Document d'audit du protocole SSL

Version	0.1
Date	11/02/2014
Rédigé par	Julien Legras
Relu par	
Approuvé par	

MISES À JOUR

Version	Date	Modifications réalisées
0.1	12/12/2013	Création du document

Table des matières

1 Objet

Ce document présente l'audit réalisé sur la partie SSL/TLS d'OpenSSL. Pour chaque partie, il y aura une présentation des recommandations/spécifications du protocole puis l'analyse du code associé dans OpenSSL.

2 Terminologie et sigles utilisés

- **RFC** : Les RFC (Request For Comments) sont un ensemble de documents qui font référence auprès de la Communauté Internet et qui décrivent, spécifient, aident à l'implémentation, standardisent et débattent de la majorité des normes, standards, technologies et protocoles liés à Internet et aux réseaux en général.
- **SSL** : Secure Sockets Layer
- **TLS** : Transport Layer Security
- **IETF** : The Internet Engineering Task Force est un groupe informel, international, ouvert à tout individu, qui participe à l'élaboration de standards Internet.
- **IANA** : Internet Assigned Numbers Authority
- **KeyExch** : Échange de clef
- **Authn** : Authentification
- **Enc** : Chiffrement
- **MAC** : Message Authentication Code

3 Description

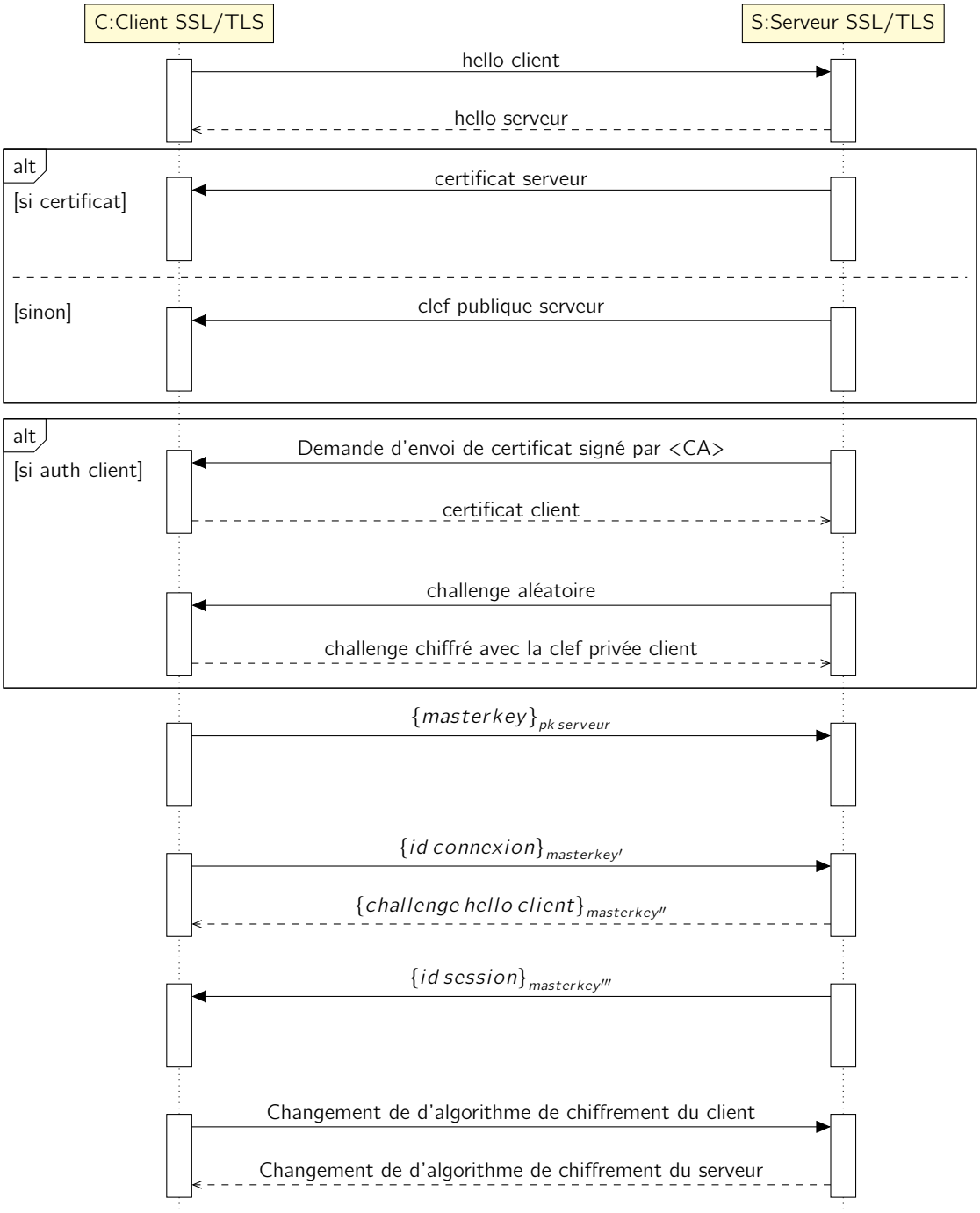
SSL et TLS (successeur de SSL) sont des protocoles de sécurisation des échanges sur internet. Les version 2 et 3 de SSL ont été développées par Netscape puis le brevet a été racheté par l'IETF en 2001 qui a publié une évolution de ce protocole en TLS. Ce protocole fonctionne selon un mode client-serveur et fournit les objectifs de sécurité suivants :

- authentification serveur/client ;
- confidentialité des données échangées ;
- intégrité des données échangées.

Du point de vue réseau, ce protocole se situe dans la couche session du modèle OSI et entre transport et application dans le modèle TCP.

4 Schéma global d'une connexion SSL/TLS

Pour simplifier la compréhension des parties suivantes, voici un schéma qui représente de manière large l'établissement d'une connexion SSL/TLS. Les données entre accolades sont chiffrées avec la clef indiquée en indice. La *masterkey* est la clef principale qui sera dérivée pour chiffrer chaque message.



hello client Version du protocole SSL avec laquelle le client souhaite communiquer, challenge, algorithmes de chiffrement supportés par le client, méthodes de compressiosn supportées par le client.

hello serveur Version du protocole SSL calculée par le serveur (plus haute version du serveur supportée également par le client), challenge, id de session, algorithmes de chiffrement supportés par le serveur, méthodes de compressiosn supportées par le serveur.

Sources = `http://www.symantec.com/connect/articles/apache-2-ssltls-step-step-part-1`, `http://datatracker.ietf.org/doc/rfc6101/?include_text=1`

5 SSL version 2

5.1 Spécifications

Il n'existe pas de RFC pour SSL version 2. En effet, ce protocole a été pensé et développé par la société Netscape Communications. Cette version est sortie en 1994. Toutefois, on trouve des morceaux d'informations dans certaines RFC (6176) et le draft de Hickman (`http://tools.ietf.org/html/draft-hickman-netscape-ssl-00`).

Algorithmes supportés

Identifiant	KeyExch	Authn	Enc	MAC
SSL_CK_RC2_128_CBC_WITH_MD5	RSA	RSA	RC2.128 CBC	MD5
SSL_CK_RC2_128_CBC_EXPORT40_WITH_MD5	RSA.512	RSA	RC4.40 CBC	MD5
SSL_CK_IDEA_128_CBC_WITH_MD5	RSA	RSA	IDEA.128 CBC	MD5
SSL_CK_DES_64_CBC_WITH_MD5	RSA	RSA	DES.56 CBC	MD5
SSL_CK_DES_192_EDE3_CBC_WITH_MD5	RSA	RSA	3DES.168 CBC	MD5
SSL_CK_RC4_128_WITH_MD5	RSA	RSA	RC4.128	MD5
SSL_CK_RC4_128_EXPORT40_WITH_MD5	RSA.512	RSA	RC4.40	MD5

Remarque Le `CK` signifie `CIPHER-KIND`.

5.2 Implémentation

Dans le code d'OpenSSL, cette version du protocole SSL se trouve dans les fichiers commençant pas `s2_` du répertoire `ssl/`. Les constantes sont déclarées dans le fichier `ssl2.h`, on retrouve bien les algorithmes du draft :

Identifiant	Constante OpenSSL
SSL_CK_RC2_128_CBC_WITH_MD5	SSL2_CK_RC2_128_CBC_WITH_MD5
SSL_CK_RC2_128_CBC_EXPORT40_WITH_MD5	SSL2_CK_RC2_128_CBC_EXPORT40_WITH_MD5
SSL_CK_IDEA_128_CBC_WITH_MD5	SSL2_CK_IDEA_128_CBC_WITH_MD5
SSL_CK_DES_64_CBC_WITH_MD5	SSL2_CK_DES_64_CBC_WITH_MD5
SSL_CK_DES_192_EDE3_CBC_WITH_MD5	SSL2_CK_DES_192_EDE3_CBC_WITH_MD5
SSL_CK_RC4_128_WITH_MD5	SSL2_CK_RC4_128_WITH_MD5
SSL_CK_RC4_128_EXPORT40_WITH_MD5	SSL2_CK_RC4_128_EXPORT40_WITH_MD5

On y trouve également des constantes non définies dans le draft avec des commentaires très succincts :

- `SSL2_CK_NULL_WITH_MD5 /* v3 */`
- `SSL2_CK_DES_64_CBC_WITH_SHA /* v3 */`
- `SSL2_CK_DES_192_EDE3_CBC_WITH_SHA /* v3 */`
- `SSL2_CK_RC4_64_WITH_MD5 /* MS hack */`
- `SSL2_CK_DES_64_CFB64_WITH_MD5_1 /* SSLeay */`
- `SSL2_CK_NULL /* SSLeay */`

Les constantes commentées avec `v3` sont présentes pour des raisons de rétro-compatilibité depuis SSL v3. Celles commentées par `SSLeay` sont des vestiges de l'ancêtre d'OpenSSL : SSLeay. Elles sont sûrement conservées pour la rétro-compatibilité avec des vieux logiciels utilisant SSLeay. La `MS hack` est spécifique à Windows

6 SSL version 3

6.1 Spécifications

La version 3 du protocole SSL est décrite dans la RFC 6101. On y trouve notamment en section A.6 la liste des algorithmes de chiffrement pouvant être utilisés avec cette version :

Algorithmes supportés

Identifiant	KeyExch	Authn	Enc	MAC
SSL_NULL_WITH_NULL_NULL	NULL	NULL	NULL	NULL
SSL_RSA_WITH_NULL_MD5	RSA	RSA	NULL	MD5
SSL_RSA_WITH_NULL_SHA	RSA	RSA	NULL	SHA1
SSL_RSA_EXPORT_WITH_RC4_40_MD5	RSAex	RSAex	RC4.40	MD5
SSL_RSA_WITH_RC4_128_MD5	RSA	RSA	RC4.128	MD5
SSL_RSA_WITH_RC4_128_SHA	RSA	RSA	IDEA.128	SHA1
SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5	RSAex	RSAex	RC2.40 CBC	MD5
SSL_RSA_WITH_IDEA_CBC_SHA	RSA	RSA	IDEA.128 CBC	SHA1
SSL_RSA_EXPORT_WITH_DES40_CBC_SHA	RSAex	RSAex	DES.40	SHA1
SSL_RSA_WITH_DES_CBC_SHA	RSA	RSA	DES.56 CBC	SHA1
SSL_RSA_WITH_3DES_EDE_CBC_SHA	RSA	RSA	3DES.168 CBC	SHA1
SSL_DH_DSS_EXPORT_WITH_DES40_CBC_SHA	DH	DSS	DES.40 CBC	SHA1
SSL_DH_DSS_WITH_DES_CBC_SHA	DH	DSS	DES.56 CBC	SHA1
SSL_DH_DSS_WITH_3DES_EDE_CBC_SHA	DH	DSS	3DES.168 CBC	SHA1
SSL_DH_RSA_EXPORT_WITH_DES40_CBC_SHA	DH	RSA	DES.40 CBC	SHA1
SSL_DH_RSA_WITH_DES_CBC_SHA	DH	RSA	DES.56 CBC	SHA1
SSL_DH_RSA_WITH_3DES_EDE_CBC_SHA	DH	RSA	3DES.168 CBC	SHA1
SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA	DHE.512	DSS	DES.40 CBC	SHA1
SSL_DHE_DSS_WITH_DES_CBC_SHA	DHE	DSS	DES.56 CBC	SHA1
SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA	DHE	DSS	3DES.168 CBC	SHA1
SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA	DHE.512	RSA	DES.40CBC	SHA1
SSL_DHE_RSA_WITH_DES_CBC_SHA	DHE	RSA	DES.56 CBC	SHA1
SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA	DHE	RSA	3DES.168 CBC	SHA1
SSL_DH_anon_EXPORT_WITH_RC4_40_MD5	DH.512	None	RC4.40	MD5
SSL_DH_anon_WITH_RC4_128_MD5	DH	None	RC4.128	MD5
SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA	DH.512	None	DES.40 CBC	SHA1
SSL_DH_anon_WITH_DES_CBC_SHA	DH	None	DES.56 CBC	SHA1
SSL_DH_anon_WITH_3DES_EDE_CBC_SHA	DH	None	3DES.168 CBC	SHA1
SSL_FORTEZZA_KEA_WITH_NULL_SHA	FRTZA	KEA	None	SHA1
SSL_FORTEZZA_KEA_WITH_FORTEZZA_CBC_SHA	FRTZA	KEA	FRTZA	SHA1
SSL_FORTEZZA_KEA_WITH_RC4_128_SHA	FRTZA	KEA	RC4.128	SHA1

6.2 Implémentation

Dans le code d'OpenSSL, cette version du protocole SSL se trouve dans les fichiers commençant pas `s3_` du répertoire `ssl/`. Les constantes sont déclarées dans le fichier `ssl3.h`, on y retrouve les algorithmes de la RFC :

Identifiant	Constante OpenSSL
SSL_NULL_WITH_NULL_NULL	
SSL_RSA_WITH_NULL_MD5	SSL3_CK_RSA_NULL_MD5
SSL_RSA_WITH_NULL_SHA	SSL3_CK_RSA_NULL_SHA
SSL_RSA_EXPORT_WITH_RC4_40_MD5	SSL3_CK_RSA_RC4_40_MD5
SSL_RSA_WITH_RC4_128_MD5	SSL3_CK_RSA_RC4_128_MD5
SSL_RSA_WITH_RC4_128_SHA	SSL3_CK_RSA_RC4_128_SHA
SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5	SSL3_CK_RSA_RC2_40_MD5
SSL_RSA_WITH_IDEA_CBC_SHA	SSL3_CK_RSA_IDEA_128_SHA
SSL_RSA_EXPORT_WITH_DES40_CBC_SHA	SSL3_CK_RSA_DES_40_CBC_SHA
SSL_RSA_WITH_DES_CBC_SHA	SSL3_CK_RSA_DES_64_CBC_SHA
SSL_RSA_WITH_3DES_EDE_CBC_SHA	SSL3_CK_RSA_DES_192_CBC3_SHA
SSL_DH_DSS_EXPORT_WITH_DES40_CBC_SHA	SSL3_CK_DH_DSS_DES_40_CBC_SHA
SSL_DH_DSS_WITH_DES_CBC_SHA	SSL3_CK_DH_DSS_DES_64_CBC_SHA
SSL_DH_DSS_WITH_3DES_EDE_CBC_SHA	SSL3_CK_DH_DSS_DES_192_CBC3_SHA
SSL_DH_RSA_EXPORT_WITH_DES40_CBC_SHA	SSL3_CK_DH_RSA_DES_40_CBC_SHA
SSL_DH_RSA_WITH_DES_CBC_SHA	SSL3_CK_DH_RSA_DES_64_CBC_SHA
SSL_DH_RSA_WITH_3DES_EDE_CBC_SHA	SSL3_CK_DH_RSA_DES_192_CBC3_SHA
SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA	SSL3_CK_DHE_DSS_DES_40_CBC_SHA
SSL_DHE_DSS_WITH_DES_CBC_SHA	SSL3_CK_DHE_DSS_DES_64_CBC_SHA
SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA	SSL3_CK_DHE_DSS_DES_192_CBC3_SHA
SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA	SSL3_CK_DHE_RSA_DES_40_CBC_SHA
SSL_DHE_RSA_WITH_DES_CBC_SHA	SSL3_CK_DHE_RSA_DES_64_CBC_SHA
SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA	SSL3_CK_DHE_RSA_DES_192_CBC3_SHA
SSL_DH_anon_EXPORT_WITH_RC4_40_MD5	SSL3_CK_ADH_RC4_40_MD5
SSL_DH_anon_WITH_RC4_128_MD5	SSL3_CK_ADH_RC4_128_MD5
SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA	SSL3_CK_ADH_DES_40_CBC_SHA
SSL_DH_anon_WITH_DES_CBC_SHA	SSL3_CK_ADH_DES_64_CBC_SHA
SSL_DH_anon_WITH_3DES_EDE_CBC_SHA	SSL3_CK_ADH_DES_192_CBC_SHA
SSL_FORTEZZA_KEA_WITH_NULL_SHA	SSL3_CK_FZA_DMS_NULL_SHA
SSL_FORTEZZA_KEA_WITH_FORTEZZA_CBC_SHA	SSL3_CK_FZA_DMS_FZA_SHA
SSL_FORTEZZA_KEA_WITH_RC4_128_SHA	SSL3_CK_FZA_DMS_RC4_SHA

Attention Les 3 algorithmes FORTEZZA sont commentés dans OpenSSL depuis le commit 89bbe14c506b9bd2fd00e6bae22a99ef1ee7ad19 de 2006.

Remarque OpenSSL déclare d'autre constantes pour utiliser SSL 3 avec Kerberos 5 :

- SSL3_CK_KRB5_DES_64_CBC_SHA
- SSL3_CK_KRB5_DES_192_CBC3_SHA
- SSL3_CK_KRB5_RC4_128_SHA
- SSL3_CK_KRB5_IDEA_128_CBC_SHA
- SSL3_CK_KRB5_DES_64_CBC_MD5
- SSL3_CK_KRB5_DES_192_CBC3_MD5
- SSL3_CK_KRB5_RC4_128_MD5
- SSL3_CK_KRB5_IDEA_128_CBC_MD5

- SSL3_CK_KRB5_DES_40_CBC_SHA
- SSL3_CK_KRB5_RC2_40_CBC_SHA
- SSL3_CK_KRB5_RC4_40_SHA
- SSL3_CK_KRB5_DES_40_CBC_MD5
- SSL3_CK_KRB5_RC2_40_CBC_MD5
- SSL3_CK_KRB5_RC4_40_MD5

6.3 Failles

6.3.1 CVE-2013-4353

Cette faille découverte le 7 janvier 2014 par Anton Johansson permet de faire planter OpenSSL avec un déréférencement de pointeur NULL et peut ainsi causer des dénis de service. Cette faille a été corrigée le même jour par Stephen Henson. Voici le patch de correction dans la fonction `ssl3_take_mac` du fichier `ssl/s3_both.c` :

```
1  -------------------------------- ssl/s3_both.c --------------------------------
2  diff --git a/ssl/s3_both.c b/ssl/s3_both.c
3  index 8de149a..0a259b1 100644
4  --- a/ssl/s3_both.c
5  +++ b/ssl/s3_both.c
6  @@ -203,7 +203,11 @@
7           {
8           const char *sender;
9           int slen;
10 -
11 +         /* If no new cipher setup return immediately: other functions will
12 +          * set the appropriate error.
13 +          */
14 +         if (s->s3->tmp.new_cipher == NULL)
15 +                 return;
16          if (s->state & SSL_ST_CONNECT)
17                  {
18                  sender=s->method->ssl3_enc->server_finished_label;
```

6.4 Attaque sur le padding CBC de Serge Vaudenay

Cette attaque fait partie de la famille des attaques par canaux auxiliaires et plus particulèrement des timing attacks. En effet, lorsqu'openssl déchiffre en mode CBC, le temps varie en fonction de la longueur du message et du padding. Pour que l'attaque fonctionne, il faut utiliser un oracle de padding qui valide ou non le padding d'un chiffré.

Ben Laurie a appliqué un correctif du code OpenSSL le 28 janvier 2013 qui rend le décodage CBC constant, que le padding soit correct ou non. Le code a été placé dans un nouvelle fonction `tls1_cbc_remove_padding` du fichier `ssl/s3_cbc.c`

6.5 CVE-2011-4576

Cette faille permettait de récupérer des données d'un déchiffrement précédent. En effet, le buffer n'était pas réinitialisé. Cette faille a été identifée et corrigée par Adam Langley le 4 janvier 2011, voici le patch appliqué :

```
1  ---------------------------------- ssl/s3_enc.c ----------------------------------
2  diff --git a/ssl/s3_enc.c b/ssl/s3_enc.c
3  index 0ddfe19..c5df2cb 100644
4  --- a/ssl/s3_enc.c
5  +++ b/ssl/s3_enc.c
6  @@ -512,6 +512,9 @@
7
8              /* we need to add 'i-1' padding bytes */
9              l+=i;
10 +            /* the last of these zero bytes will be overwritten
11 +             * with the padding length. */
12 +            memset(&rec->input[rec->length], 0, i);
13             rec->length+=i;
14             rec->input[l-1]=(i-1);
15             }
```

7 TLS version 1

7.1 Spécifications

La version 1 de TLS est décrite dans la RFC 2246 (janvier 1999). Ce protocole est assez similaire à SSL 3 mais il y a quelques différences notables. Il y est notamment prévu un mécanisme de rétrocompatibilité vers SSL 3. La plus grosse différence est que TLS, contrairement à SSL, permet de commencer une connexion non chiffré sur un port usuel tel que le 80 et bascule en mode chiffré avec la commande STARTTLS.

Du point de vue des algorithmes de chiffrement, on a globalement la même liste que SSL 3 et les algorithmes de Fortezza ont été retirés :

Algorithmes supportés

Identifiant	KeyExch	Authn	Enc	MAC
TLS_NULL_WITH_NULL_NULL	NULL	NULL	NULL	NULL
TLS_RSA_WITH_NULL_MD5	RSA	RSA	NULL	MD5
TLS_RSA_WITH_NULL_SHA	RSA	RSA	NULL	SHA1
TLS_RSA_EXPORT_WITH_RC4_40_MD5	RSAex	RSAex	RC4.40	MD5
TLS_RSA_WITH_RC4_128_MD5	RSA	RSA	RC4.128	MD5
TLS_RSA_WITH_RC4_128_SHA	RSA	RSA	IDEA.128	SHA1
TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5	RSAex	RSAex	RC2.40 CBC	MD5
TLS_RSA_WITH_IDEA_CBC_SHA	RSA	RSA	IDEA.128 CBC	SHA1
TLS_RSA_EXPORT_WITH_DES40_CBC_SHA	RSAex	RSAex	DES.40	SHA1
TLS_RSA_WITH_DES_CBC_SHA	RSA	RSA	DES.56 CBC	SHA1
TLS_RSA_WITH_3DES_EDE_CBC_SHA	RSA	RSA	3DES.168 CBC	SHA1
TLS_DH_DSS_EXPORT_WITH_DES40_CBC_SHA	DH	DSS	DES.40 CBC	SHA1
TLS_DH_DSS_WITH_DES_CBC_SHA	DH	DSS	DES.56 CBC	SHA1
TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA	DH	DSS	3DES.168 CBC	SHA1
TLS_DH_RSA_EXPORT_WITH_DES40_CBC_SHA	DH	RSA	DES.40 CBC	SHA1
TLS_DH_RSA_WITH_DES_CBC_SHA	DH	RSA	DES.56 CBC	SHA1
TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA	DH	RSA	3DES.168 CBC	SHA1
TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA	DHE.512	DSS	DES.40 CBC	SHA1
TLS_DHE_DSS_WITH_DES_CBC_SHA	DHE	DSS	DES.56 CBC	SHA1
TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA	DHE	DSS	3DES.168 CBC	SHA1
TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA	DHE.512	RSA	DES.40CBC	SHA1
TLS_DHE_RSA_WITH_DES_CBC_SHA	DHE	RSA	DES.56 CBC	SHA1
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA	DHE	RSA	3DES.168 CBC	SHA1
TLS_DH_anon_EXPORT_WITH_RC4_40_MD5	DH.512	None	RC4.40	MD5
TLS_DH_anon_WITH_RC4_128_MD5	DH	None	RC4.128	MD5
TLS_DH_anon_EXPORT_WITH_DES40_CBC_SHA	DH.512	None	DES.40 CBC	SHA1
TLS_DH_anon_WITH_DES_CBC_SHA	DH	None	DES.56 CBC	SHA1
TLS_DH_anon_WITH_3DES_EDE_CBC_SHA	DH	None	3DES.168 CBC	SHA1

Algorithmes supplémentaires de la RFC 3268 La RFC prévoit la possibilité d'étendre cette liste. Ainsi, la RFC 3268 apporte des nouvelles constantes avec AES et SHA-1 :

Identifiant	KeyExch	Authn	Enc	MAC
TLS_RSA_WITH_AES_128_CBC_SHA	RSA	RSA	AES 128 CBC	SHA1
TLS_DH_DSS_WITH_AES_128_CBC_SHA	DH	DSS	AES 128 CBC	SHA1
TLS_DH_RSA_WITH_AES_128_CBC_SHA	DH	RSA	AES 128 CBC	SHA1
TLS_DHE_DSS_WITH_AES_128_CBC_SHA	DHE	DSS	AES 128 CBC	SHA1
TLS_DHE_RSA_WITH_AES_128_CBC_SHA	DHE	RSA	AES 128 CBC	SHA1
TLS_DH_anon_WITH_AES_128_CBC_SHA	DH	NULL	AES 128 CBC	SHA1
TLS_RSA_WITH_AES_256_CBC_SHA	RSA	RSA	AES 256 CBC	SHA1
TLS_DH_DSS_WITH_AES_256_CBC_SHA	DH	DSS	AES 256 CBC	SHA1
TLS_DH_RSA_WITH_AES_256_CBC_SHA	DH	RSA	AES 256 CBC	SHA1
TLS_DHE_DSS_WITH_AES_256_CBC_SHA	DHE	DSS	AES 256 CBC	SHA1
TLS_DHE_RSA_WITH_AES_256_CBC_SHA	DHE	RSA	AES 256 CBC	SHA1
TLS_DH_anon_WITH_AES_256_CBC_SHA	DH	NULL	AES 256 CBC	SHA1

Algorithmes supplémentaires de la RFC 4132 La RFC 4132 apporte également une liste supplémentaire utilisant l'algorithme de chiffrement Camellia :

Identifiant	KeyExch	Authn	Enc	MAC
TLS_RSA_WITH_CAMELLIA_128_CBC_SHA	RSA	RSA	Camellia 128 CBC	SHA1
TLS_DH_DSS_WITH_CAMELLIA_128_CBC_SHA	DH	DSS	Camellia 128 CBC	SHA1
TLS_DH_RSA_WITH_CAMELLIA_128_CBC_SHA	DH	RSA	Camellia 128 CBC	SHA1
TLS_DHE_DSS_WITH_CAMELLIA_128_CBC_SHA	DHE	DSS	Camellia 128 CBC	SHA1
TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA	DHE	RSA	Camellia 128 CBC	SHA1
TLS_DH_anon_WITH_CAMELLIA_128_CBC_SHA	DH	NULL	Camellia 128 CBC	SHA1
TLS_RSA_WITH_CAMELLIA_256_CBC_SHA	RSA	RSA	Camellia 256 CBC	SHA1
TLS_DH_DSS_WITH_CAMELLIA_256_CBC_SHA	DH	DSS	Camellia 256 CBC	SHA1
TLS_DH_RSA_WITH_CAMELLIA_256_CBC_SHA	DH	RSA	Camellia 256 CBC	SHA1
TLS_DHE_DSS_WITH_CAMELLIA_256_CBC_SHA	DHE	DSS	Camellia 256 CBC	SHA1
TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA	DHE	RSA	Camellia 256 CBC	SHA1
TLS_DH_anon_WITH_CAMELLIA_256_CBC_SHA	DH	NULL	Camellia 256 CBC	SHA1

Algorithmes supplémentaires de la RFC 4162 La RFC 4162 ajoute l'algorithme de chiffrement SEED :

Identifiant	KeyExch	Authn	Enc	MAC
TLS_RSA_WITH_SEED_CBC_SHA	RSA	RSA	SEED CBC	SHA1
TLS_DH_DSS_WITH_SEED_CBC_SHA	DH	DSS	SEED CBC	SHA1
TLS_DH_RSA_WITH_SEED_CBC_SHA	DH	RSA	SEED CBC	SHA1
TLS_DHE_DSS_WITH_SEED_CBC_SHA	DHE	DSS	SEED CBC	SHA1
TLS_DHE_RSA_WITH_SEED_CBC_SHA	DHE	RSA	SEED CBC	SHA1
TLS_DH_anon_WITH_SEED_CBC_SHA	DH	NULL	SEED CBC	SHA1

Extensions TLS Des extensions TLS du hello client sont décrites dans les RFC 3546, 4366, 4492 et 4507 :

— **Server Name Indication** : permet d'indiquer au serveur quel est le nom du serveur qu'il demande, cela est utilisé lorsqu'il y a plusieurs hôtes virtuels sur une même machine ;

— **Maximum Fragment Length Negotiation** : sans cette extension, TLS spécifie une taille maximale fixe de fragment à 2^{14} octets. Cette extension permet aux clients d'adapter cette taille en fonction des limites de mémoire ou de bande passante. Valeurs autorisées : $2^9, 2^{10}, 2^{11}, 2^{12}$. Si la valeur n'est pas dans cette liste, le serveur doit interrompre la poignée de main.
— **Client Certificate URLs** : sans cette extension, TLS spécifie que lors l'authentification client, ce dernier doit envoyer son certificat pendant la poignée de main. Grâce à cette extension, le client peut économiser de l'espace disque en stoquant son certificat à une autre adresse
— **Trusted CA Indication** : indique les clefs de CA racines possède le client
— **Truncated HMAC** : permet de tronquer le code MAC à 10 octets pour économiser dela bande passante
— **Certificate Status Request** : indique que le client souhaite vérifier la validité du certificat du serveur avec une requête OCSP par exemple
— **Supported Elliptic Curves** : indique les courbes elliptiques supportées par le client
— **Session Ticket** : permet de rétablir une session précédente

7.2 Implémentation

Pour les constantes représentant les algorithmes disponibles, tout est dans `ssl/tls1.h`. On y trouve également les algorithmes utilisant les courbes elliptiques décrits dans le draft `http://tools.ietf.org/html/draft-ietf-tls-ecc-12` :

```
#define TLS1_CK_ECDH_ECDSA_WITH_NULL_SHA                0x0300C001
#define TLS1_CK_ECDH_ECDSA_WITH_RC4_128_SHA             0x0300C002
#define TLS1_CK_ECDH_ECDSA_WITH_DES_192_CBC3_SHA        0x0300C003
#define TLS1_CK_ECDH_ECDSA_WITH_AES_128_CBC_SHA         0x0300C004
#define TLS1_CK_ECDH_ECDSA_WITH_AES_256_CBC_SHA         0x0300C005

#define TLS1_CK_ECDHE_ECDSA_WITH_NULL_SHA               0x0300C006
#define TLS1_CK_ECDHE_ECDSA_WITH_RC4_128_SHA            0x0300C007
#define TLS1_CK_ECDHE_ECDSA_WITH_DES_192_CBC3_SHA       0x0300C008
#define TLS1_CK_ECDHE_ECDSA_WITH_AES_128_CBC_SHA        0x0300C009
#define TLS1_CK_ECDHE_ECDSA_WITH_AES_256_CBC_SHA        0x0300C00A

#define TLS1_CK_ECDH_RSA_WITH_NULL_SHA                  0x0300C00B
#define TLS1_CK_ECDH_RSA_WITH_RC4_128_SHA               0x0300C00C
#define TLS1_CK_ECDH_RSA_WITH_DES_192_CBC3_SHA          0x0300C00D
#define TLS1_CK_ECDH_RSA_WITH_AES_128_CBC_SHA           0x0300C00E
#define TLS1_CK_ECDH_RSA_WITH_AES_256_CBC_SHA           0x0300C00F

#define TLS1_CK_ECDHE_RSA_WITH_NULL_SHA                 0x0300C010
#define TLS1_CK_ECDHE_RSA_WITH_RC4_128_SHA              0x0300C011
#define TLS1_CK_ECDHE_RSA_WITH_DES_192_CBC3_SHA         0x0300C012
#define TLS1_CK_ECDHE_RSA_WITH_AES_128_CBC_SHA          0x0300C013
#define TLS1_CK_ECDHE_RSA_WITH_AES_256_CBC_SHA          0x0300C014

#define TLS1_CK_ECDH_anon_WITH_NULL_SHA                 0x0300C015
#define TLS1_CK_ECDH_anon_WITH_RC4_128_SHA              0x0300C016
```

```
#define TLS1_CK_ECDH_anon_WITH_DES_192_CBC3_SHA          0x0300C017
#define TLS1_CK_ECDH_anon_WITH_AES_128_CBC_SHA           0x0300C018
#define TLS1_CK_ECDH_anon_WITH_AES_256_CBC_SHA           0x0300C019

#define TLS1_CK_RSA_WITH_AES_128_SHA                     0x0300002F
#define TLS1_CK_DH_DSS_WITH_AES_128_SHA                  0x03000030
#define TLS1_CK_DH_RSA_WITH_AES_128_SHA                  0x03000031
#define TLS1_CK_DHE_DSS_WITH_AES_128_SHA                 0x03000032
#define TLS1_CK_DHE_RSA_WITH_AES_128_SHA                 0x03000033
#define TLS1_CK_ADH_WITH_AES_128_SHA                     0x03000034

#define TLS1_CK_RSA_WITH_AES_256_SHA                     0x03000035
#define TLS1_CK_DH_DSS_WITH_AES_256_SHA                  0x03000036
#define TLS1_CK_DH_RSA_WITH_AES_256_SHA                  0x03000037
#define TLS1_CK_DHE_DSS_WITH_AES_256_SHA                 0x03000038
#define TLS1_CK_DHE_RSA_WITH_AES_256_SHA                 0x03000039
#define TLS1_CK_ADH_WITH_AES_256_SHA                     0x0300003A

#define TLS1_CK_RSA_WITH_CAMELLIA_128_CBC_SHA            0x03000041
#define TLS1_CK_DH_DSS_WITH_CAMELLIA_128_CBC_SHA         0x03000042
#define TLS1_CK_DH_RSA_WITH_CAMELLIA_128_CBC_SHA         0x03000043
#define TLS1_CK_DHE_DSS_WITH_CAMELLIA_128_CBC_SHA        0x03000044
#define TLS1_CK_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA        0x03000045
#define TLS1_CK_ADH_WITH_CAMELLIA_128_CBC_SHA            0x03000046
```

Pour ce qui est de la poignée de main, OpenSSL utilise la même fonction que pour SSL 3 : `ssl3_connect (s3_clnt.c)/ssl3_accept (s3_srvr.c)` et la gestion des extensions se trouve dans la fonction `ssl_scan_clienthello_tlsext (t1_lib.c)`.

Extension	**Implémenté**
Server Name Indication	oui (`t1_lib.c`)
Maximum Fragment Length Negotiation	non trouvée
Client Certificate URLs	non trouvée
Trusted CA Indication	non trouvée
Truncated HMAC	non trouvée
Certificate Status Request	oui (`t1_lib.c`)
Supported Elliptic Curves	oui (`t1_lib.c`)
Session Ticket	oui (`t1_lib.c`)

7.3 Failles

7.4 Attaque sur le padding CBC de Serge Vaudenay

Voir 6.4

8 TLS version 1.1

8.1 Spécifications

La version 1.1 de TLS est spécifiée par la RFC 4346. Différences avec la version 1.0 :

— l'IV implicite est remplacé par une IV explicite (protection contre les attaques sur CBC : `http://www.openssl.org/~bodo/tls-cbc.txt`) ;

— utilisation de l'alerte `bad_record_mac` plutôt que `decryption_failed` lors des erreurs de padding ;

— les sessions fermées prématurément peuvent être reprises.

Algorithmes supplémentaires de la RFC 5054 La RFC 5054 ajoute l'échange de clef/authentification SRP :

Identifiant	KeyExch	Authn	Enc	MAC
TLS_SRP_SHA_WITH_3DES_EDE_CBC_SHA	SRP SHA1	SRP SHA1	3DES CBC	SHA1
TLS_SRP_SHA_RSA_WITH_3DES_EDE_CBC_SHA	SRP SHA1	RSA	3DES CBC	SHA1
TLS_SRP_SHA_DSS_WITH_3DES_EDE_CBC_SHA	SRP SHA1	DSS	3DES CBC	SHA1
TLS_SRP_SHA_WITH_AES_128_CBC_SHA	SRP SHA1	SRP SHA1	AES 128 CBC	SHA1
TLS_SRP_SHA_RSA_WITH_AES_128_CBC_SHA	SRP SHA1	RSA	AES 128 CBC	SHA1
TLS_SRP_SHA_DSS_WITH_AES_128_CBC_SHA	SRP SHA1	DSS	AES 128 CBC	SHA1
TLS_SRP_SHA_WITH_AES_256_CBC_SHA	SRP SHA1	SRP SHA1	AES 256 CBC	SHA1
TLS_SRP_SHA_RSA_WITH_AES_256_CBC_SHA	SRP SHA1	RSA	AES 256 CBC	SHA1
TLS_SRP_SHA_DSS_WITH_AES_256_CBC_SHA	SRP SHA1	DSS	AES 256 CBC	SHA1

Extensions TLS Une extension TLS du hello client est décrite dans la RFC 5054 :

— **srp** : permet d'indiquer le support d'algorithmes d'échange de clefs/authentification SRP ;

8.2 Implémentation

On retrouve les constantes dans `ssl/tls1.h` :

```
/* SRP ciphersuites from RFC 5054 */
#define TLS1_CK_SRP_SHA_WITH_3DES_EDE_CBC_SHA       0x0300C01A
#define TLS1_CK_SRP_SHA_RSA_WITH_3DES_EDE_CBC_SHA   0x0300C01B
#define TLS1_CK_SRP_SHA_DSS_WITH_3DES_EDE_CBC_SHA   0x0300C01C
#define TLS1_CK_SRP_SHA_WITH_AES_128_CBC_SHA        0x0300C01D
#define TLS1_CK_SRP_SHA_RSA_WITH_AES_128_CBC_SHA    0x0300C01E
#define TLS1_CK_SRP_SHA_DSS_WITH_AES_128_CBC_SHA    0x0300C01F
#define TLS1_CK_SRP_SHA_WITH_AES_256_CBC_SHA        0x0300C020
#define TLS1_CK_SRP_SHA_RSA_WITH_AES_256_CBC_SHA    0x0300C021
#define TLS1_CK_SRP_SHA_DSS_WITH_AES_256_CBC_SHA    0x0300C022
```

8.3 Failles

8.3.1 CVE-2012-2333

Un integer underflow permettait de causer un déni de service dans les protocoles TLS 1.1, 1.2 et DTLS. Cette faille a été publiée le 10 mai 2012 par Codenomicon et affecte les versions d'OpenSSL

suivantes :

— < 0.9.8x
— 1.0.0x < 1.0.0j
— 1.0.1x < 1.0.0c

Cette faille a été trouvée grâce à l'outil Fuzz-o-Matic développé par Codenomicon.

La correction suivante a été appliquée le même jour par l'équipe d'OpenSSL :

```
1  ---------------------------------- ssl/t1_enc.c ----------------------------------
2  diff --git a/ssl/t1_enc.c b/ssl/t1_enc.c
3  index 201ca9a..f7bdeb3 100644
4  --- a/ssl/t1_enc.c
5  +++ b/ssl/t1_enc.c
6  @@ -889,6 +889,8 @@
7              if (s->version >= TLS1_1_VERSION
8                  && EVP_CIPHER_CTX_mode(ds) == EVP_CIPH_CBC_MODE)
9                  {
10 +                if (bs > (int)rec->length)
11 +                    return -1;
12                 rec->data += bs;    /* skip the explicit IV */
13                 rec->input += bs;
14                 rec->length -= bs;
```

Ici, la variable `bs` est la taille de bloc utilisée dans l'algorithme de chiffrement. On constate que, sans vérification, il était possible de déplacer les pointeurs `rec->data`, `rec->input` et `rec->length` au-delà de la longueur du paquet et ainsi créer une erreur de segmentation.

8.3.2 Lucky Thirteen CVE-2013-0169

Cette attaque a été trouvée par Nadhem Alfardan et Kenny Paterson de l'Université de Londres le 5 février 2005 et a été corrigée par Adam Langley et Emilia Kasper. Versions touchées :

— ≤ 1.0.1c
— ≤ 1.0.0j
— 0.9.8x

Cette attaque reprend le principe de l'attaque de Vaudenay (6.4).

Lorsqu'un message chiffré incorrect est reçu, un message d'erreur fatale est renvoyé vers l'expéditeur. Cependant, le temps de génération de ce message d'erreur dépend du nombre d'octets valides, utilisé par un haché MAC.

Un attaquant peut donc injecter des messages chiffrés erronés dans une session TLS/DTLS en mode CBC, et mesurer le temps nécessaire à la génération du message d'erreur, afin de progressivement déterminer le contenu en clair de la session.

Il faut 2^{23} sessions TLS pour retrouver un bloc en clair. Pour mener l'attaque, le client TLS doit alors continuer en permanence à ouvrir une nouvelle session, dès que la précédente s'est terminée en erreur fatale.

9 TLS version 1.2

9.1 Spécifications

La version 1.2 de TLS est décrite dans la RFC 5246 (2008)

Algorithmes supplémentaires de la RFC 5289 La RFC 5289 ajoute les codes MAC SHA256 et SHA384 ainsi que le mode GCM (Galois Counter Mode) :

Identifiant	KeyExch	Authn	Enc	MAC
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	ECDHE	ECDSA	AES 128 CBC	SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	ECDHE	ECDSA	AES 256 CBC	SHA384
TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256	ECDH	ECDSA	AES 128 CBC	SHA256
TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384	ECDH	ECDSA	AES 256 CBC	SHA256
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	ECDHE	RSA	AES 128 CBC	SHA256
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	ECDHE	RSA	AES 256 CBC	SHA384
TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256	ECDH	RSA	AES 128 CBC	SHA256
TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384	ECDH	RSA	AES 256 CBC	SHA384
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	ECDHE	ECDSA	AES 128 GCM	SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	ECDHE	ECDSA	AES 256 GCM	SHA384
TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256	ECDH	ECDSA	AES 128 GCM	SHA256
TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384	ECDH	ECDSA	AES 256 GCM	SHA384
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	ECDHE	RSA	AES 128 GCM	SHA256
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	ECDHE	RSA	AES 256 GCM	SHA384
TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256	ECDH	RSA	AES 128 GCM	SHA256
TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384	ECDH	RSA	AES 256 GCM	SHA384

Extensions RFC 5878 et 5746

9.2 Implémentation

On retrouve les constantes dans `ssl/tls1.h` :

```
/* ECDH HMAC based ciphersuites from RFC5289 */
#define TLS1_CK_ECDHE_ECDSA_WITH_AES_128_SHA256         0x0300C023
#define TLS1_CK_ECDHE_ECDSA_WITH_AES_256_SHA384         0x0300C024
#define TLS1_CK_ECDH_ECDSA_WITH_AES_128_SHA256          0x0300C025
#define TLS1_CK_ECDH_ECDSA_WITH_AES_256_SHA384          0x0300C026
#define TLS1_CK_ECDHE_RSA_WITH_AES_128_SHA256           0x0300C027
#define TLS1_CK_ECDHE_RSA_WITH_AES_256_SHA384           0x0300C028
#define TLS1_CK_ECDH_RSA_WITH_AES_128_SHA256            0x0300C029
#define TLS1_CK_ECDH_RSA_WITH_AES_256_SHA384            0x0300C02A

/* ECDH GCM based ciphersuites from RFC5289 */
#define TLS1_CK_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256     0x0300C02B
#define TLS1_CK_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384     0x0300C02C
#define TLS1_CK_ECDH_ECDSA_WITH_AES_128_GCM_SHA256      0x0300C02D
#define TLS1_CK_ECDH_ECDSA_WITH_AES_256_GCM_SHA384      0x0300C02E
```

```
#define TLS1_CK_ECDHE_RSA_WITH_AES_128_GCM_SHA256        0x0300C02F
#define TLS1_CK_ECDHE_RSA_WITH_AES_256_GCM_SHA384        0x0300C030
#define TLS1_CK_ECDH_RSA_WITH_AES_128_GCM_SHA256         0x0300C031
#define TLS1_CK_ECDH_RSA_WITH_AES_256_GCM_SHA384         0x0300C032
```

9.3 Failles

9.3.1 CVE-2012-2333

Voir 8.3.1

9.3.2 CVE-2013-6449

Cette faille a été découverte en novembre 2013 et a été corrigée le 19 décembre 2013 par l'équipe d'OpenSSL. Elle affecte les versions < 1.0.2. Elle consiste à créer un déni de service du serveur en envoyant une structure de données mal formée. Le crash se situe dans la fonction `ssl_get_algorithm2` du fichier `ssl/s3_lib.c` Le patch correctif :

```
index bf832bb..c4ef273 100644 (file)
--- a/ssl/s3_lib.c
+++ b/ssl/s3_lib.c
@@ -4286,7 +4286,7 @@ need to go to SSL_ST_ACCEPT.
 long ssl_get_algorithm2(SSL *s)
        {
        long alg2 = s->s3->tmp.new_cipher->algorithm2;
-       if (TLS1_get_version(s) >= TLS1_2_VERSION &&
+       if (s->method->version == TLS1_2_VERSION &&
            alg2 == (SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF))
                return SSL_HANDSHAKE_MAC_SHA256 | TLS1_PRF_SHA256;
        return alg2;
```