RFC 9771 : Properties of AEAD Algorithms

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 8 mai 2025

Date de publication du RFC : Mai 2025

https://www.bortzmeyer.org/9771.html

Aujourd'hui, sur l'Internet, on n'utilise plus (ou, en tout cas, cela devrait être le cas) que des algorithmes de chiffrement AEAD ("*Authenticated Encryption with Associated Data*"). Ils assurent confidentialité **et** intégrité des données. Mais on peut être gourmand et vouloir en plus d'autres propriétés. Ce nouveau RFC classe ces propriétés et définit la terminologie.

Les algorithmes de chiffrement anciens, non-AEAD, assuraient la confidentialité, mais pas l'intégrité. Un méchant (ou un accident) pouvait modifier les données sans que ce soit détectable et le déchiffrement produisait ensuite un texte qui n'était pas le texte original. Bien sûr, s'il s'agissait d'un texte en langue naturelle, ou bien d'un texte dans un format structuré, l'utilisateurice pouvait parfois voir qu'il y avait eu un problème, puisque, avec certains modes de chiffrement, ielle ne récupérait que du n'importe quoi. Mais il serait souhaitable de détecter l'attaque (ou le problème) le plus tôt possible. Et il y a aussi des raisons de sécurité pour assurer confidentialité et intégrité en même temps. Sinon, certaines attaques (comme Poodle <https://www.slate.fr/story/93437/poodle-bug-caniche-peut-vous-devoiler>) sont possibles. Et il faut aussi compter avec le fait que les programmeurs peuvent se tromper s'ils doivent appliquer confidentialité et intégrité séparément (cf. RFC 7366 [1]).

Vérifier l'intégrité peut se faire avec une somme de contrôle avec clé (MAC), ou une technique équivalente mais l'AEAD fait mieux, en intégrant chiffrement et vérification de l'intégrité. Cela a plusieurs avantages, dont les performances (plus besoin de chiffrer et de calculer un MAC successivement) et, comme indiqué plus haut, la sécurité.

Dans la famille des normes de l'Internet, l'AEAD est décrit plus en détail dans le RFC 5116. À partir de la version 1.3 (normalisée dans le RFC 8446), TLS impose l'utilisation d'algorithmes AEAD. Même

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc7366.txt

chose pour QUIC (RFC 9000). Parmi les algorithmes AEAD courants, on trouve AES-GCM et ChaCha20-Poly1305 (RFC 8439) mais aussi Aegis <https://datatracker.ietf.org/doc/draft-irtf-cfrg-aegis-ae> ou OCB (RFC 7253).

Au passage, je trouve que le terme en anglais, "*authenticated encryption*" et sa traduction en français, chiffrement authentifié, sont trompeurs. Le service qu'ils assurent n'est pas l'authentification telle qu'on peut la faire avec RSA ou ECDSA. Je préférerais qu'on parle de chiffrement intègre.

Tout cela est bien connu depuis des années. Mais l'appétit vient en mangeant et certaines utilisations bénéficieraient de services supplémentaires. Les algorithmes existants sont sûrs par rapport aux modèles de menace « traditionnels » mais il en existe d'autres, auxquels tous les algorithmes AEAD ne répondent pas forcément. Ces services supplémentaires, pour répondre à ces modèles de menace, existaient parfois, et parfois ont déjà été développés, mais des équipes différentes leur donnent des noms différents. D'où le projet d'unification de la terminologie de ce RFC 9771. Il présente les propriétés supplémentaires (en plus de la confidentialité et de l'intégrité, que tout algorithme AEAD fournit), des noms d'algorithmes qui les fournissent et des cas d'usage. Le but est que les futurs RFC normalisant des protocoles qui dépendent de ces propriétés utilisent le vocabulaire de notre RFC.

Les propriétés supplémentaires, au-delà de la confidentialité et de l'intégrité basiques, sont listées dans la section 4. Il y a deux catégories :

— Propriétés de sécurité, lorsque la propriété permet de parer certaines attaques,
— propriétés de mise en œuvre, lorsqu'elles permettent de meilleures implémentations,
— et, pour, voir si vous savez compter jusqu'à deux, il y a aussi une catégorie un peu fourre-tout, pour les propriétés qui ajoutent de nouvelles fonctions, mais je n'en parlerai pas plus ici (consultez l'annexe A du RFC).

Avant d'exposer les propriétés additionnelles, le RFC présente les traditionnelles, en suivant le même schéma (définition, exemples, éventuels synonymes, cas d'usage, lectures pour approfondir) : confidentialité et intégrité sont ainsi exposées. Passons ensuite aux choses sérieuses, avec les propriétés de sécurité supplémentaires.

D'abord, la sécurité face aux changements ("*blockwise security*") : c'est le fait d'être sûr même quand un adversaire particulièrement puissant peut modifier le texte en clair en fonction de ce qui a déjà été chiffré. Tous les adversaires n'ont heureusement pas ce pouvoir mais cela peut arriver, par exemple lors du "*streaming*". Ainsi, la famille Deoxys <https://doi.org/10.1007/s00145-021-09397-w> résiste à ce type d'attaque.

Ensuite, l'engagement complet ("*full commitment*"). C'est l'extrême difficulté à trouver deux jeux {clé, numnique <https://www.bortzmeyer.org/nonce.html>, données en clair} qui donneront le même texte chiffré. Cela sert dans le "*message franking*" mais ne me demandez pas ce que c'est, lisez cet article <https://doi.org/10.1007/978-3-319-63697-9_3>. Ascon <https://doi.org/10.1007/s00145-021-09398-9> a cette propriété. Et, au passage, le AD de AEAD veut dire « données associées » ("*associated data*", des méta-données ajoutées au message) et les « données en clair » incluent donc ces données associées.

La résistance aux fuites ("*leakage resistance*") est une propriété des algorithmes cryptographiques (pas seulement AEAD) dont la sécurité ne diminue pas même si l'attaquant obtient des informations via un canal secondaire. Un exemple d'un tel canal secondaire est le temps de calcul (si l'attaquant peut le chronométrer et si l'algorithme n'est pas résistant aux fuites, l'attaquant pourra obtenir des informations sur la clé) ou bien la consommation électrique. Bien sûr, cela dépend de l'implémentation (essayer de faire tous les calculs en un temps constant, quelle que soit la clé) mais un algorithme résistant aux fuites est un algorithme qui ne dépend pas trop ("*mild requirement*", dit le RFC) de son implémentation concrète pour être sûr. Cette propriété est particulièrement importante pour les machines que l'attaquant peut

contrôler physiquement (cartes à puce, objets connectés) alors que mesurer le temps de calcul, et a fortiori la consommation électrique, d'un serveur Internet distant est une autre affaire. (Mais ça reste possible dans certains cas, notamment si l'attaquant est proche, par exemple chez le même hébergeur : voir l'article « "*Remote Timing Attacks Are Practical*" <https://www.usenix.org/conference/12th-usenix-security-remote-timing-attacks-are-practical> » ou bien la faille Hertzbleed <https://www.hertzbleed.com/>.)

La sécurité multi-utilisateurs ("*multi-user security*") est atteinte quand la sécurité décroit moins vite que linéairement avec le nombre d'utilisateurs (les algorithmes ont des limites quantitatives, cf. draft-irtf-cfrg-aead-
C'est indispensable pour des protocoles comme TLS, où plusieurs « utilisateurs » peuvent se retrouver sur la même session TLS. Des algorithmes comme AES-GCM ou ChaCha20-Poly1305 ont cette sécurité (si on ne dépasse pas les limites). Vous pouvez lire « "*The Multi-user Security of Authenticated Encryption : AES-GCM in TLS 1.3*" <https://link.springer.com/10.1007/978-3-662-53018-4_10> » pour les détails.

Bien sûr, il fallait mentionner la propriété de résistance aux calculateurs quantiques ("*Quantum security*"). AEAD ou pas, un algorithme est résistant au quantique si on ne connait pas d'algorithme pour calculateur quantique capable de casser cet algorithme. Les algorithmes comme AES-GCM ou ChaCha20-Poly1305 sont ainsi résistants (au prix parfois d'une augmentation raisonnable de la taille de la clé). Cette résistance permet de faire face au cas où un indiscret stocke aujourd'hui des messages qu'il ne sait pas déchiffrer, en attendant qu'il dispose d'un calculateur quantique. Notez que si on fait face à un adversaire mieux armé, qui n'a pas seulement un calculateur quantique mais peut en plus accéder à l'état quantique de votre système de chiffrement, ça devient plus compliqué. Dans ce cas très futuriste, il faut passer à des algorithmes comme QCB (cf. « "*QCB : Efficient Quantum-Secure Authenticated Encryption*" <https://link.springer.com/chapter/10.1007/978-3-030-92062-3_23> »).

Tout cela, c'étaient les propriétés de sécurité des algorithmes, liés à la définition de ces algorithmes. Mais, en pratique, on n'utilise pas un algorithme mais une mise en œuvre de l'algorithme, réalisée dans un programme. La section 4.4 du RFC liste des propriétés souhaitables des mises en œuvre (en plus de la propriété évidente que le programme soit rapide), propriétés qui n'affectent pas la sécurité (que ce soit en bien ou en mal). Par exemple, on souhaite que le programme soit léger, au sens où il tourne sur des machines contraintes (en mémoire, en processeur, en énergie, etc). Le rapport du NIST « "*NISTIR 8114 - Report on Lightweight Cryptography*" <https://nvlpubs.nist.gov/nistpubs/ir/2017/NIST.IR.8114.pdf> » est une lecture intéressante à ce sujet.

On souhaite ensuite un programme qui puisse exploiter le parallélisme de la machine qui l'exécute. Si celle-ci a plusieurs processeurs et/ou plusieurs cœurs, il serait bon que le travail puisse être réparti entre ces cœurs. Des algorithmes comme le mode CBC ne permettent pas le parallélisme lors du chiffrement.

Autres propriétés souhaitables citées par le RFC : ne pas nécessiter un travail supplémentaire ou de nouvelles ressources quand on introduit une nouvelle clé, fonctionner en une passe unique sur les données, permettre de chiffrer un flux de données continu, sans attendre sa fin…

Merci beaucoup à Manuel Pégourié-Gonnard et Florian Maury pour une relecture détaillée avec plein de bonnes remarques. Comme toujours, les erreurs et approximations sont de moi, pas des relecteurs.