RFC 6238 : TOTP: Time-Based One-Time Password Algorithm

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 7 juin 2019

Date de publication du RFC : Mai 2011

https://www.bortzmeyer.org/6238.html

Ce RFC documente le protocole TOTP, utilisé pour bâtir des systèmes d'authentification à deux facteurs. TOTP est une amélioration du protocole HOTP du RFC 4226 [1], remplaçant le simple compteur par l'heure. Ainsi, il n'est plus nécessaire que les deux machines mémorisent la même séquence.

Mais d'abord, qu'est-ce que l'authentification à deux facteurs ? Imaginons que vous vous connectiez à une interface Web, d'un établissement financier comme Paymium ou PayPal, d'un bureau d'enregistrement de noms de domaines, ou de tout autre service important. La méthode la plus courante d'authentification est le couple {identificateur, mot de passe}. Elle est simple, et facilement expliquable aux utilisat·rice·eur·s. Mais elle pose des problèmes de sécurité : le mot de passe choisi est souvent trop faible <https://xkcd.com/936/>, et les utilisat·rice·eur·s le communiquent parfois à des tiers, par exemple suite à une attaque par ingéniérie sociale. On dit traditionnellement qu'il existe trois voies d'authentification : ce qu'on est, ce qu'on a, ou bien ce qu'on sait. La biométrie utilise la première voie, et le mot de passe la troisième. TOTP, décrit dans ce RFC, va permettre d'utiliser la deuxième voie. Mais on n'utilise pas TOTP seul. Le principe de l'authentification à deux facteurs est d'avoir. . .deux facteurs. Par exemple, sur le Web, le cas le plus courant est un mot de passe classique, plus un code produit par TOTP. Les deux facteurs doivent évidemment être indépendants. Si on se connecte à sa banque depuis son ordiphone et que le générateur TOTP est sur le même ordiphone, il n'y a pas réellement d'indépendance (si le téléphone est volé ou piraté, les deux facteurs peuvent être compromis en même temps). Une solution est d'avoir mot de passe sur l'ordinateur et générateur TOTP sur l'ordiphone, ou bien mot de passe sur l'ordiphone et générateur TOTP sur un dispositif auxiliaire, genre YubiKey.

Pour prendre un exemple récent, lors de l'attaque de 2018 contre de nombreux noms de domaine moyen-orientaux <https://www.bortzmeyer.org/attaques-noms-domaine-explications.

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc4226.txt

html>, l'absence d'authentification à deux facteurs avait certainement joué, rendant relativement facile le piratage en masse.

Mais comment fonctionne TOTP ? L'idée de base est qu'on part d'un secret partagé entre les deux parties, le client et le serveur, secret qui n'est pas envoyé pour l'authentification (et donc moins vulnérable qu'un mot de passe). Ce secret sert à générer un code à usage limité dans le temps (donc, si vous l'avez envoyé au pirate par erreur, ce n'est pas forcément grave). Dans l'ancien protocole HOTP (RFC 4226), la génération du code était contrôlée par le nombre de connexions au service (et il fallait donc que les deux parties communicantes gardent trace du nombre de connexions, et soient bien synchrones sur ce point), dans le nouveau protocole TOTP, objet de ce RFC, c'est l'heure qui gouverne le code généré (et les deux parties doivent donc avoir des horloges à peu près correctes).

Voici ce que voit l'utilisateur avec le logiciel andOTP <https://github.com/andOTP/andOTP> sur Android. On voit la liste des comptes de l'utilisateur (j'ai masqué les identifiants de l'utilisateur, bien qu'ils ne soient pas totalement secrets), et le code courant (je l'ai partiellement masqué mais notez que ce n'est pas nécessaire : ils ne sont utilisables que pendant le bon intervalle).

Un petit rappel sur HOTP : HOTP, normalisé dans le RFC 4226, le code à usage unique était généré en condensant avec SHA-1 la concaténation d'un secret partagé et d'un compteur, chaque partie incrémentant le compteur lors d'une connexion. Il y a donc un risque de désynchronisation si une partie incrémente le compteur alors que l'autre a eu un hoquet et a raté cette incrémentation (RFC 4226, section 7.4).

TOTP, au contraire, utilise un condensat du secret partagé et de l'heure. Il est en effet plus facile d'avoir deux machines ayant une horloge à peu près correcte que d'avoir deux machines restant en accord sur un compteur partagé. (D'autre part, TOTP peut utiliser SHA-2 et plus seulement SHA-1.)

La section 3 du RFC résume quel était le cahier des charges de TOTP : dépendance vis-à-vis de l'heure (plus précisement du temps Unix), avec un intervalle de validité partagé, dépendance vis-à-vis d'un secret partagé entre les deux parties (et avec personne d'autre), secrets qui doivent être bien gardés par les deux parties.

L'algorithme exact (j'ai beaucoup simplifié ici) est en section 4. Un code TOTP est la condensation avec SHA-2 de la concaténation du secret partagé et de l'heure, plus exactement du nombre de secondes depuis l'"*epoch*", divisée par l'intervalle. Ainsi, pour un même couple client/serveur, deux demandes de code très rapprochées donneront le même code, contrairement à HOTP où les codes n'étaient pas réutilisés. Notez que les valeurs de l'"*epoch*" et de l'intervalle ne sont pas spécifiées dans le protocole, elle sont communiquées en dehors du protocole. On ne peut donc pas utiliser TOTP seul et espérer que cela va marcher, il faut ajouter des informations. Cela peut se faire via des cadres de référence comme OATH ou bien dans l'URI fabriqué par le serveur (par exemple otpauth://totp/ACME%20Co:john.doe@example.com pour mettre un intervalle à soixantes secondes, le format exact de ces URI est documenté par Google <https://github.com/google/google-authenticator/wiki/Key-Uri-Format>.)

La section 5, sur la sécurité, est évidement très détaillée. Elle rappelle par exemple que la transmission du secret initial doit être faite sur un canal sécurisé, par exemple avec TLS. D'autre part, ce secret, cette clé, doit être stockée de manière sûre. Si on utilise un ordiphone pour générer les codes, ce qui est fréquent, il faut tenir compte du fait qu'un ordiphone se perd ou se vole facilement. Il ne doit pas divulguer le secret simplement parce qu'un voleur a l'appareil en main. Le RFC recommande que le secret soit chiffré, et de manière à ce que la possession physique de la machine ne permettre pas de le déchiffrer. (Dans andOTP, la base de données est chiffrée avec un mot de passe choisi par l'utilisateur. À l'usage, il est assez pénible de taper ce mot de passe - forcément long et compliqué - très souvent, mais c'est le prix de la sécurité.)

TOTP repose sur l'heure et non seulement les deux horloges ne sont pas parfaitement synchrones mais en outre les délais de transmission du réseau ajoutent une incertitude. On pourrait mettre un intervalle important (la valeur recommandée est de 30 secondes) pour limiter le risque de rejet d'un code mais, d'une part, cela augmente le risque qu'un code volé puisse être réutilisé (il reste valable pendant tout l'intervalle) et d'autre part on peut jouer de malchance, par exemple si un code est généré juste avant la fin d'un intervalle. Le RFC recommande donc d'accepter les codes de l'intervalle précédent l'intervalle courant.

L'annexe A du RFC contient une mise en œuvre de TOTP en Java (avec au moins une bogue <https://www.rfc-editor.org/errata/eid4530>).

Pour envoyer, lors de l'enrôlement initial, le secret partagé qui servira de base au calcul, la méthode la plus courante est que le serveur auprès duquel on s'authentifiera la génère, puis l'affiche sous forme d'un QR-code à lire, ou sous une forme texte. Voilà comment ça se passe chez le bureau d'enregistrement Gandi. On active l'authentification à deux facteurs :

Puis on fait lire à son ordiphone le secret (j'ai masqué QR-code et secret en texte, puisqu'ils doivent rester secrets) :)

Bien d'autres services sur le Web acceptent l'authentification à deux facteurs. C'est le cas entre autres de GitLab (et donc de l'excellent service Framagit <https://framagit.org/>) dans la rubrique « Compte », des places de marché Bitcoin <https://www.bortzmeyer.org/bitcoin-marches.html> comme Kraken <https://www.kraken.com/> (qui le documente très bien <https://support.kraken.com/hc/en-us/articles/360000572186>) ou Paymium <https://www.paymium.com/> (qui, stupidement, ne le documente que comme imposant l'utilisation de Google Authenticator, ce qui est faux, ça a marché avec andOTP, remarquez, Mastodon commet presque la même faute <https://github.com/tootsuite/documentation/blob/master/Using-Mastodon/2FA.md>), des registres comme le RIPE (qui documente également très bien <https://www.ripe.net/participate/member-support/ripe-ncc-access/two-step-verification>), etc. En revanche, les banques, comme le Crédit mutuel ou la Banque postale, imposent un outil privateur, sous forme d'une application fermée, exigeant trop de permissions <https://reports.exodus-privacy.eu.org/en/reports/65980/>, et contenant des pisteurs <https://reports.exodus-privacy.eu.org/en/reports/77314/>. (Alors que le but des normes comme TOTP est justement de permettre le libre choix du logiciel. Mais un tel mépris du client est classique chez les banques.)

Côté client, j'ai choisi andOTP <https://github.com/andOTP/andOTP> sur mon ordiphone mais il y a aussi Aegis <https://github.com/beemdevelopment/Aegis>, FreeOTP <https://freeotp.github.io/>, Google Authenticator <https://support.google.com/accounts/answer/1066447?co=GENIE.Platform%3DAndroid&hl=en>, etc. Si vous êtes fana de la ligne de commande, vous pouvez même directement utiliser l'outil oathtool présenté plus loin, comme décrit dans cet article <https://gitlab.com/gitlab-org/gitlab-ce/issues/27677> (avec un astucieux script pour analyser les URI otpauth:). Il y a aussi des mises en œuvre privatrices dans du matériel spécialisé comme la YubiKey (à partir de 35 € à l'heure actuelle.)

TOTP est à l'origine issu du projet OATH (ne pas confondre avec OAuth.) Une de leurs initiatives est le développement d'outils logiciels pour faciliter le déploiement de techniques d'authentification fortes. C'est par exemple le cas de l'outil libre oathtool <http://www.nongnu.org/oath-toolkit/> et des autres logiciels du "*OATH Toolkit*". Voici une utilisation sur une Debian (le paramètre à la fin est la clé, le secret partagé) :

```
% oathtool --totp=sha256 123456789abcde
839425
```

Si on le lance plusieurs fois de suite, il affichera le même code, jusqu'à la fin de l'intervalle en cours :

```
% oathtool --totp=sha256 123456789abcde
839425
% oathtool --totp=sha256 123456789abcde
839425
% oathtool --totp=sha256 123456789abcde
839425
% oathtool --totp=sha256 123456789abcde
963302
```

Par défaut, oathtool utilise l'horloge courante. On peut lui indiquer une autre heure (cf. sa documentation <http://www.nongnu.org/oath-toolkit/man-oathtool.html> mais attention à ne pas copier/coller les exemples, ça ne marchera pas, les tirets ayant été remplacés par des caractères typographiquement plus jolis, menant à un « *"hex decoding of secret key failed"* » incompréhensible). Essayons avec les vecteurs de test indiqués dans l'annexe B de notre RFC :

```
% oathtool --totp=sha256 --digits=8 3132333435363738393031323334353637383930313233343536373839303132
62307269

% oathtool --totp=sha256 --digits=8 --now "1970-01-01 00:00:59 UTC"  3132333435363738393031323334353637383931
46119246
```

(Et, oui, les valeurs de la clé secrète indiquées dans le RFC sont fausses <https://www.rfc-editor.org/errata/eid5132>.)

Notez aussi que oathtool fait par défaut du HOTP (ici, au démarrage, puis au bout de cinq connexions) :

```
% oathtool 123456789abcde
725666

% oathtool --counter=5 123456789abcde
030068
```

Pour configurer un serveur ssh avec TOTP et l'outil de gestion de configuration Salt, vous pouvez utiliser la recette de Feth Arezki <https://framagit.org/feth/2fa-totp>.

Si on veut développer des services utilisant TOTP, il existe plein d'autres mises en œuvre comme ROTP <http://github.com/mdp/rotp> pour Ruby, onetimepass <https://pypi.org/project/onetimepass/> ou otpauth <https://pythonhosted.org/otpauth/> pour Python, etc. Pour apprendre à utiliser TOTP depuis Python, j'ai apprécié cette transcription de session <https://nbviewer.jupyter.org/github/algorithmic-space/cryptoy/blob/master/rfc6238.ipynb>.

Il ne faut pas croire que l'authentification à deux facteurs est une solution magique. Comme toujours en sécurité, elle vient avec ses propres inconvénients. (Ils sont très bien expliqués dans l'excellent article « *"Before You Turn On Two-Factor Authentication..."* <https://medium.com/@stuartschechter/before-you-turn-on-two-factor-authentication-27148cc5b9a1> ».) Le risque le plus sérieux est probablement le risque de s'« enfermer dehors » : si on perd l'ordiphone ou la clé physique, ou bien s'il tombe en panne, tout est fichu, on ne peut plus se connecter sur aucun des services où on a activé l'authentification à deux facteurs. Une seule solution : les sauvegardes. Avec andOTP, par exemple,

on peut exporter la liste des comptes et leurs secrets dans un fichier en clair (ce qui est évidemment déconseillé), chiffré avec AES et le mot de passe d'andOTP, ou bien chiffré avec PGP, la solution la plus pratique. J'ai installé openKeychain <https://www.openkeychain.org/> sur mon Android, récupéré ma clé PGP et tout marche, les sauvegardes, une fois exportées, sont copiées sur d'autres machines. Si je déchiffre et regarde en quoi consiste la sauvegarde, on y trouve un fichier JSON contenant les informations dont nous avons déjà parlé (la clé secrète), le type d'algorithme, ici TOTP, l'algorithme de condensation, l'intervalle ("*period*"), etc :

```
{
 "secret": "K...",
 "label": "framagit.org - framagit.org:stephane+frama@bortzmeyer.org",
 "digits": 6,
 "type": "TOTP",
 "algorithm": "SHA1",
 "thumbnail": "Default",
 "last_used": 1559836187528,
 "period": 30,
 "tags": []
}
```

Si on veut utiliser ces sauvegardes depuis l'outil en ligne de commande présentée plus haut, il faut prendre note de l'algorithme de condensation (ici SHA-1), du nombre de chiffres (ici 6) et de la clé (qui est encodée en base32) :

```
% oathtool --totp=sha1 --digits 6 --base32 K...
```