RFC 8555 : Automatic Certificate Management Environment (ACME)

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 11 avril 2019

Date de publication du RFC : Mars 2019

https://www.bortzmeyer.org/8555.html

Une grande partie de la sécurité du Web, et d'ailleurs de plein d'autres chose sur l'Internet, repose sur des certificats où une Autorité de Certification (AC) garantit que vous êtes bien ce que vous prétendez être. Traditionnellement, l'émission d'un certificat se faisait selon un processus manuel, lent et coûteux, à part dans quelques AC automatisées et gratuites comme CAcert <https://www.bortzmeyer.org/cacert.html>. Mais il n'existait pas de mécanisme standard pour cette automatisation. (Et CAcert n'a pas d'API, même non-standard.) Un tel mécanisme standard existe désormais, avec le protocole ACME, normalisé dans ce RFC. Son utilisateur le plus connu est l'AC Let's Encrypt.

Pour comprendre ACME, il faut d'abord revenir aux utilisations des certificats. La norme technique pour les certificats utilisés sur l'Internet se nomme PKIX et est normalisée dans le RFC 5280 [1]. PKIX est un profil (une restriction d'une norme beaucoup plus large - et bien trop large, comme le sont souvent les normes des organismes comme l'UIT ou l'ISO) de X.509. Un certificat PKIX comprend, entre autres :

- Une clé cryptographique publique, le titulaire du certificat étant supposé conserver avec soin et précaution la clé privée correspondante,
- Le nom du titulaire du certificat (X.509 l'appelle le sujet),
- Une signature de l'émetteur du certificat (l'AC).
- Des métadonnées dont notamment la date d'expiration du certificat, qui sert à garantir qu'en cas de copie de la clé privée, le copieur ne pourra pas profiter du certificat éternellement.

On note que le certificat est public. N'importe qui peut récupérer le certificat de, par exemple, un site Web. Voici un exemple avec OpenSSL et www.labanquepostale.fr pour un certificat de type EV :

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc5280.txt

```
% openssl s_client -connect www.labanquepostale.fr:443 -showcerts | openssl x509 -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            0d:8f:ec:dd:d8:7b:83:b8:a0:1e:eb:c2:a0:2c:10:9b
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert SHA2 Extended Validation Serv
        Validity
            Not Before: Sep  5 00:00:00 2018 GMT
            Not After : Sep  4 12:00:00 2020 GMT
        Subject: businessCategory = Private Organization, jurisdictionC = FR, serialNumber = 421 100 645, C
...
```

et un avec GnuTLS pour un certificat DV ("*Domain Validation*"), `mamot.fr` :

```
% gnutls-cli mamot.fr
 - subject `CN=mamot.fr', issuer `CN=Let's Encrypt Authority X3,O=Let's Encrypt,C=US', serial 0x035516154ab9
...
```

D'ailleurs, des services comme "*Certificate Transparency*" (RFC 6962), accessible entre autres en , donnent accès facilement à tous les certificats émis par les AC participantes.

Du fait que seul le titulaire connait la clé privée, la capacité à signer des messages vérifiables avec la clé publique permet d'authentifier le partenaire avec lequel on communique. Grâce à la signature de l'AC, quiconque fait confiance à cette AC particulière peut être sûr que le certificat appartient bien au titulaire. Dans l'exemple avec OpenSSL, le certificat de la Banque Postale était signé par DigiCert, si on fait confiance à DigiCert, on sait qu'on est bien connecté à la Banque Postale.

Qui sont les AC ? Ce sont la plupart du temps des entreprises commerciales qui sont payées par les titulaires de certificat, et elles sont censées vérifier la sincérité de leurs clients. Cette vérification peut être manuelle, partiellement ou totalement automatique. Normalement, les certificats de type EV ("*Extended Validation*"), comme celui de la Banque Postale plus haut, font l'objet d'une vérification manuelle. Cela permet de vérifier l'identité officielle (celle gérée par l'État) du titulaire. Les certificats DV ("*Domain Validation*"), comme celui de `mamot.fr`, eux, peuvent être validés automatiquement, ils assurent uniquement le fait que le titulaire contrôle bien le nom de domaine utilisé comme sujet. (Pour avoir tous les horribles détails, y compris les certificats OV - "*Organization Validated*" - dont je n'ai pas parlé, on peut consulter les « "*Baseline Requirements for the Issuance and Management of Publicly-Trusted Certificates*" <https://cabforum.org/baseline-requirements-documents/> » du CA/Browser Forum.) Ainsi, pour CAcert <https://www.bortzmeyer.org/cacert.html>, on doit prouver le contrôle du domaine en répondant aux courriers envoyés aux adresses publiques de contact pour le domaine.

Les certificats peuvent servir à de nombreux protocoles de sécurité mais le plus connu est TLS (normalisé dans le RFC 8446). Comme il n'est pas le seul protocole pour lequel on a des certificats, il est erroné de parler de « certificats TLS » ou, pire encore, de « certificats SSL ». TLS est un protocole client/serveur, où le client authentifie le serveur mais où le serveur authentifie rarement le client. Il est à la base de la sécurisation d'un grand nombre de services sur l'Internet, à commencer par le Web avec HTTPS (RFC 2818). L'authentification du serveur par le client est faite en vérifiant (attention, je vais simplifier) :

— Que le partenaire avec qui on parle a la clé privée (il peut signer des messages) correspondant au certificat présenté,
— Que le certificat n'a pas expiré,

— Que le certificat est signé par une AC connue du client (la clé publique de l'AC est dans le magasin du client),
— Que le nom indiqué par le client correspond à un des noms disponibles dans le certificat. Dans le cas du Web, c'est le nom de domaine dans l'URL choisi (RFC 6125).

Une fois cette authentification faite, TLS assure l'intégrité et la confidentialité de la communication.

Attention, on parle bien d'authentification, pas de confiance. Malgré ce que vous pourrez lire dans les « La sécurité pour les nuls », le fameux « cadenas vert » ne signifie pas du tout que vous pouvez faire vos achats en ligne en toute sécurité. Il indique seulement que le partenaire a bien le nom que vous avez demandé, et qu'un tiers ne pourra pas écouter ou modifier la conversation. Il n'indique **pas** que le partenaire soit digne de confiance ; l'AC ne peut pas vérifier cela ! Ainsi, dans l'exemple plus haut, TLS et l'authentification par certificat garantissent bien qu'on se connecte au serveur HTTPS de la Maison-Blanche, `www.whitehouse.gov`, mais pas que Trump dise la vérité !

J'ai parlé du magasin où se trouvent les clés des AC à qui on fait confiance. Qui décide du contenu de ce magasin ? C'est une question complexe, il n'y a pas **une** liste d'AC faisant autorité. La plupart des systèmes d'exploitation ont une liste à eux, créée en suivant des critères plus ou moins opaques. Les applications (comme le navigateur Web) utilisent ce magasin global du système ou, parfois, ont leur propre magasin, ce qui aggrave encore la confusion. Les utilisateurs peuvent (c'est plus ou moins facile) ajouter des AC ou bien en retirer.

Et comment obtient-on un certificat ? Typiquement, on crée d'abord une demande de certificat (CSR pour "*Certificate Signing Request*", cf. RFC 2986). Avec OpenSSL, cela peut se faire avec :

```
% openssl req  -new -nodes -newkey rsa:2048 -keyout server.key -out server.csr
```

On se connecte ensuite au site Web de l'AC choisie, et on lui soumet le CSR. Ici, un exemple avec CAcert :

L'AC doit alors faire des vérifications, plus ou moins rigoureuses. Par exemple, l'AC fait une requête whois, note l'adresse de courrier du domaine, envoie un message contenant un défi et le client de l'AC doit y répondre pour prouver qu'il a bien accès à cette adresse et contrôle donc bien le domaine. L'AC crée ensuite le certificat qu'elle vous renvoie. Il faut alors l'installer sur le serveur (HTTPS, par exemple). L'opération est complexe, et beaucoup d'utilisateurs débutants cafouillent.

C'est ce processus non-standard et compliqué que le protocole ACME vise à normaliser et à automatiser. Ce RFC a une longue histoire <https://datatracker.ietf.org/doc/rfc8555/> mais est déjà déployé en production par au moins une AC.

Le principe d'ACME est simple : l'AC fait tourner un serveur ACME, qui attend les requêtes des clients. Le client ACME (un logiciel comme dehydrated <https://github.com/lukas2511/dehydrated> ou certbot <https://certbot.eff.org/>) génère la CSR, se connecte au serveur, et demande un certificat signé pour un nom donné. Le serveur va alors renvoyer un **défi** qui va permettre au client de prouver qu'il contrôle bien le nom de domaine demandé. Il existe plusieurs types de défis <https://www.iana.org/assignments/acme/acme.xml#acme-validation-methods>, mais le plus simple est un nom de fichier que le serveur ACME choisit, demandant au client ACME de mettre un fichier de ce nom sur son site Web. Si le nom de fichier était `Vyzs0Oqkfa4gn4skMwszORg6vJaO73dvMLN0uX38TDw`, le serveur ACME va devenir client HTTP et chercher à récupérer `http://DOMAIN/.well-known/acme-challenge/Vy`

S'il y réussit, il considère que le client ACME contrôle bien le nom de domaine, et il signe alors le certificat, qui est renvoyé au client lorsque celui-ci soumet la CSR.

Le modèle idéal d'utilisation d'ACME est présenté dans la section 2. (En pratique, il n'est pas vraiment réalisé, notamment parce qu'il n'existe pratiquement qu'une seule AC utilisant ACME, Let's Encrypt. Il n'y a donc pas encore beaucoup de diversité.) L'espoir est qu'un jour, on puisse faire comme suit :

— On installe un serveur Web (avec des services comme le CMS),
— La procédure d'installation vous demande le nom de domaine à utiliser (ce point là n'est pas automatisable, sans même parler de la procédure de location du nom de domaine),
— Le logiciel vous propose une liste d'AC parmi lesquelles choisir (on a vu qu'il n'y en avait qu'une actuellement ; dans le futur, s'il y en a plusieurs, l'utilisateur aura sans doute autant de mal à choisir qu'il ou elle en a aujourd'hui à choisir un BE),
— Le logiciel fait tout le reste automatiquement : requête à l'AC choisie en utilisant le protocole ACME normalisé dans notre RFC, réponse au défi de l'AC via le serveur HTTP installé, récupération du certificat et configuration de TLS,
— Par la suite, c'est le logiciel qui effectuera automatiquement les demandes de renouvellement de certificat (aujourd'hui, avec les logiciels existants, c'est le point qui est le plus souvent oublié ; combien de sites Web ont annoncé fièrement qu'ils étaient désormais protégés par HTTPS, pour afficher un certificat expiré trois mois après…)

Ainsi, une procédure manuelle et pénible pourra devenir assez simple, encourageant une présence en ligne plus sécurisée. Cela pourrait devenir aussi simple que d'avoir un certificat auto-signé.

La section 4 du RFC expose de manière générale le protocole ACME (le RFC complet fait 94 pages, car il y a beaucoup de détails à spécifier). Rappelez-vous avant de la lire que, pour ACME, le **client** est celui qui demande le certificat (c'est donc typiquement un serveur Internet, par exemple un serveur HTTPS) et le **serveur** ACME est l'AC. **Quand je dirais « client » ou « serveur » tout court, il s'agira du client et du serveur ACME.**

ACME encode ses messages en JSON (RFC 8259). Le client doit d'abord avoir un compte auprès du serveur (avec Let's Encrypt, cela peut être fait automatiquement sans que l'utilisateur s'en rende compte). Par exemple, avec dehydrated, cela se fait ainsi :

```
% dehydrated --register --accept-terms
+ Generating account key...
+ Registering account key with ACME server...
+ Done!
```

Et on trouve dans le répertoire `accounts/` la clé privée du compte, et les informations du compte :

```
% cat accounts/aHR0cHM6Ly9...9yeQo/registration_info.json
{
  "id": 5...1,
  "key": {
    "kty": "RSA",
    "n": "wv...HCk",
    "e": "AQAB"
  },
  "contact": [],
  "initialIp": "2001:4b98:dc0:41:216:3eff:fe27:3d3f",
  "createdAt": "2019-03-12T19:32:20.018154799Z",
  "status": "valid"
}
```

Pour certbot, on peut le faire tourner avec l'option `-v` pour avoir les mêmes informations. certbot affiche également des messages d'ordre administratif comme :

```
Enter email address (used for urgent renewal and security notices) (Enter 'c' to
cancel): stephane+letsencrypt@bortzmeyer.org
...
Please read the Terms of Service at
https://letsencrypt.org/documents/LE-SA-v1.2-November-15-2017.pdf. You must
agree in order to register with the ACME server at
https://acme-v02.api.letsencrypt.org/directory
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(A)gree/(C)ancel: A

JWS payload:
b'{\n  "contact": [\n    "mailto:stephane+letsencrypt@bortzmeyer.org"\n  ],\n  "termsOfServiceAgreed": true,\n
{
  "id": 53367492,
  "key": { ...
  "contact": [
    "mailto:stephane+letsencrypt@bortzmeyer.org"
  ],
  "createdAt": "2019-03-15T16:07:58.29914038Z",
  "status": "valid"
}

Reporting to user: Your account credentials have been saved in your Certbot configuration directory at /etc/lets

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Would you be willing to share your email address with the Electronic Frontier
Foundation, a founding partner of the Let's Encrypt project and the non-profit
organization that develops Certbot? We'd like to send you email about our work
encrypting the web, EFF news, campaigns, and ways to support digital freedom.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(Y)es/(N)o: No
...
```

Le compte sera authentifié en utilisant une biclé (clé privée et clé publique). Il y aura ensuite quatre étapes :

— Demander un certificat,
— Répondre au défi (notez bien qu'ACME permet plusieurs types de défis possibles),
— Envoyer le CSR,
— Récupérer le certificat signé.

Mais comment transporte-t-on ces messages en JSON ? La section 6 du RFC répond à cette question : on utilise HTTPS. En prime, les messages sont signés avec JWS (RFC 7515), en utilisant la clé privée du client pour les requêtes. Voici par exemple la réponse d'un serveur ACME lorsqu'on l'interroge sur un défi en cours :

```
{
  "type": "http-01",
  "status": "pending",
  "url": "https://acme-v02.api.letsencrypt.org/acme/challenge/7TAkQBMmFqm8Rhs6Sn8SFCne2MoZXoEHCz0Px7f0dpE/136836
  "token": "mMXGXjEijKBZXl2RuL0rjlektPPpy-ozJpZ2vB4w6Dw"
}
```

Les messages d'erreur utilisent le RFC 7807. En voici un exemple :

```
{
  "type": "http-01",
  "status": "invalid",
  "error": {
    "type": "urn:acme:error:unauthorized",
    "detail": "Invalid response from http://mercredifiction.bortzmeyer.org/.well-known/acme-challenge/rE-rI
    "status": 403  ...
   [Le message d'erreur indique également typiquement l'URL demandé,
   et les adresses IP utilisées, ce qui est crucial si le serveur HTTP
   a plusieurs adresses IP, par exemple une en IPv4 et une en IPv6. Il
   faut donc bien lire tout le message d'erreur.]
```

Une liste des erreurs possibles est enregistrée à l'IANA <`https://www.iana.org/assignments/acme/acme.xml#acme-error-types`>. Voici par exemple une erreur CAA (RFC 6844) :

```
  "error": {
    "type": "urn:acme:error:caa",
    "detail": "CAA record for mercredifiction.bortzmeyer.org prevents issuance",
    "status": 403
  },
```

Comment un client ACME trouve-t-il les URL pour les différentes opérations ? Il y a un URL à connaitre, le répertoire (*"directory"*). Une requête à cet URL (par exemple `curl https://acme-v02.api.letsenc` va renvoyer un objet JSON qui contient la liste des autres URL (section 7, une des plus cruciales du RFC). Voici un exemple chez Let's Encrypt :

```
{ ...
  "meta": {
    "caaIdentities": [
      "letsencrypt.org"
    ],
    "termsOfService": "https://letsencrypt.org/documents/LE-SA-v1.2-November-15-2017.pdf",
    "website": "https://letsencrypt.org"
  },
  "newAccount": "https://acme-v02.api.letsencrypt.org/acme/new-acct",
  "newNonce": "https://acme-v02.api.letsencrypt.org/acme/new-nonce",
  "newOrder": "https://acme-v02.api.letsencrypt.org/acme/new-order",
  "revokeCert": "https://acme-v02.api.letsencrypt.org/acme/revoke-cert"
}
```

On peut ensuite créer un compte (le champ `newAccount` dans l'exemple ci-dessus) puis demander des certificats (champ `newOrder` dans l'exemple ci-dessus), ici (Let's Encrypt) en écrivant à `https://acme-v02.api`

```
{
  "payload": "ewogICJpZGVudGlmaWVycyI6IFsKICAgIHsKICAgICAgInR5cGUiOiAiZG5zIiwKICAgICAgInZhbHVlIjogInRlc3QtY
  "protected": "eyJhbGciOiAiUlMyNTYiLCAidXJsIjogImh0dHBzOi8vYWNtZS12MDIuYXBpLmxldHNlbmNyeXB0Lm9yZy9hY21lL25
  "signature": "MP9rXTjX4t1Be-y6dhPOP7JE3B401wokydUlG8gJGWqibTM_gydkUph1smtrUZ5W4RXNTEnlmiFwoiU4eHLD-8MzN5a
}
```

Eh oui, tout est signé, en JWS (RFC 7515) et base64isé (cf. section 7.4 du RFC). Ici, le décodage Base64 nous dira que la requête était :

```
{
 {
  "identifiers": [
    {
      "type": "dns",
      "value": "test-acme.bortzmeyer.fr"
    }
  ]
  }
  , {"alg": "RS256", "url":
  "https://acme-v02.api.letsencrypt.org/acme/new-order", "kid":
  "https://acme-v02.api.letsencrypt.org/acme/acct/53177050", "nonce":
  "resWMqmBBXmVkgb_JuPSuDrifC8Al6kNmIx6n6EpD1Y"}
}
```


Donc, une demande de certificat pour `test-acme.bortzmeyer.fr`.

Les autres opérations possibles avec un serveur ACME sont enregistrées à l'IANA <https://www.iana.org/assignments/acme/acme.xml#acme-resource-types>. Par exemple, on peut révoquer un certificat.

La réponse sera :


```
{
  "status": "pending",
  "expires": "2019-03-19T19:50:41.434669372Z",
  "identifiers": [
    {
      "type": "dns",
      "value": "test-acme.bortzmeyer.fr"
    }
  ],
  "authorizations": [
    "https://acme-v02.api.letsencrypt.org/acme/authz/FVMFaHS_oWjqfR-rWd6eBKMlt1EWfIcf6i7D4wU_swM"
  ],
  "finalize": "https://acme-v02.api.letsencrypt.org/acme/finalize/53177050/352606317"
}
```


Le client ACME va alors télécharger l'autorisation à l'URL indiqué, récupérant ainsi les défis qu'il devra affronter (section 7.5 du RFC). Une fois qu'il a fait ce qui lui était demandé par le serveur, il utilise l'URL donné dans le champ `finalize` pour indiquer au serveur que c'est bon, que le serveur peut vérifier. La commande `certbot` avec l'option `-v` vous permettra de voir tout ce dialogue.

Le protocole ACME peut être utilisé par d'autres AC que Let's Encrypt. Avec le client dehydrated, il suffira, quand de telles AC seront disponibles, de mettre `CA=URL` dans le fichier de configuration (l'URL par défaut est `https://acme-v02.api.letsencrypt.org/directory`). Un exemple d'autre AC utilisant ACME est BuyPass <https://www.buypass.com/> (pas testé).

Mais en quoi consistent exactement les **défis**, dont j'ai déjà parlé plusieurs fois ? La section 8 les explique. L'idée de base d'un défi ACME est de permettre de prouver qu'on contrôle réellement un identificateur, typiquement un nom de domaine. ACME ne normalise pas un type de défi particulier. Le cadre est ouvert, et de nouveaux défis pourront être ajoutés dans le futur. Le principe est toujours le même : demander au client ACME de faire quelque chose que seul le vrai titulaire de l'identificateur

pourrait faire. Un défi, tel qu'envoyé par le serveur ACME, a un type (le plus utilisé aujourd'hui, est de loin, est le défi `http-01`), un état (en attente ou bien, au contraire, validé) et un texte d'erreur, au cas où la validation ait échoué. Plusieurs défis, comme `http-01` ont également un jeton, un "*cookie*", un texte généré par le serveur, et non prévisible par le client ou par le reste du monde, et qu'il faudra placer quelque part où le serveur pourra le vérifier. Le serveur ACME ne testera que lorsque le client lui aura dit « c'est bon, je suis prêt, j'ai fait tout ce que tu m'as défié de faire ». Le RFC demande également au serveur de réessayer après cinq ou dix secondes, si la vérification ne marche pas du premier coup, au cas où le client ait été trop rapide à se dire prêt.

Le plus connu et le plus utilisé des défis, à l'heure actuelle, est `http-01`. Le client ACME doit configurer un serveur HTTP où une page (oui, je sais, le terme correct est « ressource ») a comme nom le contenu du jeton. Le serveur ACME va devenir client HTTP pour récupérer cette page et, s'il y arrive, cela prouvera que le client contrôlait bien le nom de domaine qu'il avait indiqué. De manière surprenante, et qui déroute beaucoup de débutants, le défi se fait bien sur HTTP et pas HTTPS, parce que beaucoup d'hébergements Web partagés ne donnent pas suffisamment de contrôle à l'hébergé.

Le jeton est une chaîne de caractères utilisant le jeu de caractères de Base64, pour passer partout. Voici un exemple de défi HTTP envoyé par le serveur :

```
{
  "identifier": {
    "type": "dns",
    "value": "test-acme.bortzmeyer.fr"
  },
  "status": "pending",
  "expires": "2019-03-19T19:50:41Z",
  "challenges": [
    {
      "type": "http-01",
      "status": "pending",
      "url": "https://acme-v02.api.letsencrypt.org/acme/challenge/FVMFaHS_oWjqfR-rWd6eBKMlt1EWfIcf6i7D4wU_sw
      "token": "4kpeqw7DVMrY6MI3tw1-tTq9oySN2SeMudaD32IcxNM"
    } ...
```

L'URL qu'utilisera le serveur est `http://DOMAINE-DEMANDÉ/.well-known/acme-challenge/JETON` (ou, en syntaxe du RFC 6570, `http://{domain}/.well-known/acme-challenge/{token}`). Comme expliqué plus haut, c'est bien `http://` et pas `https://`. Les URL avec `.well-known` sont documentés dans le RFC 8615 et `acme-challenge` est désormais dans le registre <`https://www.iana.org/assignments/well-known-uris/well-known-uris.xml`>.

Imaginons qu'on utilise le serveur HTTP Apache et qu'on veuille répondre à ce défi. Le plus simple est de configurer le serveur ainsi :

```
<VirtualHost *:80>
   Alias /.well-known/acme-challenge /var/lib/dehydrated/acme-challenges
   <Directory /var/lib/dehydrated/acme-challenges>
        Options None
        AllowOverride None
...
```

Cela indique à Apache que les réponses aux défis seront dans le répertoire `/var/lib/dehydrated/acme-challenge` répertoire où le client ACME dehydrated va mettre ses fichiers. Avec le serveur HTTP Nginx, le principe est le même :

```
server {
    location ^~ /.well-known/acme-challenge {
         alias /var/lib/dehydrated/acme-challenges;
    }
}
```

Bien sûr, de nombreuses autres solutions sont possibles. Le serveur HTTP peut intégrer le client ACME, par exemple. Autre exemple, le client ACME certbot inclut son propre serveur HTTP, et peut donc répondre aux défis tout seul, sans Apache.

Ensuite, on lance le client ACME, éventuellement en lui spécifiant où il doit écrire la réponse aux défis :

```
% certbot certonly --webroot -w /usr/share/nginx/html -d MONDOMAINE.eu.org
```

certbot va mettre le certificat généré et signé dans son répertoire, typiquement `/etc/letsencrypt/live/MONDOMAIN` Et on programme son système (par exemple avec cron) pour relancer le client ACME tous les jours (les clients ACME typique vérifient la date d'expiration du certificat, et n'appellent l'AC que si cette date est proche.) Notez bien qu'il est crucial de superviser l'expiration des certificats <`https://www.bortzmeyer.org/tester-expiration-certifs.html`>. On voit fréquemment des sites Web utilisant Let's Encrypt devenir inaccessibles parce que le certificat a été expiré. Beaucoup d'administrateurs système croient que parce que Let's Encrypt est « automatique », il n'y a aucun risque. Mais ce n'est pas vrai : non seulement la commande de renouvellement peut ne pas être exécutée, ou bien mal se passer mais, même si le certificat est bien renouvellé, cela ne garantit pas que le serveur HTTP soit rechargé.

Petite anecdote personnelle : pour le blog que vous êtes en train de lire, cela avait été un peu plus compliqué. En effet, le blog a deux copies, sur deux machines différentes. J'ai donc du rediriger les vérifications ACME <`https://www.bortzmeyer.org/passage-blog-lets-encrypt.html`> sur une seule des deux machines. En Apache :

```
ProxyRequests Off
ProxyPass /.well-known/acme-challenge/ http://MACHINE-DE-RÉFÉRENCE.bortzmeyer.org/.well-known/acme-chall
ProxyPreserveHost On
```

À noter qu'un serveur HTTP paresseux qui se contenterait de répondre 200 (OK) à chaque requête sous `/.well-known/acme-challenge` n'arriverait pas à répondre avec succès aux défis HTTP. En effet, le fichier doit non seulement exister mais également contenir une chaîne de caractères faite à partir d'éléments fournis par le serveur ACME (cf. section 8.3).

Un autre type de défi répandu est le défi `dns-01`, où le client doit mettre dans le DNS un enregistrement TXT `_acme-challenge.DOMAINE-DEMANDÉ` contenant le jeton. Cela nécessite donc un serveur DNS faisant autorité qui permette les mises à jour dynamiques, via le RFC 2136 ou bien via une API. Notez que le RFC recommande (section 10.2) que l'AC fasse ses requêtes DNS via un résolveur qui valide avec DNSSEC. (Le serveur ACME ne demande pas directement aux serveurs faisant autorité,

il passe par un résolveur. Attention donc à la mémorisation par les résolveurs des réponses, jusqu'au TTL.)

On peut utiliser le défi DNS avec des jokers (pour avoir un certificat pour `*.MONDOMAINE.fr`) mais c'est un peu plus compliqué (section 7.1.3 si vous voulez vraiment les détails).

D'autres types de défis pourront être ajouté dans le futur. Un registre IANA <https://www.iana.org/assignments/acme/acme.xml#acme-validation-methods> en garde la liste. Notez que des types de défis peuvent également être supprimés comme `tls-sni-01` et `tls-sni-02`, jugés à l'usage pas assez sûrs.

Le but de ce RFC est la sécurité, donc toute faiblesse d'ACME dans ce domaine serait grave. La section 10 du RFC se penche donc sur la question. Elle rappelle les deux objectifs de sécurité essentiels :

— Seul[Caractère Unicode non montré [2]]e l[Caractère Unicode non montré]e[Caractère Unicode non montré]a vrai[Caractère Unicode non montré]e titulaire d'un identificateur (le nom de domaine) peut avoir une autorisation pour un certificat pour cet identificateur,
— Une fois l'autorisation donnée, elle ne peut pas être utilisée par un autre compte.

Le RFC 3552 décrit le modèle de menace typique de l'Internet. ACME a deux canaux de communication, le canal ACME proprement dit, utilisant HTTPS, et le canal de validation, par lequel se vérifient les réponses aux défis. ACME doit pouvoir résister à des attaques passives et actives sur ces deux canaux.

ACME n'est qu'un protocole, il reçoit des demandes, envoie des requêtes, traite des réponses, mais il ne sait pas ce qui se passe à l'intérieur des machines. Les défis, par exemple, peuvent être inutiles si la machine testée est mal gérée (section 10.2). Si, par exemple, le serveur HTTP est sur un serveur avec plusieurs utilisateurs, et où tout utilisateur peut bricoler la configuration HTTP, ou bien écrire dans le répertoire `.well-known`, alors tout utilisateur sur ce serveur pourra avoir un certificat. Idem évidemment si le serveur est piraté. Et, si on sous-traite le serveur de son organisation à l'extérieur, le sous-traitant peut également faire ce qu'il veut et obtenir des certificats pour son client (≪ il n'y a pas de "*cloud*", il y a juste l'ordinateur de quelqu'un d'autre ≫).

ACME permet d'obtenir des certificats DV et ceux-ci dépendent évidemment des noms de domaine et du DNS. Un attaquant qui peut faire une attaque Kaminsky <https://www.bortzmeyer.org/comment-fonctionne-la-faille-kaminsky.html>, par exemple, peut envoyer les requêtes du serveur ACME chez lui. Plus simple, même si le RFC n'en parle guère (il se focalise sur les attaques DNS, pas sur celles portant sur les noms de domaine), un attaquant qui détourne le nom de domaine, comme vu fin 2018 au Moyen-Orient <https://www.bortzmeyer.org/dnspionage.html>, peut évidemment obtenir les certificats qu'il veut, contrairement à la légende répandue comme quoi TLS protègerait des détournements.

Comment se protéger contre ces attaques ? Le RFC recommande d'utiliser un résolveur DNS validant (vérifiant les signatures DNSSEC) ce que peu d'AC font (Let's Encrypt est une exception), de questionner le DNS depuis plusieurs points de mesure, pour limiter l'efficacité d'attaques contre le routage (cf. celle contre MyEtherWallet en avril 2018), et pourquoi pas d'utiliser TCP plutôt qu'UDP pour les requêtes DNS (ce qui présente l'avantage supplémentaire de priver de certificat les domaines dont les serveurs de noms sont assez stupides pour bloquer TCP). Voir aussi la section 11.2, qui revient sur ces conseils pratiques. Par exemple, une AC ne doit évidemment pas utiliser le résolveur DNS de son opérateur Internet, encore moins un résolveur public.

ACME est un protocole, pas une politique. L'AC reste maitresse de sa poltique d'émission des certificats. ACME ne décrit donc pas les autres vérifications qu'une AC pourrait avoir envie de faire :

2. Car trop difficile à faire afficher par LaTeX

— Acceptation par le client d'un contrat,
— Restrictions supplémentaires sur le nom de domaine,
— Autorisation ou pas des jokers dans le nom demandé,
— Liste noire de noms considérés comme sensibles, par exemple parce désignant telle ou telle marque puissante,
— Tests avec la PSL,
— Tests techniques sur la cryptographie (par exemple rejeter les clés pas assez fortes, ou bien utilisant des algorithmes vulnérables),
— Présence d'un enregistrement CAA (RFC 6844).

Les certificats DV (ceux faits avec ACME) sont sans doute moins fiables que les EV (les DV n'ont qu'une vérification automatique, avec peu de sécurité puisque, par exemple, DNSSEC n'est pas obligatoire) et il est donc prudent de limiter leur durée de validité. Let's Encrypt fait ainsi des certificats à courte durée de vie, seulement trois mois, mais ce n'est pas trop grave en pratique, puisque le renouvellement peut être complètement automatisé.

Quels sont les mises en œuvre disponibles d'ACME ? Comme le RFC est publié longtemps après les premiers déploiements, il y en a déjà pas mal. Let's Encrypt maintient une liste de clients <https://letsencrypt.org/docs/client-options/>. Personnellement, j'ai pratiqué certbot <https://certbot.eff.org/> et dehydrated <https://github.com/lukas2511/dehydrated> mais il en existe d'autres, comme acme-tiny <https://github.com/diafygi/acme-tiny>, qui semble simple et compréhensible. Un avantage que je trouve à dehydrated est qu'il est bien plus simple de garde sa clé lors des renouvellements <https://www.bortzmeyer.org/letsencrypt-certbot-keep-key.html>, par exemple pour DANE : il suffit de mettre `PRIVATE_KEY_RENEW="no"` dans le fichier de configuration. En revanche, dehydrated est à la fois pas assez et trop bavard. Pas assez car il n'a pas d'option permettant de voir la totalité du dialogue en JSON avec le serveur (contrairement à certbot) et trop car il affiche des messages même quand il n'a rien fait (parce que le certificat était encore valide pour assez longtemps). Pas moyen de le faire taire, et rediriger la sortie standard ne marche pas car on veut savoir quand il y a eu renouvellement effectif.

On dispose également de bibliothèques permettant au programmeur ou à la programmeuse de développer plus facilement un client ACME. (Par exemple `Protocol::ACME` (encore que j'ai l'impression qu'il n'est plus maintenu <https://github.com/sludin/Protocol-ACME>, un programmeur Perl disponible pour évaluer ce qui existe ?). Pour les programmeures Python, il y a le module `acme` qui est celui utilisé par le client certbot, mais qui est aussi distribué indépendamment. En Go, il y a LeGo <https://github.com/go-acme/lego>. Mais on peut aussi mettre le client ACME dans le serveur HTTP, comme le permet Apache <https://icing.github.io/mod_md/>

Et les serveurs ACME ? Évidemment, peu de gens monteront une AC mais, si vous voulez le faire, le serveur de Let's Encrypt, Boulder, est en logiciel libre <https://github.com/letsencrypt/boulder>.

Notez que ce RFC ne parle que de la validation de noms de domaines mais ACME pourra, dans le futur, être utilisé pour valider la « possession » d'une adresse IP, ou d'autres identifiants.

Et si vous voulez un résumé rapide d'ACME par ses auteurs, allez lire cet article sur le blog de l'IETF <https://www.ietf.org/blog/acme/>.