La cryptographie incrémentale : Techniques et mise en œuvre

Kevin Thiry-Atighehchi

La cryptographie incrémentale : Techniques et mise en œuvre

Révision mars 2019

Kevin Thiry-Atighehchi

GREYC, Université Caen Normandie, France

RÉSUMÉ. *La cryptographie incrémentale est un outil permettant d'obtenir des applications plus performantes et plus économes en ressources énergétiques (serveurs, appareils sur batterie). Elle se place dans le vaste contexte où des documents dont nous disposons des formes cryptographiques sont modifiés par des opérations d'édition telle que des insertions ou des suppressions de chaînes de caractères. Ré-appliquer l'algorithme de transformation cryptographique à ces documents chaque fois que leurs contenus changent peut être très long et particulièrement gourmand en ressources calculatoires. L'idée de la cryptographie incrémentale est donc de tirer profit des cas d'usages courants de la manipulation des documents non pas en recalculant une forme cryptographique à chaque modification, mais plutôt en la calculant comme une fonction rapide de la précédente forme cryptographique et de l'opération de modification par laquelle nous obtenons la version actuelle du document. Le but premier de cet article est de présenter un état de l'art de la cryptographie incrémentale, en fournissant des définitions, des propriétés intéressantes pour garantir le secret d'une modification et des exemples de systèmes. Enfin, nous discutons de leur mise en pratique.*

ABSTRACT. *The idea of Incremental Cryptography is to take advantage of document editing to allow the computation of an updated cryptographic form as a fast function of both the former and the modification by which we obtain the current version of a document. This type of cryptography is situated in a quite broad context within which we have documents undergoing frequent modifications of their content – for instance, through insertions and deletions of bytestrings. Re-applying from scratch a cryptographic transformation to a document whenever its content changes can be lengthy and particularly resource-consuming. Incremental Cryptography then allows most applications that are usually secured with standard cryptographic schemes to become more reactive and more energy-efficient. The primary purpose of this article is to present the state of the art of Incremental Cryptography, by providing definitions, interesting privacy properties with respect to modifications and examples of schemes. Finally, we discuss their practical applications.*

MOTS-CLÉS : *Cryptographie incrémentale, Cryptographie légère, Cryptographie symétrique, Cryptographie asymétrique.*

Technique et science informatiques – n° y/2017, 1-57

KEYWORDS: *Incremental cryptography, Lightweight cryptography, Symmetric cryptography, Asymmetric cryptography.*

DOI:10.3166/TSI.x.1-57 © 2017 Lavoisier

1. Introduction

Ce document a pour objet l'amélioration des performances des transformations cryptographiques, telles que le chiffrement d'un document, par l'utilisation de techniques de cryptographie incrémentale. Le résultat d'une telle transformation est une forme cryptographique, *e.g.* un chiffré ou une signature. L'objectif principal de la cryptographie incrémentale, introduite par Bellare, Goldreich et Goldwasser (1994 et 1995), est de tirer avantage de la connaissance d'une forme cryptographique d'un document pour calculer plus rapidement la transformation cryptographique d'un document différent mais apparenté. Pour des applications gérant constamment des changements dans des documents électroniques sécurisés, le développement de tels algorithmes est nécessaire afin de réduire les délais de traitement et les ressources calculatoires consommées. La cryptographie incrémentale est un vieux sujet qui peut refaire surface dans le contexte de la cryptographie légère. Supposons que nous ayons un document D qui a été modifié par une opération de mise à jour M pour donner un nouveau document D'. Ayant auparavant appliqué une transformation cryptographique $\mathcal{T}$ sur D pour produire une forme cryptographique $\mathcal{T}(D)$, un algorithme de mise à jour incrémental $\mathcal{I}$ est tel que $\mathcal{I}(D, \mathcal{T}(D), M)$ produit une forme cryptographique valide de D'. Idéalement, le temps d'exécution de $\mathcal{I}$ ne doit dépendre que du type de modification, du paramètre de sécurité, et de la quantité de données modifiées, et devrait être indépendant de la longueur du document. Par exemple, si nous notons k la taille d'un bloc, mettre à jour la forme cryptographique pour refléter le remplacement d'un seul bloc dans le document devrait s'exécuter en un temps polynomial en k indépendamment du nombre de blocs du document. Même si dans certains cas cela ne peut être accompli, différents types de complexité peuvent être intéressants. Par exemple, si nous notons n le nombre de blocs du document et si la transformation totale s'exécute en temps $\mathrm{O}(kn)$ alors une mise à jour incrémentale s'exécutant en temps $\mathrm{O}(k \log n)$ est une bonne chose. Si la transformation totale s'exécute en $\mathrm{O}(kn^2)$ et que la mise à jour s'exécute en $\mathrm{O}(kn)$, le gain est encore très appréciable. Enfin, si la différence de complexité doit être marquée, la recherche d'algorithmes incrémentaux ne doit pas non plus se faire au détriment de l'efficacité de l'algorithme de transformation qui, de préférence, doit rester comparable à celui d'un algorithme traditionnel.

En premier lieu, cet article présente une étude comparative des systèmes cryptographiques incrémentaux au niveau bloc (dont la taille est spécifiée), c'est-à-dire permettant d'effectuer au moins le remplacement d'un bloc, voire l'insertion et la suppression d'un bloc, pour ne citer que les principales opérations. Nous verrons comment de telles opérations augmentent les marges de manœuvre d'un attaquant pour falsifier des signatures. Une autre préoccupation, soulevée par Micciancio (1997), concerne la façon dont les mises à jour sont effectuées sur une forme cryptographique, car celles-

ci pourraient révéler comment le document correspondant a été obtenu par une suite d'opérations d'édition. Cette étude mettra donc aussi l'accent sur une propriété d'indépendance de l'historique, qui assure une forme de secret ou d'inconscience des opérations effectuées.

Dans la suite du papier, nous attaquons le problème important de la cryptographie incrémentale au niveau octet. Puisque les opérations d'édition d'un document consistent généralement à insérer ou supprimer des chaînes de caractères de longueur quelconque (et non un multiple de la taille d'un bloc), nous décrivons une méthode rendant possible ces opérations tout en préservant cette propriété d'inconscience. Le système proposé, initialement paru à la conférence AsiaCCS [1] (Atighehchi, Muntean, 2013), étend un système autorisant des opérations de suppression et d'insertion d'un bloc en un système incrémental au niveau octet continuant d'assurer de bonnes performances comparativement au système initial. De par cette propriété d'inconscience des opérations qu'elle assure, la méthode employée préserve le même surcoût moyen pour la taille de la forme cryptographique et le nombre d'opérations à effectuer lorsque nous appliquons la fonction de transformation (ou sa fonction inverse). Comparativement à l'analyse de complexité effectuée dans le papier original, les bornes ont ici été affinées.

Le présent document est organisé de la façon suivante :

– Une première partie se consacre à une étude comparative des systèmes cryptographiques incrémentaux. La section 2 aborde ses applications possibles et la section 3 fournit des définitions et des propriétés. Des systèmes sont présentés en section 4 et les techniques utilisées sont davantage discutées en section 5.

– La section 6 introduit une cryptographie incrémentale au niveau octet par l'extention d'un système autorisant un grand nombre d'opérations au niveau bloc, et assurant initialement l'inconscience des modifications effectuées, en un système autorisant les mêmes opérations au niveau octet et préservant cette propriété.

2. Applications

Imaginons qu'en même temps que nous écrivons une lettre, une signature numérique soit progressivement calculée par des suites de mises à jour incrémentales. Dans ce cas, cette signature pourrait être disponible dès que nous avons fini d'écrire la lettre. Nombreuses sont les applications qui peuvent profiter de la cryptographie incrémentale. Sans être exhaustif, en voici encore quelques-unes. De légères variations d'un même document, tel qu'un contrat type ou une offre standard, peuvent être envoyées par une société à différents destinataires. Si chacune des variations doit être signée, nous signons alors le document une seule fois avec un algorithme de signature incrémental et nous mettons rapidement à jour la signature pour refléter les différents noms

1. ACM Symposium on Information, Computer and Communications Security

des destinataires. L'envoi d'un flux [2] d'images horodatées et signées issues d'une caméra de vidéo-surveillance est un autre exemple où des données sont répétées avec de faibles variations. Dans le cadre de l'édition collaborative, des ébauches d'un même document peuvent être échangées entre différentes parties, où chaque ébauche, qui est chiffrée et/ou authentifiée, n'est que très légèrement différente de la précédente. Dans le même ordre d'idées nous pouvons citer l'édition à distance de textes ou programmes qui doivent être chiffrés et/ou authentifiés à chaque changement. La cryptographie incrémentale est également utile pour la maintenance d'un dictionnaire chiffré et/ou authentifié (plus particulièrement une base de données ou un système de fichiers (Blaze, 1993)) tout au long d'une multitude d'opérations de mises à jour durant laquelle des entrées isolées changent alors que la majeure partie des données reste identique.

Bellare, Goldreich et Goldwasser (1995) mettent en avant un bon exemple de la cryptographie incrémentale pour la protection contre les virus. Dans cet exemple, un processeur avec une mémoire locale inviolable accède à des fichiers stockés sur un média distant non sûr (un serveur Web ou une machine quelconque). En particulier, un virus peut inspecter et modifier ce contenu distant. Pour protéger les fichiers, le processeur calcule avec une clé stockée dans son environnement protégé une marque (tag) d'authentification pour chacun d'eux. Ces marques sont impossibles à reproduire par le virus sans la connaissance de cette clé et leurs vérifications par le processeur permet de détecter toute falsification. Comme les marques doivent être recalculés par le processeur chaque fois qu'il modifie ses fichiers, il est préférable, et c'est d'autant plus crucial que ces modifications sont fréquentes, qu'il soit capable de les mettre à jour plutôt que de les recalculer complètement. Pour mettre encore un peu plus l'accent sur la nécessité d'une cryptographie incrémentale, certains fichiers peuvent devenir trop volumineux pour qu'ils puissent être rapatriés dans l'espace mémoire local.

3. Travaux connexes

Les systèmes de signature expurgeable ou censurable, abrégés en anglais en RSS pour *Redactable Signature Scheme* (Pöhls, Samelin, 2015 ; Brzuska *et al.*, 2010) et SSS pour *Sanitizable Signature Scheme* (Ateniese *et al.*, 2005 ; Klonowski, Lauks, 2006), sont considérés hors-contexte et ne sont donc pas couverts par cette étude. Les schémas RSS permettent à un utilisateur détenteur d'un document signé de supprimer des parties du document et d'en dériver une signature valide, sans la présence du signataire. Les schémas SSS autorisent une tierce partie (le *sanitizer*) à éditer des parties du document et à en dériver une signature valide, à l'aide d'une trappe supplémentaire. Ces systèmes peuvent faire appel à des fonctions de hachage "caméléon" ou des accumulateurs à sens unique, des primitives qui auraient pu servir aux systèmes incrémentaux décrits ci-après si elles ne faisaient pas autant appel à des opérations arithmétiques coûteuses. Historiquement, l'objectif fondamental est différent de celui

2. Nous supposons un flux vidéo non compressé. Même lorsque ce flux exploite de la compression temporelle, des images clés successives peuvent n'être que très légèrement différentes.

de la cryptographie incrémentale. Il s'agit dans ce cas de pouvoir déléguer une capacité d'édition, alors que la cryptographie incrémentale n'a qu'un objectif d'efficacité comparativement à la cryptographie traditionnelle. Elle doit tirer parti du caractère évolutif des documents pour permettre au détenteur de la clé privée/secrète de mettre à jour rapidement des formes cryptographiques.

L'incrémentalité est également une caractéristique utile pour l'obfuscation de programmes (Garg, Pandey, 2015). L'obfuscation est une méthode pour brouiller le code d'un programme de sorte qu'il devienne inintelligible mais qu'il préserve la fonctionnalité du programme. Ré-obfusquer complètement un programme dès que nous modifions légèrement le code n'est pas très raisonnable, surtout lorsque les modifications sont fréquentes. Garg et Pandey ont donc proposé des méthodes pour mettre à jour l'obfuscation d'un programme efficacement, c'est-à-dire en un temps qui soit une fonction rapide de la quantité de code modifié. Puisqu'il s'agit d'une application particulière de la cryptographie, le système de Garg et Pandey n'a pas été intégré à cette étude.

4. Informations générales

Toute primitive cryptographique pourrait profiter d'une incrémentalité, c'est-à-dire de la possibilité d'effectuer des mises à jour. Dans la suite nous prenons comme exemples le chiffrement (à clé publique ou secrète) et la signature (à clé publique ou secrète). Nous pouvons également nous intéresser à l'incrémentalité d'une primitive de plus bas niveau, telle qu'une fonction de hachage, qui est alors bien utile pour concevoir un système de signature (ou de codes d'authentification) ayant cette propriété (Bellare *et al.*, 1994). Dans la pratique, de tels systèmes sont instanciés avec des fonctions pseudo-aléatoires, des permutations pseudo-aléatoires, ou des systèmes de signature (ou MAC), concrétisés par des algorithmes standardisés (par exemple SHA-3, AES, ECDSA).

Les algorithmes cryptographiques s'appliquent à des documents vus comme des suites de blocs, ou des chaînes sur un alphabet $\Sigma = \{0,1\}^s$ où s est le paramètre définissant la taille d'un bloc, qui peut dépendre du paramètre de sécurité. Nous notons alors P_i le i-ième bloc (ou symbole) du document D. Par la notation $D \in \sum^+$, nous spécifions que le document D est constitué d'un ou de plusieurs blocs. Les complexités des algorithmes sont estimées en termes d'opérations élémentaires sur cet alphabet et peuvent être traduites en opérations au niveau bit en les multipliant par un facteur polynomial en s.

Modifications de documents. Les mises à jour incrémentales sur une forme cryptographique doivent refléter les changements que nous effectuons dans les documents, tels que le remplacement, l'insertion ou la suppression de données. Dans un éditeur de texte où il nous arrive souvent de couper du texte pour le coller dans un autre, il semble approprié de considérer aussi des opérations de type couper et coller. Pour simplifier, nous supposons que tous les documents sont composés d'un nombre entier de blocs (ils peuvent être bourrés avec des techniques standard). Pour

reprendre les notations de (Buonanno *et al.*, 2002), nous notons M une opération de modification et $D\langle M\rangle$ le document qui résulte de l'opération M effectuée sur D. Nous définissons une suite d'opérations de modification sur un document D par $D\langle M_1, M_2, \ldots, M_j\rangle = (\ldots((D\langle M_1\rangle)\langle M_2\rangle)\ldots)\langle M_j\rangle$ et nous notons $\mathcal{M}$ l'espace des modifications possibles. Sans être exhaustif, cet espace peut être constitué des types d'opérations de modification suivants :

– $M = (\mathsf{DELETE}, i)$ qui supprime le i-ième bloc du document.

– $M = (\mathsf{INSERT}, i, P^*)$ qui insère le bloc P^* entre le i-ième et le $(i+1)$-ième bloc du document.

– $M = (\mathsf{REPLACE}, i, P^*)$ qui remplace le i-ième bloc du document par le bloc P^*.

La position d'une opération de modification M est l'indice de bloc qui est modifié ou supprimé, ou bien l'indice de bloc après lequel est effectué une insertion, selon le cas. Lorsqu'une modification est apportée à un document, nous supposons qu'elle est toujours cohérente, à savoir qu'elle se réfère à des blocs existants dans le document. Comme contre-exemple, une opération $(\mathsf{REPLACE}, 6, P)$ appliquée à un document de trois blocs est une mofidication incohérente.

Nous pouvons également définir des opérations qui opèrent sur des chaînes de blocs, plutôt que sur un seul bloc. De telles opérations sont par exemple DELETE-SUBTEXT qui supprime une chaîne consécutive de blocs dans le texte, ou encore INSERT-SUBTEXT qui insère une suite consécutive de blocs à un endroit donné dans le texte, ou bien encore MOVE-SUBTEXT qui déplace une suite de blocs d'un endroit à un autre dans le texte.

Remarquons que les quelques opérations que nous venons de définir produisent un seul texte, alors qu'une opération de type "couper" en produit normalement deux. Par conséquent, une telle mise à jour appliquée à une forme cryptographique doit produire deux formes cryptographiques. De la même manière, une mise à jour incrémentale reflétant l'opération "coller" prend deux formes cryptographiques et en produit une seule.

Comme dit précédemment l'incrémentalité peut s'appliquer à un grand nombre de primitives. À la manière de Bellare, Goldreich et Goldwasser (1995), la définition suivante essaie de capturer un grand ensemble de primitives usuelles, bien qu'elle puisse exceptionnellement échouer pour certaines.

DÉFINITION 1. — *Un système cryptographique incrémental est spécifié par un 4-uplet* $\Pi = (\mathcal{G}, \mathcal{T}, \mathcal{C}, \mathcal{I})$ *d'algorithmes en temps polynomial :*

– $\mathcal{G}$ *est un algorithme probabiliste de génération de clés. Il prend en entrée un paramètre de sécurité* k *(en unaire) et retourne une paire de clés* (K', K'') *où* K' *est la clé de transformation et* K'' *la clé conjuguée.*

– $\mathcal{T}$ *est un algorithme probabiliste de transformation. Il prend en entrées une clé* K' *et un document* $D \in \sum^+$ *et retourne une forme cryptographique* $t = \mathcal{T}_{K'}(D)$.

– $\mathcal{C}$ est un algorithme déterministe effectuant l'opération de transformation inverse (conjuguée) de $\mathcal{T}$. Il prend en entrées la clé conjuguée K'' et la forme cryptographique t et retourne D ou une erreur $\perp$.

– $\mathcal{I}$ est un algorithme probabiliste effectuant la mise à jour incrémentale. Il prend en entrées la clé K', (le document D,) la forme cryptographique t (associée à D), une opération de mise à jour $M \in \mathcal{M}$ et retourne la forme cryptographique mise à jour t'.

Pour toute paire de clés (K', K'') retournée par $\mathcal{G}$ et pour tout document D, nous avons $\mathcal{C}_{K''}(\mathcal{T}_{K'}(D)) = D$. De plus, pour toute paire de clés (K', K'') retournée par $\mathcal{G}$, pour tout document D et pour toute modification $M \in \mathcal{M}$, nous avons $\mathcal{C}_{K''}(\mathcal{I}_{K'}(D, \mathcal{T}_{K'}(D), M)) = D\langle M \rangle$. Un système incrémental peut être illustré avec le diagramme commutatif de la figure 1.

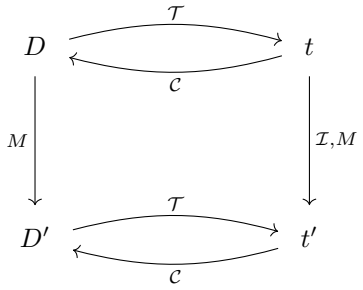


Figure 1. Schéma cryptographique incrémental

Lorsque nous nous plaçons dans un modèle de sécurité classique, nous supposons que les entrées D, t, M de $\mathcal{I}_{K'}$ sont cohérentes, à savoir que $\mathcal{C}_{K''}(t) = D$. Tout autre modèle de sécurité n'adoptant pas cette hypothèse est discuté à la section 4.2.1.

Si $K' = K''$, le système est dit symétrique et peut correspondre soit à un système de chiffrement à clé secrète ou bien à un système d'authentification de message (MAC). Si $K' \neq K''$, le système est dit asymétrique et peut correspondre à un système de chiffrement à clé publique ou bien à un système de signature électronique. Selon le contexte, nous utiliserons la notation $\mathcal{E}$ (*resp.* $\mathcal{S}$) au lieu de $\mathcal{T}$ pour désigner un chiffrement (*resp.* une signature), et $\mathcal{D}$ (*resp.* $\mathcal{V}$) au lieu de $\mathcal{C}$ pour désigner un déchiffrement (*resp.* une vérification de signature). Enfin, le symbole d'erreur $\perp$ retourné par $\mathcal{C}$ ne se distingue des autres sorties possibles que si le système fournit de l'authentification.

REMARQUE 2. — Cette définition s'éloigne de la spécification d'une fonction de hachage incrémentale (Bellare, Micciancio, 1997), pour laquelle les fonctions $\mathcal{C}$ et $\mathcal{G}$ n'existent pas, et la clé $K' = K''$ n'existe pas nécessairement. Cette définition ne couvre pas non plus les systèmes de chiffrement déterministes, comme celui proposé par Mironov *et al.* (Mironov *et al.*, 2012). Pour ce type de cryptosystèmes, les algorithmes $\mathcal{T}$ et $\mathcal{I}$ doivent être déterministes.

REMARQUE 3. — Nous supposerons que les schémas de signature numérique sont à recouvrement de message, ou bien que le document D est inclus dans la signature t,

de sorte que nous puissions fonctionnellement avoir $\mathcal{V}_{K''}(t) = D$. Nous parlons dans le deuxième cas de signature avec appendice.

4.1. Les approches triviales

Pour simplifier les choses, nous supposons qu'une modification d'un document D consiste à remplacer un de ses blocs de données. Notons σ la signature incrémentale obtenue sur un document D en utilisant un algorithme employant comme brique de base un système *non incrémental* de signature numérique noté S (la clé est implicite dans la notation). Le document est modifié en $D' = f(D)$ où f est une opération d'édition. Pour signer le document D', il suffit de calculer

$$s' = S(\sigma \| f)$$

et de retourner le 4-uplet (σ, s', f, f^{-1}). La signature incrémentale du nouveau document D' est alors

$$\sigma' = (\sigma, s', f, f^{-1}). \tag{1}$$

Pour vérifier qu'il s'agit d'une signature valide de D', nous vérifions que s' est une signature (non incrémentale) valide de $\sigma \| f$, que σ est une signature incrémentale valide de $f^{-1}(D')$ et que $D' = f(f^{-1}(D'))$.

Maintenant, soit $D'' = g(D')$ pour une opération d'édition g, et

$$s'' = S(\sigma' \| g) = S((\sigma, s', f, f^{-1}) \| g).$$

La signature incrémentale de D'' est alors le 4-uplet σ'' suivant :

$$\sigma'' = (\sigma', s'', g, g^{-1}) = ((\sigma, s', f, f^{-1}), s'', g, g^{-1}). \tag{2}$$

En supposant que le temps d'exécution de l'algorithme de signature S soit proportionnel à la taille de l'entrée, alors le temps pour obtenir la nouvelle signature s' est proportionnel à la quantité de modifications f effectuées sur D et non à la taille de D'. En ce sens l'algorithme défini ci-dessus est incrémental. Par contre, il ressort des équations (1) et (2) que la gestion de l'expansion de la forme cryptographique va être délicate.

Nous pouvons opérer de manière similaire pour le chiffrement. Notons E un algorithme de chiffrement standard. Connaissant le chiffrement $E(D)$ de D, un chiffrement de D' pourrait être $E(D) \| E(f)$. Il s'agit bien d'un chiffrement incrémental puisque la mise à jour se fait en temps proportionnel à la quantité de modifications. Un chiffré peut être mis à jour de la sorte autant de fois qu'il y a des mises à jour à effectuer sur le clair correspondant.

Problèmes d'efficacité. Sur les algorithmes décrits ci-dessus, le coût de l'algorithme conjugué, qu'il s'agisse d'une vérification de signature ou d'un déchiffrement, est proportionnel au nombre de mises à jour effectuées, ce qui est clairement un aspect

indésirable. De plus, la taille de la forme cryptographique augmente à chaque mise à jour. Une manière de pallier à ce problème est de re-signer (resp. re-chiffrer) entièrement le document lorsque la quantité de modifications atteint un certain seuil, par exemple lorsqu'elle atteint la taille du document. Cette solution permet d'avoir d'une part une expansion réduite de la taille de la forme cryptographique et d'autre part des mises à jour en temps efficaces, le tout de façon amortie. Si de telles solutions peuvent être satisfaisantes, une complexité non amortie serait plus appréciable. Dans le même ordre d'idée, qu'une forme cryptographique t soit issue de la transformation initiale $\mathcal{T}$ ou de la mise à jour incrémentale $\mathcal{I}$, la complexité de la transformation conjuguée devrait être la même.

4.2. Propriétés de sécurité

Lorsque nous concevons un système cryptographique, qu'il soit incrémental ou non, il doit être prouvé sûr dans un modèle de sécurité donné. Nous revoyons donc les notions d'infalsifiabilité et d'indistinguabilité adaptées au cas de l'incrémentalité. Puisqu'il est possible de construire un système d'authentification incrémental en s'appuyant sur une fonction de hachage incrémentale [3], nous revoyons également les propriétés importantes à préserver pour une telle primitive. Les notions présentées essaient de capturer le plus grand nombre de primitives. Celles-ci doivent donc être remaniées au cas par cas, selon la finalité du système cryptographique.

4.2.1. Infalsifiabilité

L'exemple d'application d'une signature ou d'un code d'authentification incrémental pour la protection contre les virus suggère un attaquant capable d'altérer le contenu du média distant. Si pour un système de signature (ou de code d'authentification) traditionnel, le modèle de sécurité classique permettait à l'attaquant de se faire signer des messages de son choix, dans un contexte de cryptographie incrémentale il est légitime de lui donner également droit à des opérations de mises à jour incrémentales. Avec ces nouvelles opérations il ne faut donc pas écarter la possibilité pour l'attaquant d'altérer un couple (document, signature) avant qu'une opération de mise à jour ne soit effectuée dessus. Rappelons que l'objectif de l'attaquant est de produire une signature pour un message qui n'a jamais été vu auparavant, ou plus précisément, dont la signature n'a jamais été produite par l'algorithme de signature ou l'algorithme de mise à jour incrémentale sur un couple (document, signature) valide.

Le modèle de sécurité doit prendre en compte les *attaques par altérations* (Bellare *et al.*, 1994, pages 14, 15 ; Micciancio, 1997, pages 12, 13 ; Fischlin, 1997a, pages 5, 6). Dans ce modèle, l'attaquant (un virus) peut altérer un document avant de faire une requête de mise à jour incrémentale. S'il n'altère pas les documents, il

3. Par exemple, une signature sur le résultat d'une fonction de hachage incrémentale produit une signature incrémentale (Bellare *et al.*, 1994). De la même façon dans le cas symétrique, utiliser l'algorithme HMAC (Bellare *et al.*, 1996) avec une fonction de hachage incrémentale produit un système d'authentification de message incrémental.

s'agit d'une *attaque classique* (notée AA0). S'il altère seulement les documents mais pas les signatures, il s'agit d'une *attaque par substitution de message* (notée AA1). S'il altère les documents et les signatures, il s'agit d'une *attaque par substitution totale* (notée AA2). Afin de permettre à un attaquant de tirer profit de l'accès à la fonction de mise à jour $\mathcal{I}$, nous devons donc étendre la notion d'infalsifiabilité existentielle contre des attaques à documents choisis (EUF-CMA pour Existential Unforgeability under Chosen Message Attack) en autorisant des attaques par altérations (AA pour Alteration Attack). Cette notion se définit alors par une expérience aléatoire $\mathrm{Exp}_{\Pi}^{\text{EUF-CMA-AA}x}(A, k)$ entre un challenger et un attaquant A :

$\mathbf{Exp}_{\mathbf{\Pi}}^{\mathbf{EUF\text{-}CMA\text{-}AA}x}(\boldsymbol{A}, \boldsymbol{k})$
$(K', K'') \leftarrow \mathcal{G}(k)$
$(D^*, t^*) \leftarrow A^{\mathcal{S}_{K'}(.), \mathcal{I}_{K'}(.,...)}(k, K'')$
If $D^* \in$ List, then return 0
If $\mathcal{V}_{K''}(D^*, t^*) = D^*$, then return 1
Return 0

où $\mathcal{S}_{K'}(.)$ et $\mathcal{I}_{K'}(.,\ldots)$ sont respectivement les oracles de signature et de mise à jour incrémentale, et List une liste initialement vide dans laquelle sont enregistrés tous les documents signés par l'oracle de signature et ceux (en apparence) signés par l'oracle de mise à jour. Le remplissage de cette liste lors des requêtes à $\mathcal{I}_{K'}(.,\ldots)$ est explicité ci-après. Remarquons que dans le cas symétrique l'expérience change légèrement, car le challenger ne fournit pas la clé K''=K' à l'attaquant. Ce dernier se voit alors autorisé un accès à l'oracle de vérification $\mathcal{V}_{K''}(.,.)$.

Notons que List est nécessairement non vide lors de la première requête à l'oracle de mise à jour. Nous détaillons maintenant d'une part les contraintes implicites sur l'accès à $\mathcal{I}_{K'}(.,\ldots)$, et d'autre part nous levons l'ambiguïté sur le document qui est inclus dans List lors d'une requête de ce type. Nous avons trois cas de figure :

– L'attaque CMA-AA0 impose la contrainte la plus forte sur l'accès à cet oracle : son entrée (D, t) est un couple document-signature qui est toujours valide (sans altération). L'attaquant invoque donc l'oracle $\mathcal{I}_{K'}$ sur les entrées D, t et une opération d'édition $M \in \mathcal{M}$, et se voit retourner une signature t' de $D' = D\langle M\rangle$. Le nouveau document D' est alors inclus dans List.

– Concernant l'attaque CMA-AA1, cette contrainte est relachée car un document $D \in$ List peut être altéré en D_{alt}. L'attaquant invoque l'oracle $\mathcal{I}_{K'}$ sur les entrées D_{alt}, t et une opération d'édition M, et se voit retourner une signature t''. C'est le document $D' = D\langle M\rangle$ qui est inclus dans List.

– Enfin, dans le cas de CMA-AA2, l'attaquant peut altérer les deux composantes d'un couple (D, t) en (D_{alt}, t_{alt}), avec $D \in$ List. L'attaquant invoque l'oracle $\mathcal{I}_{K'}$ sur les entrées D_{alt}, t_{alt} et une opération d'édition M, et se voit retourner une signature t''. C'est le document $D' = D\langle M\rangle$ qui est inclus dans List.

Au regard de la notion classique EUF-CMA, dans les deux derniers cas List ne contient plus les documents qui ont effectivement été signés par le signataire, mais plutôt les documents qu'il croit avoir signés (Bellare *et al.*, 1994). Les attaques par altération sont utiles pour un attaquant car si le système de signature incrémental n'a pas été conçu pour s'en protéger, l'attaquant peut profiter du comportement non prévu de $\mathcal{I}_k$ pour des entrées invalides (*i.e.* des entrées D, t telles que $\mathcal{V}_{K''}(t) \neq D$)).

DÉFINITION 4. — *Soit un système de signature incrémental* $\Pi = (\mathcal{G}, \mathcal{S}, \mathcal{V}, \mathcal{I})$ *sur un espace de modification* $\mathcal{M}$ *et* A *un attaquant pour une attaque* CMA-AAx *avec* $x \in \{0, 1, 2\}$. *Nous définissons l'avantage de l'attaquant* $Adv_{\Pi}^{\text{EUF-CMA-AA}x}$ *par :*

$$Adv_{\Pi}^{\text{EUF-CMA-AA}x}(A, k) = \Pr\left(\text{Exp}_{\Pi}^{\text{EUF-CMA-AA}x}(A, k) = 1\right).$$

Le système de signature incrémental est $(t, q_s, q_i, q_v, \mu, \epsilon)$*-sûr au sens de l'infalsifiabilité existentielle contre une attaque* CMA-AAx *si, pour n'importe quel attaquant* A *qui s'exécute en temps* t*, en faisant* q_s *requêtes à l'oracle de signature* $\mathcal{S}_{K'}(.)$*,* q_i *requêtes à l'oracle de mise à jour incrémentale* $\mathcal{I}_{K'}(.,.,.)$ *et* q_v *requêtes à l'oracle de vérification*[4] *de signature* $\mathcal{V}_{K''}(.,.)$ *(*μ *totalisant le nombre de blocs signés par* $\mathcal{S}_{K'}$ *ou* $\mathcal{I}_{K'}$*),* $Adv_{\Pi}^{\text{EUF-CMA-AA}x}(A, k)$ *est inférieur*[5] *à* ϵ.

Lorsque nous concevons un système de signature incrémental (qu'il s'agisse de signatures électroniques ou de codes d'authentification), nous pouvons nous placer dans différents modèles de sécurité, ce qui peut compliquer la tâche suivant le cas et mener à des efficacités algorithmiques différentes. Une question que nous pouvons éventuellement nous poser est, en effectuant peu de changements aux algorithmes sous-jacents, un système résistant aux attaques classiques peut-il résister à des attaques par substitution ? Un tel système peut effectivement s'assurer de la validité d'un couple (message, signature) avant d'y effectuer une mise à jour incrémentale. Pour autant, le système obtenu ne reste pas "efficacement" incrémental puisqu'une telle approche nécessite une vérification totale de la signature (ou du tag) à chaque mise à jour.

REMARQUE 5. — *Résistance aux collisions.* L'incrémentalité ne nécessite aucune notion additionnelle aux propriétés usuelles des fonctions de hachage. C'est lorsque nous utilisons cette fonction de hachage dans une signature incrémentale que des précautions doivent être prises. Se référer ci-dessus aux notions d'infalsifiabilité.

4.2.2. Indistinguabilité

Pour un système de chiffrement incrémental, il faut supposer qu'un attaquant puisse non seulement avoir accès à des opérations de chiffrement mais également à des opérations de mise à jour incrémentale, ces dernières étant une source d'informations supplémentaires pour casser la sécurité sémantique du chiffrement. Une

4. S'il s'agit d'un système à clé publique, l'attaquant dispose de la clé publique. L'oracle de vérification n'est donc plus nécessaire.

5. La notation ϵ signifie que la probabilité de réussite est une fonction négligeable du paramètre de sécurité k. Dans la suite nous continuons d'utiliser cette notation.

modification doit également rester privée, signifiant qu'il doit être impossible de déterminer le contenu du symbole modifié. Concrètement, cela signifie que si l'on inverse un bit d'un symbole du document, la confidentialité de ce nouveau bit doit être préservée. Pour prouver la sécurité sémantique, le modèle de sécurité pour un chiffrement est formalisé en utilisant la notion d'indistinguabilité *find-then-guess*, étendue (Buonanno *et al.*, 2002) pour assurer l'indistinguabilité d'une modification. L'attaquant a donc le droit d'interagir avec un oracle de chiffrement [6] $\mathcal{E}_{K'}(.)$ et avec un oracle de mise à jour incrémentale $\mathcal{I}_{K'}(.,.,.)$. Ensuite l'attaquant choisit soit deux documents (D_0, D_1) ayant le même nombre de blocs l, soit un chiffré t avec deux opérations de modification (M_0, M_1) (qui doivent être du même type et modifiant la même position). Un bit b est choisi aléatoirement et n'est pas divulgué à l'attaquant. Dans le premier cas $\mathcal{E}_{K'}$ est appliqué à D_b alors que dans le deuxième cas $\mathcal{I}_{K'}$ est appliqué à M_b et t. Dans chacun des cas le résultat est donné à l'attaquant qui peut alors continuer à interagir avec $\mathcal{E}_{K'}(.)$ et $\mathcal{I}_{K'}(.,.,.)$. Nous disons que l'attaquant réussit s'il devine correctement b dans les deux cas et nous définissons l'avantage de l'attaquant comme étant la probabilité de succès moins $1/2$.

Soit $A = (A_1, A_2)$ un attaquant ayant accès aux oracles $\mathcal{E}_K(.)$ et $\mathcal{I}_K(.,.,.)$. Cette expérience aléatoire entre le challenger et l'attaquant peut être divisée en deux sous expériences, que l'on note $\mathbf{Exp}_{\mathbf{\Pi}}^{\mathbf{IND1\text{-}CPA}}(\mathbf{A}, \mathbf{k})$ et $\mathbf{Exp}_{\mathbf{\Pi}}^{\mathbf{IND2\text{-}CPA}}(\mathbf{A}, \mathbf{k})$. La première sous expérience a pour objectif de sécurité l'indistinguabilité des chiffrés (IND1) :

$\mathbf{Exp}_{\mathbf{\Pi}}^{\mathbf{IND1\text{-}CPA}}(\mathbf{A}, \mathbf{k})$
  $(K', K'') \leftarrow \mathcal{G}(k)$
  $(D_0, D_1, s) \leftarrow A_1^{\mathcal{E}_{K'}(.),\mathcal{I}_{K'}(.,.,.)}(k)$
(D_0 *et* D_1 *sont distincts et de même longueur*)
  $b_0 \leftarrow \{0, 1\}$
  $t \leftarrow \mathcal{E}_{K'}(D_{b_0})$
  $b \leftarrow A_2^{\mathcal{E}_{K'}(.),\mathcal{I}_{K'}(.,.,.)}(k, s, t)$
  if $b = b_0$ then return 1 else return 0

La deuxième sous expérience a pour objectif l'indistinguabilité des contenus modifiés (IND2) :

6. Notons que dans le cas d'un système à clé publique, l'oracle de chiffrement est inutile car l'attaquant dispose de la clé de chiffrement K'.

$\mathbf{Exp}_{\mathbf{\Pi}}^{\mathbf{IND2\text{-}CPA}}(\mathbf{A}, \mathbf{k})$
$(K', K'') \leftarrow \mathcal{G}(k)$
$(t, M_0, M_1, s) \leftarrow A_1^{\mathcal{E}_{K'}(.),\mathcal{I}_{K'}(.,.,.)}(k)$
(M_0 *and* M_1 *sont du même type*
et modifient la même portion du chiffré)
$b_0 \leftarrow \{0, 1\}$
$t' \leftarrow \mathcal{I}_{K'}(D, t, M_{b_0})$
$b \leftarrow A_2^{\mathcal{E}_{K'}(.),\mathcal{I}_{K'}(.,.,.)}(k, s, t')$
if $b = b_0$ then return 1 else return 0

Pour les deux expériences et dans le cas asymétrique, l'algorithme A_1 prend en entrée la clé de chiffrement K'.

DÉFINITION 6. — *Soit un système de chiffrement incrémental* $\Pi = (\mathcal{G}, \mathcal{E}, \mathcal{D}, \mathcal{I})$ *sur un espace de modification* $\mathcal{M}$ *et* A *un attaquant pour une attaque* IND1-CPA. *Nous définissons l'avantage de l'attaquant* $Adv_{\Pi}^{\text{IND1-CPA}}$ *par :*

$$Adv_{\Pi}^{\text{IND1-CPA}}(A, k) = \left| \Pr\left(\text{Exp}_{\Pi}^{\text{IND1-CPA}}(A, k) = 1\right) - \frac{1}{2} \right|.$$

Le système de chiffrement incrémental est $(t, q_e, q_i, \mu, l, \epsilon)$*-sûr au sens de l'indistinguabilité des chiffrés contre une attaque CPA si, pour n'importe quel attaquant* A *qui s'exécute en temps* t*, en faisant* q_e *requêtes à l'oracle de chiffrement* $\mathcal{E}_{K'}(.)$ *et* q_i *requêtes à l'oracle de mise à jour incrémentale* $\mathcal{I}_{K'}(.)$ *(*μ *totalisant le nombre de blocs chiffrés par* $\mathcal{E}_{K'}$ *ou* $\mathcal{I}_{K'}$*),* $Adv_{\Pi}^{\text{IND1-CPA}}(A, k)$ *est inférieur à* ϵ*.*

DÉFINITION 7. — *Soit un système de chiffrement incrémental* $\Pi = (\mathcal{G}, \mathcal{E}, \mathcal{D}, \mathcal{I})$ *sur un espace de modification* $\mathcal{M}$ *et* A *un attaquant pour une attaque* IND2-CPA. *Nous définissons l'avantage de l'attaquant* $Adv_{\Pi}^{\text{IND2-CPA}}$ *par :*

$$Adv_{\Pi}^{\text{IND2-CPA}}(A, k) = \left| \Pr\left(\text{Exp}_{\Pi}^{\text{IND2-CPA}}(A, k) = 1\right) - \frac{1}{2} \right|.$$

Le système de chiffrement incrémental est $(t, q_e, q_i, \mu, l, \epsilon)$*-sûr au sens de l'indistinguabilité des modifications contre une attaque CPA si, pour n'importe quel attaquant* A *qui s'exécute en temps* t*, en faisant* q_e *requêtes à l'oracle de chiffrement* $\mathcal{E}_{K'}(.)$ *et* q_i *requêtes à l'oracle de mise à jour incrémentale* $\mathcal{I}_{K'}(.)$ *(*μ *totalisant le nombre de blocs chiffrés par* $\mathcal{E}_{K'}$ *ou* $\mathcal{I}_{K'}$*),* $Adv_{\Pi}^{\text{IND2-CPA}}(A, k)$ *est inférieur à* ϵ*.*

REMARQUE 8. — Un système de chiffrement à clé publique traditionnel s'applique normalement à des messages de petite taille (par exemple une clé), les messages chiffrés ont donc tous le même coût unitaire (par exemple celui d'appliquer une exponentiation modulaire). Un système de chiffrement à clé publique incrémental n'a du sens que pour de longs messages et fait naturellement appel plusieurs fois à la même primitive (par exemple une fonction algébrique ou issu de la théorie des nombres) pour chiffrer plusieurs blocs du message (Mironov *et al.*, 2012). Par conséquent les paramètres μ et l ont du sens dans les deux cas.

REMARQUE 9. — S'il s'agit d'un système de chiffrement authentifié, la propriété d'infalsifiabilité doit également être satisfaite.

4.2.3. Non-malléabilité

Un système de chiffrement est dit malléable s'il est possible, sans connaître la clé, de transformer un chiffré d'un document D en un chiffré valide d'un document différent mais apparenté $D' = f(D)$, pour une fonction f connue. Reprenons maintenant le système de chiffrement incrémental de la section 4.1. Si la brique de base E produit des chiffrés non malléables, ce système ne préserve pas cette non-malléabilité. Supposons qu'un attaquant soit en possession des chiffrés $E(D_1)$ et $E(D_2)$ de deux documents D_1 et D_2, distincts et de même longueur. Il récupère également les chiffrés de leurs mises à jour, $E(D_1)\|E(f_1)$ et $E(D_2)\|E(f_2)$, selon des modifications f_1 et f_2 elles aussi distinctes. L'attaquant peut alors produire les nouveaux chiffrés $E(D_1)\|E(f_2)$ et $E(D_2)\|E(f_1)$ correspondants aux nouveaux documents $f_2(D_1)$ et $f_1(D_2)$. Étonnamment, malgré l'attention qui a été portée aux attaques par substitutions dans le cadre des mises à jour de signature, la propriété de *non-malléabilité* n'a pas été abordée en cryptographie incrémentale. Il est possible de se prémunir de ce type d'attaques par l'emploi d'un mécanisme assurant une résistance forte contre les falsifications [7], même si le chiffrement initial E est malléable. Par exemple, en cryptographie symétrique, une marque d'intégrité peut être calculée sur le chiffré selon le paradigme de composition *Encrypt-then-MAC* (Bellare, Namprempre, 2000).

4.2.4. Indépendance de l'historique (ou inconscience des modifications)

Nous avons vu à la section 4.1 une manière d'obtenir des mises à jour incrémentales efficaces en contrepartie de quelques désagréments, comme l'augmentation du temps d'exécution de l'algorithme conjugué. De telles solutions révèlent l'historique des modifications. Si une telle propriété est la bienvenue dans des applications de gestion de versions, il existe des situations où ça ne l'est clairement pas. Supposons qu'une personne utilise un algorithme de signature incrémental pour produire des signatures d'engagements apparentés pris auprès de différentes parties. Une telle signature ne doit pas révéler d'informations concernant les engagements pris auprès des autres parties. En fait, une telle signature ne devrait déjà pas révéler qu'elle a été issue d'une opération de mise à jour. Autrement dit, la modification devrait être transparente.

Cette propriété de *perfect privacy* [8] ou d'*inconscience* (Bellare, Goldreich, Goldwasser, 1995 ; Micciancio, 1997 ; Buonanno *et al.*, 2002) peut se décrire comme suit.

---

7. Nous faisons référence, en cryptographie conventionnelle, au modèle de sécurité SUF-CMA (pour Strong Existential Unforgeability under Chosen Message Attack). Il correspond au modèle EUF-CMA dans lequel l'attaquant est autorisé à retourner une nouvelle signature pour un document dont une signature lui a été fournie.

8. Cette propriété sous le nom de *transparence* se retrouve également du côté des systèmes de signatures "censurables" ou "rectifiables" (*redactable signatures* et *sanitizable signatures* (Ateniese *et al.*, 2005 ; Canard *et al.*, 2008 ; Brzuska *et al.*, 2010)).

Supposons que nous ayons un document D' avec sa forme cryptographique mise à jour t' et que D' ait été obtenu par une opération d'édition de D. Il devrait être infaisable pour une personne recevant le couple (D', t') de distinguer si t' a été obtenu en exécutant l'algorithme de transformation $\mathcal{T}$ ou s'il a été obtenu en exécutant l'algorithme de mise à jour incrémental $\mathcal{I}$. Autrement dit, la distribution des formes cryptographiques produites devrait être indépendante du nombre d'opérations d'édition effectuées et de leur nature (e.g. positions et quantités de données supprimées, positions et quantités de données insérées). Cette propriété peut être définie grâce à l'expérience suivante entre un challenger et un attaquant A pour une attaque à documents choisis (CDA, en référence à une attaque CPA pour un schéma de chiffrement, et CMA pour un schéma de signature). L'attaquant A interagit avec son oracle de transformation $\mathcal{T}_{K'}(.)$ et son oracle de mise à jour $\mathcal{I}_{K'}(.,.,.)$, et enfin soumet au challenger un document D avec une modification $M \in \mathcal{M}$. Le challenger choisit aléatoirement et uniformément un bit $b_0 \in \{0,1\}$. Si $b_0 = 0$ alors le challenger retourne $\mathcal{T}_{K'}(D\langle M\rangle)$, sinon il retourne $\mathcal{I}_{K'}(M, D, \mathcal{T}_{K'}(D))$. Le résultat est donné à A qui peut ensuite faire davantage de requêtes aux oracles. Finalement, A doit retourner une hypothèse b pour la valeur de b_0. Nous considérons que A gagne si $b_0 = b$ et nous disons que le système est *transparent* si des attaquants raisonnables ne peuvent pas gagner significativement plus de la moitié du temps. Nous noterons par $\mathbf{Exp}_{\mathbf{\Pi}}^{\mathbf{Priv\text{-}CDA}}(\mathbf{A}, \mathbf{k})$ l'expérience correspondante qui retourne 1 si $b_0 = b$ et 0 sinon.

$\mathbf{Exp}_{\mathbf{\Pi}}^{\mathbf{Priv\text{-}CDA}}(\boldsymbol{A}, \boldsymbol{k})$
$(K', K'') \leftarrow \mathcal{G}(k)$
$(D, M) \leftarrow A^{\mathcal{T}_{K'}(.),\mathcal{I}_{K'}(.,...)}(k)$
$b_0 \leftarrow \{0,1\}$
If $b_0 = 0$ then
  $t \leftarrow \mathcal{T}_{K'}(D\langle M\rangle)$
Else
  $t \leftarrow \mathcal{I}_{K'}(M, D, \mathcal{T}_{K'}(D))$
$b \leftarrow A_2^{\mathcal{T}_{K'}(.),\mathcal{I}_{K'}(.,.,.)}(k, s, t)$
if $b = b_0$ then Return 1 else Return 0

DÉFINITION 10. — *Soit $\Pi = (\mathcal{G}, \mathcal{T}, \mathcal{C}, \mathcal{I})$ un système cryptographique incrémental sur l'espace des modifications $\mathcal{M}$, et soit A un attaquant pour une attaque* CDA. *Nous définissons l'avantage de l'attaquant $Adv_{\Pi}^{\text{Priv-CDA}}$ par :*

$$Adv_{\Pi}^{\text{Priv-CDA}}(A, k) = \left| \Pr\left(\text{Exp}_{\Pi}^{\text{Priv-CDA}}(A, k) = 1\right) - \frac{1}{2} \right|.$$

Nous disons que Π est $(t, q_T, q_i, \mu, \epsilon)$-sûr au sens de Priv-CDA *si, pour tout attaquant A qui s'exécute en temps t, en faisant q_T requêtes à $\mathcal{T}_{K'}(.)$ et q_i requêtes valides à $\mathcal{I}_{K'}(.,.,.)$ (avec le nombre total de symboles transformés dans toutes les requêtes de transformations et de mises à jour incrémentales égal à μ), $Adv_{\Pi}^{\text{Priv-CDA}}(A, k)$ est inférieur à ϵ. Si $Adv_{\Pi}^{\text{Priv-CDA}}(A, k) = 0$, nous disons que Π est parfaitement transparent.*

Satisfaire une telle propriété a des conséquences intéressantes sur les critères de performances du système incrémental. Si une modification est parfaitement transparente, cela signifie que la taille d'une forme cryptographique, qu'elle soit retournée par $\mathcal{T}$ ou par $\mathcal{I}$, est une seule et même fonction de la taille actuelle du document.

4.2.5. Autres propriétés

La propriété précédente peut être relâchée lorsque nous n'en avons pas besoin. Supposons qu'Alice remplisse les champs d'un formulaire type et signe ce formulaire à destination de Bob. Il peut s'agir d'un cas où Alice n'a que faire du fait que Bob apprenne que la signature a été obtenue en incrémentant une précédente signature, du moment qu'aucune information ne puisse être déduite concernant les champs des instances précédentes du formulaire signé. Il est possible de n'assurer qu'une propriété dite de *partial privacy* dont la signification peut varier. Il peut être acceptable de ne cacher que la position où un symbole a été inséré ou supprimé, ou bien de ne cacher que la quantité d'informations qui a changée.

Ces propriétés sont utiles du point de vue d'un utilisateur légitime. Du point de vue d'un attaquant, un algorithme de chiffrement incrémental peut laisser fuir certaines informations qui restaient secrètes avec un système de chiffrement traditionnel. Prenons par exemple un message que nous découpons en blocs et chiffrons chaque bloc avec un algorithme de chiffrement probabiliste. Si nous remplaçons un bloc du message, il suffit alors de rechiffrer ce bloc en laissant les autres inchangés. Si un tel système est efficace, il a l'inconvénient qu'un attaquant qui voit passer les deux messages chiffrés sache reconnaître qu'il s'agit de deux versions différentes d'un même document. Il est en fait difficile d'empêcher cette corrélation en gardant un système efficace et toute solution envisagée serait logiquement en $\mathrm{O}(n)$ opérations où n est le nombre de blocs du message.

5. Quelques systèmes

5.1. Algorithmes de hachage

Bellare et Micciancio (1997) ont introduit le paradigme *Randomize-then-Combine*, illustré Figure 2, pour construire des fonctions de hachage basées sur le problème du sac à dos. Ces constructions sont d'une part parallélisables et permettent d'autre part d'obtenir l'incrémentalité pour une opération de remplacement. Nous résumons leur idée comme suit. Nous notons N le nombre maximum de blocs qu'un message peut contenir et nous fixons $l = \log(N) + s$ où s est la taille d'un bloc. Nous notons également G un ensemble sur lequel un opérateur $\odot$ a été défini et une fonction de compression $h : \{0,1\}^l \to G$, opération aussi appelée *randomizer*. Pour chaque bloc $i = 1, \ldots, n$ du document D, nous concaténons au bloc P_i l'indice i codé sur $\log(N)$ bits de façon à obtenir un bloc augmenté $P'_i = i \| P_i$. Pour chaque bloc P'_i nous calculons le haché correspondant $y_i = h(P'_i)$. Enfin nous combinons $y_1, \ldots, y_n$ en utilisant l'opérateur pour obtenir la valeur de haché finale $y = y_1 \odot y_2 \odot \cdots \odot y_n$. Les auteurs appellent cette construction *Randomize-then-Combine* et montrent des

exemples de groupes possibles tels que $G = (\mathbb{Z}/m\mathbb{Z}, +)$ avec m de taille 160 bits, ou $G = ((\mathbb{Z}/2\mathbb{Z})^k, \oplus)$ avec $k = 160$ bits ou encore le groupe multiplicatif d'un corps $G = ((\mathbb{Z}/p\mathbb{Z})^*, \times)$ avec p un nombre premier de taille 1024 bits. Ils montrent également qu'en effectuant une élimination de Gauss le deuxième choix n'est pas résistant aux secondes préimages et suggèrent donc d'utiliser la première solution pour des raisons de performance.

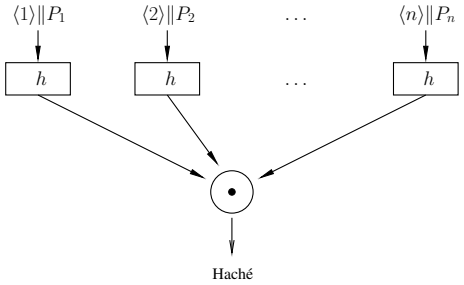


Figure 2. Illustration du paradigme Randomize-then-Combine

Afin de supporter d'autres opérations, Goi *et al.* (2001) adaptent ce paradigme pour construire la fonction de hachage PCIHF (pour Pair Chaining & Modular Arithmetic Combining Incremental Hash Function). Ils utilisent un groupe additif et chaînent les blocs par paires de sorte que $y = \sum_{i=1}^{n-1} h(P_i \| P_{i+1}) \bmod m$. Une opération $M = (\mathsf{INSERT}, j, P^*)$ se fait en temps constant. Il suffit de soustraire la contribution de $h(P_j \| P_{j+1})$ et d'ajouter les contributions de $h(P_j \| P^*)$ et $h(P^* \| P_{j+1})$. Plus généralement, l'insertion d'une suite de x blocs contigus requière une soustraction et $x + 1$ additions modulo m si elle ne se fait ni en début, ni en fin du document. Une insertion de x blocs au tout début ou à la toute fin du document requière exactement x additions et aucune soustraction [9]. Cette idée de chaînage est en fait dûe à Bellare, Goldreich et Goldwasser (1995) qui l'utilisèrent dans un contexte à clé secrète pour construire un MAC (voir XOR ci-après). Un des problèmes de PCIHF est que ce chaînage est réalisé uniquement sur les blocs du document qui ne sont pas nécessairement distincts, ce qui, avec de simples permutations des blocs du document peut mener à de multiples secondes préimages (Phan, Wagner, 2006). Néanmoins, le choix fait par Goi *et al.* est certainement volontaire pour garder une fonction de hachage "déterministe [10]" et avec une sortie de taille constante. Comme nous continuerons de le voir après, il ne semble pas exister de moyens de supporter des opérations d'insertion et de suppression de façon complètement sûre sans ajouter un surcoût de stockage (ou de communication) en $\mathrm{O}(n)$.

9. Les coûts diffèrent car insérer une suite de blocs avant le premier bloc ou après le dernier bloc ne modifie aucune des anciennes paires. Par contre, insérer ces blocs quelque part entre le premier et le dernier bloc modifie une ancienne paire, introduisant une soustraction et une addition en sus.

10. L'objectif des auteurs est de construire une fonction de hachage. Ajouter quelque chose (du pseudo-aléa) dans chaque paire de blocs nous fait sortir du cadre des fonctions de hachage.

La fonction h étant publique, Wagner (2002) montre que pour la plupart de ces constructions il existe des attaques sous-exponentielles [11]. Conséquemment, les paramètres de sécurité (les tailles des groupes) utilisés doivent être élargis, amoindrissant les performances de ces systèmes ainsi que l'intérêt de l'incrémentalité. Étant donné que ces attaques font encore l'objet d'améliorations (Minder, Sinclair, 2012 ; Nikolić, Sasaki, 2015), des précautions sont à prendre quant à l'utilisation d'opérateurs d'"agrégation" légers.

5.2. Systèmes de signature et d'authentification de message

Dans ce qui suit nous donnons des exemples de systèmes se plaçant dans les différents modèles de sécurité énoncés section 4.2.1.

Certains systèmes d'authentification de messages offrent l'incrémentalité de manière naturelle pour une opération de remplacement. C'est par exemple le cas, dans le modèle de sécurité classique, des systèmes symétriques basés sur des fonctions de hachage universelles (Carter, Wegman, 1977), tels que poly1305-AES (Bernstein, 2005) ou GMAC (McGrew, 2005 ; McGrew, Viega, 2005). Dans le cas asymétrique, si la fonction de hachage utilisée dans la signature s'appuie sur un mode de hachage en arbre, et si les valeurs des nœuds internes de l'arbre sont conservées, alors nous obtenons une signature incrémentale (Bellare *et al.*, 1994) supportant l'opération de remplacement ou d'ajout d'un bloc à la fin du fichier en $\mathrm{O}(\log n)$ évaluations d'une fonction interne (*p. ex.* une fonction de compression ou de hachage). Du fait des propriétés de la fonction de compression/de hachage, le bloc qui doit être changé peut être vérifié en $\mathrm{O}(\log n)$ évaluations de la dite fonction avant d'effectuer le changement. Le système obtenu est donc à la fois (efficacement) incrémental et *a priori* sûr contre les attaques par substitution.

GMAC. Parmi tous les codes d'authentification de message, GMAC est un de ceux qui méritent un peu plus notre attention. D'une part il fait partie du standard SP-800-38D (Dworkin, 2007) et d'autre part, comme il se base sur un polynôme à la Carter-Wegman (Wegman, Carter, 1979 ; 1981), les multiplications peuvent être faites avec des instructions matérielles (PCLMULQDQ) et le cas échéant avec des tables de précalculs, ce qui le rend dans les deux cas très performant. GMAC est très similaire à Poly1305-AES et les deux ont les mêmes propriétés algorithmiques, si bien qu'ici nous nous contentons de présenter GMAC qui est un cas spécial de GCM (Galois/Counter Mode) dans lequel nous avons retiré la confidentialité du message. Tout comme POLY1305-AES, l'algorithme GMAC supporte efficacement les opérations de remplacement d'un bloc (ou plusieurs), l'ajout d'une suite de blocs à droite (préfixation) ou à gauche du document (suffixation). Les propriétés algébriques et les

11. Notamment celles utilisant un groupe additif. Aucune attaque n'a été montrée pour le groupe multiplicatif, qui aurait alors mené à un algorithme efficace pour résoudre le logarithme discret. L'utilisation de la multiplication n'est de toute façon pas intéressant du point de vue des performances.

opérations d'édition permises par GMAC sont décrites dans (McGrew, 2005, page 3 et 4).

Les deux principales fonctions utilisées sont le chiffrement par bloc et la multiplication dans $GF(2^{128})$. Étant donné un document D divisé en blocs P_1, ..., P_n de 128 bits, le calcul du tag se résume comme suit. Nous tirons aléatoirement un vecteur $IV \in \{0,1\}^{128}$ distinct des précédents. Nous fixons l'élément secret $H = f_K(0)$ où f_K est une fonction pseudo-aléatoire (concrétisé par AES) et nous calculons $T = \mathrm{GMAC}(K, IV, M) = f_K(IV) \oplus \mathrm{GHASH}(H, M, \{\})$ où

$$\begin{aligned}\mathrm{GHASH}(H, M, \{\}) &= P_1 \cdot H^{n+1} \oplus P_2 \cdot H^n \oplus \cdots \oplus P_n \cdot H^2 \oplus (len(D) \| 0^{64}) \cdot H \\ &= ((\cdots (P_1 \cdot H) \oplus \cdots \oplus P_n) \cdot H \oplus (len(D) \| 0^{64})) \cdot H\end{aligned}$$

et $len(D)$ est la longueur du document codé sur 64 bits. Pour une opération de remplacement d'un bloc P_j par un bloc P^*, le tag T se met à jour en T' comme suit. Nous tirons aléatoirement un nouvel IV' distinct des précédents et nous calculons :

$$T' = T \oplus f_K(IV) \oplus f_K(IV') \oplus (P_j \oplus P^*) \cdot H^{n-j+2}.$$

Du fait de l'utilisation de Square-and-Multiply, cette opération se réalise en $\mathrm{O}(\log n)$ multiplications et $\mathrm{O}(1)$ évaluations de f_K. De façon similaire, une opération de modification de k blocs contigus s'effectue en $\mathrm{O}(k + \log n)$ multiplications et $\mathrm{O}(1)$ évaluations de f_K.

Pour une opération d'ajout d'un texte C constitué de l blocs $C_1, \ldots, C_l$ à gauche du document, le tag T se met à jour en T' en $\mathrm{O}(l + \log n)$ multiplications et $\mathrm{O}(1)$ évaluations de f_K. Nous procédons comme suit, en tirant aléatoirement un nouvel IV' distinct des précédents et en calculant

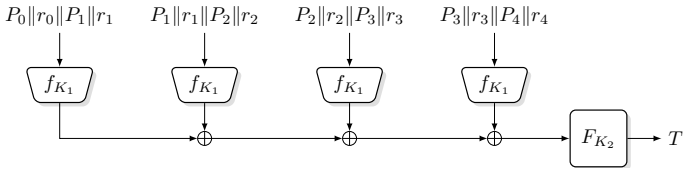
$$\begin{aligned}T' = T \oplus f_K(IV) \oplus f_K(IV') \oplus H^n \cdot \mathrm{GHASH}(H, C, \{\}) \\ \oplus H \cdot (len(C) \oplus len(D) \oplus len(C \| D) \| 0^{64}).\end{aligned}$$

Une opération d'ajout d'une suite de l blocs à droite du document s'effectue de façon similaire en $\mathrm{O}(l)$ multiplications dans $GF(2^{128})$ et $\mathrm{O}(1)$ évaluations de f_K.

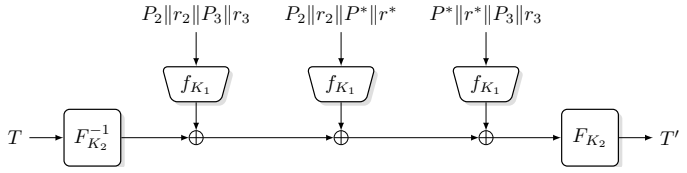
XOR. Ce schéma d'authentification conçu par Bellare, Goldreich et Goldwasser (1995, page 8), est une extension d'une proposition précédente (Bellare, Guérin, Rogaway, 1995) qui ne supportait que des opérations de remplacement. Avec la technique de chaînage employée, ce système supporte maintenant des opérations de suppression et d'insertion. Celui-ci est défini de la façon suivante. Une paire de clés symétriques (K_1, K_2) est générée. Notons f_{K_1} une fonction pseudo-aléatoire et F_{K_2} une permutation pseudo-aléatoire dont les entrées sont de taille s bits. Nous avons besoin d'un algorithme $rand$ qui étant donné une chaîne de bits c tire aléatoirement une chaîne de k bits notée r et renvoie $c \| r$. Nous découpons un document en une suite de blocs P_1, ..., P_n que nous préfixons et suffixons par deux blocs spéciaux servant de sentinelles, notés P_0 et P_{n+1}. Le MAC est alors calculé comme suit : nous appliquons la fonction $rand$ aux blocs P_0,P_1, ..., P_n, P_{n+1} pour obtenir les versions randomisées notées

$R_0, R_1, \ldots, R_n, R_{n+1}$. Nous calculons $h = \oplus_{i=0}^{n} f_{K_1}(R_i \| R_{i+1})$ que nous appelons la valeur de haché de D. Enfin nous déterminons le tag $T = F_{K_2}(h)$. Formellement, le tag d'authentification final est en fait constitué du tag T et de tous les aléas générés par $rand$. Le calcul d'un tag pour un document de 3 blocs P_1, P_2 et P_3 est illustré dans la figure 3a.

Lorsque nous appliquons l'opération $M = (\mathsf{INSERT}, i, P^*)$ sur D pour obtenir D', le tag est mis à jour de la façon suivante : nous calculons d'abord $R' = rand(P^*)$ de sorte que la nouvelle version randomisée du document soit $R_0 \| \ldots \| R_i \| R' \| R_{i+1} \| \ldots \| R_{n+1}$, nous récupérons le haché h via le tag $h = F_{K_2}^{-1}(T)$ et nous le mettons à jour en calculant $h' = h \oplus f_{K_1}(R_i \| R_{i+1}) \oplus f_{K_1}(R_i \| R') \oplus f_{K_1}(R' \| R_{i+1})$. Enfin, nous produisons le nouveau tag $T' = F_{K_2}(h')$. La suite des étapes est schématisée dans la figure 3b sur l'exemple d'une opération $M = (\mathsf{INSERT}, 2, P^*)$. Pour effectuer l'opération $M = (\text{delete}, i)$ sur D, nous calculons $R' = rand(P_{i-1})$ et nous mettons à jour le haché en calculant $h' = h \oplus f_{K_1}(R_{i-1} \| R_i) \oplus f_{K_1}(R_i \| R_{i+1}) \oplus f_{K_1}(R' \| R_{i+1}) \oplus d$ où $d = f_{K_1}(R_{i-2} \| R_{i-1}) \oplus f_{K_1}(R_{i-2} \| R')$ si $i \geq 2$ et 0 si $i = 1$. Encore une fois, le tag est simplement $T' = F_{K_2}(h')$. Toutes les opérations citées sont possibles efficacement, à l'exception de la division d'un document.



(a) Calcul d'un tag pour un message de 3 blocs $M = P_1 \| P_2 \| P_3$



(b) Mise à jour du tag pour l'insertion d'un bloc

Figure 3. Schéma d'authentification incrémental s'appuyant sur l'opérateur XOR

Ce schéma satisfait la transparence des modifications et est prouvé sûr dans le modèle de sécurité classique. Nous pouvons remarquer qu'il ne l'est plus si nous donnons la possibilité à l'attaquant d'altérer les documents. Bellare, Goldreich et Goldwasser (1995) fournissent l'exemple suivant : l'attaquant demande un MAC pour un document contenant 5 blocs, noté $abcde$. Ensuite il altère le contenu de ce document en cde et demande une suppression du second bloc. Il obtient alors un MAC valide pour un document qui n'a jamais été vu auparavant, le document $abce$.

Schémas basés sur les arbres. Bellare, Goldreich et Goldwasser (1995) proposent un système d'authentification résistant aux intrusions basé sur un arbre 2-3 qui sup-

porte toute sorte d'opérations, insertion, suppression et remplacement d'un bloc, et également des opérations de type couper ou coller, chacune d'elles en un nombre logarithmique d'opérations. Bien que ce type de systèmes puisse s'instancier dans le cas asymétrique, nous nous plaçons ici dans le cas symétrique. Nous notons MA un algorithme d'authentification de message non incrémental et MA_K la fonction de marquage induite par MA avec la clé K (par exemple $\mathrm{MA}_K = f_K$, où f_K est pris dans une famille de fonctions pseudo-aléatoires). Nous notons également VMA l'algorithme de vérification correspondant et VMA_K la fonction prenant en entrée le message m et le tag t et vérifiant si $\mathrm{MA}_K(m) = t$. L'idée est de construire à partir de ces primitives un système d'authentification incrémental qui, lorsque nous modifions une partie du document, vérifie rapidement que cette partie est valide avant d'effectuer la modification.

Pour simplifier la compréhension, nous résumons l'idée en utilisant un arbre binaire parfaitement équilibré ne supportant que des opérations de remplacement et nous supposons que les documents sont de taille $n = 2^h$ blocs où h est un entier positif. Notons D le document, considéré comme une liste de n blocs $P_1, \ldots, P_n$. Notons V l'ensemble des nœuds de l'arbre et $\mathrm{Tag} : V \to \{0,1\}^*$ la fonction qui associe un tag à chaque nœud. Ce schéma d'authentification prend comme entrées la clé K et le document D, et retourne un tag (une marque d'authentification) correspondant à un arbre binaire équilibré de tags. Ces tags sont calculés des feuilles jusqu'à la racine de la façon suivante :

1. Pour chaque bloc $P \in D$, $\mathrm{Tag}(w) = \mathrm{MA}_K(P)$ où $w \in V$ est une feuille de l'arbre.

2. Pour chaque nœud interne $w \in V$ qui n'est pas la racine, $\mathrm{Tag}(w) = \mathrm{MA}_K(\mathrm{Tag}(w_0), \mathrm{Tag}(w_1))$ où w_0 et w_1 sont les nœuds fils gauche et droite.

3. Pour le nœud racine noté λ, $\mathrm{Tag}(\lambda) = \mathrm{MA}_K(\mathrm{Tag}(\lambda_0), \mathrm{Tag}(\lambda_1), \alpha, cnt)$ où λ_0 et λ_1 sont les fils du nœud racine, α est le nom du document et cnt la version du document.

Pour remplacer un bloc P_j du document par le bloc P^*, nous vérifions d'abord que le chemin d'authentification de la racine jusqu'à ce bloc est valide. Au fur et à mesure que les tags du chemin sont vérifiés, nous chargeons les valeurs en entrée de la fonction Tag dans une mémoire protégée. Après avoir remplacé le bloc P_j par P^*, nous recalculons le tag de la feuille correspondante. Nous incrémentons alors cnt et nous recalculons les tags des nœuds internes sur le chemin d'authentification en nous servant des valeurs stockées dans la mémoire protégée, les tags des autres nœuds restant inchangés. L'opération nécessite au total h évaluations de VMA_K et h évaluations de MA_K.

Notons que pour obtenir un système de signature numérique incrémental il suffit de remplacer le système de code d'authentification sous-jacent par un système de signature numérique.

Pour supporter les autres opérations et des documents de taille quelconque, nous pouvons remplacer cet arbre binaire par un arbre 2-3 qui a toutes ses feuilles au même

niveau (à la même hauteur) et où chaque nœud interne a 2 ou 3 fils. Comme les arbres binaires, un arbre 2-3 est un arbre ordonné qui a donc toutes ses feuilles dans l'ordre. Ainsi, stocker un symbole dans chaque feuille de l'arbre définit une chaîne sur Σ. L'intérêt des arbres 2-3 est le support d'opérations d'insertion ou suppression d'une feuille en $\mathrm{O}(\log n)$ changements dans la topologie de l'arbre (un changement structurel consistant en l'ajout ou l'omission d'un sommet ou d'une arête). Étant donné que les changements dans l'arbre restent localisés au niveau de la feuille insérée/supprimée et au voisinage de ses ancêtres, le nombre d'évaluations de MA_K qui en résulte reste en $\mathrm{O}(\log n)$. Les opérations couper ou coller, plus délicates, peuvent également s'effectuer en $\mathrm{O}(\log n)$ évaluations de MA_K (Bellare, Goldreich, Goldwasser, 1995). Pour supporter la recherche rapide il suffit d'ajouter un compteur à chaque sommet de l'arbre spécifiant le nombre de feuilles couvertes par le sous-arbre enraciné à ce sommet. Le fonctionnement du système reste très similaire à celui utilisant l'arbre binaire à une exception près, lors d'une opération de modification nous devons également vérifier la cohérence des compteurs sur le chemin d'authentification avant d'y effectuer la mise à jour des tags.

Un avantage considérable d'un tel système est qu'il est prouvé sûr même si l'attaquant peut altérer les tags d'authentification. Nous pouvons donc stocker l'intégralité de la signature dans une mémoire non protégée.

Micciancio (1997) propose une variante randomisée de l'arbre 2-3 qui rend toute modification transparente, conférant à cette structure de données la propriété d'*indépendance de l'historique*. En fait, n'importe quelle structure probabiliste pourrait être utilisée pour obtenir la transparence des modifications, telle qu'une skip-liste non déterministe (Pugh, 1990). L'intérêt du système de Micciancio est la construction d'un tout nouvel arbre, qui, à notre connaissance, est le plus performant des arbres probabilistes, loin devant les skip-listes.

IncXMACC. Fischlin (Fischlin, 1997a) propose le système d'authentification de message noté IncXMACC qui, comme le système XOR (Bellare, Goldreich, Goldwasser, 1995) présenté ci-avant, se base sur les XOR MACs (Bellare, Guérin, Rogaway, 1995). L'algorithmique est très similaire à XOR, en ce sens qu'un chaînage pair à pair est une fois de plus utilisé pour garantir l'ordre des blocs du message. Nous ne décrivons donc ici que certaines de ses caractéristiques qui diffèrent réellement de ce que nous avons déjà vu.

Le premier intérêt de ce système réside dans la taille du tag d'authentification qui est en $\mathrm{O}(s + n \log s)$ où s est le paramètre de sécurité. Pour résumer l'idée, le chaînage pair à pair n'est pas basé sur des valeurs aléatoires comme c'est fait dans XOR (qui doivent être assez grandes pour garantir la sécurité du système) mais sur les incréments d'un compteur. Pour donner un exemple, notons c la valeur initiale du compteur et considérons un document de 4 blocs P_1, P_2, P_3, P_4 associés aux valeurs c, $c+1$, $c+2$ et $c+3$. Le chaînage pair à pair est donc constitué des maillons $[c, c+1]$, $[c+1, c+2]$ et $[c+2, c+3]$. Grossièrement, l'algorithme consiste à appliquer une fonction pseudo-aléatoire d'abord sur chaque bloc du document concaténé à l'incrément correspondant, ensuite sur chaque paire d'incréments issus du chaînage, et puis

à XORer les résultats pour obtenir une partie du tag. Étant donné que les documents sont de taille polynomiale en s, le chaînage est réalisé sur un nombre polynomial en s d'incréments. Les incréments fournis comme données auxiliaires du tag sont donc codés sur $\mathrm{O}(\log s)$ bits, à comparer aux $\mathrm{O}(s)$ bits réclamés par XOR pour le bourrage aléatoire de chaque bloc. Fischlin compare donc la taille du tag de IncXMACC à celle d'un tag produit par le système XOR ou par un système basé sur un arbre, qui dans les deux cas est en $\mathrm{O}(ns)$.

Le deuxième intérêt de ce système est qu'il est prouvé sûr contre les attaques par substitution (en protégeant le tag). Pour rappel le système basé sur un arbre (Bellare, Goldreich, Goldwasser, 1995 ; Micciancio, 1997) est sûr contre les attaques par substitution totale. IncXMACC satisfait une propriété de sécurité effectivement moins forte, mais, à l'inverse de ce système, profite de modifications en temps proportionnel à la quantité de données modifiées et d'un tag plus court. Pour résumer, ce nouveau système combine donc certains des avantages de XOR et du système basé sur un arbre, en améliorant l'aspect de la taille du tag.

Pour en venir aux aspects négatifs, contrairement à XOR (Bellare, Goldreich, Goldwasser, 1995) ou à un système basé sur un arbre "probabiliste" (Micciancio, 1997), ce dernier système ne satisfait pas la propriété de transparence des modifications. Prenons le même exemple de document à 3 blocs et considérons une opération d'insertion d'un bloc P^* entre P_1 et P_2. L'algorithme incrémental de IncXMACC est alors assez intuitif : il suffit d'incrémenter le compteur pour obtenir la valeur $c+4$ et de scinder le maillon $[c, c+1]$ pour produire les maillons $[c, c+4]$ et $[c+4, c+1]$. Les incréments dans le chaînage n'étant plus consécutifs, l'opération effectuée est alors bien visible.

5.3. Systèmes de chiffrement

Encrypt-then-Incremental-MAC. Buonanno *et al*. (2002) construisent un chiffrement authentifié, noté rECB-XOR, selon le paradigme *Encrypt-then-MAC*. Ils proposent un mode de chiffrement de type ECB randomisé (rECB) et suggèrent de calculer un tag par-dessus avec un système d'authentification de message incrémental, par exemple XOR (Bellare, Goldreich, Goldwasser, 1995), afin d'obtenir un mode de chiffrement authentifié. Le mode de chiffrement (sans la partie authentification) est construit comme suit. Étant donné une famille de permutations pseudo-aléatoires F et une taille de bloc de s bits, nous découpons un message D en blocs P_1, P_2, ..., P_n, nous choisissons aléatoirement des blocs r_0, r_1, ..., $r_n \in \{0,1\}^s$ et calculons :

$$F_{K'}(r_0), F_{K'}(r_1 \oplus r_0), F_{K'}(P_1 \oplus r_1), \ldots, F_{K'}(r_n \oplus r_0), F_{K'}(P_n \oplus r_n).$$

Notons alors cette liste, notre chiffré, par t et détaillons comment les opérations d'édition du message y sont répercutées. Pour une opération (DELETE, i), le chiffré t' du nouveau message correspond à la liste de blocs précédente dépourvu des deux

blocs $F_{K'}(r_i \oplus r_0), F_{K'}(P_i \oplus r_i)$. Pour une opération $(\mathsf{REPLACE}, i, P^*)$, nous tirons aléatoirement un bloc $r'_i \in \{0,1\}^s$, et remplaçons dans t la paire de blocs $F_{K'}(r_i \oplus r_0), F_{K'}(P_i \oplus r_i)$ par la paire $F_{K'}(r'_i \oplus r_0), F_{K'}(P^* \oplus r'_i)$. Enfin, pour une opération $(\mathsf{INSERT}, i, P^*)$ qui consiste à insérer P^* entre P_i et P_{i+1}, nous tirons aléatoirement un bloc $r' \in \{0,1\}^s$, et insérons dans t la nouvelle paire $F_{K'}(r' \oplus r_0), F_{K'}(P^* \oplus r')$ entre les paires $F_{K'}(r_i \oplus r_0), F_{K'}(P_i \oplus r_i)$ et $F_{K'}(r_{i+1} \oplus r_0), F_{K'}(P_{i+1} \oplus r_{i+1})$.

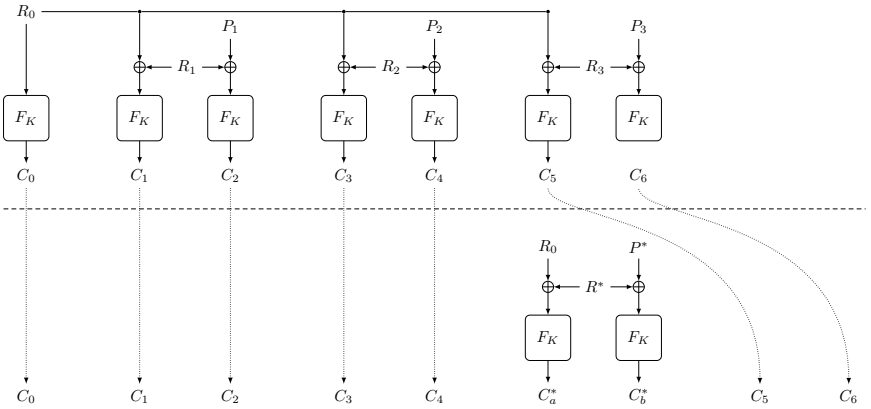


Figure 4. Chiffrement d'un document $D = P_1 \| P_2 \| P_3$, puis insertion d'un bloc P^* entre P_2 et P_3

Ce système de chiffrement incrémental est ind-CPA et permet des mises à jour du type insertion, suppression et modification de bloc. De plus, si le MAC calculé sur le chiffré est cryptographiquement sûr, le système composé est ind-CCA. Puisqu'une mise à jour incrémentale ne change potentiellement qu'une petite partie du chiffré, l'incrémentalité du MAC voit son intéret. Le système composé obtenu supporte alors les opérations usuelles et respecte la propriété de transparence d'une modification. À l'exception de la suppression qui est gratuite, les autres opérations s'exécutent en temps proportionnel au nombre de blocs changés/insérés.

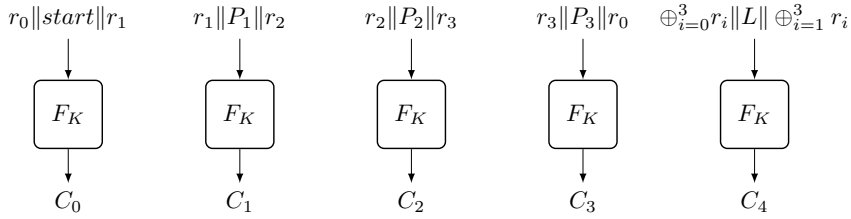
Buonanno *et al.* proposent également deux modes de chiffrement authentifié supportant des opérations de manière transparente. Leur particularité est qu'ils ne se basent pas sur un paradigme générique mais plutôt sur un système de somme de contrôle intégré (Jutla, 2001 ; Rogaway *et al.*, 2003). Le premier, une version incrémentale de IAPM (pour *Integrity Aware Parallelizable Mode* (Jutla, 2001)), ne supporte que des opérations de remplacement, alors que le second, nommé RPC, supporte également des opérations d'insertion et suppression. Nous ne détaillons que le second.

RPC. Nous décrivons la version corrigée (Wang *et al.*, 2006) qui évite des attaques de falsification. Ce système est légèrement moins performant que inc-IAPM mais permet des opérations d'insertion et suppression. Le mode est spécifié par les paramètres s et r, où s est la taille d'un bloc et r la quantité de bourrage aléatoire. Le document est divisé en une suite de blocs $P_1, \ldots, P_n$ de taille $s - 2r$. Les blocs $C_0, C_1, \ldots, C_n$,

C_{n+1} du chiffré sont alors calculés en utilisant la permutation pseudo-aléatoire $F_{K'}$ de la façon suivante :

$$F_{k'}(r_0\|\text{start}\|r_1), F_{k'}(r_1\|P_1\|r_2), F_{k'}(r_2\|P_2\|r_3),$$
$$\ldots, F_{k'}(r_n\|P_n\|r_0), F_{k'}(\oplus_{i=0}^{n} r_i\|L\|\oplus_{i=1}^{n} r_i)$$

où start indique le commencement du message et L est la taille du message. Le déchiffrement se fait en calculant $F_{K'}^{-1}$ sur chaque bloc du chiffré et en vérifiant que le premier bloc contient bien un chiffrement de start, que les $\{r_i\}$ sont correctement chaînés, et que le déchiffrement du dernier bloc donne les sommes attendues. Si cette vérification réussit alors l'algorithme retourne le document, sinon il retourne le symbole $\perp$.



(a) Chiffrement d'un message de 3 blocs $M = P_1\|P_2\|P_3$

Figure 5. Schéma de chiffrement authentifié RPC

Nous décrivons rapidement les algorithmes incrémentaux pour effectuer une insertion, une suppression ou un remplacement de bloc. Prenons l'exemple d'une opération $M = (\mathsf{INSERT}, i, P^*)$, à savoir une insertion d'un bloc P^* juste après le bloc P_i. Nous déchiffrons le bloc contenant P_i pour déterminer les valeurs r_i, r_{i+1}. Nous choisissons un nouveau bloc aléatoire r^* et nous calculons $C'_i = F_{K'}(r_i\|P_i\|r^*)$ et $C^* = F_{K'}(r^*\|P^*\|r_{i+1})$. Quelle que soit l'opératon effectuée (y compris une suppression), nous choisissons aléatoirement de nouveaux blocs r_0 et r_1, nous recalculons les sommes de contrôle et les blocs chiffrés correspondants C'_0, C'_1 et C'_{n+1}. Le chiffré mis à jour est alors C'_0, C'_1, $\ldots$, C_{i-1}, C'_i, C^*, C_{i+1}, $\ldots$, C_n, C'_{n+1}. Remarquons enfin que, bien que non signalée dans les spécifications de RPC, la fusion (concaténation) de deux documents est possible aux prix de quelques déchiffrements et rechiffrements, toujours en prenant soin de regénérer les deux premiers blocs aléatoires. La nouvelle somme de contrôle peut en effet être recalculée en un nombre constant d'additions.

Ces systèmes ont l'inconvénient de démultiplier la taille des chiffrés et le nombre d'évaluations du chiffrement par bloc. Supposons que nous utilisions un chiffrement par bloc de 128 bits (par exemple AES). En prenant pour référence la taille d'un document en clair, la taille du chiffré en utilisant rECB-XOR est multipliée par 4 et le nombre d'évaluations d'AES par 6. En ce qui concerne inc-IAPM la taille du chiffré

et le nombre de calculs d'AES sont multipliés par 2. Enfin, pour RPC, en choisissant $r = 48$, la taille du chiffré et le nombre de calculs d'AES sont multipliés par 4. Bien que tous ces systèmes soient prouvés ind-CCA sûr (Buonanno *et al.*, 2002) (dans le modèle de sécurité classique pour l'infalsifiabilité), la borne de sécurité de RPC concernant l'indistinguabilité est assez faible, de l'ordre de $q_i/2^{48}$.

Chiffrement déterministe à clé publique. Un chiffrement déterministe est intéressant pour l'indexation et pour effectuer une recherche rapide sur des données chiffrées. Ce type de mécanisme a aussi son utilité pour faire du chiffrement convergent que nous combinons avec des techniques de dé-duplication pour économiser de l'espace de stockage sur le Cloud. Mironov *et al.* (Mironov *et al.*, 2012) traitent le sujet de l'incrémentalité pour l'opération de remplacement dans un système de chiffrement déterministe à clé publique. Une de leurs constructions s'appuie sur un partitionnement aléatoire du document selon une technique *"sample-then-extract"* (Nisan, Zuckerman, 1993), utilisée pour préserver le taux de min-entropie du document initial sur chacune des parties qui en est issue. De manière plus précise, étant donné un document D de taille n bits, il s'agit de choisir aléatoirement une partition $S_1, \ldots, S_{n/t}$ des positions des bits $\{1, \ldots, n\}$. La partition est alors constituée des projections $P_{S_1}, \ldots, P_{S_{n/t}}$. En choisissant t de manière à avoir le taux de min-entropie voulu, chacune des projections (un bloc de t bits) peut alors être chiffrée avec un système de chiffrement déterministe à clé publique. L'incrémentalité s'effectue simplement : l'inversement d'un bit du document nécessite le rechiffrement de la projection qui a été touchée.

Il n'est pas étonnant que seule l'opération de remplacement soit supportée. Pour les besoins du déterminisme, la projection (une partition aléatoire selon la technique *"sample-then-extract"*) est choisie à l'initialisation du système par l'algorithme de génération de clés. La projection n'a pas besoin d'être mise à jour pour rendre une telle opération possible. À première vue, il semble que toute tentative pour supporter une opération d'insertion/suppression en mettant à jour la partition rende le système non déterministe. Une question que nous pouvons nous poser est la suivante : existe-t-il des opérations qui permettent de mettre à jour efficacement cette projection tout en s'assurant qu'elle continue de dépendre entièrement de la longueur du document modifié et de la sortie de l'algorithme de génération de clés ? Qu'en est-il, par exemple, d'une opération du type suffixation d'un bloc ?

6. Discussion

La table 1 récapitule les caractéristiques des systèmes que nous avons décrits. Nous supposons que les documents manipulés contiennent $\mathrm{O}(n)$ blocs. La colonne "Résistance aux falsifications" indique si le système ne résiste pas aux falsifications (a), s'il résiste aux attaques classiques (b), aux attaques par substitution de message (c) ou aux attaques par substitution totale (d). Les complexités asymptotiques des opérations de modification sont décrites en termes d'opération(s) de base. Une opération de base du système peut être l'évaluation d'une fonction de hachage, d'une fonction pseudo-

aléatoire (PRF), d'une permutation pseudo-aléatoire (PRP), d'une fonction de signature ou d'un MAC. Il peut s'agir également d'une opération arithmétique ou logique. Pour simplifier les complexités des opérations de modification, nous ne quantifions pas le nombre de fois que nous utilisons chaque opération de base. Alternativement, nous écrivons une seule complexité asymptotique quantifiant l'utilisation de l'opération de base la plus fréquente (en dehors de l'opération $\oplus$ qui est négligée pour certains systèmes). La dernière colonne donne la taille asymptotique (en O) de la forme cryptographique produite par le système. Pour simplifier davantage les comparaisons, cette table suppose que la taille des objets cryptographiques, la taille d'une clé, la taille d'un tag (ou d'une signature) et la taille d'un bloc du document valent toutes s.

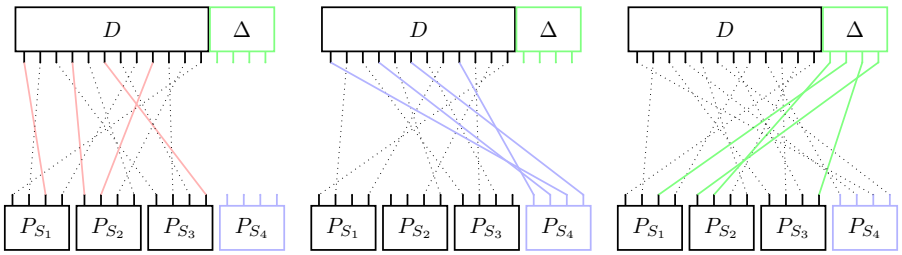
Les articles décrivant les différents systèmes cités ci-avant ne sont parfois pas exhaustifs quant aux opérations de modification permises. Certaines opérations sont absentes soit par oubli, soit parce qu'elles ne sont pas possibles efficacement, ou simplement parce qu'il existe une contrainte (implicite) empêchant ces opérations. Il s'agit le plus souvent des opérations de fusion de deux documents, ou de division d'un document en deux documents. Lorsqu'une telle opération n'est pas décrite et que nous ne pouvons pas déduire facilement l'algorithme permettant de la réaliser, la table 1 suppose alors l'utilisation de l'algorithme de transformation initiale pour effectuer cette opération. Sa complexité se retrouve donc en $\mathrm{O}\big(n\big)$. Quelquefois, nous pouvons nous-même remarquer que ces opérations ne peuvent pas être efficaces. Prenons l'exemple de RPC. Pour effectuer la division d'un document en deux parties (de tailles égales pour simplifier) et calculer les formes cryptographiques correspondantes, il nous faut calculer deux nouvelles sommes de contrôle, ce qui nous oblige à déchiffrer au moins la moitié du document afin de calculer la première somme et en déduire la deuxième. Dans chacun des deux nouveaux documents, quelques valeurs aléatoires sont regénérées, le chaînage entre les valeurs aléatoires est rétabli, et les quelques blocs qui ont changé sont rechiffrés. Si nous supposons le chiffrement par bloc de coût identique au déchiffrement (par bloc), alors le temps d'exécution d'une telle opération est réduit approximativement d'un facteur 2 en comparaison du chiffrement de la concaténation des deux documents.

Prenons ensuite l'exemple du chiffrement incrémental déterministe de Mironov *et al.* (Mironov *et al.*, 2012). Pour ce schéma, exceptionnellement, n est le nombre de bits du document. Ce schéma requiert une permutation aléatoire des positions des bits, puis un découpage de la suite de bits résultante en blocs de t bits (les projections), afin de préserver le taux de min-entropie sur les blocs à chiffrer. La partition aléatoire est choisie au moment de la génération des clés, ce qui suppose des documents de taille fixe et une seule opération de modification possible. Pour que des opérations du type *insertion d'un bloc P^* entre le bloc i et $i+1$* (ou du type *suppression du bloc i*) soient possibles, il faut que nous puissions mettre à jour la partition aléatoire. Nous aimerions bien la mettre à jour efficacement en utilisant l'information i, mais pour les besoins du déterminisme la seule valeur supplémentaire dont peut dépendre la partition est la longueur finale (après modification) du document. La variabilité de l'indice i d'une telle opération est, en effet, incompatible avec l'aspect déterministe du système. La solution est alors de regénérer entièrement la partition sur les bits de

Tableau 1. Comparaison des systèmes en termes de propriétés de sécurité et de performances asymptotiques (utilisant la notation grand O*)*

Type de système incrémental	Système	Opération(s) de base	Transparence des modifications	Résistance aux falsifications (a,b,c,d)	Insertion de k blocs	Suppression de k blocs	Remplacement de k blocs	Préfixation de k blocs	Suffixation de k blocs	Fusion de 2 documents	Division d'un document	Taille de la forme cryptographique
Hachage	AdHASH (Bellare, Micciancio, 1997)	fonction de hachage idéale et addition sur $\mathbb{Z}/m\mathbb{Z}$	✓	a	n	n	k	n	k	n	n	s
	MuHASH (Bellare, Micciancio, 1997)	fonction de hachage idéale et multiplication sur $\mathbb{Z}/p\mathbb{Z}$	✓	a	n	n	k	n	k	n	n	s
	PCIHF (Goi *et al.*, 2001)	fonction de hachage idéale et addition sur $\mathbb{Z}/m\mathbb{Z}$	✓	a	k	1	k	k	k	1	n	s
Signature ou code d'authentification de message	GMAC (McGrew, 2005)	PRF, $\times$ et $\oplus$ dans $GF(2^{128})$	✓	b	n	n	$k + \log n$	$k + \log n$	k	$\log n$	n	s
	XOR (Bellare, Goldreich, Goldwasser, 1995)	PRP, PRF et $\oplus$	✓	b	k	1	k	k	k	1	n	ns
	2-3 arbre (Bellare, Goldreich, Goldwasser, 1995)	signature ou MAC	✗	d	$k + \log n$	$\log n$	$k + \log n$	$k + \log n$	$k + \log n$	$\log n$	$\log n$	ns
	Oblivious Tree (Micciancio, 1997)	signature ou MAC	✓	d	$k + \log n$	$\log n$	$k + \log n$	$k + \log n$	$k + \log n$	$\log n$	$\log n$	ns
	IncXMACC (Fischlin, 1997a)	PRF et $\oplus$	✗	c	k	1	k	k	k	1	n	$s + n \log s$
Chiffrement authentifié	rECB-XOR (Buonanno *et al.*, 2002)	PRP, PRF et $\oplus$	✓	b	k	1	k	k	k	n	n	ns
	RPC (Buonanno *et al.*, 2002)	PRP et $\oplus$	✓	b	k	1	k	k	k	1	n	ns
Chiffrement à clé publique déterministe	Système générique de (Mironov *et al.*, 2012)	chiffrement à clé publique déterministe (non incrémental)	✓	a	n	n	k	n	n	n	n	ns

l'ensemble des parties du document. Cela nous amène, avec une forte probabilité, à rechiffrer toutes les parties du document. La situation est différente avec une opération qui reste toujours localisée au même endroit, telle que l'opération de *suffixation d'un bloc*, dont une solution se trouve dans le mélange de Knuth (Knuth, 1969). Ce mélange est réalisé par une suite de transpositions. Notons $\mathcal{S}_n$ l'ensemble des permutations de $\{1, \ldots, n\}$ et σ_n une permutation aléatoire de $\mathcal{S}_n$. Nous définissons une application $f : \mathcal{S}_n \times \{1, \ldots, n+1\}$ dans $\mathcal{S}_{n+1}$ de la manière suivante : l'image de (σ_n, k) est la composée de $(n+1\ k)$ (la transposition qui échange $n+1$ et k) et de σ. Puisque f est une bijection entre deux ensembles finis, l'image d'une variable aléatoire de loi uniforme sur $\mathcal{S}_n \times \{1, \ldots, n+1\}$ par f est donc une variable aléatoire de loi uniforme sur $\mathcal{S}_{n+1}$. Afin de supporter efficacement la suffixation d'un nouveau bloc de taille t bits, nous étendons la permutation σ_n de $\mathcal{S}_n$ en une permutation σ_{n+t} de $\mathcal{S}_{n+t}$, par une composition avec t transpositions. L'opération est illustrée par un exemple dans la figure 6. Cette transformation introduit une nouvelle projection et change au plus t des précédentes projections, requérant donc une nouvelle opération de chiffrement et au plus t rechiffrements. Remarquons enfin que si la génération pseudo-aléatoire de ces transpositions est réalisée à l'aide d'un paramètre fixé (lors de la génération des clés) et d'un compteur de bit, la suite complète des transpositions reste la même quel que soit le document, et le système reste donc déterministe.



(a) Projections défaites pour D (b) Projections refaites pour D (c) Nouvelles projections pour Δ

Figure 6. Exemple d'une mise à jour déterministe des projections pseudo-aléatoires, pour une opération de suffixation d'un bloc noté Δ. L'exemple suppose une taille de bloc fictive de 4 bits, et illustre les changements dans les projections lorsque la permutation correspondante est composée avec quatre transpositions aléatoires.

Plusieurs choses peuvent être constatées en regardant les données de cette table. Le système de signature (ou d'authentification) de Micciancio (1997) est celui qui offre le maximum de propriétés de sécurité et le meilleur compromis entre les complexités calculatoires de chacune des opérations. Son point faible est la taille de la forme cryptographique produite. Un système de signature incrémental pourrait être obtenu en utilisant une fonction de hachage incrémentale comme PCIHF, dont les complexités sont très bonnes, mais son utilisation est problématique car il faut s'assurer que les blocs du document sont tous distincts. Cette contrainte peut néanmoins être enlevée en concaténant à chacun de ces blocs un bloc aléatoire, de manière à assurer un ordre "probabiliste" tel qu'employé dans le système symétrique XOR.

Le système IncXMACC est intéressant de par la taille de la forme cryptographique qu'il produit, mais l'opération de division ne semble pas supportée [12] et la propriété de transparence des modifications n'est pas satisfaite. Afin de rendre une modification transparente au sens de la définition 10, nous devons nous assurer que les tags produits, par l'algorithme de transformation initiale ou par l'algorithme de mise à jour, ne dépendent que de choix aléatoires et de la taille des documents. Il s'agit d'un problème de structure de données qui peut être résolu, en effectuant peu de changements au système original, si nous restreignons l'ensemble des opérations possibles aux opérations de type *insertion*, *suppression* et *remplacement* [13]. Notons σ_n une permutation tirée aléatoirement (uniformément) dans $\mathcal{S}_n$. Il suffit en effet d'appliquer le chaînage par paires sur les incréments permutés par σ_n et d'appliquer les astuces algorithmiques du mélange de Knuth (Knuth, 1969) pour mettre à jour efficacement la permutation, et donc le chaînage. Considérons un document de $n-1$ blocs, dont les blocs sont indexés par des paires d'entiers dans l'ensemble $\{1,\ldots,n\}$, sans aucune répétition. Nous traitons deux cas, l'insertion et la suppression, appuyés par des exemples :

– $(\mathsf{INSERT}, i, P^*)$. Nous désirons insérer un bloc juste après le i-ième. L'idée est d'agrandir (efficacement) la permutation. Supposons que σ_n est une permutation aléatoire uniforme de $\overline{S_n}$, l'inclusion naturelle de S_n dans S_{n+1} laissant fixe $n+1$. Notons $(n+1\ i)$ la transposition qui échange i et $n+1$. Étant donné un entier j tiré aléatoirement dans $\{1,\ldots,n+1\}$, la composée $\sigma' = \sigma_n \circ (n+1\ j)$ est une permutation aléatoire uniforme de $\mathcal{S}_{n+1}$. Notons maintenant $val_i = \sigma'(i)$ pour $i \in \{1,\ldots,n+1\}$, et considérons le $(n+1-i)$-cycle σ'' de $\mathcal{S}_{n+1}$ qui laisse fixe les i premières valeurs, et décale d'un cran vers la droite toutes les autres, *i.e.* $\sigma''(val_j) = val_j$ pour $j \in \{1,\ldots,i\}$, $\sigma''(val_{i+1}) = val_{n+1}$ et $\sigma''(val_j) = val_{j-1}$ pour $j \in \{i+2,\ldots,n\}$. Cette dernière application étant une bijection, la composée $\sigma'' \circ \sigma'$ est une permutation aléatoire uniforme de $\mathcal{S}_{n+1}$. Ceci nous permet de conclure qu'un chaînage par paires sur la liste des entiers permutés par $\sigma'' \circ \sigma'$ est sans biais. Ces deux opérations nécessitent tout au plus de défaire quatre paires et d'en reconstruire quatre autres.

– (DELETE, i). Nous voulons supprimer le i-ième bloc. Nous tirons aléatoirement une permutation σ de $\mathcal{S}_n$, et nous notons $val_i = \sigma(i) \in \{1,\ldots,n-1\}$ la première composante de la paire qui indexe le i-ième bloc. Notre objectif est que la suppression retourne un chaînage par paires sans biais, donc parfaitement aléatoire. Nous considérons la fonction $h\colon \mathcal{S}_n \to \mathcal{S}_{n-1}$ définie par $h = g \circ f$ avec

$$\begin{aligned} f\colon \mathcal{S}_n &\to \mathcal{S}_n \\ \sigma &\mapsto \sigma \circ (n \quad val_i), \end{aligned}$$

où $(n \quad val_i)$ est la transposition qui échange n et val_i, et $g\colon \mathcal{S}_n \to \mathcal{S}_{n-1}$ est la fonction qui supprime n. Si σ est une permutation tirée aléatoirement et uniformément dans $\mathcal{S}_n$, alors $\sigma' = f(\sigma)$ suit également la loi uniforme sur $\mathcal{S}_n$, et ce quelle que soit

12. L'auteur indique seulement qu'elle est plus coûteuse que celle des systèmes basées sur les arbres, sans fournir plus de détails.

13. La *suffixation* et la *préfixation* aussi, si elles ne concernent qu'un seul bloc

la loi de val_i. En outre, $g(\sigma')$ suit la loi uniforme sur $\mathcal{S}_{n-1}$, car chacune des $(n-1)!$ images a exactement n antécédents. Ceci nous permet de conclure qu'un chaînage par paires sur la liste des entiers permutés par $g(\sigma')$ est sans biais. Concrètement, notre application composée déplace l'entier val_i afin qu'il indexe un autre bloc, et supprime n qui indexe alors le bloc qu'on veut supprimer. Ces deux opérations nécessitent tout au plus de défaire quatre paires et d'en reconstruire trois autres.

Le chaînage par paires sur les éléments de cette nouvelle permutation peut donc être mis à jour efficacement. Prenons l'exemple de la liste de 7 incréments permutés 4 1 3 7 2 5 6, chaînés par paires de façon à ordonner les blocs du document. Nous désirons insérer dans le document un bloc juste après le 2ème. Nous choisissons, pour ce faire, un élément au hasard dans la liste, par exemple 2. Nous remplaçons 2 par 8, et insérons 2 entre 1 et 3. Cette idée résoud ce problème de transparence des opérations lorsqu'elle est employée conjointement avec le chaînage par paire. En effet, qu'il s'agisse d'un chaînage par paires de 7 premiers entiers permutés suivi d'une insertion du 8ème selon notre méthode, ou d'un chaînage par paires des 8 premiers entiers permutés, les résultats sont indistinguables. Partant de cette dernière liste, nous supprimons alors le 4ème bloc, indexé par 3. Nous échangeons 3 et 8, puis supprimons 8. Encore une fois, en supposant que la composition de toutes les opérations précédant cette suppression générait une permutation aléatoire de $\{1, \ldots, 8\}$, cette composition augmentée d'une suppression produit bien une permutation aléatoire de $\{1, \ldots, 7\}$.

Exemples :
- Une insertion :

Permutation pour 6 blocs	: 4 1 3 7 2 5 6
Chaînage (6 paires)	: [4 1], [1 3], [3 7], [7 2], [2 5], [5 6]
Insertion après 2ème bloc	: 4 1 **8** 3 7 **2** 5 6
Chaînage à jour (7 paires)	: [4 1], [**1 2**], [**2 3**], [3 7], [**7 8**], [**8 5**], [5 6]

- Une suppression :

Permutation pour 7 blocs	: 4 1 2 3 7 8 5 6
Chaînage (7 paires)	: [4 1], [1 2], [2 3], [3 7], [7 8], [8 5], [5 6]
Suppression du 4ème bloc	: 4 1 2 7 **3** 5 6
Chaînage à jour (6 paires)	: [4 1], [1 2], [**2 7**], [**7 3**], [**3 5**], [5 6]

Du côté du chiffrement authentifié symétrique, les systèmes ne sont pas nombreux et les choix de conception (Buonanno *et al*., 2002) ne leurs permettent pas d'effectuer certaines opérations efficacement, comme la fusion ou la division de documents. Par ailleurs, du point de vue de l'infalsifiabilité des formes cryptographiques, ils ne sont prouvés sûr que contre les attaques classiques. En y regardant de plus près, parmi les solutions les plus immédiates, combiner le mode rECB avec XOR (en employant le paradigme *Encrypt-then-MAC*, se référer à (Buonanno *et al*., 2002)) ne semble pas être le meilleur choix. Alternativement, dans une première étape, nous pouvons par exemple chiffrer indépendamment chaque bloc du document en utilisant un mode de chiffrement sans état (tel que CBC avec l'IV choisi aléatoirement). N'importe quelle

opération "bloc par bloc" est alors possible sur le chiffré, dont les algorithmes correspondants sont laissés au lecteur. Nous utilisons ensuite le paradigme *Encrypt-then-MAC* pour authentifier le document chiffré à l'aide du système d'authentification basé sur un arbre de Micciancio. Le système ainsi composé permet toute sorte d'opérations efficacement [14] et satisfait une propriété de sécurité plus forte. Effectivement, l'intégrité du chiffré est assuré même en cas d'*attaques par substitution totale*.

Rappelons-nous qu'un système conventionnel (non incrémental) de signature ou d'authentification de message génère une forme cryptographique de longueur en $\Theta(s)$, où s est le paramètre de sécurité. Dans le cas d'un chiffrement non incrémental, la taille du chiffré est généralement proche de celle du clair correspondant.

Des observations peuvent être dégagées des mécanismes incrémentaux que nous avons vus. L'opération de remplacement d'un bloc ou l'ajout d'un bloc à la fin du document est possible efficacement avec des mécanismes classiques. Prenons par exemple un mode de hachage en arbre pour une fonction de hachage. Il suffit de stocker les valeurs des nœuds internes de l'arbre pour rendre possible un remplacement de bloc en temps efficace. En l'occurrence, la mise à jour du haché peut se faire en $\mathrm{O}\big(\log n\big)$ évaluations de la fonction de compression. Stocker les valeurs intermédiaires d'un calcul contenant beaucoup d'opérandes permet en général des mises à jour efficaces, la contrepartie étant l'espace de stockage supplémentaire utilisé.

La plupart des systèmes incrémentaux supportent des opérations d'insertion ou suppression en un temps proportionnel à la quantité de données modifiées. Permettre d'effectuer de telles opérations, et de surcroît en toute transparence, mène à des systèmes générant des transformations cryptographiques de grande taille. Pour la majorité des systèmes de signature resp. d'authentification de message, la taille de la signature resp. du tag est en $\mathrm{O}\big(ns\big)$, où s est le paramètre de sécurité. Dans le cas d'un système de chiffrement, la taille du chiffré est doublée voire quadruplée en comparaison de la taille du texte clair.

Rendre un système incrémental consiste à diminuer les dépendances dans les flots de données à traiter. **Nous distinguons les différentes techniques employées par les systèmes de signature incrémentale afin d'assurer l'ordre des blocs :**

– Le chaînage par paires directement sur les blocs du document, tel qu'employé dans la fonction de hachage PCIHF. Cette fonction couplée à un système de signature standard a l'avantage de garder une signature courte, mais elle suppose que les blocs du document sont tous distincts.

– Le chaînage par paires sur des blocs tirés aléatoirement, tel qu'employé dans le système XOR. À titre d'exemple, un document de 3 blocs P_1, P_2 et P_3 sera étendu en 3 blocs $r_1 \| r_2 \| P_1$, $r_2 \| r_3 \| P_2$ et $r_3 \| r_4 \| P_3$ avant d'appliquer une fonction

14. Pour k blocs remplacés/ajoutés au document clair, $\mathrm{O}\big(k\big)$ blocs sont changés dans le document chiffré. Les complexités des mises à jour du système composé sont donc celles du système de Micciancio. Pour k changements dans le document clair, nous devons calculer $\mathrm{O}\big(k + \log n\big)$ MACs pour obtenir la forme cryptographique finale.

pseudo-aléatoire ou une fonction de hachage sur chacun des nouveaux blocs. Cette méthode assure un ordre avec une probabilité qui dépend de la taille, notée e, des valeurs aléatoires R_i. Pour une sécurité de 128 bits, ces valeurs doivent être de largeur au moins 64 bits, car le nombre de blocs à ordonner introduit une perte de sécurité. Remarquons que cette méthode se généralise par un chaînage de d-uplet de blocs. Ces d-uplets sont construits en faisant glisser une fenêtre de d blocs sur la suite $(r_i)_{1\leq i\leq n+d-1}$. Si d est grand, il est possible de diminuer la taille e des blocs aléatoires pour une probabilité équivalente, permettant ainsi de réduire l'expansion de la signature. La contre-partie est une efficacité des mises à jour inversement proportionnelle, car une opération de type insertion ou suppression modifie le contenu d'au plus d d-uplets. En effet, notons $E_{i,j}$ pour $i \neq j$ l'événement $\{(r_i, r_{i+1}, \ldots, r_{i+d-2}, r_{i+d-1}) = (r_j, r_{j+1}, \ldots, r_{j+d-2}, r_{j+d-1})\}$, et E l'événement $\{\exists i, j \in \{1, \ldots, n\},\ i \neq j\ t.q.\ E_{i,j}\}$, qui représente la possibilité pour un attaquant de permuter ne serait-ce que deux blocs du document. La probabilité d'occurrence de E peut être bornée comme :

$$P(E) \leq \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} P(E_{i,j}) = \frac{n(n-1)}{2^{de+1}}.$$

La taille de la signature étant dominée par la suite des valeurs auxiliaires $(r_i)_{i\geq 1}$, diminuer e et agrandir d pour un même niveau de sécurité d'à peu près $ed - 2\log_2 n$, par exemple en prenant $e = 1$ bit et $d = 128$, diminuera drastiquement ce surcoût. En contrepartie, le coût calculatoire d'une modification de type insertion ou suppression se verra augmenter d'un facteur presque d.

– Le chaînage par paires sur des nombres consécutifs, tel qu'employé dans le système IncXMACC. À titre d'exemple, un document de 3 blocs P_1, P_2 et P_3 sera étendu en 3 blocs $\langle 1\rangle \| \langle 2\rangle \| P_1$, $\langle 2\rangle \| \langle 3\rangle \| P_2$ et $\langle 3\rangle \| \langle 4\rangle \| P_3$ avant d'appliquer une fonction pseudo-aléatoire ou une fonction de hachage sur chacun des nouveaux blocs. Comparativement au chaînage par paires du point précédent, les incréments s'écrivent avec un codage de $\mathrm{O}\big(\log s\big)$ bits, car les documents sont de taille polynomiale en s. Bien que la taille de la signature dépende toujours du nombre de blocs, elle est considérablement réduite comparativement aux signatures produites avec un chaînage par paires sur des blocs pseudo-aléatoires (p. ex. tel qu'il est réalisé avec le schéma XOR). Les valeurs pseudo-aléatoires générées pour XOR sont de taille cryptographique, au contraire des valeurs de IncXMACC, initialement consécutives pour la première signature, dont le codage nécessite environ $\log n = \mathrm{O}\big(\log s\big)$ bits (p. ex. 20 bits pour un document de moins de 2^{20} blocs). Les modifications, telles qu'elles sont définies dans la proposition IncXMACC, ne sont pas transparentes, car elles ne préservent pas l'ordre initial des incréments, un ordre "croissant". Néanmoins, nous avons remarqué qu'il y avait un intérêt à adopter un ordre initial aléatoire uniforme, car un chaînage par paires sur une permutation de nombres consécutifs nous permet de préserver efficacement la transparence des modifications de type *insertion*, *remplacement* et *suppression*.

– Les modes opératoires en arbre pour une fonction cryptographique, comme celui utilisé par Micciancio. La fonction interne utilisée est un système de signature, mais

une fonction de hachage est également possible si le nœud racine est signé. Comme pour la méthode précédente avec le chaînage par paires, elle a l'inconvénient de produire des signatures de longueur proportionnelle à la longueur du document.

En ce qui concerne les propriétés d'infalsifiabilité, rendre un système sûr contre les attaques par substitution nécessite également de générer une longue série de valeurs cryptographiques. Par exemple, les deux systèmes que nous avons vu génèrent des transformations qui ont des tailles en $\mathrm{O}(ns)$ et $\mathrm{O}(s + n \log s)$. Pour situer les choses dans un contexte chronologique, la conception d'un système de signature incrémental qui soit résistant aux attaques par substitution et qui produise des signatures courtes est d'abord posé comme un problème ouvert par Bellare *et al.* (1994). Les deux systèmes présentés (Bellare, Goldreich, Goldwasser, 1995 ; Fischlin, 1997a) voient alors le jour et Fischlin (Fischlin, 1997b) montre une borne inférieure $\Omega(n)$ sur la taille d'une telle signature. Plus précisément, il montre que lorsque l'algorithme incrémental accède un seul bloc resp. un nombre constant de blocs cette borne est de $\Omega(n)$ resp. $\Omega(\sqrt{n})$.

Tous ces systèmes permettent d'effectuer des opérations sur une chaîne de blocs, mais en réalité nous voudrions effectuer des opérations sur des chaînes de bits ou d'octets, puisque nous manipulons le plus souvent des octets. Dans l'état actuel des choses, insérer ou supprimer un nombre d'octets qui n'est pas un multiple de la taille d'un bloc pose des problèmes d'alignement [15]. Un alignement inexact nous oblige à retraiter tous les blocs qui suivent la position de la modification pour un coût en $\mathrm{O}(n)$ évaluations de la primitive sous-jacente. Une solution simple à ce problème est d'utiliser un seul octet par bloc traité par le système incrémental, mais elle n'arrange pas l'efficacité du système puisqu'elle démultiplie son temps d'exécution et la taille des formes cryptographiques produites.

Si nous voulons construire des systèmes cryptographiques supportant des opérations au niveau octet, nous pouvons vouloir nous assurer du respect des propriétés que nous avons vues, comme par exemple la transparence des modifications. Idéalement, nous aimerions donc construire des mécanismes incrémentaux réduisant la taille des formes cryptographiques, permettant efficacement des opérations au niveau octet et s'assurant du respect de tout ou partie des propriétés de sécurité que nous avons énoncées (éventuellement remaniées au cas de l'incrémentalité au niveau octet). Dans la section suivante, nous nous intéressons à une manière simple d'étendre des systèmes incrémentaux au niveau bloc afin de supporter un plus grand nombre d'opérations, et ce tout en préservant leur transparence.

7. Incrémentalité au niveau octet et transparence des opérations

Les systèmes cryptographiques incrémentaux conçus jusqu'ici sont des systèmes permettant des mises à jour incrémentales au niveau bloc. Ils peuvent se révéler inexploitables puisqu'ils ne permettent pas d'effectuer des insertions ou suppressions

15. Il y a un "désalignement" si, après une modification, le premier bit de chacun des blocs localisés après cette modification ne se retrouve plus en première position.

de données de longueurs arbitraires, une caractéristique essentielle pour l'édition de documents. Nous proposons donc une méthode qui étend un système cryptographique incrémental par bloc en un système incrémental au niveau octet préservant suffisamment les performances du système initial. Cette méthode transforme une mise à jour au niveau octet en une mise à jour au niveau bloc et utilise ensuite un système incrémental au niveau bloc comme une boite noire. À chaque modification effectuée sur le document, et quel que soit le nombre de ces opérations, notre approche assure une propriété d'inconscience des modifications effectuées et conserve le même surcoût moyen pour la taille de la transformation cryptographique et le nombre d'opérations à effectuer lorsque nous appliquons l'algorithme conjugué.

Cette section est structurée de la façon suivante. Après avoir rappelé des définitions dans la section 7.1, nous exposons les inconvénients des systèmes cryptographiques incrémentaux actuels et nous introduisons en section 7.2 une méthode générique pour les étendre en des systèmes véritablement incrémentaux. Les questions concernant la sécurité sont ensuite traitées dans la section 7.3.

7.1. Quelques rappels : la cryptographie et les mises à jour de documents

Les systèmes cryptographiques prennent en entrée un document D divisé en une suite de blocs de taille fixe $P_0, P_1, ..., P_n$. Les documents sont alors vus comme une chaîne de blocs sur un alphabet $\sum = \{0,1\}^{8N}$ où N est la taille d'un bloc en nombre d'octets. Ces systèmes cryptographiques sont parfois définis comme des modes opératoires sur ces blocs.

Opérations sur les documents. Nous rappelons que $\mathcal{M}_B$ est l'espace des modifications possibles au niveau bloc, défini pour les systèmes cryptographiques incrémentaux au niveau bloc. Nous redéfinissons légèrement les notations vues en section 4 pour que les principales opérations de modification $M \in \mathcal{M}_B$ soient vues comme des opérations plus générales $M = (\text{blocks\_substitute}, i, j, \beta)$ qui substituent les blocs $i+1$ jusqu'à $j-1$ (inclus) par β, une chaîne éventuellement vide constituée d'un nombre entier de blocs. Dans le cas particulier où $j = i+1$, l'opération consiste à insérer β entre les deux blocs consécutifs d'indices i et $i+1$.

Système cryptographique incrémental au niveau bloc. Selon la définition générique vue en section 4, un système cryptographique incrémental est spécifié par un 4-uplet d'algorithmes que nous notons $\Pi = (\mathcal{K}, \mathcal{T}, \mathcal{I}, \mathcal{C})$ où $\mathcal{K}$, $\mathcal{T}$, $\mathcal{I}$ et $\mathcal{C}$ sont respectivement les algorithmes de génération de clé(s), de transformation, de mise à jour incrémentale et de transformation inverse. Les quatres algorithmes d'un système cryptographique incrémental Π pourront également être notés, selon le contexte dans lequel ils se placent, $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{I}, \mathcal{D})$ pour un système de chiffrement incrémental et $\Pi = (\mathcal{K}, \mathcal{S}, \mathcal{I}, \mathcal{V})$ pour un système de signature incrémental.

Nous pouvons distinguer plusieurs types de cryptosystèmes incrémentaux : (i) **dans le cas asymétrique** : les systèmes de chiffrement à clé publique (ou bien les systèmes de signature numérique) pour lesquels la clé de déchiffrement K'' (resp.

la clé de signature K') est gardée secrète, alors que la clé de chiffrement K' (resp. la clé de vérification K'') est publique ; **(*ii*) dans le cas symétrique** : les modes de chiffrement, les systèmes de signatures symétriques (ou codes d'authentification), ou encore les systèmes de chiffrement authentifiés qui combinent chiffrement et authentification, pour lesquels la clé $K''(= K')$ est gardée secrète.

La définition fournie est en fait très simplifiée. Au regard de certains systèmes de signature numérique *avec appendice*, la forme cryptographique t devrait normalement être interprétée comme un couple d'éléments (D, t') où D est le document et t' est l'appendice de la signature. Ici, dans le cas d'un système de signature asymétrique (resp. symétrique) nous désignerons par t l'appendice de signature (resp. le tag).

Comme nous l'avons vu précédemment, un système de chiffrement incrémental doit satisfaire une propriété d'indistinguabilité des chiffrés et des modifications (sous certaines contraintes) et un système de signature incrémental doit satisfaire une propriété d'infalsifiabilité. Nous renvoyons le lecteur à la section 3 concernant la propriété qui nous intéresse le plus ici, celle de l'inconscience d'une modification. Nous dirons d'un système satisfaisant cette propriété qu'il est *parfaitement privé*.

7.2. Systèmes cryptographiques incrémentaux au niveau octet

S'il existe des algorithmes cryptographiques de mise à jour incrémentale supportant des opérations telles que des insertions, avec un coût proportionnel à la quantité de données à insérer (ou logarithmique en la taille du document), et des suppressions avec un coût constant, cette complexité n'est valide que lorsque la chaîne d'octets résultante (le document modifié D') n'a pas de problème d'alignement, c'est-à-dire lorsque la taille de la donnée à insérer ou à supprimer est un multiple de la taille d'un bloc.

Nous illustrons des solutions en prenant comme exemple un système de chiffrement incrémental au niveau bloc. Une façon simple d'obtenir l'incrémentalité au niveau octet avec un tel système, est de n'utiliser qu'un seul octet par bloc mais cette faible charge utile a pour conséquence un coût excessif pour l'expansion du chiffré (en taille) et pour le nombre de chiffrements par blocs à effectuer lors du chiffrement initial, et dans une moindre mesure lors d'une mise à jour. Une autre solution est d'autoriser l'insertion d'une taille variable de données sans assurer la contiguïté. En effet, un bloc clair pourrait être partiellement utilisé avant d'être chiffré, ce qui éliminerait le besoin de rechiffrer tous les blocs consécutifs puisque les données ne sont pas réalignées. Une telle solution est utilisée dans (Huang, Evans, 2011), dans lequel un document est divisé en plusieurs parties dont les tailles suivent une distribution géométrique bornée. Le document partitionné est chiffré grâce au système de chiffrement authentifié RPC (Buonanno *et al.*, 2002). Pour exploiter ce dernier, chaque bloc (ou partie) de longueur variable est encapsulé dans la partie utilisable (la charge utile) d'un bloc chiffré avec RPC. La borne sur le nombre d'octets à insérer correspond à la taille de cette charge utile. Cette méthode n'est cependant pas sans défaut. Premièrement, l'absence de bourrage standard pour chaque bloc encapsulé rend le système

final non sûr. Ensuite, la borne supérieure pour la distribution pourrait être choisie arbitrairement de sorte que nous trouvions un compromis entre l'expansion du chiffré, le nombre total de chiffrements (resp. déchiffrement) par bloc pour l'algorithme de chiffrement (resp. déchiffrement) et le nombre de chiffrements par bloc pour l'algorithme de mise à jour incrémentale. Finalement, le défaut majeur est que l'algorithme de mise à jour ne prend pas en compte le réajustement de la distribution. C'est d'autant plus problématique que les insertions sont fréquentes et ne concernent que des petites quantités de données.

Un système cryptographique incrémental au niveau octet conçu selon notre paradigme doit assurer : (i) La propriété d'inconscience des opérations effectuées puisque des tests statistiques pourraient révéler des régions aberrantes dans la suite des parties de longueurs variables. Nous saurions donc si une forme cryptographique est issue d'une suite de mises à jour ; (ii) Comparativement à la taille du message, la préservation (en moyenne) du surcoût (en %) pour la taille de la forme cryptographique et le nombre d'opérations à effectuer lorsque nous appliquons l'algorithme conjugué.

PROPRIÉTÉ 11. — *Un système cryptographique incrémental au niveau octet conçu selon notre paradigme est parfaitement privé si :*

1. La distribution des longueurs de blocs ne dépend pas d'informations concernant les modifications effectuées. Les opérations de mises à jour sont implémentées de sorte que cette distribution soit préservée.

2. Le système incrémental par bloc utilisé pour opérer ces blocs est lui-même parfaitement privé.

Cette propriété est démontrée dans la sous-section 7.3.1.

7.2.1. Algorithme de partitionnement en blocs de longueurs variables

La chaîne d'octets D de taille S est divisée en une suite de blocs de taille variable $(B_i)_{i=0..n}$ dont les longueurs correspondantes $(u_i)_{i=0..n}$ sont générées à partir d'une même distribution de probabilité ϕ sur l'ensemble $\{1, \ldots, N\}$. De manière plus précise, étant donnée une suite $(l_i)_{i\geq 0}$ avec $l_i \xleftarrow{\phi} \{1, \ldots, N\}$ pour tout i, il existe un plus petit n tel que $\sum_{i=0}^{n-1} l_i < S$ et $\sum_{i=0}^{n} l_i \geq S$. Nous avons donc $u_i = l_i$ pour tout $i < n$, et $u_n = S - \sum_{i=0}^{n-1} l_i$.

Par commodité, nous définissons une opération $Partition$ qui prend en entrées la distribution de probabilité ϕ, le document D et retourne une forme partitionnée $\overline{D}$ (accompagnée de la suite de longueurs correspondantes). Nous supposerons que ϕ est un paramètre implicite de cette opération. Par exemple, avec une distribution uniforme discrète $\mathcal{U}([1,\ N])$ et une taille de bloc fictive N de 4 octets, un document de 19 octets pourrait être fractionné en 9 parties $(B_i)_{i=0..6}$ de longueurs $(2, 4, 3, 2, 3, 1, 4, 1, 3)$.

7.2.2. Opérations sur les documents

Pour obtenir un cryptosystème véritablement incrémental, nous définissons un espace de modifications au niveau octet $\mathcal{M}_b$. Les opérations générales à grains

fins $M \in \mathcal{M}_b$ que nous devons considérer sont $M = (\text{bytes\_substitute}, i, j, \delta)$ qui substituent les octets $i+1$ à $j-1$ (inclus) par δ, une chaine d'octets d'une longueur quelconque (possiblement vide). Considérant un document de S octets $D = b_1 b_2 \dots b_S$, une modification $(\text{bytes\_substitute}, i, j, \delta)$ peut être interprétée comme prenant la chaîne $b_1 b_2 \dots b_{i-1} b_i b_j b_{j+1} \dots b_{S-1} b_S$ et insérant δ immédiatement après l'octet d'indice i de sorte que nous obtenions le nouveau document $D' = b_1 b_2 \dots b_{i-1} b_i \delta b_j b_{j+1} \dots b_{S-1} b_S$. Cette opération permet de dériver les opérations usuelles d'édition d'un document :

– $M_1 = (delete, i, j)$ supprime les données de l'octet i à j (inclus).

– $M_2 = (insert, i, \delta)$ insère δ (non vide) entre les $i^{\text{ième}}$ et $(i+1)^{\text{ième}}$ octets.

– $M_3 = (replace, i, \delta)$ remplace par δ (non vide) les données commençant par l'octet i jusqu'à l'octet $(i + |\delta| - 1)$ (inclus).

Idéalement, un algorithme de mise à jour appliqué sur les entrées $\overline{D}$ et M retourne une partition à jour $\overline{D'}$ pour le nouveau document D'. Nous pouvons donc voir M_i ($i = 1, 2$ ou 3) comme décrivant le nom d'une fonction et de paramètres définissant une position ou une région dans la chaîne d'octets à modifier. Prenons l'exemple de M_2. Puisque nous manipulons une partition $\overline{D}$ de D, l'application de l'opération M_2 sur $\overline{D}$ suggère l'appel d'une fonction $insert(i, \delta, \overline{D})$, où $\overline{D} = partition(D)$. La sous-section 7.2.4 explique comment sont réalisées les opérations de mise à jour et donne un exemple.

7.2.3. Notations

Les principales notations utilisées sont décrites dans la table 2. $\|$ est l'opérateur de concaténation. δ désigne la chaîne d'octets à insérer. $|.|$ désigne la taille d'une donnée en nombre d'octets ou le nombre d'éléments dans une suite (ou liste). Nous définissons une fonction $f(n) = \sum_{m=0}^{n} u_m$ qui accumule les termes de u_0 à u_n. L'utilisation d'indices sentinelles $-\infty$ et $+\infty$ sont nécessaires dans le cas d'une modification au tout début ou tout à la fin du document. Nous utilisons également une fonction de bourrage pad prenant en entrée un bloc de longueur variable B_k, et le bourrant avec le nombre minimum de bits pour obtenir un bloc de taille N. Le choix de ces bits de bourrage dépendent du contexte d'utilisation. Ainsi, si nous devons l'utiliser dans un schéma de signature ou de chiffrement assurant l'authenticité, pad suppose que le bloc est de taille strictement inférieur à N et retourne le bloc bourré $B_k || 10^{8(N-u_k)-1}$ (c'est-à-dire B_k concaténé à une suite de bits commençant par le bit 1 suivi de $8(N-u_k)-1$ bits 0). Cependant, s'il doit être utilisé dans un système de chiffrement sans authentification, pad retourne le bloc bourré $B_k || 0^{8(N-u_k)}$. Dans le premier cas, le bourrage est désambiguïsant et donc réversible (il suffit de repérer et d'enlever le dernier bit à 1 suivi des 0), alors que dans le deuxième cas c'est la longueur associée u_k qui permet de dé-remplir le bloc.

Tableau 2. Notations utilisées pour manipuler les partitions

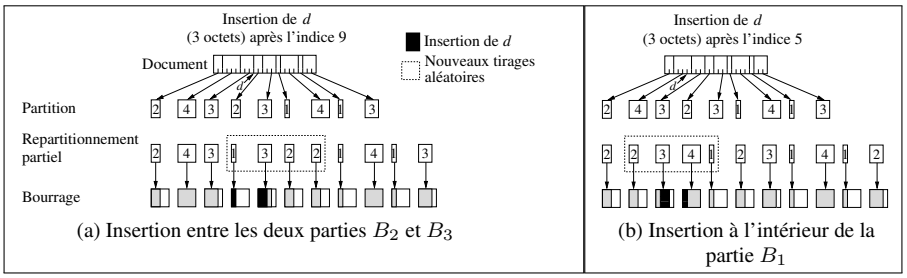
$P[a, b]$	chaîne d'octets partielle de P de l'octet a jusqu'à l'octet b (inclus)
B_i	bloc de longueur variable appartenant à la partition de D
$\overline{D} = B = (B_i)$	liste des B_i formant la partition
$u_i = \|B_i\|$	taille de B_i, en nombre d'octets
$u_i \xleftarrow{\phi} \{1, \ldots, N\}$	u_i est tiré au sort dans l'ensemble $\{1, \ldots, N\}$ selon ϕ
$u = (u_i)$	liste des longueurs u_i
$\|B\| = \|u\|$	nombre de blocs dans la partition
B'_i	bloc de longueur variable appartenant à la nouvelle sous-partition
$B' = (B'_i)$	liste des B'_i formant la nouvelle sous-partition
$u'_i = \|B'_i\|$	taille d'un B'_i, en nombre d'octets
$u' = (u'_i)$	liste des longueurs u'_i
k_0	indice d'un bloc B_{k_0} après lequel le repartitionnement commence
k_1	indice d'un bloc B_{k_1} avant lequel le repartitionnement se termine
$-\infty$, $+\infty$	indices des sentinelles, utilisés par convention pour un octet ou un bloc
$u_{-\infty}$, $u_{+\infty}$	longueurs des sentinelles, valant 0
$B_{-\infty}$, $B_{+\infty}$	blocs sentinelles, désignant une chaîne vide

7.2.4. Approche algorithmique

Le challenge est le suivant, après une opération de modification $M \in \mathcal{M}_b$, la suite résultante des longueurs (u_i) doit apparaître comme si chaque u_i (excepté le dernier) avait été tiré aléatoirement selon la distribution de probabilité ϕ.

Pour insérer une donnée δ dans le document (pour simplifier, nous considérons une insertion effectuée entre deux parties), les deux premières solutions qui viennent à l'esprit sont les suivantes : (i) Nous partitionnons δ selon le même procédé (aléatoire) utilisé pour partitionner D, c'est-à-dire en $\overline{\delta}$ et nous l'insérons au bon emplacement dans $\overline{D}$; (ii) Nous répétons des tirages aléatoires jusqu'à ce que nous trouvions une suite $(u_i)_{i=0\ldots s}$ satisfaisant $\sum_{i=0}^{s} u_j = |\delta|$, et nous insérons la partition $(B^\delta_0, B^\delta_1, \ldots, B^\delta_s)$ de δ au bon emplacement dans $\overline{D}$. Aucune de ces deux approches ne sont satisfaisantes puisqu'elles produisent un biais observable après une répétition d'insertions. Les conséquences néfastes sont la non-satisfaction de la propriété d'inconscience et d'autre part des surcoûts (en espace et en temps) non contrôlés. En effet, ces blocs de longueurs variables doivent être bourrés par un certain nombre (variable) d'octets pour que nous puissions utiliser un système incrémental au niveau bloc, et l'utilisation répétée d'une de ces deux approches ne garantit pas un taux de remplissage des blocs constant en moyenne. Notre approche consiste donc à employer une synchronisation de marches aléatoires, un pas aléatoire dans la marche correspondant à un tirage aléatoire $u_i \sim \phi$. Ainsi, nous verrons qu'effectuer une modification tout en respectant une unique distribution des longueurs mène quasi systématiquement à un

repartitionement d'une sous-chaîne d'octets inchangée du document, dont la longueur reste assez limitée.



(a) Insertion entre les deux parties B_2 et B_3

(b) Insertion à l'intérieur de la partie B_1

Figure 7. Insertion de 3 octets, $\forall i\ u_i \sim \mathcal{U}([1,\ N]),\ N = 4$

Étude de cas de l'insertion de δ à un octet localisé entre le premier octet de B_{k-1} et le dernier octet de B_k. Deux cas peuvent survenir : (i) **cas 1** : l'insertion est faite entre le dernier octet de B_{k-1} et le premier octet de B_k; (ii) **cas 2** : l'insertion est faite entre deux octets de B_k.

Soit une suite de longueurs $(u_i)_{i=0..n}$ indépendantes et identiquement distribuées (i.i.d.) selon une distribution de probabilité discrète. Le problème est de générer une suite de tirages i.i.d. $(u'_i)_{i=k..l}$ de cette distribution jusqu'à ce que nous trouvions un couple d'indices (l, m) satisfaisant l'égalité suivante

$$\sum_{j=k}^{l} u'_j = \sum_{j=k}^{m} u_j + |\delta| \,. \tag{3}$$

La suite résultante de blocs de données de longueurs variables est alors B_0, $\ldots, B_{k-1}, B'_k, \ldots, B'_l, B_{m+1}, \ldots, B_n$ où la sous-séquence de blocs $(B'_i)_{i=k..l}$ (de longueurs respectives $(u'_i)_{i=k..l}$) qui contient la donnée insérée remplace la sous-séquence $(B_i)_{i=k..m}$. Regardons l'exemple illustrant le **cas 1** dans la figure 7a: nous avons un document de 19 octets et nous voulons y insérer une donnée d de 3 octets immédiatement après le neuvième octet. Pour cela, il est fort probable que la sous-séquence de blocs $(B_j)_{j\geq 3}$ nécessite d'être repartitionnée selon notre distribution d'une manière cohérente avec l'équation 3. Comme il peut être remarqué, le premier tirage aléatoire $u'_3 = 1$ indique que B'_3 ne peut pas recevoir les 3 octets à insérer, ce qui nous amène au second tirage aléatoire $u'_4 = 3$. Si la somme des tirages $u'_3 + u'_4 = 4$ couvre l'insertion de d, un octet reste à pourvoir et il n'est pas suffisant pour les deux octets suivants de B_3, ce qui nous amène au troisième tirage $u'_5 = 2$. Une fois encore, la somme des tirages $u'_3 + u'_4 + u'_5 = 6$ couvre les cinq premiers octets $(|d| + u_3)$ de $d \| B_3$ mais un octet reste à pourvoir et est insuffisant pour les trois octets (u_4) de B_4, ce qui nous amène au quatrième tirage $u'_6 = 2$. Finalement, la somme des tirages $u'_3 + u'_4 + u'_5 + u'_6 = 8$ couvre la donnée d et le contenu de la sous-séquence de blocs (B_3, B_4), ce qui met fin à ce repartitionnement partiel. La figure 7b illustre le **cas 2**. Comme nous pouvons le voir, cette insertion fragmente B_1 et nous force ainsi à

inclure ce bloc dans le repartitionnement. Finalement la repartition partielle résultante couvre la donnée à insérer et le contenu de la précédente sous-partition (B_1, B_2).

Une suppression se ramène au cas de l'insertion. Prenons l'exemple de la suppression de 4 octets consécutifs, les deux derniers octets de B_1 et les deux premiers octets de B_2. Nous nous ramenons alors à la situation dans laquelle nous insérons une donnée de 3 octets dans la séquence comprenant 7 blocs de longueurs $(2, 2, 3, 1, 4, 1, 3)$. L'insertion de cette donnée de 3 octets se fait entre les deux premiers blocs. Dans cet exemple où $N = 4$ octets, une suppression revient donc à effectuer une insertion d'au plus 6 octets.

L'algorithme 1 retourne la sous-partie repartitionnée incluant la modification avec les indices indiquant les parties non touchées. Par exemple, une sortie (k_0, k_1, B', u') de cet algorithme signifie que la précédente partition B reste intacte de l'indice 0 à k_0 (inclus) et de l'indice k_1 (inclus) à $|u| - 1$. Par conséquent, la sous-partition $(B_{k_0+1}, \ldots, B_{k_1-1})$ est remplacée par $(B'_0, \ldots, B'_{|u'|-1})$. Nous décrivons également un algorithme Top qui traduit une opération dans $\mathcal{M}_b$ en une opération dans $\mathcal{M}_B$. Celui-ci est utilisé dans la construction générique présentée dans la sous-section 7.2.5.

7.2.5. Construction générique

Nous définissons un système cryptographique incrémental au niveau octet par un 4-uplet d'algorithmes $\Xi = (SetUp, Transformation, IncUpdate, Conjugate)$. Nous écrivons des algorithmes génériques pour la génération de clés, la transformation et la mise à jour incrémentale. Comme il ne semble pas possible d'écrire un algorithme générique pour l'opération conjuguée de la transformation, nous proposons une version qui correspond à l'opération de déchiffrement d'un système de chiffrement et un autre correspondant à l'opération de vérification d'un système de signature. Tous ces algorithmes s'appuient sur un système cryptographique incrémental au niveau bloc, noté Π, qui opère sur des chaînes sur un alphabet $\sum = \{0, 1\}^{8N}$.

Un algorithme $SetUp$ prend en entrée un paramètre de sécurité k et retourne la paire de clés générée par $\Pi.\mathcal{K}(k)$. Si celui-ci correspond à un système symétrique alors $K' = K''$, sinon $K' \neq K''$. L'algorithme 3 partitionne aléatoirement le document D en des blocs de longueurs variables B_i, bourre chacun des blocs B_i pour obtenir des blocs de taille fixe P_i, les concatène et applique l'algorithme de transformation à la chaîne de blocs résultante. La forme cryptographique retournée est alors (u, t). Notons que pour les systèmes de chiffrement, la façon de bourrer les blocs importe peu contrairement aux systèmes de signatures pour lesquels une technique de bourrage standard est obligatoire (par exemple une suite de bits commençant par un seul 1 et suivi d'un certain nombre consécutif de 0). Pour éviter les falsifications triviales, ce bourrage standard doit apparaître quelle que soit la taille du bloc. C'est pourquoi, pour éviter des répétitions de nos algorithmes, nous les présentons en supposant l'utilisation d'une fonction de bourrage correcte et d'une distribution de probabilité ϕ sur un ensemble $\{1, \ldots, N'\}$ où $N' < N$ dans le cas d'un système de signature ou $N' = N$ dans le cas d'un système de chiffrement (sans authentification).

Algorithm 1 BytesUpdate

Require: Une distribution de probabilité ϕ sur l'ensemble $\{1, \ldots, N'\}$ où $N' \leq N$, la partition B d'un document D, la liste u des longueurs correspondantes, une opération au niveau octet $M' = (\text{bytes\_substitute}, i, j, \delta)$.

1: $B' \leftarrow ()$; $u' \leftarrow ()$; $c \leftarrow |\delta|$;
2: **if** $i = -\infty$ **then**
3: $k_0 \leftarrow -\infty$; go to step 10;
4: **if** $i = |D|$ **then**
5: $k_0 \leftarrow |u| - 1$; $k_1 \leftarrow +\infty$; go to step 15;
6: $k_0 \leftarrow \text{argmin}_n \left(f(n) = \sum_{m=0}^{n} u_m | f(n) \geq i\right)$;
7: $l \leftarrow f(k_0) - i$;
8: **if** $l > 0$ **then**
9: $\delta \leftarrow B_{k_0}[1, u_{k_0} - l] \| \delta$; $c \leftarrow c + u_{k_0} - l$; $k_0 \leftarrow k_0 - 1$;
10: $k_1 \leftarrow \text{argmin}_n \left(f(n) = \sum_{m=0}^{n} u_m | f(n) \geq j - 1\right)$;
11: $l \leftarrow f(k_1) - j + 1$;
12: **if** $l > 0$ **then**
13: $\delta \leftarrow \delta \| B_{k_1}[u_{k_1} - l + 1, u_{k_1}]$; $c \leftarrow c + l$;
14: $k_1 \leftarrow k_1 + 1$
15: $x \xleftarrow{\phi} \{1, \ldots, N'\}$;
16: **if** $x = c$ **then**
17: $u' \leftarrow (u', x)$; $B' \leftarrow (B', \delta)$; go to step 27;
18: **if** $x < c$ **then**
19: $c \leftarrow c - x$; $u' \leftarrow (u', x)$; $B' \leftarrow (B', \delta[1, x])$;
20: $\delta \leftarrow \delta[x + 1, c]$; go to step 15;
21: **else**[$x > c$]
22: **if** $k_1 = |u|$ **then**
23: $u' \leftarrow (u', c)$; $B' \leftarrow (B', \delta)$; $k_1 = +\infty$;
24: go to step 27;
25: $c \leftarrow c + u_{k_1}$; $\delta \leftarrow \delta \| B_{k_1}$; $k_1 \leftarrow k_1 + 1$;
26: go to step 16;
27: **return** (k_0, k_1, B', u');

L'algorithme 4 décrit l'opération de déchiffrement d'un système de chiffrement. Il déchiffre t et récupère la chaîne de blocs bourrés P_i, enlève les bourrages pour obtenir les blocs B_i et finalement concatène ces derniers pour reconstruire le document D.

L'algorithme 5 décrit l'opération de vérification d'un système de signature. Il partitionne D selon une suite de longueurs prédéfinie u, bourre chacun des blocs de longueurs variables en des blocs de taille fixe et concatène ces derniers pour obtenir P. La validité de la signature t vis-à-vis de la chaîne P est alors contrôlée. Si elle est valide, l'algorithme retourne le document D, sinon $\perp$.

Algorithm 2 Top

Require: Une distribution de probabilité ϕ, une opération $M' \in \mathcal{M}_b$, la partition B d'un document D, la liste u des longueurs correspondantes

Ensure: Une opération $M \in \mathcal{M}_B$, la liste des longueurs u' des parties repartitionnées

1: $P \leftarrow$ "";
2: $(k_0, k_1, B', u') \leftarrow BytesUpdate(\phi, B, u, M')$;
3: **for** $i = 0$ to $|u'| - 1$ **do**
4: $\quad P_i \leftarrow pad(B'_i[1, u'_i])$; $P \leftarrow P \| P_i$;
5: $M \leftarrow$ (blocks_substitute, k_0, k_1, P);
6: **return** (M, u');

L'algorithme 6 décrit l'opération de mise à jour incrémentale. Il convertit une opération de modification au niveau octet en une opération au niveau bloc compatible avec le système cryptographique sous-jacent. La suite de longueurs u' qui est récupérée correspond à la repartition partielle qui prend en compte la modification. Les indices d'exclusion x_2 and x_3 sont alors récupérés : (*i*) x_2 correspond à l'indice jusqu'auquel la partition originale u est inchangée; (*ii*) de la même façon, x_3 correspond à l'indice à partir duquel la partition est inchangée. La forme cryptographique t est mise à jour en t' selon l'opération au niveau bloc, en utilisant $\Pi.\mathcal{I}$. La partition de longueurs u est mise à jour en v (entre les indices exclus x_2 et x_3). La forme cryptographique retournée par l'algorithme est alors (v, t').

Require: Une distribution de probabilité ϕ, un document D, une clé K'
$(u, \overline{D}) \leftarrow Partition(\phi, D)$;
for $i = 0$ to $|u| - 1$ **do**
$P_i \leftarrow pad(B_i)$;
$P \leftarrow P_0||P_1|| \ldots ||P_{|u|-1}$;
$t \leftarrow \Pi.\mathcal{T}(K', P)$;
return (u, t);

Algorithm 3 – Transformation

Require: La forme cryptographique (u, t), une clé K''
$P \leftarrow \Pi.\mathcal{D}(K'', t)$;
for $i = 0$ to $|u| - 1$ **do**
$B_i \leftarrow P_i[1, u_i]$;
$D \leftarrow B_0||B_1|| \ldots ||B_{|u|-1}$;
return D;

Algorithm 4 – Conjugate (Déchiffrement d'un système de chiffrement)

Require: Un document D, la forme cryptographique (u, t), une clé K''
$m \leftarrow 1$;
for $i = 0$ to $|u| - 1$ **do**
$P_i \leftarrow pad(D[m, m + u_i - 1])$;
$m = m + u_i$;
$P \leftarrow P_0||P_1|| \ldots ||P_{|u|-1}$;
if $\Pi.\mathcal{V}(K'', P, t) = P$ **then**
return D;
else
return $\perp$;

Algorithm 5 – Conjugate (Vérification d'un système de signature)

Require: Une distribution ϕ, une forme cryptographique (u, t), une opération de modification $M' \in \mathcal{M}_b$, la partition $\overline{D}$, une clé K'
$(M, u') \leftarrow \mathrm{Top}(\phi, M', \overline{D}, u)$;
$(x_1, x_2, x_3, x_4) \leftarrow M$;
$t' \leftarrow \Pi.\mathcal{I}(K', t, M)$;
$v \leftarrow (u', (u_i)_{i=x_3..|u|-1})$;
$v \leftarrow ((u_i)_{i=0..x_2}, v)$;
return (v, t');

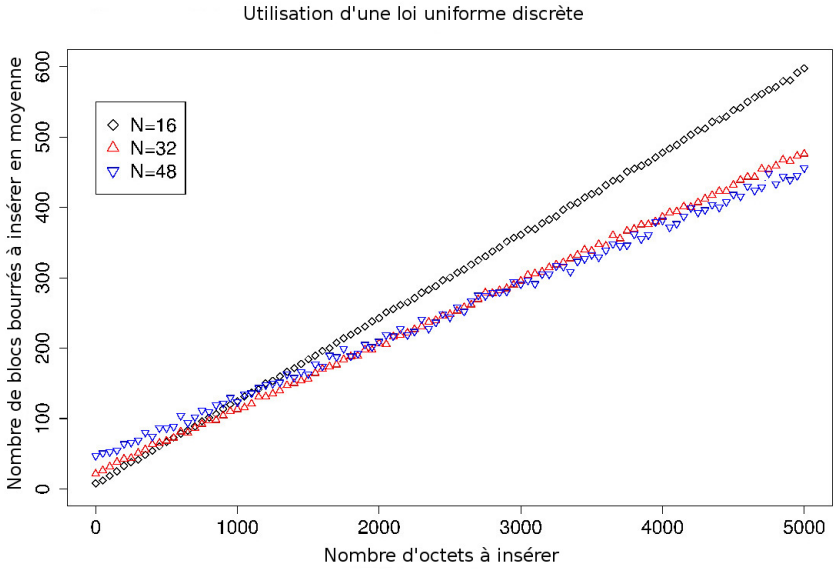
Algorithm 6 – IncUpdate

Figure 8. Algorithmes de Ξ

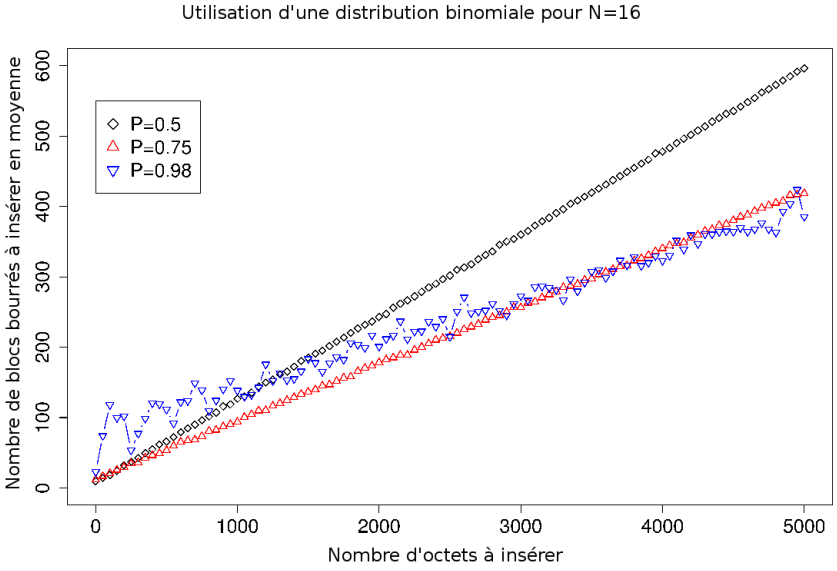
7.2.6. Analyse de l'efficacité

Analyse empirique. Le choix de la loi ϕ et de ses paramètres dépendent certainement de la configuration de l'application cible. À première vue, nous pourrions choisir les paramètres qui limitent l'expansion de la forme cryptographique sans nous soucier de leurs effets sur le coût calculatoire des opérations de mise à jour, de transformation ou de son opération inverse. Pour obtenir un aperçu du temps d'exécution d'une mise à jour, des simulations ont été effectuées en supposant l'utilisation d'un système incrémental au niveau bloc opérant sur des blocs de 16 octets. Un tel choix de taille de bloc peut correspondre à un algorithme de chiffrement symétrique, comme par exemple rECB (Buonanno *et al.*, 2002) instancié avec AES. Nous voudrions estimer de façon adéquate les performances d'un système incrémental étendu selon notre méthode. Les observations statistiques (Figures 9a, 9b) montrent le nombre moyen de blocs à insérer en fonction de la taille de la donnée à insérer, en raison de la traduction d'une opération $M' \in \mathcal{M}_b$ en une opération $M \in \mathcal{M}_B$.

Trois distributions discrètes (uniforme, binomiale et géometrique) peuvent convenir pour le choix d'une longueur variable u_i. Nous pouvons remarquer que pour une



(a) $\forall\, i\ u_i \sim \mathcal{U}([1,\ N])$



(b) $\forall\, i\ u_i \sim \mathcal{B}(N-1, p) + 1$

Figure 9. Coût de l'insertion

insertion, le temps d'exécution moyen de l'algorithme de mise à jour semble être linéaire en le nombre d'octets à insérer. Par ailleurs, comme nous pouvions nous y attendre, les plages d'optimalité sont différentes et dépendent des paramètres choisis. Par exemple, en supposant que la taille u_i suit une loi binomiale, les résultats de la figure 9b suggèrent que pour des insertions de moins de 3 Méga-octets, le nombre de blocs insérés est plus intéressant lorsque P=0.75 que lorsque P=0.98.

Puisque le choix des paramètres de la distribution de probabilité utilisés pour un document est définitif, nous devons connaître le comportement de l'application avant de faire ce choix, particulièrement au regard des statistiques sur les tailles des insertions si le but est d'optimiser les ressources calculatoires. En effet, pour des insertions concernant en moyenne des données de petite taille, un taux de remplissage plus bas pour les blocs en entrées du système incrémental au niveau bloc correspondra à un plus faible nombre de blocs insérés. Les ressources de stockage peuvent aussi être optimisées lorsque nous avons un taux de remplissage des blocs élevé, mais en contrepartie de performances moins bonnes pour l'insertion de petites données. Ces ressources de stockage sont toujours une préoccupation, quel que soit le type de cryptographie utilisée. En effet, plus spécifiquement pour le chiffrement incrémental, un taux de remplissage bas correspond à un bourrage des blocs important. Mais cela l'est aussi pour des signatures. Si nous prenons l'exemple d'un système de signature incrémentale "tamper-proof" basée sur une structure de données équilibrée comme l'Oblivious Tree, le nombre de nœuds de l'arbre auxquels sont attachés des valeurs cryptographiques est élevé lorsque le taux de remplissage est bas. Enfin, un dernier élément à prendre en compte, mais dans une moindre mesure, est la liste des longueurs u qui prend plus d'espace mémoire quand le taux de remplissage est bas.

Modélisation. Plus précisément, soient $(X_i)_{i\geq 1}$ et $(Y_i)_{i\geq 1}$ des suites de variables aléatoires i.i.d. prenant leurs valeurs dans $\{1,\ldots,N'\}$ selon la distribution ϕ. Nous définissons les marches aléatoires S_n et T_m telles que $S_n = \sum_{i=1}^{n} X_i$ et $T_m = \sum_{i=1}^{m} Y_i$ et le sous-ensemble aléatoire Z de $\mathbb{N}^2$ tel que $Z = \{(n,m) \in \mathbb{N}^2 \,;\, S_n - T_m = c\}$ où $c \in \mathbb{N}$. Notons que c représente le nombre d'octets contigus à insérer. Si (n,m) et (n',m') sont distincts dans Z alors soit $n < n'$ et $m < m'$ ou $n' < n$ et $m' < m$. En d'autres termes $n \leq n'$ et $m' \leq m$ mènent à $(n,m) = (n',m')$. En effet, si $n \leq n'$ et $m' \leq m$ alors $\sum_{i=1}^{n} X_i = \sum_{i=1}^{m} Y_i + c$ et $\sum_{i=1}^{n'} X_i = \sum_{i=1}^{m'} Y_i + c$ impliquent $\sum_{i=n+1}^{n'} X_i = -\sum_{i=m'+1}^{m} Y_i$. Par conséquent Z est de la forme $\{(n_k,m_k); k \in \mathbb{N}^*\}$ et les suites (n_k) et (m_k) sont strictement croissantes. La question est alors de déterminer la loi du couple de variables aléatoires (n_1,m_1).

Analyse algorithmique. Considérons l'algorithme décrit ci-dessous qui prend en entrée la valeur initiale c_0 pour c. D'abord, il faut noter que nous avons deux manières de terminer l'algorithme, avec les traces d'exécution (5,1,2) ou (9,2). Nous pouvons raisonner sur le nombre n_1 ou m_1 de tirages aléatoires.

Nous remarquons alors ce qui suit :

– Si $c_0 > N'$, selon l'équation de Wald (Wald, 1945) le nombre moyen d'exécutions consécutives de l'étape 3 est borné supérieurement par $c_0/\mathbb{E}(X)$, après quoi

1: $X \xleftarrow{\phi} \{1, \ldots, N'\}$;
2: **if** $X = c$ **then** stop;
3: **if** $X < c$ **then**
4: $\quad c \leftarrow c - X$;
5: $\quad$ go to step 1;
6: **if** $X > c$ **then**
7: $\quad Y \xleftarrow{\phi} \{1, \ldots, N'\}$;
8: $\quad c \leftarrow c + Y$;
9: $\quad$ go to step 2;

nous avons $c \leq N'$.

– Si $c_0 \leq N'$, chaque fois que l'étape 1 (ou 7) est exécutée nous avons $c \leq N'$ et donc une probabilité non-nulle $P(x = c)$ (ou $P(y = X - c)$ respectivement) de terminer. Supposons que ϕ soit une loi uniforme. Si nous notons d_1 le nombre total de tirages aléatoire ($d_1 = n_1 + m_1$) alors $\mathbb{E}(n_1) \leq \mathbb{E}(d_1)$. Il se trouve que d_1 suit une loi géométrique de paramètre $p = \frac{1}{N'}$, et par conséquent nous avons le système d'équations :

$$\begin{cases} \mathbb{E}(n_1) + \mathbb{E}(m_1) = N' \\ \mathbb{E}(n_1) - \mathbb{E}(m_1) \leq \frac{2N'}{N'+1} \end{cases}$$

de sorte que $\mathbb{E}(n_1) \leq 1 + N'/2$. Des bornes supérieures sont également possibles pour les autres distributions citées précédemment avec une approche plus pessimiste.

En supposant une distribution uniforme et $c_0 > N'$, soit d_0 le nombre de tirages aléatoires nécessaires pour satisfaire le prédicat $c \leq N'$ pour la première fois et d_1 le nombre de tirages aléatoires nécessaires pour terminer l'algorithme depuis la première satisfaction de ce prédicat. Par conséquent, nous pouvons borner supérieurement $\mathbb{E}(n_1)$ comme suit :

$$\mathbb{E}(n_1) \leq \mathbb{E}(d_0) + \mathbb{E}(d_1) \leq \frac{2c_0}{1 + N'} + \frac{N'}{2} + 1.$$

Si nous considérons qu'une insertion peut prendre place à l'intérieur d'un bloc alors nous pouvons borner supérieurement le premier moment (la moyenne) de n_1 par une fonction affine $\frac{2c_0}{1+N'} + \frac{N'}{2} + 3$. Par conséquent le temps d'exécution moyen AT_{Ξ}^{inc} pour insérer une chaîne d'octets δ avec le système étendu peut être borné supérieurement par une fonction de $|\delta|$ et de $|D|$:

$$AT_{\Xi}^{inc}(\delta, |D|) \leq (a\,|\delta| + b) T_{\Pi}^{inc}(1, (|D| + |\delta|)/m)$$

où a et b sont des constantes, $T_{\Pi}^{inc}(1, (|D| + |\delta|)/m)$ est le temps d'exécution pour insérer un seul bloc (dans un document totalisant $(|D| + |\delta|)/m$ blocs) en utilisant le système incrémental au niveau bloc, avec $m = \mathbb{E}(X)$. Nous laissons le lecteur remarquer que le temps d'exécution moyen d'une suppression est borné supérieurement par

$a' T_{\Pi}^{inc}(1, (|D| + |\delta|)/m)$, où a' est une constante. La table 3 résume l'intérêt de notre construction appliquée aux différents systèmes vus précédemment. Les complexités des mises à jour sont ici asymptotiques. Nous remarquons que le système étendu Ξ rend efficace un plus grand nombre de mises à jour, tout en conservant la propriété de transparence.

Tableau 3. Apports de la solution Ξ appliquée à différents systèmes satisfaisant une propriété de transparence des mises à jour. Nous supposons que le document est de longueur n blocs et que la taille d'un bloc dans le système Π est de N octets. Les complexités se réfèrent au nombre de blocs (n) ou à la quantité de blocs modifiés (k), voire les deux, et sont des valeurs moyennes asymptotiques. En ce qui concerne les systèmes incrémentaux par bloc, nous supposons que nous utilisons l'algorithme de transformation initiale lorsque l'utilisation de l'algorithme de mise à jour n'est pas possible.

Système	Transparence d'une opération	Insertion de δ avec $\lvert\delta\rvert = kN$	Insertion de δ avec $(k-1)N < \lvert\delta\rvert < kN$ et $k << n$	Suppression de σ octets avec $\lvert\sigma\rvert = kN$	Suppression de σ octets avec $(k-1)N < \lvert\sigma\rvert < kN$
$\Pi :=$ RPC, rECB ou XOR	✓	k	n	1	n
Extension avec Ξ	✓	k	k	1	1
$\Pi :=$ *Oblivious Tree*	✓	$k + \log n$	n	$\log n$	n
Extension avec Ξ	✓	$k + \log n$	$k + \log n$	$\log n$	$\log n$

L'espace de stockage moyen requis pour la suite u dépend du choix de la distribution et est borné supérieurement par $\frac{|D|+N'}{\mathbb{E}(X)} \left\lceil \frac{\log N'}{8} \right\rceil$. Cela représente, même pour une distribution simple telle que $\mathcal{U}([1,\ N'])$, un petit surcoût de stockage.

Dans un souci de clarté et de simplicité, nous considérons que les longueurs de blocs sont décrites dans une simple liste u. Avec une telle structure de données le positionnement (l'accès) à un octet particulier, la mise à jour ou l'ajout d'une longueur dans la liste se font en $\mathrm{O}\big(|u|\big)$ additions. Bien que ces additions soient négligeables devant les évaluations d'une brique cryptographique, leur nombre pourrait se faire sentir pour des documents de grandes tailles. La meilleure solution est d'accumuler partiellement ces longueurs et d'organiser les valeurs obtenues dans une structure arborescente équilibrée, de telle manière que la longueur attachée à un nœud soit égale à la somme des longueurs attachées à ses fils. Dans ce cas, les opérations d'accès à un octet, de mise à jour ou d'ajout d'une longueur se font toutes en $\mathrm{O}\big(\log |u|\big)$ opérations de comparaison et d'addition. Enfin, pour ne pas rompre la propriété d'*indépendance de l'historique* qui nous intéresse, cette structure de données doit la satisfaire éga-

lement, son choix pouvant s'orienter vers un *Oblivious Tree* (Micciancio, 1997), un *Treap* (Aragon, Seidel, 1989), ou une skip-liste (Pugh, 1990).

7.3. Sécurité du système étendu

La méthode que nous avons définie permet d'étendre un système cryptographique incrémental au niveau bloc en une version autorisant des modifications au niveau de l'octet. Nous allons maintenant montrer que cette méthode préserve la propriété d'inconscience d'une modification, et nous montrerons ensuite qu'elle préserve le secret du document, le secret des données modifiées et enfin l'infalsifiabilité.

7.3.1. Inconscience

Par souci de simplification, nous introduisons deux nouvelles fonctions qui ne sont utiles que dans la démonstration fournie ci-après :

– $PPartition$ qui prend en entrée un document, le partitionne aléatoirement, et bourre chacune des parties (avec Pad) de sorte qu'on obtienne une suite de blocs de taille fixe.

– $PPupdate$ qui prend en entrée une partition issue de $PPartition$ et une opération $M_b \in \mathcal{M}_b$, et retourne une partition mise à jour (dont chaque partie est bourrée).

Esquisse d'une démonstration pour la propriété 11. Soit $M_b \in \mathcal{M}_b$ une opération au niveau octet permettant d'obtenir le document D' à partir de D. D'après le point (1), $PPartition(D')$ est indistinguable de $PPupdate(PPartition(D), M_b)$. Par suite, $\Pi.\mathcal{T}(PPartition(D'))$ est indistinguable de $\Pi.\mathcal{T}(PPupdate(PPartition(D), M_b))$. Notons alors $M_B \in \mathcal{M}_B$ l'opération au niveau bloc permettant d'obtenir $PPupdate(PPartition(D), M_b)$ à partir de $PPartition(D)$. D'après le point (2), nous avons que $\Pi.\mathcal{T}(PPupdate(PPartition(D), M_b))$ est indistinguable de $\Pi.\mathcal{I}(PPartition(D), M_B)$. Par transitivité, $\Pi.\mathcal{T}(PPartition(D'))$ est donc indistinguable de $\Pi.\mathcal{I}(PPartition(D), M_B)$. □

En supposant que le système incrémental par bloc sous-jacent satisfasse la propriété d'inconscience, les sorties de $\Xi.IncUpdate$ et $\Xi.Transformation$ sont toujours parfaitement indistinguables si la proposition suivante est vraie :

PROPOSITION 12. — *Pour tout $D = b_1 ... b_S$ où$(b_i)_{i=1..S}$ sont les octets ordonnés de D et $|\delta| = c$, nous avons les assertions suivantes :*

– $insert(i, \delta, partition(D))$ *et* $partition(b_1 b_2 \dots b_i \delta b_{i+1} \dots b_S)$ *retournent une partition d'une même distribution de probabilité.*

– $delete(i, j, partition(D))$ *et* $partition(b_1 b_2 \dots b_{i-1} b_{j+1} \dots b_S)$ *retournent une partition d'une même distribution de probabilité.*

– $replace(i, \delta, partition(D))$ *et* $partition(b_1 b_2 \dots b_{i-1} \delta b_{i+c} \dots b_S)$ *retournent une partition d'une même distribution de probabilité.*

PREUVE. — Prenons la première assertion, et considérons la partition $\overline{D}$ constituée des parties $(B_i)_{i=1\dots n}$ de longueurs $(u_i)_{i=1\dots n}$. Supposons qu'en sortie de $insert(i, \delta, \overline{D})$, nous ayons une nouvelle sous-partition dont la suite des longueurs est $(u'_j)_{j=k\dots l}$, avec $u_i \xleftarrow{\phi} \{1, \dots, N\}$ à l'exception éventuellement du dernier terme u'_l si la synchronisation n'a pas lieu. Nous avons donc deux cas de figure :

– **La synchronisation se produit.** Cette sous-partition accueille la donnée δ et une partie des données de D, celles de la sous-partition $(B_j)_{j=k\dots m}$. Considérons les sorties de $insert(i, \delta, \overline{D})$ et de $partition(b_1 b_2 \dots b_i \delta b_{i+1} \dots b_S)$. Nous avons dans le premier cas, en raison de l'équation (3)

$$\sum_{j=k}^{l} u'_j = \sum_{j=k}^{m} u_j + c,$$

satisfaite par notre processus de mise à jour, une suite de termes générés selon une même distribution de probabilité. Le dernier terme u_n de la partition totale du document dépend, de la même façon dans les deux cas, d'un dernier tirage aléatoire, de la longueur finale S' du document et de la somme des précédents termes générés. C'est clair pour le deuxième cas, et ça l'est aussi pour le premier cas. En effet, le raisonnement se fait toujours sur les longueurs des parties. Regardons d'abord le résultat $\overline{D}$ du partitionnement de D. Nous considérons donc une suite de réalisations $(l_i)_{i\geq 1}$ avec $l_i \xleftarrow{\phi} \{1, \dots, N\}$ pour tout i. Il existe un plus petit n tel que $\sum_{i=1}^{n-1} l_i < S$ et $\sum_{i=1}^{n} l_i \geq S$. Nous avons donc $u_i = l_i$ pour tout $i < n$, et $u_n = S - \sum_{i=1}^{n-1} l_i = S - \sum_{i=1}^{n-1} u_i$. Une fois l'insertion effectuée, le dernier terme se réécrit

$$\begin{aligned} u_n = S - \sum_{i=1}^{n-1} u_i &= S - \sum_{i=1}^{n-1} u_i \\ &= S' - \sum_{i=1}^{k-1} u_i - \sum_{i=k}^{l} u'_i - \sum_{i=m+1}^{n-1} u_i \end{aligned}$$

où tous les termes $(u_i)_{i=1\dots k-1}$, $(u'_i)_{i=k\dots l}$ et $(u_i)_{i=m+1\dots n-1}$ sont effectivement distribués i.i.d. selon la même loi.

– **La synchronisation ne se produit pas.** Le processus de repartitionnement de l'opération $insert(i, \delta, \overline{D})$ atteint la fin du document. Le dernier terme de la partition totale du document dépend toujours d'un tirage aléatoire, de la longueur finale du document et de la somme des précédents termes.

Par conséquent, les distributions des sorties de ces deux opérations sont indistinguables. Les mêmes observations peuvent être faites pour les deux dernières assertions. ■

En ce qui concerne l'indistinguabilité des chiffrés/mises à jour (resp. l'infalsifiabilité de la signature), la sécurité de notre système de chiffrement (resp. signature) étendu se réduit directement à la sécurité du système de chiffrement (resp. signature) incrémental par bloc sous-jacent.

7.3.2. Indistinguabilité

THÉORÈME 13. — *Supposons que* Π *soit un système de chiffrement* $(t', q_e^{\Pi}, q_i^{\Pi}, \epsilon)$*-sûr au sens* IND1-CPA*, alors* Ξ *est également un système de chiffrement* $(t, q_e^{\Pi}, q_i^{\Pi}, \epsilon)$*-sûr au sens* IND1-CPA*, avec*

$$t = t' - q_e^{\Pi} t_{part} - q_i^{\Pi} t_{up}$$

où t_{part} *et* t_{up} *sont les temps pour effectuer respectivement une partition avec le traitement par* $\Pi.\mathcal{E}_{K'}$ *de chaque partie et une mise à jour de la partition avec le traitement par* $\Pi.\mathcal{I}_{K'}$ *de chaque partie des données repartitionnées.*

PREUVE. — Premièrement, nous supposons ici que les valeurs de t_{part} et t_{up} sont bornées supérieurement puisque, d'une part, les documents (et la donnée à insérer) peuvent être de longueurs différentes et, d'autre part, les algorithmes de partitionnement et repartitionnement sont probabilistes. Supposons qu'un adversaire $\mathcal{A}$ puisse $(t, q_e^{\Xi}, q_i^{\Xi}, \epsilon)$-casser le système de chiffrement incrémental étendu Ξ, alors nous pouvons construire un adversaire $\mathcal{A}'$ qui peut $(t', q_e^{\Pi}, q_i^{\Pi}, \epsilon')$-casser le système de chiffrement incrémental sous-jacent Π. L'adversaire $\mathcal{A}'$ utilise $\mathcal{A}$ comme une sous-routine, agit comme le challenger pour $\mathcal{A}$ et simule son environnement en répondant à ses requêtes (aux oracles simulés $\Xi.\mathcal{E}_{K'}(.)$ et $\Xi.\mathcal{I}_{K'}(.,.,.)$). Pour ce faire, $\mathcal{A}'$ partitionne ou repartitionne les données lui-même, selon le cas, et applique la procédure de bourrage pour convertir une opération dans $\mathcal{M}_b$ en une opération dans $\mathcal{M}_B$. Ainsi, il peut effectuer les requêtes correspondantes à ses propres oracles ($\Pi.\mathcal{E}_{K'}(.)$ ou $\Pi.\mathcal{I}_{K'}(.,.,.)$).

À un moment donné, $\mathcal{A}$ choisit un couple de documents $d_{\mathcal{A}} = (D_1, D_2)$ de même taille et les envoie à $\mathcal{A}'$, qui les partitionne selon les mêmes tirages aléatoires [16] en des suites de blocs de longueurs variables $\overline{D_1}$ et $\overline{D_2}$, les bourre et les concatène pour obtenir les suites de blocs de taille fixe (D'_1, D'_2), stocke la liste de longueurs consécutives correspondante u et définit son choix comme étant $d_{\mathcal{D}} = (D'_1, D'_2)$.

$\mathcal{A}'$ envoie son choix à son challenger et reçoit un document chiffré C. Il transmet C accompagné de u à $\mathcal{A}$, ce dernier pouvant continuer les interactions avec les oracles simulés $\Xi.\mathcal{E}_{K'}(.)$ et $\Xi.\mathcal{I}_{K'}(.,.,.)$. Finalement, $\mathcal{A}$ devine $b \in \{0,1\}$ et $\mathcal{A}'$ renvoie le même résultat. ■

THÉORÈME 14. — *Supposons que* Π *soit un système de chiffrement* $(t', q_e^{\Pi}, q_i^{\Pi}, \epsilon)$*-sûr au sens* IND2-CPA*, alors* Ξ *est également un système de chiffrement* $(t, q_e^{\Pi}, q_i^{\Pi}, \epsilon)$*-sûr au sens* IND2-CPA*, avec*

$$t = t' - q_e^{\Pi} t_{part} - q_i^{\Pi} t_{up}$$

16. Le second document est partitionné exactement comme le premier.

où t_{part} et t_{up} sont les temps pour effectuer respectivement une partition avec le traitement par $\Pi.\mathcal{E}_{K'}$ de chaque partie et une mise à jour de la partition avec le traitement par $\Pi.\mathcal{I}_{K'}$ de chaque partie des données repartitionnées.

PREUVE. — Premièrement, nous supposons ici que les valeurs de t_{part} et t_{up} sont bornées supérieurement puisque, d'une part, les documents (et la donnée à insérer) peuvent être de longueurs différentes et, d'autre part, les algorithmes de partitionnement et repartitionnement sont probabilistes. Supposons qu'un adversaire $\mathcal{A}$ puisse $(t, q_e^{\Xi}, q_i^{\Xi}, \epsilon)$-casser le système de chiffrement incrémental étendu Ξ, alors nous pouvons construire un adversaire $\mathcal{A}'$ qui peut $(t', q_e^{\Pi}, q_i^{\Pi}, \epsilon')$-casser le système de chiffrement incrémental sous-jacent Π. L'adversaire $\mathcal{A}'$ utilise $\mathcal{A}$ comme une sous-routine, agit comme le challenger pour $\mathcal{A}$ et simule son environnement en répondant à ses requêtes (aux oracles simulés $\Xi.\mathcal{E}_{K'}(.)$ et $\Xi.\mathcal{I}_{K'}(.,.,.)$). Pour ce faire, $\mathcal{A}'$ partitionne ou repartitionne les données lui-même, selon le cas, et applique la procédure de bourrage pour convertir une opération dans $\mathcal{M}_b$ en une opération dans $\mathcal{M}_B$. Ainsi, il peut effectuer les requêtes correspondantes à ses propres oracles ($\Pi.\mathcal{E}_{K'}(.)$ ou $\Pi.\mathcal{I}_{K'}(.,.,.)$).

À un moment donné, $\mathcal{A}$ choisit un chiffré (u, C) (issu de ses précédentes requêtes) avec un couple de modifications $(M_1, M_2) \in \mathcal{M}_b^2$ et les envoie au challenger simulé $\mathcal{A}'$. Nous supposons que $\mathcal{A}'$ a stocké tous les textes clairs issus des requêtes. Il effectue alors le même repartitionnement pour les deux modifications, bourre les parties correspondantes et les concatène pour obtenir un couple de modifications $(M_1', M_2') \in \mathcal{M}_B^2$. Il stocke la liste u' de longueurs correspondantes à la sous-partie repartitionnée et définit son choix comme étant $m = (M_1', M_2')$.

$\mathcal{A}'$ envoie son choix (C, m) à son propre challenger et reçoit un document chiffré C'. À partir de u et u' il déduit la liste de longueurs mise à jour v correspondante à C'. Il transmet alors (v, C') à $\mathcal{A}$, qui peut ensuite continuer les interactions avec les oracles simulés $\Xi.\mathcal{E}_{K'}(.)$ et $\Xi.\mathcal{I}_{K'}(.,.,.)$. Finalement, $\mathcal{A}$ devine $b \in \{0, 1\}$ et $\mathcal{A}'$ renvoie le même résultat. ■

7.3.3. Infalsifiabilité

THÉORÈME 15. — *Soit un $(t, q_s^{\Xi}, q_i^{\Xi}, \epsilon)$-falsificateur existentiel $\mathcal{F}$ contre le système de signature incrémental étendu Ξ, alors il existe un $(t', q_s^{\Xi}, q_i^{\Xi}, \epsilon)$-falsificateur existentiel $\mathcal{F}'$ contre le système de signature incrémental sous-jacent Π tel que :*

$$t = t' - q_s^{\Xi} t_{part} - q_i^{\Xi} t_{up}$$

où t_{part} et t_{up} sont les temps pour effectuer respectivement une partition avec le traitement par $\Pi.\mathcal{S}_{K'}$ de chaque partie et une mise à jour de la partition avec le traitement par $\Pi.\mathcal{I}_{K'}$ de chaque partie des données repartitionnées.

PREUVE. — Supposons que le falsificateur $\mathcal{F}'$ utilise $\mathcal{F}$ comme une sous-routine. $\mathcal{F}'$ simule l'environnement de $\mathcal{F}$ de la façon suivante. Chaque fois que $\mathcal{F}$ demande une signature sur le document D, la requête est redirigée vers $\mathcal{F}'$ qui partitionne lui-même D selon la distribution de probabilité définie dans le système, bourre chacune des parties de cette partition, les concatène pour obtenir P et demande une signature sur P à son

propre oracle de signature $\Pi.\mathcal{S}_{K'}(.)$. Ensuite, il transmet la signature résultante (u, s) à $\mathcal{F}$, où u est l'information auxiliaire représentant la suite de longueurs des parties. Chaque fois que $\mathcal{F}$ demande une mise à jour incrémentale sur une signature (u, t) selon une opération de modification appartenant à $\mathcal{M}_b$, la requête est transmise à $\mathcal{F}'$ qui traduit cette opération en une opération appartenant à $\mathcal{M}_B$ et demande à son propre oracle $\Pi.\mathcal{I}_{K'}(.,.,.)$ d'appliquer cette dernière. Le résultat (v, s') est alors transmis à $\mathcal{F}$. Finalement, à un moment donné, $\mathcal{F}$ trouve une nouvelle signature (w, s_n) (associée à un nouveau document D_n) qui n'a jamais été demandée à l'oracle simulé et qui ne provient pas non plus de l'oracle de mise à jour incrémentale simulé. $\mathcal{F}'$ doit juste assigner sa nouvelle signature à s_n (associée à un document P_n qu'il peut aisément déterminer à partir de D_n et w). Le couple (P_n, s_n) constitue une falsification valide. ■

7.4. Remarques sur la réduction de la taille des formes cryptographiques

Pour réduire significativement la taille des formes cryptographiques, nous ne devons plus utiliser des systèmes incrémentaux au niveau bloc comme des boites noires. Une stratégie possible dans le cas du chiffrement incrémental est la suivante. Nous partitionnons le message en parties de longueur l où l est un nombre d'octets choisi aléatoirement entre 1 et s^2, s étant la taille d'un bloc (d'un algorithme de chiffrement par bloc) en nombre d'octets. Nous choisissons s^2 comme borne à titre d'exemple. Nous chiffrons alors chaque partie à l'aide d'un système de chiffrement sans état et aléatoire (par exemple CTR ou CBC avec un IV aléatoire). Le chiffré résultant est alors une suite de parties chiffrées. Si un bit change dans une partie, il suffit de rechiffrer cette partie avec un nouvel IV. Effectuer une mise à jour plus complexe du chiffré et de manière transparente consiste alors à employer la méthode des marches aléatoires décrite ci-avant. Afin que le chiffré obtenu soit infalsifiable, nous pouvons générer un tag (par exemple avec HMAC) pour chaque partie chiffrée, et ensuite appliquer un MAC incrémental au niveau bloc sur la concaténation des tags. Pour que le système de chiffrement authentifié résultant permette des mises à jour transparentes, le MAC incrémental utilisé doit lui aussi être transparent. Nous pouvons alors utiliser le système de Micciancio (1997) basé sur un arbre dynamique, l'*Oblivious Tree*, dans lequel les feuilles de l'arbre sont les tags de chacune des parties.

La stratégie susmentionnée réduit la taille des formes cryptographiques au détriment de mises à jour moins performantes. Si nous étudions la réduction de taille que nous pouvons obtenir avec celle-ci, nous remarquons que comparativement au nombre n de blocs du texte clair, le nombre moyen de valeurs aléatoires (les IV) présents dans le chiffré est largement diminué. Puisqu'il y a en moyenne un IV tous les $s/2$ blocs, l'espace occupé par les IV est en moyenne de $2n$ octets. Cela permet donc d'obtenir une taille de chiffré (sans compter les données d'authentification) proche de $n(s + 2)$ octets. De même, puisque le nombre moyen de blocs authentifiés par tag est proche

de $s/2$, le nombre moyen de tags "internes" dans l'arbre [17] est proche de $4n/(3s)$. Par conséquent, la taille totale moyenne (IV + tags) est proche de $n(10/3+s)$ octets. Une telle taille est à comparer avec ce que nous aurions obtenu en utilisant le système RPC (avec $r = 48$, voir section 4) étendu avec notre système (section 7.2.5), à savoir une taille d'au moins $4ns$ octets.

8. Conclusion

Dans cet article, nous avons fait le tour des mécanismes employés par les systèmes cryptographiques incrémentaux et avons introduit une extension aux précédents systèmes autorisant des opérations plus réalistes, tout en préservant leur transparence. Des directions ont également été proposées pour atténuer la principale difficulté de cette cryptographie, la réduction de la taille des formes cryptographiques. Ayant montré qu'il était possible d'y remédier par simple composition de systèmes cryptographiques, il est raisonnable de penser que davantage de progrès peuvent être faits. Notamment, un problème qui doit être adressé est celui de l'impact négatif que peut avoir cette réduction sur l'efficacité des mises à jour.

Bibliographie

Aragon C. R., Seidel R. G. (1989). Randomized Search Trees. In *Proceedings of the 30th annual symposium on foundations of computer science*, p. 540–545. Washington, DC, USA, IEEE Computer Society.

Ateniese G., Chou D. H., Medeiros B. de, Tsudik G. (2005). Sanitizable signatures. In *Computer security - ESORICS 2005, 10th european symposium on research in computer security, milan, italy, september 12-14, 2005, proceedings*, p. 159–177.

Atighehchi K., Muntean T. (2013). Towards Fully Incremental Cryptographic Schemes. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, p. 505–510. New York, NY, USA, ACM.

Bellare M., Canetti R., Krawczyk H. (1996). Message Authentication using Hash Functions-The HMAC Construction. *CryptoBytes*, vol. 2.

Bellare M., Goldreich O., Goldwasser S. (1994). Incremental Cryptography: The Case of Hashing and Signing. In *Advances in Cryptology – CRYPTO'94*, vol. 839, p. 216-233. Springer Berlin Heidelberg.

Bellare M., Goldreich O., Goldwasser S. (1995). Incremental Cryptography and Application to Virus Protection. In *Proceedings of the Twenty-seventh Annual ACM Symposium on Theory of Computing*, p. 45–56. New York, NY, USA, ACM.

Bellare M., Guérin R., Rogaway P. (1995). XOR MACs: New Methods for Message Authentication Using Finite Pseudorandom Functions. In *Proceedings of the 15th Annual Inter-*

17. Pour un arbre de Merkle binaire, il y a approximativement n nœuds internes pour n feuilles. Pour un arbre 2-3, comme l'*Oblivious Tree*, leur nombre est d'approximativement $2n/3$.

national Cryptology Conference on Advances in Cryptology, p. 15–28. London, UK, UK, Springer-Verlag.

Bellare M., Micciancio D. (1997). A New Paradigm for Collision-free Hashing: Incrementality at Reduced Cost. In *Proceedings of the 16th Annual International Conference on Theory and Application of Cryptographic Techniques*, p. 163–192. Berlin, Heidelberg, Springer-Verlag.

Bellare M., Namprempre C. (2000). Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. In T. Okamoto (Ed.), *Advances in Cryptology — ASIACRYPT 2000*, p. 531–545. Berlin, Heidelberg, Springer Berlin Heidelberg.

Bernstein D. J. (2005). The Poly1305-AES Message-Authentication Code. In *Fast Software Encryption: 12th International Workshop, FSE 2005, Paris, France, February 21-23, 2005, Revised Selected Papers*, p. 32–49.

Blaze M. (1993). A Cryptographic File System for UNIX. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, p. 9–16. New York, NY, USA, ACM.

Brzuska C., Busch H., Dagdelen O., Fischlin M., Franz M., Katzenbeisser S. *et al*. (2010). Redactable Signatures for Tree-structured Data: Definitions and Constructions. In *Proceedings of the 8th International Conference on Applied Cryptography and Network Security*, p. 87–104. Berlin, Heidelberg, Springer-Verlag.

Buonanno E., Katz J., Yung M. (2002). Incremental Unforgeable Encryption. In *Revised Papers from the 8th International Workshop on Fast Software Encryption*, p. 109–124. London, UK, UK, Springer-Verlag.

Canard S., Laguillaumie F., Milhau M. (2008). TrapdoorSanitizable Signatures and Their Application to Content Protection. In *Applied Cryptography and Network Security, 6th International Conference, ACNS 2008, New York, NY, USA, June 3-6, 2008. Proceedings*, p. 258–276.

Carter J. L., Wegman M. N. (1977). Universal Classes of Hash Functions (Extended Abstract). In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, p. 106–112. New York, NY, USA, ACM.

Dworkin M. J. (2007). *SP 800-38D. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*. Rapport technique. Gaithersburg, MD, United States.

Fischlin M. (1997a). Incremental Cryptography and Memory Checkers. In *Advances in Cryptology – EUROCRYPT '97*, p. 293–408. Springer-Verlag.

Fischlin M. (1997b). Lower Bounds for the Signature Size of Incremental Schemes. In *38th Annual Symposium on Foundations of Computer Science, FOCS '97*, p. 438–447. IEEE Computer Society.

Garg S., Pandey O. (2015). Incremental Program Obfuscation. *IACR Cryptology ePrint Archive*, vol. 2015, p. 997.

Goi B.-M., Siddiqi M. U., Chuah H.-T. (2001). Incremental Hash Function Based on Pair Chaining & Modular Arithmetic Combining. In *Progress in Cryptology – INDOCRYPT 2001*, vol. 2247, p. 50-61. Springer Berlin Heidelberg.

Huang Y., Evans D. (2011). Private Editing Using Untrusted Cloud Services. In *ICDCS Workshops*, p. 263-272.

Jutla C. S. (2001). Encryption modes with almost free message integrity. In *Advances in Cryptology - EUROCRYPT 2001*, p. 529–544. Springer-Verlag.

Klonowski M., Lauks A. (2006). Extended Sanitizable Signatures. In M. S. Rhee, B. Lee (Eds.), *Information Security and Cryptology – ICISC 2006: 9th International Conference, Busan, Korea, November 30 - December 1, 2006. Proceedings*, p. 343–355. Berlin, Heidelberg, Springer Berlin Heidelberg.

Knuth D. E. (1969). *The Art of Computer Programming, Volume II: Seminumerical Algorithms*. Addison-Wesley.

McGrew D. (2005). Efficient authentication of large, dynamic data sets using Galois/Counter Mode (GCM). In *3RD INTERNATIONAL IEEE SECURITY IN STORAGE WORKSHOP (SISW'05)*, p. 89–94. IEEE.

McGrew D., Viega J. (2005). The galois/counter mode of operation (gcm).

Micciancio D. (1997). Oblivious Data Structures: Applications to Cryptography. In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, p. 456–464. New York, NY, USA, ACM.

Minder L., Sinclair A. (2012, avril). The extended k-tree algorithm. *J. Cryptol.*, vol. 25, n° 2, p. 349–382.

Mironov I., Pandey O., Reingold O., Segev G. (2012). Incremental Deterministic Public-key Encryption. In *Proceedings of the 31st Annual International Conference on Theory and Applications of Cryptographic Techniques*, p. 628–644. Springer-Verlag.

Nikolić I., Sasaki Y. (2015). Refinements of the K-tree Algorithm for the Generalized Birthday Problem. In *Proceedings, Part II, of the 21st International Conference on Advances in Cryptology — ASIACRYPT 2015 - Volume 9453*, p. 683–703. New York, NY, USA, Springer-Verlag New York, Inc.

Nisan N., Zuckerman D. (1993). Randomness is Linear in Space. *Journal of Computer and System Sciences*, vol. 52, p. 43–52.

Phan R. C. W., Wagner D. (2006, mars). Security Considerations for Incremental Hash Functions Based on Pair Block Chaining. *Comput. Secur.*, vol. 25, n° 2, p. 131–136.

Pöhls H. C., Samelin K. (2015, Aug). Accountable Redactable Signatures. In *2015 10th International Conference on Availability, Reliability and Security*, p. 60-69.

Pugh W. (1990, juin). Skip Lists: A Probabilistic Alternative to Balanced Trees. *Commun. ACM*, vol. 33, n° 6, p. 668–676.

Rogaway P., Bellare M., Black J. (2003). OCB: A block-cipher mode of operation for efficient authenticated encryption. *ACM Trans. Inf. Syst. Secur.*, vol. 6, n° 3, p. 365–403.

Wagner D. (2002). A generalized birthday problem. In *Proceedings of the 22Nd Annual International Cryptology Conference on Advances in Cryptology*, p. 288–303. London, UK, UK, Springer-Verlag.

Wald A. (1945). Some Generalizations of the Theory of Cumulative Sums of Random Variables. *The Annals of Mathematical Statistics*, vol. 16, n° 3, p. pp. 287-293.

Wang C. C., Kao M.-C., Yeh Y.-S. (2006). Forgery Attack on the RPC Incremental Unforgeable Encryption Scheme. In *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security*, p. 361–361. New York, NY, USA, ACM.

Wegman M. N., Carter L. (1979). New Classes and Applications of Hash Functions. In *20th Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 29-31 October 1979*, p. 175–182.

Wegman M. N., Carter L. (1981). New Hash Functions and Their Use in Authentication and Set Equality. *J. Comput. Syst. Sci.*, vol. 22, n° 3, p. 265–279.