Publication des clefs OpenPGP

Damien Goutte-Gattat <dgouttegattat@incenp.org>

2018/08/01

Historique des versions

Version 1.1 2018/08/01
Plus d'informations sur les rôles et propriétés des serveurs de clefs publics.

Version 1.0 2017/05/31
Première version publique.

Résumé

Un problème fréquemment rencontré avec OpenPGP [https://tools.ietf.org/html/rfc4880] est celui de la *distribution* des clefs publiques : comment Alice peut-elle transmettre sa clef publique à Bob, étape préalable indispensable à toute communication sécurisée ?

Depuis les premières versions de PGP, plusieurs méthodes ont été élaborées pour tenter de résoudre ce problème. Cet article les passe en revue.

Table des matières

1. Les serveurs de clefs

La première méthode de publication des clefs, la plus connue et celle historiquement associée au monde OpenPGP, consiste à enregistrer la clef auprès d'un *serveur de clefs*, qui la rendra disponible à quiconque la demande.

1.1. Un peu d'histoire

Les premiers « serveurs de clefs » n'étaient en fait rien de plus que de simples serveurs FTP sur lesquels était disponible un trousseau de clefs publiques, par convention sous le chemin `/pub/pgp/keys/pubring.gpg` et dans un format directement utilisable par PGP. Un exemple d'un tel serveur était `ftp.pgp.net`, dont le contenu était « mirroré » sur plusieurs serveurs nationaux (`ftp.uk.pgp.net`, `ftp.de.pgp.net`, etc.).

Cette méthode est désormais complètement obsolète et la plupart des serveurs FTP de l'époque ne répondent même plus. À l'heure où j'écris ces lignes (décembre 2016), `ftp.pl.pgp.net` est l'un des rares encore actifs. Il héberge un trousseau [ftp://ftp.pl.pgp.net/pub/pgp/keys/pubring.pgp] contenant un peu plus de cinquante mille clefs, la plus récente datant de 2001.

Pour offrir davantage de fonctionnalités que ne le permettait la simple publication d'un trousseau à travers FTP, Michael Graff [http://www.flame.org/~explorer/] a écrit le premier véritable serveur de clefs à proprement parler, le PGP Keyserver Software,[1] utilisant un protocole basé sur le courrier électronique [http://www.pgp.net/pgpnet/email-help-en.html]. Bien que lui aussi tombé aujourd'hui en désuétude, ce protocole est toujours partiellement supporté (en lecture seulement, sans possibilité de déposer une clef) par certains serveurs modernes. Ainsi, on pourra par exemple chercher les clefs associées à l'adresse `alice@example.org` en envoyant à `<pgp-public-keys@pool.sks-keyservers.net>` un message contenant le sujet `INDEX alice@example.org`.

La deuxième génération de serveurs est apparue en 1996 avec le OpenPGP Public Key Server [http://pks.sourceforge.net/] (PKS), écrit par Marc Horowitz dans le cadre de sa thèse au MIT [http://www.mit.edu/afs/net.mit.edu/project/pks/thesis/paper/thesis.html]. Outre le protocole à base d'e-mail repris de la génération précédente, ce serveur a introduit le *HTTP Keyserver Protocol* (HKP). Basé sur le protocole HTTP comme son nom l'indique (quoiqu'il ait parfois été appelé *Horowitz Keyserver Protocol* en référence à son auteur), HKP fut décrit ultérieurement dans un brouillon IETF [https://tools.ietf.org/id/draft-shaw-openpgp-hkp-00.txt]. La spécification n'a jamais atteint le stade de RFC mais le protocole n'en est pas moins resté et est toujours aujourd'hui le protocole utilisé par les serveurs modernes.

Aujourd'hui, les serveurs modernes, justement, tournent pour la plupart sous Synchronizing Key Server [https://bitbucket.org/skskeyserver/sks-keyserver/wiki/Home] (SKS), écrit initialement par Yaron Minsky et qui a largement supplanté PKS. Contrairement à ce que son nom pourrait laisser croire, SKS n'est pas le premier logiciel serveur permettant la synchronisation des serveurs de clefs (PKS le faisait déjà), mais SKS utilise pour cela un protocole dédié (sur le port 11370) couplé à un algorithme de réconciliation permettant de réduire le traffic nécessaire pour synchroniser deux serveurs (PKS utilisait une approche plus naïve et moins efficace).

1.2. Le protocole HKP

Le protocole HKP est une application particulière du protocole HTTP. Par défaut, il utilise le port TCP 11371 ou, dans sa version sécurisée par TLS (HKPS), le port HTTPS standard 443.[2]

Il peut aussi passer par un port arbitraire, que l'on indiquera aux clients via un enregistrement SRV (RFC 2782 [https://tools.ietf.org/html/rfc2782]) de la forme `_pgpkey-http._tcp.`*`servername.example`* (ou `_pgpkey-https` pour la version HKPS). Par exemple, l'enregistrement SRV suivant :

```
_pgpkey-http._tcp.keyserver.example IN SRV 10 0 49152 server1.example
```

informera un client souhaitant utiliser le serveur `keyserver.example` qu'il doit contacter la machine `server1.example` sur le port 49152.

Pour rechercher et récupérer une clef, il faut demander au serveur, via une requête `GET`, une ressource de la forme `/pks/lookup?op=`*`action`*`&search=`*`cible`*, où *`action`* peut

[1] Une copie des sources (non-libres) de la version 2.5 est toujours disponible [ftp://ftp.uni-jena.de/pub/security/PGP/utils/keyserver/keyserver.2.5.tar.asc] pour les curieux.

[2] Outre le chiffrement de la communication, un avantage accessoire de HKPS par rapport à HKP est que le port HTTPS standard a moins de risque d'être bloqué par un pare-feu un peu trop restrictif que le port 11371.

être `index` pour obtenir une liste des clefs correspondant au critère `cible`, ou `get` pour obtenir une clef précise.

Le paramètre *`cible`* sera soit un motif à rechercher dans les noms et adresses e-mails associés aux clefs, soit, s'il est précédé de `0x`, l'empreinte de la clef recherchée, sous sa forme complète (de préférence) ou tronquée aux 8 derniers octets (ce qu'on appelle « l'identifiant long » ou *long key ID*) ou aux 4 derniers octets (« l'identifiant court » ou *short key ID*).

Par défaut, le serveur renvoie une page HTML contenant les informations demandées ; en ajoutant le paramètre `options=mr`, le serveur renvoie une réponse sous forme de texte brut aisément analysable (`mr` signifie *Machine-readable*).

On peut illustrer le fonctionnement du protocole HKP en interrogeant directement un serveur depuis la ligne de commande avec des outils comme wget ou curl. Par exemple, pour chercher la clef du robot *Edward* (mis en place par la FSF dans le cadre de son initiative Autodéfense courriel [https://emailselfdefense.fsf.org/fr/]) :

```
$ wget --quiet -O - \
  'http://pool.sks-keyservers.net:11371/pks/lookup?'\
  'op=index&search=edward@fsf.org&options=mr'
info:1:1
pub:F357AA1A5B1FA42CFD9FE52A9FF2194CC09A61E8:1:2048:1404075834::
uid:Edward the GPG Bot <edward@fsf.org>:1404075834::
uid:Edward, the GPG Bot <edward-en@fsf.org>:1404098786::
uid:Edward, l'amichevole bot GnuPG <edward-it@fsf.org>:1406839147::
uid:Edward, le gentil robot de GnuPG <edward-fr@fsf.org>:1404139478::
[…]
```

Les curieux pourront consulter le brouillon IETF sus-mentionné pour savoir exactement comment interpréter la réponse du serveur (section 5.2, *Machine Readable Indexes*) ; en gros, le serveur indique avoir trouvé une clef publique, et donne les caractéristiques de celle-ci et la liste des *User IDs* associées.

Pour rapatrier la clef trouvée :

```
$ wget --quiet -O -
  'http://pool.sks-keyservers.net:11371/pks/lookup?'\
  'op=get&search=0x9FF2194CC09A61E8&options=mr'
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: SKS 1.1.5
Comment: Hostname: pgp.surfnet.nl

mQENBFOwfzoBCADpwK6sGC3EUzgD7IW1X5ZDR1nC5/rcXacAJLarPpvQBEz4pwjTjoAzATM7
F9RwIzJ3hJTZHiYaQY4cfiGlKSnrd8GPC8A4QkxXIaQ0hLpcsBSbtZJpo2iOzL2fRHmW2Zln
[…]
I6JWazAmnhoRyOEAld6ORPNW1EUPBsIhfazP3v5SG5NXDAjYMHH/MbX872FhoBWerfHpi1yy
ZPHSkkXIUAaY
=exlR
-----END PGP PUBLIC KEY BLOCK-----
```

Pour déposer une clef sur un serveur, on envoie une requête `POST` sur la ressource `/pks/add`, le corps de la requête, au format `x-www-form-urlencoded`, étant simplement constitué d'une variable `keytext` contenant la clef publique en « armure ASCII ».

Par exemple, si le fichier `clef.asc` contient une telle clef publique, on peut l'envoyer au serveur comme suit :

```
$ curl --data-urlencode keytext@clef.asc \
  http://pool.sks-keyservers.net:11371/pks/add
<html><body>Key block added to key server database.
New public keys added: <br>1 key(s) added successfully.</br></html></body>
```

1.3. Les serveurs publics

L'Internet renferme de quelques dizaines à quelques centaines de serveurs de clefs publics (c'est-à-dire utilisables par tout le monde, que ce soit pour y déposer une clef ou pour en rechercher une). Ils sont gérés par des volontaires, qui sont aussi bien des particuliers que des entreprises ou, souvent, des organismes de recherche ou d'enseignement supérieur (par exemple, un des plus anciens serveurs, et un des plus connus, est celui du MIT, `pgp.mit.edu`).

Ces serveurs se synchronisent régulièrement entre eux et le choix du serveur à interroger ou auprès duquel déposer sa clef n'a en principe pas d'importance. En fait, on évitera même, en règle générale, de référencer un serveur particulier par son propre nom (comme `pgp.mit.edu` déjà cité) : on utilisera plutôt des *pools*, des ensembles de serveurs répondant derrière un même nom DNS (à la manière des serveurs NTP du *NTP Pool Project*).

Le principal pool, sks-keyservers.net [https://sks-keyservers.net/], rassemble les serveurs tournant sous SKS ; il est géré par Kristian Fiskerstrand (un des mainteneurs de SKS). À l'heure où j'écris ces lignes, il référence 109 serveurs, joignables derrière le nom générique `pool.sks-keyservers.net`. Des « sous-pools » permettent également de regrouper certains de ces serveurs en fonction de leur localisation géographique (ainsi, `eu.pool.sks-keyservers.net` est un pool de serveurs européens, et `na.pool.sks-keyservers.net` est un pool de serveurs nord-américains) ou de leur connectivité (ainsi, `ipv6.pool.sks-keyservers.net` ne contient que des serveurs joignables en IPv6, et `ha.pool.sks-keyservers.net` ne contient que des serveurs « à haute disponibilité »).

Le sous-pool `hkps.pool.sks-keyservers.net`, lui, regroupe les serveurs offrant la version TLS du protocole HKP, sur le port HTTPS standard 443. À noter, tous les serveurs de ce pool utilisent un certificat X.509 signé par une autorité de certification spécifique [https://sks-keyservers.net/verify_tls.php], gérée par Kristian Fiskerstrand lui-même et exclusivement dédiée aux serveurs SKS.

Ceux qui souhaitent monter leur propre serveur de clefs sous SKS pourront consulter avec profit le document Building a PGP SKS Keyserver [https://keyserver.mattrude.com/guides/building-server/], par Matt Rude.

1.3.1. Rôles des serveurs publics

Les serveurs de clefs publics ont deux cas d'utilisation qu'il est utile de bien différencier : ① la *découverte initiale* d'une clef à partir d'un UID (*User ID*, typiquement une adresse e-mail), et ② la *mise à jour* d'une clef à partir de son adresse.

Dans le premier cas, Bob cherche à obtenir la clef d'Alice sans disposer d'aucune information préalable sur ladite clef : tout ce qu'il a à sa disposition est l'adresse e-mail d'Alice. Il demande donc à un serveur de clefs la (ou les) clef(s) associée(s) à cette adresse. Ici, les serveurs de clefs agissent comme un *annuaire*. Il est important de noter que les responsabilités de cet annuaire sont limitées : en particulier, il ne lui incombe pas de fournir la moindre garantie que la clef renvoyée appartient bien à qui elle prétend appartenir. Il n'y a pas d'*authentification* des clefs, les serveurs de clefs ne sont pas des autorités de certification.

C'est précisément pour pallier cette limitation qu'ont été développées les méthodes alternatives de distribution des clefs, décrites dans la suite de cet article.

Dans le second cas, Bob possède déjà une copie de la clef publique d'Alice, et souhaite la *rafraîchir* pour récupérer toute modification éventuellement apportée par Alice depuis qu'il a obtenu sa clef initialement. Les modifications en question pouvant inclure :

- un changement des algorithmes préférés d'Alice ;
- un changement de la date d'expiration de la clef principale ou des sous-clefs ;
- l'introduction de nouvelles sous-clefs ;
- la révocation de sous-clefs ;
- la révocation de la clef principale elle-même.

Dans ce second cas d'utilisation, le fait que les serveurs de clefs ne fournissent aucune authentification n'est pas un problème. Ce que Bob demande aux serveurs n'est pas « je veux la clef associée à `<alice@example.org>` », mais « je veux (la dernière version de) la clef `DFF9C8A3FE6663F9DD157E16F5C95C96DD4C784D` ». Peu importe qu'il puisse exister plusieurs dizaines de clefs au nom d'Alice sur les serveurs, il n'y en a qu'une seule dont l'empreinte correspond à celle de la vraie clef d'Alice que Bob connaît déjà.

1.3.2. De l'irrévocabilité d'une publication sur les serveurs

Une propriété fondamentale des serveurs de clefs publics est qu'ils fonctionnent en mode *append-only*. Le protocole HKP ne prévoit *pas* la possibilité une clef d'un serveur, ni de retirer des composants obsolètes d'une clef (par exemple retirer une sous-clef devenue inutile parce que révoquée). Même l'administrateur d'un serveur ne peut pas en modifier le contenu (du moins pas facilement ; théoriquement il est évidemment toujours possible d'agir directement sur les fichiers de la base de données du serveur, mais les logiciels serveurs de clefs ne fournissent aucun outil pour ça), et même s'il le pouvait ça ne servirait de toute façon à rien puisqu'une clef retirée *manu militari* d'un serveur, non seulement serait toujours disponible sur les autres serveurs du réseau, mais en plus (comme les serveurs se synchronisent entre eux régulièrement) serait rapidement ré-importée depuis les autres serveurs.

1.4. Les serveurs LDAP

En plus du protocole HKP, il est aussi possible d'utiliser le protocole LDAP [https://tools.ietf.org/html/rfc4511]. PGP Corporation, qui fut un temps l'éditeur de PGP avant son rachat par Symantec, a fourni un schéma permettant de stocker des clefs PGP dans une base LDAP ; ce schéma est aujourd'hui disponible à plusieurs [https://github.com/sp4ke/ldap-openpgp-keyserver/blob/master/schema/pgp-keyserver.schema] endroits [https://github.com/therevmj/schema2ldif/blob/master/schema/pgp-keyserver.schema] sur Internet.

Un avantages des serveurs LDAP sur les serveurs HKP, selon l'utilisation que l'on veut faire du serveur, réside dans la possibilité de mettre en place des contrôles d'accès (ne pas autoriser n'importe qui à consulter ou modifier le contenu du serveur), ce que le protocole HKP ne permet pas.

Du coup, c'est une option intéressante pour un serveur que l'on veut *non-public* ou du moins *pas complètement public* : par exemple, un serveur distribuant les clefs du personnel d'une entreprise (on veut bien que tout le monde puisse y récupérer des clefs, mais seuls les membres du personnel peuvent y déposer une clef, et seulement la leur).

Le wiki de GnuPG contient une page [https://wiki.gnupg.org/LDAPKeyserver] consacrée à l'installation d'un tel serveur.

Une convention initiée par PGP veut que le serveur de l'entreprise *`Example Corporation`*, servant des clefs pour des adresses en `example.com`, soit accessible sous le nom `keys.example.com`. Grâce à cette convention, quand Alice veut envoyer un message à Bob (`bob@example.com`), PGP pourra récupérer automatiquement la clef de Bob en interrogeant `keys.example.com`.

Une variante de ce mécanisme utilise un enregistrement SRV, comme dans l'exemple suivant :

```
_pgpkey-ldap._tcp.example.com. 3600 IN SRV 10 10 389 servername.example.com.
```

qui indique que les clefs pour les adresses en `example.com` peuvent être obtenues auprès du serveur *`servername.example.com`* sur le port 389.

Il faut noter que ces mécanismes de récupération automatique des clefs sur un serveur LDAP ne sont pas implémentés actuellement par GnuPG, même si la page de manuel dit le contraire. Personne ne semble s'être plaint de ce bug.

1.5. Utilisation avec GnuPG

1.5.1. Dirmngr

Depuis la version 2.1, Gpg délègue toutes les interactions avec les serveurs de clefs à Dirmngr, un démon chargé de toutes les tâches impliquant un accès réseau. Comme les autres démons auxiliaires du projet GnuPG (GPG-Agent, Scdaemon), il est automatiquement lancé par Gpg quand il est nécessaire.

Dirmngr utilise par défaut le pool `hkps.pool.sks-keyservers.net` déjà cité, ce qui devrait convenir à la plupart des utilisateurs qui n'ont alors pas besoin de se soucier davantage de Dirmngr.

Ceux qui veulent utiliser un autre serveur ou pool de serveurs devront renseigner l'option `keyserver` dans le fichier de configuration de Dirmngr, soit `$GNUPGHOME/dirmngr.conf`.[3]

Si vous choisissez d'utiliser un serveur HKPS dont le certificat n'est pas signé par une autorité de certification reconnue du système d'exploitation, vous devrez ajouter l'option `hkp-cacert` qui pointera vers un fichier contenant le(s) certificat(s) racine(s) au format PEM. Notez que ce n'est pas nécessaire pour le pool `hkps.pool.sks-keyservers.net`, dont le certificat racine est connu de GnuPG.

Depuis la version 2.1.10, Dirmngr peut utiliser le réseau Tor pour accéder aux serveurs de clefs (cela nécessite évidemment une installation fonctionnelle de Tor), grâce aux options suivantes :

```
use-tor
keyserver hkp://jirk5u4osbsr34t5.onion
```

L'adresse `jirk5u4osbsr34t5.onion` est celle du pool de `sks-keyservers.net`.

Après toute modification du fichier de configuration de Dirmngr, signalez au démon de recharger sa configuration avec l'outil gpgconf :

```
$ gpgconf --reload dirmngr
```

Enfin, ce n'est normalement jamais utile, mais signalons que vous pouvez communiquer directement avec Dirmngr en utilisant l'outil gpg-connect-agent. Par exemple, pour connaître le ou les serveur(s) que Dirmngr est actuellement configuré pour utiliser :

```
$ gpg-connect-agent --dirmngr
> KEYSERVER
S KEYSERVER hkps://hkps.pool.sks-keyservers.net
```

[3] Avant la version 2.1.16, Dirmngr n'avait pas de serveur de clefs configuré par défaut, de sorte qu'il était indispensable de renseigner explicitement l'option `keyserver`.

```
OK
^D
```

Utilisez la commande **HELP** pour la liste des commandes acceptées par le démon.

1.5.2. Chercher et récupérer une clef

Utilisez la commande `--search-keys` de Gpg pour interroger les serveurs de clefs. Par exemple, pour chercher (à nouveau) la clef du robot Edward déjà mentionné :

```
$ gpg2 --search-keys edward-fr@fsf.org
gpg: data source: https://37.191.25.53:443
(1)     Edward the GPG Bot <edward@fsf.org>
        Edward, the GPG Bot <edward-en@fsf.org>
        Edward, l'amichevole bot GnuPG <edward-it@fsf.org>
        Edward, le gentil robot de GnuPG <edward-fr@fsf.org>
          2048 bit RSA key 9FF2194CC09A61E8, created: 2014-06-29
Keys 1-1 for "edward-fr@fsf.org". Enter number(s), N)ext, or Q)quit >
```

Gpg affiche la liste des clefs correspondant au motif que vous avez demandé. Si la clef que vous cherchiez s'y trouve, entrez simplement son numéro pour la télécharger et l'importer dans votre trousseau. Ici, une seule clef a été trouvée et il suffit d'entrer `1` pour la rapatrier :

```
Keys 1-1 of 1 for "edward-fr@fsf.org". Enter number(s) N)ext, or Q)uit  > 1
gpg: key 9FF2194CC09A61E8: public key "Edward, le gentil robot de GnuPG <edward-fr@fsf.org>" import
gpg: Total number processed: 1
gpg:               imported: 1
```

Pour importer depuis les serveurs une clef dont vous connaissez déjà l'empreinte complète ou, à défaut, l'identifiant long (8 derniers octets) ou l'identifiant court (4 derniers octets), utilisez la commande `--receive-keys` (ou `--recv-keys` suivie de l'empreinte ou de l'identifiant :

```
$ gpg2 --receive-keys 9FF2194CC09A61E8
```

Notez qu'il est aujourd'hui pratiquement trivial de générer une clef OpenPGP avec un identifiant court de son choix. Ces identifiants ne devraient plus être utilisés aujourd'hui. Il faut leur préférer les identifiants longs ou, mieux, les empreintes complètes.

1.5.3. Publier une clef

Publier une clef sur les serveurs se fait simplement via l'intermédiaire de la commande `--send-keys`, suivie de l'empreinte ou de l'identifiant (long ou court, comme d'habitude[4]) :

```
$ gpg2 --send-keys 9FF2194CC09A61E8
```

1.5.4. Rafraîchir les clefs

Pour *rafraîchir* une clef déjà dans votre trousseau, c'est-à-dire obtenir la dernière version publiée de cette clef (avec toutes les éventuelles nouvelles identités, sous-clefs, et signatures), utilisez la commande `--refresh-keys`.

[4] De manière générale, sauf mention contraire, chaque fois que le manuel de GnuPG indique qu'une commande attend un *identifiant de clef* (*key ID*), vous pouvez spécifier au choix l'empreinte complète, l'identifiant long, ou l'identifiant court.

Sans arguments, cette commande rafraîchit l'intégralité du trousseau. Elle peut aussi accepter des arguments limitant les clefs à rafraîchir, soit sous la forme d'identifiants de clefs, soit d'identités (complètes ou partielles).

Par exemple, pour rafraîchir toutes les clefs associées à des adresses en `@fsf.org` et `@slackware.com` :

```
$ gpg2 --refresh-keys @fsf.org @example.com
gpg: refreshing 2 keys from hkps://hkps.pool.sks-keyservers.net
gpg: key 9FF2194CC09A61E8: "Edward, le gentil robot de GnuPG <edward-fr@fsf.org>" not changed
gpg: key 6A4463C040102233: "Slackware Linux Project <security@slackware.com>" not changed
gpg: Total number processed: 2
gpg:              unchanged: 2
```

2. Publication des clefs dans le DNS

Au fil du temps, plusieurs méthodes ont été imaginées pour publier une clef OpenPGP dans l'arbre du DNS.

2.1. La méthode moderne : DANE

DANE (*DNS-based Authentication of Named Entities*) est, d'après la charte du groupe de travail [https://tools.ietf.org/wg/dane/charters?item=charter-dane-2017-02-16.txt] du même nom à l'IETF, « un ensemble de mécanismes permettant à des applications Internet d'établir des communications cryptographiquement sûres en exploitant des informations publiées dans le DNS ». En gros, le principe est de publier des clefs cryptographiques sous des noms prévisibles dans le DNS, pour permettre leur découverte et leur récupération automatiques ; la signature des enregistrements DNS via DNSSEC apportant l'authentification nécessaire.

La première production du groupe DANE a été le type d'enregistrement DNS `TLSA` (RFC 6698 [https://tools.ietf.org/html/rfc6698]), utilisé pour publier les certificats X.509 permettant d'authentifier un serveur TLS.

Pour OpenPGP, le groupe DANE a créé le type d'enregistrement `OPENPGPKEY`, standardisé dans le RFC 7929 [https://tools.ietf.org/html/rfc7929]. Le standard définit notamment deux choses : où trouver l'enregistrement (le *owner name* ou *record name*, c'est-à-dire le nom de domaine associé à l'enregistrement), et ce qu'il contient (le *record data* ou `RDATA`).

Le *owner name* d'un enregistrement `OPENPGPKEY` est constitué comme suit :

- les 28 premiers octets du condensat SHA2-256 de la partie locale de l'adresse e-mail associée à la clef, représentés en hexadécimal ;
- le composant fixe `_openpgpkey` ;
- le domaine de l'adresse e-mail.

Par exemple, l'enregistrement `OPENPGPKEY` pour la clef d'Alice `<alice@example.org>` sera situé sous le nom suivant :

```
2bd806c97f0e00af1a1fc3328fa763a9269723c8db8fac4f93af71db._openpgpkey.example.org.
```

Utilisez la commande suivante pour reproduire le premier composant :

```
$ echo -n alice | sha256sum | cut -c1-56
```

Le *RDATA* est simplement composé de la clef publique, sous la forme d'une série de paquets OpenPGP constituant une *Transferable Public Key* comme défini par le RFC 4880 (c'est ce que Gpg produit avec la commande `--export`). Pour minimiser la taille de l'enregistrement, il est recommandé de publier une version minimale de la clef, en supprimant : les *User Attributes* (des images que l'on peut associer à une clef, et qui supposément représentent le visage de son propriétaire), les sous-clefs expirées, les anciennes auto-certifications, et tout ou partie des certifications tierces.

GnuPG prend en charge le RFC 7929 depuis la version 2.1.9. D'une part, il peut récupérer automatiquement une clef publiée dans le DNS via le mécanisme générique `auto-key-locate` (décrit dans une section suivante). D'autre part, il fournit l'option `--export-options export-dane` pour générer l'enregistrement OPENPGPKEY pour une clef donnée :

```
$ gpg2 --export-options export-dane --export alice@example.org
$ORIGIN _openpgpkey.example.org.
; DFF9C8A3FE6663F9DD157E16F5C95C96DD4C784D
; Alice <alice@example.org>
2bd806c97f0e00af1a1fc3328fa763a9269723c8db8fac4f93af71db TYPE61 \# 4011 (
        99020d045860075b011000bbe0b751b46ea0bd28a51e84702ab65efc5211e206
        fccc6d272284386cd45fa2ab425601eca7058d9ef5975495ac95d3426f33fda1
        […]
        2c8766621f7ddd78213e5ea500235b9a95890a98c3b394acc2d2edca98d4b619
        b5f48e189b33de3a1ce0ef
        )
```

La sortie produite est directement intégrable dans un fichier de zone DNS. Notez qu'elle utilise la syntaxe générique du RFC 3597 [https://tools.ietf.org/html/rfc3597], ce qui lui permet d'être lue par un serveur DNS ne reconnaissant pas explicitement le type `OPENPGPKEY`.[5]

Si une clef a plusieurs adresses dans des domaines différents, on peut demander à ne générer les enregistrements que pour les adresses dans un domaine donné grâce à l'option `--export-filter`. Par exemple, si la clef d'Alice est associée aux adresses *`alice@example.org`* et *`alice@other.example`*, la commande suivante permet de ne générer l'enregistrement que pour l'adresse en *`example.org`* :

```
$ gpg2 --export-options export-dane \
  --export-filter "keep-uid=uid =~ example.org" \
  --export alice@example.org
```

Naturellement, il serait complètement irréaliste de demander à chaque utilisateur de publier lui-même sa clef dans le DNS — ne serait-ce que parce tout le monde ne contrôle pas le domaine de son adresse e-mail. DANE pour OpenPGP ne prend tout son sens que si ce sont les fournisseurs de service de messagerie qui s'occupent de publier les clefs de leurs clients. C'est ce que propose par exemple le fournisseur Posteo.de [https://posteo.de/en/site/encryption#posteo-keys].

2.2. Les méthodes historiques

2.2.1. L'enregistrement CERT

Le type d'enregistrement `CERT` est défini par le RFC 4398 [https://tools.ietf.org/html/rfc4398] comme permettant de publier toutes sortes de clefs cryptographiques, notamment des certificats X.509 et des clefs OpenPGP.

[5] Le serveur Bind prend en charge le type `OPENPGPKEY` dans les fichiers de zone depuis ses versions 9.9.7 et 9.10.2.

Aujourd'hui, le type CERT est informellement déprécié et son utilisation est déconseillée y compris par ses inventeurs [https://mailarchive.ietf.org/arch/msg/openpgp/-Xi3-td6He0ZOuoh9ms_j2w7wXQ], au profit des types définis par le groupe DANE (TLSA pour les certificats X.509, OPENPGPKEY pour les clefs OpenPGP).

Le standard définit deux « sous-types » consacrés aux clefs OpenPGP : le sous-type PGP, qui permet de publier une clef complète ; et le sous-type IPGP, qui permet de publier l'empreinte d'une clef, une URL vers la clef proprement dite, ou les deux.

Voici deux exemples d'enregistrements CERT :

```
alice.example.org. IN CERT PGP 0 0 [Clef publique]
alice.example.org. IN CERT IPGP 0 0 14 [Empreinte][URL vers la clef]
```

2.2.2. Les enregistrements PKA

Les enregistrements PKA (*Public Key Association*) ont été inventés indépendamment par les développeurs de GnuPG et n'ont jamais fait l'objet d'une standardisation. Il en existe deux versions.

La première version utilisait un enregistrement de type TXT. Le *owner name* était formé simplement en remplaçant le @ de l'adresse e-mail par le composant _pka ; le RDATA était une chaîne de texte contenant l'empreinte de la clef ou un pointeur vers la clef elle-même, ou les deux.

Voici un exemple d'un tel enregistrement pour la clef d'Alice :

```
alice._pka.example.org. IN TXT "v=1;fpr=DFF9C8A3FE6663F9DD157E16F5C95C96DD4C784D;uri=https://exampl
```

Les enregistrements PKAv1 ne sont plus supportés depuis GnuPG 2.1.3, qui a introduit à la place les enregistrements PKAv2, qui diffèrent de la première version comme suit :

- dans le *owner name*, la partie locale de l'adresse e-mail est encodé en Base32-pour-les-êtres-humains [http://philzimmermann.com/docs/human-oriented-base-32-encoding.txt] ;
- l'enregistrement est du type CERT décrit dans la section précédente, avec le sous-type IPGP ;

Comme pour les enregistrements OPENPGPKEY, GnuPG peut générer un enregistrement PKAv2 grâce à l'option `--export-options export-pka` :

```
$ gpg2 --export-options export-pka --export alice@example.org
$ORIGIN _pka.example.org.
; DFF9C8A3FE6663F9DD157E16F5C95C96DD4C784D
; Alice <alice@example.org>
kei1q4tipxxu1yj79k9kfukdhfy631xe TYPE37 \# 26 0006 0000 00 14 DFF9C8A3FE6663F9DD157E16F5C95C96DD4C7
```

Bien que toujours pris en charge par les dernières versions de GnuPG, les enregistrements PKAv2, non-standardisés et spécifiques à GnuPG, devraient probablement être évités en faveur du type standard OPENPGPKEY.

3. Le protocole OpenPGP Web Key Service

Le *OpenPGP Web Key Service* (WKS) est une initiative des développeurs de GnuPG pour faciliter la distribution des clefs. Elle a été motivée, d'une part, par les problèmes bien connus des serveurs de clefs (notamment, l'absence totale de garantie qu'une clef trouvée sur un tel serveur est légitime), et d'autre part, par la lenteur du déploiement des méthodes basées sur le DNS (notamment, le faible déploiement de DNSSEC). Ce proto-

cole est déjà implémenté par GnuPG et est en cours de standardisation à l'IETF [https://tools.ietf.org/id/draft-koch-openpgp-webkey-service-03.txt].

Il comprend deux éléments distincts : le *Web Key Directory* (WKD) permettant de trouver automatiquement une clef à partir d'une adresse e-mail, et le *Web Key Directory Update Protocol* permettant à un utilisateur de communiquer sa clef publique à son fournisseur de messagerie.

3.1. Le Web Key Directory

Le protocole *Web Key Directory* est assez simple et repose sur deux concepts : un enregistrement SRV (optionel) et une URI « bien connue » (*well-known*) au sens du RFC 5785 [https://tools.ietf.org/html/rfc5785].

Pour illustrer le principe, nous rechercherons la clef de, vous l'avez deviné, Alice `<alice@example.org>`.

La première étape est de demander le (ou les) enregistrement(s) SRV associé(s) au nom `_openpgpkey._tcp.example.org.`, afin d'obtenir un nom d'hôte et un numéro de port. En l'absence d'enregistrement SRV sous ce nom, on prendra par défaut comme nom d'hôte `example.org` et comme numéro de port 443 (le port HTTPS standard).[6] Ici, nous supposerons qu'il n'y a pas d'enregistrement SRV.

La seconde (et dernière) étape est d'envoyer une requête HTTPS à l'adresse « bien connue » suivante :

`https://`*`example.org/`*`.well-known/openpgkpey/hu/`*`kei1q4tipxxu1yj79k9kfukdhfy631xe`*

La chaîne `kei1q4tipxxu1yj79k9kfukdhfy631xe` est le condensat SHA-1 de la partie locale de l'adresse e-mail (`alice`, alice), encodé en Z-Base32.[7] Vous pouvez reproduire cette chaîne comme suit :

```
$ echo -n alice | openssl dgst -sha1 -binary | zbase32
```

Le serveur doit renvoyer directement la clef publique d'Alice.

Comme pour la publication des clefs dans le DNS, il est inimaginable de demander à chaque utilisateur d'avoir son nom de domaine et son serveur web sur lequel publier sa clef. C'est au fournisseur du service de messagerie qu'il reviendra de gérer le Web Key Directory.

GnuPG prend en charge les Web Key Directories via le mécanisme générique `auto-key-locate` décrit plus loin.

3.2. Le Web Key Directory Update Protocol

Le *Web Key Directory Update Protocol* vise à permettre aux utilisateurs d'un service de messagerie de transmettre leur clef publique audit fournisseur, afin que celui-ci approvisionne son *Web Key Directory*.

En dépit de son nom, ce protocole n'est en réalité pas lié au concept de *Web Key Directory* : un *Web Key Directory* peut être approvisionné par d'autres moyens,[8] et le protocole peut

[6] L'intérêt de cette indirection est que le fournisseur du service de messagerie n'a peut-être pas envie de faire fonctionner un serveur web directement sur le nom `example.org`.

[7] L'encodage Z-Base32 [http://philzimmermann.com/docs/human-oriented-base-32-encoding.txt], ou « Base32 pour les êtres humains », est une variante de l'encodage Base32 utilisant un jeu de caractères supposément plus lisible que l'ensemble [A-Z][2-7].

[8] Comme mon propre *Web Key Directory*, par exemple, qui est manuellement approvisionné par mes soins (en même temps ce n'est pas dur, il ne contient qu'une seule clef, la mienne…).

aussi servir à approvisionner d'autres systèmes de distribution de clefs (comme le DNS par exemple).

Le principe repose à nouveau sur une adresse « bien connue », et sur un échange d'e-mails entre l'utilisateur et son fournisseur.

La première étape pour le titulaire de la clef est d'obtenir l'adresse e-mail de *soumission de clef*, par une requête sur l'URL suivante (en supposant une adresse e-mail dans le domaine `example.org`) :

`https://example.org/.well-known/openpgpkey/submission-address`

La réponse du serveur doit consister un une ligne, représentant l'adresse e-mail de soumission. La deuxième étape est d'obtenir la clef publique de chiffrement associée à cette adresse, en interrogeant soit le DNS, soit le *Web Key Directory* de `example.org`.

Une fois en possession de l'adresse de soumission et de la clef associée, le titulaire envoie sa propre clef par mail chiffré à l'adresse en question (les curieux pourront consulter le brouillon IETF sus-mentionné pour le détail du format du message de soumission, et de tous les messages suivants).

À la réception du message de soumission, le fournisseur répond par un message également chiffré (avec la clef publique qu'il vient de recevoir) et contenant un nonce.

Pour prouver à son fournisseur qu'il possède bien la clef privée associée à la clef publique qu'il vient de soumettre, le titulaire doit déchiffrer ce message, en extraire le nonce, et le renvoyer au serveur. Celui-ci peut alors publier la clef dans son Web Key Directory, dans le DNS, ou les deux.

GnuPG fournit une implémentation d'un serveur WKS (**gpg-wks-server**) avec les instructions [https://gnupg.org/blog/20160830-web-key-service.html] pour l'installer et l'intégrer à un serveur de messagerie. Le projet fournit également un client en ligne de commande (**gpg-wks-client**), automatisant toutes les étapes décrites ci-avant. À terme, le protocole devra être implémenté directement dans les logiciels de messagerie (nativement ou par un greffon) — il l'est déjà dans KMail, le client de messagerie de KDE, et il est en cours d'implémentation dans Enigmail, le greffon OpenPGP de Thunderbird.

4. Publication dans les e-mails

Le principe de cette méthode est d'ajouter à chacun de ses e-mails, un en-tête dédié annonçant la clef de l'expéditeur.

Cet en-tête, appelé `OpenPGP`, peut contenir :

- l'empreinte de la clef (que l'on peut remplacer par son identifiant long ou son identifiant court, même si ce n'est pas une bonne idée) ;
- une URL pointant vers la clef proprement dite ;
- une valeur indiquant si l'expéditeur préfère recevoir des messages chiffrés, des messages signés, ou des messages chiffrés *et* signés.

Voici un exemple d'un tel en-tête :

```
OpenPGP: id=DFF9C8A3FE6663F9DD157E16F5C95C96DD4C784D;
 url=https://example.org/~alice/key.txt; preference=signencrypt
```

L'en-tête `OpenPGP` est décrit dans un brouillon IETF [https://www.ietf.org/archive/id/draft-josefsson-openpgp-mailnews-header-07.txt] dont la dernière version a expiré en 2015, et qui n'a donc jamais atteint le stade de RFC. Il est néanmoins implémenté, au minimum, par le greffon Enigmail pour Thunderbird.

5. La découverte automatique des clefs avec GnuPG

Maintenant qu'on sait comment publier une clef, il reste à voir comment *trouver* trouver une clef lorsqu'on en a besoin. Nous avons déjà vu la commande **--search-keys** pour rechercher une clef sur un serveur de clefs.

GnuPG a introduit un mécanisme plus générique appelé *auto-key-locate*, pour obtenir automatiquement une clef à partir d'une adresse e-mail. Le but est que Bob puisse obtenir la clef d'Alice simplement avec la commande suivante :

```
$ gpg2 --locate-keys alice@example.org
```

La découverte automatique des clefs se configure avec l'option `--auto-key-locate`, qui prend en paramètre une liste de mécanismes de publication de clefs à explorer. Par défaut, cette liste est vide et la découverte automatique des clefs est donc complètement désactivée. Les valeurs possibles de la liste sont :

- `keyserver`, pour interroger les serveurs de clefs publics (soit la méthode historique « classique » du monde OpenPGP) ;
- `dane`, pour interroger le DNS à la recherche d'un enregistrement `OPENPGPKEY` (la méthode DNS moderne) ;
- `cert`, pour interroger le DNS à la recherche d'un enregistrement `CERT` (la méthode DNS historique standard) ;
- `pka`, pour interroger le DNS à la recherche d'un enregistrement PKAv2 (la méthode DNS historique propre à GnuPG) ;
- `wkd`, pour interroger le *Web Key Directory* du domaine de l'adresse e-mail.

L'ordre dans lequel ces mécanismes sont spécifiés est significatif : GnuPG testera chaque mécanisme dans l'ordre indiqué jusqu'à obtenir la clef demandée. Par exemple, avec `--auto-key-locate dane,cert,keyserver`, GnuPG cherchera d'abord un enregistrement `OPENPGPKEY`, puis un enregistrement `CERT`, puis interrogera un serveur de clefs.

Avant toute recherche à l'extérieur, GnuPG vérifie préalablement si la clef n'est pas déjà disponible dans le trousseau local. Pour modifier ce comportement, vous pouvez utiliser les valeurs spéciales `nodefault`, qui inhibe complètement la recherche préalable dans le trousseau local, ou `local`, qui permet d'indiquer à quel moment la recherche dans le trousseau local doit avoir lieu. Par exemple, `--auto-key-locate nodefault,wkd` demande à GnuPG de ne chercher la clef demandée que dans les Web Key Directories (sans vérifier dans le trousseau local), tandis que `--auto-key-locate dane,local` lui demande de chercher d'abord dans le DNS et après seulement de vérifier dans le trousseau local.

Enfin, la valeur spéciale `clear` efface la liste courante des mécanismes utilisés. Elle est utile sur la ligne de commande pour ignorer ponctuellement la liste éventuellement définie dans le fichier de configuration de GnuPG.

Voici l'*auto-key-locate* en action :

```
$ gpg2 --auto-key-locate dane,wkd --locate-key alice@example.org
gpg: key F5C95C96DD4C784D: public key "Alice <alice@example.org>" imported
gpg: Total number processed: 1
gpg:               imported: 1
gpg: automatically retrieved 'alice@example.org' via DANE
```

Il s'agit d'un exemple fictif, je n'ai évidemment pas la main sur le domaine `example.org` pour y publier la clef d'Alice… Si vous voulez tester par vous-même, vous pouvez tenter de récupérer ma propre clef (associée à l'adresse

`dgouttegattat@incenp.org`), qui est publiée à la fois dans le DNS et dans un *Web Key Directory*.

6. Et l’authentification des clefs ?

Une question voisine de la *distribution* des clefs est celle de leur *authentification* : une fois que Bob a récupéré, via l’un des canaux décrits dans cet article, une clef prétendant appartenir à Alice, comment peut-il être sûr que c’est effectivement la sienne ?

Répondre à cette question est normalement le rôle du *modèle de confiance* (traité dans un précédent article [https://incenp.org/dvlpt/docs/confiance-openpgp.html]) utilisé par Bob. On n’attend pas, en principe, du mécanisme de distribution de clefs qu’il apporte une quelconque garantie d’authenticité.

C’est particulièrement vrai de la distribution par l’intermédiaire des serveurs de clefs publics : tout le monde peut y déposer une clef associée à n’importe quel nom et n’importe quelle adresse e-mail, c’est à la portée de quiconque peut faire **gpg2 --send-keys**.

En revanche, les méthodes basées sur le DNS et les Web Key Directories nécessitent, pour publier une clef, la participation (et donc le contrôle) du domaine de l’adresse e-mail associée à la clef. Ça ne constitue pas un obstacle insurmontable à la publication d’une fausse clef par un attaquant motivé, mais ça élève significativement la difficulté de la tâche.

En fait, dans GnuPG il était initialement prévu qu’une clef récupérée dans une zone DNS signée par DNSSEC soit automatiquement considérée comme au moins marginalement valide ; la lenteur du déploiement de DNSSEC a finalement conduit les développeurs de GnuPG à abandonner cette idée, au bénéfice du maintien de la stricte séparation entre distribution et authentification : la méthode de récupération d’une clef ne doit pas influencer le modèle de confiance.

Donc, dans le modèle de la toile de confiance (qui est toujours pour l’instant le modèle par défaut de GnuPG), la validité d’une clef récupérée par quelque méthode que ce soit est toujours déterminée uniquement par les certifications portées par cette clef, selon les règles décrites [https://incenp.org/dvlpt/docs/confiance-openpgp.html#sect.wot.rules] dans l’article cité ci-dessus.

À terme, le modèle de confiance par défaut devrait devenir celui de la « confiance au premier contact » (*Trust on First Use*, TOFU), où une clef fraîchement récupérée devrait se voir attribuer une validité au moins marginale, à moins que l’utilisateur n’en décide autrement.