]}$ is some ordering of all the keywords in the system.

Definition 3.4.1 (L4 as defined by [Cas+15])**.** In leakage profile L4, records must be ordered, that is, $\boldsymbol{W}[i]$ is a sequence of bit strings. Leakage L4 consists in the occurrence pattern of keywords across records and between records. Formally, L4 is defined as:

$$(\{(i,j) : \boldsymbol{W}[i][j] = w_k\})_{k\in[N]}$$

Definition 3.4.2 (L3 as defined by [Cas+15])**.** Leakage profile L3, named *Fully-revealed occurrence pattern with keyword order*, is defined as leakage L4 in a setting where records do not have duplicate keywords (they are still ordered, though).

Definition 3.4.3 (L2 as defined by [Cas+15])**.** In profile L2, records can be sets (unordered). L2, named *Fully-revealed occurrence pattern*, is defined as:

$$(\{i : w_k \in \boldsymbol{W}[i]\})_{k\in[N]}$$

Definition 3.4.4 (L1 as defined by [Cas+15])**.** L1, named *Query-revealed occurrence pattern*, consists of the same kind of information than in L2 but is only revealed as queries match the keywords. Formally, L1 is defined as:

$$(\{i : q \in \boldsymbol{W}[i]\})_{q\in\boldsymbol{q}}$$

Other papers on LAAs are [ZKP16] that presents an improved active attack and [Kel+16] that further extends the analysis to attacks using no auxiliary information about the records but only about the queries, and targets SE protocols providing range queries. Note that all existing LAAs target non-multi-user SE protocols. Table 3.1 lists the characteristics of the main leakage-abuse attacks.

Table 3.1: Main Leakage-Abuse Attacks

Paper	**Leakage necessary**	**auxiliary information**	**output**	**comments**
[IKK12]	L1	keyword co-occurrence probability	queries	
[Cas+15]	L1	(partial) keyword co-occurrence probability	queries	named "counting attack" ; improves over attack from [IKK12]
[Cas+15]	L3	some records	records	Section 5.1.1 of [Cas+15]
[Cas+15]	L2	planted records	records	Section 5.2.2 of [Cas+15]
[ZKP16]	L1	planted records	queries	
[Kel+16]	L1	query distribution	records	
[Kel+16]	result length	query distribution	records	

4 The Dilemma of MUSE: On the difficulty of trading privacy for efficiency

Security and privacy always come at a cost. In this chapter, we show that the decisions that were made in the design of prior MUSE schemes, with efficiency as the main objective, condemned the schemes to very serious issues related to privacy. As a result, all MUSE schemes prior to the ones proposed in this thesis can only protect the privacy of records and queries under assumptions that are unrealistic. In particular these protocols are insecure in presence of even the smallest user-server collusion, which we argue *must* be considered as part of the threat model for MUSE.

We show that, while the notion of Leakage Abuse Attack (LAA) is very useful in the study of privacy in MUSE against user-server collusions, the existing results on LAAs can be misleading when applied to the MUSE problem because they were all designed in the single-user setting. We extend the previous classification of leakage profiles to fix this issue.

We show that the privacy issues we identified also affect the MKSE protocol (see Section 3.3.2) and protocols that derive from it, despite the fact that MKSE was specifically designed for such type of collusions. We describe a practical attack against some software named Mylar implementing MKSE, we analyze the reasons why the security analysis in the paper introducing MKSE [PZ13] does not cover such issues, and we argue that our findings should allow to settle a controversy that arose on the security of the Mylar software.

4.1 The Case for User-Server Collusions

At the origin of research on searchable encryption is the idea that users should not have to put too much trust in a cloud service provider. Nevertheless, one must not forget that the number one type of victim of cyber threats is the end user. The large majority of security incidents start with the corruption of a user account or machine. Most incidents of cloud computing systems actually are caused by the corruption of a user which has access to the system, instead of a direct breach of the cloud system itself.

As a result, when extending searchable encryption to the multi-user setting, it seems natural to consider that some users may fall under the control of an attacker, or worse, be adversaries from the beginning. Also, the adversary that controls these users might be the same one that controls the CSP, which can be modeled by a collusion between these users and the CSP. Naturally, taking such collusions into account is "better" in terms of security: A system that is secure against such collusions will also be secure against a server alone, or a collusion of only users, or even a single user. However, such threat model is also more challenging. We argue that considering user-server collusions in MUSE is not simply "better", it is *mandatory*. In other words, the idea of a MUSE system makes little sense under the assumption that users may not collude with the server.

First of all, it seems very unlikely that someone able to compromise a cloud service provider is unable to take control of at least one user. This is even more true as we consider settings where the number of users is very large, making the monitoring of all users impossible. Second and most importantly, it appears that assuming the absence of user-server collusions makes it automatically possible — and thus preferable — to use a single-user SE protocol instead of a MUSE one.

We explain the latter argument: most MUSE papers that do not consider user-server collusions [DRD08; Bao+08; YLW11; Asg+13] invoke the use case of a company wanting to outsource some data storage tasks where users would all be employees of the company. In this kind of use case where all users are supposedly under the control of a central authority (the company), it could make sense to assume that users will not collude with the server. However, the very fact that there is a central authority makes it possible not to use a multi-user SE system. Indeed the company can very well set up an in-house server to which employees can connect when they need to search the outsourced data and that acts as a single user in a SSE protocol with the cloud service provider. One could fear that the cost of maintaining such in-house server could deter the company from choosing this solution; but state-of-the-art single-user SE protocols are so efficient compared to existing MUSE protocols that the SSE option can only be cheaper by far. Recall that in the OXT SSE protocol [Cas+13] and its variants, queries can be any boolean combination of keywords and the search procedure running time is only linear with the access pattern of the least common keyword. We refer the reader to [Cas+14] for practical performance measurements of an OXT derivative. By contrast, all existing MUSE protocols have a search time linear with the total number of records being searched.

This leads us to the conclusion that the only use cases that are interesting for MUSE protocols require that user-server collusions are taken into account. We see the MKSE protocol [PZ13] and its significant impact as a confirmation of this conclusion.

4.2 Insecurity of Iterative Testing Against User-Server Collusions

It is not surprising that MUSE protocols that were not designed to be secure against user-server collusions have serious privacy issues in presence of such collusions. It is interesting, though, to note that they all follow a common algorithmic structure which, alone, makes it impossible for these protocols to be secure against these collusions.

We now describe this "common algorithmic structure" which we call *iterative testing*, that is defined in Algorithm 2. In a MUSE protocol following the iterative testing structure, an encrypted record is viewed by the server as a list of encrypted keywords. Search is performed by having the server apply the trapdoor it received against targeted records, *keyword by keyword*. This is done through some boolean function we can call "Test" that takes as input the trapdoor and an encrypted keyword and outputs "True" if and only if the encrypted keyword matches the trapdoor. This Test function is iterated over each encrypted keyword of an encrypted record (hence the name of "iterative testing") until there is a match, in which case the server adds the record id to the query result, or until all keywords in the record were tested without matching.

It is easy to verify that all MUSE protocols presented in Section 3.3.2 except [Ham+18a] follow the iterative testing structure:

Algorithm 2: The *iterative testing* algorithmic structure

```
Algorithm: Search
Input: $t_{r,s}, \boldsymbol{\Delta}, \overline{\boldsymbol{W}}$
Output: $p_{r,s}$
Initialize $p_{r,s}$ as an empty set ;
for $d \in Auth(r)$ do
    for $\overline{w} \in \overline{\boldsymbol{W}}[d]$ do
        if Test$(t_{r,s}, \overline{w}) = True$ then
            $p_{r,s} \leftarrow p_{r,s} \cup \{d\}$ ;
```

- From Figure 1 of [Bao+08] it is clear that keywords are encrypted and sent one by one in the algorithm named GenIndex, and that encrypted keywords are tested separately in the algorithm named Search.
- In Section 4.2 and 4.3 of [DRD08] usage of the iterative testing structure is clear as well (encrypted keywords in an encrypted record are denoted as $\{c_{w1}, c_{w2}, \ldots, c_{wn}\}$).
- Same remark for the beginning of Section 6 of [PZ13], where the MK.Enc algorithm processes a single keyword and MK.Match a single encrypted keyword.
- In [Kia+16] it is obvious in the algorithm "Test" of the protocol (Section 4.2 of [Kia+16]) which operates on a ciphertext noted C_k that represents a single keyword in a single record, and outputs "True" or "False" depending on whether the tested ciphertext matches the trapdoor or not.

To the best of our knowledge, all further extensions and variants of the above-mentioned protocols also keep the iterative testing structure. Note that the protocol of Hwang and Lee [HL07] is not in this list. This protocol is a bit different because its definition of records is different (tuples of keywords instead of sets). However, it suffers from the same kind of issues as protocols based on iterative testing.

Naturally, iterative testing allows the server to learn the access pattern, that is, which record matched the query. But with iterative testing the server sees more than this: it sees which was the *encrypted keyword* that matched. A consequence of this is that the server can see when two queries, possibly originating from different readers, match the same encrypted keyword. The server knows then that these two queries are identical. Formally:

$$\big(\mathsf{Test}(t_{r,s}, \overline{\boldsymbol{W}}[i]) = \mathrm{True}\big) \wedge \big(\mathsf{Test}(t_{r',s'}, \overline{\boldsymbol{W}}[i]) = \mathrm{True}\big) \implies q_{r,s} = q_{r',s'}$$

In a multi-user setting where some users can collude with the server, this ability of the server to identify queries from different readers as identical has dramatic consequences on privacy. We give a simple example, illustrated in Figure 4.1. Consider a system with 4 readers and 3 records. Reader bad has access to record W_1, and happens to be under the control of the adversary, who also controls the server. It is then unavoidable that the privacy of W_1 is breached, as the adversary has a somehow "legitimate" access to it. In particular, if bad sends a query for keyword "secret", and the

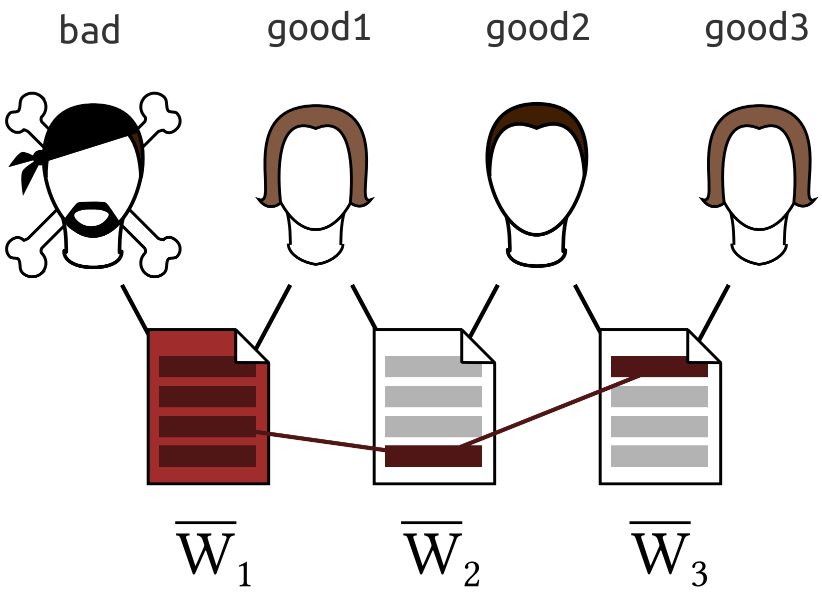


Figure 4.1: An illustration of our attack example against the iterative testing structure

server observes a match at position 3 of $\overline{W}_1$, the server knows that encrypted keyword $\overline{W}_1[3]$ is an encryption of the word "secret". Now if reader good1, having access to records W_1 and W_2, sends a query, the adversary can do the following reasoning:

- If the query matches $\overline{W}_1$ at position 3, then this query is for keyword "secret". If moreover it matches record $\overline{W}_2$, the matching encrypted keyword encrypts the word "secret" too.

- otherwise the query does not contain "secret" and matching encrypted keywords do not encrypt the word "secret"

As a result the privacy of the queries of good1, as well as the privacy of record W_2, can be compromised simply by the corruption of reader bad (which, recall, *does not* have access to W_2).

The privacy breach can propagate quite fast across the system: If user good2 sends a query for keyword "secret", the adversary can recover the query and see if W_3 contains this keyword as well.

This clearly goes against the security properties one would expect from a MUSE system. In Section 4.3.3 we report on statistical experiments that give an idea about how fast information is recovered by even a very small fraction of users colluding with the server in a MUSE protocol that is based on iterative testing. These experiments also show that there is a clear difference in terms of privacy between an iterative-testing-based MUSE protocol and a protocol that would only leak the access pattern.

The iterative testing structure, although simple, intuitive and rather efficient, is then a problem in MUSE protocols because it results in unacceptable privacy violations in presence of user-server collusions.

4.3 Leakage-Abuse Attacks Applied to MUSE

The problems with iterative testing come from that the server sees too much information. While this information — similarities between queries — may not seem too harmful in itself, we saw that it can have dramatic consequences on privacy as soon as the server gets extra information from colluding users. This situation is similar to what happens in leakage abuse attacks. It is then natural to wonder if the work on LAAs, especially the categorization of Cash et al., would have been useful in identifying privacy issues in MUSE with regard to user-server collusions.

In this section we show that, in fact, a naive application of the categorization of Cash et al. on iterative-testing-based MUSE protocols would have probably led to the erroneous conclusion that *“these protocols are exposed to the same kind of attacks as any SE protocol leaking the access pattern”*. We introduce a new leakage profile corresponding to the leakage caused by iterative testing. We show that, while this may seem equivalent to the L1 profile of [Cas+15] (strict access pattern leakage), the two profiles are actually very different when studied in the multi-user setting.

We insist on that we are not trying to criticize the work of Cash et al. or blame the authors: their work was restricted to the single-user setting, and it is not surprising that it cannot be applied “as is” on the multi-user setting.

4.3.1 The $\mathsf{L}_{\mathsf{KWAP}}$ leakage profile

In Algorithm 3 we formally describe the leakage that is induced by the iterative testing structure. Each query reveals the occurrence pattern of keywords and the order of the corresponding encrypted keywords. Because the ordering of encrypted keywords can be completely arbitrary in the iterative testing structure, we apply a random permutation on each record at the beginning of the algorithm. We call this leakage profile *Keyword-Access Pattern Leakage* or $\mathsf{L}_{\mathsf{KWAP}}$.

Algorithm 3: An algorithm that computes the $\mathsf{L}_{\mathsf{KWAP}}$ leakage of a MUSE history

```
Algorithm: $\mathsf{L}_{\mathsf{KWAP}}$
Input: $\boldsymbol{W}, \boldsymbol{q}, Auth$
for $W_d \in \boldsymbol{W}$ do
    $W'_d \leftarrow$ a random permutation of $W_d$ ;
return $\{ ((r,s), \{(d,i) : d \in Auth(r) \wedge q_{r,s} \in W'_d[i]\}) \ \forall q_{r,s} \in \boldsymbol{q} \}$
```

A natural question is: can $\mathsf{L}_{\mathsf{KWAP}}$ be associated to one of the leakage profiles studied by Cash et al.? We refer the reader to Section 3.4 for a formal description of these 4 leakage profiles, and simply remind the following facts:

- L1 or “Query-revealed occurrence pattern” corresponds to protocols in which each query reveals the access pattern (the result) of the query;
- L2 or “Fully-revealed occurrence pattern” corresponds to schemes where occurrence patterns across records are visible even before any query is sent;
- L3 or “Fully-revealed occurrence pattern with keyword order” corresponds to schemes where,

additionally to the access pattern, the order of keywords in the plaintext records is visible before any query is sent.

As to L4, it reveals too much information to be of interest in our study.

Because $\mathsf{L}_{\mathsf{KWAP}}$ is clearly of the "query-revealed" type and it does not reveal the order of keywords in records, one might want to identify it as L1. This intuition is reinforced by a paragraph in Section 2.1.1 of [Gru+16] which describes the difference between L3 and L2:

> In some implementations of the appended PRF scheme the client will sort the PRF values before uploading. This is actually better for security, as information about the order of first occurrences of keywords in each document is not leaked. The leakage is captured by the following model [the L2 model].

This paragraph seems to indicate that having encrypted keywords in arbitrary order results in a L2 profile. $\mathsf{L}_{\mathsf{KWAP}}$ would then be a query-revealed variant of L2, that is, L1.

4.3.2 $\mathsf{L}_{\mathsf{KWAP}}$ is not "just access pattern"

It is probably true that $\mathsf{L}_{\mathsf{KWAP}}$ and L1 are equivalent in the single-user setting. We show that, in the multi-user one, this is not the case. We continue with the example used in Section 4.2. Recall, with this example we showed that the $\mathsf{L}_{\mathsf{KWAP}}$ leakage allows the server to notice that several queries coming from different readers are actually identical, and this quickly led to a violation of privacy of record W_3 even though it should be inaccessible to the corrupted reader bad. If we replay the same scenario with a leakage that is limited to the access pattern of queries (and the benign leakage and revealed content, as always), the loss of privacy regarding W_3 is essentially non-existent. Indeed here is what the adversary sees in the same scenario with access pattern leakage only:

- t_{bad} matches $\overline{W}_1$
- t_{good1} matches $\overline{W}_1$ and $\overline{W}_2$
- t_{good2} matches $\overline{W}_2$ and $\overline{W}_3$

More formally, we have:

$$\begin{aligned}
&L1(\mathcal{H}_{\mathsf{example}}) = \{\\
&\quad(\mathsf{bad}, 1) : (1,),\\
&\quad(\mathsf{good1}, 1) : (1, 2),\\
&\quad(\mathsf{good2}, 1) : (2, 3)\\
&\}
\end{aligned}$$

The only thing the adversary can deduce from this information is that W_1 contains "secret". Observing that t_{good1} matches $\overline{W}_1$ does not imply that q_{good1} corresponds to the keyword "secret". Finally, barely anything can be deduced from the access pattern of q_{good2}.

This does not mean that access-pattern leakage in a MUSE protocol provides absolute privacy against user-server collusions. Access pattern *does* leak some information about records and queries that are not supposed to be accessible to the corrupted user. For instance, observing that some query does not match $\overline{W}_1$ would have implied that the query *does not* correspond to "secret". Also, in the same scenario where all records would consist of a single keyword (recall, the server knows the length of records), the server would know for sure that all records consist in the keyword "secret". What our example shows is that a L1 leakage profile is much less harmful in the multi-user setting than a $\mathsf{L}_{\mathsf{KWAP}}$ profile.

4.3.3 Statistical experiments comparing $\mathsf{L}_{\mathsf{KWAP}}$ to L1

We report on several statistical experiments we ran that compare the harmfulness of an $\mathsf{L}_{\mathsf{KWAP}}$ leakage profile compared to the leakage profiles defined in [Cas+15]. These experiments are heavily inspired by the experiments presented in [Cas+15]. We model a MUSE system using real-world data taken from the ENRON email dataset [Enron] and we simulate attacks based on each type of leakage. In each case we measure the amount of information that is recovered by the attacker.

We perform two experiments: one is on attacks trying to recover queries, and the other is on attacks trying to recover records. In both experiments, the system is built the following way:

- We take as corpus a subset of 30,000 emails from the ENRON dataset;
- We collect all words in the corpus, remove stop words and apply the Porter Stemming algorithm [Por80]. We then select either the 500 or the 5,000 most common words, and we call them the *vocabulary*. The vocabulary size is chosen in order to stay close to the experiments of [Cas+15], that is 500 for query recovery attacks and 5,000 for record recovery attacks.
- For each email, we create a record containing the words from the vocabulary that are present in the email.
- we create 50 readers, and each is given authorization for $3,000$ randomly-chosen records.
- We mark some records as "revealed", representing records the adversary has a direct access to, modeling the collusion of some users.

Query recovery attacks For the experiment on query recovery, the number of revealed records is set to various values from 0.1% of all records to all of the records.

- The attack using access pattern works as follow: for each keyword of the vocabulary, the adversary notes down its occurrence pattern among revealed records. Then for each query, the access pattern of this query is compared with the list of occurrence patterns: if there is only one keyword which known occurrence pattern matches the observed access pattern, we mark the query as recovered.
- The attack using $\mathsf{L}_{\mathsf{KWAP}}$ is even simpler: each query that matches a revealed record is recovered, because the adversary knows the plaintext of every encrypted keyword in a revealed record and sees matching encrypted keywords thanks to $\mathsf{L}_{\mathsf{KWAP}}$.

The results of these experiments are represented in Figure 4.2. We can see a clear difference of efficiency between the two attacks, especially for low ratio of revealed records.

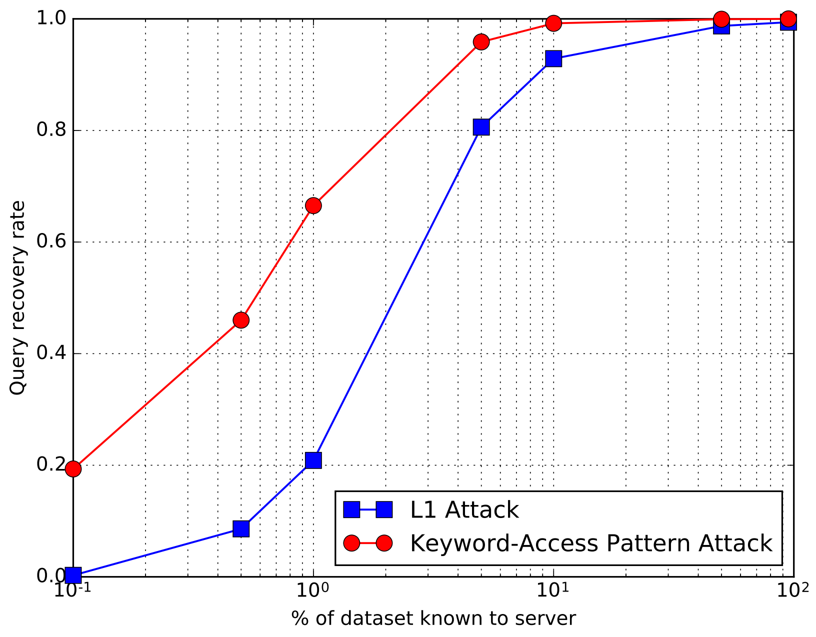


Figure 4.2: Recovery rate (percentage of queries recovered as a function of the fraction of the database initially known) of our query recovery attack against L_{KWAP}, compared to an attack against L1 in the same scenario.

Record recovery attacks For this experiment, the number of revealed records is set to 20. Note that this corresponds to 0.07% of all records only. We simulate a varying number of queries, namely $10,000$, $30,000$ and $60,000$. Note that in a system with 50 readers, $60,000$ queries is only $1,200$ queries per reader on average.

- The attack using L_{KWAP} leakage is essentially the one we described in Section 4.2: the adversary keeps track of encrypted keywords matching the same queries and builds equivalence classes from them; then, when a class contains an encrypted keyword that is present in one of the (few) revealed records, all the encrypted keywords in this class are revealed.
- We do not show results with a L1 leakage since the amount of recovered keywords is almost null. This is not very surprising with 0.07% of records revealed, given the results of the previous experiments represented in Figure 4.2.
- However we simulate an attack using the L3 leakage profile, because the results are very close to the ones for a L_{KWAP} leakage profile when a great number of queries is performed. Recall that L3 is a fully-revealed profile, meaning that the adversary sees relations before any query is performed. We use the same attack as in [Cas+15]: any keyword present in a revealed record is recovered in all other records.

The results of these experiments are represented in Figure 4.3 and are quite impressive: with $60,000$ queries observed by the server, and only 0.07% of records known through collusion, We observe that 90% of all records have at least 10% of their content recovered and one record out of two has at least 30% of its content recovered. This shows that the smallest collusion leads to the corruption of the whole system in a MUSE protocol with $\mathsf{L}_{\mathsf{KWAP}}$ leakage, so that no record is "safe" as soon as the adversary gets access to one of the users. We can see that with a large number of queries observed by the server, the amount of keywords recovered is very close to with an L3 profile, which corresponds in the multi-user setting to the single-key architecture used in [Bao+08] and [DRD08] and which Popa and Zeldovich tried to move away from in [PZ13].

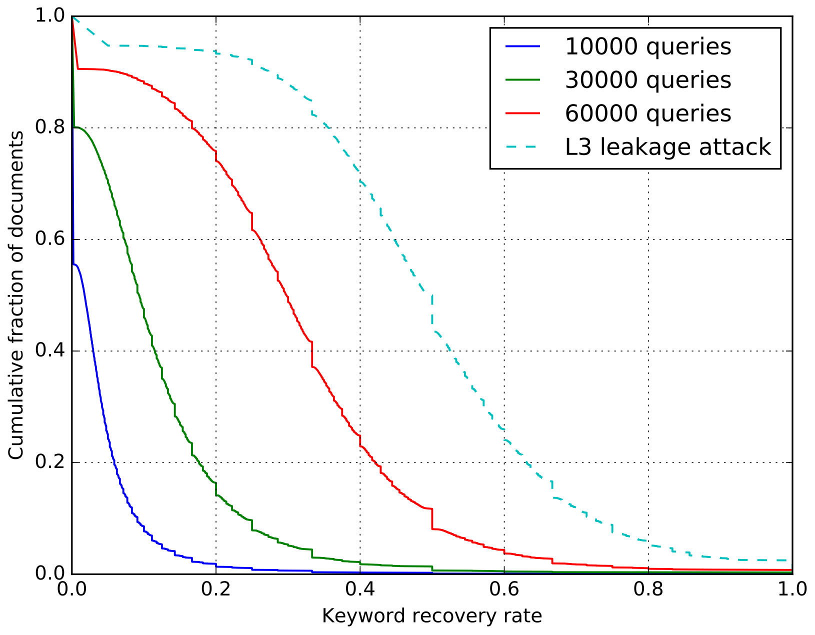


Figure 4.3: Recovery rate of our keyword recovery attack against $\mathsf{L}_{\mathsf{KWAP}}$, compared to an attack against L3 in the same scenario.

As a conclusion, while moving away from the single-key architecture was a necessary step made in [PZ13] towards privacy against user-server collusion in MUSE, keeping the iterative testing structure makes the protocol still very vulnerable. Our experiments show that while the vulnerability caused by iterative testing is lesser than the one caused by the single-key architecture, the difference is actualy very small, and we can safely say that a MUSE protocol following the iterative testing structure fails to protect privacy against user-server collusions.

4.4 Lessons from the MKSE Protocol

In Section 3 of the MKSE paper [PZ13], the authors clearly state privacy against user-server collusions as the main goal (we preserve the original emphasis):

> At a high level, the following security guarantees are desirable. If some user was not given access to a document, the user should not be able to read the contents of that

document or search over that document, *even if the user colludes with the server.*

As a result, our claim that MKSE fails to protect privacy against user-server collusions because of it following the iterative testing structure is in direct contradiction with their claim. This, together with the high impact of the MKSE paper (see Section 3.3.2), makes some more study necessary.

4.4.1 Practical Attack against Mylar

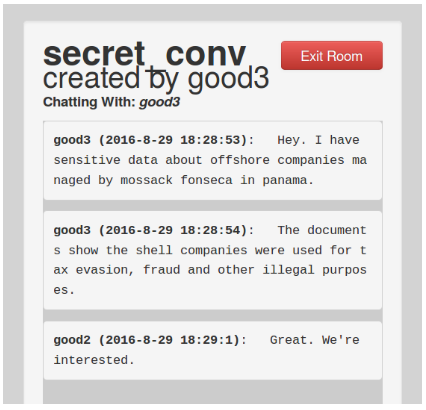
The designers of the MKSE protocol provided an implementation as part of a software platform named "Mylar" [Pop+14]. The source code of Mylar is publicly available[1]. Mylar is presented as *"an experimental research platform for building web applications with end-to-end encryption."*. It comes as a plugin to the Meteor javascript framework [Meteor]. The authors of Mylar also provide some web applications that were built using Mylar. One of them is the "EncChat" application, a fork of some chat application based on Meteor that was modified to use Mylar. In EncChat, a record key (in the sense of MUSE syntax) is created for each chat room, and to each message is associated a MUSE record containing the words in the message. Each participant to the chat room is given an authorization in the sense of MUSE that allows her to search the messages of this chat room, using the MKSE protocol [PZ13]. To confirm the weakness of the MKSE protocol against user-server collusions, we break the security of the EncChat application in a situation and an adversarial model that clearly correspond to what the designers of the MKSE protocol and the Mylar framework claim to address.

We take the following situation, that is essentially the same as the one we described in Section 4.2: There are 4 users named `bad`, `good1`, `good2` and `good3`, and 3 chat rooms named `casual_chat`, `casual_chat2` and `secret_conv`. As the names suggest, user `bad` is under the control of the adversary while the other users are not, and `secret_conv` contains sensitive messages that the adversary might want to recover. Authorizations are as follows: `bad` and `good1` are in `casual_chat`, `good1` and `good2` are in `casual_chat2`, and `good2` and `good3` are in `secret_conv`. As a result, it is clear that the adversary should not have access to the content of `secret_conv`, even by controlling both the server and user `bad`. Note that the situation and the adversary model is similar to what is presented by the designers of the MKSE protocol and the Mylar framework in several talks [Mylar-NDSI; Mylar-RWC].

Our attack works as follows:

- The EncChat server is started with debug logging turned on for the MKSE component. As a result the server writes some detailed debugging information about the search process on its `stdout`, including what encrypted keywords are tested and which ones match which query. A Python script that we wrote is started and constantly reads and parse the `stdout` of the server to extract the useful information.
- `good2` and `good3` have a very sensitive conversation in chat room `secret_conv`.
- some conversations are happening in the other chat rooms where some keywords present in `secret_conv` are present too.

[1]`https://css.csail.mit.edu/mylar/`



secret_conv
created by good3
Chatting With: *good3*

good3 (2016-8-29 18:28:53): Hey. I have sensitive data about offshore companies managed by mossack fonseca in panama.

good3 (2016-8-29 18:28:54): The documents show the shell companies were used for tax evasion, fraud and other illegal purposes.

good2 (2016-8-29 18:29:1): Great. We're interested.

```
### REBUILT CONVERSATION ###
### CHATROOM: 'secret_conv' ###
good3 (2016-08-29 18:28:53): [fonseca, data,
                              mossack, panama]
good3 (2016-08-29 18:28:54): [evasion, tax]
```

Figure 4.4: The conversation `secret_conv` seen by user `good2` in the EncChat application, compared to the output of our attack program run by the adversary controlling the server and user `bad`.

- users perform some search operations, among which appear some keywords that appear in all chat rooms.
- search operations performed by corrupted user `bad` are notified to our attack script together with the keyword that was searched.

Our attack script [Mylar-Atk] does the following:

- it builds equivalence classes of encrypted keywords across different records that matched a common trapdoor;
- for trapdoors coming from known queries sent by the corrupted reader it keeps track of the queried keyword and the encrypted keywords that matched;
- Finally by combining the two previous types of information it is able to partially rebuild records from the encrypted keywords present in the same equivalence classes as encrypted keywords that matched known queries.

As a result, our attack script is able to recover keywords from `secret_conv`, as can be seen in Figure 4.4. Because the attack against Mylar is exactly the one we simulated in Section 4.3.3 illustrated in Figure 4.3, the experiments results give a good measure of the efficiency the attack would have in more realistic scenarios.

4.4.2 Limits of the Security Analysis of [PZ13]

Our attack against Mylar gives a strong, concrete evidence of the reality of the vulnerabilities we identified, but is only a trivial application of the simple theoretical attack we described against iterative testing in Section 4.2. As a result, while this practical attack is necessary, a more interesting question would be: How is it possible that such vulnerability in MKSE and Mylar was not identified earlier, especially given the amount of attention they both got ?

In this section we show how the security definition of [PZ13] fails to model the view of a real-world adversary, how this makes the definition fail to prevent attacks such as the one we described, and we give suggestions on how to improve the security definition.

Security definitions in [PZ13] are modeled as indistinguishability games where the adversary is given access to some oracles, following the kind of definition that is commonly used for "traditional" symmetric and asymmetric encryption. Again, following what is usual in traditional encryption, some restrictions are defined on the kind of queries the adversary can make to the oracles, in order to avoid cases where the adversary can trivially win the game. However in the case of MKSE, these restrictions prevent the game from modeling the view of the adversary in a normal MUSE protocol execution.

There are two security definitions in [PZ13]: *Data Hiding* and *Token Hiding*, corresponding more or less to record privacy and query privacy, respectively. We refer the reader to [PZ13] for a complete description of the definitions. We focus on the *Token Hiding* game. In this game a record is chosen as the *challenge record*; the adversary will have to distinguish two keywords named *challenge keywords* encrypted with the key of the challenge record. The adversary has access to two oracles, one that creates encrypted keywords for the given record key and keyword, and one that creates trapdoors for the given reader key and keyword. The restrictions for these oracles are that either the keyword is not one of the challenge keywords, or the given user does not have access to the challenge record.

Such restrictions seem necessary because they prevent the adversary from trivially winning the game: let the challenge keywords be "foo" and "bar", it would be sufficient for the adversary to request a trapdoor for keyword "foo" from a reader that has access to the challenge record, and to apply this trapdoor on the challenge encrypted keyword. Also, ruling out such case is not a problem because we accept that the records being accessible by a user that is under the control of the adversary are recovered by the adversary.

Nevertheless in a real-world MUSE system, not all trapdoors will come from readers under the control of the adversary. The problem with the token hiding game is that it is unable to model such trapdoors. We come back to our example from Section 4.2 illustrated in Figure 4.1: for the adversary to recover keyword "secret" in record W_2, it is necessary that reader `good1`, which has access to record W_2, sends a trapdoor for this keyword. But `good1` is not under the control of the adversary, only reader `bad` is. In the token hiding game, there is no way to model this event if we set W_2 as the challenge record. This makes the token hiding property meaningless, because it only proves the privacy of keywords if they are never matched by a trapdoor. Said differently, the system is only secure if it is not used. It is then natural that MKSE can be affected by our attack and yet satisfy the security definition of [PZ13].

One way to "repair" the definition of token hiding could be to add an oracle where the keyword is chosen at random by the oracle instead of being specified by the adversary. With no restrictions on this oracle, the game could be able to model all possible scenarios. However the best solution, because it is more clear and less error-prone, seems to be the use of a simulation-based definition following the methodology of Cash et al., where one defines the history, the adversary view of a given history and the leakage from this history.

4.4.3 Notes on the "Popa vs. Grubbs" controversy

We are not the only ones criticizing the security of Mylar and MKSE. Grubbs et al. in [Gru+16] raised a number of concerns on Mylar as well. The designers of Mylar and MKSE tried to defend their work against the critics of Grubbs et al., and this started a public controversy in the (small) world of research on Searchable Encryption.

In this section we argue that our findings on MKSE should be able to close this debate. We also show that, even if the issues and vulnerabilities we present on MKSE seem simple when they are explained, they were not identified by any side of the controversy. This gives us confidence in the importance of our work on the insecurity of previous MUSE protocols.

On the arguments of Grubbs et al. Grubbs et al. describe many different issues with Mylar in [Gru+16], grouped in three categories: Section 6 of [Gru+16] describes attacks using the fact that some metadata is not encrypted in Mylar, which may reveal some sensitive information to the adversary; Section 7 applies existing leakage abuse attacks based on the access pattern on Mylar; finally Section 8 studies active attacks where the server sends maliciously crafted messages to users, for instance by sending malicious keys and authorizations to users.

We argue that the vulnerability of Mylar and MKSE that we identified due to iterative testing improves over the vulnerabilities of Mylar pointed out by Grubbs et al., because it is either more fundamental or more severe.

- The weakness of iterative testing against user-server collusions is more fundamental than the attacks related to metadata management in Mylar (Section 6 of [Gru+16]) because metadata management is an implementation topic, and could potentially be fixed through a few modifications to the code of Mylar. In contrast, no implementation can be secure if it implements a protocol that is insecure in the first place.
- Regarding privacy against active attacks (Section 8 of [Gru+16]), it is also less fundamental than the vulnerability we found because security against active adversaries sending malicious messages to honest users is a very challenging objective which is still an emerging research topic, even in single-user SE.
- Finally the leakage-abuse attacks applied against Mylar in Section 7 of [Gru+16] are basic attacks using the access pattern leakage only. In Sections 4.2 and 4.3.3 of this manuscript we showed that much more powerful attacks are possible using the fact that MKSE has a $\mathsf{L}_{\mathsf{KWAP}}$ leakage profile that is strictly greater than access pattern leakage.

On the arguments of Popa et al. We do not focus on the arguments Popa et al. bring against Sections 6 and 8 of [Gru+16], because they are not much related with our findings.

Regarding Section 7 of [Gru+16] about access pattern attacks against MKSE, Popa et al. give two arguments [Mylar-Website-Security]: First, they claim that this is out of their scope because the Mylar paper [Pop+14] and the Mylar webpage [Mylar-Webpage] both acknowledge that *"Mylar does not protect against access pattern leakage attacks"*. Second, they claim that the situation of Mylar is no different from most other searchable encryption protocols since almost all of them leak the access pattern.

What our results show is that acknowledging that MKSE leaks the access pattern is an understatement. As we showed, a scheme whose leakage would be limited to the access pattern would be significantly more secure than a scheme with $\mathsf{L}_{\mathsf{KWAP}}$ leakage like MKSE. This also invalidates the claim that the situation of Mylar is similar to the situation of other SE schemes. Finally Popa et al. refer to single-user SE protocols when they mention other SE protocols leaking the access pattern, and we showed that the same leakage can have consequences that vary between the single-user and the muti-user settings.

To conclude, we think that our contributions could have helped close the controversy by providing stronger arguments for the insecurity of Mylar and MKSE.

4.5 The Dilemma of MUSE

Multi-reader-multi-writer searchable encryption is a difficult problem. It shouldn't be surprising, then, that reconciling privacy and efficiency in MUSE is hard. As we pointed out in Section 3.3, single-user SE is a special case of MUSE, so that any MUSE protocol is a valid SSE protocol, but the converse is not true. MUSE can then only be harder than SSE. Now SSE itself is difficult, as the numerous recent papers on leakage-abuse attacks show. Citing Hugo Krawczyk at the 6th Bar Ilan Winter School on Cryptography introducing his talk on "Searchable Encryption" [Krawczyk06]:

> The truth is that I have mixed feelings about this area and about our ability to do things that are practical and sufficiently secure.

In this Chapter we showed that MUSE protocols existing before this PhD study (thus not including the protocols from Hamlin et al.) are clearly on the low end of the security spectrum: Even assuming honest-but-curious adversaries, the smallest user-server collusion leads to the voiding of all privacy guarantees. Arguments in defense of these protocols are that they are still better than using no encryption at all, which is the current situation in cloud computing, and that these kinds of weaknesses may be the price to pay for efficiency.

In the sequel of this manuscript, we try to challenge this pessimistic view. We study ways to improve the privacy of MUSE protocols, often –but not always!– at the price of some efficiency, in an attempt to provide a better trade-off between the two than what the iterative testing structure provides. Moreover, we design MUSE protocols that represent different trade-offs between privacy and efficiency, to give users some choice regarding their preference for one or the other.

Figure 4.5 represents the situation of research on MUSE prior to this thesis regarding the privacy-versus-efficiency dilemma. Of course all measurements of "privacy" and "efficiency" are relative and quite imprecise in this graph. The graph illustrates our conclusions that single-key protocols may be efficient but they do not offer enough privacy to be a satisfying solution to the MUSE problem. It also illustrates that MKSE is an attempt to move away from this situation, but because MKSE is still based on the iterative testing structure it fails to provide privacy against user-server collusions. The improvement from single-key protocols to MKSE is thus minimal. We also include the use of parallel SSE protocols in the graph, corresponding to HSWW18-I and the alternative we suggest using state-of-the-art SSE. It then remains a broad domain to explore, from protocols that would finally provide high privacy against user-server collusions even at the price of limited scalability, to protocols that would only sacrifice a small amount of efficiency for a significant gain in privacy, and hopefully protocols that improve on both criteria from the current state of the art.

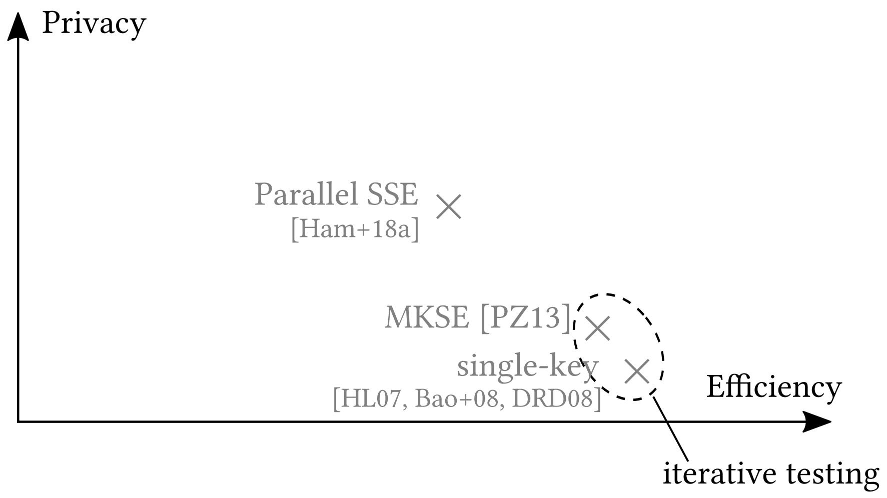


Figure 4.5: Location of previous MUSE protocols regarding the privacy-versus-efficiency dilemma

5 A First Generation of Secure MUSE protocols: choosing between access pattern leakage and scalability issues

In this chapter we present a simple way to build MUSE protocols whose privacy properties hold even in the presence of user-server collusions. The core idea is to have not one but two servers that we assume do not collude together. We then show a number of techniques that leverage this two-server architecture in order to build secure MUSE protocols.

This idea of a two-server architecture is at the core of all three MUSE protocols presented in this manuscript (in this chapter and the next one), which are the first MUSE protocols to be secure against collusions, if one does not count the trivial solution of having parallel SSE systems (see Section 3.3.2).

The first protocol, named DH-AP-MUSE, offers the same level of privacy as using parallel SSE systems while having a minimal cost for the users: namely, readers do not have to download and re-upload records they are given access to, unlike with parallel SSE systems, while the reader workload for search operations remains low and is only linear with the number of records that actually match the query. The second protocol, named RMÖ15, has a heavier workload for the readers in that the size of responses is linear with the total number of records searched, but the privacy guarantees are much stronger: both servers learn nothing beyond the benign leakage and the revealed content (defined in Section 3.2.2).

5.1 Record Preparation and the use of Two Non-Colluding Servers

5.1.1 Record Preparation

Previous MUSE protocols were shown in Chapter 4 to let the server see similarities between encrypted keywords or queries that belong to different users. As a result, the corruption of one user impacts the privacy of other users, even if they are "far" from the corrupted user in the authorization graph.

In the case of the single-key MUSE protocols, illustrated in part (a) of Figure 5.1, the problem comes from that a trapdoor can be applied to any encrypted keyword of any encrypted record after transformation. A single user colluding with the server can then have access to the whole database. The idea of Popa and Zeldovich in [PZ13], illustrated in part (b) of Figure 5.1, is to have one key for each record instead of a single master key. This way, a trapdoor sent by a user can only be applied on records this user was authorized to search. This technique brings some additional costs because the server must transform a trapdoor for each targeted record instead of transforming it once as in

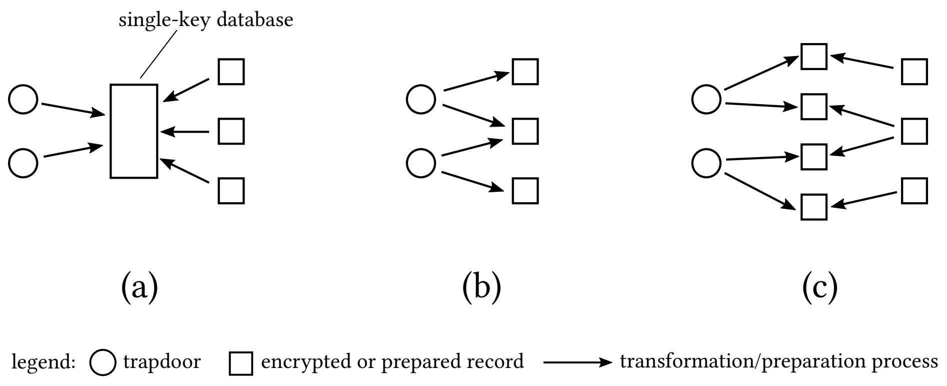

single-key database
(a)
(b)
(c)
legend: ◯ trapdoor
□ encrypted or prepared record
transformation/preparation process

Figure 5.1: An illustration that shows the difference between a single-key structure (a), a multi-key structure (b), and a structure with record preparation (c).

Squares represent (encrypted) records, circles represent readers and their trapdoors, and arrows represent transformation processes.

a single-key architecture, but that may just be the price for privacy. Moreover this cost is born by the server and not by the users.

In Chapter 4 we showed that the changes introduced in [PZ13] were not sufficient. In the MKSE protocol [PZ13], trapdoors from different readers are still applied on the same encrypted record and the server can see when two matching trapdoors are targeting the same keyword. We showed that the consequences of this approach on privacy are about as bad as with the single-key architecture. One solution would be that for each reader authorized to search some record, the encrypted record is transformed into a “copy” that is specific to this reader. We call this step **record preparation**, illustrated in part (c) of Figure 5.1. With record preparation, trapdoors from different readers are not applied on the same prepared record, and we will show that this allows to avoid privacy issues like the ones arising in MKSE.

Intuitively, record preparation in the protocols we present works in the same way as trapdoor transformation, and we have:

$$\mathsf{Transform}(\mathsf{Encrypt}(w,\rho),\gamma) = \mathsf{Prepare}(\mathsf{Encrypt}(w,\gamma),\rho) \tag{5.1}$$

Where w is some keyword, ρ is the reader key used to create trapdoors and γ is the record key used to encrypt keywords.

Naturally, record preparation results in a significant amount of additional computation and storage. In a system with M records, each having N authorized readers, $M \times N$ prepared records must be computed and stored. Again, we could simply say that this may be the price of privacy. Recall that, before this thesis, not a single MUSE protocol provided security against user-server collusions. Also, there does not seem to be any simple way to avoid this duplication of records. Note that the paper of Hamlin et al. [Ham+18a] supports our arguments: protocol HSWW18-I resorts to record duplication, and protocols HSWW18-II and HSWW18-III avoid duplication through the use of cryptographic obfuscation (see Section 3.3.2 of this manuscript) which as of today is still far from being practical. We see this as a confirmation that finding an efficient way out of duplication is highly non-trivial.

5.1.2 Non-Colluding Servers

A question that arises is then: Who will perform this record preparation step? The answer cannot be the readers or writers, as this does not seem practical with very large datasets. Having the server perform record preparation is also problematic: record preparation as we build it in our protocols processes records keyword-by-keyword (see Equation (5.1)) and the entity performing preparation is able to link a prepared keyword to the encrypted keyword it originates from, leading to the exact same privacy issue than in MKSE [PZ13]. Finding a record preparation process that hides such kind of information seems as hard as avoiding record duplication.

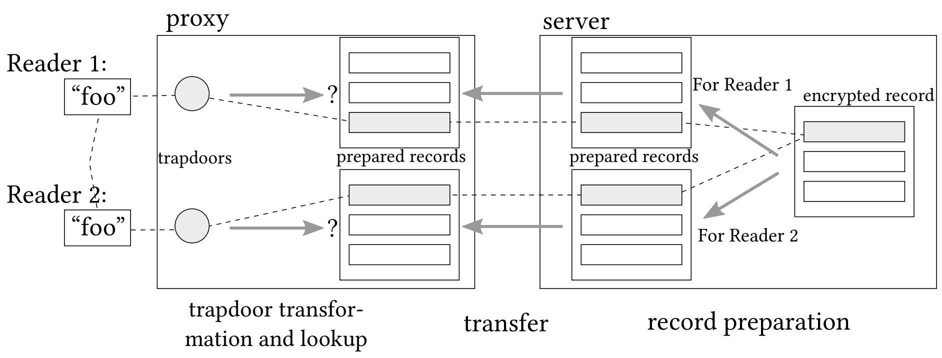


Figure 5.2: An illustration of our use of two non-colluding servers.

Here two trapdoors for the same keyword are sent by different readers. The proxy sees which prepared keywords these trapdoor match but does not see that these prepared keywords originate from the same encrypted keyword. As to the server, it sees this common origin but does not see the trapdoors and where they match.

There is, however, one simple solution that consists in having two separate servers: one server performs the record preparation and sends the prepared record to the other server, named the **proxy**, that transforms trapdoors and apply them on the prepared records. This solution is illustrated in Figure 5.2. The idea is that the proxy sees the matching prepared keywords but does not see which encrypted keywords they correspond to, while the server sees the relations between prepared and encrypted keywords but does not see the matches.

This separation requires that the two servers do not collude together. Before we describe our protocol in details, we explain why our threat model remains sound even with this additional assumption, and in particular why our protocols can be considered "more secure" than prior solutions. It may seem strange that, when trying to design MUSE protocols that protect against user-server collusions, we have to add an assumption that there are two servers not colluding with each other. However these are different kinds of collusions, and we argue that a collusion between two well-identified servers is much easier to prevent than a collusion between a server and any user. As already mentioned at the beginning of Section 4.1, users are typically the weakest point in a cloud computing system. Also, because there is a large number of users, there is a large number of possible user-server collusions to prevent. Monitoring each user seems impossible, but monitoring two CSPs to prevent them from colluding seems very realistic. This is especially true if CSPs are run by large, publicly known companies that must build and maintain a reputation and a relation of trust with their customers, while users are more likely to be anonymous, ephemeral entities. Finally, legitimate users have a common interest to prevent the servers from colluding with each other, so they can afford to pay a small fee to some independent audit company to monitor the servers.

Hence, while not having to rely on the assumption to have two non-colluding CSPs would definitely give a better threat model, adding this assumption seems worth the cost if this allows to build MUSE protocols that are secure against user-server collusions. Note that the protocols of Hamlin et al. [Ham+18a] achieve security against user-server collusions without having to rely on two separate servers, but this is essentially because the reader is in charge of record preparation, which is a major limitation regarding scalability. Using two servers can be seen as a way to "relieve" readers from this task, which is necessary in order to target large databases.

5.1.3 Search Pattern Privacy

In our two-server architecture, the servers cannot tell when two trapdoors from different readers contain the same keyword. Regarding trapdoors from a same reader however, so far the proxy would see if they are similar or not. Leaking such information –called the *search pattern* of the reader– is considered acceptable in most of the current literature on SE. However our record preparation mechanism gives us a simple way to obtain search pattern privacy with a very limited cost for the users.

A reader can simply create a new reader key and send it to the server. The server must then compute the prepared records corresponding to this new key and send them to the proxy, and the proxy can discard the previous prepared records and start using the new ones. This is equivalent to deleting a reader and creating a new one with the same authorizations. This operation creates a significant workload for the server, but this can be acceptable for two reasons: first, it does not occur frequently and second, it can be performed ahead of time as a long-term background task with a predictable amount of work and deadline that is known in advance, meaning that it does not impact search time and that resources will be cheaper.

About the frequency of key updates, the duration of the time period between two key updates is entirely up to the user. In the security analysis of our protocol in Section 5.2.2 we show that both servers get no information about the search pattern if a user never sends the same query twice during a same time period. This means that a user would typically set the time period depending on the number of queries (and probably results) she is able to remember.

Finally about the long-term and ahead-of-time nature of the task, it comes from the fact that the reader can create and send the new keys in advance, giving the server time to perform the preparation. It is actually better for security that the key update schedule is defined ahead of time, as a sudden request for a key update could indicate an intention of the user to re-send a previous query, defeating the point of the key update mechanism.

5.2 The DH-AP-MUSE protocol: fast MUSE with Access Pattern Leakage

Recall that H is a secure hash function (modeled as a random oracle) that hashes any bit string into a DDH-hard group $\mathbb{G}$ of order ζ. We describe the DH-AP-MUSE protocol using a lighter syntax than the one we defined in Section 3.2.1 because the simplicity of the protocol allows it. It should be easy to see that it can also be described with the defined syntax.

- The writer owning record $W_d \in \{0,1\}^*$ picks record key γ_d at random from $\mathbb{Z}_\zeta^*$. She encrypts this record into $\overline{W}_d$ by computing:

  $$\overline{W}_d \leftarrow \{H(w)^{\gamma_d} \ \forall w \in W_d\}$$

  She sends $\overline{W}_d$ to the server and γ_d to the proxy.

- The writer owning record W_d can authorize a reader r to search W_d simply by notifying the proxy and the server.

- For each time period l, reader r picks a random reader key $\rho_{r,l} \in \mathbb{Z}_\zeta^*$ and sends it to the server.

- When the server receives reader key $\rho_{r,l}$, it computes the prepared encrypted record $\overline{\overline{W}}_{d,r,l}$ for each $d \in Auth(r)$:

  $$\overline{\overline{W}}_{d,r,l} \leftarrow \{\overline{w}^{\rho_{r,l}} \ \forall \overline{w} \in \overline{W}_d\}$$

  The server sends $\overline{\overline{W}}_{d,r,l}$ to the proxy.

- $q_{r,l,s}$ denotes the s-th query of reader r during the l-th time period. For each such query, reader r creates the corresponding trapdoor $t_{r,l,s}$:

  $$t_{r,l,s} \leftarrow H(q_{r,l,s})^{\rho_{r,l}}$$

  $t_{r,l,s}$ is sent to the proxy.

- When receiving trapdoor $t_{r,l,s}$, the proxy performs the following steps for each record the querying reader is authorized to search, that is for each $d \in Auth(r)$:

  - the proxy computes the transformed trapdoor $t'_{r,l,s,d}$:

    $$t'_{r,l,s,d} = (t_{r,l,s})^{\gamma_d}$$

  - the proxy looks for value $t'_{r,l,s,d}$ in prepared record $\overline{\overline{W}}_{d,r,l}$. If the value is present, we say that W_d matches.

  The proxy sends the ids of the matching records to the querying reader.

We assume that a reader does not send similar queries during the same time period, that is, $q_{r,l,s} \neq q_{r,l,s'}$. However queries from different readers during the same time period can be similar, that is, we can have $q_{r,l,s} = q_{r',l,s'}$.

5.2.1 Similarities with the DH-PSI protocol

The protocol is similar to a Private Set Intersection (PSI) protocol presented in [HFH99] (see also [Kis+17]), which we call "DH-PSI", that is solely based on the Diffie-Hellman protocol. A (one-sided) PSI protocol involves a sender with set Y and a receiver with set X, and the receiver must learn $X \cap Y$ and the size of Y while the sender must learn nothing beyond the size of X. Remark that set membership test, which is what our MUSE protocol does, is a special case of set intersection: $q \in W_q$ is equivalent to $\{q\} \cap W_q \neq \emptyset$. In DH-PSI [HFH99], the receiver picks a random value $\alpha \in \mathbb{Z}_\zeta^*$ and sends $\{H(x)^\alpha \; \forall x \in X\}$ to the sender. The sender picks a random value $\beta \in \mathbb{Z}_\zeta^*$ and sends both $\{(H(x)^\alpha)^\beta \; \forall x \in X\}$ and $\{H(y)^\beta \; \forall y \in Y\}$. Finally the receiver computes $\{\left(H(y)^\beta\right)^\alpha \; \forall y \in Y\}$ and is able to see which elements of X are in Y without learning anything about the elements in $Y - X$. Interestingly, this protocol was shown in [Kis+17] to be the fastest existing PSI protocol when one set is much larger than the other one, which corresponds to our case because we are looking for a single value (the queried keyword) in a set (the record).

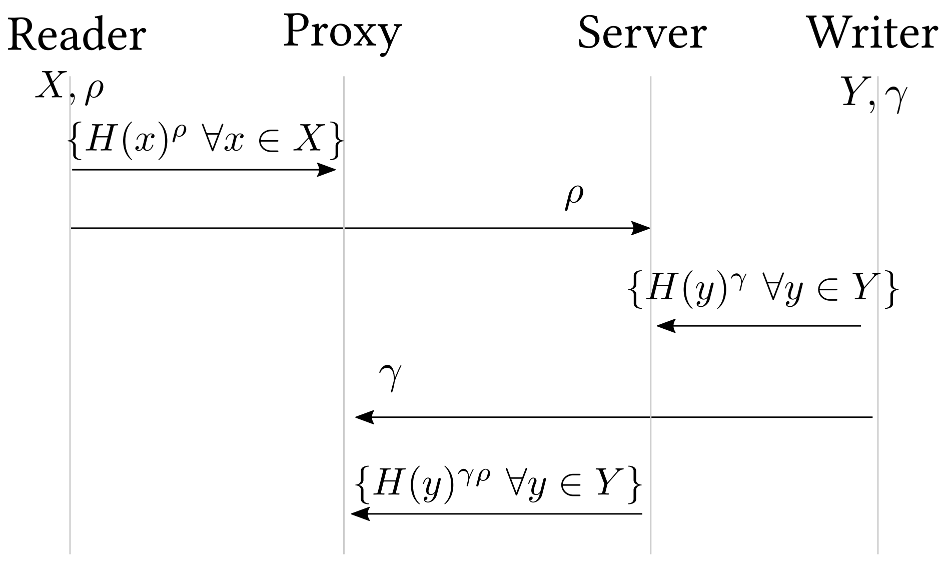

Reader
Proxy
Server
Writer
X, ρ
Y, γ
$\{H(x)^\rho \; \forall x \in X\}$
ρ
$\{H(y)^\gamma \; \forall y \in Y\}$
γ
$\{H(y)^{\gamma\rho} \; \forall y \in Y\}$

Figure 5.3: An illustration of the DH-AP-MUSE protocol using the notation of a PSI protocol.

Our protocol can actually be considered as an "outsourced" version of the protocol of [HFH99]. The reader of MUSE would be the receiver in PSI and the writer would be the sender, but instead of interacting together in a direct manner, the receiver sends her masked set to the proxy and her secret to the server while the sender sends her masked set to the server and her secret key to the proxy. Both the proxy and the server apply the key they received on the masked set they received in order to compute a "double-masked" set. Finally the proxy determines the intersection between the double-masked set it computed and the one transmitted by the server. The result is returned to the reader as the response to its query.

Figure 5.3 illustrates our protocol in a way that shows the similarities with the DH-PSI protocol. In particular, the reader is represented as having a set X, while in our case it would rather be a single element being the queried keyword. The secret of the writer is noted γ because it corresponds to the record key in a MUSE protocol, and the reader's secret is noted ρ because it corresponds to the reader key in our protocol.

5.2.2 Security Analysis

The security of the protocol derives from the hardness of the DDH problem in an almost obvious way. Intuitively, both the proxy and the server receive keywords that are "masked" by some key they do not know, the key being a reader key in the case of the proxy and a record key in the case of the server. However we still give a rigorous proof based on the simulation technique.

Remark on sending keys to the servers and the importance of the assumptions One may fear that giving secret keys to the servers (namely, record keys are given to the proxy and reader keys are given to the server) could allow attacks. For instance on reception of a trapdoor t from a reader authorized to search records №1 and 2 the proxy could, additionally to transforming it using the keys of records №1 and 2, transform it with the key of record №3 as well, that is, compute $(t)^{\gamma_3}$. Record №3 could be owned by a corrupted user colluding with the proxy and thus its content would be known entirely by the proxy. However this "additional" transformed trapdoor alone is not sufficient for the proxy to recover the content of the query or any other sensitive information. Trapdoor t was computed as $H(q)^{\rho}$ for some queried keyword q and the key ρ of the querying reader for the current time period. Since the proxy does not know ρ, t and $(t)^{\gamma_3}$ both look completely random to the proxy. The server, however, does know ρ, but our assumptions prevent the proxy from sending this additional transformed trapdoor $(t)^{\gamma_3}$ to the server: this would either be against our assumption that the adversary is honest-but-curious (this additional trapdoor is not a message that follows the protocol because the trapdoor was not meant to be transformed for record №3, and the server can verify that), or against our assumption that the proxy and the server do not collude (in case the server would willingly accept this illegitimate transformed trapdoor). A similar, symmetric situation appears if the server tries to create additional prepared records using additional reader keys: the obtained additional prepared records are useless without cooperation of the proxy. As a result, such attack are out of our threat model and are impossible as long as our assumptions hold, meaning that the proxy and the server are honest-but-curious adversaries that do not collude together.

Security Proof Warm-up

We prove security against the server and proxy separately, but because the two proofs are very similar we start by giving an overview of them.

In each proof we build a simulator following Definition 3.2.3 that takes the leakage as input and that outputs a simulated view. For privacy against the server, the only elements of the view that are not trivial to build are the encrypted keywords of non-revealed records. Indeed the simulator is given the content of revealed records, and it has all the keys since it is the one creating them. For privacy against the proxy, the elements of the view that are not trivial to build are the non-revealed trapdoors and the non-revealed prepared records, but also the prepared records corresponding to records being revealed but to readers that are not corrupted, meaning non-revealed trapdoors will be applied to these prepared records.

Most of these elements that are not trivial to build are generated by replacing the call to hash function H, modeled as a programmable Random Oracle (RO), by a uniform random sampling from $\mathbb{G}$. We say "most of them" because some prepared keywords in the simulated view of the proxy are instead generated by taking a trapdoor and transforming it (using the record key), in order to have the trapdoor matching the resulting prepared record so that access pattern is preserved.

We show that the output of simulators are indistinguishable from the real view of the adversary with a sequence of hybrid simulators where each simulator simulates one more element of the view than the previous one. The output of any two successive hybrid simulators are shown to be indistinguishable with a reduction to the DDH problem in $\mathbb{G}$, using the following embedding of a DDH instance g^a, g^b, g^c:

$$H(w^*) \leftrightarrow g^a, \quad \gamma^* \leftrightarrow b, \quad \overline{w}^* \leftrightarrow g^c$$

Where w^* is the keyword corresponding to the trapdoor (or prepared keyword or encrypted keyword, depending on the case) that is simulated in one hybrid simulator but not in the other, called the **pivot** trapdoor/prepared keyword/encrypted keyword, and γ^* (or ξ^* for privacy against the proxy) is the record key (resp. reader key) corresponding to the pivot element. The embedding is done by programming the RO as follows: On input w^* it outputs g^a, and on any other input it outputs $g^{\mathcal{O}[w]}$ where $\mathcal{O}[w]$ is previously picked uniformly at random if it was not already set, as is usually done with ROs. There is a subtlety in the programming of the RO because it must be programmed before the value w^* is available for the reduction. We give more details on this point further below.

The embedding of b is made possible by the way we define the RO: encrypting a keyword using b as the record key or reader key is done with the following function that uses g^b which is known to the reduction algorithm:

$$w \mapsto (g^b)^{\mathcal{O}[w]}$$

Finally the embedding of c is done by using g^c as the pivot trapdoor/prepared keyword/encrypted keyword.

The only aspect that remains to be addressed in each of the proofs is the correctness of the embedding, essentially making sure that the value we replace by g^c does not appear anywhere else. Hence the requirement that records do not contain duplicates and that a reader does not send two identical queries during the same time period.

About programming the RO Note that programming the random oracle requires to know the value of w^*. The reduction will only know this value when $\mathcal{D}_1$ returns (see Definition 3.2.3), while it has to program the oracle before $\mathcal{D}_1$ starts. Popa and Zeldovich suggest in [PZ13] a way to overcome this difficulty: The reduction makes a "bet" on which query to the oracle will be for the keyword that will end up being w^*. It can also bet that $\mathcal{D}_1$ will never query the oracle for this keyword. For instance the reduction can bet that the keyword given for the 3rd query to the oracle will be the same keyword occupying the position corresponding to w^* in the history produced by $\mathcal{D}_1$. When $\mathcal{D}_1$ returns, the reduction can check whether its bet was correct or not. If it was, the reduction can continue, otherwise it halts and gives a random answer to the DDH problem. If the bet was that the keyword is not queried and the bet was correct, this means that the reduction can program the RO using the value of w^* present in the history returned by $\mathcal{D}_1$. Because $\mathcal{D}_1$ runs in polynomial time, there are only a polynomial number of possible bets, thus a non-negligible advantage of the distinguisher still results in a non-negligible advantage of the reduction.

Privacy Against the Server

We show that DH-AP-MUSE has a leakage no greater than the benign leakage and the revealed content against a honest-but-curious server that colludes with any number of users but not with the proxy. Note that there is no access pattern in this leakage: in this protocol, only the proxy sees the access pattern. The corresponding simulator is described in Algorithm 4.

```
Algorithm 4: Simulator for the server view
  Input: The benign leakage and revealed content
  Create all record keys and all reader keys ;
  for each record id $d$ do
      if $W_d$ is revealed then
          Encrypt it using the record key $\gamma_d$ previously generated;
      else
          Set $W_d$ to a set of random bit strings
          using the length of $W_d$ from the benign leakage;
          Encrypt $W_d$
  Output: All encrypted records, all reader keys and the record keys of revealed records
```

We show that the output of this simulator is indistinguishable from a real view using a sequence of hybrid simulators, where each of them simulates one more non-revealed encrypted keyword than the previous simulator. As a consequence the first hybrid simulator corresponds to the real view and the last one corresponds to Algorithm 4. All simulators have the entire history as input except the last one that only has the benign leakage and revealed content.

We then show that the output of two successive simulators are indistinguishable using a reduction to the DDH problem in $\mathbb{G}$. The reduction performs the following embedding of a DDH problem instance g^a, g^b, g^c as was previously described:

$$H(w^*) \leftrightarrow g^a, \quad \gamma^* \leftrightarrow b, \quad \overline{w}^* \leftrightarrow g^c$$

Where w^* is the "pivot keyword" that is simulated in one simulator but not in the other (for instance this could be *"the third keyword of the second non-revealed record"*).

The embedding is correct if the view corresponds to the output of one simulator in the case where $c = ab$ and the other simulator in the case where c is random. The only difference between these two outputs is that the pivot encrypted keyword $\overline{w}^*$ is simulated in one simulator and properly generated in the other. All other values of the simulator output must be the same whatever the answer to the DDH problem is. As a result we must check that the value $\overline{w}^*$ does not appear anywhere else in the output. This is satisfied thanks to the fact that records are represented as sets in our protocol, that is, they do not have duplicate elements. As a result, any keyword w of the pivot record must be different from w^* and its encrypted keyword will be either generated as $(g^b)^{\mathcal{O}[w]}$ or with random sampling. Keywords in other records correspond to a different record key so their encryption/simulation is not a problem either.

As a result distinguishing the output of two successive simulator is at least as hard as solving the DDH problem in $\mathbb{G}$, thus the output of Algorithm 4 is indistinguishable from a real view, and this ends the proof.

Privacy Against the Proxy

We now show that DH-AP-MUSE has a leakage no greater than the access pattern, the benign leakage and the revealed content against a honest-but-curious proxy that colludes with any number of users but not with the server. The corresponding simulator is described in Algorithm 5.

```
Algorithm 5: Simulator for the proxy view
  Input: The access pattern, the benign leakage and revealed content
  Create all record keys and all reader keys ;
  for each $r, l, s$ do
      if $q_{r,l,s}$ is revealed then
          Create $t_{r,l,s}$ as normal;
      else
          Create $t_{r,l,s}$ as a random element of $\mathbb{G}$;
  for each record id $d$, each $r$ such that $d \in Auth(r)$ and each $l$ do
      if $W_d$ is revealed and $r$ is corrupted then
          Encrypt and prepare as normal;
      else
          Initialize $\overline{\overline{W}}_{d,r,l}$ as an empty set;
          for each $s$ such that $d \in a_{r,l,s}$ (known from the access pattern) do
              Add $(t_{r,l,s})^{\gamma_d}$ to $\overline{\overline{W}}_{d,r,l}$;
          Add random elements to $\overline{\overline{W}}_{d,r,l}$ until it has the proper size (known from the benign
            leakage);
  Output: All $t_{r,l,s}$ values, all $\overline{\overline{W}}_{d,r,l}$ values, all record keys $\gamma_d$ and the keys of corrupted
          users
```

This time we show that the output of Algorithm 5 is indistinguishable from a real view of the proxy using not one but two sequences of simulators: The first sequence will correspond to the simulation of prepared records and the second sequence to the simulation of trapdoors. The first simulator of the first sequence corresponds to the real world experiment, and does not simulate any prepared keyword nor any trapdoor. Then, each simulator in the first sequence will simulate one more prepared keyword than the previous simulator, until all prepared records that are simulated in the final simulator are simulated (see Algorithm 5). The first simulator of the second sequence is the last simulator of the first sequence, that is, it simulates prepared records in the same manner as the final simulator but simulates none of the trapdoors. Finally each simulator in the second sequence simulates one more trapdoor than the previous simulator. As a result the last simulator of the second sequence is Algorithm 5.

We start by showing that two successive simulators from the first sequence have indistinguishable outputs. The pivot keyword is characterized by d^*, i^*, r^*, l^* such that the second simulator simulates prepared keyword $\overline{\overline{W}}_{d^*,r^*,l^*}[i^*]$ but the first one does not. If $\overline{\overline{W}}_{d^*,r^*,l^*}[i^*]$ matches a trapdoor, the output distributions of the two simulators are more than indistinguishable, they are identical. Indeed in this case the second simulator will not generate this prepared keyword at random but by transforming the corresponding trapdoor, and the resulting value will be the same as what the first simulator would have obtained, as a consequence of the correctness of the protocol.

If $\overline{\overline{W}}_{d^*,r^*,l^*}[i^*]$ does not match any trapdoor though, the second simulator will simulate it through random sampling, and again we show the two outputs are indistinguishable with a reduction to the DDH problem. This time the embedding of the DDH instance g^a, g^b, g^c is as follows:

$$H(W_{d^*}[i^*]) \leftrightarrow g^a, \quad \xi_{r^*,l^*} \leftrightarrow b, \quad \overline{\overline{W}}_{d^*,r^*,l^*}[i^*] \leftrightarrow g^c$$

Again, the correctness of the embedding of the DDH instance requires that the reduction does not have to use the value g^c for anything else than the pivot prepared keyword. The argument for this is the same as for privacy against the server: a keyword does not appear twice in a record, and other (prepared) records will use different record keys (or different reader keys).

Finally, we show the indistinguishability of the output of two simulators from the second sequence. This time there are r^*, l^*, s^* such that the second simulator simulates t_{r^*,l^*,s^*} but the first one does not. The embedding used is the following:

$$H(q_{r^*,l^*,s^*}) \leftrightarrow g^a, \quad \xi_{r^*,l^*} \leftrightarrow b, \quad t_{r^*,l^*,s^*} \leftrightarrow g^c$$

Here the correctness of the embedding comes from our assumption that a reader will not send two identical queries in the same time period, and this ends the proof.

5.2.3 Performance Analysis

Intuitively, what makes the protocol scalable is that the workload of a writer is linear with the number of keywords she uploads, the workload of a reader is linear with the number of queries she sends, the workload of the server is a long-term task which does not affect search time, and the workload of the proxy is no greater than the one of the server in MKSE [PZ13].

To give a more precise performance evaluation, we consider a system with A writers owning B records each, each record containing N keywords, and C readers each having access to D records. We assume that all readers have the same time period.

In our protocol, each writer performs $B \times N$ exponentiations and hashing in $\mathbb{G}$. The server performs $C \times D \times N$ exponentiations in $\mathbb{G}$ for each time period, and the proxy must perform D exponentiations for each trapdoor it receives. The workload of readers is only a single exponentiation and hashing per trapdoor, and response reception requires essentially no resources (the final response is received in plain text).

Note that preparation and transformation are tasks that are "embarrassingly parallel", meaning that they can be parallelized with no effort. They also have strong data locality, meaning that each elementary task is applied on a small portion of the whole dataset, allowing the use of distributed infrastructures like MapReduce. Also, record preparation, performed by the server, is a predictable amount of work without any burst and with long-term deadlines. Resources tend to be cheaper for this kind of workload[1]. Finally, note that the proxy can discard prepared records at the end of each time period, making its space consumption about the same as the server.

We stress that DH-AP-MUSE should have faster search operations than MKSE. MKSE does not have a record preparation step, but this makes no difference regarding search speed because preparation is processed ahead of time and prepared records are sent to the proxy. Search in DH-AP-MUSE is then very similar to search in MKSE, with the trapdoor being transformed for each prepared record to search. Now, each such transformation requires the computation of a bilinear pairing in MKSE while DH-AP-MUSE, which does not use pairings, only requires an exponentiation in the DDH-hard group. Even if MKSE could benefit from pairing pre-computation [CS10] since the second input of the pairing is fixed, the exponentiation operation in DH-AP-MUSE can only be faster as bilinear pairings are notoriously slow to compute. It is then interesting to remark that DH-AP-MUSE seems to outperform MKSE both in terms of privacy *and* in terms of complexity, search speed being arguably more important than server storage for most use cases of MUSE.

Implementation and performance measurements

Another advantage of DH-AP-MUSE is its great simplicity, which makes its implementation an easy task. We implemented the algorithms of DH-AP-MUSE in less than 100 lines of C, using the Sodium crypto library [libsodium].

Performance measurements on a Amazon EC2 t2.micro instance[2] gave the following running times:

- trapdoor generation and keyword encryption: 0.1 ms per keyword
- record preparation (including insertion of prepared keywords in a bloom filter): 60 μs per keyword
- trapdoor transformation: 60 μs per transformation

This means that a server hosted on a single t2.micro instance could handle (in terms of computation) the preparation of records for 100 readers each having access to 40,000 records assuming records of 10,000 keywords each and a time period of one month. The same machine should be able, as a proxy, to transform the trapdoor of a reader searching 40,000 records in under 3 s. Using a machine with a faster CPU (t2.micro has a frequency of 2.40 GHz) or with more cores (using multi-threading) should scale capacity accordingly. “Scaling out” (increasing the number of machines working on the task in a parallel fashion) should also increase the capacity in a linear fashion due to the embarrassingly parallel nature and high data locality of the task.

5.2.4 Further Remarks

Our work on the DH-AP-MUSE protocol, corresponding to this section (Section 5.2), will be presented at the ICICS 2018 conference [RMÖ18a].

[1]see https://www.slideshare.net/rasmusekman/aws-an-introduction-to-bursting-gp2-t2

[2]`https://aws.amazon.com/ec2/instance-types/`

5.3 The RMÖ15 protocol: Very low leakage at the price of scalability

The DH-AP-MUSE protocol has a very low cost for users and a moderate cost for the servers, but it leaks the access pattern. While leaking the access pattern is still considered acceptable in the literature on SE and MUSE, the increasing number of access-pattern-based attacks (see Section 3.4) makes it necessary to study protocols with smaller leakage profiles, in case these attacks become too much of a concern.

In this Section we show that, with a small and simple modification of DH-AP-MUSE, we obtain a MUSE protocol in which none of the servers gets more information than the benign leakage and the revealed content. This however is at the cost of a reader having to process one response for each searched record. While this side-effect on scalability is unavoidable, we give solutions to lower this user-side cost.

5.3.1 Idea of the Protocol

We want to prevent the proxy from learning anything about the result of the queries. Recall that in DH-AP-MUSE, the proxy sees which records match a trapdoor and simply sends the ids of these matching records to the reader the trapdoor came from. We proposed a protocol that we name RMÖ15 in which the proxy returns encrypted values it cannot decrypt and it is the reader that decrypts them and gets the query result.

Requiring that the proxy does not learn *anything* about the query results makes it unavoidable that the reader must process an amount of data that is linear with the number of records that are searched: Indeed if the amount of data sent to the reader was depending on the number of matching records, the proxy would get an idea of this number and this is already more information than what we are willing to reveal. In the next chapter, we study protocols where we allow the proxy to learn the number of matching records in order to improve the scalability, and show that it is much more challenging to design such protocols yet feasible.

The technique we use to prevent the proxy from seeing the query result, illustrated in Figure 5.4 is the following: the server, after computing a prepared record, encodes the prepared record using a hashing structure called Bloom Filter (BF) which we present in details in the next section. A BF consists of an array of buckets and allows one to test the presence of an element in the encoded set by mapping the element to a number of buckets and checking the content of these buckets. The server encrypts the BF bucket-by-bucket before sending it to the proxy to prevent the proxy from learning anything about the prepared record, using a standard IND-CPA symmetric cipher and a key, called **transmission key**, that is known to the querying reader but not to the proxy.

As to the proxy, after transforming a trapdoor using some record key, it performs the first part of the procedure for looking up a BF: it hashes the transformed trapdoor to get the positions of the corresponding buckets and reads these buckets. Because these buckets are encrypted it cannot do anything with them and simply sends them to the querying reader. The reader is then able to finish the BF lookup procedure by decrypting the buckets using her transmission key and combining them together to decide whether the queried keyword is present in the corresponding record or not.

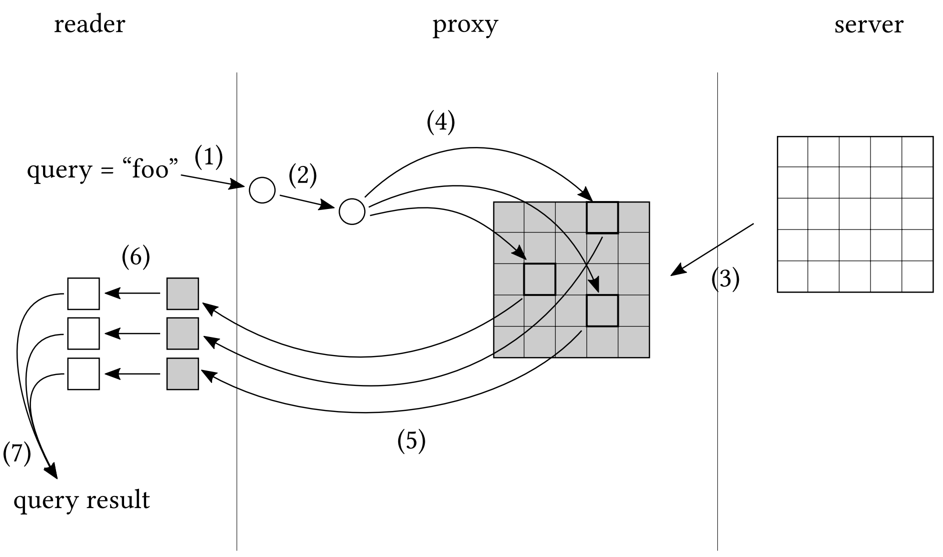


Figure 5.4: An illustration of the RMÖ15 protocol.

Gray filling represents encryption under the transmission cipher. The steps represented are (1) trapdoor creation, (2) trapdoor transformation, (3) the bucket-by-bucket encryption of the BF encoding the prepared record, (4) the mapping of the transformed trapdoor into positions on the BF, (5) the transmission of the mapped buckets to the reader, (6) the decryption of these buckets and (7) the combination of these buckets to decide on the presence or absence of the queried keyword.

5.3.2 Bloom Filters

Introduced by Bloom in [Blo70] and further studied by Broder and Mitzenmacher in [BM03], Bloom Filters (BFs) are a hash structure that aims at efficiently testing membership in a set. A BF is an array B of M bits associated with η random hash functions $h_1, \ldots, h_\eta : \{0,1\}^* \to [M]$. B is initialized by setting all the array values to zero and one inserts an element $x \in S$ in B by setting $B[h_i(x)]$ to 1 for all i. Finally one checks the presence of x in the set S encoded by B by testing whether $B[h_i(x)]$ is equal to 1 for all $i \in [\eta]$; if this is not the case then x cannot be in S, otherwise x is in S with high probability. Following [RR17] we use the notation h_* to denote the set of all positions corresponding to an element x or a set S:

$$h_*(x) = \{h_i(x)\ \forall i \in [\eta]\}$$

$$h_*(S) = \bigcup_{x \in S} h_*(x)$$

We will denote by BF_S the Bloom Filter encoding set S when there is no ambiguity about what parameters M and $(h_i)_{i\in[\eta]}$ where used.

The event that x appears to be encoded in B while it is actually not in S is called a *false positive*. Dong et al. [DCW13] show that the probability for $x \notin S$ to cause a false positive is negligible in the number of hash functions η. As a consequence, setting the number of hash functions as greater or equal to the security parameter results in a false positive probability negligible in the security parameter.

Broder and Mitzenmacher in [BM03] show that the optimal value for η, that minimizes the false positive probability for a given M and set size N, is:

$$\eta = \ln(2)\frac{M}{N} \ . \tag{5.2}$$

They also show that with this value of η about half of the bits are set after insertion of all the elements in S.

5.3.3 Protocol Description

We only describe hereafter the differences with respect to the DH-AP-MUSE protocol presented in Section 5.2. In addition to the building blocks of DH-AP-MUSE, the RMÖ15 protocol uses Bloom Filters parametrized by M and $(h_i)_{i\in[\eta]}$, as well as a symmetric encryption scheme (CPA.KeyGen, CPA.Encrypt, CPA.Decrypt) satisfying IND-CPA security.

- Each reader r generates a key k_r called *transmission key* using algorithm CPA.KeyGen, additionally to the other keys specified in DH-AP-MUSE. The transmission key does not need to be refreshed at each time period.

- The server, after computing some prepared record $\overline{\overline{W}}_{d,r,l}$, encodes it into a BF $B_{d,r,l}$ computed as follows:

$$B_{d,r,l} \leftarrow BF_{\overline{\overline{W}}_{d,r,l}}$$

The server then encrypts the BF bucket-by-bucket into $\overline{B}_{d,r,l}$:

$$\overline{B}_{d,r,l}[i] \leftarrow \mathsf{CPA.Encrypt}(k_r, B_{d,r,l}[i])$$

The encrypted BF $\overline{B}_{d,r,l}$ is then sent to the proxy.

- The proxy, after computing the transformed trapdoor $t'_{r,l,s,d}$, finds the BF positions corresponding to this value and extracts the buckets of $\overline{B}_{d,r,l}$ corresponding to these positions:

  $$p_{r,l,s,d} \leftarrow \{\overline{B}_{d,r,l}[i] \; \forall i \in h_*(t'_{r,l,s,d})\}$$

  $p_{r,l,s,d}$ is sent to reader r.

- Reader r, upon receipt on $p_{r,l,s,d}$, decrypts and combines the buckets to finish the BF lookup procedure:

  $$a_{r,l,s,d} \leftarrow \text{True if } \mathsf{CPA.Decrypt}(k_r, x) = 1 \; \forall x \in p_{r,l,s,d}, \text{ False otherwise}$$

  The final response $a_{r,l,s}$ consists of all record ids d such that $a_{r,l,s,d}$ is true.

5.3.4 Security Analysis

The security proof for this protocol is very similar to the one for DH-AP-MUSE. Therefore, we only give the differences with respect to DH-AP-MUSE instead of re-writing a complete proof.

Privacy against the server is the same as in DH-AP-MUSE. This is obvious as the view of the server is unmodified except the global BF parameters and the transmission key of each reader. These values are independent from the history and from the other values in the view of the server, so a simulator can generate them in a trivial manner. This suffices to show that the server in the improved RMÖ15 protocol does not get more information than in DH-AP-MUSE, that is the benign leakage and the revealed content.

Regarding privacy against the proxy, the proof has few differences with the one of DH-AP-MUSE. In RMÖ15, the simulator does not have to ensure that the view it outputs matches the access pattern, because the proxy does not see the access pattern. As a result, the simulator simply generates random BFs and encrypts them using the transmission cipher. Prepared records corresponding to a revealed record and a corrupted reader are still generated as in the real view, like in DH-AP-MUSE. The simulation of encrypted BFs is unnoticeable for the distinguisher thanks to the IND-CPA security property of the transmission cipher, and we can use a sequence of hybrid simulators as we did in the security analysis of DH-AP-MUSE. Note that the encrypted BF does not need to satisfy any condition on the access pattern, and thus the simulator does not need the access pattern in its input. Hence the proxy in the improved RMÖ15 protocol learns nothing about the history beyond the benign leakage and the revealed content. This is what makes the privacy guarantees of RMÖ15 higher than the one of DH-AP-MUSE.

5.3.5 Optimization using a stream cipher

Because BF buckets contain a single bit, for efficiency reasons it is much better to use a stream cipher for implementing the CPA cipher. We quickly recall that a stream cipher expands a symmetric key and a nonce into a long, random-looking keystream that is combined with the plaintext using a bit-wise XOR operation to produce the ciphertext. A typical example of a modern stream cipher is Salsa20 [Ber08].

A stream cipher is a perfect fit for the implementation of the CPA cipher in our RMÖ15 protocol thanks to the bit-wise nature of its encryption: The i-th bit of $\overline{B}_{d,r,l}$ corresponds to the i-th bucket of $B_{d,r,l}$. This would not be the case with a block cipher where a single bit of the output, taken separately, does not correspond to anything. With a block cipher, either each one-bit bucket of $B_{d,r,l}$ would be encrypted as a 128-bits ciphertext (a typical output size for modern block ciphers), or buckets would be grouped together before being encrypted but the proxy would still be forced to send 128-bit ciphertexts to the reader.

Using a stream cipher, it is much easier to achieve efficiency. The proxy simply sends

$$\{\overline{B}_{d,r,l}[i] \ \forall i \in h_*(t'_{r,l,s,d})\}$$

that is only η bits long, together with the nonce used by the server to encrypt $\overline{B}_{d,r,l}$ and the buckets positions $h_*(t'_{r,l,s,d})$. The reader uses the nonce and her transmission key to re-generate the keystream, and is able to recover each plaintext bucket by extracting the corresponding bit of the keystream and XORing it with the encrypted bucket.

For a large M (number of buckets in a bloom filter), re-generating the entire keystream can be costly, while only η bits of this keystream are used. In this case the use of a cipher with random read access to the keystream (which is the case for Salsa20 [Ber08] and of the counter mode of operation for block ciphers [Bel+97]) can increase efficiency event further. With random reads the reader is able to re-generate only the blocks of the keystream that correspond to the positions of the received buckets, resulting in a constant cost regardless of the size of the BF.

5.3.6 Performance Analysis

The cost for the server and the proxy is essentially the same as in DH-AP-MUSE. Regarding the server, the cost of encrypting a BF is negligible compared to the cost of generating the prepared record first. Regarding the proxy, again the difference in cost should be negligible.

As we already explained in Section 5.3.1, the cost for the reader is linear with the number of searched records. However with the use of a stream cipher as described above, the cost per search record is quite limited: The amount of data received is of $\eta + 64 + \eta \log_2(M)$ bits (respectively for the encrypted buckets, the nonce (taking nonce size for Salsa20 [Ber08]) and the buckets positions) and computation only consists in generating at most η blocks of the keystream, then XORing each encrypted bucket with its corresponding keystream bit, and finally testing whether all decrypted buckets are equal to 1. Computation should likely be accelerated through a clever use of CPU instructions.

5.3.7 Further Remarks

Enhancements over the [RMÖ15] article The RMÖ15 protocol was originally presented in a paper [RMÖ15], though in a more complicated and less efficient version. In [RMÖ15], the prepared index is kept by the server and the proxy retrieves the buckets using a Private Information Retrieval (PIR) protocol. Privacy against the proxy is obtained by having the server encrypt the PIR responses using the transmission cipher, and it is these encrypted responses that the proxy sends to the querying reader.

In [RMÖ15] we made use of a specificity of the PIR protocol in order to reduce its cost. The proxy only sends half of a normal query, which halves the time necessary for the server to apply the query. The response is then a vector of ciphertexts, only one of which contains the desired bucket (no technique is present in [RMÖ15] to retrieve all η buckets with a single PIR query). The proxy knows which of these ciphertexts should contain the desired bucket, and can discard all the other ones, thus sending only one ciphertext per bucket to the reader.

Despite these techniques, the version of the RMÖ15 protocol presented in this manuscript is much faster than the version of [RMÖ15]. First, the ciphertexts received by the reader in [RMÖ15] are not ciphertexts of a symmetric cipher as in the version of this manuscript, but an additively homomorphic cipher (see Section 6.2.2 of this manuscript), which are much longer and for which decryption is costlier. Also, the fact that searching in the version of [RMÖ15] requires some interaction between the server and the proxy makes a great difference in terms of speed, no matter how powerful the two servers are. Recall that in RMÖ15, just as in DH-AP-MUSE, interaction between the server and the proxy happens ahead of the search operation which only involves a reader and the proxy.

Another difference between the RMÖ15 protocol as presented in this manuscript and the version of [RMÖ15] is that in [RMÖ15], record preparation is performed for every new query. It is only later that we realized that record preparation could be done only at regular time intervals as long as a reader was not sending the same query twice in the same time period.

Finally, the protocol presented in [RMÖ15] makes use of bilinear pairings because the trapdoor transformation mechanism of MKSE [PZ13] had be re-used "as is". Pairings appear to be unnecessary in all the protocols we designed in this thesis, and a standard DDH-hard group suffices, which makes operations much cheaper.

Implementation The RMÖ15 protocol as presented in [RMÖ15] was implemented in Python using the "Charm crypto" framework [Charm-Crypto] as a deliverable to the User Centric Networking (UCN) European research project [UCN]. The main goal of the UCN research project was the development of a platform called *Personal Information Hub* (PIH), essentially a small computer located at a user's home that collects, stores and manages all of the user's private data and interacts with various cloud service providers. The rationale behind such infrastructure is to enable the user with a complete control over who accesses her personal data and how it is accessed.

Figure 5.5 shows the location of the "MUSE" primitive in the architecture of the UCN project. MUSE was used the following way in UCN: the PIH would be a writer in a MUSE system, uploading records corresponding to various parts of the user's personal data. In a second time, several data analysis companies could negotiate access to this records with the user. One concrete example that was studied in the UCN project was movie recommendation, where the user's PIH would upload records containing information about the user's preference and history regarding movies. For instance one record could be the set of favourite movies, another one the set of all movies watched in the last 6 months, etc... Then, a data analysis company could negotiate access to such records for a large number of users, and search them using MUSE.

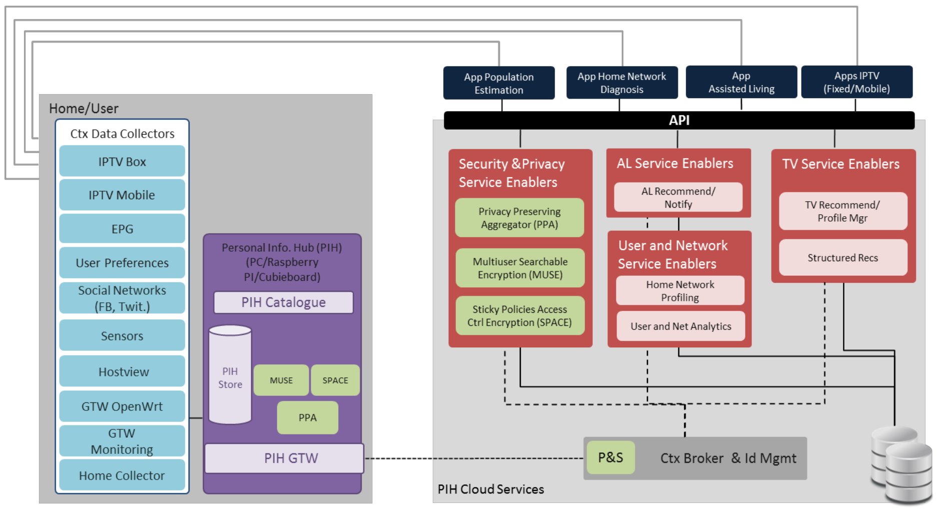


Figure 5.5: The UCN Reference Architecture
(source: [UCN Final Report])

5.4 Conclusion

This chapter shows that accepting to rely on two non-colluding servers and to replicate an encrypted record for each reader that was given access to it appear to be a mandatory trade-off for the design of secure MUSE protocols given the current state of the art. Until a breakthrough occurs in the field, such trade-off seem to be the only way to reach practical MUSE protocols that can protect the privacy of records and queries in a realistic threat model that takes user-server collusions into account.

The two protocols that we presented, DH-AP-MUSE and RMÖ15, are both simple to implement and to use and secure against user-server collusions, addressing different requirements at the discretion of MUSE users. DH-AP-MUSE should have a speed comparable to the previously existing MUSE protocols, outperforming MKSE in terms of search time and being only a bit slower than single-key protocols. RMÖ15 should be slower with readers having limited capacities and searching a very large amount of records, but the information leakage to the servers is extremely small, making it a safer option in the face of leakage abuse attacks based on access pattern leakage.

Figure 5.6 represents this situation, locating our two protocols DH-AP-MUSE and RMÖ15 in the privacy/efficiency graph, comparing them to the prior art that was already located in Figure 4.5.

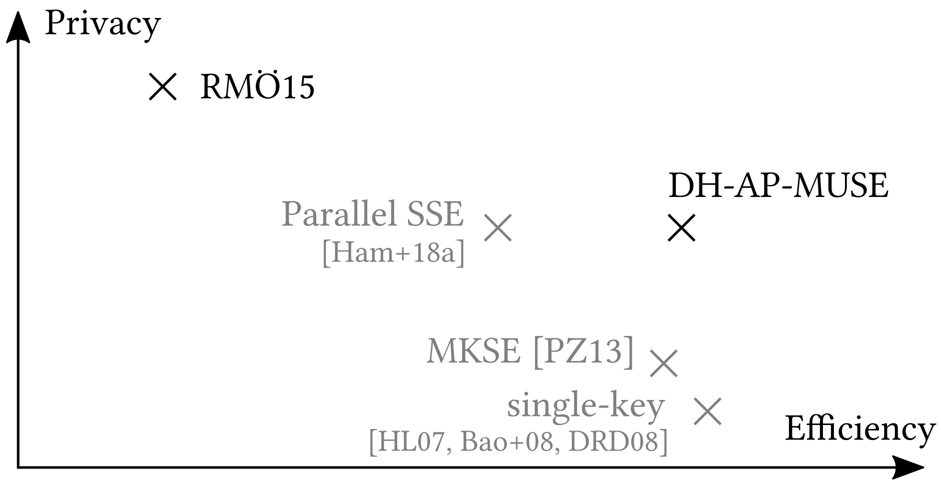


Figure 5.6: Location of protocols RMÖ15 and DH-AP-MUSE regarding the privacy-versus-efficiency dilemma

6 A Secure yet Scalable MUSE protocol

The two protocols presented in the previous chapter, DH-AP-MUSE and RMÖ15, have very different locations in the privacy/efficiency graph, as it can be seen in Figure 5.6: DH-AP-MUSE scales very well but leaks the access pattern of queries, while the RMÖ15 protocol has a very low leakage profile at the price of scalability issues.

In this chapter we describe a MUSE protocol, named RMÖ18, that reaches a middle ground between these two extremes. RMÖ18 has a better leakage profile than DH-AP-MUSE while having a better scalability than the RMÖ15 protocol. Unlike DH-AP-MUSE and RMÖ15 that were quite simple, RMÖ18 is very involved. Also, it depends on new primitives, or rather the modification of existing primitives such as Garbled Bloom Filters and Oblivious Transfer.

While the complexity of the RMÖ18 protocol makes its implementation more challenging than the protocols presented in the previous chapter, we see it as a promising solution for the future when access pattern attacks will become more powerful and protocols like DH-AP-MUSE will not be considered sufficiently secure any more. Indeed, RMÖ18 prevents access pattern attacks unlike DH-AP-MUSE while remaining efficient at the user side unlike RMÖ15.

6.1 Result Filtering, Result Length Leakage and Response Unlinkability

The main shortcoming of the RMÖ15 protocol regarding scalability is that the workload of the reader using RMÖ15 is linear with the number of records that are searched. This goes directly against our main objective that is to enable users with limited resources to search datasets composed of a great number of records. We call **result filtering** the fact that a reader receives a response whose size depends on the number of matching records instead of the total number of records being searched.

As already mentioned in Section 6.1, result filtering in a MUSE protocol implies that the server (or in our case the proxy) learns at least the number of records that match a query, an information called **result length**. It is sufficient for the server to measure the size of the response sent to the reader to discover the result length.

We thus set the following goal: we aim at designing a MUSE protocol with result filtering whose leakage profile is limited to the result length (and, as always, the benign leakage and the revealed content). Note that this is a rather challenging goal, as we set the leakage profile to the lowest possible. It could be interesting as future work to see if a slightly greater leakage profile could allow better performances without sacrificing too much privacy.

In the protocols presented in the previous chapter, the proxy did not need to interact with the server during a search operation. All the work required from the server was performed ahead of time, and the transformation of trapdoors as well as their application to prepared records only involved the reader and the proxy. With our new goal of combining result filtering and strict result length leakage, it seems very difficult to avoid proxy-server interaction during the processing of a query. The reasons are similar to what makes it difficult to design a MUSE protocol with a single server, which we covered already in Section 5.1.2: if the proxy was able to transform and apply a trapdoor without any interaction with the server, and because it sees the result length, it could "replay" the transformation and application process on only a part of the records that were supposed to be searched, without telling the reader, in order to see how this affects the result length. Repeating this process gives a way to identify which records are matching the query and which are not, meaning that the proxy learns the access pattern of queries, and the goal of having a leakage profile limited to result length cannot be reached any more. One solution would be to prevent the proxy from transforming and applying trapdoors on a subset of the targeted records, but we do not see any simple way to do so. Research on such solutions would be an interesting topic for future work.

Having a lookup protocol that requires interaction between the proxy and the server, on the other hand, gives a simple way to prevent this type of issue. The proxy could not replay lookups because doing so would require the cooperation of the server, which goes against our assumption that they do not collude together, or would require to "trick" the server by sending forged messages, which goes against our assumption that adversaries are honest-but-curious. As a result we are looking for a solution where the proxy transforms a trapdoor, then sends some query to the server, and finally receives a reply from which it can learn nothing except the result length.

The fact that the previous protocols process each transformed trapdoor separately can be seen as an obstacle to revealing only the result length. Nevertheless, even if the proxy receives a separate response from the server for each transformed trapdoor (meaning one response for each record being searched), and even if it is able to see whether each response is positive or negative, it is unable to learn the access pattern if it cannot associate a response with a query. We call this property **response unlinkability**.

With a lookup protocol satisfying response unlinkability, the design of a MUSE protocol satisfying our requirements seems feasible, as illustrated in Figure 6.1: the proxy sends a sequence of queries, one for each record that must be searched, and receives a sequence of responses consisting of an encrypted result and an encrypted id. The proxy can decrypt the encrypted result to see if the response corresponds to a match or a miss, but cannot decrypt the encrypted id. It simply forwards the encrypted ids corresponding to positive responses to the querying reader, which is able to decrypt them and get the query result.

The difficulty then is to find a way to assure response unlinkability, that is, to make responses indistinguishable to the proxy except for their positive or negative nature.

6.2 New primitives for response unlinkability

Our solution to achieve response unlinkability uses two existing primitives, namely Garbled Bloom Filters (GBFs) and Additively-Homomorphic Encryption (AHE) based Oblivious Transfer (OT), that we adapt to our needs.

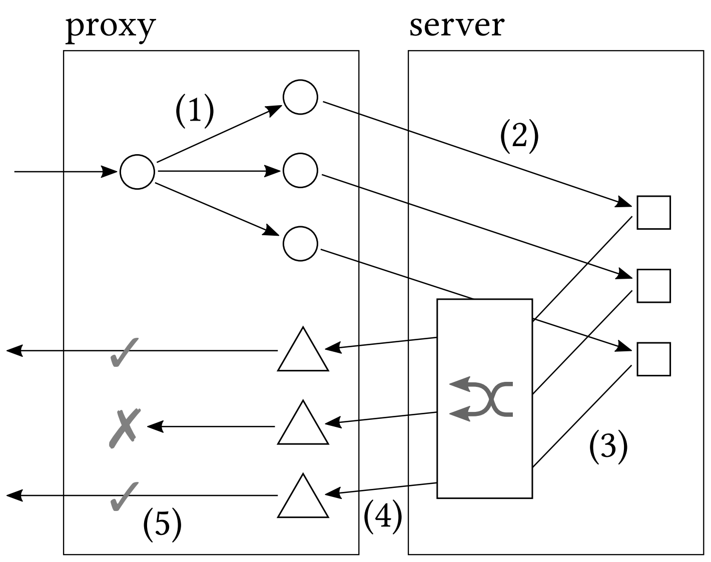


Figure 6.1: An illustration of the notion of response unlinkability

After the transformed trapdoors (1) are sent to the server to be applied on the prepared records (2), The corresponding responses are sent in random order (3 and 4, where the block between 3 and 4 represents random re-ordering). Thanks to response unlinkability, the proxy is unable to tell which response corresponds to which query or record. It is only able to filter out the negative responses and forward the positive ones (5).

GBFs are a variant of BFs where the buckets corresponding to an element reveal nothing beyond the presence or absence of the element in the encoded set. OT allows a receiver to retrieve an item in a database hosted by a sender without the sender learning what item was retrieved, and without the receiver learning anything about the rest of the database.

We modify the GBF construction into what we call Zero-sum Garbled Bloom Filters (ZGBF) so that the retrieved buckets do not even reveal what value was looked up in the set, which is necessary for response unlinkability, and we show how to obtain an OT protocol where two responses containing the same value are indistinguishable.

Intuitively, our protocol consists in the server encoding the prepared records as ZGBFs and the proxy looking up these ZGBF using the OT protocol. As a result the only information the proxy can learn from a response are some ZGBF buckets, and the only information that can be derived from these buckets are whether the response is positive or negative. These building blocks enable a MUSE protocol that reconciles response filtering and strict result length leakage. In the next two subsections we present in detail these two building blocks.

6.2.1 Zero-Sum Garbled Bloom Filters

We present a new hashing structure named Zero-Sum Garbled Bloom Filter (ZGBF) that is a variant of Bloom Filter (BF) compatible with response unlinkability and is adapted from the existing construction of Garbled Bloom Filter (GBF).

Garbled Bloom Filters

Garbled Bloom Filters (GBFs) were introduced by Dong et al. in [DCW13]. GBF is a variant of BF (see Section 5.3.2) that has some security properties making it suitable for the design of Private Set Intersection (PSI) protocols. PSI was already covered in Section 5.2.1 of this manuscript, and we refer the reader to [PSZ14] for a review of most recent PSI schemes that includes the one of [DCW13].

Like a BF, a GBF is an array of length M associated with k random hash functions $h_1, \ldots, h_\eta : \{0,1\}^* \to [M]$. However the components of a GBF are not bits but bit strings of length λ. One inserts x in a GBF B by ensuring that $\bigoplus_{i\in[\eta]} B[h_i(x)] = x$, and checks the presence of x in B by testing whether this equation holds or not.

During insertion, each bucket is filled with random bit strings as in a XOR secret sharing scheme, except the buckets that were already set by the insertion of a previous element that are left unchanged. After all the elements of the set are inserted in the GBF, components that were never written during insertion are filled with random bit strings. Algorithm 6 gives a more formal description of how a GBF is built. As with "normal" Bloom Filters, we will denote by GBF_S a GBF encoding set S when there is no ambiguity as to the parameters used.

The security property of GBFs can be informally described as follows:

Theorem 6.2.1 (Security of GBF – informal)**.** Let S and C be two sets of bit strings; Given only $\{GBF_S[i]\ \forall i \in h_*(C)\}$ one cannot get any information about $S - C$.

The security of GBFs is formally defined in Theorem 4 in [DCW13] which we reformulate in an equivalent way in Theorem 6.2.2 of this chapter. This theorem requires the definition of the

Algorithm 6: An algorithm for building a GBF representing set S with parameters $M, (h_i)_{i=1\dots\eta}, \lambda$

```
**Algorithm:** GBF.Build
**Input:** $S, M, (h_i)_{i=1\dots\eta}, \lambda$
**Output:** $B$
Initialize $B$ as an empty array of length $M$ ;
for $x \in S$ do
    if $\exists j \in h_*(x)\ :\ (B[j]$ is empty$)$ then
        for $i \in h_*(x) - \{j\}$ do
            if $B[i]$ is empty then
                $B[i] \xleftarrow{\$} \{0,1\}^\lambda$ ;
        set $B[j] \leftarrow x \oplus \bigoplus_{i \in h_*(x)-\{j\}} B[i]$ ;
    else
        Abort. ;
Fill any remaining empty component with fresh random values ;
```

intersection between a GBF and a BF sharing the same parameters (see Section 4.2 of [DCW13]) as follows:

Definition 6.2.1 (Intersection between a GBF and a BF)**.** Let $M, (h_i)_{i=1\dots\eta}$ and λ be some GBF parameters. Let S and C be two sets, and let GBF_S and BF_C be built with parameters $M, (h_i)_{i=1\dots\eta}$ (and λ for the GBF). The intersection of GBF_S and BF_C, noted $GBF_S \cap BF_C$, is defined as:

$$(GBF_S \cap BF_C)[i] \leftarrow \begin{cases} GBF_S[i] & \text{if } BF_C[i] = 1 \\ \text{a random value} & \text{otherwise} \end{cases}$$

Dong et al show that $GBF_S \cap BF_C$ is a correct GBF encoding $S \cap C$. We also define the notion of "extraction" of a GBF with a BF, which is equivalent to the notion of intersection but will make security proofs simpler (we study the security of GBFs in details in Chapter 7). With extraction, "non-selected" components are simply dropped, or equivalently set to a special "empty" value, instead of being replaced by a random value. It should be obvious that one obtains as much information from a uniform independent random value than from a fixed value.

Definition 6.2.2 (Extraction of a GBF with a BF)**.** Let $M, (h_i)_{i=1\dots\eta}$ and λ be some GBF parameters. Let S and C be two sets, and let GBF_S and BF_C be built with parameters $M, (h_i)_{i=1\dots\eta}$ (and λ for the GBF). The extraction of GBF_S using BF_C, noted $\mathsf{Extract}(BF_C, GBF_S)$, is defined as:

$$\mathsf{Extract}(BF_C, GBF_S)[i] \leftarrow \begin{cases} GBF_S[i] & \text{if } BF_C[i] = 1 \\ \text{empty} & \text{otherwise} \end{cases}$$

We now remind Theorem 4 of [DCW13] in a slightly reformulated but equivalent form:

Theorem 6.2.2 (Security of GBF (Theorem 4 of [DCW13]))**.** Let λ and $N \in \mathbb{N}$ and let $\eta = \lambda$ and $M = N\eta / \ln(2)$; let $(h_i)_{i \in [\eta]}$ be a sequence of random oracles $\{0,1\}^* \to [M]$. Given sets S and C of size at most N, we have

$$(S, C, GBF_S \cap BF_C) \overset{c}{\equiv} (S, C, GBF_{S \cap C} \cap BF_C)$$

Where S and C have at most N elements. Equivalently with our "extraction" notation:

$$(S, C, \mathsf{Extract}(BF_C, GBF_S)) \overset{c}{\equiv} (S, C, \mathsf{Extract}(BF_C, GBF_{S \cap C}))$$

This security property make GBF a good candidate as a building block for our RMÖ18 protocol because we want the receiver (the proxy in RMÖ18) to get no information from a lookup beyond whether the result is positive —the element was found— or negative —the element was not found—. But there is a problem: looking up a GBF for the presence of an element x requires, after retrieving the GBF items $\{GBF_S[i] \; \forall i \in h_*(x)\}$, the comparison of their sum to the value x. This comparison is impossible in our protocol because response unlinkability requires that the proxy, upon reception of the response containing the GBF items, is unable to tell which transformed trapdoor this response corresponds to, that is, it does not know what the value of x is.

Zero-Sum Garbled Bloom Filters

Algorithm 7: An algorithm for building a ZGBF representing set S with parameters $M, (h_i)_{i=1\ldots\eta}, \lambda$

```
Algorithm: ZGBF.Build
Input: $S, M, (h_i)_{i=1\ldots k}, \lambda$
Output: $B$
Initialize $B$ as an empty array of length $M$ ;
for $x \in S$ do
    if $\exists j \in h_*(x) \; : \; (B[j]$ is empty$)$ then
        for $i \in h_*(x) - \{j\}$ do
            if $B[i]$ is empty then
                $B[i] \xleftarrow{\$} \{0,1\}^\lambda$ ;
        set $B[j] \leftarrow \bigoplus_{i \in h_*(x) - \{j\}} B[i]$ ;
    else
        Abort. ;
Fill any remaining empty component with fresh random values ;
```

Fortunately, we notice that GBFs can be modified in the following way without losing their security property: insertion would consist in having the corresponding bucketssum up to zero instead of to x, and lookup would consist in comparing the sum of the items with zero. We call the resulting construction **Zero-Sum Garbled Bloom Filters** (ZGBFs) and we formally define algorithms ZGBF.Build and ZGBF.Check in this section.

```
Algorithm 8: An algorithm for deciding whether some ZGBF buckets correspond to an
element that is present or absent in a ZGBF
  Algorithm: ZGBF.Check
  Input: $(b_i)_{i=1\dots\eta}$
  Output: True or False
  return True if $\bigoplus_i b_i = 0$ else False
```

The preservation of correctness is obvious. More interestingly, the original security property is also preserved. One way of showing that security is preserved would be to notice that the original proof by Dong et al. [DCW13] does not require that the value the buckets sum to is the value that is being encoded, so it would apply to ZGBFs as well. In Chapter 7 we show that this proof has flaws, but fortunately we are able to give an new proof in Section 7.4.2 that shows the security of both GBFs and ZGBFs.

6.2.2 Oblivious Transfer with Response Unlinkability

In this section, we present an existing OT protocol compatible with response unlinkability. Let B be an array of size M hosted at a server. An OT protocol allows a client to retrieve $B[i]$ in a privacy preserving manner with the following algorithms:

- $\mathsf{OT.KeyGen}(1^\lambda) \to K_{OT}$
- $\mathsf{OT.Query}(K_{OT}, i, M) \to Q$
- $\mathsf{OT.Apply}(Q, B) \to P$
- $\mathsf{OT.Open}(K_{OT}, P) \to x$

The protocol is correct if $x = B[i]$, and secure if Q reveals no information and P reveals nothing beyond $B[i]$.

Lipmaa describes in Section 4 of [Lip05] how to modify Stern's Private Information Retrieval (PIR) protocol [Ste98] into an OT protocol secure in the honest-but-curious model. The general idea of Stern's PIR protocol is to use an Additively Homomorphic cipher AH, that is, an IND-CPA-secure encryption scheme with algorithms AH.KeyGen, AH.Enc and AH.Dec and an additional algorithm AH.Add that takes two ciphertexts c_1 and c_2 as input having respective plaintext m_1 and m_2, and outputs a valid ciphertext that decrypts to $m_1 + m_2$. We will note $\mathsf{AH.Add}(c_1, c_2)$ as $c_1 + c_2$.

Stern's PIR protocol consists then in building Q as:

$$Q[j] \leftarrow \begin{cases} \mathsf{AH.Enc}(1) \text{ if } i = j \\ \mathsf{AH.Enc}(0) \text{ otherwise} \end{cases}$$

OT.Apply consists then in performing $P \leftarrow B \times Q$ where multiplication of a ciphertext by a scalar is seen as repeated additions (see Section 2.1 of [Mel+16]):

$$a \times \mathsf{AH.Enc}(b) = \sum_{i=0}^{a} \mathsf{AH.Enc}(b) = \mathsf{AH.Enc}(ab)$$

$$\mathsf{AH.Dec}(a \times \mathsf{AH.Enc}(b)) = ab$$

As a result, P decrypts to

$$1 \times B[i] + \sum_{j \neq i} 0 \times B[j] = B[i]$$

This protocol is not an OT protocol because it only provides privacy against the server and not against the client. Lipmaa shows that it can be transformed in an OT protocol secure against honest-but-curious parties simply by adding fresh encryptions of zero to the results of homomorphic computations. This requires that the AH cipher is a public-key scheme (typical ciphers for AH would be Paillier encryption or ciphers based on the Learning With Errors (LWE) problem (see [LPR13])).

Adding encryptions of zero is actually a form of *ciphertext sanitization* (See [DS16]), a technique that makes a ciphertext indistinguishable from any other ciphertext with the same plaintext. (This is linked to the notion of **circuit privacy** in homomorphic encryption, see [Bou+16]). We then claim that P is computationally indistinguishable from a fresh encryption of $B[i]$ in the OT protocol we use. We define the algorithm OT.Forge that performs such encryption and whose output is thus indistinguishable from a real OT response decrypting to the same value.

This OT protocol, unlike others, still preserves response unlinkability because a response is simply a public-key ciphertext so the client (the proxy in our case) is able to open it without having to remember which query it corresponds to.

OT recursiveness and techniques for multi-query OT

The OT protocol we just described is not very efficient because the size of a query is linear with the size of the database B. Also, an OT query can only retrieve a single component of B, while in the MUSE protocol we are building the proxy must retrieve η buckets of the same ZGBF.

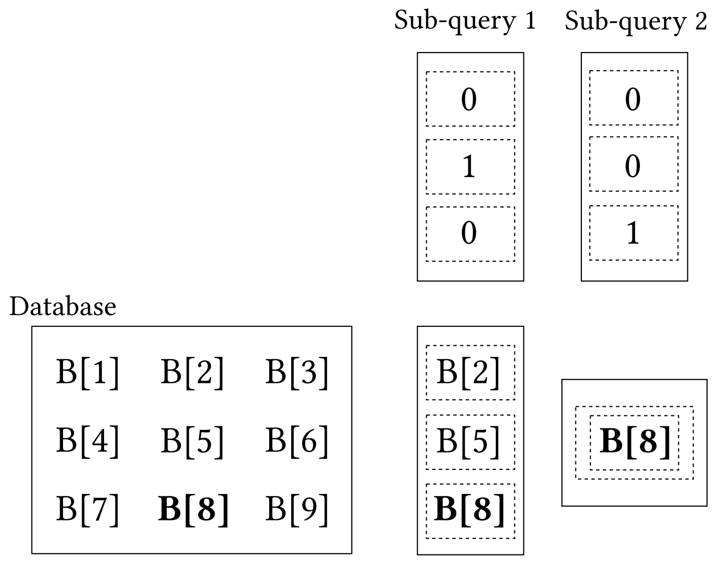


Figure 6.2: An illustration of PIR/OT recursiveness
Dashed boxes represent encryption under the AH cipher.

Recursiveness, illustrated in Figure 6.2 is a well-known technique to make Stern's PIR protocol more efficient, and can be used in our OT protocol. Recursiveness is a technique that can reduce bandwidth costs by sending two (or more) small OT queries, that we name *sub-queries*, instead of a single large one. B is split into n blocks of M/n elements, the first sub-query is applied on each block to retrieve its $(i \bmod \frac{M}{n})$-th element, and the second sub-query is applied on the result of the first step to retrieve the result of the $\lceil i/\frac{M}{n} \rceil$-th block. Recursiveness significantly reduces the bandwidth consumption of the OT protocol we use at the price of a slight increase in computation at the server. See also Section 4 of [KO97] and section II.B of [Mel+14].

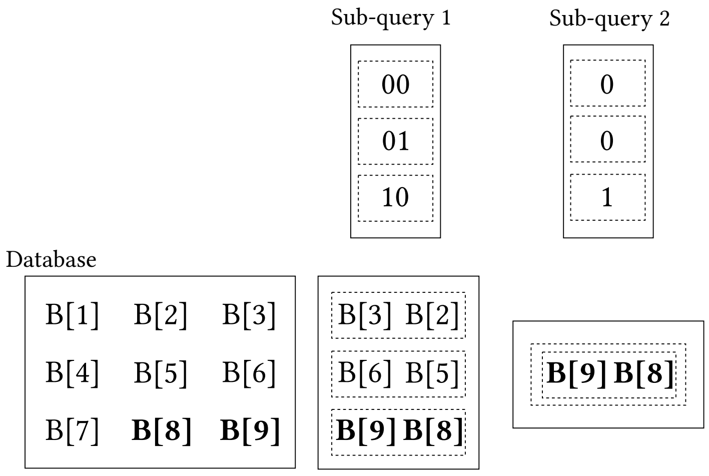


Figure 6.3: An illustration of OT encoding

Values are written in base 2^{λ}, that is, 10 represents the number 2^{λ}. Note that the two values retrieved are in the same "row" of the database.

Multi-query OT is another property that is desirable for our OT protocol, that consists in the ability to retrieve more than one component of the remote database (the ZGBF buckets in our case) with a single query. Retrieving components $B[i_1], B[i_2], \ldots$ with a single query can be done easily by generating $Q[i_k]$ as $\mathsf{AH.Enc}(2^{(k-1)\lambda})$ so that the response P will decrypt to $\sum_k 2^{(k-1)\lambda} B[i_k]$. We call this technique **OT encoding**, illustrated in Figure 6.3. The problem with OT encoding is that it does not suit OT recursiveness. There is a simple solution to this problem: If all the components to retrieve are in the same "row" of B (as we defined it in OT recursiveness), then the two techniques can be used together without conflict: the first sub-query uses OT encoding to retrieve all the targeted components and the second sub-query selects the (single) proper row among the results of the application of the first query. As a result we should be able to retrieve all the targeted components of B *in a single OT query* while still benefiting from the efficiency of recursive OT.

Modifying the ZGBF construction to guarantee that all the components corresponding to an element are in the same row can be done by building B as $\sqrt{M}$ successive ZGBF, each being a "row" of B, and deciding which row an element x should be inserted or looked up in by hashing x in the range $[\sqrt{M}]$. We call the resulting data structure *In-Line ZGBF*.

Towards the end of this thesis, a paper by Angel et al. was published at IEEE S&P 2018 [Ang+]

that presents new techniques to improve the efficiency of Stern's PIR, building on top of the work of Aguilar-Melchor et al. [Mel+16]. The techniques in [Ang+] allow both significant savings in terms of bandwidth consumption but also provide efficient multi-query PIR. They are probably applicable to our MUSE protocol and should significantly increase its performance. The incorporation of these new techniques seems to be a promising direction for future enhancements of our protocol.

6.3 The RMÖ18 MUSE Protocol

We now give a formal description of the RMÖ18 MUSE protocol. Since the protocol is more complex than the ones of the previous chapter, each algorithm deserves a dedicated description. Like RMÖ15, the RMÖ18 protocol requires an IND-CPA secure symmetric cipher called "transmission cipher", defined by algorithms CPA.KeyGen, CPA.Encrypt and CPA.Decrypt. This cipher will be used by the server to encrypt record ids before attaching them to the responses so that the proxy cannot read them but the querying reader can. The global parameters for RMÖ18 are the security parameter λ, the DDH-hard group $\mathbb{G}$ of order ζ with the hash function H and the ZGBF parameters η, M and $(h_i)_{i=1\dots\eta}$

- The writer owning record $W_d \in \{0,1\}^*$ picks record key γ_d at random from $\mathbb{Z}_\zeta^*$. She encrypts this record into $\overline{W}_d$ by computing:

$$\overline{W}_d \leftarrow \mathsf{Writer.Encrypt}(W_d, \gamma_d)$$

  (algorithm listings are given further below). The writer sends $\overline{W}_d$ to the server and γ_d to the proxy.

- The writer owning record W_d can authorize a reader r to search W_d simply by notifying the proxy and the server.

- For each time period l, reader r picks a random reader key $\rho_{r,l} \in \mathbb{Z}_\zeta^*$ and sends it to the server. Reader r also generates a transmission key k_r by running CPA.KeyGen and sends it to the server.

- The proxy must create a key for the OT protocol:

$$K_{\mathsf{OT}} \leftarrow \mathsf{OT.KeyGen}(1^\lambda)$$

- When the server receives reader key $\rho_{r,l}$, it computes the prepared encrypted record $\overline{\overline{W}}_{d,r,l}$ for each $d \in Auth(r)$ by running $\mathsf{Server.Prepare}(\overline{W}_d, \rho_{r,l})$

$$\overline{\overline{W}}_{d,r,l} \leftarrow \mathsf{Server.Prepare}(\overline{W}_d, \rho_{r,l})$$

  Where algorithm Server.Prepare is defined further below. It does not send $\overline{\overline{W}}_{d,r,l}$ but stores it instead. Prepared records of expired time periods can be discarded.

- $q_{r,l,s}$ denotes the s-th query of reader r during the l-th time period. For each such query, reader r creates the corresponding trapdoor $t_{r,l,s}$:

  $$t_{r,l,s} \leftarrow \mathsf{Reader.Trapdoor}(q_{r,l,s}, \rho_{r,l})$$

  $t_{r,l,s}$ is sent to the proxy.

- When the proxy receives trapdoor $t_{r,l,s}$, the following sub-protocol is executed:
  - the proxy computes the proxy query $t'_{r,l,s}$ as follows and sends it to the server:

    $$t'_{r,l,s} \leftarrow \mathsf{Proxy.Transform}(t_{r,l,s}, \boldsymbol{\gamma}, Auth, K_{\mathsf{OT}})$$

  - the server computes the server response $p'_{r,l,s}$ as follows and sends it to the proxy:

    $$p'_{r,l,s} \leftarrow \mathsf{Server.Process}(t'_{r,l,s}, \boldsymbol{k})$$

  - the proxy computes the filtered response $p_{r,l,s}$ as follows and sends it to reader r:

    $$p_{r,l,s} \leftarrow \mathsf{Proxy.Filter}(p'_{r,l,s}, K_{\mathsf{OT}})$$

- Finally when the filtered response $p_{r,l,s}$ is received by reader r, it is decrypted as follows:

  $$a_{r,l,s} \leftarrow \{\mathsf{CPA.Decrypt}(x, k_r)\ \forall x \in p_{r,l,s}\}$$

```
Algorithm: Writer.Encrypt
Input: $(W_d, \gamma_d)$
Output: $\overline{W}_d$
$\overline{W}_d \leftarrow \{H(w)^{\gamma_d}\ \forall w \in W_d\}$ ;
```

```
Algorithm: Server.Prepare
Input: $(\overline{W}_d, \rho_{r,l})$
Output: $B_{d,r,l}$
$\overline{\overline{W}}_{d,r,l} \leftarrow \{\overline{w}^{\rho_{r,l}}\ \forall \overline{w} \in \overline{W}_d\}$;
```

```
Algorithm: Reader.Trapdoor
Input: $(q_{r,l,s}, \rho_{r,l})$
Output: $t_{r,l,s}$
$t_{r,l,s} \leftarrow H(q_{r,l,s})^{\rho_{r,l}}$
```

```
**Algorithm:** Proxy.Transform
**Input:** $(t_{r,l,s}, \gamma, Auth, K_{\mathsf{OT}})$
**Output:** $t'_{r,l,s}$
Initialize $t'_{r,l,s}$ as an empty set ;
for $d \in Auth(r)$ do
    $c_{r,l,s,d} \leftarrow (t_{r,s,l})^{\gamma_d}$ ;
    $Q \leftarrow \mathsf{OT.Query}(K_{\mathsf{OT}}, h_*(c_{r,l,s,d}))$ ;
    Add $(d, Q)$ to $t'_{r,l,s}$ ;
```

```
**Algorithm:** Server.Process
**Input:** $(t'_{r,l,s}, \boldsymbol{k})$
**Output:** $p'_{r,l,s}$
Initialize $p'_{r,l,s}$ as an empty set ;
for $(d, Q) \in t'_{r,l,s}$ in random order do
    $B_{d,r,l,s} \leftarrow \mathsf{ZGBF.Build}(\overline{\overline{W}}_{d,r,l}, h_*)$;
    $P \leftarrow \mathsf{OT.Apply}(Q, B_{d,r,l,s})$ ;
    $\overline{d} \leftarrow \mathsf{CPA.Encrypt}(d, \boldsymbol{k}[r])$ ;
    Append $(\overline{d}, P)$ to $p'_{r,l,s}$ ;
```

```
**Algorithm:** Proxy.Filter
**Input:** $(p'_{r,l,s}, K_{\mathsf{OT}})$
**Output:** $p_{r,l,s}$
$p_{r,l,s} \leftarrow \{\overline{d}\ \forall(\overline{d}, P) \in p'_{r,l,s} : \mathsf{ZGBF.Check}(\mathsf{OT.Open}(K_{\mathsf{OT}}, P)) = \text{True}\}$
```

ZGBFs must be re-built for each query As in protocols DH-AP-MUSE and RMÖ15, record preparation in RMÖ18 is only performed when a reader renews its key. However, note that the encoding of a prepared record $\overline{\overline{W}}_{d,r,l}$ into a ZGBF $B_{d,r,l,s}$ must still be repeated at each new query (in algorithm Server.Process). This is required for response unlinkability: indeed, different transformed trapdoors from the same user targeting the same record can have some positions in common, that is, the following can happen:

$$h_*(c_{r,l,s_1,d}) \cap h_*(c_{r,l,s_2,d}) \neq \emptyset$$

With ZGBFs being re-used for different queries, such event would result in the two corresponding responses containing some identical bucket values, which would let the proxy learn that these two responses correspond to the same record, breaking response unlinkability.

Another rationale for the renewal of the ZGBF is step 2 of the proof of security against the Proxy (see algorithm Server.Process$_2$), where simulated ZGBF buckets are re-generated at each query.

Fortunately, building ZGBFs is a much less expensive operation than modular exponentiation in $\mathbb{G}$, and can be done ahead of time in order not to impact the execution time of search operations.

6.3.1 Privacy Against the Proxy

We show that RMÖ18 has a leakage no greater than the result length, the benign leakage and the revealed content against a honest-but-curious proxy that colludes with any number of users but not with the server.

We use a proof method that is similar to the one used in the previous chapter, by showing how one can simulate the view of the adversary using the leakage as sole input. However because the protocol and the proof are more complex than the ones in the previous chapter, we further complete the discussion with formal definitions.

For the sake of readability, our proof makes use of several incremental steps. In each step i we define a simulator $\mathcal{S}_i$ that takes $\mathcal{H}$ as input. $\mathcal{S}_0$ outputs $\mathcal{V}(\mathcal{H})$ and for $i = 1 \dots 5$ we show that $\mathcal{S}_i(\mathcal{H}) \overset{c}{\equiv} \mathcal{S}_{i-1}(\mathcal{H})$. Thus by transitivity we have $\mathcal{S}_5(\mathcal{H}) \overset{c}{\equiv} \mathcal{V}(\mathcal{H})$. Finally we show that one can build an algorithm that has the same exact output distribution as S_5 while having $\mathcal{L}(\mathcal{H})$ as input instead of $\mathcal{H}$, and this ends the proof. The common structure for all simulators is given in Algorithm $\mathcal{S}_i$. Since $\mathcal{S}_0$ must output the real, non-simulated view, it simply calls the algorithms from the real protocol defined in Section 6.3, that is, Reader.Trapdoor$_0$ = Reader.Trapdoor etc. For each subsequent simulator we only list the algorithms that differ from the ones in the previous simulator, and we highlight differences in gray. Note that a variable that is computed in some algorithm in the simulator is accessible inside subsequent algorithms called by the simulator. For instance in $\mathcal{S}_1$, variables $c_{r,l,s,d}$ are used in Server.Prepare$_1$ while they are computed in Proxy.Transform$_1$.

```
Algorithm: $\mathcal{S}_i$
Input: the entire history
Create all keys ;
for $r \in R,\ d \in Auth(r)$ do
    $\overline{W}_d \leftarrow \mathsf{Writer.Encrypt}_i(W_d, \gamma_d)$;
    for each $l$ do
        $\overline{\overline{W}}_{d,r,l} \leftarrow \mathsf{Server.Prepare}_i(\overline{W}_d, \rho_{r,l})$;
for each $r, l, s$ do
    $t_{r,l,s} \leftarrow \mathsf{Reader.Trapdoor}_i(q_{r,l,s}, \rho_{r,l})$ ;
    $t'_{r,l,s} \leftarrow \mathsf{Proxy.Transform}_i(t_{r,l,s}, \gamma_d, Auth, K_{\mathsf{OT}})$ ;
    $p'_{r,l,s} \leftarrow \mathsf{Server.Process}_i(t'_{r,l,s}, \overline{\overline{\boldsymbol{W}}}, \boldsymbol{k})$ ;
Output: all $t_{r,l,s}$ values, all $t'_{r,l,s}$ values, all $p'_{r,l,s}$ values, all record keys $\gamma_d$ and the keys of
        corrupted users
```

In $\mathcal{S}_1$, algorithm Server.Process$_1$ does not call OT.Apply, and instead creates $P_{r,s,w}$ by directly encrypting the components of $B_{d,r,l,s}$ that the OT query Q was supposed to retrieve, using the OT.Forge algorithm. The $P_{r,s,w}$ variables are the only variables of the simulator output to be affected by these changes, and each fabricated $P_{r,s,w}$ is indistinguishable from a real one thanks to ciphertext sanitization resulting from the OT protocol; as a result, from the hybrid argument, the output of $\mathcal{S}_1$ is indistinguishable from the one of $\mathcal{S}_0$.

```
Algorithm: $\mathsf{Server.Process}_1$
Input: $(t'_{r,l,s}, \boldsymbol{k})$
Output: $p'_{r,l,s}$
Initialize $p'_{r,l,s}$ as an empty set ;
for $(d, Q) \in t'_{r,l,s}$ in random order do
    $B_{d,r,l,s} \leftarrow \mathsf{ZGBF.Build}(\overline{\overline{W}}_{d,r,l}, h_*)$;
    $\mathsf{buckets}_{r,l,s,d} \leftarrow \{B_{d,r,l,s}[j]\ \forall j \in h_*(c_{r,l,s,d})\}$;
    $P_{r,l,s,d} \leftarrow \mathsf{OT.Forge}(\mathsf{buckets}_{r,l,s,d})$;
    $\overline{d} \leftarrow \mathsf{CPA.Encrypt}(d, \boldsymbol{k}[r])$ ;
    Append $(\overline{d}, P_{r,l,s,d})$ to $p'_{r,l,s}$ ;
```

In $\mathcal{S}_2$, Server.Process$_2$ does not use the content of non-revealed indexes; Instead the query result $a_{r,s}$ is used, which is computed by $\mathcal{S}_2$ beforehand from the history: If $d \in a_{r,s}$, $P_{r,l,s,d}$ is created as a forged OT response containing shares of zero, and is thus a positive response; otherwise it is created as a forged OT response containing random values, and is thus a negative response. This procedure is equivalent to building $B_{d,r,l,s}$ as a regular ZGBF from the set $\overline{\overline{W}}_{d,r,l} \cup \{c_{r,l,s,d}\}$, while the corresponding procedure of Server.Process$_1$ is equivalent to a intersection between an ZGBF built from $\overline{\overline{W}}_{d,r,l}$ and a BF built from $\{c_{r,l,s,d}\}$. As a result, the security property of ZGBF implies that a $P_{r,l,s,d}$ variable built by $\mathcal{S}_2$ is indistinguishable from the same $P_{r,l,s,d}$ variable built by $\mathcal{S}_1$. Again, one can then show using the hybrid argument that the output of $\mathcal{S}_2$ is indistinguishable from the output of $\mathcal{S}_1$.

```
Algorithm: Server.Process$_2$
Input: $(t'_{r,l,s}, \boldsymbol{k})$
Output: $p'_{r,l,s}$
Initialize $p'_{r,l,s}$ as an empty set ;
for $(d, Q) \in t'_{r,l,s}$ in random order do
    buckets$_{r,l,s,d} \leftarrow \eta$ random values;
    if $d \in a_{r,l,s}$ then
        buckets$_{r,l,s,d}[1] \leftarrow \bigoplus_{j \neq 1}$ buckets$_{r,l,s,d}[j]$;
    $P_{r,l,s,d} \leftarrow$ OT.Forge(buckets$_{r,l,s,d}$);
    $\overline{d} \leftarrow$ CPA.Encrypt$(d, \boldsymbol{k}[r])$ ;
    Append $(\overline{d}, P_{r,l,s,d})$ to $p'_{r,l,s}$ ;
```

In $\mathcal{S}_3$, Server.Process$_3$ sends random values instead of $\overline{d}$. The output of $\mathcal{S}_3$ is indistinguishable from the output of $\mathcal{S}_2$ thanks to the IND-CPA security of the transmission cipher and the hybrid argument.

```
Algorithm: Server.Process$_3$
Input: $(t'_{r,l,s}, \boldsymbol{k})$
Output: $p'_{r,l,s}$
Initialize $p'_{r,l,s}$ as an empty set ;
for $(d, Q) \in t'_{r,l,s}$ in random order do
    buckets$_{r,l,s,d} \leftarrow \eta$ random values;
    if $d \in a_{r,l,s}$ then
        buckets$_{r,l,s,d}[1] \leftarrow \bigoplus_{j \neq 1}$ buckets$_{r,l,s,d}[j]$;
    $P_{r,l,s,d} \leftarrow$ OT.Forge(buckets$_{r,l,s,d}$);
    Random $\overline{d}$ ;
    Append $(\overline{d}, P_{r,l,s,d})$ to $p'_{r,l,s}$ ;
```

$\mathcal{S}_4$ does not use the query results but only their result length. For each query, $\mathcal{S}_4$ just creates the proper number of positive and negative responses. All positive (respectively negative) responses were already created the same way as in $\mathcal{S}_3$, and they are sent in random order, so the output of $\mathcal{S}_4$ has the same exact distribution as the one of $\mathcal{S}_3$.

```
Algorithm: Server.Process$_4$
Input: $(t'_{r,l,s}, \boldsymbol{k})$
Output: $p'_{r,l,s}$
Initialize $p'_{r,l,s}$ as an empty set ;
for $i = 1 \ldots |Auth(r)|$ do
    buckets$_{r,l,s,d}$ $\leftarrow \eta$ random values;
    if $i \leq |a_{r,l,s}|$ then
        buckets$_{r,l,s,d}[1] \leftarrow \bigoplus_{j \neq 1}$ buckets$_{r,l,s,d}[j]$;
    $P_{r,l,s,d} \leftarrow$ OT.Forge(buckets$_{r,l,s,d}$);
    Random $\overline{d}$ ;
    Append $(\overline{d}, P_{r,l,s,d})$ to $p'_{r,l,s}$ ;
```

$\mathcal{S}_5$ does not use non-revealed queries. User trapdoors $t_{r,s}$ are just random values; as to $p'_{r,s}$ variables, they are not affected by the change: Indeed thanks to the changes introduced in the previous simulators, algorithm Server.Process$_5$ does not use any variable that depends on $t_{r,s}$. The output of $\mathcal{S}_5$ is indistinguishable from the output of $\mathcal{S}_4$ from a reduction similar as in the security proof of DH-AP-MUSE.

```
Algorithm: Reader.Trapdoor$_5$
Input: $(q_{r,l,s}, \rho_{r,l})$
Output: $t_{r,l,s}$
$t_{r,l,s} \xleftarrow{\$} \mathbb{G}$;
```

Finally the detailed description of $\mathcal{S}_{\text{RMÖ18}}$ is given. $\mathcal{S}_{\text{RMÖ18}}$ is the same as $\mathcal{S}_5$ but only has the leakage as input. The output of $\mathcal{S}_{\text{RMÖ18}}$ is then identical to the one of $\mathcal{S}_5$, and *a fortriori* they are indistinguishable.

Wrapping up, the output of $\mathcal{S}_{\text{RMÖ18}}$ is indistinguishable from the real view (the output of $\mathcal{S}_0$) and the input of $\mathcal{S}_{\text{RMÖ18}}$ is limited to the result lenght, the revealed content and the benign leakage. This ends the proof that RMÖ18 has the claimed leakage profile against the proxy according to our definition of security (Definition 3.2.3).

□

6.3.2 Privacy Against the Server

As in the RMÖ15 protocol described in Section 5.3, the view of the server in RMÖ18 is very similar to the one in the DH-AP-MUSE protocol. The only differences are the fact that in RMÖ18, the server receives transmission keys and OT queries.

Because transmission keys are independent of the rest of the server's view and because OT queries reveal no information, it is easy to see that the leakage to the server in RMÖ18 is no greater than the one in DH-AP-MUSE, that is, no greater than the benign leakage and the revealed content.

```
Algorithm: $\mathcal{S}_{\text{RMÖ18}}$
Input: the result length of queries, the benign leakage and the revealed content
Create all keys ;
for $r \in R',\ d \in Auth(r)$ do
    $\overline{W}_d \leftarrow \mathsf{Writer.Encrypt}(W_d, \gamma_d)$;
    for each $l$ do
        $\overline{\overline{W}}_{d,r,l} \leftarrow \mathsf{Server.Prepare}(\overline{W}_d, \rho_{r,l})$;
for each $r, l, s$ do
    if $r \in R'$ then
        $t_{r,l,s} \leftarrow \mathsf{Reader.Trapdoor}(q_{r,l,s}, \rho_{r,l})$ ;
    else
        $t_{r,l,s} \xleftarrow{\$} \mathbb{G}$ ;
    $t'_{r,l,s} \leftarrow \mathsf{Proxy.Transform}(t_{r,l,s}, \gamma_d, Auth, K_{\mathsf{OT}})$ ;
    if $r \in R'$ then
        Initialize $p'_{r,l,s}$ as an empty set ;
        for $i = 1 \ldots |Auth(r)|$ do
            $\mathsf{buckets}_{r,l,s,d} \leftarrow \eta$ random values;
            if $i \leq |a_{r,l,s}|$ then
                $\mathsf{buckets}_{r,l,s,d}[1] \leftarrow \bigoplus_{j \neq 1} \mathsf{buckets}_{r,l,s,d}[j]$;
            $P_{r,l,s,d} \leftarrow \mathsf{OT.Forge}(\mathsf{buckets}_{r,l,s,d})$;
            Random $\overline{d}$ ;
            Append $(\overline{d}, P_{r,l,s,d})$ to $p'_{r,l,s}$ ;
    else
        $p'_{r,l,s} \leftarrow \mathsf{Server.Process}(t'_{r,l,s}, \overline{\overline{\boldsymbol{W}}}, \boldsymbol{k})$ ;
Output: all $t_{r,l,s}$ values, all $t'_{r,l,s}$ values, all $p'_{r,l,s}$ values, all record keys $\gamma_d$ and the keys of
        corrupted users
```

6.3.3 Efficiency Analysis

RMÖ18 is clearly efficient for the users, which was the first objective regarding scalability. Thanks to this efficiency, RMÖ18 is much more suitable for large databases than protocols like RMÖ15 or than approaches using several parallel SSE systems.

The workload for the servers however is quite substantial, and the protocol is quite complex to implement, especially the OT component with our modifications. It would be very interesting as future work to use the work of [Mel+16] that presents an efficient implementation of Stern's PIR to implement the OT protocol in order to obtain practical measurements.

We thus give some figures on the theoretical complexity of RMÖ18 on the server side. As with the performance analysis of DH-AP-MUSE (Section 5.2.3), we consider a system with A writers owning B records each, each record containing N keywords, and C readers each having access to D records.

Storage and communication cost during upload

- The server must store $A \times B \times N$ elements of $\mathbb{G}$ for the encrypted records and one symmetric key for each reader. It must also store the prepared records, accounting for $C \times D \times N$ elements of $\mathbb{G}$ (recall Figure 5.1).
- The proxy only has to store the record keys.

Computation and Communication during Search

- For each query being sent, the proxy must perform D exponentiations in $\mathbb{G}$, $\eta \times D$ hashing and creates D OT queries (one query per record thanks to OT encoding). The on-line execution time of query creation can be quite fast if AH ciphertexts are created in advanced and just "put together" to create an OT query. Data sent from the proxy to the server consists in $D \times 2\sqrt{\mathcal{O}(N)}$ ciphertexts from AH; Where $\mathcal{O}(N)$ comes from the size of a ZGBF. The exact size of a ZGBF will depend on how ZGBF parameters are optimized, which is left as future work.
- the cost for the server to apply the queries on the prepared indexes is $D\left(\mathcal{O}(N) \times \mathsf{FMA}\right) + F\sqrt{\mathcal{O}(N)} \times \mathsf{FMA}$ where FMA is the cost of the "Fused Multiply and Add" operation described in [Mel+16] and F is the expansion factor of AH. The amount of data sent from the server to the proxy is $DF^2\eta\lambda$ bits.
- Finally the cost of filtering should be negligible with regard to the rest of the search protocol.

Because record preparation is done ahead of time, the main performance bottleneck during search should be the OT protocol, whose performance depends a lot on the underlying AHE implementation and various optimizations as found in [Mel+16].

6.4 Further Remarks and Implementation

Enhancements over the [RMÖ18b] article The RMÖ18 protocol was described in a paper presented at the sixth international workshop on Security in Cloud Computing (SCC 2018) [RMÖ18b]. As with RMÖ15 (see Section 5.3.7), the version described in this manuscript differs from the version that was published. The version of [RMÖ18b], like the published version of RMÖ15 [RMÖ15], makes use of bilinear pairings and requires that the server performs record preparation for every new query. Again, we realized during the design of DH-AP-MUSE that the use of bilinear pairings was in fact unnecessary and that record preparation could be performed at a regular time interval instead of at each query.

Implementation We built a proof-of-concept implementation for the TREDISEC European research project [TREDISEC], written in Python using the Charm crypto framework [Charm-Crypto] as with RMÖ15 (see Section 5.3.7). The TREDISEC project produced a framework that facilitates the selection, the deployment and the management of cloud security primitives. Our implementation of RMÖ18 was one of these primitives, and was used to demonstrate the usability of the framework.

The performance of this implementation however does not represent well the performance that the RMÖ18 protocol could achieve. This is because of the difficulty to implement the OT protocol RMÖ18 depends on in a efficient way. Efficiency of PIR and OT protocols is a research topic that is currently very active, and the recent progress made by Aguilar-Melchor et al. [Mel+16] and Angel et al. [Ang+] regarding the efficiency of the PIR protocol [Ste98] we use as a base in RMÖ18 (see Section 6.2.2) gives us confidence in that RMÖ18 can be made capable of processing large databases in reasonable time. Nevertheless, implementing the solutions presented in [Mel+16] and [Ang+] is a highly non-trivial task. Although both Angel et al. and Aguilar-Melchor et al. implemented their solutions, the implementation [SealPIR-GitHub] of [Ang+] was still unavailable as to May 2018, and applying our modifications (see Section 6.2.2) to the implementation [XPIR-GitHub] of [Mel+16] proved to be very difficult. Implementing RMÖ18 with the state-of-the-art techniques in PIR protocols is then an important objective for future work.

6.5 Conclusion

The RMÖ18 protocol shows a way towards solutions to the MUSE problem that combine both a small leakage in presence of user-server collusions and some degree of scalability. Despite its involved nature and a quite substantial server-side complexity, RMÖ18 is the first MUSE protocol that does not reveal the access pattern while at the same time having a very low user-side complexity. The notion of response unlinkability that we developped, as well as the new primitives we had to design for the construction of RMÖ18, should pave the way for future secure and scalable MUSE protocols. Finally, RMÖ18 demonstrates that we are not condemned to a choice between leaking the access pattern and having a high user-side complexity, closing the gap that was left open at the end of the previous chapter.

Figure 6.4 completes our map of MUSE protocols on the privacy/efficiency graph, with the addition of the RMÖ18 protocol. It clearly shows the gap that RMÖ18 closes, as well as the fact

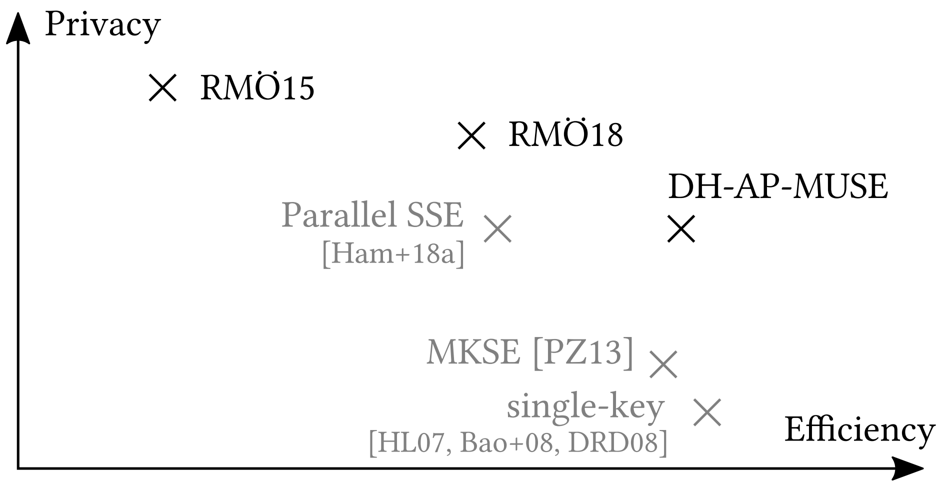


Figure 6.4: Location of protocol RMÖ18 and all other MUSE protocols regarding the privacy-versus-efficiency dilemma

that our three protocols RMÖ15, RMÖ18 and DH-AP-MUSE cover a large range of privacy and efficiency combinations.

7 On the security of Garbled Bloom Filters

In the design of our RMÖ18 protocol, we built a novel hash structure named Zero-sum Garbled Bloom Filters (ZGBFs) that is a modification of an existing structure named Garbled Bloom Filters (GBFs) first presented by Dong et al. in [DCW13].

In this chapter, we show a flaw in the original proof of security for GBF given by Dong et al. in [DCW13], and we show that it applies to more recent GBF variants. However this does not mean that GBF are insecure. Instead, we give a new proof that GBF satisfy their claimed security property. This proof also applies to our ZGBF variant.

7.1 Original proof of security by Dong et al. [DCW13]

The proof Dong et al. give for Theorem 6.2.2 is reproduced below. The only modifications we made consist in adapting the notation to match the one we define in Section 6.2.1. Namely, what is written $GBF_{C\cap S}$, $GBF'_{C\cap S}$ and $GBF''_{C\cap S}$ in the original text is written $GBF_S \cap BF_C$, $GBF_{S\cap C}$ and $(GBF_S \cap BF_C) \cap BF_{S\cap C}$ respectively in ours. We give a quick overview of their proof: In their "case 1", they show that the probability that some element of $S - C$ has all its positions in $h_*(C)$[1] is negligible; then in "case 2" they argue that if no element of $S - C$ has all its elements in $h_*(C)$, the distribution of $GBF_S \cap BF_C$ is thus identical to the one of $GBF_{S\cap C}$. They invoke *"the security of the XOR-based secret sharing scheme"* to argue that an element of $S - C$ of which one of the buckets was re-randomized during intersection cannot leave any trace in the resulting GBF (this is the argument we will go against in Section 7.2).

> **(Proof of Theorem 6.2.2 as it appears in [DCW13])**
> Given $GBF_S \cap BF_C$, we modify it to get $(GBF_S \cap BF_C) \cap BF_{S\cap C}$. We scan $GBF_S \cap BF_C$ from the beginning to the end and for each location i, we modify $(GBF_S \cap BF_C)[i]$ using the following procedure:
>
> - If $(GBF_S \cap BF_C)[i]$ is a share of an element in $C \cap S$, then do nothing.
> - Else if $(GBF_S \cap BF_C)[i]$ is a random string, do nothing.
> - Else if $(GBF_S \cap BF_C)[i]$ is a share of an element in $S - C \cap S$, replace it with a uniformly random λ-bit string.
>
> The result is $(GBF_S \cap BF_C) \cap BF_{S\cap C}$. Every $(GBF_S \cap BF_C)[i]$ must fall into one of these three cases, so there is no unhandled case.
>
> Now we argue that the distribution of $(GBF_S \cap BF_C) \cap BF_{S\cap C}$ is identical to $GBF_{S\cap C}$. To see that, let's compare each location in $(GBF_S \cap BF_C) \cap BF_{S\cap C}$ and $GBF_{S\cap C}$.

[1]the h_* notation was defined in Section 5.3.2

From Algorithm 1 and the above procedure, we can see that $(GBF_S \cap BF_C) \cap BF_{S\cap C}$ and $GBF_{S\cap C}$ contain only shares of elements in $S \cap C$ and random strings. Because $(GBF_S \cap BF_C) \cap BF_{S\cap C}$ and $GBF_{S\cap C}$ use the same set of hash functions, for each $0 \leq i \leq m-1$, $((GBF_S \cap BF_C) \cap BF_{S\cap C})[i]$ is a share of an element in $C \cap S$ iff $GBF_{S\cap C}$ is a random string. The distribution of a share depends only on the element and the random strings are uniformly distributed. So the distribution of every location in $(GBF_S \cap BF_C) \cap BF_{S\cap C}$ and $GBF_{S\cap C}$ are identical therefore the distributions of $(GBF_S \cap BF_C) \cap BF_{S\cap C}$ and $GBF_{S\cap C}$ are identical.

Then we argue that the distribution of $(GBF_S \cap BF_C) \cap BF_{S\cap C}$ is identical to $GBF_S \cap BF_C$ except for a negligible probability η.

Case 1, $GBF_S \cap BF_C$ encodes at least one elements in $S - C \cap S$. In this case the distribution of $(GBF_S \cap BF_C) \cap BF_{S\cap C}$ differs from the distribution of $GBF_S \cap BF_C$. From Theorem 3, the probability of each element in $S - C \cap S$ being encoded in $GBF_S \cap BF_C$ is ϵ. Since there are $d = |S| - |C \cap S|$ elements in $S - C \cap S$, the probability of at least one element is falsely contained in $GBF_S \cap BF_C$ is:

$$\eta = \text{[skipped...]} \leq 2d\epsilon$$

As we can see η is negligible if ϵ is negligible.

Case 2: $GBF_S \cap BF_C$ encodes only elements from $C \cap S$. In this case, each element of $S - C \cap S$ may leave up to $\eta - 1$ shares in $GBF_S \cap BF_C$. The only difference between $GBF_S \cap BF_C$ and $(GBF_S \cap BF_C) \cap BF_{S\cap C}$ is that in $(GBF_S \cap BF_C) \cap BF_{S\cap C}$, all "residues" shares of elements in $S - C \cap S$ are replaced by random strings. From the security of the XOR-based secret sharing scheme, the residue shares should be uniformly random (otherwise they leak information about the elements). Thus the procedure does not change the distribution when modifying $GBF_S \cap BF_C$ into $(GBF_S \cap BF_C) \cap BF_{S\cap C}$. So the distributions of $GBF_S \cap BF_C$ and $(GBF_S \cap BF_C) \cap BF_{S\cap C}$ are identical. The probability of this case is at least $1 - \eta$.

Since $(GBF_S \cap BF_C) \cap BF_{S\cap C} \equiv GBF_{S\cap C}$ always holds and $GBF_S \cap BF_C \equiv (GBF_S \cap BF_C) \cap BF_{S\cap C}$ in case 2, we can conclude that $Pr[GBF_S \cap BF_C \equiv GBF_{S\cap C}] \geq 1 - \eta$ thus

$|Pr[D(GBF_S \cap BF_C) = 1] - Pr[D(GBF_{S\cap C}) = 1]| \leq \eta.$ □

7.2 Invalidation of the proof in [DCW13]

The end of the proof contains the following assertion: *"$(GBF_S \cap BF_C) \cap BF_{S\cap C} \equiv GBF_{S\cap C}$ always holds and $GBF_S \cap BF_C \equiv (GBF_S \cap BF_C) \cap BF_{S\cap C}$ holds in case 2"*. This should result in $GBF_{S\cap C} \equiv GBF_S \cap BF_C$ in case 2. We invalidate this claim by giving a counter-example. Let the number of hash functions be $\eta = 3$; let x and y be two elements of $S - C$ such that $h_1(x) = h_1(y)$ and that for all $i \neq 1$, $h_i(x) \in h_*(C)$ and $h_i(y) \in h_*(C)$. This example is illustrated in Figure 7.1. Note that this example can be situated in the case 2 of the proof of [DCW13] as it does not require any element of S to have all its positions in $h_*(C)$.

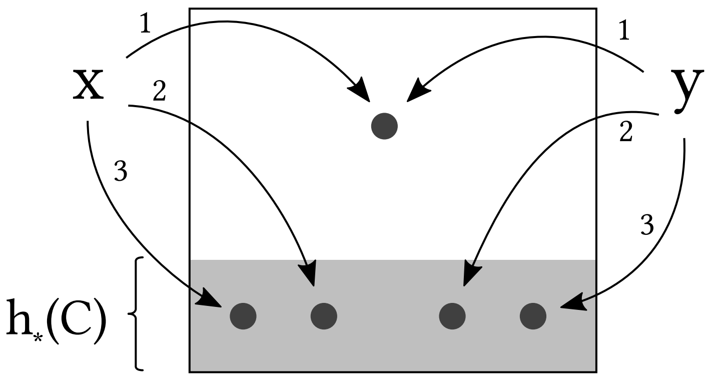


Figure 7.1: Illustration of our counter-example.

By definition, because $x \in S$ and $y \in S$, GBF_S satisfies the following equations where we note $GBF_S[h_i(x)]$ as x_i (and similarly with y):

$$x_1 \oplus x_2 \oplus x_3 = x \tag{7.1}$$

$$y_1 \oplus y_2 \oplus y_3 = y \tag{7.2}$$

$$x_1 = y_1 \tag{7.3}$$

Combining (7.1), (7.2) and (7.3) gives:

$$x \oplus y = x_1 \oplus x_2 \oplus x_3 \oplus y_1 \oplus y_2 \oplus y_3$$
$$x \oplus y = x_2 \oplus x_3 \oplus y_2 \oplus y_3$$

If we re-write the latter equation without our short-hand notation, we have that GBF_S satisfies the following:

$$\begin{aligned} &x \oplus y = \\ &GBF_S[h_2(x)] \oplus GBF_S[h_3(x)] \oplus GBF_S[h_2(y)] \oplus GBF_S[h_3(y)] \end{aligned} \tag{7.4}$$

Regarding $GBF_S \cap BF_C$, it does not satisfy equations (7.1) and (7.2) anymore because the component $GBF_S[h_1(x)]$ was replaced by a fresh random value during the intersection operation; but it still satisfies equation (7.4) as it only involves components that were not re-randomized during intersection, thanks to the fact that $h_2(x), h_3(x), h_2(y)$ and $h_3(y)$ are in $h_*(C)$ (see definition of GBF intersection in Section 6.2.1).

On the other hand $GBF_{S\cap C}$ does not satisfy (7.4) except with very small probability, since $x, y \notin S \cap C$. As a result a GBF where relation (7.4) *does not* hold is a valid outcome for the distribution of $GBF_{S\cap C}$ but not for the distribution of $GBF_S \cap BF_C$. Those distributions cannot be identical, and the proof given in [DCW13] of Theorem 6.2.2 is wrong. The same counter-example can also be used to invalidate the claims that *"$GBF''_{C\cap S} \equiv GBF'_{C\cap S}$"* and that *"$GBF_{C\cap S} \equiv GBF''_{C\cap S}$"*.

This is not just a typo in [DCW13], but truly a flaw in the proof. Recall, the proof uses the fact that any $x \in S - C$ has, with overwhelming probability, one of its positions, say $h_1(x)$, out of

$h_*(C)$. As a result this component is overwritten during intersection (or never retrieved in a PSI scenario). Dong et al. then invoke *"the security of the XOR-based secret sharing scheme"* to argue that no information can be obtained about $x_1 \oplus x_2 \oplus x_3$. But the GBF construction is not the exact same thing as a XOR secret sharing scheme, and the argument does not hold. More precisely, in a GBF the component $GBF_S[h_1(x)]$ (or x_1) may not be independent from other components in the GBF and in particular its value can be tied to the value of other components that may be in $h_*(C)$ and are thus "visible", which is the case with components y_2 and y_3 in our example.

7.2.1 Generalization of the counter-example

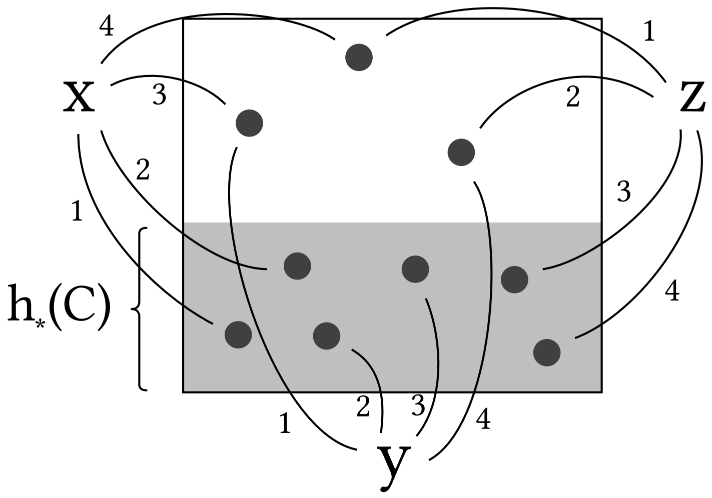


Figure 7.2: An example of a more general counter-example involving 3 elements of $S - C$.

We give a larger class of situations where the same claims prove wrong. Let $P(S, C)$ (or just P if there is no ambiguity about the inputs) be the set of positions that appear an odd number of times in $(h_*(x)\ \forall x \in S - C)$:

$$P(S,C) = \{p \in h_*(S - C)\ :\ |\{(x,i) \in (S - C) \times [\eta]\ :\ h_i(x) = p\}| \bmod 2 = 1\}$$

Then GBF_S satisfies the following relation, of which (7.4) is a special case, and which is obtained the same way as (7.4) was obtained:

$$\bigoplus_{S \cap C} x = \bigoplus_{p \in P} GBF_S[p] \tag{7.5}$$

Moreover if $P \subset h_*(C)$, none of the concerned components are re-randomized during intersection so $GBF_S \cap BF_C$ satisfies the same relation, that is:

$$\bigoplus_{S \cap C} x = \bigoplus_{p \in P} (GBF_S \cap BF_C)[p] \tag{7.6}$$

Figure 7.2 illustrates such a more general case with 4 hash functions and involving 3 elements x, y and z where $GBF_S \cap BF_C$ would satisfy the following relation (but $GBF_{S \cap C}$ would not):

$$x \oplus y \oplus z = x_1 \oplus x_2 \oplus y_2 \oplus y_3 \oplus z_3 \oplus z_4$$

7.3 Other GBF constructions

We describe the consequences of our findings on the other GBF constructions that were inspired by the one of Dong et al., namely the ones of Pinkas et al. [PSZ14, Section 4.3], and Rindal and Rosulek [RR17].

7.3.1 Pinkas et al. [PSZ14]: same situation as Dong et al. [DCW13]

The construction of Pinkas et al. presents many optimizations over the one of Dong et al., for instance through the use of *random OT* (see [Ash+13]) instead of "classical" OT, but it also has a more essential difference with the construction of Dong et al. in that the sum of the components associated to an element does not need to be equal to the element itself. Instead, all component values are all chosen uniformly at random and the for each element in her set, sender sends to the receiver[2] a "summary value" that is the sum of the components corresponding to this element, that is:

$$\{\bigoplus_i GBF_S[h_i(s)] \quad \forall s \in S\}$$

The receiver retrieves the components corresponding to her own elements via OT and computes similar sums for these elements. Finally, the receiver compares the sums she computed with the sums she received to learn which elements are in both sets.

The reasoning used by Pinkas et al. to show the security of this construction is similar to the reasoning of Dong et al., namely, that unless there is a $s \in S$ such that $h_*(s) \subset h_*(C)$ then the view of the receiver is not just computationally indistinguishable from a simulated view, but truly independent from $S - C$.

Unfortunately the same problem appears as with the proof of Dong et al. Take for instance the example of Figure 7.1: if x and y are in the sender's set S, then the receiver must have received these two values:

$$\begin{aligned} K_x &= GBF_S[h_1(x)] \oplus GBF_S[h_2(x)] \oplus GBF_S[h_3(x)] \\ K_y &= GBF_S[h_1(y)] \oplus GBF_S[h_2(y)] \oplus GBF_S[h_3(y)] \end{aligned}$$

and because $h_1(x) = h_1(y)$ the XOR of these two received values is equal to:

$$\begin{aligned} &K_x \oplus K_y = \\ &GBF_S[h_2(x)] \oplus GBF_S[h_3(x)] \oplus GBF_S[h_2(y)] \oplus GBF_S[h_3(y)] \end{aligned} \tag{7.7}$$

Which only depends on values the receiver knows. The receiver is then able to detect the presence of x and y in $S - C$ by testing whether any two summary values have their sum equal to (7.7).

Again, and as with the GBF construction of Dong et al. [DCW13], the construction of Pinkas et al. is actually secure since our new proof given in Section 7.4 should also apply. This means that such S and C are actually very hard to find. Nevertheless this shows that the security of the GBF construction of [PSZ14] cannot be proven simply by invoking the low probability to have $h_*(s) \subset h_*(C)$ for some $s \in S$.

[2]the notion of *sender* and *receiver* here are in the context ot OT which we describe in Section 6.2.2

7.3.2 Rindal and Rosulek [RR17]: no apparent issues

Rindal and Rosulek [RR17] present a new PSI protocol following the idea of the GBF-based PSI protocol of Pinkas et al. [PSZ14, Section 4.3] we presented in the previous section. They keep most of the ideas of Pinkas et al., including the use of random OT and the optimizations it enables, as well as the idea of having the sender sending summary values. However they build these summary values in a different way, which is essentially as follows:

$$K_x = H\left(x \,||\, \bigoplus_{i \in h_*(x)} GBF_S[i]\right)$$

Where H is a secure hash function and $||$ denotes concatenation.

Interestingly, the presence of the hash function breaks the algebraic properties of the summary values that were used in the previous section, meaning that all the counter-examples we gave so far do not apply to the construction of Rindal and Rosulek. This construction may thus be secure even against someone knowing a subset $X \subset S - C$ such that $P(X, C) \subset h_*(C)$. But more importantly, the security proof Rindal and Rosulek give for their construction [RR17, Section 5.3] differs a lot from the proofs of Dong et al. [DCW13] and of Pinkas et al. [PSZ14], and we did not find in the paper of Rindal and Rosulek the issue we identified in [DCW13] and [PSZ14].

Note however that the construction of Rindal and Rosulek and of Pinkas et al. cannot always be used as a drop-in replacement of the original construction of Dong et al. This is the case for our RMÖ18 protocol (see Section 6.2.1) that uses Garbled Bloom Filters but where the receiver looks up several GBFs and must be unable to know what response (in the form of components retrieved) comes from what filter. This requires that the receiver must be able to decide on the result of a lookup ("present" or "absent") using only the components retrieved and without remembering what was the component that was being looked for. Such a property could not be reached in a trivial way using the construction of Rindal and Rosulek (or even the one of Pinkas et al.) because the sending of summary values by the sender requires that the receiver knows what to compare these values with, which requires that the receiver knows what GBF the values correspond to. This shows why the study of the security proof of constructions other than the one of Rindal and Rosulek is still relevant.

7.4 New Proof of Security

We now give a new proof that GBFs satisfy the security property claimed by [DCW13]. Note that the whole proof applies to ZGBFs as well, as one could replace "GBF" by "ZGBF" in the whole section.

7.4.1 New case distinction

Our proof follows the idea of the proof of Dong et al. [DCW13]: we consider two cases, one that occurs with negligible probability and one in which the two distributions are actually identical. Hence the two distributions are indistinguishable. What differs between our proof and the one of

[DCW13] is the case separation: as we saw, the assumption of case 2 of [DCW13] that no element in $S - C$ has all its positions in $h_*(C)$ is not sufficient to have $GBF_S \cap BF_C \equiv GBF_{S \cap C}$.

Instead, we make the following remark: it is very unlikely that there is some subset X of $S - C$ such that all the positions in $h_*(X)$ being mapped by a single element in X happens to be in $h_*(C)$. Said differently, for any subset $X \subset S - C$ there is at least one position in $h_*(X)$ that is both out of $h_*(C)$ and corresponds to a single element of X. Note that this covers the situation described in Section 7.2.1 (an thus our counter-examples in Figures 7.1 and 7.2 too): if all the position in $h_*(S - C)$ mapped an odd number of times are in $h_*(C)$, then all the positions out of $h_*(C)$ are mapped at least 2 times. More formally we define the *mapped-once positions of* X, noted $\boldsymbol{m}(X)$, and the *never-mapped positions of* X, noted $\boldsymbol{n}(X)$, as follows:

Definition 7.4.1 (Mapped-once and never-mapped positions of a set)**.** Let $X \subset \{0,1\}^*$. The set of mapped-once positions of X is defined as:

$$\boldsymbol{m}(X) := \{p \in h_*(X) \ : \ \exists!(x,i) \in X \times [\eta] \ : \ h_i(x) = p\}$$

Similarly, the set of never-mapped positions of X is defined as:

$$\boldsymbol{n}(X) := \{p \in h_*(X) \ : \ \nexists(x,i) \in X \times [\eta] \ : \ h_i(x) = p\}$$

The never-mapped positions of X correspond to the zeroes in BF_X.

We then have the following:

Theorem 7.4.1. Let $X \subset S - C$, then:

$$Pr[\boldsymbol{m}(X) \subset h_*(C)] \leq negl(\lambda) \tag{7.8}$$

Proof: Our explanation is in two parts: First we argue that the size of $\boldsymbol{m}(X)$ is of size greater than η; the probability that all these positions are in $h_*(C)$ is thus lower than the probability for one element to have all its positions in $h_*(C)$, which is already negligible (proved by Dong et al. in their “case 1”). We start by showing that $|\boldsymbol{m}(X)| \geq \eta$ with overwhelming probability. Consider the sequence of sets $X_1, X_2, \ldots, X$ where each set has one more element than the previous one. The number of mapped-once positions of $X_i = X_{i-1} \cup \{x\}$ for some i and some x is then the number of mapped-once positions of X_{i-1} plus the number of elements of $h_*(x)$ that are in $\boldsymbol{n}(X_{i-1})$ (some new mapped-once positions), minus the number of elements of $h_*(x)$ that are in $\boldsymbol{m}(X_{i-1})$ (positions that are not mapped-once anymore). Statistically, we thus have the following expected difference:

$$E[|\boldsymbol{m}(X_i)| - |\boldsymbol{m}(X_{i-1})|] = \eta\frac{\boldsymbol{n}(X_{i-1})}{M} - \eta\frac{\boldsymbol{m}(X_{i-1})}{M} \tag{7.9}$$

That is:

$$\begin{aligned}
\boldsymbol{m}(X_1) &= \eta \\
E[\boldsymbol{m}(X_2)] &= \boldsymbol{m}(X_1) + \eta\frac{M-\eta}{M} - \eta\frac{\eta}{M} \\
&= 2\eta\left(1 - \frac{\eta}{M}\right) \\
&\ldots
\end{aligned}$$

Now because $|X| \leq |S - C| \leq |S| \leq N$, and due to the way GBF parameters are created (see Section 5.3.2) BF_X and *a fortiori* BF_{X_i} should have not less than half of its bits unset, so $\boldsymbol{n}(X_i) \geq M/2$ and $\eta \frac{\boldsymbol{n}(X_{i-1})}{M} \geq \eta/2$. At the same time $\boldsymbol{m}(X_{i-1})$ is always very small compared to M. It should then obvious that $|\boldsymbol{m}(X)| \geq \eta$. Finally as we already explained, the probability for $\boldsymbol{m}(X)$ to be a subset of $h_*(C)$ is negligible because it is less than the probability for a single element to have all its positions in $h_*(C)$, given that we already show it has a size greater than η. □

7.4.2 New Proof of Security

We give a new proof of Theorem 6.2.2, that is, we show that:

$$(S, C, \mathsf{Extract}(BF_C, GBF_S)) \overset{c}{\equiv} (S, C, \mathsf{Extract}(BF_C, GBF_{S \cap C}))$$

We consider two cases as it is done by Dong et al. [DCW13]: The first case is where there is a $X \subset S - C$ such that $\boldsymbol{m}(X) \subset h_*(C)$. From Theorem 7.4.1, This case happens with negligible probability. The second case is thus where there is no such X, and we show that in this case the distributions are identical by showing that any outcome of one distribution is a valid outcome of the other. Let (S, C, B) be an outcome of the right-hand distribution, that is, of $(S, C, \mathsf{Extract}(BF_C, GBF_{S \cap C}))$; we show how to build a GBF B' that is a valid outcome of GBF_S such that $Extract(BF_C, B') = B$. We build B' the following way: We start from B which, recall, is a Garbled Bloom Filter with all its components not in domain of C being empty. We will insert each element of $S - C$ in B, keeping components that were already set untouched. Insertion happens just as in the GBF.Build algorithm. When all elements have been inserted, the remaining components are filled with random values, just as in the end of GBF.Build. If the algorithm did not halt, the resulting B' encodes every element of $S \cap C$ (from the initial values from B) and every element of $S - C$ (that we just inserted). As a result, B' is a valid GBF_S and $Extract(BF_C, B')$ is a valid outcome for $Extract(BF_C, GBF_S)$.

We now show that the algorithm does not halt. Recall, the building algorithm halts when an element that must be inserted only maps to positions that are not empty. Since we are in the case where no $X \subset S - C$ satisfies $\boldsymbol{m}(X) \subset h_*(C)$, there must be a position in $h_*(S - C)$ that is not in $h_*(C)$ and which is mapped by a single element $y \in S - C$. As a result if $(S - C) - \{y\}$ was inserted without halting, then the final y can be inserted without halting as well. This reasoning can be repeated to show that $(S - C) - \{y\}$ can be inserted without halting as well, and recursively $S - C$ can be inserted entirely without halting.

Finally given an outcome B of $Extract(BF_C, GBF_S)$ one can trivially build a valid GBF B' encoding $S \cap C$ such that $Extract(BF_C, B') = B$: it suffices to fill all empty components of B with random values. As a result we have $Extract(BF_C, GBF_S) \equiv Extract(BF_C, GBF_{S \cap C})$ in our second case, and this ends the proof of Theorem 6.2.2. □

Note that this proof would also apply to the construction of Pinkas et al. [PSZ14].

7.5 Related Work

Security issues in the paper of Dong et al. [DCW13] where identified by Rindal and Rosulek [RR17] and by Lambæk [Lam16], but none of these issues apply on the protocol that we study in this chapter. Indeed, [DCW13] describes two protocols: one that aims at providing security against honest-but-curious adversaries, which is the one that is being studied in this chapter, and one that aims at providing security against malicious adversaries. The issues identified in [RR17] and [Lam16] only concern the malicious-security protocol, and do not apply to the honest-but-curious-security protocol (both present the honest-but-curious-security protocol as satisfying its claimed properties).

By contrast, the issues we identify concern the security of the GBF construction. This property is invoked in the security proofs for both the honest-but-curious-security protocol and the malicious-security one, so the two protocols are affected. The issue we identify is thus different, and more general, than the ones identified in [RR17] and [Lam16].

7.6 Conclusion

The Garbled Bloom Filter data structure attracted a significant amount of attention in the domain of cryptography for cloud security, and seems to enable a large amount of efficient new protocols. In particular, it is very important for us as it is a central piece of our RMÖ18 protocol whose design would not have been possible without the properties of GBFs. It is then of great importance that the proof for the privacy properties of GBFs has no flaws, not only for our our RMÖ18 protocol but for potentially many others.

In this chapter, corresponding to the work we published at DBSec 2018 [RÖ18], we showed that the original proof of security of GBFs relies on an analogy between GBFs and XOR-based secret sharing that is not entirely correct and allows counter-examples disproving the main argument used in the proof. Fortunately we are able to give a new proof that shows that GBFs do satisfy the security property that we and others rely on.

8 Conclusion

With cloud computing already a very widespread phenomenon and expectations that the cloud computing market keeps on growing in the near future, it is urgent to develop technologies that protect the privacy of users against the potential corruption of Cloud Service Providers (CSPs). Nevertheless, these necessary improvements in term of privacy must not reduce too much the functionalities cloud computing can offer.

Searchable Encryption (SE) is a family of cryptographic protocols that allow users to search data hosted at a remote server while protecting the privacy of the data and the search queries. While SE has been studied extensively in the literature in the setting with a single user, MUSE is a more general variant where the cloud-hosted database is owned and accessed by multiple users which do not trust each other *a priori.* MUSE is by nature even harder to solve than SSE; however its study seems necessary as databases in SE get larger and as new regulations like the European GDPR [GDPR], or older ones as the US HIPAA [HIPAA], make it mandatory to enforce the right of individuals to have control over their own personal data.

In this thesis, we showed that the threat model that was used by many early papers on MUSE, that did not include the users as part of the threat, was not relevant. Not only was it not practical, as one should likely not use a MUSE protocol in a situation where all users can be trusted; but this threat model also encouraged the use of design patterns, like the "iterative testing structure" we describe, which are incompatible with privacy in a more realistic threat model. We showed that, while the need for MUSE protocols being secure against corrupted users was already acknowledged in the research community, none of the existing protocols reached such privacy properties, including protocols that were specifically designed with this objective in mind.

By identifying the common design pattern that causes all existing MUSE protocols to be vulnerable to collusions, by giving a thorough study of how it causes such vulnerability and by showing why the previous security analyses were not able to reveal such vulnerability, we contributed to preventing such privacy issue to occur in future MUSE protocols. The new definition of security we give for MUSE protocols, which follows the standard one used in the single-user setting and seems much less error-prone than the ones used before, is another contribution in the same direction.

We then present techniques that led to the design of the first MUSE protocols secure against user-server collusions. We present three MUSE protocols, each offering a different combination of privacy and efficiency properties, covering a large number of situations and needs. While there is still room for improvement on many aspects with these protocols, especially regarding server-side complexity, we hope our work demonstrated that it is possible to provide privacy against user-server collusions in MUSE without sacrificing on user-side complexity which is the main focus regarding scalability in cloud computing. Hopefully, the techniques we design for the design of such MUSE protocols will pave the way for future protocols. We showed that the use of two non-colluding servers, as well as the duplication of every record for every reader authorized to search it, allow the construction of MUSE protocols that are secure in a realistic threat model, and are thus able to improve the state of the art on MUSE protocols, despite seeming overly costly at first sight.

Our study of the security of previous MUSE protocols already has documented impact on research on MUSE, and our work on the design of new MUSE protocols also made us improve the security of some highly popular cryptographic primitives that we had to use in our protocols.

Future Work Given that moving away from the iterative testing structure in the design of MUSE protocols required to completely change the way of building such protocols, much progress remains to be done in MUSE constructions.

In this thesis we present 3 MUSE protocols that represent various trade-off between privacy and efficiency, as can be seen in Figure 6.4. However there is still much to "explore" on this graph. In particular, it would be interesting to see if the server-side complexity of the RMÖ18 protocol could be reduced significantly by allowing a leakage profile slightly bigger (in RMÖ18 the proxy sees the result length and the server sees nothing beyond the benign leakage and revealed content), while still being smaller than the one of DH-AP-MUSE (access pattern leakage) for which there is a growing concern in the research community.

In the event where the current doubts on the security of (strict) access pattern leakage are disproved, it could also be interesting to see if a leakage profile slightly bigger than the one of DH-AP-MUSE (where the proxy sees the access pattern and the server sees nothing) could also lead to significant improvements in terms of efficiency. Typically, one could hope to find a way to reach sub-linear search operations as it is now common in single-user SE. Currently, the only MUSE protocols that can provide sub-linear search operations are the single-key ones (see Section 3.3.2), which are extremely weak against user-server collusions.

Of course, any result that would contribute to removing the need for the assumption to have two non-colluding servers would be a significant contribution. This thesis showed that such assumption seemed, as of today, the simplest way to achieve privacy against user-server collusions in MUSE protocols; and we have the feeling that there is no simple way to obtain similar privacy guarantees with a single server. But we could be wrong. Nevertheless, even a rather complex one-server solution would be a great achievement since there may be scenarios where it would be unrealistic to rely on a 2-server architecture.

Other "obvious" directions for future work would consist in trying to incorporate the advanced features that have been developed in single-user SE, such as improved query expressiveness (boolean queries, range queries, aiming to provide a full replacement to current database systems), and backward/forward privacy (i.e. proper security for database dynamism).

Finally, another interesting direction for future work would be the modeling of users as fully malicious adversaries in the threat model. Indeed, while there are strong arguments in favor of modeling the CSP as an honest-but-curious adversary (public exposure, large number of employees ...), these arguments do not apply on end users. As a result, an attacker taking control of the CSP and some users could be unable to alter the behavior of the CSP but have no difficulty in doing so with the corrupted users. Finding a definition of security that captures this kind of situation would be a challenging but very promising task.

Bibliography

[AES] *ADVANCED ENCRYPTION STANDARD (AES).* 197. National Institute of Standards and Technology (NIST), Nov. 26, 2001. URL: https://doi.org/10.6028/NIST.FIPS.197.

[Ang+] S. Angel, H. Chen, K. Laine, and S. Setty. "PIR with compressed queries and amortized query processing". In: *2018 IEEE Symposium on Security and Privacy (SP).* Vol. 00, pp. 1011–1028. DOI: 10.1109/SP.2018.00062. URL: doi.ieeecomputersociety.org/10.1109/SP.2018.00062.

[Asg+13] Muhammad Rizwan Asghar, Giovanni Russello, Bruno Crispo, and Mihaela Ion. "Supporting complex queries and access policies for multi-user encrypted databases". In: *CCSW'13, Proceedings of the 2013 ACM Cloud Computing Security Workshop, Co-located with CCS 2013, Berlin, Germany, November 4, 2013.* 2013, pp. 77–88. DOI: 10.1145/2517488.2517492.

[Ash+13] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. "More efficient oblivious transfer and extensions for faster secure computation". In: *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013.* 2013, pp. 535–548. DOI: 10.1145/2508859.2516738. URL: http://doi.acm.org/10.1145/2508859.2516738.

[AWS] *Page "What is Cloud Computing?" on the website of Amazon Web Services.* https://aws.amazon.com/what-is-cloud-computing/. Accessed on May 2018.

[Bao+08] Feng Bao, Robert H. Deng, Xuhua Ding, and Yanjiang Yang. "Private Query on Encrypted Data in Multi-user Settings". In: *Information Security Practice and Experience, 4th International Conference, ISPEC 2008, Sydney, Australia, April 21-23, 2008, Proceedings.* 2008, pp. 71–85. DOI: 10.1007/978-3-540-79104-1_6.

[Bel+97] Mihir Bellare, Anand Desai, E. Jokipii, and Phillip Rogaway. "A Concrete Security Treatment of Symmetric Encryption". In: *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997.* 1997, pp. 394–403. DOI: 10.1109/SFCS.1997.646128. URL: https://doi.org/10.1109/SFCS.1997.646128.

[Ber08] Daniel J. Bernstein. “The Salsa20 Family of Stream Ciphers”. In: *New Stream Cipher Designs - The eSTREAM Finalists.* 2008, pp. 84–97. DOI: 10.1007/978-3-540-68351-3_8. URL: https://doi.org/10.1007/978-3-540-68351-3_8.

[Blo70] Burton H. Bloom. “Space/Time Trade-offs in Hash Coding with Allowable Errors”. In: *Commun. ACM* 13.7 (July 1970), pp. 422–426. ISSN: 0001-0782. DOI: 10.1145/362686.362692. URL: http://doi.acm.org/10.1145/362686.362692.

[BLS04] Dan Boneh, Ben Lynn, and Hovav Shacham. “Short Signatures from the Weil Pairing”. In: *J. Cryptology* 17.4 (2004), pp. 297–319. DOI: 10.1007/s00145-004-0314-9. URL: https://doi.org/10.1007/s00145-004-0314-9.

[BM03] Andrei Z. Broder and Michael Mitzenmacher. “Survey: Network Applications of Bloom Filters: A Survey”. In: *Internet Mathematics* 1.4 (2003), pp. 485–509. DOI: 10.1080/15427951.2004.10129096.

[BMO17] Raphaël Bost, Brice Minaud, and Olga Ohrimenko. “Forward and Backward Private Searchable Encryption from Constrained Cryptographic Primitives”. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017.* 2017, pp. 1465–1482. DOI: 10.1145/3133956.3133980. URL: http://doi.acm.org/10.1145/3133956.3133980.

[Bon+04] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. “Public Key Encryption with Keyword Search”. In: *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings.* 2004, pp. 506–522. DOI: 10.1007/978-3-540-24676-3_30. URL: https://doi.org/10.1007/978-3-540-24676-3_30.

[Bös+14] Christoph Bösch, Pieter Hartel, Willem Jonker, and Andreas Peter. “A Survey of Provably Secure Searchable Encryption”. In: *ACM Computing Surveys* 47.2 (Aug. 25, 2014), pp. 1–51. ISSN: 03600300. DOI: 10.1145/2636328. (Visited on 12/17/2014).

[Bos16] Raphael Bost. “$\sum o\varphi o\varsigma$: Forward Secure Searchable Encryption”. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016.* 2016, pp. 1143–1154. DOI: 10.1145/2976749.2978303. URL: http://doi.acm.org/10.1145/2976749.2978303.

[Bos18] Raphaël Bost. “Searchable Encryption — New Constructions of Encrypted Databases”. PhD thesis. 2018.

[Bou+16] Florian Bourse, Rafaël Del Pino, Michele Minelli, and Hoeteck Wee. "FHE Circuit Privacy Almost for Free". In: *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II.* 2016, pp. 62–89. DOI: 10.1007/978-3-662-53008-5_3. URL: https://doi.org/10.1007/978-3-662-53008-5_3.

[BR93] Mihir Bellare and Phillip Rogaway. "Random Oracles are Practical: A Paradigm for Designing Efficient Protocols". In: *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993.* 1993, pp. 62–73. DOI: 10.1145/168588.168596. URL: http://doi.acm.org/10.1145/168588.168596.

[BSW16] Mihir Bellare, Igors Stepanovs, and Brent Waters. "New Negative Results on Differing-Inputs Obfuscation". In: *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II.* 2016, pp. 792–821. DOI: 10.1007/978-3-662-49896-5_28. URL: https://doi.org/10.1007/978-3-662-49896-5_28.

[Cas+13] David Cash, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. "Highly-Scalable Searchable Symmetric Encryption with Support for Boolean Queries". In: *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I.* 2013, pp. 353–373. DOI: 10.1007/978-3-642-40041-4_20. URL: https://doi.org/10.1007/978-3-642-40041-4_20.

[Cas+14] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. "Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation". In: *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014.* 2014. URL: https://www.ndss-symposium.org/ndss2014/dynamic-searchable-encryption-very-large-databases-data-structures-and-implementation.

[Cas+15] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. "Leakage-Abuse Attacks Against Searchable Encryption". In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015.* 2015, pp. 668–679. DOI: 10.1145/2810103.2813700.

[ChaCha20] Daniel J. Bernstein. *ChaCha, a variant of Salsa20.* 2008. URL: http://cr.yp.to/chacha/chacha-20080128.pdf.

[Charm-Crypto] *Charm: A tool for rapid cryptographic prototyping.* http://charm-crypto.io/.

[CS10] Craig Costello and Douglas Stebila. “Fixed Argument Pairings”. In: *Progress in Cryptology - LATINCRYPT 2010, First International Conference on Cryptology and Information Security in Latin America, Puebla, Mexico, August 8-11, 2010, Proceedings.* 2010, pp. 92–108. DOI: 10.1007/978-3-642-14712-8_6. URL: https://doi.org/10.1007/978-3-642-14712-8_6.

[Cur+06] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. “Searchable symmetric encryption: improved definitions and efficient constructions”. In: *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, Ioctober 30 - November 3, 2006.* 2006, pp. 79–88. DOI: 10.1145/1180405.1180417.

[Dav+17] Dave Bartoletti, Lauren E. Nelson, Liz Herbert, Paul Miller, Charlie Dai, Andras Cser, and Andre Kindness. “Predictions 2018: Cloud Computing Accelerates Enterprise Transformation Everywhere”. In: *Forrester* (2017). URL: https://go.forrester.com/blogs/predictions-2018-cloud-computing-accelerates-enterprise-transformation-everywhere/.

[DCW13] Changyu Dong, Liqun Chen, and Zikai Wen. “When private set intersection meets big data: an efficient and scalable protocol”. In: *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS’13, Berlin, Germany, November 4-8, 2013.* 2013, pp. 789–800. DOI: 10.1145/2508859.2516701.

[DRD08] Changyu Dong, Giovanni Russello, and Naranker Dulay. “Shared and Searchable Encrypted Data for Untrusted Servers”. In: *Data and Applications Security XXII, 22nd Annual IFIP WG 11.3 Working Conference on Data and Applications Security, London, UK, July 13-16, 2008, Proceedings.* 2008, pp. 127–143. DOI: 10.1007/978-3-540-70567-3_10.

[DS16] Léo Ducas and Damien Stehlé. “Sanitization of FHE Ciphertexts”. In: *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I.* 2016, pp. 294–310. DOI: 10.1007/978-3-662-49890-3_12. URL: https://doi.org/10.1007/978-3-662-49890-3_12.

[Enron] *Enron Email Dataset.* http://www.cs.cmu.edu/~enron/. Accessed on May 2016.

[EÖM14] Kaoutar Elkhiyaoui, Melek Önen, and Refik Molva. “Privacy Preserving Delegated Word Search in the Cloud”. In: *SECRYPT 2014 - Proceedings of the 11th International Conference on Security and Cryptography, Vienna, Austria, 28-30 August, 2014.* 2014, pp. 137–150. DOI: 10.5220/0005054001370150.

[Ful+17] Benjamin Fuller, Mayank Varia, Arkady Yerukhimovich, Emily Shen, Ariel Hamlin, Vijay Gadepally, Richard Shay, John Darby Mitchell, and Robert K. Cunningham. "SoK: Cryptographically Protected Database Search". In: *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. 2017, pp. 172–191. DOI: 10.1109/SP.2017.10.

[GDPR] *European General Data Protection Regulation*. https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32016R0679. Accessed on May 2018.

[Gen09] Craig Gentry. "Fully homomorphic encryption using ideal lattices". In: *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*. 2009, pp. 169–178. DOI: 10.1145/1536414.1536440. URL: http://doi.acm.org/10.1145/1536414.1536440.

[Gol87] Oded Goldreich. "Towards a Theory of Software Protection and Simulation by Oblivious RAMs". In: *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*. 1987, pp. 182–194. DOI: 10.1145/28395.28416. URL: http://doi.acm.org/10.1145/28395.28416.

[Gru+16] Paul Grubbs, Richard McPherson, Muhammad Naveed, Thomas Ristenpart, and Vitaly Shmatikov. "Breaking Web Applications Built On Top of Encrypted Data". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*. 2016, pp. 1353–1364. DOI: 10.1145/2976749.2978351.

[Ham+18a] Ariel Hamlin, Abhi Shelat, Mor Weiss, and Daniel Wichs. "Multi-Key Searchable Encryption, Revisited". In: *Public-Key Cryptography - PKC 2018 - 21st IACR International Conference on Practice and Theory of Public-Key Cryptography, Rio de Janeiro, Brazil, March 25-29, 2018, Proceedings, Part I*. 2018, pp. 95–124. DOI: 10.1007/978-3-319-76578-5_4. URL: https://doi.org/10.1007/978-3-319-76578-5_4.

[Ham+18b] Ariel Hamlin, Abhi Shelat, Mor Weiss, and Daniel Wichs. "Multi-Key Searchable Encryption, Revisited". In: *IACR Cryptology ePrint Archive* 2018 (2018), p. 18. URL: http://eprint.iacr.org/2018/018.

[HFH99] Bernardo A. Huberman, Matthew K. Franklin, and Tad Hogg. "Enhancing privacy and trust in electronic communities". In: *EC*. 1999, pp. 78–86. DOI: 10.1145/336992.337012. URL: http://doi.acm.org/10.1145/336992.337012.

[HIPAA] *Health Insurance Portability and Accountability Act of 1996*. https://www.congress.gov/bill/104th-congress/house-bill/3103. Accessed on May 2018.

[HL07] Yong Ho Hwang and Pil Joong Lee. “Public Key Encryption with Conjunctive Keyword Search and Its Extension to a Multi-user System”. In: *Pairing-Based Cryptography - Pairing 2007, First International Conference, Tokyo, Japan, July 2-4, 2007, Proceedings.* 2007, pp. 2–22. DOI: 10.1007/978-3-540-73489-5_2.

[IKK12] M Islam, Mehmet Kuzu, and Murat Kantarcioglu. “Access pattern disclosure on searchable encryption: Ramification, attack and mitigation”. In: *Network and Distributed System Security Symposium (NDSS'12)* (2012).

[IPS15] Yuval Ishai, Omkant Pandey, and Amit Sahai. “Public-Coin Differing-Inputs Obfuscation and Its Applications”. In: *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II.* 2015, pp. 668–697. DOI: 10.1007/978-3-662-46497-7_26. URL: https://doi.org/10.1007/978-3-662-46497-7_26.

[Jar+13] Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. “Outsourced symmetric private information retrieval”. In: *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013.* 2013, pp. 875–888. DOI: 10.1145/2508859.2516730. URL: http://doi.acm.org/10.1145/2508859.2516730.

[Kel+16] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. “Generic Attacks on Secure Outsourced Databases”. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016.* 2016, pp. 1329–1340. DOI: 10.1145/2976749.2978386.

[Kia+16] Aggelos Kiayias, Ozgur Oksuz, Alexander Russell, Qiang Tang, and Bing Wang. “Efficient Encrypted Keyword Search for Multi-user Data Sharing”. In: *Computer Security - ESORICS 2016 - 21st European Symposium on Research in Computer Security, Heraklion, Greece, September 26-30, 2016, Proceedings, Part I.* 2016, pp. 173–195. DOI: 10.1007/978-3-319-45744-4_9. URL: https://doi.org/10.1007/978-3-319-45744-4_9.

[Kis+17] Ágnes Kiss, Jian Liu, Thomas Schneider, N. Asokan, and Benny Pinkas. “Private Set Intersection for Unequal Set Sizes with Mobile Applications”. In: *PoPETs* 2017.4 (2017), pp. 177–197. DOI: 10.1515/popets-2017-0044. URL: https://doi.org/10.1515/popets-2017-0044.

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition.* CRC Press, 2014. ISBN: 9781466570269.

[KM15] Neal Koblitz and Alfred J. Menezes. “The random oracle model: a twenty-year retrospective”. In: *Des. Codes Cryptography* 77.2-3 (2015), pp. 587–

610. DOI: 10.1007/s10623-015-0094-2. URL: https://doi.org/10.1007/s10623-015-0094-2.

[KM17] Seny Kamara and Tarik Moataz. "Boolean Searchable Symmetric Encryption with Worst-Case Sub-linear Complexity". In: *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part III.* 2017, pp. 94–124. DOI: 10.1007/978-3-319-56617-7_4. URL: https://doi.org/10.1007/978-3-319-56617-7_4.

[KO97] Eyal Kushilevitz and Rafail Ostrovsky. "Replication is NOT Needed: SINGLE Database, Computationally-Private Information Retrieval". In: *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997.* 1997, pp. 364–373. DOI: 10.1109/SFCS.1997.646125. URL: https://doi.org/10.1109/SFCS.1997.646125.

[KPR12] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. "Dynamic searchable symmetric encryption". In: *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012.* 2012, pp. 965–976. DOI: 10.1145/2382196.2382298. URL: http://doi.acm.org/10.1145/2382196.2382298.

[Krawczyk06] *Hugo Krawczyk: Probabilistic Searchable Symmetric Encryption.* https://www.youtube.com/watch?v=WAefzYSJLDw. Video on YouTube by user "barilanuniversity"; accessed on March 2018.

[Lam16] Mikkel Lambaek. "Breaking and Fixing Private Set Intersection Protocols". In: *IACR Cryptology ePrint Archive* 2016 (2016), p. 665. URL: http://eprint.iacr.org/2016/665.

[libsodium] *The Sodium crypto library.* https://libsodium.org. Accessed on May 2018.

[Lin17] Yehuda Lindell. "How to Simulate It - A Tutorial on the Simulation Proof Technique". In: *Tutorials on the Foundations of Cryptography.* 2017, pp. 277–346. DOI: 10.1007/978-3-319-57048-8_6. URL: https://doi.org/10.1007/978-3-319-57048-8_6.

[Lip05] Helger Lipmaa. "An Oblivious Transfer Protocol with Log-Squared Communication". In: *Information Security, 8th International Conference, ISC 2005, Singapore, September 20-23, 2005, Proceedings.* 2005, pp. 314–328. DOI: 10.1007/11556992_23. URL: https://doi.org/10.1007/11556992_23.

[LPR13] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. "On Ideal Lattices and Learning with Errors over Rings". In: *J. ACM* 60.6 (2013), 43:1–43:35. DOI: 10.1145/2535925. URL: http://doi.acm.org/10.1145/2535925.

[Mel+14] Carlos Aguilar Melchor, Joris Barrier, Laurent Fousse, and Marc-Olivier Killijian. “XPIRe: Private Information Retrieval for Everyone”. In: *IACR Cryptology ePrint Archive* 2014 (2014), p. 1025. URL: `http://eprint.iacr.org/2014/1025`.

[Mel+16] Carlos Aguilar Melchor, Joris Barrier, Laurent Fousse, and Marc-Olivier Killijian. “XPIR : Private Information Retrieval for Everyone”. In: *PoPETs* 2016.2 (2016), pp. 155–174.

[Meteor] *Meteor*. `https://www.meteor.com/`. Accessed on May 2016.

[Mylar-Atk] *Web Page of [RMÖ17]*. `http://www.eurecom.fr/~vanrompa/iterative-testing/`.

[Mylar-NDSI] *Presentation Video of TODO*. `https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/popa`. Relevant part starts at 07:09.

[Mylar-RWC] *Raluca Ada Popa RWC 2017*. `https://youtu.be/-nV2Rdddk0M?t=7m33s`. Video on Youtube by user ”Real World Crypto”; Relevant part starts at 07:33.

[Mylar-Webpage] *Mylar*. `https://css.csail.mit.edu/mylar/`. Accessed on March 2018.

[Mylar-Website-Security] *Page of the Mylar website*. `https://css.csail.mit.edu/mylar/security.html`. Accessed on May 2018.

[Pap+14] Vasilis Pappas, Fernando Krell, Binh Vo, Vladimir Kolesnikov, Tal Malkin, Seung Geol Choi, Wesley George, Angelos Keromytis, and Steve Bellovin. “Blind Seer: A Scalable Private DBMS”. In: *Proceedings of the 2014 IEEE Symposium on Security and Privacy*. SP ’14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 359–374. ISBN: 978-1-4799-4686-0. DOI: `10.1109/SP.2014.30`.

[Pop+14] Raluca Ada Popa, Emily Stark, Steven Valdez, Jonas Helfer, Nickolai Zeldovich, and Hari Balakrishnan. “Building Web Applications on Top of Encrypted Data Using Mylar”. In: *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2014, Seattle, WA, USA, April 2-4, 2014.* 2014, pp. 157–172.

[Por80] M.F. Porter. “A Survey of Provably Secure Searchable Encryption”. In: *Program* 14.3 (1980), pp. 130–137.

[PSZ14] Benny Pinkas, Thomas Schneider, and Michael Zohner. “Faster Private Set Intersection Based on OT Extension”. In: *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014.* 2014, pp. 797–812. URL: `https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/pinkas`.

[PZ13] Raluca A. Popa and Nickolai Zeldovich. “Multi-Key Searchable Encryption”. In: *IACR Cryptology ePrint Archive* 2013 (2013), p. 508. URL: `http://eprint.iacr.org/2013/508`.

[RMÖ15] Cédric Van Rompay, Refik Molva, and Melek Önen. "Multi-user Searchable Encryption in the Cloud". In: *Information Security - 18th International Conference, ISC 2015, Trondheim, Norway, September 9-11, 2015, Proceedings.* 2015, pp. 299–316. DOI: 10.1007/978-3-319-23318-5_17. URL: https://doi.org/10.1007/978-3-319-23318-5_17.

[RMÖ17] Cédric Van Rompay, Refik Molva, and Melek Önen. "A Leakage-Abuse Attack Against Multi-User Searchable Encryption". In: *PoPETs* 2017.3 (2017), p. 168. DOI: 10.1515/popets-2017-0034. URL: https://doi.org/10.1515/popets-2017-0034.

[RMÖ18a] Cédric Van Rompay, Refik Molva, and Melek Önen. "Fast Two-Server Multi-User Searchable Encryption with Strict Access Pattern Leakage". In: *Information and Communications Security - 20th International Conference, ICICS 2018, Lille, France, October 29-31, 2018, Proceedings.* to be published. 2018.

[RMÖ18b] Cédric Van Rompay, Refik Molva, and Melek Önen. "Secure and Scalable Multi-User Searchable Encryption". In: *Proceedings of the Sixth ACM International Workshop on Security in Cloud Computing, SCC@AsiaCCS 2018, Songdo, Incheon, Korea, June 4, 2018.* 2018. DOI: 10.1145/3201595.3201597. URL: http://doi.acm.org/10.1145/3201595.3201597.

[RÖ18] Cédric Van Rompay and Melek Önen. "Breaking and Fixing the Security Proof of Garbled Bloom Filters". In: *Data and Applications Security and Privacy XXXII - 32nd Annual IFIP WG 11.3 Conference, DBSec 2018, Bergamo, Italy, July 16-18, 2018, Proceedings.* 2018.

[RR17] Peter Rindal and Mike Rosulek. "Improved Private Set Intersection Against Malicious Adversaries". In: *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part I.* 2017, pp. 235–259.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". In: *Commun. ACM* 21.2 (1978), pp. 120–126. DOI: 10.1145/359340.359342. URL: http://doi.acm.org/10.1145/359340.359342.

[SealPIR-GitHub] *SealPIR source code on GitHub.* https://github.com/pung-project/sealpir.

[SPS14] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. "Practical Dynamic Searchable Encryption with Small Leakage". In: *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014.* 2014.

[Ste98] Julien P. Stern. "A New Efficient All-Or-Nothing Disclosure of Secrets Protocol". In: *Advances in Cryptology - ASIACRYPT '98, International Conference on the Theory and Applications of Cryptology and Information Security, Beijing, China, October 18-22, 1998, Proceedings.* 1998, pp. 357–371. DOI: 10.1007/3-540-49649-1_28. URL: https://doi.org/10.1007/3-540-49649-1_28.

[SWP00] Dawn Xiaodong Song, David A. Wagner, and Adrian Perrig. "Practical Techniques for Searches on Encrypted Data". In: *2000 IEEE Symposium on Security and Privacy, Berkeley, California, USA, May 14-17, 2000.* 2000, pp. 44–55. DOI: 10.1109/SECPRI.2000.848445.

[Tan14] Qiang Tang. "Nothing is for Free: Security in Searching Shared and Encrypted Data". In: *IEEE Trans. Information Forensics and Security* 9.11 (2014), pp. 1943–1952. DOI: 10.1109/TIFS.2014.2359389. URL: https://doi.org/10.1109/TIFS.2014.2359389.

[TREDISEC] *TREDISEC european research project.* http://www.tredisec.eu/.

[UCN] *Website of the User Centric Networking (UCN) european research project.* https://usercentricnetworking.eu/.

[UCN Final Report] *UCN deliverable D7.5: Final Report.* http://usercentricnetworking.eu/wp-content/uploads/UCN_D7.5_Final-Report.pdf.

[XPIR-GitHub] *XPIR source code on GitHub.* https://github.com/XPIR-team/XPIR.

[Yan+14] Jun Yang, Zheli Liu, Jin Li, Chunfu Jia, and Baojiang Cui. "Multi-key Searchable Encryption without Random Oracle". In: *2014 International Conference on Intelligent Networking and Collaborative Systems, Salerno, Italy, September 10-12, 2014.* 2014, pp. 79–84. DOI: 10.1109/INCoS.2014.143. URL: https://doi.org/10.1109/INCoS.2014.143.

[Yan+15] Jun Yang, Chuan Fu, Nan Shen, Zheli Liu, Chunfu Jia, and Jin Li. "General Multi-key Searchable Encryption". In: *29th IEEE International Conference on Advanced Information Networking and Applications Workshops, AINA 2015 Workshops, Gwangju, South Korea, March 24-27, 2015.* 2015, pp. 89–95. DOI: 10.1109/WAINA.2015.18. URL: https://doi.org/10.1109/WAINA.2015.18.

[YLW11] Yanjiang Yang, Haibing Lu, and Jian Weng. "Multi-User Private Keyword Search for Cloud Computing". In: IEEE, Nov. 2011, pp. 264–271. ISBN: 978-1-4673-0090-2 978-0-7695-4622-3. DOI: 10.1109/CloudCom.2011.43. (Visited on 10/03/2014).

[ZKP16] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. "All Your Queries Are Belong to Us: The Power of File-Injection Attacks on Searchable Encryption". In: *25th USENIX Security Symposium (USENIX Security 16).* Austin, TX: USENIX Association, Aug. 2016, pp. 707–720. ISBN: 978-1-931971-32-4.

List of Symbols Used

We list some symbols that are used several times across the manuscript and recall what they represent:

Latin Alphabet

- $a_{r,l,s}$: result of query $q_{r,l,s}$; In Definition 2.2.1, a is used with b and c as temporary variables
- $Auth$: function mapping a reader to the ids of records this reader is authorized to search
- d: A record id
- e: a bilinear map
- $\mathbb{G}$: A group where the DDH problem is hard
- g: a generator of the DDH-hard group
- h_i (with i a natural number): A Bloom Filter hash function
- h_e: a cryptographic hash function used with the bilinear pairing
- H: a secure hash function from bit strings to the DDH-hard group $\mathbb{G}$
- $\mathcal{H}$: the history (instanciation) of a MUSE protocol execution
- i and j: used as temporary variables
- M: number of buckets in a BF/GBF/ZGBF
- n: used as temporary variable
- $p_{r,l,s}$: response corresponding to query $q_{r,l,s}$
- P: an response in an OT protocol
- $q_{r,l,s}$ or $\boldsymbol{q}[r][l][s]$: the s-th query of reader $r \in R$ during time period l
- R: set of all readers in the system
- $\mathcal{S}$ or $\mathcal{S}_i$ for $i \in \mathbb{N}$: a simulator as defined by Definition 3.2.3
- $t_{r,l,s}$: trapdoor (encrypted query sent by a reader) corresponding to query $q_{r,l,s}$
- W_d or $\boldsymbol{W}[d]$: the record with id d
- $\overline{W}_d$: the record W_d after encryption
- x and y: used as temporary variables

Greek Alphabet

- γ_d: the record key used to encrypt record W_d
- $\Delta_{r,d}$: in some MUSE protocols (not the ones we present in this manuscript), the *delta token* enabling the transformation of trapdoors from reader r to transformed trapdoors applicable to record W_d
- η: the number of hash functions in a BF/GBF/ZGBF
- λ: the security parameter
- ζ: the order of the DDH-hard group
- $\rho_{r,l}$: the key used by reader r to create trapdoors during time period l

List of Figures

1 Une illustration du contexte multi-utilisateurs vii
2 Une illustration du phénomène à l'origine de la vulnérabilité de la structure en test itéré xi
3 Taux de décryption dans une de nos simulations, à nombre de requêtes observées variable. xii
4 Une capture d'écran d'une conversation dans une application de messagerie protégée par le programme Mylar, et le résultat de notre programme appliquant notre attaque contre Mylar. xiii
5 Une illustration de la différence entre une structure à clé unique (a) telle qu'employé avant MKSE, la structure à clé multiple introduite par MKSE (b), et la structure avec préparation que nous proposons (c). xiv
6 Une illustration de notre structure à deux serveurs. xv
7 Une illustration du fonctionnement de RMÖ18 xix
8 Un graphe situant tous les protocoles BDCMU à la fin de cette thèse au regard du compromis entre sécurité et performance qu'ils offrent. xxi

1.1 A graph locating all MUSE protocols as of the end of this thesis regarding the privacy-efficiency trade-off they achieve. 7

4.1 An illustration of our attack example against the iterative testing structure 38
4.2 Recovery rate (percentage of queries recovered as a function of the fraction of the database initially known) of our query recovery attack against $\mathsf{L}_{\mathsf{KWAP}}$, compared to an attack against L1 in the same scenario. 42
4.3 Recovery rate of our keyword recovery attack against $\mathsf{L}_{\mathsf{KWAP}}$, compared to an attack against L3 in the same scenario. 43
4.4 The conversation `secret_conv` seen by user `good2` in the EncChat application, compared to the output of our attack program run by the adversary controlling the server and user `bad`. 45
4.5 Location of previous MUSE protocols regarding the privacy-versus-efficiency dilemma 49

5.1 An illustration that shows the difference between a single-key structure (a), a multi-key structure (b), and a structure with record preparation (c). 52
5.2 An illustration of our use of two non-colluding servers. 53
5.3 An illustration of the DH-AP-MUSE protocol using the notation of a PSI protocol. . 56
5.4 An illustration of the RMÖ15 protocol. 64
5.5 The UCN Reference Architecture 69

5.6 Location of protocols RMÖ15 and DH-AP-MUSE regarding the privacy-versus-efficiency dilemma 70

6.1 An illustration of the notion of response unlinkability 75
6.2 An illustration of PIR/OT recursiveness 80
6.3 An illustration of OT encoding 81
6.4 Location of protocol RMÖ18 and all other MUSE protocols regarding the privacy-versus-efficiency dilemma 92

7.1 Illustration of our counter-example. 97
7.2 An example of a more general counter-example involving 3 elements of $S - C$. . . 98

List of Tables

3.1 Main Leakage-Abuse Attacks 32

List of Algorithms

1 La structure algorithmique dite de « test itéré » x

2 The *iterative testing* algorithmic structure 37
3 An algorithm that computes the $\mathsf{L}_{\mathsf{KWAP}}$ leakage of a MUSE history 39

4 Simulator for the server view 59
5 Simulator for the proxy view 60

6 An algorithm for building a GBF representing set S with parameters $M, (h_i)_{i=1\dots\eta}, \lambda$ 77
7 An algorithm for building a ZGBF representing set S with parameters $M, (h_i)_{i=1\dots\eta}, \lambda$ 78
8 An algorithm for deciding whether some ZGBF buckets correspond to an element that is present or absent in a ZGBF 79