RFC 8945 : Secret Key Transaction Authentication for DNS (TSIG)

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 1 décembre 2020

Date de publication du RFC : Novembre 2020

https://www.bortzmeyer.org/8945.html

Le DNS a des vulnérabilités à plusieurs endroits, notamment des risques d'usurpation, qui permettent de glisser une réponse mensongère à la place de la bonne. Il existe plusieurs solutions pour ces problèmes, se différenciant notamment par leur degré de complexité et leur facilité à être déployées. TSIG ("*Transaction SIGnature*"), normalisé dans ce RFC (qui remplace le RFC 2845 [1]), est une solution de vérification de l'intégrité du canal, permettant à deux machines parlant DNS de s'assurer de l'identité de l'interlocuteur. TSIG est surtout utilisé entre serveurs DNS **maîtres** et **esclaves**, pour sécuriser les transferts de zone (aujourd'hui, presque tous les transferts entre serveurs faisant autorité sont protégés par TSIG).

TSIG repose sur l'existence d'une clé secrète, partagée entre les deux serveurs, qui sert à générer un HMAC permettant de s'assurer que le dialogue DNS a bien lieu avec la machine attendue, et que les données n'ont pas été modifiées en route. L'obligation de partager une clé secrète le rend difficilement utilisable, en pratique, pour communiquer avec un grand nombre de clients. Mais, entre deux serveurs faisant autorité pour la même zone, ce n'est pas un problème (section 1.1 du RFC). On peut gérer ces secrets partagés manuellement. Même chose entre un serveur maître et un client qui met à jour les données par "*dynamic update*" (RFC 2136).

Bien sûr, si tout le monde utilisait DNSSEC partout, le problème de sécuriser le transfert de zones entre serveurs faisant autorité serait moins grave (mais attention, DNSSEC ne signe pas les délégations et les colles <https://www.afnic.fr/observatoire-ressources/papier-expert/le-dns-ca-colle-ou-ca-
>, cf. la section 10.2 de notre RFC). En attendant un futur lointain « tout-DNSSEC », TSIG fournit une solution simple et légère pour éviter qu'un méchant ne se glisse dans la conversation entre maître et esclave (par exemple grâce à une attaque BGP) et ne donne à l'esclave de fausses informations. (Il existe

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc2845.txt

aussi des solutions non-DNS comme de transférer la zone avec rsync au dessus de SSH. Notez d'autre part que TSIG peut servir à d'autres choses que le transfert de zones, comme l'exemple des mises à jour dynamiques cité plus haut. Par contre, pour sécuriser la communication entre le client final et le résolveur <https://www.bortzmeyer.org/resolveur-dns.html>, il vaut mieux utiliser DoT - RFC 7858 - ou DoH - RFC 8484.)

Comment fonctionne TSIG ? Les sections 4 et 5 décrivent le protocole. Le principe est de calculer, avec la clé secrète, un HMAC des données transmises, et de mettre ce HMAC (cette « signature ») dans un pseudo-enregistrement TSIG qui sera joint aux données, dans la section additionnelle. À la réception, l'enregistrement TSIG (obligatoirement le dernier de la section additionnelle) sera extrait, le HMAC calculé et vérifié.

Pour éviter des attaques par rejeu, les données sur lesquelles portent le HMAC incluent l'heure. C'est une des causes les plus fréquentes de problèmes avec TSIG : les deux machines doivent avoir des horloges très proches (section 10, qui recommande une tolérance - champ "*Fudge*" - de cinq minutes).

Le format exact des enregistrements TSIG est décrit en section 4. L'enregistrement est donc le dernier, et est mis dans la section additionnelle (voir des exemples plus loin avec dig et tshark). Le type de TSIG est 250 et, évidemment, ces pseudo-enregistrements ne doivent pas être gardés dans les caches. Plusieurs algorithmes de HMAC sont possibles. Un est obligatoire et recommandé, celui fondé sur SHA-256 (que j'utilise dans les exemples par la suite). On peut toujours utiliser SHA-1 et MD5 (avant que vous ne râliez : oui, SHA-1 et MD5 ont de gros problèmes <https://eprint.iacr.org/2020/014.pdf> mais cela n'affecte pas forcément leur utilisation en HMAC, cf. RFC 6151). Un registre IANA <https://www.iana.org/assignments/tsig-algorithm-names/tsig-algorithm-names.xml#tsig-algorithm- contient les algorithmes actuellement possibles (cf. aussi section 6 du RFC).

Le nom dans l'enregistrement TSIG est le nom de la clé (une raison pour bien le choisir, il inclut typiquement le nom des deux machines qui communiquent), la classe est `ANY`, le TTL nul et les données contiennent le nom de l'algorithme utilisé, le moment de la signature, et bien sûr la signature elle-même.

Le fait que la clé soit secrète implique des pratiques de sécurité sérieuses, qui font l'objet de la section 8. Par exemple, l'outil de génération de clés de BIND crée des fichiers en mode `0600`, ce qui veut dire lisibles uniquement par leur créateur, ce qui est la moindre des choses. Encore faut-il assurer la sécurité de la machine qui stocke ces fichiers. (Voir aussi le RFC 2104).

Voyons maintenant des exemples concrets où `polly` (`192.168.2.27`) est un serveur DNS maître pour la zone `example.test` et `jadis` (`192.168.2.29`) un serveur esclave. Pour utiliser TSIG afin d'authentifier un transfert de zone entre deux machines, commençons avec BIND. Il faut d'abord générer une clé. Le programme `tsig-keygen`, livré avec BIND, génère de nombreux types de clés (d'où ses nombreuses options). Pour TSIG, on veut du SHA-256 (mais `tsig-keygen` connait d'autres algorithmes) :

```
% tsig-keygen -a HMAC-SHA256  polly-jadis
key "polly-jadis" {
 algorithm hmac-sha256;
 secret "LnDomTT+IRf0daLYqxkstFpTpmFfcOvyxtRaq2VhHSI=";
};
```

(Avant BIND 9.13, le programme à utiliser était `dnssec-keygen`, qui ne fait désormais plus que du DNSSEC.) On a donné à la clé le nom des deux machines entre lesquelles se fera la communication (le nom est affiché dans le journal lors d'un transfert réussi, et à plusieurs autres endroits, donc il vaut mieux le choisir long et descriptif), plus un chiffre pour distinguer d'éventuelles autres clés. La clé est partagée entre ces deux machines. On met alors la clé dans la configuration de BIND, sur les deux machines (`tsig-keygen` produit directement de la configuration BIND). Naturellement, il faut transférer la clé de manière sécurisée entre les deux machines, pas par courrier électronique ordinaire, par exemple. Sur le serveur maître, on précise que seuls ceux qui connaissent la clé peuvent transférer la zone :

```
zone "example.test" {
         type master;
         file "/etc/bind/example.test";
         allow-transfer {  key polly-jadis; };
};
```

Les autres clients seront rejetés :

```
Apr  6 17:45:15 polly daemon.err named[27960]: client @0x159c250 192.168.2.29#51962 (example.test): zone transfe
```

Mais, si on connait la clé, le transfert est possible (ici, un test avec dig) :

```
% dig -y hmac-sha256:polly-jadis:LnDomTT+IRf0daLYqxkstFpTpmFfcOvyxtRaq2VhHSI= @polly.sources.org AXFR example.te

; <<>> DiG 9.16.1-Debian <<>> -y hmac-sha256 @polly.sources.org AXFR example.test
; (1 server found)
;; global options: +cmd
example.test.  86400 IN SOA polly.sources.org. hostmaster.sources.org. 2020040600 7200 3600 604800 43200
example.test.  86400 IN NS polly.sources.org.
example.test.  86400 IN SOA polly.sources.org. hostmaster.sources.org. 2020040600 7200 3600 604800 43200
polly-jadis.  0 ANY TSIG hmac-sha256. 1586195193 300 32 pJciUWLOkg1lc6J+msC+dzYotVSOr8PSXgE6fFU3UwE= 25875 NOERR
;; Query time: 10 msec
;; SERVER: 192.168.2.27#53(192.168.2.27)
;; WHEN: Mon Apr 06 17:46:33 UTC 2020
;; XFR size: 3 records (messages 1, bytes 267)
```

On voit que dig affiche fidèlement tous les enregistrements, y compris le pseudo-enregistrement de type TSIG qui contient la signature. Un point de sécurité, toutefois (merci à Jan-Piet Mens pour sa vigilance) : avec ce `-y`, quiconque a un compte sur la même machine Unix peut voir la clé avec `ps`. Il peut être préférable de la mettre dans un fichier et de dire à dig d'utiliser ce fichier (`dig -k polly-jadis.key ...`).

Maintenant que le test marche, configurons le serveur BIND esclave :

```
// To avoid copy-and-paste errors, you can also redirect tsig-keygen's
// output to a file and then include it in named.conf with 'include
// polly-jadis.key'.
key "polly-jadis" {
    algorithm hmac-sha256;
              secret "LnDomTT+IRf0daLYqxkstFpTpmFfcOvyxtRaq2VhHSI=";
              };

zone "example.test" {
       type slave;
       masters {192.168.2.27;};
       file "example.test";
};

server 192.168.2.27 {
        keys {
          polly-jadis;
        };
};
```

et c'est tout : l'esclave va désormais utiliser TSIG pour les transferts de zone. Le programme tshark va d'ailleurs nous afficher les enregistrements TSIG :

```
    Queries
        example.test: type AXFR, class IN
            Name: example.test
            [Name Length: 12]
            [Label Count: 2]
            Type: AXFR (transfer of an entire zone) (252)
            Class: IN (0x0001)
    Additional records
        polly-jadis: type TSIG, class ANY
            Name: polly-jadis
            Type: TSIG (Transaction Signature) (250)
            Class: ANY (0x00ff)
            Time to live: 0
            Data length: 61
            Algorithm Name: hmac-sha256
            Time Signed: Apr  6, 2020 19:51:36.000000000 CEST
            Fudge: 300
            MAC Size: 32
            MAC
                [Expert Info (Warning/Undecoded): No dissector for algorithm:hmac-sha256]
                    [No dissector for algorithm:hmac-sha256]
                    [Severity level: Warning]
                    [Group: Undecoded]
            Original Id: 37862
            Error: No error (0)
            Other Len: 0
```

Et si l'esclave est un nsd et pas un BIND ? C'est le même principe. On configure l'esclave :

```
key:
    name: polly-jadis
    algorithm: hmac-sha256
    secret: "LnDomTT+IRf0daLYqxkstFpTpmFfcOvyxtRaq2VhHSI="

zone:
    name: "example.test"
    zonefile: "example.test"
    allow-notify: 192.168.2.27 NOKEY
    request-xfr: 192.168.2.27 polly-jadis
```

Et le prochain `nsd-control transfer example.test` (ou bien la réception de la notification) déclenchera un transfert, qui sera ainsi enregistré :

```
Apr 07 06:46:48 jadis nsd[30577]: notify for example.test. from 192.168.2.27 serial 2020040700
Apr 07 06:46:48 jadis nsd[30577]: [2020-04-07 06:46:48.831] nsd[30577]: info: notify for example.test. from
Apr 07 06:46:48 jadis nsd[30565]: xfrd: zone example.test written received XFR packet from 192.168.2.27 with
Apr 07 06:46:48 jadis nsd[30565]: [2020-04-07 06:46:48.837] nsd[30565]: info: xfrd: zone example.test writte
```

Pour générer des clés sans utiliser BIND, on peut consulter mon autre article sur TSIG <https://www.bortzmeyer.org/tsig-sans-bind.html>.

On l'a vu, TSIG nécessite des horloges synchronisées. Une des erreurs les plus fréquentes est d'oublier ce point. Si une machine n'est pas à l'heure, on va trouver dans le journal des échecs TSIG comme (section 5.2.3 du RFC), renvoyés avec le code de retour DNS `NOTAUTH` (code 9) et un code `BADTIME` dans l'enregistrement TSIG joint :

```
Mar 24 22:14:53 ludwig named[30611]: client 192.168.2.7#65495: \
         request has invalid signature: TSIG golgoth-ludwig-1: tsig verify failure (BADTIME)
```

Ce nouveau RFC ne change pas le protocole du RFC 2845. Un ancien client peut interopérer avec un nouveau serveur, et réciproquement. Seule change vraiment la procédure de vérification, qui est unilatérale. Une des motivations pour la production de ce nouveau RFC venait de failles de sécurité de 2016-2017 (CVE-2017-3142, CVE-2017-3143 et CVE-2017-11104). Certes, elle concernaient des mises en œuvre, pas vraiment le protocole, mais elles ont poussé à produire un RFC plus complet et plus clair.

Le débat à l'IETF au début était « est-ce qu'on en profite pour réparer tout ce qui ne va pas dans le RFC 2845, ou bien est-ce qu'on se limite à réparer ce qui est clairement cassé ? » Il a été tranché en faveur d'une solution conservatrice : le protocole reste le même, il ne s'agit pas d'un « TSIG v2 ».