ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ

Факултет по телекомуникации

Специалност: Телекомуникации

Курсова работа по Архитектури и протоколи за мрежови комуникации с повишено ниво на сигурност

Тема:

Протоколи SSL и TLS

Студент: инж. Николай Проданов Фак. №: 111321045 Група: 232

Преподавател: доц. д-р инж. Мария Ненова

Дата: .....

Подпис: .....

София, 2022

Съдържание

Увод

В съвремието интернетът и мрежовата свързаност са неизменима част от живота ни. Ние използваме телефони и компютри за плащане на сметки, правене на покупки, работа, социални мрежи и много други неща. Според изчисления на учени на планетата има повече телефони от колкото живи хора, ако към това число включим и навлизащите IoT технологии, в които комуникацията ще играе централна роля. В това число не би могло да не се включи и TLS протоколът като начин за защита на данните при протичане на комуникацията.

Глава 1: Същност на протокола

Transport Layer Security (TLS) протоколът е широко използван протокол за сигурност чиято цел да е осигури поверителност, интегритет и автентикация на връзката през IP базиран мрежа като интернет. Основната му функция е осигуряването на защитен канал за данни където информацията изпратена по него се криптира. Това е силно застъпено в съвременният интернет и е основна част от HTTPS протокола. TLS е предназначен да се използва над TCP като предоставя сигурност върху TCP сесията, това му свойство позволява той лесно да се интегрира в работата на всеки протокол или приложение ползващо TCP като протокол на транспортния слой.

В началото на създаването на интернета, сигурността не била сред приоритетите в резултат на което протоколите използвани в него не са наблягани на сигурността. Тази функционалност е била оставена на потребителите, които от своя страна можели свободно да дефинират как приложенията им комуникират през мрежата. В резултат на което повечето участници в интернет били незащитени. Това не било такъв проблем, понеже тогава интернет се състоял от малък брой участници, основно университети. В наши дни това би направило интернет напълно не приложим.

SSL (secure socket layer) и TLS били създадени за да предоставят сигурност в комуникацията в незащитена инфраструктура. Това означава че SSL и TLS могат да се допълнят към съществуващият протоколен стек на услуги и приложения и да им предоставят сигурност при комуникацията през

незащитената мрежа напълно прозрачно. TLS е предвиден да надгради транспортният слой (основно TCP). От там идва и наименованието на протокола.

TLS протоколът има четири основни цели:

- Сигурност – основната цел. Предоставяна на защитен канал между две крайни възела които искат да комуникират по между си.
- Оперативна съвместимост – позволява на независими разработчици (3rd party developers) да разработват програми и библиотеки които биха могли да комуникират помежду си използвайки общо приети криптографски алгоритми.
- Възможност за разширение – позволява TLS протокола да служи за среда за разработване и използване (deployment) на криптографски алгоритми. Това на практика превръща TLS в платформа, която е независима от криптографските примитиви които ще се използват за криптиране сесия, изчисляване на хешове за интегритет и други функционалности. Това позволява лесната миграция между примитиви без да се налага промяна в самият протокол или създаване на нов.
- Работоспособност – възможност да се осигури добро бързодействие след като всички гореспоменати цели са изпълнение. Това означава, намаляване на сложни криптографски изчисления изискващи изчислителни ресурси и време и добро кеширане на вече изпълнени операции с цел избягване те да се изчисляват отново (ако не е необходимо).

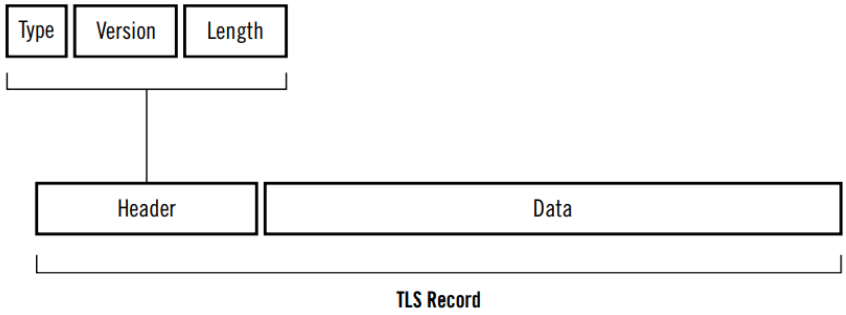
Глава 2: Начин на работа на протокола

В тази глава ще се разгледа начинът на работа TLS 1.2 който вече е определен като ненадежден и е надграден TLS 1.3. Между двете версии има големи разлики във връзка със сигурността и бързодействието на протокола. Но основният начин на работа и структура на съобщенията не е значително променена.

2.1 Протокол запис

Протоколът запис може да се представи като протокол с по-високо ниво на абстракция. Има за задача да предава, и опционално да криптира, всички съобщения от по-ниско ниво през виртуалният канал за връзка (TCP channel).

В структурата си всяко запис съобщение се състои от етикет и тяло. В тялото се пренасят съобщенията на под-протоколите. Това се вижда на Фиг. 1.



Фиг. 1 – структура на Record съобщението

По формално протоколът може да бъде представен със следният код на C. Показан на Фиг. 2.

```
struct {
    uint8 major;
    uint8 minor;
} ProtocolVersion;

enum {
    change_cipher_spec(20),
    alert(21),
    handshake(22),
    application_data(23)
} ContentType;

struct {
    ContentType type;
    ProtocolVersion version;
    uint16 length; /* Maximum length is 2^14 (16,384) bytes. */
    opaque fragment[TLSPlaintext.length];
} TLSPlaintext;
```

Фиг. 2 – формат на Record съобщението

В допълнение към полетата от Фиг. 2, към всеки TLS запис се назначава 64 битов уникален пореден номер, който не се изпраща по канала. Всяка стана във комуникацията си назначава собствена уникална поредица и следи броят на приетите записи. Тези поредици се използват като част от защита към атаки с отговор.

Идеята на запис протоколът е да абстрахира под-протоколите от процедурите и особеностите на обмяната на съобщения през канала. Това дефинира 4 основни функции които се изпълняват от запис протокола:

- Прехвърляне на съобщенията – запис протоколът прехвърля буфери от информация подадени му от под-протоколите. Ако буфер стане по-голям от разрешеният лимит (16 384 байта) то той се фрагментира.

Обратният случай също е възможен, където запис протоколът комбинира няколко буфера от един протокол в един запис (съобщение).

- Криптиране на пакета и проверка за интегритет – В началото на сесията съобщенията се предават без да се криптират. Това е понеже двете стани още на са се договорели за параметрите на сесията. След ръкостискането, с което се договарят параметрите на връзката, запис протоколът за почва да криптира съобщенията и да ги проверява спрямо договорените параметри.
- Компресия – едно време замислена като допълнение към протокола. Но компресията се прави от имплементациите на HTTP сървърите още от преди TLS и се използва при не защитени връзки. Поради това тази функция не вижда практически приложение.
- Възможност за разширяване – запис протоколът поема процедурите по пренасяне и криптиране на съобщенията но делегира всички останали функционалности към под-протоколи. Този подход дава възможност на TLS да се разширява, тъй като нови под-протоколи могат лесно да бъдат добавени

Основната TLS спецификация дефинира четири под-протокола: протокол за ръкостискане (handshake protocol), протокол за смяна на шифърни спецификации (cipher spec protocol), протокол на приложението (application data protocol) и протокол при тревога.

2.2 Протокол за ръкостискане

Това е най-сложната част от TLS протокола. По време на ръкостискането двете страни договарят параметрите с които ще се извършва комуникацията и се автентикират. През тази фаза се обменят между 6 до 10 в значимост от използваните функционалности. В практиката се срещат три вида ръкостискане:

- Пълно ръкостискане със сървърна автентикация
- Ръкостискане възстановяващо предишна сесия
- Ръкостискане със сървърна и клиентска автентикация

Етикета на това съобщение може да се види на Фиг. 3.

```
struct {
    HandshakeType msg_type;
    uint24 length;
    HandshakeMessage message;
} Handshake;
```

Фиг. 3 – общ формат на съобщенията при ръкостискане

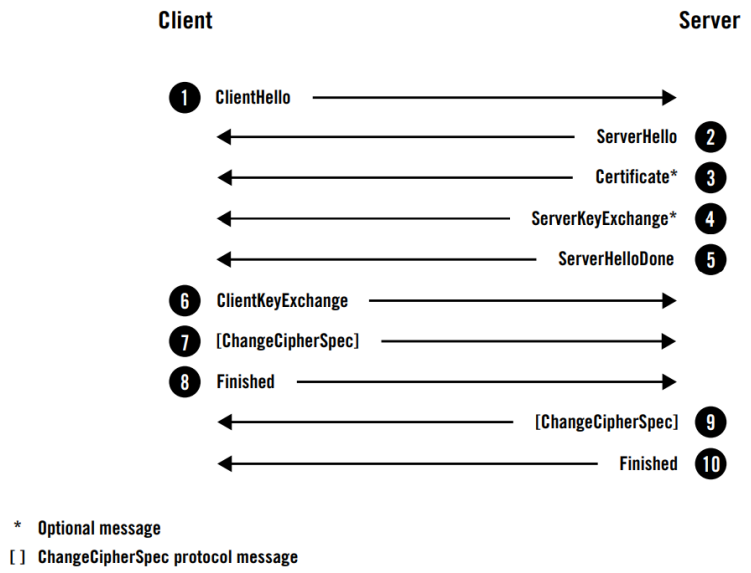
Тук се слагат типът на съобщението, неговата големина, и буфер на тялото.

2.2.1 Пълно ръкостискане със сървърна автентикация

Всяка сесия започва със ръкостискане. Ако до преди не е била създавана сесия, се използва пълно ръкостискане с да се договорят параметрите на сесията.

По време на ръкостискането сървърът и клиентът ще изпълнят четири основни задачи:

1) Размяна на възможностите и определяне на параметрите на връзката
2) Валидация на представеният сертификат или друг метод на автентикация
3) Съгласие да се ползва тайна (secret) за защита на сесията
4) Проверка че съобщенията за ръкостискане не са били променени от 3та страна



Фиг. 4 – пълно ръкостискане с автентикация на сървъра

Схема на обмяната на съобщения при това ръкостискане може да се види на Фиг. 4.

1. Клиентът започва ръкостискането, изпращайки възможностите си към сървъра.
2. Сървъра изпраща избраните параметри за сесията
3. Сървъра изпраща веригата си от сертификати

4. В зависимост от избраният начин за обмен на ключовете сървърът изпраща допълнела за генериране на главната тайна (master secret)
5. Сървърът уведомява че е приключил преговорите от негова стана
6. Клиентът изпраща допълнителна информация нужна за генериране на главната тайна (master secret)
7. Клиента преминава към криптирана комуникация и уведомява сървъра
8. Клиентът изпраща MAC (message authentication code – подобно на чек сума) от съобщенията от ръкостискането които е приел и изпратил
9. Сървърът преминава към криптирана комуникация и уведомява клиента
10. Сървърът изпраща MAC (message authentication code – подобно на чек сума) от съобщенията от ръкостискането които е приел и изпратил

След този процес се предполага че сесията е изградена и обменът на данни от двете стани може да започне.

Нека да разгледаме по внимателно формата на съобщенията:

2.2.1.1 Клиент Здравей (ClientHello) съобщение

Със това съобщение се инициира процедурата по ръкостискането. То съдържа в себе си възможностите и предпочитанията на клиента, като подържани криптографски суитове. На Фиг. 5 е показано такова съобщение.

```
Handshake protocol: ClientHello
    Version: TLS 1.2
    Random
        Client time: May 22, 2030 02:43:46 GMT
        Random bytes: b76b0e61829557eb4c611adfd2d36eb232dc1332fe29802e321ee871
    Session ID: (empty)
    Cipher Suites
        Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
        Suite: TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
        Suite: TLS_RSA_WITH_AES_128_GCM_SHA256
        Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
        Suite: TLS_DHE_RSA_WITH_AES_128_CBC_SHA
        Suite: TLS_RSA_WITH_AES_128_CBC_SHA
        Suite: TLS_RSA_WITH_3DES_EDE_CBC_SHA
        Suite: TLS_RSA_WITH_RC4_128_SHA
    Compression methods
        Method: null
    Extensions
        Extension: server_name
            Hostname: www.feistyduck.com
        Extension: renegotiation_info
        Extension: elliptic_curves
            Named curve: secp256r1
            Named curve: secp384r1
        Extension: signature_algorithms
            Algorithm: sha1/rsa
            Algorithm: sha256/rsa
            Algorithm: sha1/ecdsa
            Algorithm: sha256/ecdsa
```

Фиг. 5 – примерно съобщение ClientHello

Съобщението има проста структура. Ще разгледаме някой от основните полета в нея:

- Protocol version – показва коя е най-високата версия на TLS протокола която клиентът поддържа.
- Random – Клиентът и сървърът предоставят произволно генерирани байтове по време на преговорите. Това прави всяко ръкостискане уникално, само по себе си и играе важна роля срещу защитата от атаки използващи отговор.
- Session ID – използва се при възстановяване на предишна сесия.
- Cipher suites – Списък от всички криптографски суитове които клиентът поддържа.

- Compression – съдържа един или повече методи за компресия които клиентът поддържа. Практически не се използва.
- Extensions – Използва се да пренася информация необходима за разширенията на TLS протокола

2.2.1.2 Сървър Здравей (ServerHello) съобщение

```
Handshake protocol: ServerHello
        Version: TLS 1.2
        Random
                Server time: Mar 10, 2059 02:35:57 GMT
                Random bytes: 8469b09b480c1978182ce1b59290487609f41132312ca22aacaf5012
        Session ID: 4cae75c91cf5adf55f93c9fb5dd36d19903b1182029af3d527b7a42ef1c32c80
        Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
        Compression method: null
        Extensions
                Extension: server_name
                Extension: renegotiation_info
```

Фиг. 6 – примерно съобщение ServerHello

Сървърното съобщение е подобно на това на клиента с разликата че при всяко поле има само една опция. Тази която сървъра е избрал.

Сървърът може да не поддържа същата TLS версия като клиента. При което той изпраща най-високата версия която поддържа с надеждата че клиента ще го приеме.

2.2.1.3 Certificate - Съобщение за поддаване на сертификати

Това съобщение се използва за да пренася X.509 веригата от сертификати на сървъра. Първо се изпраща основният сертификат и след това всички междинни. Root сертификата не бива да се изпраща.

Това съобщение е опционално, понеже не всички суитове използват автентикация чрез сертификати. В допълнение в това съобщение могат да се доставят не само X.509 сертификати. Някой суитове го използват за обмяна на PGP ключове.

2.2.1.4 ServerKeyExchange – съобщение за обмяна на ключове

Идеята на това съобщение е да пренася допълнителна информация необходима за обмяната на ключове. Съдържанието му се променя спрямо договорения суит. Някои суитове не го използват, поради което то е опционално и може да се пропусне.

2.2.1.5 ServerHelloDone – съобщение че сървъра е свършил с преговорите

Сървъра уведомява клиента че е изпратил цялата необходима информация за договаряне и чака следващи съобщения от клиента.

2.2.1.6 ClientKeyExchange – съобщение за обмяна на ключове

С това съобщение се носи информацията от клиента необходима за генериране на ключа. Това е задължително съобщение, но съдържанието му зависи от избраният суит.

2.2.1.7 ChangeCipherSpec – съобщение за смяна към криптирана комуникация

Това съобщение се изпраща за да сигнализира приемащата страна че изпращачът е получил цялата необходима информация да имплементира параметрите на сесията, да генерира съответните ключове и че преминава към криптиране на съобщенията си. Това съобщение се изпраща и от сървърната част и от клиентската.

2.2.1.8 Finished – съобщение че ръкостискането е приключило

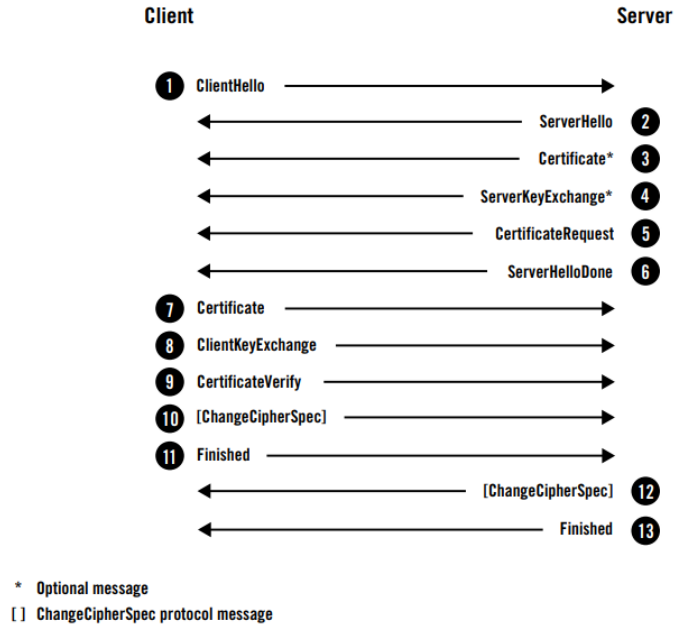
Сигнализира другата страна че ръкостискането е приключило успешно. Това съобщение е криптирано и поради това съдържа информация с която да се верифицира интегритета на сесията.

2.2.2 Ръкостискане с автентикация на клиента

Въпреки че автентификацията от стана на сървъра и на клиента технически не е задължителна. Автентикация на сървъра универсално се изисква. Ако сървъра избере суит който изисква автентикация, сървърът е задължен да изпрати Certificate съобщение след това.

За автентикация на клиента, сървърът трябва да изпрати CertificateRequest съобщение към клиента което съдържа списък със всички методи за проверки на сертификата които познава. След това, подобно са сървъра, клиентът изпраща Certificate съобщение и потвърждава че той е притежател на личния ключ с CertificateVerify съобщение.

Само автентикират сървър може да изисква автентикация от клиента.



Фиг. 7 – ръкостискане с автентикация на клиента

2.2.2.1 CerticateRequest – съобщение искащо автентикация

С това съобщение сървъра изисква клиентът да се автентикира като предоставя валидни формати на публични ключове, и алгоритмите за проверка на подписа на сертификат, които поддържа.

2.2.2.2 CertificateVerify – съобщение за верификация на идентичността

Изпраща се от клиента като доказателство че той е притежателят на личният ключ съответстващ на публичният ключ изпратен със сертификата в извратеното съобщение Certificate. В това съобщение съдържа подписана сигнатура с на всички получени и предадени съобщения които са участвали в ръкостискането.

2.3 Обмен на ключове

Процесът по размяна на ключовете е най-интересната част от ръкостискането. В TLS, сигурността на сесията зависи от 48 битов споделен ключ наречен главна тайна (master secret). При обмена на ключове целта е да се изгенерира стойност наречена основна тайна (premaster secret). Основната тайна в последствие се използва за да се генерира главната тайна.

Най често срещаните алгоритми за обмен на ключове са представени на Таблица 1. Изборът на алгоритъм зависи от избраният суит.

Key Exchange	Description
dh_anon	Diffie-Hellman (DH) key exchange without authentication
dhe_rsa	Ephemeral DH key exchange with RSA authentication
ecdh_anon	Ephemeral Elliptic Curve DH (ECDH) key exchange without authentication (RFC 4492)
ecdhe_rsa	Ephemeral ECDH key exchange with RSA authentication (RFC 4492)
ecdhe_ecdsa	Ephemeral ECDH key exchange with ECDSA authentication (RFC 4492)
krb5	Kerberos key exchange (RFC 2712)
rsa	RSA key exchange and authentication
psk	Pre-Shared Key (PSK) key exchange and authentication (RFC 4279
dhe_psk	Ephemeral DH key exchange with PSK authentication (RFC 4279)
rsa_psk	PSK key exchange and RSA authentication (RFC 4279)
srp	Secure Remote Password (SRP) key exchange and authentication (RFC 5054)

Таблица 1

Без значение от използваният алгоритъм, ServerKeyExchange съобщението винаги се изпраща преди ClientKeyExchange. Понеже не всички алгоритми имат нужда да предават информация с това съобщение, (ако тя вече се знае) то може да не се изпрати по време на преговарянето. В повечето случаите когато има нужда да се предава информация, в съобщението се добавя и сигнатура, което практически е подпис върху параметрите които изпраща.

Това се използва за автентикация. Ако клиентът може да декриптира сигнатурата с публичният ключ, предоставен в сертификата от сървъра, и при сравнение на декриптираната чек сума със тази на приетите параметри има съвпадение, то клиентът може да е сигурен че тази сигнатура е подписана с личният ключ на сървъра и информацията не е подправена. Пример за структурата на ServerKeyExchange съобщението при различните алгоритми може да се види на Фиг. 8.

```
struct {
        select (KeyExchangeAlgorithm) {
                case dh_anon:
                        ServerDHParams params;
                case dhe_rsa:
                        ServerDHParams params;
                        Signature params_signature;
                case ecdh_anon:
                        ServerECDHParams params;
                case ecdhe_rsa:
                case ecdhe_ecdsa:
                        ServerECDHParams params;
                        Signature params_signature;
                case rsa:
                case dh_rsa:
                        /* no message */
        };
} ServerKeyExchange;
```

Фиг. 8 – съдържание на ServerKeyExchange съобщението

Съобщението ClientKeyExchange, за разлика от предишното разглеждано съобщение, е задължително. Както при сървъра, то се използва за да пренася параметри необходими за генерацията на основната тайна (premaster secret). Пример за структурата му може да се види в псевдокода на Фиг. 9.

```
struct {
        select (KeyExchangeAlgorithm) {
                case rsa:
                        EncryptedPreMasterSecret;
                case dhe_dss:
                case dhe_rsa:
                case dh_dss:
                case dh_rsa:
                case dh_anon:
                        ClientDiffieHellmanPublic;
                case ecdhe:
                        ClientECDiffieHellmanPublic;
        } exchange_keys;
} ClientKeyExchange;
```

Фиг. 9 – съдържание на ClientKeyExchange съобщението

2.3.1 RSA алгоритъм за обмен на ключове

Това е най-лесният метод за обмен на ключове. Клиентът генерира пре-главна тайна, което е 46 байтово произволно число, криптира го с публичният ключ, предоставен от сървъра, и го изпраща със ClientKeyExchange съобщението.

Ниската сложност на този алгоритъм е и неговата слабост. Понеже за криптиране се използва публичният ключ на сървъра. Тези ключове се сменят веднъж на няколко години. Означава че ако някой успее да се сдобие с личният ключ на сървъра, той може да декриптира параметрите и да си генерира основната и главната тайна, компрометирайки поверителността на сесията.

2.3.2 Diffie-Hellman алгоритъм за обмен на ключове

Това е алгоритъм който позволява на две страни, които искат да комуникират през незащитена среда, да изградят общ ключ с който да криптират съобщенията които ще си обменят.

Идеята при този алгоритъм е да се използва математическа функция, която да може лесно да се изчислява, но да е много трудно от полученият резултат да се стигне до стойностите на входните аргументи на тази функция. Това е аналогично на смесването на цветове. Ако се вземат два цвята и се смесят се получава трети който е комбинация от двата. Но обратното би било много по трудно, от смесеният цвят да се изкарат двата цвята участвали в образуването му.

За работата на този алгоритъм са нужни шест параметъра. Два от тях (dh_p и dh_g) се наричат домейн параметри и се селектират от сървъра. По време на преговорите, клиентът и сървърът генерират другите четири параметъра, като сървъра генерира два и клиента генерира два. След това те си ги изпращат с KeyExchange съобщенията си. Когато една страна получи всички шест параметъра, тя може да започне да генерира главната тайна.

2.4 Автентикация

Автентикацията в TLS е пряко свързана със процесът на обмен на ключове с цел да се спестят бавни криптографски операции. В повечето случаи автентификацията ще се основава на автентикация посредством публичен ключ (RSA, ECDSA) предоставен чрез сертификат. След потвърждение на валидността на сертификата, е ход на избраният метод на автентикация да определи как ще се използва публичният ключ за да автентикира другата страна.

Когато се използва RSA за обмен на ключове, клиентът генерира произволно число като основна тайна и я криптира използвайки

публичният ключ на сървъра. Сървъра, притежавайки съответстващият частен ключ, декриптира съобщението за да се сдобие с основната тайна. Тук автентикацията е имплицитна. Предполага се че ако сървъра не би могъл да декриптира съобщението и последователно на това няма да може да генерира главната тайна и да комуникира с клиента.

При използване на DHE метода, сървъра предоставя своите параметри за генериране на основната тайна, като също ги подписва със своят частен ключ. Клиентът, притежавайки публичният ключ, може да верифицира дали приетите параметри наистина са били изпратени от сървъра.

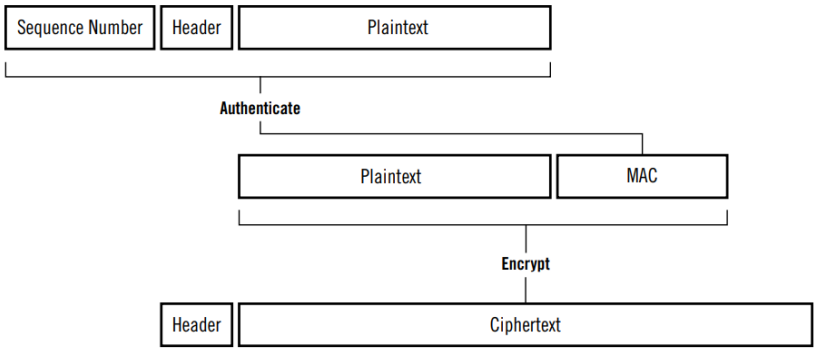
В допълнение с параметрите в сигнатурата се съдържа и произволна информация, генерирана от сървъра и клиента с цел да направи сесията уникална. Това не позволява на трето лице да манипулира процеса на ръкостискане като инжектира прихванатата сигнатура по време на обмена на ключове.

2.5 Криптиране

TLS поддържа много алгоритми и методи да криптиране на информацията, използвайки шифри като 3DES, AES, ARIA, CAMEL-LIA, RC4, и SEED, като AES е най-широко използван. Поддържат се три метода на криптиране: поточно, блоково и автентикационно криптиране.

2.5.1 Поточно криптиране

Когато се използва поточен шифър, криптирането се състои от две стъпки. В първата се генерира MAC на комбинацията от поредният номер, етикета на записа както и явният текст на информацията (в не-криптиран вариант). След това MAC-а, заедно с информацията се криптират и се поставят в тялото на записва. Това може да се види на Фиг. 10. Добавянето на MAC в криптираният текст гарантира интегритета на съобщението и предпазва сесията от reply атаки.



Фиг. 10

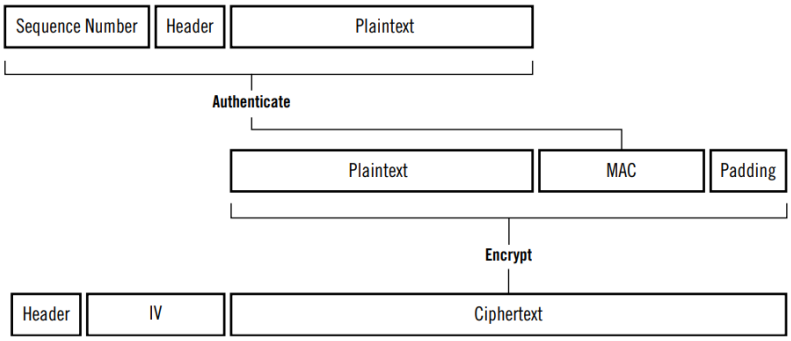
2.5.2 Боково криптиране

При използване на блокови шифри, процесът на криптиране се усложнява. Това се получава поради спецификите на блоковите шифри. Процесът може да се раздели на следните стъпки:

1. Изчислява се MAC от поредния номер на записа, етикета му и явният текст.

2. Към явният текст се добавят байтове, тъй като информацията която ще се криптира трябва да е с размер кратен на блоковият размер на шифъра.
3. Генерира се произволен инициализационен вектор със същият размер равен на размерът на блока на шифъра. Това се използва за да направи блоковият алгоритъм не-детерминистичен.
4. Използва се CBC режим на криптиране за да се криптира явният текст, MAC-а, както и допълващите байтове.

Този процес е показан на Фиг. 11.



Фиг. 11

2.6 Протокол на приложението

Протоколът на приложението е просто абстракция с която се пренасят съобщенията от приложението което ползва TLS. За TLS това са просто буфери с информация. Тези буфери се предават на запис протокола, който ги пакетира, криптира, фрагментира в зависимост от установените параметри на сесията.

2.7 Протокол при тревога

Тези съобщения се използват за примитивна форма на сигнализация за да информират обратната страна при необичайни ситуации. Използват се за пренасяна на съобщения за грешка. Структурата на съобщенията за този протокол са показани на Фиг. 12.

```
struct {
        AlertLevel level;
        AlertDescription description;
} Alert;
```

Фиг. 12

AlertLevel полето носи сериозността на тревогата, което може да е или warning или fatal. AlertDescription носи кодът на съответната грешка.

Съобщенията маркирани с fatal, водят до незабавно терминиране на връзката и анулиране на валидността на сесията. При изпращане на съобщение маркирано с warning, изпращаната страна не терминира връзката но получателят може да реагира на това като изпрати съобщение за тревога маркирано с fatal и да разпадне сесията.

2.8 Терминиране на връзката

Съобщенията за терминиране на връзката (closure alerts) се използват за разпадане на TLS връзката в нормални условия. Когато едната страна иска да затвори връзката тя изпраща close_notify съобщение за тревога. Когато другата страна го получи, то спира назначеното предаване на всички чакащи съобщения и изпраща close_notify обратно.

Простият процес на терминиране предпазва връзката от атаки при които прекратяват обмена на съобщения. Без процеса на терминиране, двете стани не биха могли да различат дали са атакувани или просто едната страна е решила да затвори връзката.

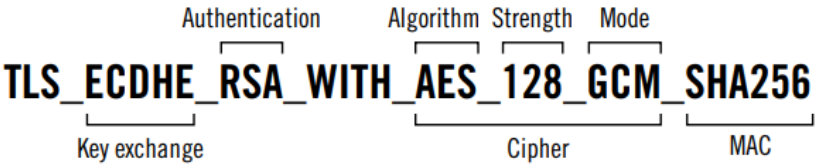
2.9 Криптографски операции

2.10 Методи за шифриране

TLS протоколът позволява голяма гъвкавост при имплементиране на суитове. Шифър суит е съвкупност от криптографски примитиви и допълнителни параметри, дефиниращи как сигурността по връзката би работила. Един суит дефинира следните атрибути:

- Метод за автентикация
- Метод за обмен на ключове
- Алгоритъм за криптиране
- Дължина на ключът за криптиране
- Режим на шифриране
- MAC алгоритъм

Имената на суитовете се изграждат по следният начин. Метод за обмен на ключове, метод за автентикация, алгоритъм за криптиране, дължина на ключът за криптиране, режим на шифриране и MAC алгоритъм. Пример за това може да се види на фиг.



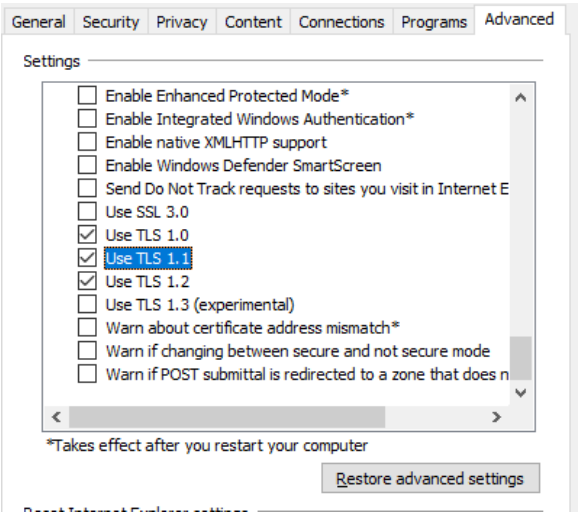
Фиг. 13

Въпреки че името не може да покаже всички параметри които влизат в суита. То дава информация за най-важните му компоненти и представа как работи. Пълна информация за съответен суит може да се намери в RFC-то в което е дефинирано.

Глава 3: Методи за конфигуране на TLS.

3.1 Конфигуране на TLS в Windows.

В операционната система Windows TLS може да се активира от Control Panel > Internet options > Advanced tab. Тук може да се види контекстно меню с тикчета. В края на този списък може да се видят маркирани версиите на TLS които е разрешено да се използват в момента. Това се вижда на Фиг. 14.

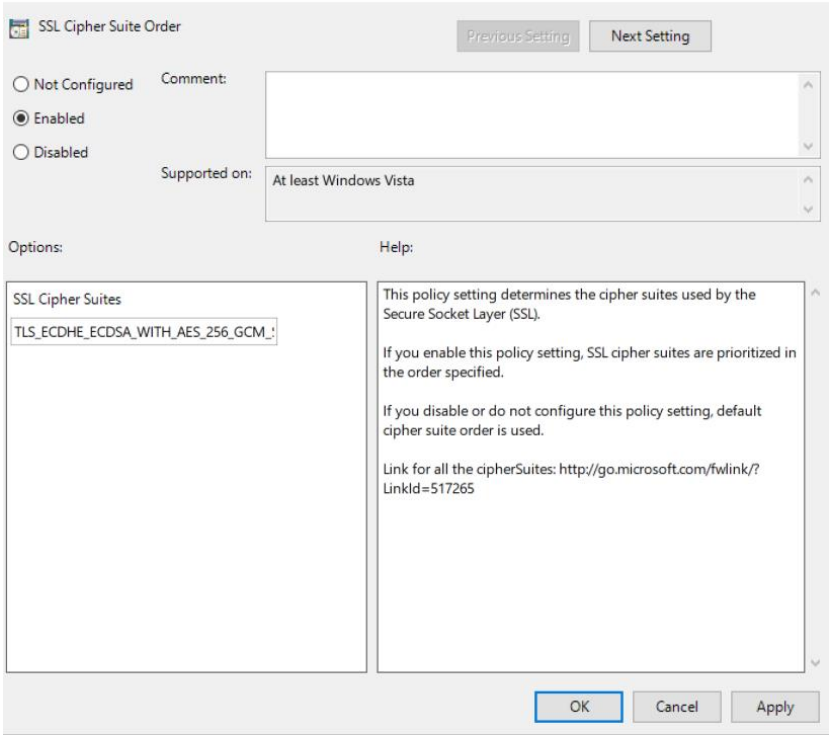


Фиг. 14

3.1.1 Конфигуриране на приоритета на различните шифри в TLS

1. Computer Configuration > Administrative Templates > Network > SSL Configuration Settings
2. За да може да се променя подредбата която е по подразбиране, трябва да се натисне радио бутонът, Enable.

3. Натиска се дясно копче върху SSL Cipher Suites правоъгълника. След това излиза pop-up меню при което се натиска Select all.
4. Полученият резултат може да се копира в някой текстови редактор и там да се направят промени по подредбата.
5. Замества се списъка със модифицираният.
6. Натиска се бутона OK или Apply



Фиг. 15

3.1.2 Използване на командата certutil (програмата certutil.exe) за менажиране на сертификати и доверени сертификати в windows.

Certutil позволява да се добавят и премахват сертификати и лични ключове. Тази програма дава възможност да се променя windows keystore (или truststore), което се използва от различни вградени windows програми както и 3rd party приложения които искат да създадат TLS връзка.

- Добавяне на сертификат – Синтаксисът на командата е следният:

  certutil [options] -addstore certificatestorename infile

  Като certificatestorename трябва да е уникално име във самият keystore a infile е пътят към сертификата който искаме да добавим
- Премахване на сертификат – Синтаксисът на командата е следният:

  certutil [options] -delstore certificatestorename certID

  Като certificatestorename трябва да е уникално име във самият keystore a certID е инфикикатор на съответният сертификат
- Достъпване на сертификат – Синтаксисът на командата е следният:

  certutil [options] -viewstore [certificatestorename [certID [outputfile]]]

  Като certificatestorename трябва да е уникално име във самият keystore a certID е инфикикатор на съответният сертификат. Може също да се добави и outputfile в който ще се запише информацията от изхода на командата. Когато тази команда се изивка без името на сертификата тя върща всички налични сертификати на машинта.

3.2 Конфигуриране на TLS в Linux (RHEL).

Понеже конфигурацията за приоритета при избиране на шифри и други настройки на TLS в Линукс се настройват от конфигурациите на съответните сървърни и клиентски приложения, тук ще се разгледа само механизма за добавяне и премахване на сертификати.

3.2.1 Метод за добавяне на доверени сертификати към Linux операционната система

- Добавяне на сертификат – Синтаксисът на командата е следният:

  trust anchor infile

  Където infile е пътят към сертификата който искаме да добавим

- Премахване на сертификат – Синтаксисът на командата е следният:

  trust anchor --remove "pkcs11:id=%AA%BB%CC%DD%EE;type=cert"

  Като тук се специфицира идентификатора на сертификата. Премахване е възможно и с пътя към сертификата ако той се намера на файловата система. Програмата ще извелче индентификатора от сертификат файла.

- Достъпване на сертификат – Синтаксисът на командата е следният:

  trust list

  Тази команда принтирва информация за всички инсталирани сертификати на машината

3.3 Начин за използване и настройка на TLS във Java виртуалната машина (JVM).

Java виртуалната машина намира широко приложение в съвременният свят. Голям брой сървърни приложения използват тази виртуална машина. Затова би било практически ценно да се разгледа как може да се променят настройките на TLS протокола в нея.

На Фиг. 16 е показан код на езикът Kotlin в който се отваря SSLServerSocket и се задават параметри за TLS връзката.

```
6.fun main() {
7.    val factory = SSLServerSocketFactory.getDefault()!!
8.    (factory.createServerSocket(8083) as SSLServerSocket).use { server ->
9.        server.needClientAuth = true
10.       server.enabledCipherSuites = arrayOf("TLS_AES_128_GCM_SHA256")
11.       server.enabledProtocols = arrayOf("TLSv1.3")

12.       server.accept().use { client ->
13.           val bytes = client.getInputStream().use { stream -> stream.readAllBytes() }
14.           if(bytes != null) {
15.               println(String(bytes))
16.           }
17.           client.getOutputStream().use { stream -> stream .write("Message Received!".encodeToByteArray()) }
18.       }
19.   }
20.}
```

Фиг. 16 – примерен код на TLS сървър.

На ред 8 се отваря сокета, който слуша на порт 8083. Тук е използван метод ‘use’ който гарантира че ще затвори ресурса след използването му. На ред 9 настройваме сървъра да изисква от клиента сертификат за автентикация. На ред 10 ограничаваме TLS протокола да използва само шифрите предоставени в масива. Ако клиентът не поддържа нито един от зададените шифри то TLS сесия не можа да се осъществи. На ред 10 ти задаваме допустимите версии в които може да работи TLS протокола. В примера на Фиг. 16 това е само TLS версия 1.3. Ако клиентът не поддържа 1.3 то TLS връзка не може да се осъществи. В следващите редове сървъра започва да слуша за клиенти. Когато един клиент се свърже се създава клиентски сокет. През сокета се взима информацията идваща от клиента и на клиента му се изпраща съобщение че информацията е приета.

Чрез специфициране на допустими шифри и версии сериозно се намалява полето за атака върху връзката. Разбира се TLS не е нужно да се настройва само от изпълняващият код. В този пример ние създаваме сървър сокет със

параметри по подразбиране, които от своя стана са настроени в Java вирулентната машина. Тези параметри са декларирани в jre/conf/security/java.security, но могат да се променят само за текущ процес, като процесът се стартира със системни параметри със същите имена като дефинираните във файла.

Цитирани източници

[1] „What is Transport Layer Security (TLS)?," [Онлайн]. Available: https://www.cloudflare.com/learning/ssl/transport-layer-security-tls/.

[2] „RFC 5246 The Transport Layer Security (TLS) Protocol," [Онлайн]. Available: https://datatracker.ietf.org/doc/html/rfc5246.

[3] „Working with Certificates and SSL," [Онлайн]. Available: https://docs.oracle.com/cd/E19830-01/819-4712/ablqw/index.html.

[4] „JAVA SSL Configuration," [Онлайн]. Available: https://www.baeldung.com/java-https-client-certificate-authentication.

[5] I. Ristić, Bullerproof SSL and TLS, 2015.

[6] „Какво е SSL?," [Онлайн]. Available: https://bg.education-wiki.com/9492606-what-is-ssl.

[7] Oracle, „Sample Code Illustrating a Secure Socket Connection Between a Client and a Server," [Онлайн]. Available: https://docs.oracle.com/javase/10/security/sample-code-illustrating-secure-socket-connection-client-and-server.htm#JSSEC-GUID-A4D59ABB-62AF-4FC0-900E-A795FDC84E41.