RFC 8422 : Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) Versions 1.2 and Earlier

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 7 août 2018

Date de publication du RFC : Août 2018

https://www.bortzmeyer.org/8422.html

Ce RFC décrit les algorithmes cryptographiques à base de courbes elliptiques utilisés dans TLS. Il remplace le RFC 4492 [1].

Plus exactement, il normalise les algorithmes utilisés dans les versions de TLS allant jusqu'à 1.2 incluse. L'utilisation des courbes elliptiques par TLS 1.3 est décrite dans le RFC sur TLS 1.3, le RFC 8446. Les deux points importants de ce nouveau RFC sont :

— L'utilisation de courbes elliptiques pour un échange Diffie-Hellman de la clé de session TLS (ce qu'on nomme ECDHE),
— Les algorithmes de signature à courbes elliptiques ECDSA et EdDSA pour authentifier le pair TLS.

Commençons par l'échange de clés de session (section 2). TLS nécessite que les deux pairs se mettent d'accord sur une clé de chiffrement symétrique qui sera ensuite utilisée pendant toute la session, avec des algorithmes comme AES. Une des façons de synchroniser cette clé de session est qu'un des pairs la génère aléatoirement, puis la chiffre avec la clé publique (chiffrement asymétrique) de son pair avant de lui transmettre (cela marche avec RSA mais je n'ai pas l'impression qu'il y ait un moyen normalisé de faire cela avec les courbes elliptiques). Une autre façon est d'utiliser un échange Diffie-Hellman. Contrairement à l'échange Diffie-Hellman originel, celui présenté dans ce RFC, ECDHE, utilise la cryptographie sur courbes elliptiques. (J'ai simplifié pas mal : par exemple, l'échange ECDHE ne donnera pas directement la clé de session, celle-ci sera en fait dérivée de la clé obtenue en ECDHE.) Le principal avantage de Diffie-Hellman est de fournir de la sécurité même en cas de compromission ultérieure de la clé privée.

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc4492.txt

Notre RFC présente trois variantes d'ECDHE, selon la manière dont l'échange est authentifié, l'une utilisant ECDSA ou EdDSA, l'autre utilisant le traditionnel RSA, et la troisième n'authentifiant pas du tout, et étant donc vulnérable aux attaques de l'Homme du Milieu, sauf si une authentification a lieu en dehors de TLS. (Attention, dans le cas de ECDHE_RSA, RSA n'est utilisé que pour authentifier l'échange, la génération de la clé se fait bien en utilisant les courbes elliptiques.)

Lorsque l'échange est authentifié (`ECDHE_ECDSA` - qui, en dépit de son nom, inclut EdDSA - ou bien `ECDHE_RSA`), les paramètres ECDH (Diffie-Hellman avec courbes elliptiques) sont signés par la clé privée (ECDSA, EdDSA ou RSA). S'il n'est pas authentifié (`ECDH_anon`, mais notez que le nom est trompeur, bien que le E final - "*ephemeral*" - manque, la clé est éphémère), on n'envoie évidemment pas de certificat, ou de demande de certificat.

Voilà, avec cette section 2, on a pu générer une clé de session avec Diffie-Hellman, tout en authentifiant le serveur avec des courbes elliptiques. Et pour l'authentification du client ? C'est la section 3 de notre RFC. Elle décrit un mécanisme `ECDSA_sign` (là encore, en dépit du nom du mécanisme, il fonctionne aussi bien pour EdDSA), où le client s'authentifie en signant ses messages avec un algorithme à courbes elliptiques.

Les courbes elliptiques ont quelques particularités qui justifient deux extensions à TLS que présente la section 4 du RFC. Il y a "*Supported Elliptic Curves Extension*" et "*Supported Point Formats Extension*", qui permettent de décrire les caractéristiques de la courbe elliptique utilisée (on verra plus loin que la deuxième extension ne sert plus guère). Voici, vue par tshark, l'utilisation de ces extensions dans un `ClientHello` TLS envoyé par OpenSSL :

```
Extension: elliptic_curves
               Type: elliptic_curves (0x000a)
               Length: 28
               Elliptic Curves Length: 26
               Elliptic curves (13 curves)
                   Elliptic curve: secp256r1 (0x0017)
                   Elliptic curve: secp521r1 (0x0019)
                   Elliptic curve: brainpoolP512r1 (0x001c)
                   Elliptic curve: brainpoolP384r1 (0x001b)
                   Elliptic curve: secp384r1 (0x0018)
                   Elliptic curve: brainpoolP256r1 (0x001a)
                   Elliptic curve: secp256k1 (0x0016)
...
Extension: ec_point_formats
               Type: ec_point_formats (0x000b)
               Length: 4
               EC point formats Length: 3
               Elliptic curves point formats (3)
                   EC point format: uncompressed (0)
                   EC point format: ansiX962_compressed_prime (1)
                   EC point format: ansiX962_compressed_char2 (2)
```

La section 5 du RFC donne les détails concrets. Par exemple, les deux extensions citées plus haut s'écrivent, dans le langage de TLS (cf. section 4 du RFC 5246) :

```
enum {
          elliptic_curves(10),
          ec_point_formats(11)
} ExtensionType;
```

La première extension permet d'indiquer les courbes utilisées. Avec celles du RFC 7748, cela donne, comme possibilités :

```
enum {
               deprecated(1..22),
               secp256r1 (23), secp384r1 (24), secp521r1 (25),
               x25519(29), x448(30),
               reserved (0xFE00..0xFEFF),
               deprecated(0xFF01..0xFF02),
               (0xFFFF)
} NamedCurve;
```

`secp256r1` est la courbe P-256 du NIST, `x25519` est la Curve-25519 de Bernstein. Notez que beaucoup des courbes de l'ancien RFC 4492, jamais très utilisées, ont été abandonnées. (Les courbes se trouvent dans un registre IANA <`https://www.iana.org/assignments/tls-parameters/tls-parameters.xml#tls-parameters-8`>.)

Normalement, dans TLS, on peut choisir séparément l'algorithme de signature et celui de condensation (cf. section 7.4.1.4.1 du RFC 5246). Avec certains algorithmes comme EdDSA dans sa forme « pure », il n'y a pas de condensation séparée et un « algorithme » bidon, `Intrinsic` (valeur 8 <`https://www.iana.org/assignments/tls-parameters/tls-parameters.xml#tls-parameters-18`>) a été créé pour mettre dans le champ « algorithme de condensation » de l'extension `signature_algorithms`.

Voici une négociation TLS complète, vue par curl :

```
%  curl -v https://www.nextinpact.com
...
* Connected to www.nextinpact.com (2400:cb00:2048:1::6819:f815) port 443 (#0)
...
* Cipher selection: ALL:!EXPORT:!EXPORT40:!EXPORT56:!aNULL:!LOW:!RC4:@STRENGTH
...
* SSL connection using TLSv1.2 / ECDHE-ECDSA-AES128-GCM-SHA256
* ALPN, server accepted to use h2
* Server certificate:
*  subject: C=US; ST=CA; L=San Francisco; O=CloudFlare, Inc.; CN=nextinpact.com
...
> GET / HTTP/1.1
> Host: www.nextinpact.com
> User-Agent: curl/7.52.1
> Accept: */*
```

On voit que l'algorithme utilisé par TLS est `ECDHE-ECDSA-AES128-GCM-SHA256`, ce qui indique ECDHE avec ECDSA. Le certificat du serveur doit donc inclure une clé « courbe elliptique ». Regardons ledit certificat :

```
% openssl s_client -connect www.nextinpact.com:443 -showcerts | openssl x509 -text
Certificate:
...
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: C = GB, ST = Greater Manchester, L = Salford, O = COMODO CA Limited, CN = COMODO ECC Domain Vali
...
        Subject: OU = Domain Control Validated, OU = PositiveSSL Multi-Domain, CN = ssl378410.cloudflaressl.com
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
...
                ASN1 OID: prime256v1
                NIST CURVE: P-256
...
            X509v3 Subject Alternative Name:
                DNS:ssl378410.cloudflaressl.com, DNS:*.baseballwarehouse.com, DNS:*.campusgroups.com, DNS:*.cret
    Signature Algorithm: ecdsa-with-SHA256
```

On a bien une clé sur la courbe P-256.

Quel est l'état des mises en œuvre de ces algorithmes dans les bibliothèques TLS existantes ? ECDHE et ECDSA avec les courbes NIST sont très répandus. ECDHE avec la courbe Curve25519 est également dans plusieurs bibliothèques TLS. Par contre, EdDSA, ou ECDHE avec la courbe Curve448, sont certes implémentés mais pas encore largement déployés.

Les changements depuis le RFC 4492 sont résumés dans l'annexe B. Souvent, une norme récente ajoute beaucoup de choses par rapport à l'ancienne mais, ici, pas mal de chose ont au contraire été retirées :

— Plusieurs courbes peu utilisées disparaissent,
— Il n'y a plus qu'un seul format de point accepté ("*uncompressed*"),
— Des algorithmes comme toute la famille ECDH_ECDSA (ECDH et pas ECDHE, donc ceux dont la clé n'est pas éphémère) sont retirés.

Parmi les ajouts, le plus important est évidemment l'intégration des « courbes Bernstein », Curve25519 et Curve448, introduites par le RFC 7748. Et il y avait également plusieurs erreurs techniques dans le RFC 4492 <https://www.rfc-editor.org/errata/rfc4492>, qui sont corrigées par notre nouveau RFC.

Et, pendant que j'y suis, si vous voulez générer un certificat avec les courbes elliptiques, voici comment faire avec OpenSSL :

```
% openssl ecparam -out ec_key.pem -name prime256v1 -genkey
% openssl req -new -key ec_key.pem  -nodes -days 1000 -out cert.csr
```

J'ai utilisé ici la courbe P-256 (`prime256v1` est encore un autre identificateur pour la courbe NIST P-256, chaque organisme qui normalise dans ce domaine ayant ses propres identificateurs). Si vous voulez la liste des courbes que connait OpenSSL :

```
% openssl ecparam -list_curves
```

Ce blog est accessible en TLS <https://www.bortzmeyer.org/https-blog.html> mais pas avec des courbes elliptiques. En effet, l'AC que j'utilise, CAcert <https://www.bortzmeyer.org/cacert.html>, ne les accepte hélas pas (« *"The keys you supplied use an unrecognized algorithm. For security reasons these keys can not be signed by CAcert."* ») Il y a des raisons pour cela <https://blog.cacert.org/2014/03/cacert-enforces-rules-for-signing-certificates-cacert-verscharft-di> mais c'est quand même déplorable. (Enfin, j'accepte quand même ECDHE.)

Enfin, un échange TLS complet vu par tshark est visible ici (en ligne sur `https://www.bortzmeyer.org/files/tls-ec-echange.txt`).

Merci à Manuel Pégourié-Gonnard pour sa relecture vigilante.