# RFC 2845 : Secret Key Transaction Authentication for DNS (TSIG)

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 25 mars 2010

Date de publication du RFC : Mai 2000

https://www.bortzmeyer.org/2845.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 initialement dans ce RFC, 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.) Depuis, le RFC 8945 [1] a été publié et est désormais la norme.

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. 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 1.4). Même chose entre un serveur maître et un client qui met à jour les données par "*dynamic update*" (section 1.3 et RFC 2136). En revanche, pour permettre à des clients d'authentifier leur résolveur, il vaudrait mieux utiliser SIG(0) (RFC 2931) (ou des techniques non-DNS comme IPsec). La section 1.2 de notre RFC cite pourtant cet exemple d'authentification du résolveur, qui n'a jamais été déployé en pratique.

Bien sûr, si tout le monde utilisait DNSSEC partout, le problème n'existerait pas. Mais, en attendant ce futur lointain, 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

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc8945.txt

fausses informations. (Il existe aussi des solutions non-DNS comme de transférer la zone avec rsync au dessus de SSH.)

Comment fonctionne TSIG ? Les sections 3 et 4 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 3.3, qui précise que la tolérance - champ "*Fudge*" - est de cinq minutes).

Le format exact des enregistrements TSIG est décrit en section 2. 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 mais un seul est obligatoire (et est le plus répandu), MD5 (RFC 1321 et RFC 2104). Un registre IANA <https://www.iana.org/assignments/tsig-algorithm-names> contient les algorithmes actuellement possibles. (Le RFC 4635 a ajouté ceux de la famille SHA.)

Le nom dans l'enregistrement TSIG est le nom de la clé (une raison pour bien le choisir), 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 des sections 5 et 6. 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.

Voyons maintenant des exemples concrets où `ludwig` (`192.168.2.1`) est un serveur DNS maître pour la zone `example.test` et `golgoth` (`192.168.2.7`) 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 `dnssec-keygen`, livré avec BIND, génère de nombreux types de clés (d'où ses nombreuses options). Pour TSIG, on veut du MD5 (mais `dnssec-keygen` connait d'autres algorithmes, tapez la commande sans arguments pour avoir la liste) :

```
% dnssec-keygen -a HMAC-MD5 -b 512  -n HOST  golgoth-ludwig-1
Kgolgoth-ludwig-1.+157+56575
```

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 trouve la même dans le fichier `Kgolgoth-ludwig-1.+157+56575.private` et dans `Kgolgoth-ludwig-1.+157+56575.key` :

```
% cat Kgolgoth-ludwig-1.+157+56575.private
Private-key-format: v1.2
Algorithm: 157 (HMAC_MD5)
Key: iGSDB9st+5/xq0AZnwGWAv2ToNJFjsB33fXOh8S9VaI26k4SS7zm4uJVD2MBbYviLB9pF1fWZSUAanOYaRa9JQ==
Bits: AAA=
```

On met alors la clé dans la configuration de BIND, sur les deux machines :

```
key golgoth-ludwig-1. {
algorithm "hmac-md5";
secret "iGSDB9st+5/xq0AZnwGWAv2ToNJFjsB33fXOh8S9VaI26k4SS7zm4uJVD2MBbYviLB9pF1fWZSUAanOYaRa9JQ==";
};
```

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 golgoth-ludwig-1; };
};
```

Les autres seront rejetés :

```
Mar 24 21:41:01 ludwig named[30611]: client 192.168.2.7#65523: zone transfer 'example.test/AXFR/IN' denied
```

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

```
% dig -y hmac-md5:golgoth-ludwig-1:iGSDB9st+5/...Ra9JQ== @192.168.2.1 AXFR example.test

; <<>> DiG 9.5.0-P2 <<>> -y hmac-md5 @192.168.2.1 AXFR example.test
; (1 server found)
;; global options:  printcmd
example.test.           86400   IN      SOA     ludwig.example.test. hostmaster.example.test. 2010032400 36000 3
example.test.           86400   IN      NS      ludwig.example.test.
example.test.           86400   IN      TXT     "foobar"
example.test.           86400   IN      SOA     ludwig.example.test. hostmaster.example.test. 2010032400 36000 3
golgoth-ludwig-1.     0       ANY     TSIG    hmac-md5.sig-alg.reg.int. 1269463439 300 16 6TTFl73Lzd5KdDs367+ZoA
;; Query time: 29 msec
;; SERVER: 192.168.2.1#53(192.168.2.1)
;; WHEN: Wed Mar 24 21:43:59 2010
;; XFR size: 4 records (messages 1, bytes 244)
```

On voit que dig affiche fidèlement tous les enregistrements, y compris le pseudo-enregistrement de type TSIG qui contient la signature.

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

```
zone "example.test" {
       type slave;
       masters {192.168.2.1;};
       file "example.test";
};

server 192.168.2.1 {
        keys {
     golgoth-ludwig-1;
    };
};
```

et c'est tout : l'esclava 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
            Type: AXFR (Request for full zone transfer)
            Class: IN (0x0001)
    Answers
        example.test: type SOA, class IN, mname ludwig.example.test
            Name: example.test
    ...
    Additional records
        golgoth-ludwig-1: type TSIG, class ANY
            Name: golgoth-ludwig-1
            Type: TSIG (Transaction Signature)
            Class: ANY (0x00ff)
            Time to live: 0 time
            Data length: 58
            Algorithm Name: hmac-md5.sig-alg.reg.int
            Time signed: Mar 24, 2010 21:51:17.000000000
            Fudge: 300
            MAC Size: 16
            MAC
                No dissector for algorithm:hmac-md5.sig-alg.reg.int
            Original Id: 59013
            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: golgoth-ludwig-1
algorithm: hmac-md5
secret: "iGSDB9st+5/xq0AZnwGWAv2ToNJFjsB33fXOh8S9VaI26k4SS7zm4uJVD2MBbYviLB9pF1fWZSUAanOYaRa9JQ=="

zone:
name: "example.test"
zonefile: "example.test"

allow-notify: 192.168.2.1 NOKEY
request-xfr: 192.168.2.1 golgoth-ludwig-1
```

Et le prochain `nsdc update` (ou bien la réception de la notification) déclenchera un transfert, qui sera ainsi enregistré :

```
Mar 24 22:10:07 golgoth nsd[9885]: Notify received and accepted, forward to xfrd
Mar 24 22:10:07 golgoth nsd[9966]: Handle incoming notify for zone example.test
Mar 24 22:10:07 golgoth nsd[9966]: xfrd: zone example.test written \
          received XFR from 192.168.2.1 with serial 2010032403 to disk
Mar 24 22:10:07 golgoth nsd[9966]: xfrd: zone example.test committed \
          "xfrd: zone example.test received update to serial 2010032403 at time 1269465007 \
          from 192.168.2.1 in 1 parts TSIG verified with key golgoth-ludwig-1"
```

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 (sections 4.5.2 et 4.6.2 du RFC) :

```
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)
```