Розділ 15. Механізми безпеки комп'ютерних мереж

15.1. Віртуальні приватні мережі (VPN).
15.2. Протоколи автентифікації RADIUS.
15.3. Протоколи SSL/TLS.

15.1. Віртуальні приватні мережі (VPN)

VPN (англ. Virtual Private Network «Віртуальна приватна мережа») – узагальнена назва технологій, що дозволяють забезпечити одне або кілька мережевих з'єднань поверх іншої мережі, наприклад Інтернет.

Незважаючи на те, що для комунікації використовуються мережі з меншим або невідомим рівнем довіри (наприклад, публічні мережі), рівень довіри до побудованої логічної мережі не залежить від рівня довіри до базових мереж завдяки використанню засобів криптографії (шифрування, автентифікації, інфраструктури відкритих ключів, засобів захисту від повторів і змін повідомлень, що передаються по логічній мережі).

Залежно від застосовуваних протоколів та призначення VPN може забезпечувати з'єднання трьох видів: вузол-вузол, вузол-мережа та мережа-мережа.

Рівні реалізації

Зазвичай VPN розгортають рівнях не вище мережевого, оскільки застосування криптографії цих рівнях дозволяє використовувати у незмінному вигляді транспортні протоколи (такі як TCP, UDP).

Користувачі Microsoft Windows позначають терміном VPN одну з реалізацій віртуальної мережі – **PPTP**, причому використовується часто не для створення приватних мереж.

Найчастіше для створення віртуальної мережі використовується інкапсуляція протоколу **PPP** в будь-який інший протокол – **IP** (такий спосіб використовує реалізація **PPTP** – Point-to-Point Tunneling Protocol) або Ethernet (**PPPoE**) (хоча вони мають відмінності).

Технологія VPN останнім часом використовується не лише для створення власне приватних мереж, а й деякими провайдерами « останньої милі » на пострадянському просторі для надання виходу до Інтернету.

При належному рівні реалізації та використання спеціального програмного забезпечення мережа VPN може забезпечити високий рівень шифрування інформації, що передається.

Структура VPN

VPN складається з двох частин: «внутрішня» (підконтрольна) мережа, яких може бути кілька, і «зовнішня» мережа, через яку проходить інкапсульоване з'єднання (зазвичай використовується Інтернет).

Можливе також підключення до віртуальної мережі окремого комп'ютера.

Підключення віддаленого користувача до VPN здійснюється за допомогою сервера доступу, який підключений як до внутрішньої, так і зовнішньої (загальнодоступної) мережі. При підключенні віддаленого користувача (або під час встановлення з'єднання з іншою захищеною мережею) сервер доступу вимагає проходження процесу ідентифікації, а потім процесу автентифікації. Після успішного проходження обох процесів віддалений користувач (віддалена мережа) наділяється повноваженнями для роботи в мережі, тобто відбувається процес авторизації.

Класифікація VPN

Класифікувати рішення VPN можна за кількома основними параметрами:

За рівнем захищеності використовуваного середовища:

– Захищені. Найпоширеніший варіант віртуальних приватних мереж. З його допомогою можна створити надійну та захищену мережу на основі ненадійної мережі, як правило, Інтернету. Прикладом захищених VPN є: IPSec, OpenVPN та PPTP.

– Довірчі. Використовуються у випадках, коли середовище, що передає, можна вважати надійним і необхідно вирішити лише завдання створення віртуальної підмережі в рамках більшої мережі. Проблеми безпеки стають неактуальними. Прикладами подібних рішень VPN є: Multi-protocol label switching (MPLS) і L2TP (Layer 2 Tunnelling Protocol) (точніше буде сказати, що ці протоколи перекладають завдання безпеки на інші, наприклад L2TP, як правило, використовується в парі з IPSec).

За способом реалізації:

– У вигляді спеціального програмно-апаратного забезпечення. Реалізація мережі VPN здійснюється за допомогою спеціального комплексу програмно-апаратних засобів. Така реалізація забезпечує високу продуктивність і, зазвичай, високий рівень захищеності.

– У вигляді програмного рішення. Використовується персональний комп'ютер із спеціальним програмним забезпеченням, що забезпечує функціональність VPN.

– Інтегроване рішення. Функціональність VPN забезпечує комплекс, що вирішує також завдання фільтрації мережевого трафіку, організації мережевого екрана та забезпечення якості обслуговування.

За призначенням:

– Intranet VPN. Використовується для об'єднання в єдину захищену мережу кількох розподілених філій однієї організації, які обмінюються даними відкритими каналами зв'язку.

– Remote-access VPN. Використовується для створення захищеного каналу між сегментом корпоративної мережі (центральним офісом або філією) та одиночним користувачем, який, працюючи вдома, підключається до корпоративних ресурсів із домашнього комп'ютера, корпоративного ноутбука, смартфона або інтернет-кіоску.

– Extranet VPN. Використовується для мереж, до яких підключаються «зовнішні» користувачі (наприклад, замовники або клієнти). Рівень довіри до

них набагато нижчий, ніж до співробітників компанії, тому потрібне забезпечення спеціальних «рубежів» захисту, які запобігають чи обмежують доступ останніх до особливо цінної, конфіденційної інформації.

– Internet VPN. Використовується провайдерами для надання доступу до інтернету, зазвичай, якщо по одному фізичному каналу підключаються декілька користувачів. Протокол **PPPoE** став стандартом **ADSL** – підключення. **L2TP** був широко поширений у середині 2000-х років у будинкових мережах: на той час внутрішньомережевий трафік не оплачувався, а зовнішній коштував дорого. Це дозволяло контролювати витрати: коли VPN-з'єднання вимкнено, користувач нічого не платить. В даний час (2012) дротовий інтернет – дешевий або безлімітний, а на стороні користувача найчастіше є маршрутизатор, на якому вмикати-вимикати інтернет не так зручно, як на комп'ютері. Тому L2TP-доступ відходить у минуле.

– Client/server VPN. Цей варіант забезпечує захист даних між двома вузлами (не мережами) корпоративної мережі. Особливість цього варіанта в тому, що VPN будується між вузлами, що знаходяться, як правило, в одному сегменті мережі, наприклад, між робочою станцією та сервером. Така необхідність часто виникає у тому випадку, як у однієї фізичної мережі необхідно створити кілька логічних мереж. Наприклад, коли треба розділити трафік між фінансовим департаментом та відділом кадрів, які звертаються до серверів, що знаходяться в одному фізичному сегменті. Цей варіант схожий на технологію VLAN, але замість поділу трафіку використовується його шифрування.

За типом протоколу:

– Існують реалізації віртуальних приватних мереж під TCP/IP, IPX та AppleTalk. Але на сьогоднішній день спостерігається тенденція до загального переходу на протокол TCP/IP, і більшість рішень VPN підтримує саме його. Адресація в ньому найчастіше вибирається відповідно до стандарту RFC5735, з діапазону Приватних мереж TCP/IP.

За рівнем мережевого протоколу:

– За рівнем мережевого протоколу з урахуванням зіставлення з рівнями еталонної мережевої моделі ISO/OSI.

VPN-з'єднання на маршрутизаторах

Зі зростанням популярності VPN-технологій багато користувачів стали активно налаштовувати VPN-з'єднання на маршрутизаторах задля підвищення безпеки в мережі. VPN-з'єднання, налаштоване на маршрутизаторі, шифрує мережевий трафік всіх під'єднаних пристроїв, у тому числі і тих, які не підтримують технології VPN.

Багато маршрутизаторів підтримують VPN-з'єднання та мають вбудований VPN-клієнт. Існують маршрутизатори, для яких потрібне програмне забезпечення з відкритим вихідним кодом, таке як DD-WRT, OpenWrt і Tomato для того, щоб підтримувати протокол OpenVPN.

Вразливості

Використання технології **WebRTC**, яка за замовчуванням включена в кожному браузері, дозволяє третій стороні визначити реальну публічну IP-адресу пристрою, що працює через VPN. Це є прямою загрозою для конфіденційності, оскільки, знаючи справжню IP-адресу користувача, можна однозначно ідентифікувати її в мережі. Для запобігання витоку адреси рекомендується або повністю відключити WebRTC в налаштуваннях браузера, або встановити спеціальний додаток.

VPN вразливі для атаки, яка називається дактилоскопією трафіку веб-сайту. Дуже коротко: це пасивна атака перехоплення; хоча супротивник тільки спостерігає зашифрований трафік з VPN, він все ще може здогадатися, який сайт відвідується, тому що всі веб-сайти мають певні шаблони трафіку. Зміст передачі, як і раніше, приховано, але до якого веб-сайту він підключається, більше не є секретом.

Приклади VPN

–IPSec (IP security) – часто використовується поверх IPv4.

–PPTP (point-to-point tunneling protocol) – розроблявся спільними зусиллями кількох компаній, включаючи Microsoft.

–PPPoE (PPP (Point-to-Point Protocol) over Ethernet)

–L2TP (Layer 2 Tunnelling Protocol) – використовується в продуктах компаній Microsoft та Cisco.

–L2TPv3 (Layer 2 Tunnelling Protocol version 3).

–OpenVPN SSL – VPN з відкритим вихідним кодом, підтримує режими PPP, bridge, point-to-point, multi-client server.

–freelan SSL P2P – VPN з відкритим вихідним кодом.

–Hamachi – програма для створення однорангової VPN-мережі.

–NeoRouter – zeroconf (не потребує налаштування) програма для забезпечення прямого з'єднання комп'ютерів за NAT, є можливість вибрати свій сервер.

Багато великих провайдерів пропонують свої послуги з організації VPN-мереж для бізнес-клієнтів.

OpenVPN

OpenVPN – вільна реалізація технології віртуальної приватної мережі (VPN) з відкритим вихідним кодом для створення зашифрованих каналів типу точка або сервер-клієнти між комп'ютерами. Вона дозволяє встановлювати з'єднання між комп'ютерами, що знаходяться за NAT і мережевим екраном без необхідності зміни їх налаштувань. OpenVPN була створена Джеймсом Йонаном (James Yonan) і поширюється під ліцензією GNU GPL.

Для забезпечення безпеки керуючого каналу та потоку даних OpenVPN використовує бібліотеку OpenSSL. Це дозволяє використовувати весь набір алгоритмів шифрування, доступних у цій бібліотеці. Також може використовуватися пакетна автентифікація HMAC для забезпечення більшої безпеки та апаратне прискорення для покращення продуктивності шифрування. Ця бібліотека використовує OpenSSL, а точніше протоколи SSLv 3/ TLSv 1.2.

OpenVPN використовується в операційних системах Solaris, OpenBSD, FreeBSD, NetBSD, GNU/ Linux, Apple Mac OS X, QNX, Microsoft Windows, Android, iOS.

Автентифікація

OpenVPN пропонує користувачеві кілька видів автентифікації.

– Передвстановлений ключ – найпростіший метод.

– Сертифікатна автентифікація – найбільш гнучкий метод налаштування.

– За допомогою логіну та пароля може використовуватися без створення клієнтського сертифікату (серверний сертифікат все одно потрібен).

Мережа

OpenVPN проводить усі мережеві операції через TCP- або UDP-транспорт. У загальному випадку переважним є UDP з тієї причини, що через тунель проходить трафік мережевого рівня і вище OSI, якщо використовується TUN-з'єднання, або трафік канального рівня і вище, якщо використовується TAP. Це означає, що OpenVPN для клієнта виступає протоколом канального або навіть фізичного рівня, а значить, надійність передачі даних може забезпечуватися вищими за рівнями OSI, якщо це необхідно. Саме тому протокол UDP за своєю концепцією найбільш близький до OpenVPN, тому що він, як і протоколи канального та фізичного рівнів, не забезпечує надійності з'єднання, передаючи цю ініціативу вищим рівням. Якщо ж налаштувати тунель на роботу з TCP, сервер у типовому випадку отримуватиме TCP-сегменти OpenVPN, які містять інші TCP-сегменти від клієнта. Також можлива робота через більшу частину проксі-серверів, включаючи HTTP, SOCKS, через NAT та мережеві фільтри. Сервер може бути налаштований на призначення мережевих установок клієнту. Наприклад: IP-адреса, налаштування маршрутизації та параметри з'єднання. OpenVPN пропонує два різні варіанти мережевих інтерфейсів, використовуючи драйвер TUN/TAP. Можливо створити тунель мережевого рівня, званий TUN, і канального рівня – TAP, здатний передавати Ethernet -трафік. Також можливе використання бібліотеки компресії LZO для стиснення потоку даних. Використовуваний порт 1194 виділено Internet Assigned Numbers Authority для роботи цієї програми. Версія 2.0 дозволяє одночасно керувати кількома тунелями на відміну від версії 1.0, що дозволяла створювати лише 1 тунель на 1 процес.

Використання в OpenVPN стандартних протоколів TCP та UDP дозволяє йому стати альтернативою IPsec у ситуаціях, коли Інтернет-провайдер блокує деякі VPN-протоколи.

15.2. Протоколи автентифікації RADIUS

RADIUS (англ. Remote Authentication in Dial-In User Service) – протокол для реалізації автентифікації, авторизації та збору відомостей про використані ресурси, розроблений для передачі відомостей між центральною платформою та обладнанням. Цей протокол застосовувався для системи тарифікації використаних ресурсів конкретним користувачем/абонентом. Центральна платформа та обладнання Dial-Up доступу (Network Access Server (NAS, не

плутати зі сховищем) із системою автоматизованого обліку послуг (білінгу)), RADIUS використовується як протокол **AAA**:

– англ. Authentication – процес, що дозволяє автентифікувати (перевірити справжність) суб'єкта за його ідентифікаційними даними, наприклад, за логіном (ім'я користувача, номер телефону тощо) та паролем.

– англ. Authorization – процес, що визначає повноваження ідентифікованого суб'єкта на доступ до певних об'єктів або сервісів.

– англ. Accounting – процес, що дозволяє вести збір відомостей (облікових даних) про використані ресурси. Первинними даними (тобто традиційно передаються за протоколом RADIUS) є величини вхідного та вихідного трафіку: в байтах/октетах (з недавніх пір в гігабайтах). Однак протокол передбачає передачу даних будь-якого типу, що реалізується за допомогою **VSA** (**V** endor **S** pecific **A** ttributes).

Протокол RADIUS був розроблений Карлом Рігні (англ. Carl Rigney) у фірмі Livingston Enterprises для їх серверів доступу (Network Access Server) серії PortMaster до мережі інтернет, і пізніше, в 1997, був опублікований як **RFC 2058** і **RFC 2059** (поточні версії **RFC 2865** та **RFC 2866**).

На даний момент існує кілька комерційних і RADIUS-серверів, що вільно розповсюджуються. Вони дещо відрізняються один від одного за своїми можливостями, але більшість підтримує списки користувачів у текстових файлах, **LDAP**, різних баз даних.

Облікові записи користувачів можуть зберігатися в текстових файлах, різних базах даних або зовнішніх серверах.

Часто для віддаленого моніторингу використовується **SNMP**.

Існують проксі-сервери для RADIUS, які спрощують централізоване адміністрування та/або дозволяють реалізувати концепцію інтернет-роумінгу. Вони можуть змінювати вміст RADIUS-пакету на льоту (з метою безпеки або для перетворення між діалектами).

Популярність RADIUS-протоколу багато в чому пояснюється:

– відкритістю до наповнення новою функціональністю при збереженні працездатності зі застарілим обладнанням;

– надзвичайно високою реактивністю при обробці запитів через використання UDP як транспорт пакетів;

– добре паралелізується алгоритмом обробки запитів;

– здатністю функціонувати в кластерних (Cluster) архітектурах (наприклад OpenVMS) і мультипроцесорних (SMP) платформах (DEC Alpha, HP Integrity) – як для підвищення продуктивності, так і для реалізації відмовостійкості.

Існує протокол DIAMETER (поточні версії RFC 3588 та RFC 3589), який покликаний замінити RADIUS, надаючи механізм міграції.

Можливості

Будучи частиною білінгової системи, RADIUS-сервер є інтерфейсом взаємодії з телекомунікаційною системою або сервером (наприклад, маршрутизатором або комутатором) і може реалізовувати для такої системи такі сервіси:

Загальні

– Створення та зберігання облікових записів користувачів (абонентів).

– Керування обліковим записом користувача (абонента) з персонального інтерфейсу (наприклад, веб-кабінету).

– Створення карток доступу (логін або PIN-код) для надання послуг, з деяким лімітом дії (Dial-Up доступу до Інтернету та карткової IP-телефонії).

– Ручне та автоматичне блокування облікового запису абонента після досягнення заданого критерію або ліміту.

– Збір та аналіз статистичної інформації про сесії користувача та всієї системи, що обслуговується (у тому числі CDR).

– Створення звітів з різних статистичних параметрів.

– Створення, друк та відправка рахунків до оплати.

– Автентифікація всіх запитів у RADIUS-сервер з системи, що обслуговується (поле Secret).

Автентифікація

– Перевірка облікових даних користувача (у тому числі шифрованих) на запит системи, що обслуговується.

Авторизація

– Видача стану блокування облікового запису користувача.

– Видача дозволу на ту чи іншу послугу.

– Сортування даних на основі аналізу статистичної інформації (наприклад, динамічна маршрутизація) та видача результату сортування на на запит.

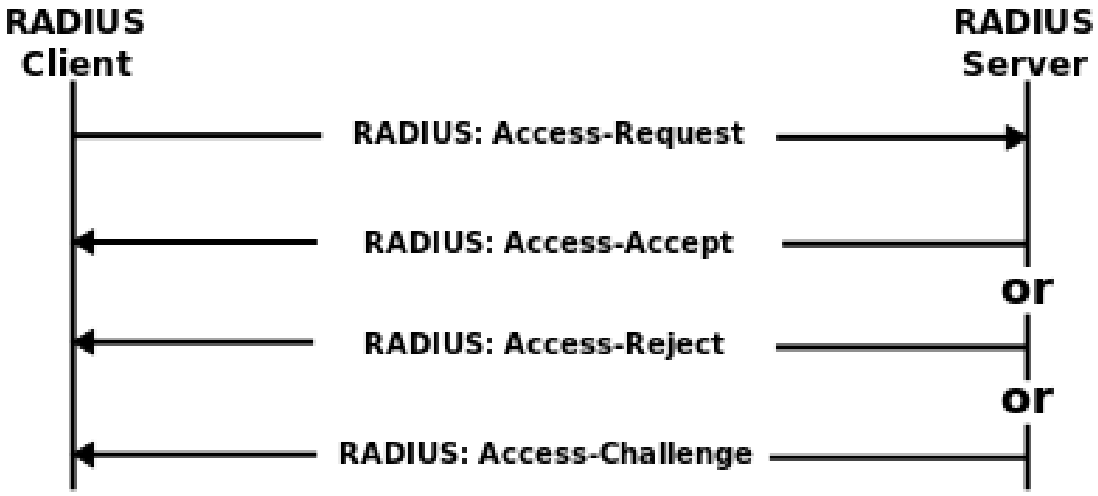

Рисунок 15.1 – Автентифікація та авторизація через RADIUS-сервер

Облік (Accounting)

– Онлайн-облік засобів абонента: повідомлення про початок і кінець сесії з боку системи, що обслуговується.

– Проміжні повідомлення про продовження сесії (Interim-пакети).

– Автоматичне примусове завершення дії сесії на системі, що обслуговується в рамках послуги (packet of disconnection).

– BOOT message – спеціальний пакет, який відправляється телекомунікаційною системою на RADIUS-сервер під час запуску (перезапуску) системи з метою примусового завершення всіх сесій.

В даний час протокол RADIUS використовується для доступу до віртуальних приватних мереж (VPN), точок бездротового (Wi-Fi) доступу, Ethernet-комутаторів, DSL та інших типів мережевого доступу. Завдяки відкритості, простоті впровадження, постійному вдосконаленню протокол RADIUS зараз є фактично стандартом для віддаленої автентифікації.

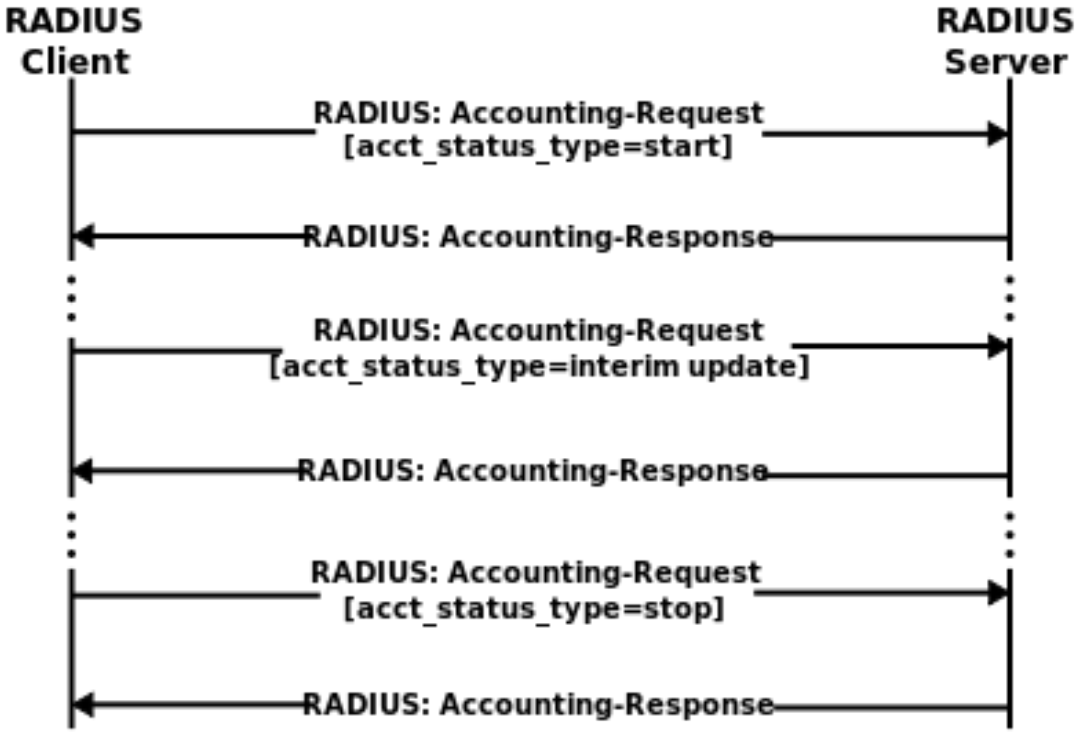

Рисунок 15.2 – Онлайн облік через RADIUS-сервер

Процес автентифікації та авторизації

Для визначення принципу роботи протоколу RADIUS необхідно розглянути наступне.

Ноутбуки та IP-телефон представляють пристрої користувача, з яких необхідно виконати автентифікацію та авторизацію на мережевих серверах доступу (NAS): точці Wi-Fi доступу, маршрутизаторі, VPN-сервері та IP АТС.

RADIUS реалізується у вигляді інтерфейсу між NAS (RADIUS-клієнт) та RADIUS-сервером – програмним забезпеченням, що встановлюється на

комп'ютері (сервері) або іншому спеціалізованому пристрої. Сервер взаємодіє з пристроєм користувача безпосередньо, а лише через мережевий сервер доступу.

Користувач надсилає запит на мережевий сервер для отримання доступу до певного мережевого ресурсу за допомогою сертифікату доступу. Сертифікат надсилається на сервер через мережевий протокол канального рівня (наприклад, PPP у разі комутованого доступу, DSL у разі використання відповідних модемів тощо). NAS, своєю чергою, посилає повідомлення запиту доступу на RADIUS-сервер (RADIUS Access Request). Запит включає сертифікати доступу, представлені у вигляді імені користувача та пароля або сертифікату безпеки, отримані від користувача. Запит може містити додаткові параметри: мережеву адресу пристрою користувача, телефонний номер, інформацію про фізичну адресу, з якою користувач взаємодіє з NAS.

Сервер **перевіряє інформацію на коректність**, використовуючи схеми **автентифікації**:

– **PAP** (Password Authentication Protocol) (RFC 1334) – простий автентифікаційний протокол, який використовується для автентифікації користувача стосовно мережевого сервера доступу (NAS). PAP використовується протоколом PPP. Майже всі сервери доступу підтримують PAP. PAP передає зашифрований пароль через мережу і, отже, при перехопленні трафіку пароль може бути схильний до атаки перебору. Тому PAP зазвичай використовується в тому випадку, коли сервер не підтримує захищені протоколи, такі як CHAP, EAP і т.п.

– **CHAP** (Challenge Handshake Authentication Protocol) (RFC 1994) – широко поширений алгоритм автентифікації, що передбачає передачу не самого пароля користувача, а непрямих відомостей про нього. У разі використання CHAP сервер віддаленого доступу надсилає клієнтові рядок запиту. На основі цього рядка та пароля користувача клієнт обчислює ґеш-код MD5 та передає його серверу. Ґеш-функція є алгоритмом одностороннього (необоротного) шифрування, оскільки значення ґеш-функції для блоку даних легко обчислити, а визначити вихідний блок з ґеш-коду з математичної точки зору неможливо за прийнятний час. Сервер, якому доступний пароль користувача, виконує ті ж обчислення і порівнює результат з ґеш-кодом, отриманим від клієнта. У разі збігу облікові дані клієнта віддаленого доступу вважаються справжніми.

– **MD5** (Message-Digest algorithm 5) (RFC 1321) – широко використовується криптографічна функція з 128 бітовим ґешем. В алгоритмі знайдено низку вразливостей, внаслідок чого Міністерство внутрішньої безпеки США не рекомендує використання MD5 у майбутньому.

– **EAP** (розширюваний протокол автентифікації) (RFC 3748) дозволяє перевіряти справжність при підключеннях віддаленого доступу за допомогою різних механізмів автентифікації. Точна схема автентифікації узгоджується клієнтом віддаленого доступу та сервером, що виконує автентифікацію (ним може бути сервер віддаленого доступу або RADIUS сервер). За промовчанням у маршрутизацію та віддалений доступ включена підтримка протоколів EAP-TLS та MD5-Challenge (MD5-завдання). Підключення інших модулів EAP до сервера,

що використовує маршрутизацію та віддалений доступ, забезпечує підтримку інших методів EAP. Протокол EAP дозволяє вести вільний діалог між клієнтом віддаленого доступу та системою автентифікації. Такий діалог складається із запитів системи автентифікації на необхідну їй інформацію та відповідей клієнта віддаленого доступу. Наприклад, Коли протокол EAP використовується з генераторами кодів доступу, сервер, що виконує автентифікацію, може окремо запитувати у клієнта віддаленого доступу ім'я користувача, ідентифікатор і код доступу. Після відповіді на кожен запит клієнт віддаленого доступу проходить певний рівень перевірки автентичності. Коли на всі запити будуть отримані відповіді, перевірка справжності клієнта віддаленого доступу успішно завершується.

Схеми автентифікації, що використовують протокол EAP, називаються типами EAP. Для успішної автентифікації клієнт віддаленого доступу та сервер, що виконує автентифікацію, повинні підтримувати один і той же тип EAP.

Тепер повернемося до RADIUS сервера, який перевіряє інформацію, отриману від NAS. Сервер перевіряє ідентичність користувача, а також коректність додаткової інформації, яка може міститися у запиті: мережева адреса пристрою користувача, телефонний номер, стан рахунку, його привілеї при доступі до запитуваного мережевого ресурсу. За результатами перевірки RADIUS сервер надсилає NAS один із трьох типів відгуків:

– **Access-Reject** показує, що цей запит користувача неправильний. За бажання сервер може включити текстове повідомлення в Access-Reject, яке може бути передано клієнтом користувачеві. Ніякі інші атрибути (крім Proxy-State) не дозволені у Access-Reject.

– **Access-Challenge**. Запит додаткової інформації від користувача, наприклад, другий пароль, пін-код, номер картки тощо. Цей відгук також використовується для більш повного автентифікаційного діалогу, де захисний тунель виконується між пристроєм користувача та сервером RADIUS, так що сертифікати доступу ховаються від NAS.

– **Access Accept**. Користувачеві дозволено доступ. Оскільки користувач автентифікований, RADIUS сервер перевіряє авторизацію на використання запитаних користувачем ресурсів. Наприклад, користувачеві може бути дозволений доступ через бездротову мережу, але заборонено доступ до мережі VPN.

Стандарти

Визначений у:

– RFC 2865 Remote Authentication Dial In User Service (RADIUS).
– RFC 2866 Облік RADIUS.

Також має відношення до:

– RFC 2548 Специфічні атрибути RADIUS Microsoft від постачальника.
– RFC 2607 З’єднання проксі та впровадження політики в роумінгу.
– RFC 2618 MIB клієнта автентифікації RADIUS.
– RFC 2619 Сервер автентифікації RADIUS MIB.
– RFC 2620 RADIUS Accounting Client MIB.
– RFC 2621 RADIUS обліковий сервер MIB.

– RFC 2809 Реалізація примусового тунелювання L2TP через RADIUS.

– RFC 2867 Модифікації обліку RADIUS для підтримки тунельного протоколу.

– RFC 2868 Атрибути RADIUS для підтримки тунельного протоколу.

– RFC 2869 Розширення RADIUS.

– Вимоги до серверів доступу до мережі RFC 2882: розширені практики RADIUS.

– RFC 3162 RADIUS і IPv6.

– RFC 3575 IANA, міркування щодо RADIUS.

– RFC 3576 Розширення динамічної авторизації для RADIUS.

– RFC 4672 MIB клієнта динамічної авторизації RADIUS.

– RFC 4673 MIB сервера динамічної авторизації RADIUS.

– RFC 3579 Підтримка RADIUS для EAP.

– RFC 3580 Інструкції з використання RADIUS IEEE 802.1X.

– RFC 4014 Підпараметр атрибутів RADIUS для параметра інформації агента ретрансляції DHCP.

DIAMETER

DIAMETER – сеансовий протокол, створений, частково, для подолання деяких обмежень протоколу RADIUS. Забезпечує взаємодію між клієнтами з метою автентифікації, авторизації та обліку різних сервісів (AAA, англ. authentication, authorization, accounting). Є основним протоколом архітектури IMS.

В основі протоколу DIAMETER лежить концепція створення базового протоколу з можливістю його розширення для надання сервісів AAA з появою нових технологій доступу.

Опис наведено у стандартах: RFC 6733 (Diameter Base Protocol), RFC 3589 (Diameter Command Codes for 3GPP), RFC 4006 (Diameter Credit-Control Application).

Порівняння з протоколом RADIUS

Назва DIAMETER – гра слів, що відображає перевагу нового протоколу над попередником RADIUS (діаметр – подвоєний радіус). Diameter не має зворотної сумісності до RADIUS, але надає механізми міграції. Серед відмінностей між протоколами особливо вирізняють:

– Підтримка транспортних протоколів з гарантованою доставкою (TCP або SCTP замість UDP). TCP транспорт для RADIUS ще у процесі стандартизації IETF.

– Захист даних на мережевому та транспортному рівнях (IPsec або TLS). Transport Layer Security for RADIUS ще в процесі стандартизації IETF.

– Підтримка переходу з RADIUS, незважаючи на те, що Diameter не повністю сумісний з RADIUS.

– Збільшений адресний простір для пар атрибут-значення (AVP) та ідентифікаторів (32 bit замість 8 bit).

– Модель клієнт-сервер. Як виняток, тим не менш, підтримуються деякі повідомлення, що ініціюються сервером.

– Підтримка stateful та stateless моделей використання.

– Динамічне виявлення вузлів (використовуючи DNS SRV та NAPTR).

– Можливість узгодження функціональних можливостей вузлів.

– Підтримка механізмів надійної доставки на рівні додатків, описані механізми відмовостійкості та state machine (RFC 3539).

– Повідомлення про помилки.

– Поліпшена підтримка мобільності.

– Поліпшена розширюваність; можливість використання команд і атрибутів користувача.

15.3. Протоколи SSL/TLS

SSL (Secure Sockets Layer) та **TLS (Transport Level Security)** – криптографічні протоколи, що забезпечують захищену передачу даних у комп'ютерній мережі. Вони широко використовуються у веб-браузерах, а також під час роботи з електронною поштою, обміну миттєвими повідомленнями та в IP-телефонії. Існує **SSL 1.0, 2.0 і 3.0**, на **підставі протоколу SSL 3.0 було розроблено та прийнято TLS 1.0.** Його також часто називають **SSL 3.1**. **SSL 1.0, 2.0, 3.0 і TLS**

Версія SSL 1.0 ніколи не була оприлюднена. Версія 2.0 була випущена в лютому 1995 року, але містила багато недоліків безпеки, які призвели до розробки SSL 3.0 версії.

Як тільки різні компанії (Microsoft) почали робити спроби розробки власних безпечних протоколів транспортування, IETF вирішило втрутитися і визначити стандарт протоколу шифрування. Згодом, за підтримки безлічі компаній, на підставі протоколу **SSL 3.0** було розроблено та прийнято стандарт RFC, який отримав ім'я **TLS 1.0**. Його також часто називають **SSL 3.1**.

Хоча TLS і SSL мають суттєві відмінності у реалізації, розробники зазвичай помічають лише деякі з них, а кінцеві користувачі їх зовсім не розрізняють. Проте **TLS 1.0 та SSL 3.0 несумісні**. Значна відмінність у тому, що **TLS вимагає певні алгоритми шифрування, які SSL не підтримує**. Таким чином, TLS сервер повинен "відкотитися" (downgrade) до SSL 3.0 для роботи з клієнтами, що використовують SSL 3.0.

З'єднання, захищене протоколом TLS, має одну або кілька наступних **властивостей**:

– Безпека: симетричне шифрування захищає передану інформацію від прочитання сторонніми особами.

– Автентифікація: "особистість" учасника з'єднання можна перевірити за допомогою асиметричного шифрування.

– Цілісність: кожне повідомлення містить код (**Message Authentication Code, MAC**), за допомогою якого можна перевірити, що дані не були змінені або втрачені в процесі передачі.

Так як більшість протоколів зв'язку можуть бути використані як з TLS/SSL, так і без нього, при встановленні з'єднання необхідно явно вказати серверу, чи клієнт встановлюватиме TLS. Один спосіб досягти цього – використовувати порт, яким з'єднання завжди встановлюється з допомогою TLS (наприклад, 443 для HTTPS). Інший спосіб – використовувати спеціальну команду серверу від клієнта переключити з'єднання на TLS (наприклад, STARTTLS для протоколів електронної пошти).

Версії SSL/TLS

SSL 1.0, 2.0 і 3.0

Протокол SSL був спочатку розроблений компанією Netscape Communications. Версія 1.0 ніколи не була оприлюднена. Версії 2.0 була випущена в лютому 1995 року, але містила багато недоліків безпеки, які призвели до розробки SSL версії 3.0. **SSL версії 3.0, випущений у 1996 році, послужив основою для створення протоколу TLS 1.0,** стандарту протоколу Internet Engineering Task Force (IETF), який вперше був визначений у RFC 2246 у січні 1999 року. Visa, Master Card, American Expressта багато інших організацій мають ліцензію на використання протоколу SSL для комерційних цілей у мережі Інтернет. Тим самим SSL розширюється відповідно до проекту про підтримку прямої та зворотної сумісності та переговорів між з'єднаннями в одноранговій мережі. З березня 2011 року, згідно з RFC 6176, TLS-клієнти не повинні використовувати протокол SSL 2.0 при запиті підключення до сервера, і сервери повинні відхиляти такі запити.

TLS 1.0

TLS 1.0 вперше був визначений в RFC 2246 у січні 1999 як оновлення версії SSL 3.0. Як зазначено в RFC, «відмінності між цим протоколом і SSL 3.0 не є критичними, але вони значні для появи несумісності при взаємодії TLS 1.0 і SSL 3.0». TLS 1.0 дійсно включає засоби, за допомогою яких реалізація підключення TLS до SSL 3.0 послабить безпеку.

TLS 1.1

TLS 1.1 презентували в RFC 4346 у квітні 2006 року. Це оновлення TLS версії 1.0. Значні зміни в цій версії включають:

– додано захист від атак, що використовують режим зчеплення блоків шифротексту (Cipher Block Chaining);

- неявний вектор ініціалізації (IV) був замінений на явний IV;
- було проведено зміну обробки помилок;

– введено підтримку IANA реєстрації параметрів.

TLS 1.2

TLS 1.2 був анонсований у RFC 5246 у серпні 2008 року. Він ґрунтується на раніше запропонованій версії TLS 1.1.

TLS 1.3

TLS 1.3 був визнаний стандартом у RFC 8446 у серпні 2018 року.

SSL

Надамо опис протоколу SSL**.**

SSL спочатку розроблений компанією Netscape для додавання протоколу – HTTPS до свого веб-браузера Netscape Navigator, тому що компанія вважала, що безпечне з'єднання між клієнтом і сервером в першу чергу послужить успіхом розвитку Інтернету як інструменту бізнесу.

Через неможливість гарантувати безпеку мережі, через яку передається інформація найкращим способом захистити її було вибрано шифрування та дешифрування на кінцях з'єднання, що встановлюється відповідно. Netscape могли б вбудувати цей підхід безпосередньо у свій браузер, але це не надало б єдиного рішення, яке інші програми могли б використовувати. Потрібно отримати більш загальний, незалежний від застосування підхід.

У результаті Netscape розробив протокол SSL, що працює поверх TCP, а також надає TCP-подібний інтерфейс для додатків вищого рівня. Теоретично, однією з переваг SSL для розробників була можливість замінити всі традиційні TCP виклики на нові SSL виклики. Специфічні деталі того, як SSL шифрує та дешифрує дані були відносно прозорі.

Опис протоколу SSL

Протокол SSL розміщується між двома протоколами: **протоколом, який використовує програма-клієнт** (HTTP, FTP, LDAP, TELNET тощо) і **транспортним протоколом TCP/IP.** SSL захищає дані, виступаючи як фільтр для обох сторін і передає їх далі на транспортний рівень.

Роботу протоколу можна розділити на два рівні:

– Шар протоколу підтвердження підключення (Handshake Protocol Layer).
– Шар протоколу запису.

Перший шар: протокол підтвердження підключення

Перший шар, у свою чергу, складається з трьох підпротоколів:

– Протокол підтвердження підключення (Handshake Protocol).
– Протокол зміни параметрів шифру (Cipher Spec Protocol).
– Попереджувальний протокол (Alert Protocol).

Протокол підтвердження підключення здійснює ланцюжок обміну даними, що в свою чергу починає автентифікацію сторін і погоджує шифрування, ґешування та стиснення. Наступний етап – автентифікація учасників, що здійснюється також протоколом підтвердження підключення.

Протокол зміни параметрів шифру використовується зміни даних ключа (keying material) – інформації, що використовується створення ключів шифрування. Протокол складається з одного повідомлення, в якому сервер говорить, що відправник хоче змінити набір ключів.

Попереджувальний протокол містить повідомлення, яке показує сторонам зміну статусу або повідомляє про можливу помилку. Зазвичай попередження надсилається тоді, коли підключення закрито та отримано неправильне повідомлення, повідомлення неможливо розшифрувати або користувач скасовує операцію.

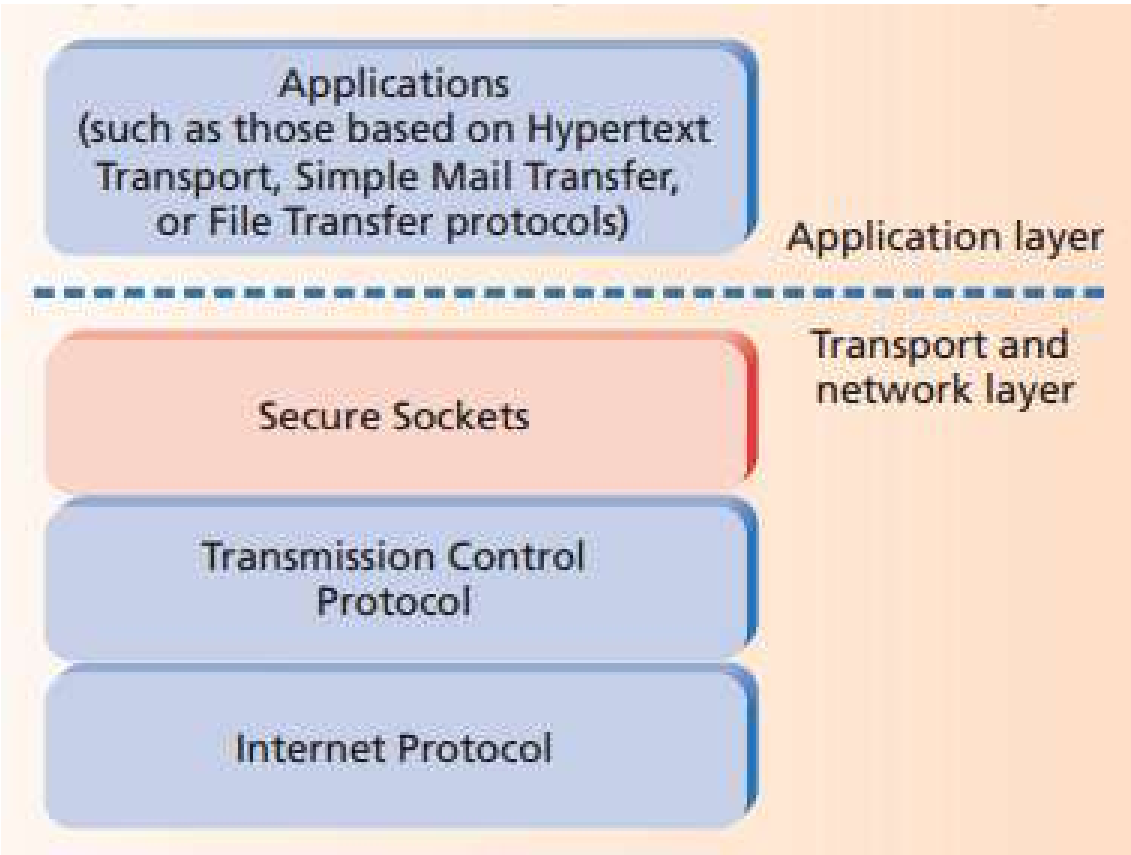

Рисунок 15.3 – Опис протоколу SSL

Другий шар: Протокол запису

Протокол запису (Record Layer) – це рівневий протокол. На кожному рівні повідомлення включають поля для довжини, опису та перевірки. Протокол запису приймає повідомлення, які потрібно передати, фрагментує дані в керовані блоки, розумно стискає дані, застосовуючи MAC (message authentication code), шифрує та передає результат. Отримані дані він розшифровує, перевіряє, розпаковує, збирає та доставляє до верхніх рівнів клієнта.

Існує чотири протоколи запису:

- Протокол рукостискання (handshake protocol).
- Протокол тривоги (alert protocol).
- Протокол зміни шифру (the change cipher spec protocol).
- Протокол програми (application data protocol);

Принцип роботи SSL

Принцип роботи SSL і двох фаз: **фаза рукостискання** і **фаза передачі**. Під час фази рукостискання клієнт та сервер використовують шифрування відкритим ключем для того, щоб визначити параметри секретного ключа, що використовується клієнтом та сервером для шифрування під час фази передачі даних.

Клієнт ініціює рукостискання посилаючи "hello"-повідомлення серверу. Таке повідомлення містить перелік алгоритмів симетричного шифрування (cipher specs), що підтримуються клієнтом. Сервер відповідає схожим

"hello"-повідомленням, вибравши при цьому найбільш підходящий алгоритм шифрування з отриманого списку. Далі сервер відправляє сертифікат, який містить його **публічний ключ**.

Сертифікат – це набір даних, що підтверджує справжність. Підтверджена третя сторона, відома як **центр сертифікації** (CA), генерує сертифікат та перевіряє його справжність. Щоб отримати сертифікат, сервер повинен використовувати безпечні канали для відправлення свого публічного ключа до центру сертифікації. Він генерує **сертифікат, який містить**:

– власний ID;

– ID сервера;

– публічний ключ сервера та іншу інформацію.

А **також центр сертифікації створює відбиток (digest) сертифікату**, який є контрольною сумою. Далі **центр сертифікації створює підпис сертифікату (certificate signature**), який формується шляхом шифрування відбитка сертифікату приватним ключем центру сертифікації.

Для **перевірки сертифікату сервера** клієнт використовує публічний ключ центру сертифікації для розшифровування підпису.

Потім клієнт самостійно зчитує відбиток сертифікату сервера та звіряє з розшифрованим. Якщо вони не збігаються, то сертифікат було підроблено. Природно, для розшифровки підпису клієнт повинен мати публічний ключ центру авторизації. Тому клієнт зберігає список публічних ключів підтверджених центрів сертифікації. За фактом, багато браузерних програм мають подібний список, що знаходиться безпосередньо в їх коді.

Коли клієнт встановив справжність сервера (сервер також може запросити сертифікат клієнта), сервер використовує шифрування відкритим ключем для визначення **секретного ключа** для обміну інформацією.

Фаза рукостискання завершується відправкою "finished"-повідомлень, як тільки обидві сторони готові розпочати використання секретного ключа. Починається фаза передачі даних, під час якої кожна сторона розбиває вихідні повідомлення фрагменти і прикріплює до них коди авторизації повідомлень **MAC** (message authentication code). **Код авторизації повідомлення** – це зашифрований відбиток, обчислений на основі вмісту повідомлень. З міркувань безпеки, він не збігається із секретним ключем та обчислюється разом із секретним ключем на стадії рукостискання. Для отримання повноцінного пакета SSL кожна зі сторін об'єднує дані фрагмента, код авторизації повідомлення, заголовки повідомлення і шифрують з використанням секретного. При отриманні пакета кожна зі сторін розшифровує його і звіряє отриманий код авторизації повідомлення зі своїм. Якщо вони не збігаються, то пакет було підроблено.

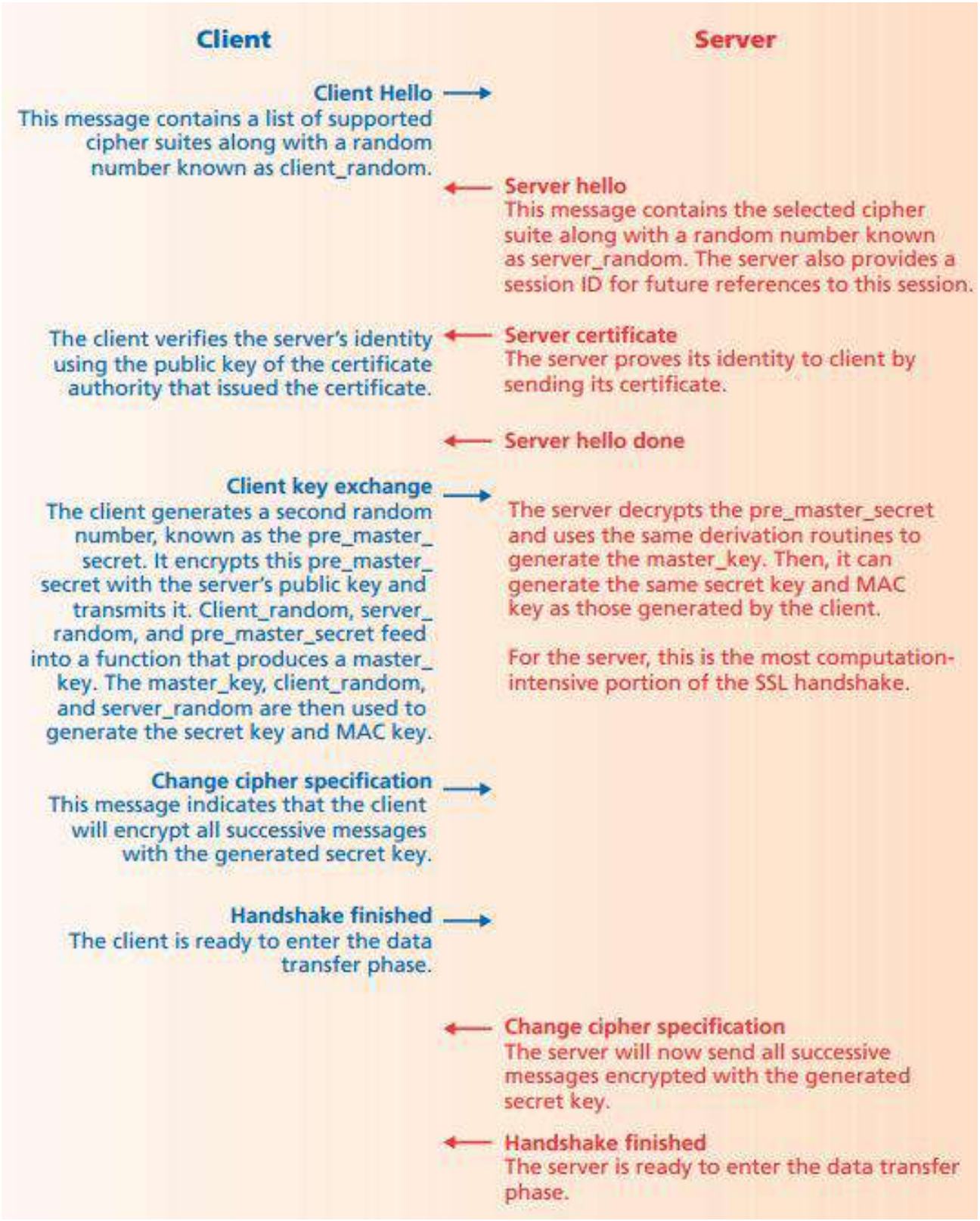

Рисунок 15.4 – Принцип роботи SSL

Цифрові сертифікати

Протокол SSL використовує сертифікати для перевірки з'єднання. Сертифікати розміщені на безпечному сервері та використовуються для шифрування даних та ідентифікації веб-сайту.

Способи отримання SSL-сертифікату:

– Використовувати сертифікат, виданий центром сертифікації (Certification authority).

– Використати самопідписаний сертифікат.

– Використовувати "порожній" сертифікат.

Самопідписаний сертифікат – сертифікат, створений самим користувачем, – у цьому випадку видавець сертифікату збігається з власником сертифікату.

«Порожній» сертифікат – сертифікат, що містить фіктивну інформацію, що використовується як тимчасова для налаштування SSL і перевірки його функціональності в даному середовищі.

Ґешування

Ґеш-значення є ідентифікатором повідомлення, його розмір менший за розмір оригінального повідомлення. Найвідомішими ґеш-алгоритмами є **MD5 (Message Digest 5)**, який створює 128-бітове ґеш-значення, **SHA-1 (Standard Hash Algorithm)**, що створює 160-бітове ґеш-значення, **SHA-2** та **SHA-3**. Результат роботи алгоритму ґешування – значення, яке використовується для перевірки цілісності передачі даних.

Шифрування

Існує два основні способи шифрування даних:

– симетричний ключ (загальний секретний ключ);

– асиметричний ключ (відкритий ключ).

Відкритий ключ

Суть асиметричного шифрування у тому, що використовується пара ключів. Один з них використовується як відкритий (як правило, він публікується в самому сертифікаті власника), другий ключ називається секретним – він тримається в таємниці і ніколи нікому не відкривається. Обидва ключі працюють у парі: один використовується для запуску протилежних функцій іншого ключа. Якщо відкритий ключ використовується для того, щоб зашифрувати дані, розшифрувати їх можна тільки секретним ключем і навпаки. Такий взаємозв'язок дозволяє робити дві важливі речі.

Будь-який користувач може отримати відкритий ключ за призначенням і використовувати його для шифрування даних, які може розшифрувати тільки користувач, який володіє секретним ключем. (RSA)

Якщо заголовок шифрує дані, використовуючи секретний ключ, кожен може розшифрувати дані, використовуючи відповідний відкритий ключ. Саме це є основою цифрових підписів. (DSA)

RSA – найпоширеніший алгоритм шифрування з використанням асиметричних ключів.

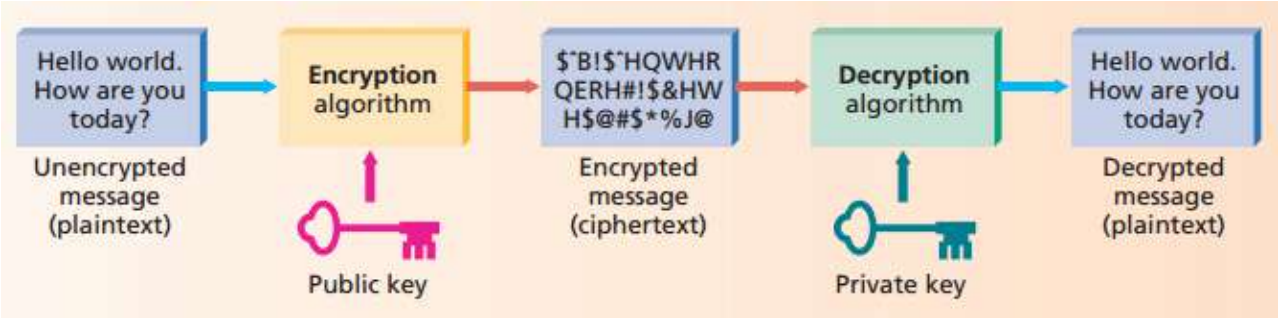

Рисунок 15.5 – Шифрування з використанням асиметричних (несиметричних, з відкритим ключем) алгоритмів

Секретний ключ

При шифруванні секретним ключем використовується той самий ключ для шифрованих даних. Якщо сторони хочуть обмінятися зашифрованими повідомленнями в безпечному режимі, то обидві сторони повинні мати однакові симетричні ключі. Такий тип шифрування використовується великого обсягу даних. Зазвичай використовуються алгоритми DES, 3-DES, RC2, RC4 та AES.

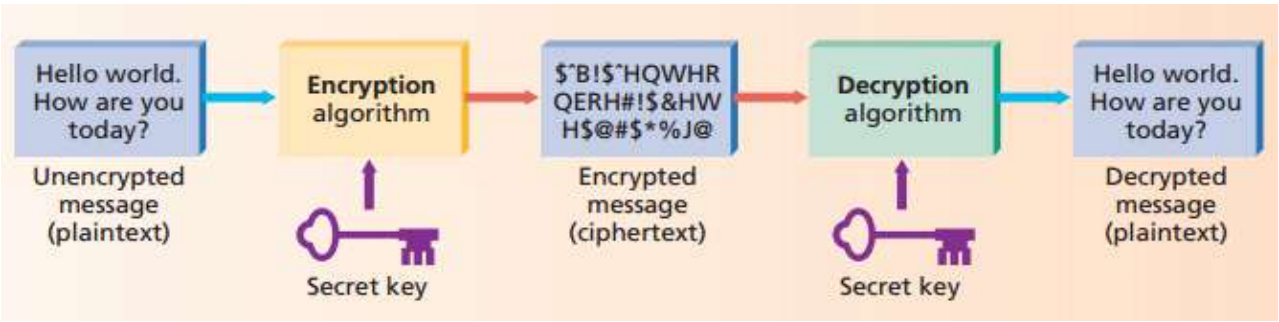

Рисунок 15.6 – Шифрування з використанням симетричних алгоритмів

Комбінований підхід

Алгоритми симетричного шифрування часто включають встановлену кількість добавок та зсувів. Такі алгоритми часто використовують ключ для допомоги при бітових маніпуляціях або для того, щоб дані, що шифруються, стали більш випадковими. Іншими словами, при збільшенні розміру секретного ключа може збільшитися випадковість даних, що шифруються, але не обов'язково зросте складність обчислень при розшифровці. Однак, шифрування відкритим ключем використовує ключ як експоненту, тому значне збільшення ключа сильно збільшує кількість обчислень, потрібних для шифрування/дешифрування даних.

Таким чином, хоча алгоритми шифрування відкритим ключем не стикаються з проблемою розподілу, з якою стикаються алгоритми шифрування секретним ключем, вони потребують значно більше обчислювальної потужності.

Для використання сильних сторін обох типів алгоритмів протоколи безпеки зазвичай використовують шифрування відкритим ключем передачі секретних ключів. Як тільки секретний ключ доставлено, починається передача даних з використанням симетричного шифрування.

Існує ще один підхід, який використовує відкритий ключ як угоду, а не спосіб доставки секретного ключа іншим. Обидві сторони обмінюються публічними ключами та незалежно генерують секретний ключ.

Найпоширенішою формою такої угоди є протокол **Діффі-Хеллмана**. Хоча SSL підтримує протокол Діффі-Хеллмана, більшість транзакцій SSL не використовують його. Натомість використовується алгоритм RSA, який вирішує проблему розподілу секретних ключів.

Застосування

При проектуванні програм **SSL** реалізується "під" будь-яким іншим протоколом прикладного рівня, таким як **HTTP, FTP, SMTP, NNTP і XMPP**, таким чином забезпечуючи "прозорість" його використання.

Історично SSL був використаний насамперед з надійними транспортними протоколами, такими як **Transmission Control Protocol (TCP)**. Проте він також був реалізований з датаграмними транспортними протоколами, такими як **User Datagram Protocol (UDP)** та **Datagram Control Protocol (DCCP)**, використання якого було стандартизовано, що призвело до появи терміну **Datagram Transport Layer Security (DTLS)**.

Веб-сайти

Часте використання протоколу SSL призвело до формування протоколу **HTTPS (Hypertext Transfer Protocol Secure),** який підтримує шифрування. Дані, які передаються за протоколом HTTPS, «упаковуються» в криптографічний протокол SSL або TLS, забезпечуючи тим самим захист цих даних. Такий спосіб захисту широко використовується у світі Веб для додатків, у яких важлива безпека з'єднання, наприклад, у платіжних системах. На відміну від HTTP, для HTTPS за замовчуванням використовується TCP-порт 443.

Використання та реалізація

Спочатку віртуальні приватні мережі (VPN) на основі SSL розроблялися як додаткова та альтернативна технологія віддаленого доступу на основі IPsec VPN. Однак такі фактори, як достатня надійність та дешевизна, зробили цю технологію привабливою для організації VPN. Також SSL набув широкого застосування в електронній пошті.

Найбільш поширена реалізація SSL – криптографічний пакет з відкритим вихідним кодом OpenSSL, заснований на **SSLeay**, написаної Еріком Янгом. Остання версія OpenSSL підтримує SSLv3. Пакет призначений для створення та управління різного роду сертифікатами. Також до його складу входить бібліотека підтримки SSL різними програмами. Бібліотека використовується, наприклад, модулем SSL у поширеному HTTP-сервері Apache.

Специфікація протоколу записів SSL

Формат заголовка записів SSL

Всі дані в SSL пересилаються у вигляді записів (рекордів) – об'єктів, які складаються із заголовка та деякої кількості даних. Кожен заголовок рекорду містить 2 або 3 байти коду довжини. Якщо старший біт у першому байті коду довжини рекорду дорівнює 1, тоді рекорд немає заповнювача і повна довжина заголовка дорівнює 2 байтам, інакше рекорд містить заповнювач, і повна довжина заголовка дорівнює 3 байтам. У разі довгого (3 байти) заголовка другий за старшинством біт першого байти має спеціальне значення. Якщо він дорівнює 0 – рекорд є інформаційним, якщо він дорівнює 1 – рекорд є security escape. Код довжини рекорду не включає число байт заголовка. Для 2-байтового заголовка його довжина обчислюється так:

RECORD-LENGTH = ((byte[0] & 0x7F) << 8) | byte[1];

Тут byte[0] перший отриманий байт, а byte[1] другий отриманий байт.

Для 3-байтового заголовка довжина рекорду обчислюється так:

RECORD-LENGTH = ((byte[0] & 0x3F) <<8) | byte[1];

IS-ESCAPE = (byte[0] & 0x40)! = 0;

PADDING = byte [2];

Значення **PADDING** специфікує кількість байтів, доданих відправником до вихідного рекорду. Дані заповнювача використовуються для того, щоб зробити довжину рекорду кратною розміру блоку шифру. Відправник додає **PADDING** після наявних даних, а потім шифрує все це, так як довжина цього масиву кратна розміру блоку використовуваного шифру. Оскільки відомий обсяг даних, що передаються, заголовок повідомлення може бути сформований з урахуванням обсягу **PADDING**. Отримувач повідомлення дешифрує все поле даних і отримує вихідну інформацію, потім обчислює справжнє значення **RECORD-LENGTH**, при цьому **PADDING** з поля дані видаляється.

Формат інформаційних записів SSL

Частина даних рекорду SSL складається з трьох компонентів:

1. **MAC-DATA[MAC-SIZE]**
2. **ACTUAL-DATA[N]**
3. **PADDING-DATA[PADDING]**

Де:

– **MAC-DATA** – код автентифікації повідомлення;

– **MAC-SIZE** – функція алгоритму обчислення ґеш-суми, що використовується;

– **ACTUAL-DATA** – реально передані дані або поле даних повідомлення;

– **PADDING-DATA** – дані **PADDING** (при блоковому шифруванні);

MAC-DATA = **HASH** [**SECRET**, **ACTUAL-DATA**, **PADDING-DATA**, **SEQUENCE-NUMBER**]

Тут **SECRET** передається ґеш-функції першим, потім слідує **ACTUAL-DATA** і **PADDING-DATA**, за якими передається **SEQUENCE-NUMBER** – порядковий номер.

Значення **SECRET** залежить від того, хто надсилає повідомлення. Якщо це робить клієнт, **SECRET** дорівнює **CLIENT-WRITE-KEY**. Якщо клієнт отримує повідомлення, **SECRET** дорівнює **CLIENT-READ-KEY**.

Порядковий номер є 32-бітовим кодом, який передається ґеш-функції у вигляді 4 байт, використовуючи мережевий порядок передачі «від старшого до молодшого». Порядковий номер – лічильник для сервера чи клієнта. Для кожного напряму передачі використовується пара лічильників – для відправника та для одержувача; щоразу, коли надсилається повідомлення, лічильник збільшує своє значення на 1.

Отримувач повідомлення використовує очікуване значення порядкового номера передачі **MAC** (тип ґеш-функції визначається параметром **CIPHER-CHOICE**). Обчислене значення **MAC-DATA** має збігатися з переданим значенням. Якщо порівняння не пройшло, повідомлення вважається пошкодженим, що призводить до виникнення помилки, яка спричиняє закриття з'єднання.

Остаточна перевірка відповідності виконується, коли використовується блоковий шифр. Обсяг даних у повідомленні (**RECORD-LENGTH**) повинен бути кратний розміру блоку шифру. Якщо ця умова не виконана, повідомлення вважається пошкодженим, що призводить до розриву з'єднання.

Для 2-байтового заголовка максимальна довжина повідомлення дорівнює 32767 байтів, для 3-байтового 16383 байтів. Повідомлення протоколу діалогу SSL повинні відповідати одиночним рекордам протоколу SSL, а повідомлення прикладного протоколу можуть займати кілька рекордів SSL.

Протокол діалогу SSL

Протокол діалогу SSL містить дві основні фази.

Фаза 1

Перша фаза використовується встановлення конфіденційного каналу комунікацій. Ця фаза ініціалізує з'єднання, коли обидва партнери обмінюються повідомленнями "hello". Клієнт надсилає повідомлення **CLIENT-HELLO**. Сервер отримує це повідомлення, обробляє його та посилає у відповідь повідомлення **SERVER-HELLO**.

У цей момент сервер і клієнт мають достатньо інформації, щоб знати, чи потрібен новий **master key.** Якщо ключ не потрібен, сервер і клієнт переходять у фазу 2.

Коли виникає необхідність створення нового **master key**, повідомлення сервера **SERVER-HELLO** вже містить достатньо даних для того, щоб клієнт міг згенерувати **master key**. Ці дані включають підписаний сертифікат сервера, список базових шифрів та ідентифікатор з'єднання (випадкове число, згенероване сервером, яке використовується протягом усієї сесії). Після генерації клієнтом **master key** він посилає серверу повідомлення **CLIENT-MASTER-KEY** або повідомлення про помилку, коли клієнт і сервер не можуть узгодити базовий шифр.

Після визначення **master key** сервер надсилає клієнту повідомлення **SERVER-VERIFY**, яке автентифікує сервер.

Фаза 2

Фаза 2 називається фазою автентифікації. Оскільки сервер вже автентифіковано першої фазі, то другої фазі здійснюється автентифікація клієнта. Сервер надсилає запит клієнту, і якщо клієнт має необхідну інформацію – він надсилає позитивний відгук, якщо ж ні – повідомлення про помилку. Коли один партнер виконав автентифікацію іншого партнера, він надсилає повідомлення **finished**. У разі клієнта повідомлення **CLIENT-FINISHED** містить зашифровану форму ідентифікатора **CONNECTION-ID**, яку повинен верифікувати сервер. Якщо вірифікація була невдалою, сервер надсилає повідомлення **ERROR**.

Коли один із партнерів надіслав повідомлення **finished –** він повинен приймати повідомлення до тих пір, поки не отримає повідомлення **finished** від іншого партнера, і тільки коли обидва партнери надіслали та отримали повідомлення **finished**, протокол діалогу SSL закінчить свою роботу. З цього моменту починає роботу прикладний протокол.

Типовий протокол обміну повідомленнями

Нижче наведено кілька варіантів обміну повідомленнями в рамках протоколу діалогу SSL. Клієнт – **C**, сервер – **S.** "{smth}key" означає що "smth" зашифровано за допомогою ключа.

Таблиця 11.1 – За відсутності ідентифікатора сесії

Client-hello	C ® S:	challenge, cipher_specs
Server-hello	S ® C:	connection-id,server_certificate,cipher_specs
Client-master-key	C ® S:	{master_key}server_public_key
Client-finish	C ® S:	{connection-id}client_write_key
Server-verify	S ® C:	{challenge}server_write_key
Server-finish	S ® C:	{new_session_id}server_write_key

Таблиця 11.2 – Ідентифікатор сесії знайдено клієнтом та сервером

Client-hello	C ® S:	challenge, session_id, cipher_specs
Server-hello	S ® C:	connection-id, session_id_hit
Client-finish	C ® S:	{connection-id}client_write_key
Server-verify	S ® C:	{challenge}server_write_key
Server-finish	S ® C:	{session_id}server_write_key

Таблиця 11.3 – Використаний ідентифікатор сесії та автентифікація клієнта

Client-hello	C ® S:	challenge, session_id, cipher_specs
Server-hello	S ® C:	connection-id, session_id_hit
Client-finish	C ® S:	{connection-id}client_write_key
Server-verify	S ® C:	{challenge}server_write_key
Request-certificate	S ® C:	{auth_type,challenge'}server_write_key
Client-certificate	C ® S:	{cert_type,client_cert, response_data}client_write_key

Server-finish	S ® C:	{new_session_id}server_write_key

Автентифікація та обмін ключами

SSL підтримує 3 типи автентифікації:

–автентифікація обох сторін (клієнт-сервер),

–автентифікація сервера з неавтентифікованим клієнтом,

–повна анонімність.

Зазвичай для автентифікації використовуються алгоритми: **RSA, DSA, ECDSA.**

Якщо сервер автентифікований, його повідомлення про сертифікацію має забезпечити правильний сертифікаційний ланцюжок, що веде до прийнятного центру сертифікації. Простіше кажучи, автентифікований сервер має надати допустимий сертифікат клієнту. Кожна сторона відповідає за перевірку того, що сертифікат іншої сторони ще не закінчився і не було скасовано. Щоразу, коли сервер автентифікується, канал стійкий (безпечний) до спроби перехоплення даних між веб-сервером і браузером, але повністю анонімна сесія за своєю суттю вразлива до такої атаки. Анонімний сервер не може автентифікувати клієнта.

Головна мета процесу обміну ключами – створення секрету клієнта (pre_master_secret), відомого тільки клієнту і серверу.

Секрет (pre_master_secret) використовується для створення загального секрету (master_secret).

Загальний секрет необхідний для того, щоб створити повідомлення для перевірки сертифікату, ключів шифрування, секрету MAC (message authentication code) та повідомлення "finished".

Відсилаючи повідомлення "finished", сторони вказують, що вони знають вірний секрет (pre_master_secret).

Анонімний обмін ключами

Повністю анонімна сесія може бути встановлена під час використання алгоритму RSA або Діффі-Хеллмана для створення ключів обміну. У разі використання RSA клієнт шифрує секрет (pre_master_secret) за допомогою відкритого ключа несертифікованого сервера. Відкритий ключ клієнт дізнається із повідомлення обміну ключами від сервера. Результат надсилається у повідомленні обміну ключами від клієнта. Оскільки перехоплювач не знає закритого ключа сервера, йому неможливо розшифрувати секрет (pre_master_secret). У разі використання алгоритму Діффі-Хеллмана відкриті параметри сервера містяться в повідомленні обміну ключами від сервера, і клієнту надсилають у повідомленні обміну ключами. Перехоплювач, який знає приватних значень, зможе знайти секрет (pre_master_secret).

Автентифікація та обмін ключами під час використання RSA

У цьому випадку обмін ключами та автентифікація сервера може бути скомбінована. Відкритий ключ також може міститись у сертифікаті сервера або може бути використаний тимчасовий ключ RSA, який надсилається у повідомленні обміну ключами від сервера. Коли використовується тимчасовий ключ RSA, повідомлення обміну підписуються сервером RSA або сертифікат

DSS. Сигнатура містить поточне значення повідомлення Client_Hello.random, таким чином, старі сигнатури та старі часові ключі не можуть повторюватися. Сервер може використовувати тимчасовий ключ RSA лише один раз для створення сесії. Після перевірки сертифікату сервера клієнт шифрує секрет (pre_master_secret) з допомогою відкритого ключа сервера. Після успішного декодування секрету (pre_master_secret) створюється повідомлення "finished", тим самим сервер демонструє, що він знає приватний ключ, який відповідає сертифікату сервера.

Коли RSA використовується для обміну ключами, для автентифікації клієнта використовується повідомлення для перевірки сертифікату клієнта. Клієнт підписується значенням, обчисленим з master_secret та всіх попередніх повідомлень протоколу рукостискання. Ці повідомлення рукостискання містять сертифікат сервера, який ставить у відповідність сигнатурі сервера повідомлення Server_Hello.random, якому ставить у відповідність сигнатуру поточному повідомленню рукостискання.

Автентифікація та обмін ключами під час використання протоколу Діффі-Хеллмана

У цьому випадку сервер може також підтримувати алгоритм Діффі-Хеллмана, що містить конкретні параметри, або може використовувати повідомлення обміну ключами від сервера для посилення набору тимчасових параметрів, підписаних сертифікатами DSS або RSA. Тимчасові параметри ґешуються з повідомленням hello.random перед підписанням, щоб зловмисник не зміг зробити повторення старих параметрів. У будь-якому випадку, клієнт може перевірити сертифікат або сигнатуру для впевненості, що параметри належать серверу.

Якщо клієнт має сертифікат, що містить параметри алгоритму Diffie-Hellman, то сертифікат також містить інформацію, потрібну для того, щоб завершити обмін ключами. У цьому випадку клієнт і сервер повинні будуть згенерувати однакові Diffie-Hellman результати (pre_master_secret) щоразу, коли вони встановлюють з'єднання. Для того, щоб запобігти збереженню секрету (pre_master_secret) в пам'яті комп'ютера на час довше, ніж необхідно, секрет повинен бути переведений в загальний секрет (master_secret) настільки швидко, наскільки це можливо. Параметри клієнта повинні бути сумісні з тими, що підтримує сервер для того, щоб працював обмін ключами.

Відновлення сесії

Творці SSL знали, що алгоритми шифрування відкритим ключем обчислювально складні, і клієнт, що створює кілька нових з'єднань до одного і того ж сервера протягом короткого проміжку часу може навантажити сервер, що призведе до помітних тимчасових затримок відповіді. Однак, якщо клієнт і сервер вже встановили з'єднання, йому відповідатиме унікальний ідентифікатор сесії, що дозволяє посилатися на нього і використовувати такий же секретний ключ при наступних з'єднаннях в рамках деякого часового відрізка. Безумовно, такий підхід привносить певний ризик у безпеку з'єднання. Тому, якщо

необхідно, клієнт може створити новий ідентифікатор і секретний ключ для даної сесії. Microsoft Internet Explorer, наприклад, роблять цю операцію кожні 2 хвилини.

Рисунок 15.7 – Відновлення сесії

Адміністрація

Обслуговування сертифікатів та ключів

Якщо клієнт планує звернутися до сервера, який вимагає клієнтської автентифікації, він повинен зберігати свій сертифікат та пов'язаний із ним приватний ключ. Сервер повинен зберігати свій сертифікат і пов'язаний з ним приватний ключ.

Сховище ідентифікаторів сесій

Клієнт та сервер зобов'язані зберігати ідентифікатори сесій та пов'язані з ними секретні ключі для використання під час відновлення з'єднання.

Протокол TLS

Принцип роботи TLS

Протокол TLS поділяється на два шари:

– TLS Record.

– TLS Handshake.

Підтвердження зв'язку (TLS Handshake)

1. Клієнт надсилає повідомлення **ClientHello**, що вказує версію SSL або TLS і методи шифрування, що підтримуються клієнтом (англ. CipherSuite). Це повідомлення містить випадкове число (набір байт), яке використовується в наступних обчисленнях. Протокол також дозволяє вказати методи стиснення даних, що підтримуються клієнтом.
2. Сервер відповідає повідомленням **ServerHello**, яке містить метод шифрування, вибраний сервером зі списку, запропонованого клієнтом, а також ідентифікатор сесії та ще одне випадкове число. Також сервер надсилає свій цифровий сертифікат. Якщо сервер потрібен сертифікат для автентифікації клієнта, на цьому кроці він може надіслати клієнту запит такого сертифікату.
3. Клієнт перевіряє сертифікат сервера.
4. Клієнт відправляє випадкове число, яке клієнт та сервер використовують для шифрування наступних повідомлень. Сам рядок із байт шифрується публічним ключем сервера.
5. Якщо сервер вимагає від клієнта сертифікат, клієнт надсилає набір байт, зашифрований його секретним ключем, і свій цифровий сертифікат, або оповіщення про відсутність сертифікату.
6. Сервер перевіряє сертифікат клієнта.
7. Клієнт та сервер надсилають одне одному повідомлення **ChangeCipherSpec**, оголошуючи про зміну режиму передачі даних із незахищеного на захищений.
8. Клієнт відправляє повідомлення **Finished**, зашифроване секретним ключем, і таким чином завершує підтвердження зв'язку зі свого боку.
9. Аналогічні дії здійснює сервер.
10. Протягом цієї сесії клієнт та сервер можуть обмінюватися повідомленнями, зашифрованими секретним ключем.

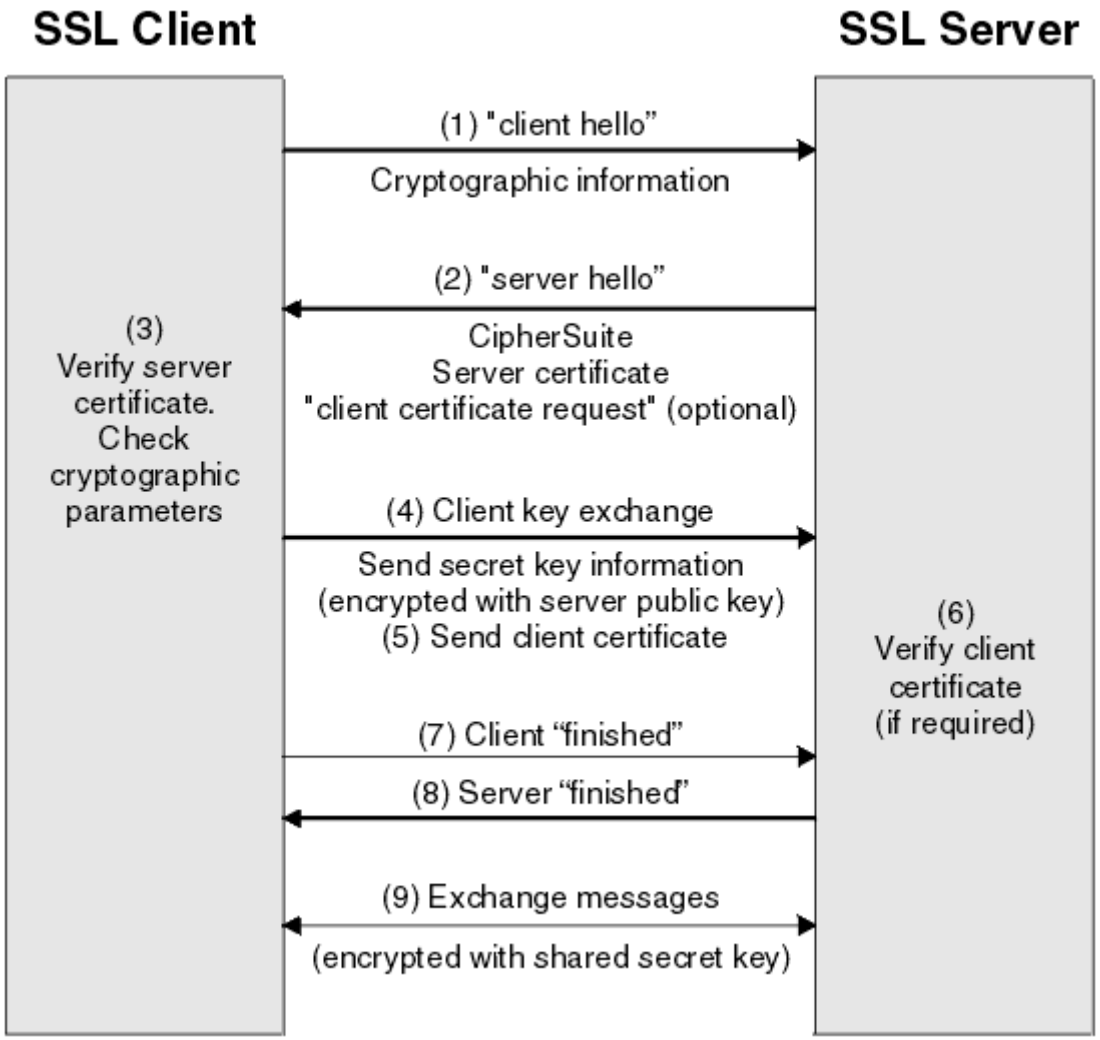

Рисунок 15.8 – Підтвердження зв'язку (handshake)

Відновлення сесії

1. Клієнт надсилає повідомлення **ClientHello за** допомогою ID сесії, яку потрібно відновити.

2. Сервер перевіряє, чи має він у кеші відповідний ідентифікатор. Якщо є і сервер здатний відновити сесію, він надсилає клієнтові повідомлення **ServerHello** з тим самим ID сесії. Якщо ні, сервер генерує новий ID сесії та виконує процедуру handshake з клієнтом.

3. Клієнт і сервер обмінюються повідомленнями **ChangeCipherSpec**, а потім **Finished**.

4. Передача даних захищеним каналом відновлюється.

Протокол запису (TLS Record)

Цей шар захищає дані за допомогою ключів, отриманих при підтвердженні зв'язку, та перевіряє цілісність та джерело вхідних повідомлень. Він виконує такі функції:

– Розбиття вихідних повідомлень на блоки потрібного розміру та "склеювання" вхідних повідомлень.

– Стиснення вихідних повідомлень та розпакування вхідних (використовується не завжди).

– Застосування коду автентифікації до вихідних повідомлень та перевірку вхідних за допомогою MAC.

– Шифрування вихідних повідомлень та дешифрування вхідних повідомлень.

Після обробки протоколом TLS Record зашифровані дані передаються на TCP для передачі.

Склад запису

– Content type: тип повідомлення – підтвердження зв'язку (22), звичайне повідомлення (23) або оповіщення (21).

– Version: версія SSL/TLS.

– Length: довжина частини повідомлення, що залишилася.

– Payload: власне зашифровані дані.

– MAC: код автентифікації.

– Padding: "відступ" для отримання потрібного розміру повідомлення.

Byte	+0	+1	+2	+3
0	Content type			
1..4	Version		Length	
5..n	*Payload*			
n..m	MAC			
m..p	Padding (block ciphers only)			

Рисунок 15.9 – Склад запису

Цифрові сертифікати (стандарт X.509)

– Зручний спосіб показати, що хтось володіє публічним ключем.

– Випускаються центрами сертифікації (**Certificate Authority, CA**): GlobalSign, Comodo та ін.

– **PKI (public key infrastructure)** – механізм, що регулює поширення та використання сертифікатів (включаючи створення, відгук та автентифікацію).

– Список довірених CA підтримується програмою (у браузерів свої списки).

– Сертифікати підписуються іншими сертифікатами, що підвищує надійність.

– Сертифікат може бути відкликаний. Система підтримує список таких сертифікатів (**Certificate Revocation List, CRL**). На стороні CA список оновлюється кожні кілька годин.

Отримання сертифікату

1. Користувач генерує ключ і надсилає запит серверу CA.
2. CA відповідає повідомленням зі своїм сертифікатом.
3. Користувач збирає дані, необхідні для видавання сертифікату (email, відбиток ключа тощо).
4. Користувач надсилає дані до CA, зашифрувавши їх публічним ключем CA.
5. CA перевіряє отримані дані та надсилає сертифікат користувачу.

Структура сертифікату

– Власне сертифікат:
  - Версія.
  - Серійний номер.
  - Емітент (той, хто випустив сертифікат).
  - Суб'єкт.
  - Публічний ключ суб'єкта.
  - Період дії.
  - Додаткові поля.
– Алгоритм підпису сертифікату.
– Значення підпису сертифікату.

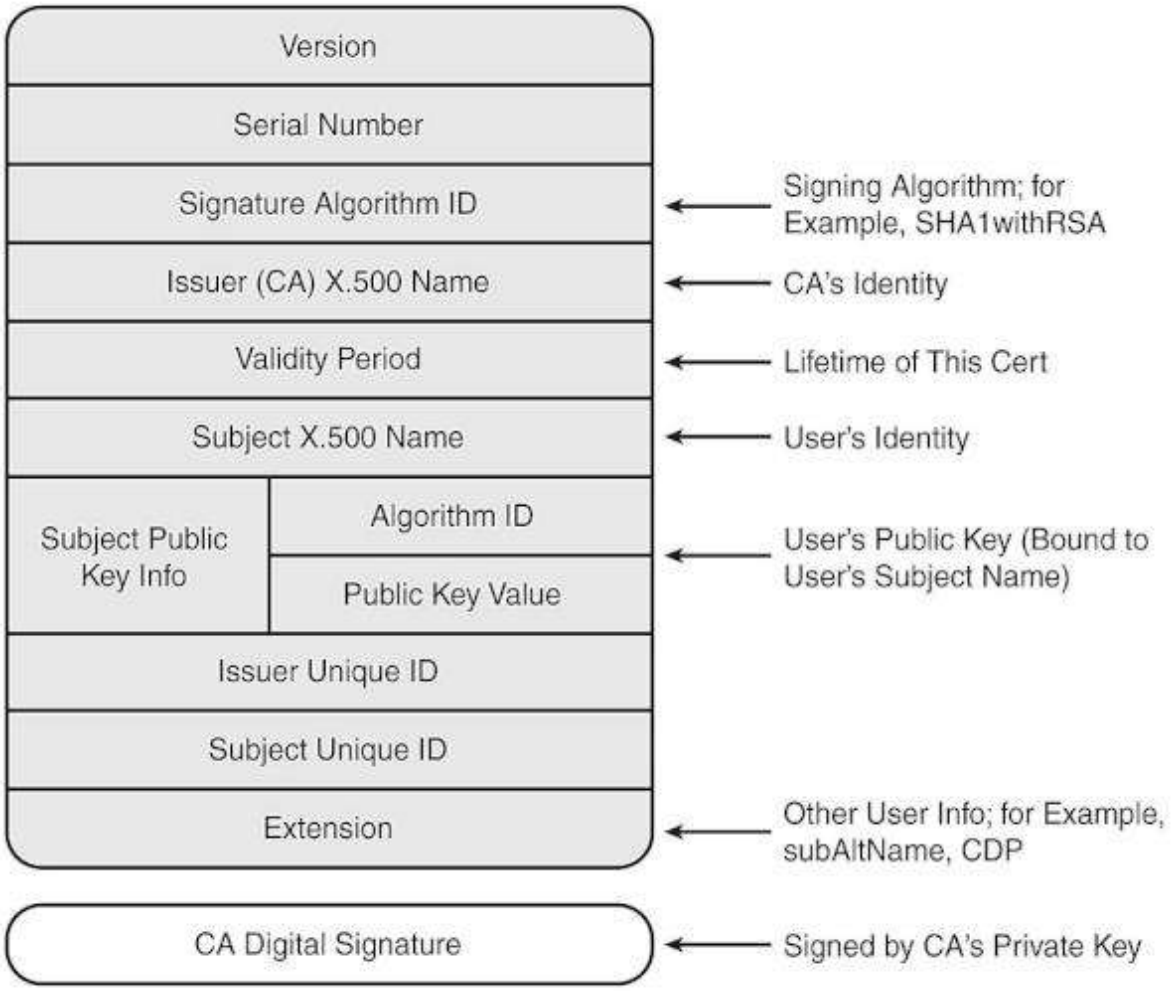

Рисунок 15.10 – Структура сертифікату X.509

Заходи безпеки в TLS

– Захист від downgrade-атаки – зниження версії протоколу до попередньої (менш захищеної) версії або менш надійного алгоритму шифрування.

– Нумерація послідовних записів програми та використання порядкового номера в коді автентифікації повідомлення (MAC).

– Використання ключа в ідентифікаторі повідомлення (тільки власник ключа може створити код автентифікації повідомлення).

– Повідомлення, яким закінчується підтвердження зв'язку («Finished»), містить ґеш всіх handshake-повідомлень, надісланих обома сторонами, що дозволяє перевірити автентичність вибраних параметрів TLS-з'єднання.

– Псевдовипадкова функція ділить подані їй на вхід дані навпіл, застосовує до половин різні ґеш-алгоритми (MD5 і SHA-1), а потім XOR'іт результати для отримання MAC. Це підвищує безпеку у випадку, якщо в одному з алгоритмів виявиться вразливість.

Ключові відмінності SSL та TLS

– Автентифікація повідомлень: в TLS використовується HMAC, що працює з будь-якою ґеш-функцією (а не тільки з MD5 або SHA, як у SSL).

– Генерація ключа: в TLS під час створення ключа використовується псевдовипадкова функція стандарту HMAC; в SSL – RSA, Diffie-Hellman або Fortezza / DMS.

– Перевірка сертифікату: SSL перевірка вимагає передачі складної послідовності повідомлень; у TLS інформація про перевірку повністю передається у повідомленнях під час handshake.

– Методи шифрування: SSL підтримує лише алгоритми RSA, Diffie-Hellman та Fortezza/DMS. У TLS відмовилися від підтримки Fortezza/DMS, але можливе додавання нових методів шифрування в наступних версіях.

Види можливих атак

– Атака за словником.
– Атака відображенням.
– Атака протоколу рукостискання.
– Зламування SSL-з'єднань усередині ЦОД.
– BEAST атака.
– Розкриття шифрів.
– Атака «зловмисник посередині».
– THC-SSL-DOS.
– SSLstrip.