RFC 8387 : Practical Considerations and Implementation Experiences in Securing Smart Object Networks

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 2 janvier 2019

Date de publication du RFC : Mai 2018

https://www.bortzmeyer.org/8387.html

On le sait, la sécurité de l'Internet des Objets est absolument catastrophique. Les objets vendus au grand public, par exemple, non seulement envoient toutes vos données personnelles quelque part chez le vendeur de l'objet, mais en outre permettent en général facilement à des tiers de copier ces données. Ce RFC décrit les défis que pose la sécurisation de réseaux d'objets connectés, spécialement pour ceux qui disposent de ressources (comme l'électricité) limitées.

Le problème n'est pas purement technique : si les innombrables objets connectés que vous avez reçus comme cadeau de Noël cette année ont une sécurité abyssalement basse, c'est parce que les vendeurs sont totalement incompétents, et qu'ils ne font aucun effort pour apprendre. Cela n'a la plupart du temps rien à voir avec le manque de ressources (processeur, électricité) de l'objet. Les télévisions connectées ou voitures connectées n'ont pas une meilleure sécurité, alors qu'elles disposent de bien plus de ressources qu'un Raspberry Pi 1, qui peut pourtant faire du SSH ou TLS sans problème. Mais les vendeurs agissent dans une impunité totale, et ne sont donc pas motivés pour améliorer les choses. Aucun vendeur d'objets connectés n'a jamais eu à subir les conséquences (légales ou commerciales) d'une faille de sécurité. Les objets à la sécurité pourrie se vendent aussi bien que les autres. Et la loi ne protège pas le client. Les mêmes qui s'indignent contre la vente de bitcoins à des particuliers sans expérience financière ne protestent jamais contre la vente d'objets espions facilement piratables.

Mais comme la plupart des RFC, ce document se focalise sur l'aspect technique du problème, et suggère des solutions techniques. Je suis pessimiste quant aux chances de déploiement de ces solutions, en raison de l'absence de motivations (cf. ci-dessus). Mais il est vrai que ce RFC vise plutôt les objets utilisés en milieu industriel que ceux destinés au grand public.

D'abord, les défis auxquels font face les objets connectés (section 3 du RFC). D'abord, il y a le fait que, dans certains objets, les ressources sont limitées : le processeur est très lent, la mémoire est réduite, et la batterie n'a pas des réserves infinies. On a vu que ces ressources limitées sont parfois utilisées comme

excuses pour ne pas mettre en œuvre certaines techniques de sécurité (une télévision connectée qui utilise HTTP et pas HTTPS avec l'argument que la cryptographie est trop coûteuse !) Mais, pour certaines catégories d'objets, l'argument est réel : un capteur industriel peut effectivement ne pas avoir autant de ressources que l'expert en sécurité le souhaiterait. Certains objets sont vendus à des prix individuels très bas et doivent donc être fabriqués à faible coût. Conséquences : il faut utiliser CoAP et non pas HTTP, les clés cryptographiques ne peuvent pas être trop longues, l'engin ne peut pas être allumé en permanence et n'est donc pas forcément joignable, son interface utilisateur très réduite ne permet pas de définir un mot de passe, etc.

Outre le fait que ces ressouces limitées sont souvent un faux prétexte pour justifier la paresse des vendeurs, le RFC note que le développement de ces objets se fait souvent dans le mauvais ordre : on conçoit le matériel puis, bien après, on cherche les solutions de sécurité qui arriveraient à tourner sur ce matériel. Pour sortir de l'état catastrophique de la sécurité des objets connectés, il faudrait procéder en sens inverse, et intégrer la sécurité dès le début, concevant le matériel en fonction des exigences de sécurité.

Deuxième défi pour le concepteur d'un objet connecté, l'avitaillement ("*provisioning*"). Comment mettre dans l'objet connectés les informations nécessaires, mots de passe, clés et autres données ? Par exemple, si une technique de sécurité utilise un mot de passe, mettre le même dans chaque objet lors de sa fabrication est facile, mais évidemment très mauvais pour la sécurité. Mais demander à M. Michu, qui a acheté un gadget connecté, de rentrer un mot de passe avec une interface à deux boutons et un minuscule écran n'est pas évident. Et si une entreprise a acheté une centaine de capteurs, faut-il une intervention manuelle sur chacun, pour mettre le mot de passe ? Le RFC considère que c'est sans doute le principal défi pour la sécurité des objets connectés. Le problème est difficile à résoudre car :

— L'interface utilisateur des objets connectés est minimale, voire inexistante.
— Les utilisateurs typiques ne sont pas compétents en ce domaine, et la plupart du temps ne sont même pas conscients du fait qu'ils ont désormais une responsabilité d'administrateur système. C'est d'ailleurs une des raisons pour lesquelles le terme d'« objet » est dangereux : il empêche de percevoir cette responsabilité. Ces engins sont des ordinateurs, avec les risques associés, et devraient être traités comme tels.
— Ces objets sont souvent livrés en très grande quantité ; toute solution d'avitaillement doit passer à l'échelle.

Résultat, les objets connectés sont souvent vendus avec des informations statiques, et une identité stable, qui facilite certes des aspects de l'administration du réseau, comme le suivi d'un objet donné, mais met sérieusement en danger la sécurité.

En outre, troisième défi, chaque objet ayant souvent des capacités limitées, la communication est fréquemment relayée par des passerelles. Même lorsque l'objet a une visibilité directe, il faut souvent passer par une passerelle car l'objet peut passer la majorité de son temps endormi, donc injoignable, afin d'économiser l'électricité. Les modèles traditionnels de sécurité, fondés sur le principe de bout en bout, ne fonctionnent pas bien dans ce contexte.

Pour affronter ces défis, notamment celui de l'avitaillement, la section 4 de notre RFC décrit un modèle de déploiement, où les identités des objets sont dérivées de clés cryptographiques, auto-générées sur chaque objet, comme les CGA du RFC 3972 [1] ou les HIT du RFC 7401. Dans ce modèle, chaque objet génère une paire de clés. L'opération de génération de l'identité consiste typiquement à appliquer une fonction de condensation cryptographique à la concaténation d'une clé publique de l'engin et d'une information locale. Une fois qu'on connait l'identité d'une machine, on peut communiquer avec elle de manière sûre, puisqu'elle peut signer ses messages avec sa clé privée, qui n'est jamais communiquée.

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc3972.txt`

Avec ce modèle, il n'y a plus de mots de passe définis en usine et identiques sur toutes les machines. Mais cela impose, au moment du déploiement, de récolter ces identités (par exemple via un code QR affiché par l'objet, ou bien via NFC) pour les enregistrer dans la base de données qui sera utilisée par le, la ou les administrateurs du réseau. Ce mécanisme permettra d'authentifier les objets, mais pas aux objets d'authentifier un éventuel partenaire, par exemple celui qui leur envoie des ordres. Pour cela, il faudra en autre indiquer la clé publique de ce partenaire au moment de l'installation, ce qui nécessitera un mécanisme de communication, par exemple via le port USB. On ne pourra pas sortir les objets du carton et aller les poser sur le terrain sans autre formalité. Mais la sécurité est à ce prix.

Ces identités stables posent potentiellement un problème de vie privée. Il faut donc prévoir leur renouvellement, soit périodiquement, soit lorsque l'objet change de propriétaire. Un bouton ou autre mécanisme « oublie tout et crée une nouvelle identité » est donc nécessaire.

Une fois qu'on a ces identités, on peut les utiliser de plusieurs façons, de bout en bout ou bien via des intermédiaires. Plusieurs architectures et protocoles sont possibles. Ce serait par exemple le cas de HIP. Mais les auteurs du RFC privilégient une solution qui se situerait au niveau des applications, bâtie sur CoAP (RFC 7252) car cela permettrait :

— aux machines intermédiaires de servir de caches pour les objets qui dorment,
— à ces mêmes machines d'effectuer des tâches de filtrage ou d'agrégation sans remettre en cause la sécurité,
— à cette solution de fonctionner même en présence de ces "*middleboxes*" qui empêchent souvent tout déploiement d'une solution dans les couches plus basses.

S'agit-il d'une idée en l'air, d'un vague projet ? Non, les auteurs du RFC ont déjà identifié plusieurs bibliothèques logicielles qui peuvent permettre de mettre en œuvre ces idées en fournissant des opérations cryptographiques nécessaires à cette architecture, même à des machines peu gonflées :

— AvrCryptolib <http://www.das-labor.org/wiki/AVR-Crypto-Lib/en>, pour AVR,
— Relic <https://github.com/relic-toolkit/relic>, portable, car écrit en C (d'autres bibliothèques utilisent totalement ou partiellement le langage d'assemblage), inclut RSA et Curve25519,
— TinyECC <http://discovery.csc.ncsu.edu/software/TinyECC/>, écrit en nesC pour TinyOS,
— Wiselib <https://github.com/ibr-alg/wiselib> en C++, et qui a été testé avec succès sur Arduino Uno,
— MatrixSSL <http://www.matrixssl.org/> (rebaptisé depuis « TLS Toolkit »), conçu pour des objets mieux dotés (processeur de 32 bits).

Certains systèmes d'exploitation sont spécialement conçus pour des objets contraints, et disposent des bibliothèques nécessaires. C'est le cas par exemple de mbed <https://www.mbed.com/en/technologies/security/mbed-tls/>, qui tourne sur plusieurs membres de la famille ARM Cortex.

Ce n'est pas tout d'avoir du code, encore faut-il qu'il tourne de manière efficace. La section 6 de notre RFC est dédiée aux tests de performance, notamment lors des opérations de signature. C'est ainsi que RSA avec une clé de 2 048 bits et le code d'AvrCryptolib prend vraiment trop de temps (sur une machine apparemment non spécifiée) pour être utilisable.

ECDSA avec TinyECC sur un Arduino Uno tourne par contre en un temps supportable. Sans utiliser le code en langage d'assemblage qui est disponible dans cette bibliothèque, la consommation de RAM reste la même mais le temps d'exécution augmente de 50 à 80 %. D'autres mesures figurent dans cette section, avec diverses bibliothèques, et divers algorithmes. La conclusion est la même : la cryptographie asymétrique, sur laquelle repose l'architecture de sécurité proposée dans notre RFC est réaliste sur des objets très contraints mais probablement uniquement avec les courbes elliptiques. Les courbes recommandées sont celles du RFC 7748, bien plus rapides que celles du NIST (tests avec la bibliothèque NaCl <http://munacl.cryptojedi.org/>).

Les problèmes concrets ne s'arrêtent pas là. Il faut aussi voir que certains objets contraints, comme l'Arduino Uno, n'ont pas de générateur aléatoire matériel. (Contrairement à, par exemple, le Nordic nRF52832 <http://infocenter.nordicsemi.com/pdf/nRF52832_PS_v1.3.pdf>.) Ces engins n'ayant pas non plus d'autres sources d'entropie, générer des nombres aléatoires de qualité (RFC 4086), préliminaire indispensable à la création des clés, est un défi.

La section 7 du RFC décrit une application développée pour illustrer les principes de ce RFC. Il s'agit de piloter des capteurs qui sont éteints pendant l'essentiel du temps, afin d'économiser leur batterie, et qui se réveillent parfois pour effectuer une mesure et en transmettre le résultat. Des machines sont affectées à la gestion de la communication avec les capteurs et peuvent, par exemple, conserver un message lorsque le capteur est endormi, le distribuant au capteur lorsqu'il se réveille. Plus précisément, il y avait quatre types d'entités :

— Les capteurs, des Arduino Mega (un processeur de seulement 8 bits) utilisant Relic <https://github.com/relic-toolkit/relic>, soit 29 ko dans la mémoire flash pour Relic et 3 pour le client CoAP,
— Le courtier, qui est allumé en permanence, et gère la communication avec les capteurs, selon un modèle publication/abonnement,
— L'annuaire, où capteurs et courtier peuvent enregistrer des ressources et les chercher,
— L'application proprement dite, tournant sur un ordinateur généraliste avec Linux, et communiquant avec l'annuaire et avec les courtiers.

Le modèle de sécurité est TOFU ; au premier démarrage, les capteurs génèrent les clés, et les enregistrent dans l'annuaire, suivant le format du RFC 6690. (L'adresse IP de l'annuaire est codée en dur dans les capteurs, évitant le recours au DNS.) Le premier enregistrement est supposé être le bon. Ensuite, chaque fois qu'un capteur fait une mesure, il l'envoie au courtier en JSON signé en ECDSA avec JOSE (RFC 7515). Il peut aussi utiliser CBOR avec COSE (RFC 8152). La signature peut alors être vérifiée. On voit qu'il n'y a pas besoin que la machine de vérification (typiquement celle qui porte l'application) soit allumée en même temps que le capteur. Ce prototype a bien marché. Cela montre comment on peut faire de la sécurité de bout en bout bien qu'il y ait au moins un intermédiaire (le courtier).

La question de la faisabilité de l'architecture décrite dans ce RFC est discutée plus en détail dans la section 8.1. On entend souvent que les objets connectés n'ont pas de vraie sécurité car « la cryptographie, et surtout la cryptographie asymétrique, sont bien trop lentes sur ces objets contraints ». Les expériences faites ne donnent pas forcément un résultat évident. La cryptographie asymétrique est possible, mais n'est clairement pas gratuite. RSA avec des clés de tailles raisonnables pourrait mettre plusieurs minutes à signer, ce qui n'est pas tolérable. Heureusement, ce n'est pas le seul algorithme. (Rappelons que cela ne s'applique qu'aux objets contraints. Il n'y a aucune bonne raison de ne pas utiliser la cryptographie pour des objets comme une télévision ou une caméra de surveillance.) La loi de Moore joue ici en notre faveur : les microcontrôleurs de 32 bits deviennent aussi abordables que ceux de 8 bits.

Quant aux exigences d'énergie électrique, il faut noter que le plus gourmand, et de loin, est la radio (cf. Margi, C., Oliveira, B., Sousa, G., Simplicio, M., Paulo, S., Carvalho, T., Naslund, M., et R. Gold, « "*Impact of Operating Systems on Wireless Sensor Networks (Security) Applications and Testbeds*" <http://www.webscience.org.br/wiki/images/a/a6/Testbed-eval.pdf> », à WiMAN en 2010). Signer et chiffrer, ce n'est rien, par rapport à transmettre.

L'ingénierie, c'est toujours faire des compromis, particulièrement quand on travaille avec des systèmes aussi contraints en ressources. La section 8 du RFC détaille certains de ces compromis, expliquant les choix à faire. Ainsi, la question de la fraîcheur des informations est délicate. Quand on lit le résultat d'une mesure sur le courtier, est-on bien informé sur l'ancienneté de cette mesure ? Le capteur n'a en général pas d'horloge digne de ce nom et ne peut pas se permettre d'utiliser NTP. On peut donc être amené à sacrifier la résolution de la mesure du temps aux contraintes pratiques.

Une question aussi ancienne que le modèle en couches est celle de la couche où placer la sécurité. Idéalement, elle devrait être dans une seule couche, pour limiter le code et le temps d'exécution sur les objets contraints. En pratique, elle va sans doute se retrouver sur plusieurs couches :

— Couche 2 : c'est la solution la plus courante aujourd'hui (pensez à une carte SIM et aux secrets qu'elle stocke). Mais cela ne sécurise que le premier canal de communications, ce n'est pas une sécurité de bout en bout. Peu importe que la communication de l'objet avec la base 4G soit signée et chiffrée si le reste du chemin est non authentifié, et en clair !
— Couche 3 : c'est ce que fait IPsec, solution peu déployée en pratique. Cette fois, c'est de bout en bout mais il est fréquent qu'un stupide intermédiaire bloque les communications ainsi sécurisées. Et les systèmes d'exploitation ne fournissent en général pas de moyen pratique permettant aux applications de contrôler cette sécurité, ou même simplement d'en être informé.
— Couche 4 ou Couche 7 : c'est l'autre solution populaire, une grande partie de la sécurité de l'Internet repose sur TLS, par exemple. (Ou, pour les objets contraints, DTLS.) C'est évidemment la solution la plus simple à contrôler depuis l'application. Par contre, elle ne protège pas contre les attaques dans les couches plus basses (un paquet TCP RST malveillant ne sera pas gêné par TLS) mais ce n'est peut-être pas un gros problème, cela fait juste un déni de service, qu'un attaquant pourrait de toute façon faire par d'autres moyens.
— On peut aussi envisager de protéger les données et pas la communication, en signant les données. Cela marche même en présence d'intermédiaires divers mais cette solution est encore peu déployée en ce moment, par rapport à la protection de la couche 2 ou de la couche 7.

Enfin, dernière étude sur les compromis à faire, le choix entre la cryptographie symétrique et la cryptographie asymétrique, bien plus gérable en matière de distribution des clés, mais plus consommatrice de ressources. On peut faire de la cryptographie symétrique à grande échelle mais, pour la cryptographie asymétrique, il n'y a pas encore de déploiements sur, disons, des centaines de millions d'objets. On a vu que dans certains cas, la cryptographie asymétrique coûte cher. Néanmoins, les processeurs progressent et, question consommation énergétique, la cryptographie reste loin derrière la radio. Enfin, les schémas de cryptographie des objets connectés n'utiliseront probablement la cryptographie asymétrique que pour s'authentifier au début, utilisant ensuite la cryptographie symétrique pour l'essentiel de la communication.

Enfin, la section 9 de notre RFC résume les points importants :

— Il faut d'abord établir les exigences de sécurité puis seulement après choisir le matériel en fonction de ces exigences,
— La cryptographie à courbes elliptiques est recommandée,
— La qualité du générateur de nombres aléatoires, comme toujours en cryptographie, est cruciale pour la sécurité,
— Si on commence un projet aujourd'hui, il vaut mieux partir directement sur des microcontrôleurs de 32 bits,
— Il faut permettre la génération de nouvelles identités, par exemple pour le cas où l'objet change de propriétaire,
— Et il faut prévoir les mises à jour futures (un objet peut rester en service des années) par exemple en terme de place sur la mémoire flash (le nouveau code sera plus gros).

La section 2 de notre RFC décrit les autres travaux menés en matière de sécurité, en dehors de ce RFC. Ainsi, la spécification du protocole CoAP, le « HTTP du pauvre objet » (RFC 7252) décrit comment utiliser CoAP avec DTLS ou IPsec. Cette question de la sécurité des objets connectés a fait l'objet d'un atelier de l'IAB en 2011, atelier dont le compte-rendu a été publié dans le RFC 6574.