Secure HTTP

در سه فصل قبل ویژگی‌های HTTP مورد بررسی قرار گرفت که به شناسایی و احراز هویت کاربران کمک می‌کند. این تکنیک‌ها در یک جامعه دوستانه به خوبی کار می‌کنند، اما به اندازه کافی قوی نیستند تا از تراکنش‌های مهم در برابر جامعه متشکل از دشمنان با انگیزه و متخاصم محافظت کنند.

این فصل یک فناوری پیچیده‌تر و تهاجمی‌تر را برای ایمن کردن تراکنش‌های HTTP از شنود و دستکاری با استفاده از رمزنگاری دیجیتال ارائه می‌کند.

Making HTTP Safe

مردم از تراکنش‌های وب برای کارهای جدی استفاده می‌کنند. بدون امنیت قوی، آن‌ها در انجام خرید آنلاین و بانکداری احساس راحتی نخواهند کرد. شرکت‌ها نمی توانند بدون اینکه بتوانند دسترسی را محدود کنند، اسناد مهم را روی سرورهای وب قرار دهند. وب به فرم امن HTTP نیاز دارد.

فصل‌های قبلی در مورد برخی از روش‌های سبک برای ارائه احراز هویت (تأیید هویت Basic و Digest) و یکپارچگی پیام (digest qop = auth-int) صحبت کردند. این طرح‌ها برای بسیاری از اهداف خوب هستند، اما ممکن است برای خریدهای بزرگ، تراکنش‌های بانکی یا دسترسی به داده‌های محرمانه به اندازه کافی قوی نباشند. برای این تراکنش‌های جدی‌تر، HTTP را با فناوری رمزگذاری دیجیتال ترکیب می‌کنیم.

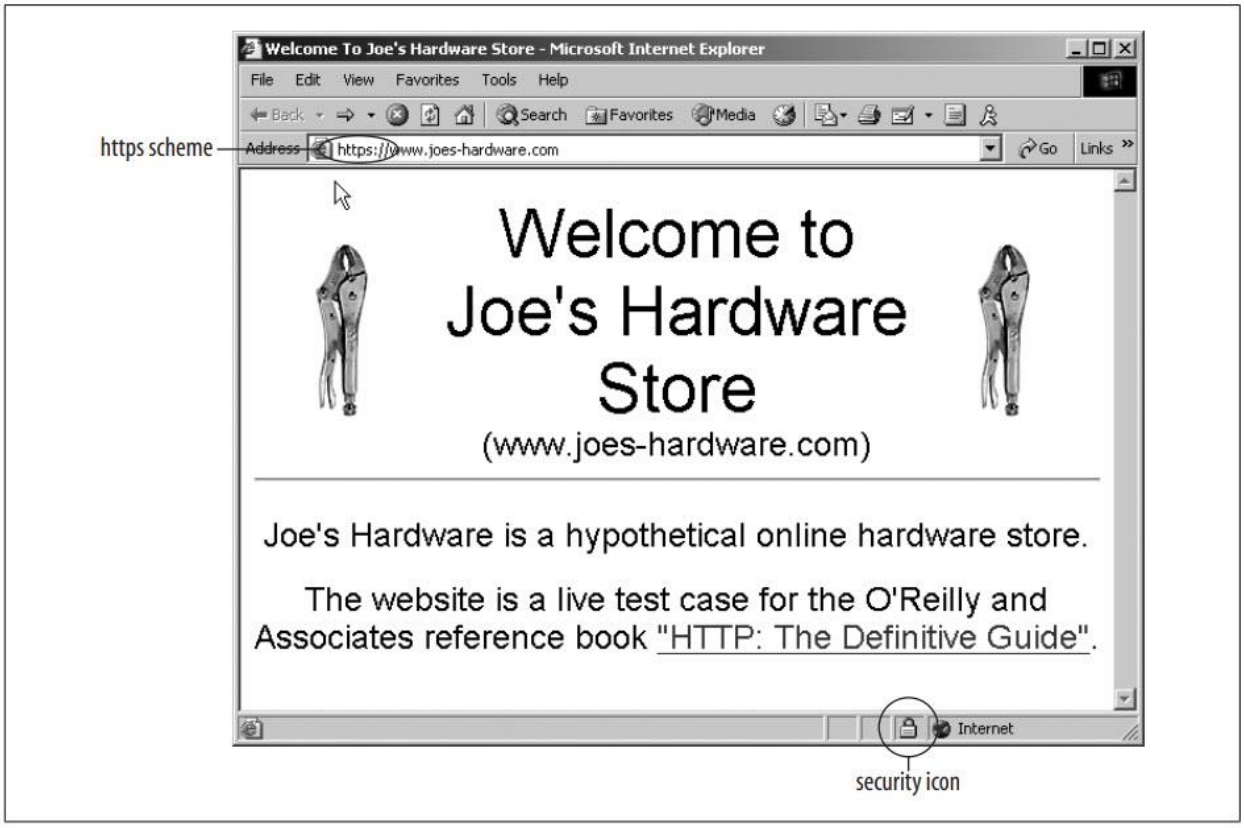
نسخه ایمن HTTP باید کارآمد، قابل حمل، دارای مدیریت آسان و سازگار با دنیای در حال تغییر باشد. همچنین باید الزامات اجتماعی و دولتی را برآورده کند. ما به یک فناوری برای امنیت HTTP نیاز داریم که موارد زیر را ارائه دهد:

- احراز هویت سرور (کاربران می‌دانند که با سرور واقعی صحبت می‌کنند، نه ساختگی)
- احراز هویت کلاینت (سرورها می‌دانند که با کاربر واقعی صحبت می‌کنند، نه جعلی)
- یکپارچگی (کلاینت‌ها و سرورها از تغییر داده‌هایشان در امان هستند)
- رمزگذاری (کلاینت‌ها و سرورها به صورت خصوصی بدون ترس از استراق سمع صحبت می‌کنند)
- کارایی (الگوریتمی به اندازه کافی سریع برای استفاده از کلاینت‌ها و سرورهای ارزان قیمت)
- Ubiquity (پروتکل‌ها تقریباً توسط همه کلاینت‌ها و سرورها پشتیبانی می‌شوند)
- مقیاس پذیری اداری (ارتباط امن فوری برای هر کسی، در هر مکان)
- سازگاری (از بهترین روش‌های امنیتی شناخته شده روز پشتیبانی می‌کند)
- بقای اجتماعی (برآورنده نیازهای فرهنگی و سیاسی جامعه)

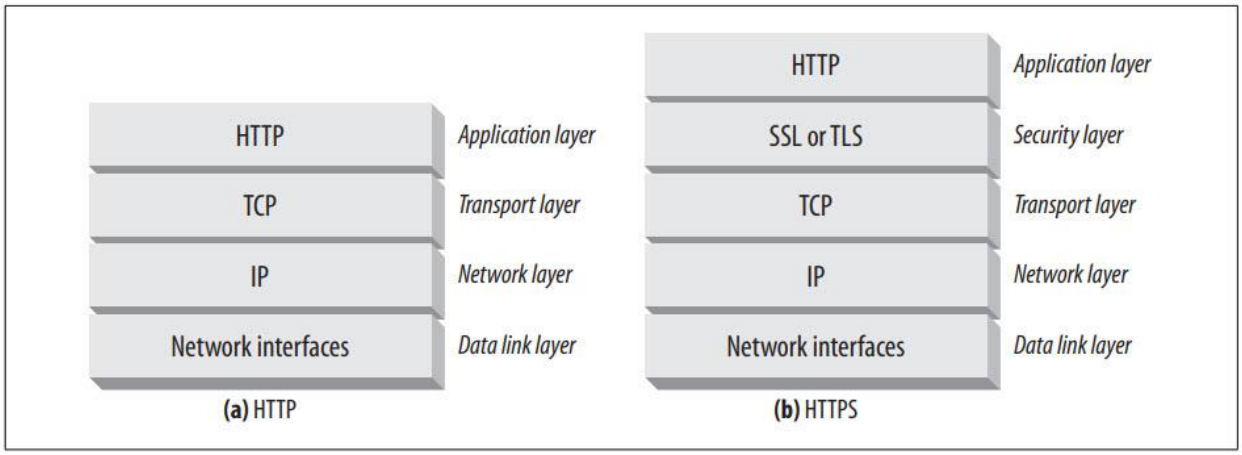
HTTPS

HTTPS محبوب‌ترین شکل امن HTTP است. این توسط Netscape Communications Corporation پیشگام بود و توسط تمام مرورگرها و سرورهای اصلی پشتیبانی می‌شود.

می‌توانید تشخیص دهید که آیا یک صفحه وب از طریق HTTPS به جای HTTP دسترسی داشته است، زیرا URL به جای http:// با طرح https:// شروع می‌شود (برخی از مرورگرها نیز نشانه‌های امنیتی نمادین را نشان می‌دهند، همانطور که در شکل زیر نشان داده شده است).



هنگام استفاده از HTTPS، تمام داده‌های درخواست و پاسخ HTTP قبل از ارسال در سراسر شبکه رمزگذاری می‌شوند. HTTPS با ارائه یک لایه امنیتی رمزنگاری در سطح انتقال - با استفاده از SSL یا جانشین آن، TLS - در زیر HTTP (شکل زیر) کار می‌کند. از آنجایی که SSL و TLS بسیار شبیه هستند، در این کتاب ما از اصطلاح "SSL" برای نشان دادن SSL و TLS استفاده می‌کنیم.



از آنجایی که بیشتر کارهای سخت رمزگذاری و رمزگشایی در کتابخانه‌های SSL انجام می‌شود، کلاینت‌ها و سرورهای وب برای استفاده از HTTP ایمن نیازی به تغییر منطق پردازش پروتکل خود ندارند.

در بیشتر موارد، آن‌ها به سادگی نیاز دارند که تماس‌های ورودی/خروجی TCP را با تماس‌های SSL جایگزین کنند و چند تماس دیگر را برای پیکربندی و مدیریت اطلاعات امنیتی اضافه کنند.

Digital Cryptography

قبل از اینکه به تفصیل در مورد HTTPS صحبت کنیم، باید پیشینه کمی در مورد تکنیک‌های رمزگذاری رمزنگاری شده مورد استفاده توسط SSL و HTTPS ارائه دهیم. در چند بخش بعدی، ما یک آغازگر سریع از ملزومات رمزنگاری دیجیتال ارائه خواهیم داد. اگر قبلاً با فن آوری و اصطلاحات رمزنگاری دیجیتال آشنا هستید، به راحتی به بخش «HTTPS: The Details» بروید.

Ciphers

الگوریتم‌هایی برای رمزگذاری متن برای غیرقابل خواندن آن برای افراد غیرمجاز

Keys

پارامترهای عددی که رفتار رمزها را تغییر می‌دهند.

Symmetric-key cryptosystems

الگوریتم‌هایی که از یک کلید برای رمزگذاری و رمزگشایی استفاده می‌کنند.

Asymmetric-key cryptosystems

الگوریتم‌هایی که از کلیدهای مختلفی برای رمزگذاری و رمزگشایی استفاده می‌کنند.

Public-key cryptography

سیستمی که ارسال پیام‌های مخفی را برای میلیون‌ها کامپیوتر آسان می‌کند.

Digital signatures

Checksum هایی که تأیید می‌کنند پیام جعلی یا دستکاری نشده است.

Digital certificates

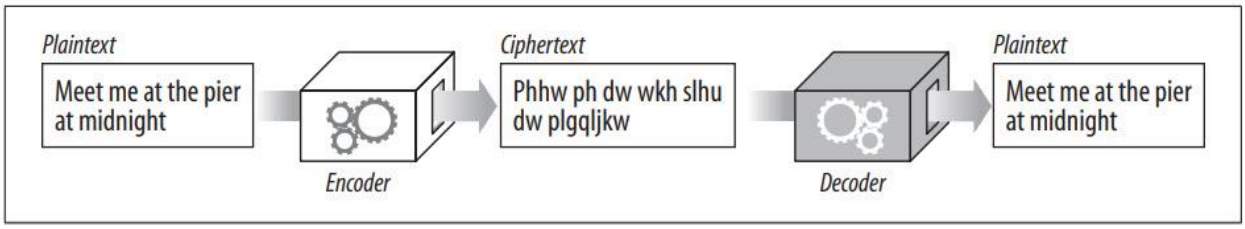
اطلاعات شناسایی، تایید شده و توسط یک سازمان مورد اعتماد امضا شده است.

The Art and Science of Secret Coding

Cryptography هنر و علم رمزگذاری و رمزگشایی پیام‌ها است. هزاران سال است که مردم از روش‌های رمزنگاری برای ارسال پیام‌های مخفی استفاده می‌کنند. با این حال، Cryptography می‌تواند بیشتر از رمزگذاری پیام‌ها برای جلوگیری از خواندن توسط افراد غیرمجاز انجام دهد. همچنین می‌توان از آن برای جلوگیری از دستکاری پیام‌ها استفاده کرد. رمزنگاری حتی می‌تواند برای اثبات اینکه شما واقعاً یک پیام یا تراکنش را نوشته‌اید، درست مانند امضای دست‌نویس شما روی چک یا مهر و موم برجسته روی یک پاکت استفاده شود.

Ciphers

Cryptography مبتنی بر کدهای مخفی به نام Cipher است. Cipher یک طرح کدگذاری است - یک راه خاص برای رمزگذاری یک پیام و یک روش همراه برای رمزگشایی Secret. پیام اصلی، قبل از اینکه کدگذاری شود، اغلب Plaintext یا Cleartext نامیده می‌شود. پیام رمزگذاری شده، پس از اعمال رمز، اغلب Ciphertext نامیده می‌شود. شکل زیر یک مثال ساده را در این مورد نشان می‌دهد.



هزاران سال است که از رمزها برای تولید پیام‌های مخفی استفاده می‌شود. افسانه‌ها حاکی از آن است که ژولیوس سزار از رمز چرخشی سه کاراکتری استفاده می‌کرد که در آن هر کاراکتر در پیام با یک کاراکتر در سه موقعیت حروف الفبا به جلو جایگزین می‌شد. در الفبای مدرن ما، A با D، B با E و غیره جایگزین می‌شود.

به عنوان مثال، در شکل زیر، پیام "Meet me at the Pier at Midnight" در متن رمزی " phhw ph dw wkh slhu dw plgqljkw" با استفاده از رمز rot3 (چرخش با ۳ کاراکتر) رمزگذاری می‌شود. متن رمز را می‌توان با اعمال کدگذاری معکوس، چرخاندن -۳ کاراکتر در حروف الفبا، به پیام متنی اصلی برگرداند.

برای سادگی مثال، ما علائم نگارشی یا فضای خالی را نمی‌چرخانیم، اما شما می‌توانید این کار را هم انجام دهید.

Cipher Machines

رمزها به عنوان الگوریتم‌های نسبتاً ساده شروع شدند، زیرا انسان‌ها باید خودشان رمزگذاری و رمزگشایی را انجام دهند. از آنجایی که رمزها ساده بودند، مردم می‌توانستند با استفاده از مداد و کاغذ و کتاب‌های کد، کدها را شناسایی کنند. با این حال، برای افراد باهوش نیز این امکان وجود داشت که به راحتی کدها را بشکنند.

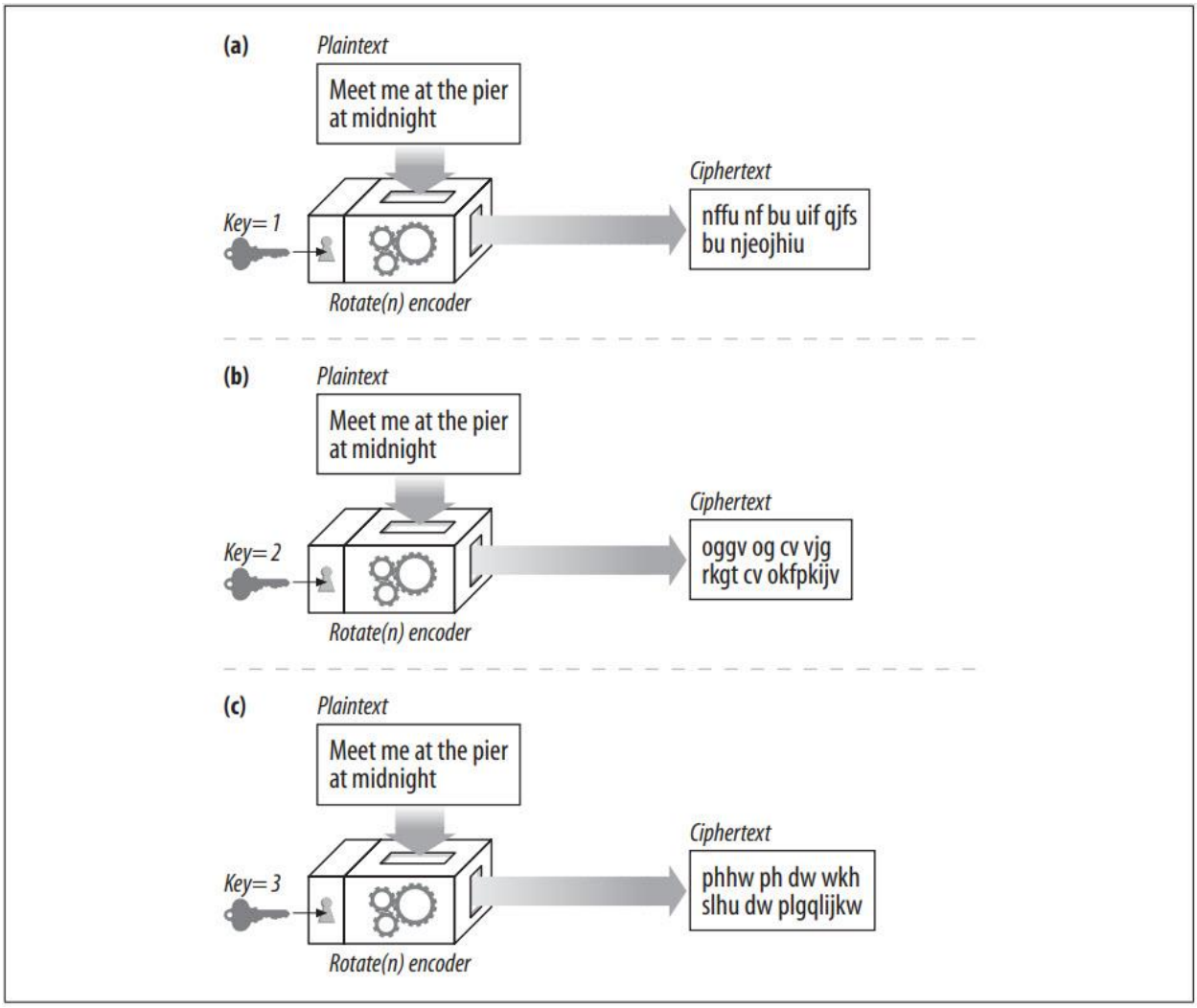
با پیشرفت تکنولوژی، مردم شروع به ساخت ماشین‌هایی کردند که می‌توانستند با استفاده از رمزهای بسیار پیچیده‌تر پیام‌ها را به سرعت و با دقت رمزگذاری و رمزگشایی کنند. این ماشین‌های رمزگذاری می‌توانند به جای انجام چرخش‌های ساده، کاراکترها را جایگزین کنند، ترتیب کاراکترها را جابه‌جا کنند، و پیام‌ها را برش دهند تا شکستن کدها بسیار سخت‌تر شود.

Keyed Ciphers

از آنجایی که الگوریتم‌های کد و ماشین‌ها می‌توانستند به دست دشمن بیفتند، اکثر ماشین‌ها شماره‌گیری داشتند که می‌توان آن‌ها را روی تعداد زیادی مقادیر مختلف تنظیم کرد که نحوه کار رمز را تغییر می‌داد. حتی اگر دستگاه به سرقت رفته باشد، بدون تنظیمات شماره‌گیری مناسب (مقادیر کلید) رمزگشا کار نمی‌کند.

این پارامترهای رمز را کلید می‌نامیدند. برای اینکه فرآیند رمزگشایی به درستی کار کند، باید کلید مناسب را در دستگاه رمزگذاری وارد کنید. کلیدهای رمز باعث می‌شوند که یک ماشین رمز مانند مجموعه‌ای از بسیاری از ماشین‌های رمزگذاری مجازی عمل کند که هر کدام به دلیل داشتن مقادیر کلیدی متفاوت، رفتار متفاوتی دارند.

شکل زیر نمونه ای از رمزهای کلیددار را نشان می‌دهد.



الگوریتم رمز، رمز اولیه rotate-by-N است. مقدار N توسط کلید کنترل می‌شود. همان پیام ورودی، "Meet me at the Pier at Midnight"، که از طریق یک دستگاه رمزگذاری ارسال می‌شود، بسته به مقدار کلید، خروجی‌های متفاوتی تولید می‌کند. امروزه تقریباً همه الگوریتم‌های رمزنگاری از کلیدها استفاده می‌کنند.

Digital Ciphers

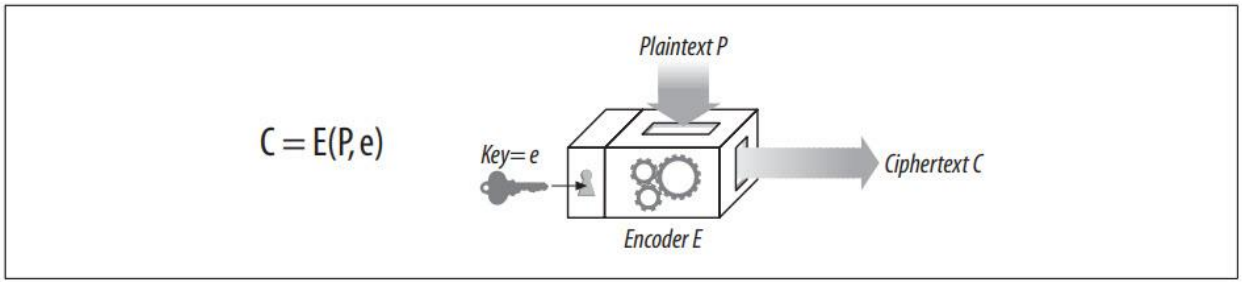
با ظهور محاسبات دیجیتال، دو پیشرفت عمده رخ داد:

- رمزگذاری و الگوریتم‌های رمزگشایی پیچیده امکان پذیر شد و از محدودیت‌های سرعت و عملکرد ماشین‌های مکانیکی رها شد.

- پشتیبانی از کلیدهای بسیار بزرگ امکان پذیر شد، به طوری که یک الگوریتم رمز می‌تواند تریلیون‌ها الگوریتم رمز مجازی را به دست آورد که هر کدام با مقدار کلید متفاوت است. هر چه کلید طولانی‌تر باشد، ترکیب‌های بیشتری از رمزگذاری‌ها امکان‌پذیر است و شکستن کد با حدس زدن تصادفی کلیدها سخت‌تر می‌شود.

برخلاف کلیدهای فلزی فیزیکی یا تنظیمات شماره‌گیری در دستگاه‌های مکانیکی، کلیدهای دیجیتال فقط اعداد هستند. این مقادیر کلید دیجیتال، ورودی به الگوریتم‌های رمزگذاری و رمزگشایی هستند. الگوریتم‌های کدگذاری توابعی هستند که تکه‌ای از داده‌ها را می‌گیرند و بر اساس الگوریتم و مقدار کلید آن را رمزگذاری/رمزگشایی می‌کنند.

با توجه به یک پیام متنی ساده به نام P، یک تابع رمزگذاری به نام E، و یک کلید رمزگذاری دیجیتالی به نام e، می توانید یک پیام متن رمز شده C ایجاد کنید (شکل زیر).

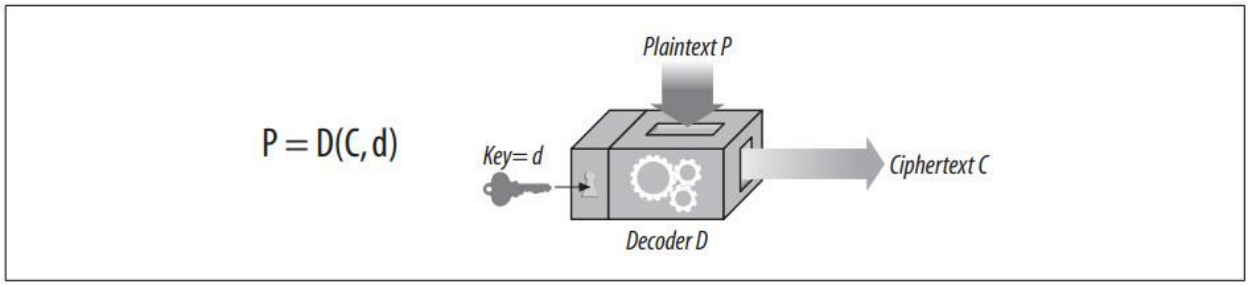


می‌توانید با استفاده از تابع رمزگشای D و کلید رمزگشایی d، متن رمز C را به متن اصلی P رمزگشایی کنید. البته، توابع رمزگشایی و رمزگذاری معکوس یکدیگر هستند. رمزگشایی رمزگذاری P پیام اصلی P را برمی‌گرداند.

Symmetric-Key Cryptography

بیایید با جزئیات بیشتری در مورد نحوه کار کلیدها و رمزها با هم صحبت کنیم. بسیاری از الگوریتم‌های رمز دیجیتال رمزهای کلید متقارن نامیده می‌شوند، زیرا از همان مقدار کلید برای رمزگذاری استفاده می‌کنند که برای رمزگشایی استفاده می‌کنند (e = d). بیایید کلید را k صدا کنیم.

در رمزنگاری کلید متقارن، هم فرستنده و هم گیرنده باید یک کلید مخفی مشترک، k، برای برقراری ارتباط داشته باشند. فرستنده از کلید مخفی مشترک برای رمزگذاری پیام استفاده می‌کند و متن رمزی حاصل را به گیرنده می‌فرستد. گیرنده متن رمز را می‌گیرد و تابع رمزگشایی را همراه با همان کلید مخفی مشترک برای بازیابی متن اصلی اصلی اعمال می‌کند (شکل زیر).



برخی از الگوریتم‌های رمزنگاری متقارن رایج عبارتند از DES، Triple-DES، RC2 و RC4.

Key Length and Enumeration Attacks

بسیار مهم است که کلیدهای مخفی مخفی بمانند. در بیشتر موارد، الگوریتم‌های رمزگذاری و رمزگشایی دانش عمومی هستند، بنابراین کلید تنها چیزی است که مخفی است!

یک الگوریتم رمزگذاری خوب، نفوذگر را مجبور می‌کند تا تک تک مقادیر کلیدی ممکن در جهان را برای شکستن کد امتحان کند. آزمایش تمام مقادیر کلیدی بوسیله Brute Force، حمله Enumeration نامیده می‌شود. اگر فقط چند مقدار کلیدی ممکن وجود داشته باشد، یک فرد بدخواه می‌تواند همه آن‌ها را با Brute Force مرور کند و در نهایت کد را بشکند. اما اگر مقادیر کلیدی زیادی وجود داشته باشد، ممکن است روزها، سال‌ها یا حتی به اندازه عمر کیهان طول بکشد تا به دنبال کلیدی باشد که رمز را بشکند.

تعداد مقادیر کلید ممکن بستگی به تعداد بیت‌های موجود در کلید و تعداد کلیدهای ممکن معتبر دارد. برای رمزهای کلید متقارن، معمولاً همه مقادیر کلید معتبر هستند. یک کلید ۸ بیتی فقط ۲۵۶ کلید ممکن، یک کلید ۴۰ بیتی دارای ۲ به توان ۴۰ کلید ممکن (حدود یک تریلیون کلید) و یک کلید ۱۲۸ بیتی است. کلید حدود ۳۴۰،۰۰۰،۰۰۰،۰۰۰،۰۰۰،۰۰۰،۰۰۰،۰۰۰،۰۰۰،۰۰۰،۰۰۰،۰۰۰،۰۰۰،۰۰۰،۰۰۰،۰۰۰ کلید ممکن را تولید می‌کند.

برای رمزنگاری‌های متقارن ، کلیدهای ۴۰ بیتی برای تراکنش‌های کوچک و غیر بحرانی به اندازه کافی ایمن در نظر گرفته می‌شوند. با این حال، آن‌ها توسط ایستگاه‌های کاری پرسرعت امروزی که اکنون می‌توانند میلیاردها محاسبه در ثانیه انجام دهند، شکستنی هستند.

در مقابل، کلیدهای ۱۲۸ بیتی برای رمزنگاری کلید متقارن بسیار قوی در نظر گرفته می‌شوند. در واقع، کلیدهای بلند چنان تأثیری بر امنیت رمزنگاری دارند که دولت ایالات متحده کنترل صادرات را بر روی نرم افزارهای رمزنگاری که از کلیدهای طولانی استفاده می‌کنند، قرار داده است تا از ایجاد کدهای مخفی توسط سازمان‌های بالقوه متخاصم جلوگیری کند که آژانس امنیت ملی ایالات متحده (NSA) خود قادر به شکستن کدهای مخفی نباشد.

کتاب عالی Applied Cryptography ،Bruce Schneier (John Wiley & Sons)، شامل جدولی است که زمان لازم برای شکستن رمز DES را با حدس زدن همه کلیدها، با استفاده از فناوری و اقتصاد ۱۹۹۵، توصیف می‌کند. گزیده‌هایی از این جدول در جدول زیر نشان داده شده است.

Attack cost	40-bit key	56-bit key	64-bit key	80-bit key	128-bit key
$100,000	2 secs	35 hours	1 year	70,000 years	10^{19} years
$1,000,000	200 msecs	3.5 hours	37 days	7,000 years	10^{18} years
$10,000,000	20 msecs	21 mins	4 days	700 years	10^{17} years
$100,000,000	2 msecs	2 mins	9 hours	70 years	10^{16} years
$1,000,000,000	200 usecs	13 secs	1 hour	7 years	10^{15} years

با توجه به سرعت ریزپردازنده‌های ۱۹۹۵، مهاجمی که مایل به خرج کردن ۱۰۰۰۰۰ دلار در سال ۱۹۹۵ باشد می‌تواند یک کد DES 40 بیتی را در حدود ۲ ثانیه بشکند. کامپیوترهای سال ۲۰۰۲ در حال حاضر ۲۰ برابر سریعتر از سال ۱۹۹۵ هستند. مگر اینکه کاربران به طور مکرر کلیدها را تغییر دهند. توجه داشته باشید که کلیدهای ۴۰ بیتی در برابر مخالفان با انگیزه ایمن نیستند.

اندازه کلید استاندارد DES 56 بیتی امن‌تر است. در اقتصاد سال ۱۹۹۵، یک حمله ۱ میلیون دلاری هنوز چندین ساعت طول می‌کشد تا رمز را بشکند. اما شخصی که به ابرکامپیوترها دسترسی داشته باشد می‌تواند با استفاده از Brute Force در عرض چند ثانیه کد را بشکند.

در مقابل، اعتقاد بر این است که کلیدهای ۱۲۸ بیتی DES، که از نظر اندازه مشابه کلیدهای Triple-DES هستند، توسط هر کسی و به هر قیمتی با استفاده از یک حمله Brute Force غیرقابل شکستن هستند.

Establishing Shared Keys

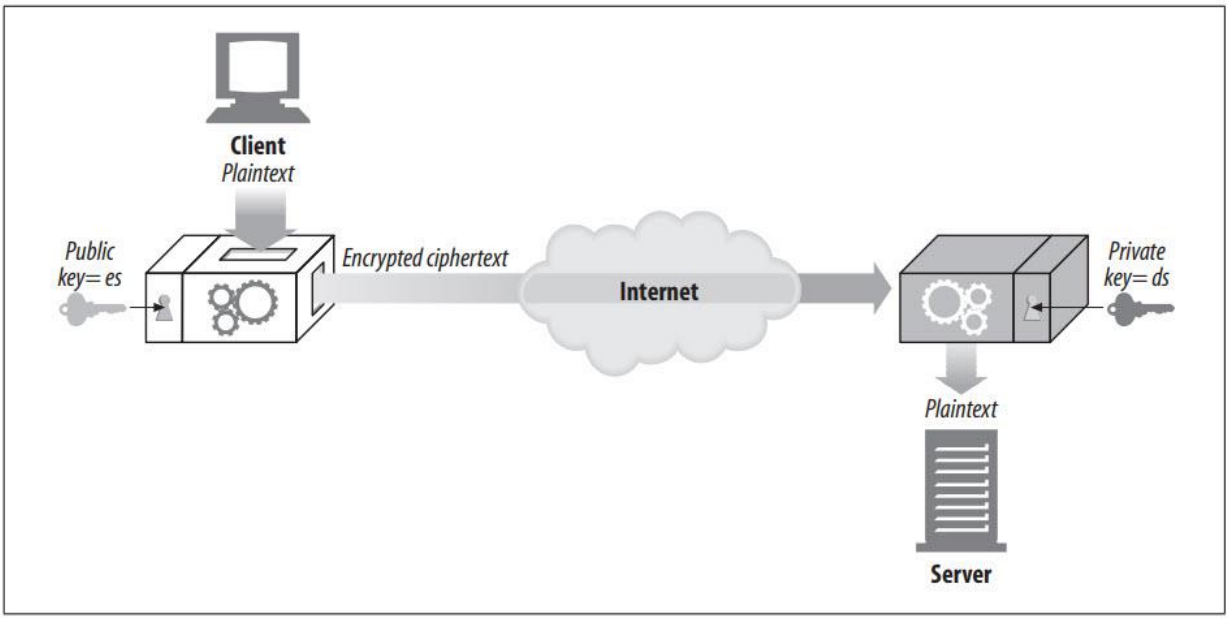
یکی از معایب رمزنگاری‌های متقارن این است که فرستنده و گیرنده قبل از اینکه بتوانند با یکدیگر صحبت کنند باید یک کلید مخفی مشترک داشته باشند.

اگر می‌خواهید با فروشگاه سخت‌افزار Joe به‌طور امن صحبت کنید، شاید بعد از تماشای برنامه‌ای برای بهسازی خانه در تلویزیون عمومی، چند ابزار نجاری سفارش دهید، باید یک کلید مخفی خصوصی بین خود و www.joes-hardware.com ایجاد کنید. اگر می‌خواهید هر چیزی را با خیال راحت سفارش دهید شما به راهی برای تولید کلید مخفی و به خاطر سپردن آن نیاز دارید. هم شما و هم Joe’s Hardware و هر کاربر اینترنتی دیگری هزاران کلید برای تولید و به خاطر سپردن دارید.

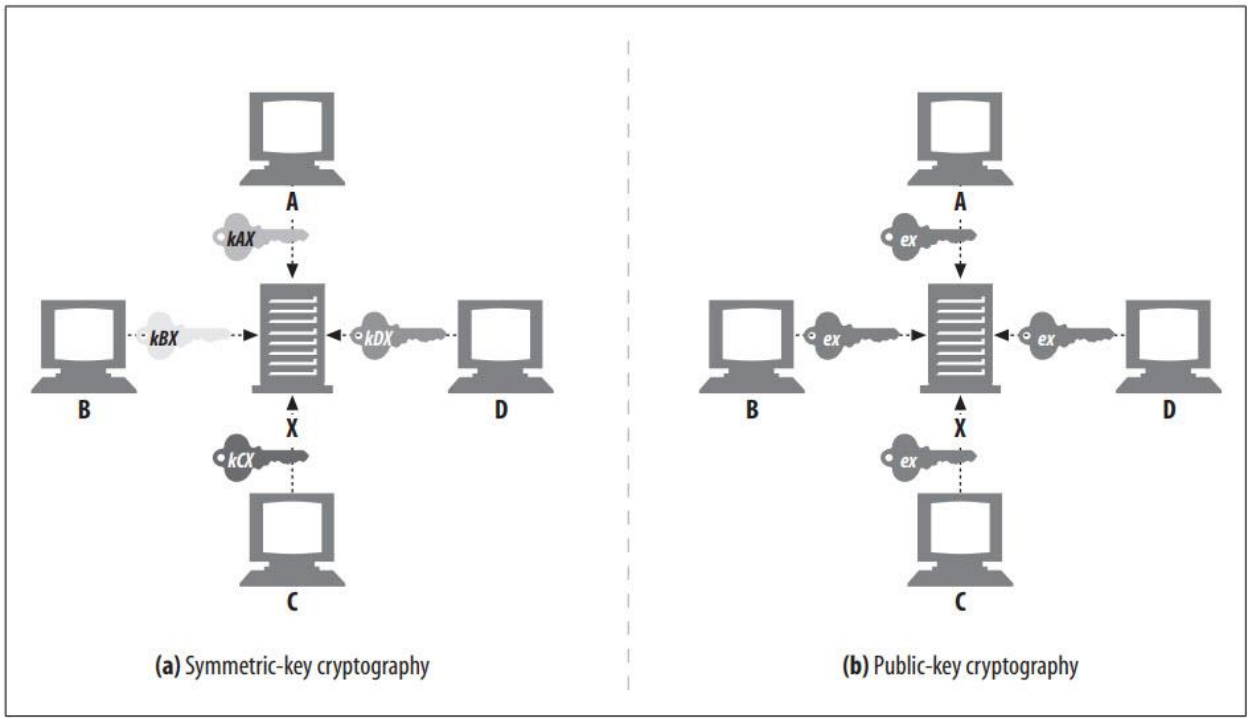
در نظر بگیرید که آلیس (A)، باب (B) و کریس (C) همگی می‌خواستند با Joe's Hardware (J) صحبت کنند. A، B، و C هر کدام باید کلیدهای مخفی خود را با J ایجاد کنند. A به کلید kAJ، B به کلید kBJ و C به کلید kCJ نیاز دارد. هر جفت طرف در ارتباط به کلید خصوصی خود نیاز دارد. اگر N گره وجود داشته باشد و هر گره باید به طور ایمن با تمام گره‌های N-1 دیگر صحبت کند، تقریباً N به توان ۲ کل کلیدهای مخفی وجود دارد: یک کابوس مدیریتی.

Public-Key Cryptography

به جای یک کلید رمزگذاری/رمزگشایی واحد برای هر جفت میزبان، رمزنگاری کلید عمومی از دو کلید نامتقارن استفاده می‌کند: یکی برای رمزگذاری پیام‌ها برای یک میزبان و دیگری برای رمزگشایی پیام‌های میزبان. کلید رمزگذاری به طور عمومی برای جهان شناخته شده است (بنابراین نام رمزنگاری کلید عمومی است)، اما تنها میزبان کلید رمزگشایی خصوصی را می‌داند (شکل زیر را ببینید).



این کار ایجاد کلید را بسیار آسان‌تر می‌کند، زیرا همه می‌توانند کلید عمومی را برای یک میزبان خاص پیدا کنند. اما کلید رمزگشایی مخفی نگه داشته می‌شود، بنابراین فقط گیرنده می‌تواند پیام‌های ارسال شده به آن را رمزگشایی کند.



تصویر بالا را در نظر بگیرید. گره X می‌تواند کلید رمزگذاری ex خود را بگیرد و آن را به صورت عمومی منتشر کند. (همانطور که بعداً خواهیم دید، بیشتر جستجوی کلید عمومی در واقع از طریق گواهی‌های دیجیتال انجام می‌شود، اما جزئیات نحوه یافتن کلیدهای عمومی در حال حاضر اهمیت چندانی ندارد - فقط بدانید که آن‌ها در جایی برای عموم در دسترس هستند.) اکنون هر کسی که بخواهد پیامی به گره X ارسال کند می‌تواند از همان کلید عمومی معروف استفاده کند.

حتی اگر همه می‌توانند پیام‌های X را با یک کلید رمزگذاری کنند، هیچ کس به جز X نمی‌تواند پیام‌ها را رمزگشایی نماید، زیرا فقط X دارای کلید خصوصی رمزگشایی dx است. تقسیم کلیدها به هر کسی اجازه می‌دهد پیامی را رمزگذاری کند، اما توانایی رمزگشایی پیام‌ها را فقط به مالک محدود می‌کند. این کار ارسال امن پیام‌ها به سرورها را برای گره‌ها آسان‌تر می‌کند، زیرا آن‌ها فقط می‌توانند کلید عمومی سرور را جستجو کنند.

فناوری رمزگذاری کلید عمومی امکان استقرار پروتکل‌های امنیتی را برای هر کاربر رایانه در سراسر جهان فراهم می‌کند. به دلیل اهمیت زیاد ساخت یک مجموعه فناوری کلید عمومی استاندارد، یک ابتکار عظیم استانداردهای زیرساخت کلید عمومی (PKI) برای بیش از یک دهه در حال انجام است.

RSA

چالش هر سیستم رمزنگاری نامتقارن با کلید عمومی این است که مطمئن شود هیچ فردبدخواهی نمی‌تواند کلید خصوصی و مخفی را محاسبه کند - حتی اگر همه سرنخ‌های زیر را داشته باشد:

- کلید عمومی (که هر کسی می‌تواند آن را دریافت کند، زیرا عمومی است)
- یک تکه از متن رمز رهگیری شده (به دست آمده با ردیابی شبکه)
- یک پیام و متن رمزی مرتبط با آن (با اجرای Encoder بر روی هر متنی به دست می‌آید)

یکی از محبوب‌ترین سیستم‌های رمزنگاری کلید عمومی که تمام این نیازها را برآورده می‌کند، الگوریتم RSA است که در MIT اختراع شد و متعاقباً توسط RSA Data Security تجاری شد. با توجه به یک کلید عمومی، یک قطعه متن ساده دلخواه، متن رمزی مرتبط از رمزگذاری متن ساده با کلید عمومی، خود الگوریتم RSA و حتی کد منبع اجرای RSA، شکستن کد برای یافتن کلید خصوصی مربوطه مشکلی به اندازه محاسبه اعداد اول بزرگ می‌باشد. اعتقاد بر این است که یکی از سخت‌ترین مسائل در تمام علوم کامپیوتر همین موضوع محاسبه اعداد اول است. بنابراین، اگر بتوانید راهی سریع برای تبدیل اعداد بزرگ به اعداد اول پیدا کنید، نه تنها می‌توانید به حساب‌های بانکی سوئیس نفوذ کنید، بلکه می‌توانید جایزه تورینگ را نیز ببرید.

جزئیات رمزنگاری RSA شامل برخی از ریاضیات پیچیده است، بنابراین ما در اینجا به آن‌ها نخواهیم پرداخت. کتابخانه‌های زیادی وجود دارد که به شما امکان می‌دهد الگوریتم‌های RSA را بدون نیاز به مدرک دکترا در نظریه اعداد انجام دهید.

Hybrid Cryptosystems and Session Keys

رمزنگاری نامتقارن با کلید عمومی بسیار خوب است، زیرا هر کسی می‌تواند پیام‌های ایمن را فقط با دانستن کلید عمومی آن به سرور عمومی ارسال کند. دو گره برای برقراری ارتباط ایمن، ابتدا نیازی به مذاکره با یک کلید خصوصی ندارند.

اما الگوریتم‌های رمزنگاری کلید عمومی از نظر محاسباتی کند هستند. در عمل، مخلوطی از هر دو طرح متقارن و نامتقارن استفاده می‌شود. به عنوان مثال، استفاده از رمزنگاری کلید عمومی برای برقراری ارتباط امن بین گره‌ها معمول است، اما سپس از آن کانال امن برای تولید و برقراری ارتباط یک کلید متقارن موقت و تصادفی برای رمزگذاری بقیه داده‌ها از طریق رمزنگاری متقارن سریع‌تر استفاده می‌شود.

Digital Signatures

تا کنون، ما در مورد انواع رمزهای کلیددار صحبت کرده‌ایم که از کلیدهای متقارن و نامتقارن استفاده می‌کنند تا به ما امکان رمزگذاری و رمزگشایی پیام‌های مخفی را بدهند.

علاوه بر رمزگذاری و رمزگشایی پیام‌ها، از سیستم‌های رمزنگاری می‌توان برای امضای پیام‌ها استفاده کرد و ثابت کرد که چه کسی پیام را نوشته و ثابت می‌کند پیام دستکاری نشده است. این تکنیک که امضای دیجیتال نامیده می‌شود برای گواهینامه‌های امنیت اینترنت مهم است که در قسمت بعدی به آن می‌پردازیم.

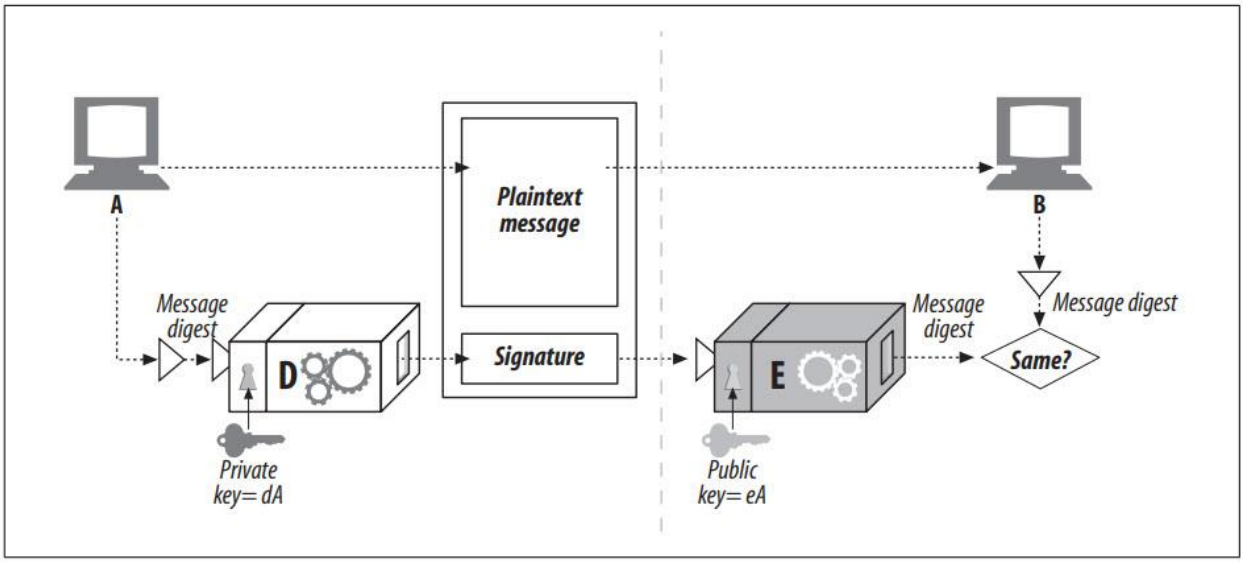
Signatures Are Cryptographic Checksums

امضای دیجیتال Checksum های رمزنگاری ویژه‌ای هستند که به یک پیام متصل می‌شوند. آن‌ها دو فایده دارند:

- امضاها ثابت می‌کنند که نویسنده، پیام را نوشته است. از آنجایی که فقط نویسنده دارای کلید خصوصی فوق محرمانه نویسنده است، فقط نویسنده می‌تواند این Checksum ها را محاسبه کند. Checksum به عنوان یک "امضای" شخصی نویسنده عمل می‌کند.
- امضاها از دستکاری پیام جلوگیری می‌کنند. اگر یک مهاجم مخرب پیام را در حین ارسال تغییر دهد، Checksum دیگر مطابقت نخواهد داشت و از آنجایی که Checksum شامل کلید خصوصی و مخفی نویسنده است، مهاجم قادر نخواهد بود یک Checksum صحیح برای پیام دستکاری شده بسازد.

امضاهای دیجیتال اغلب با استفاده از فناوری نامتقارن و کلید عمومی تولید می‌شوند. کلید خصوصی نویسنده به عنوان نوعی "اثر انگشت" استفاده می‌شود، زیرا کلید خصوصی فقط توسط مالک شناخته می‌شود.

شکل زیر مثالی را نشان می‌دهد که چگونه گره A می‌تواند پیامی را به گره B ارسال کند و آن را امضا کند:



- گره A پیام با طول متغیر را در یک Digest با اندازه ثابت قرار می‌دهد.
- گره A یک تابع "امضا" را برای Digest اعمال می‌کند که از کلید خصوصی کاربر به عنوان پارامتر استفاده می‌کند. از آنجا که فقط کاربر کلید خصوصی را می‌داند، یک تابع امضای صحیح

نشان می‌دهد که امضاکننده مالک است. در شکل بالا، ما از تابع رمزگشا D به عنوان تابع امضا استفاده می‌کنیم، زیرا شامل کلید خصوصی کاربر می‌شود.

- هنگامی که امضا محاسبه شد، گره A آن را به انتهای پیام اضافه می‌کند و هم پیام و هم امضا را به گره B می‌فرستد.
- در صورت دریافت، اگر گره B بخواهد مطمئن شود که گره A واقعاً پیام را نوشته است و پیام دستکاری نشده است، گره B می‌تواند امضا را بررسی کند. گره B امضای درهم شده با کلید خصوصی را می‌گیرد و تابع معکوس را با استفاده از کلید عمومی اعمال می‌کند. اگر Digest بدون بسته بندی با نسخه Digest خود گره B مطابقت نداشته باشد، یا پیام در حین ارسال دستکاری شده یا فرستنده کلید خصوصی گره A را نداشته است (و بنابراین گره A نبوده است).

Digital Certificates

در این بخش، در مورد گواهی‌های دیجیتال، «کارت‌های شناسایی» اینترنت صحبت می‌کنیم. گواهی‌های دیجیتال (که اغلب «certs» نامیده می‌شوند) حاوی اطلاعاتی درباره یک کاربر یا شرکتی است که توسط یک سازمان مورد اعتماد تضمین شده است.

همه ما انواع مختلفی از هویت را حمل می‌کنیم. برخی از شناسنامه‌ها، مانند گذرنامه و گواهینامه رانندگی، به اندازه کافی برای اثبات هویت در بسیاری از موقعیت‌ها مورد اعتماد هستند. به عنوان مثال، گواهینامه رانندگی ایالات متحده مدرک کافی برای اثبات هویت است که به شما اجازه می‌دهد برای شب سال نو سوار هواپیما به نیویورک شوید و این مدرک کافی برای سن شما است.

اشکال قابل اعتمادتر شناسایی، مانند گذرنامه، توسط دولت بر روی کاغذ مخصوص امضا و مهر می‌شود. جعل آن‌ها سخت‌تر است، بنابراین ذاتاً سطح بالاتری از اعتماد را دارند. برخی از نشان‌ها و کارت‌های هوشمند شرکتی شامل لوازم الکترونیکی برای کمک به تقویت هویت شرکت مخابراتی هستند. برخی از سازمان‌های دولتی فوق محرمانه حتی باید قبل از اعتماد به شناسه شما، اثر انگشت یا الگوهای مویرگی شبکیه شما را با شناسه شما مطابقت دهند!

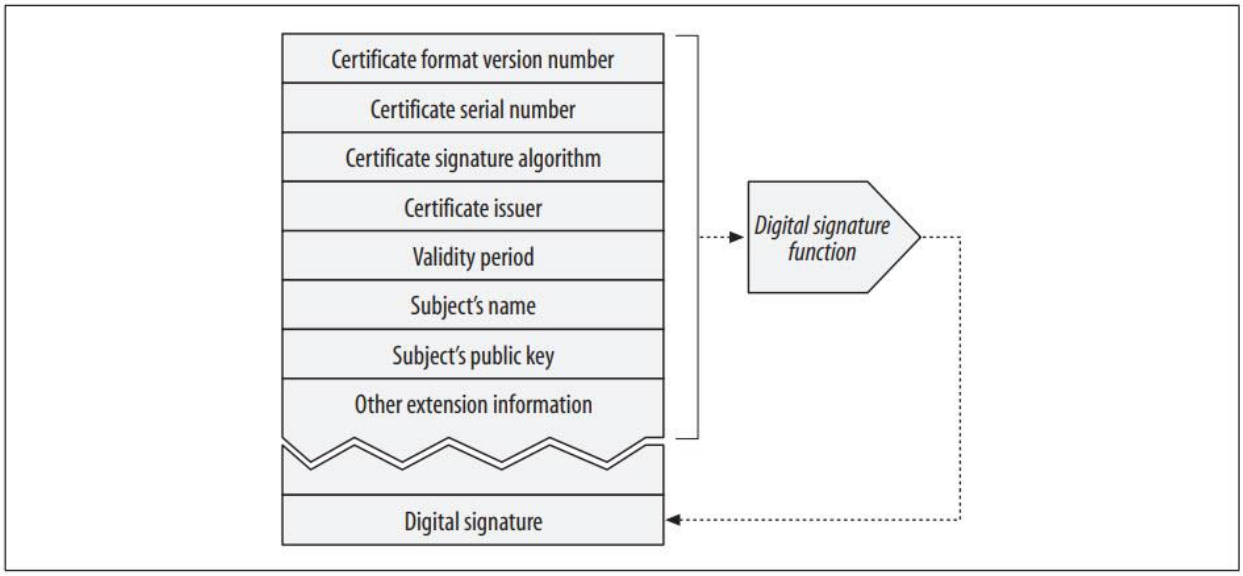
سایر اشکال شناسنامه، مانند کارت ویزیت، امکن جعل نسبتا آسان‌تری دارند، بنابراین مردم کمتر به این اطلاعات اعتماد دارند. آن‌ها ممکن است برای تعاملات حرفه‌ای خوب باشند، اما احتمالاً هنگام درخواست وام مسکن، مدرک کافی برای اشتغال ندارند.

The Guts of a Certificate

گواهی‌های دیجیتال همچنین حاوی مجموعه‌ای از اطلاعات هستند که همه آن‌ها به صورت دیجیتالی توسط مرجع صدور گواهی یا Certificate Authority رسمی امضا شده‌اند. گواهی‌های دیجیتال پایه معمولاً شامل موارد اساسی مشترک برای شناسه‌های چاپی هستند، مانند:

- Subject's name (person, server, organization, etc.)
- Expiration date
- Certificate issuer (who is vouching for the certificate)
- Digital signature from the certificate issuer

علاوه بر این، گواهی‌های دیجیتال اغلب حاوی کلید عمومی Subject و همچنین اطلاعات توصیفی درباره موضوع و الگوریتم امضای مورد استفاده هستند. هر کسی می‌تواند یک گواهی دیجیتال ایجاد کند، اما همه نمی‌توانند از یک مرجع معتبر امضاکننده برای تضمین اطلاعات گواهی و امضای گواهی با کلید خصوصی آن استفاده کنند. یک ساختار گواهی معمولی در شکل زیر نشان داده شده است.



X.509 v3 Certificates

متأسفانه، هیچ استاندارد واحد و جهانی برای گواهی‌های دیجیتال وجود ندارد. انواع مختلفی از گواهی‌های دیجیتال وجود دارد، همانطور که همه کارت‌های شناسایی چاپ شده حاوی اطلاعات یکسانی در یک مکان نیستند. خبر خوب این است که اکثر گواهینامه‌هایی که امروزه مورد استفاده قرار می‌گیرند اطلاعات خود را به شکل استانداردی به نام X.509 v3 ذخیره می‌کنند. گواهینامه‌های X.509 v3 یک روش استاندارد برای ساختاردهی اطلاعات گواهی در فیلدهای قابل تجزیه ارائه می‌کنند. انواع مختلف گواهی‌ها دارای مقادیر

فیلد متفاوتی هستند، اما اکثر آن‌ها از ساختار X.509 v3 پیروی می‌کنند. فیلدهای یک گواهی X.509 در جدول زیر توضیح داده شده است.

Field	Description
Version	The X.509 certificate version number for this certificate. Usually version 3 today.
Serial Number	A unique integer generated by the certification authority. Each certificate from a CA must have a unique serial number.
Signature Algorithm ID	The cryptographic algorithm used for the signature. For example, "MD2 digest with RSA encryption".
Certificate Issuer	The name for the organization that issued and signed this certificate, in X.500 format.
Validity Period	When this certificate is valid, defined by a start date and an end date.
Subject's Name	The entity described in the certificate, such as a person or an organization. The subject name is in X.500 format.
Subject's Public Key Information	The public key for the certificate's subject, the algorithm used for the public key, and any additional parameters.
Issuer Unique ID (optional)	An optional unique identifier for the certificate issuer, to allow the potential reuse of the same issuer name.
Subject Unique ID (optional)	An optional unique identifier for the certificate subject, to allow the potential reuse of the same subject name.
Extensions	An optional set of extension fields (in version 3 and higher). Each extension field is flagged as critical or noncritical. Critical extensions are important and must be understood by the certificate user. If a certificate user doesn't recognize a critical extension field, it must reject the certificate. Common extension fields in use include: *Basic Constraints* Subject's relationship to certification authority *Certificate Policy* The policy under which the certificate is granted *Key Usage* Restricts how the public key can be used
Certification Authority Signature	The certification authority's digital signature of all of the above fields, using the specified signing algorithm.

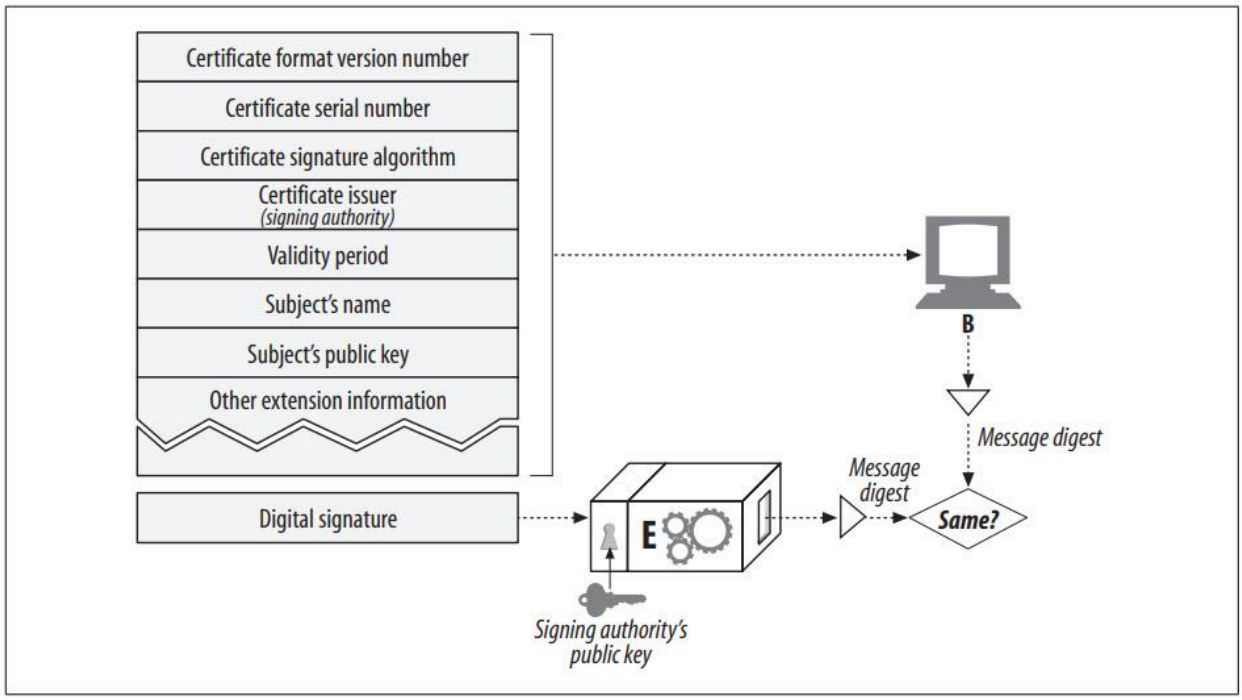
انواع مختلفی از گواهی‌های مبتنی بر X.509 وجود دارد، از جمله گواهی‌های وب سرور، گواهی‌های ایمیل کلاینت، گواهی‌های امضای کد نرم‌افزار و گواهی‌های Certificate Authority.

Using Certificates to Authenticate Servers

هنگامی که یک تراکنش وب ایمن از طریق HTTPS ایجاد می‌کنید، مرورگرهای مدرن به طور خودکار گواهی دیجیتال را برای سروری که به آن متصل است واکشی می‌کنند. اگر سرور گواهینامه نداشته باشد، اتصال ایمن با شکست مواجه می‌شود. گواهی سرور حاوی فیلدهای زیادی است، از جمله:

- Name and hostname of the web site
- Public key of the web site
- Name of the signing authority
- Signature from the signing authority

وقتی مرورگر گواهی را دریافت می‌کند، مرجع امضاکننده را بررسی می‌کند. اگر یک مرجع امضاکننده عمومی و معتبر باشد، مرورگر از قبل کلید عمومی خود را می‌داند (مرورگرها با گواهی‌های بسیاری از مقامات امضاکننده از قبل نصب شده نصب می‌شوند)، بنابراین می‌تواند امضا را همانطور که در بخش قبلی، «امضای دیجیتالی» بحث کردیم، تأیید کند. شکل زیر نشان می‌دهد که چگونه یکپارچگی یک گواهی با استفاده از امضای دیجیتال آن تأیید می‌شود.



اگر مرجع امضاکننده ناشناخته باشد، مرورگر مطمئن نیست که باید به مرجع امضاکننده اعتماد کند یا خیر و معمولاً یک کادر محاوره‌ای را برای کاربر نمایش می‌دهد تا آن را بخواند و ببیند آیا به امضاکننده اعتماد دارد یا خیر. امضاکننده ممکن است بخش فناوری اطلاعات محلی یا یک فروشنده نرم افزار باشد.

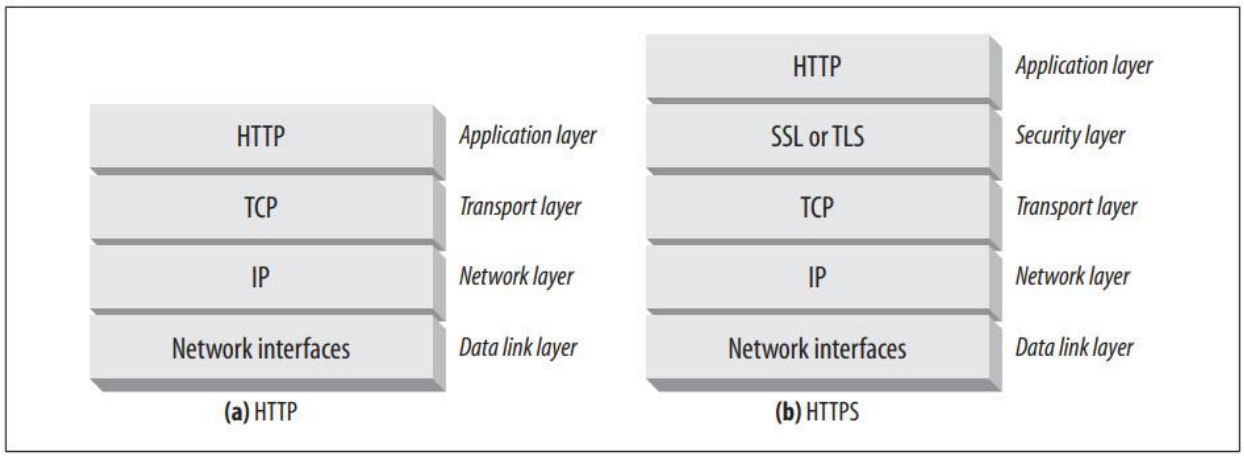
HTTPS: The Details

HTTPS محبوب‌ترین نسخه امن HTTP است. این پروتکل به طور گسترده در تمام مرورگرها و سرورهای تجاری اصلی پیاده سازی و در دسترس است. HTTPS پروتکل HTTP را با مجموعه‌ای قدرتمند از تکنیک‌های رمزنگاری متقارن، نامتقارن و مبتنی بر گواهی ترکیب می‌کند، که HTTPS را بسیار امن می‌کند، اما همچنین بسیار انعطاف‌پذیر و آسان برای مدیریت در سراسر اینترنت است.

HTTPS رشد برنامه‌های کاربردی اینترنتی را سرعت بخشیده است و نیروی اصلی در رشد سریع تجارت الکترونیک مبتنی بر وب بوده است. HTTPS همچنین در مدیریت گسترده و ایمن برنامه‌های کاربردی وب توزیع شده بسیار مهم بوده است.

HTTPS Overview

HTTPS همان HTTP است که از طریق یک لایه انتقال امن ارسال می‌شود. HTTPS به جای ارسال پیام‌های HTTP رمزگذاری نشده به TCP و در سراسر اینترنت (بخش a شکل زیر)، پیام‌های HTTP را ابتدا به یک لایه امنیتی ارسال می‌کند که آن‌ها را قبل از ارسال به TCP رمزگذاری می‌کند (بخش b شکل زیر).



امروزه لایه امنیتی HTTP توسط SSL و جایگزین مدرن آن یعنی TLS پیاده سازی می‌شود. ما از روش رایج استفاده از اصطلاح "SSL" به معنای SSL یا TLS پیروی می‌کنیم.

HTTPS Schemes

امروزه، HTTP امن اختیاری است. بنابراین، هنگام درخواست از یک وب سرور، به راهی نیاز داریم که به وب سرور بگوییم نسخه پروتکل امن HTTP را انجام دهد. این در طرح URL انجام می‌شود.

در HTTP معمولی و غیر ایمن، پیشوند طرح http URL است، مانند:

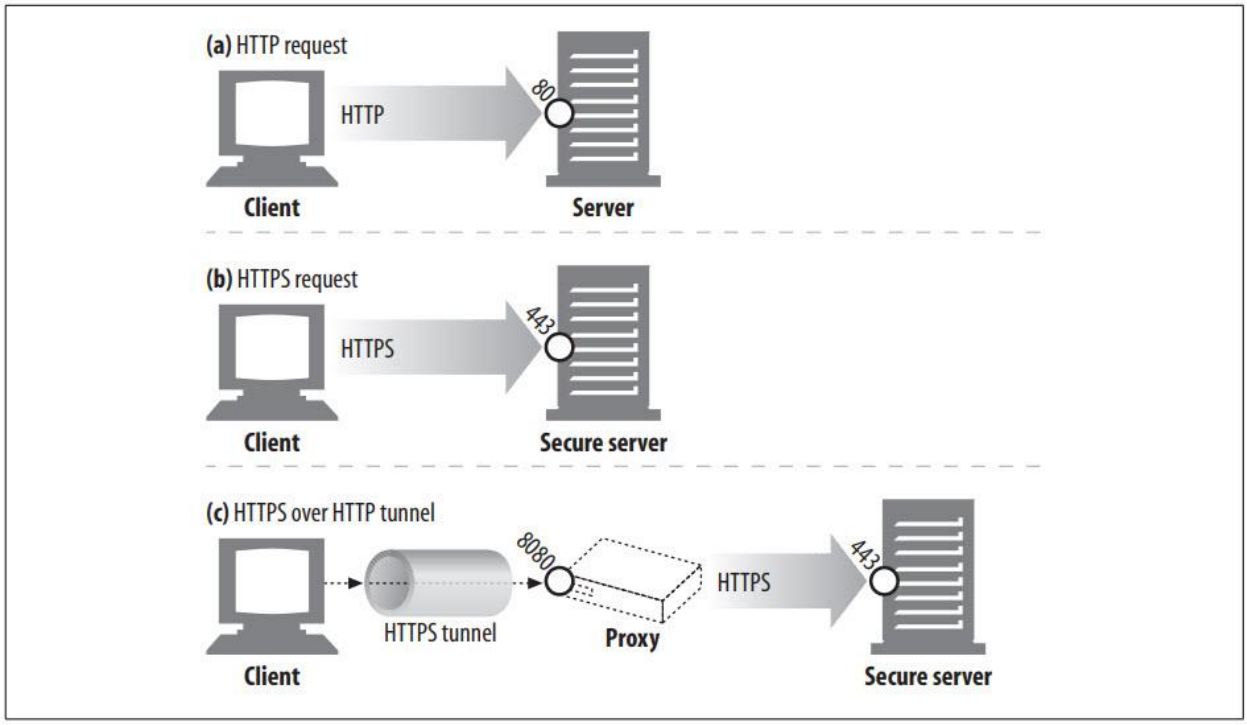
http://www.joes-hardware.com/index.html

در پروتکل امن HTTPS، پیشوند طرح https URL است، مانند:

https://cajun-shop.securesites.com/Merchant2/merchant.mv?Store_Code=AGCGS

هنگامی که از یک کلاینت (مانند یک مرورگر وب) خواسته می‌شود تا تراکنشی را روی یک منبع وب انجام دهد، طرح URL را بررسی می‌کند:

- اگر URL دارای طرح http باشد، کلاینت یک اتصال به سرور در پورت ۸۰ (به طور پیش فرض) باز می‌کند و دستورات HTTP قدیمی را برای آن ارسال می‌کند (بخش a شکل زیر).
- اگر URL دارای طرح https باشد، کلاینت یک اتصال به سرور در پورت ۴۴۳ (به طور پیش‌فرض) باز می‌کند و سپس با سرور «Handshake می‌کند»، برخی از پارامترهای امنیتی SSL را با سرور در فرمت باینری مبادله می‌کند و سپس دستورات HTTP رمزگذاری‌شده را دنبال می‌کند. (بخش b شکل زیر).



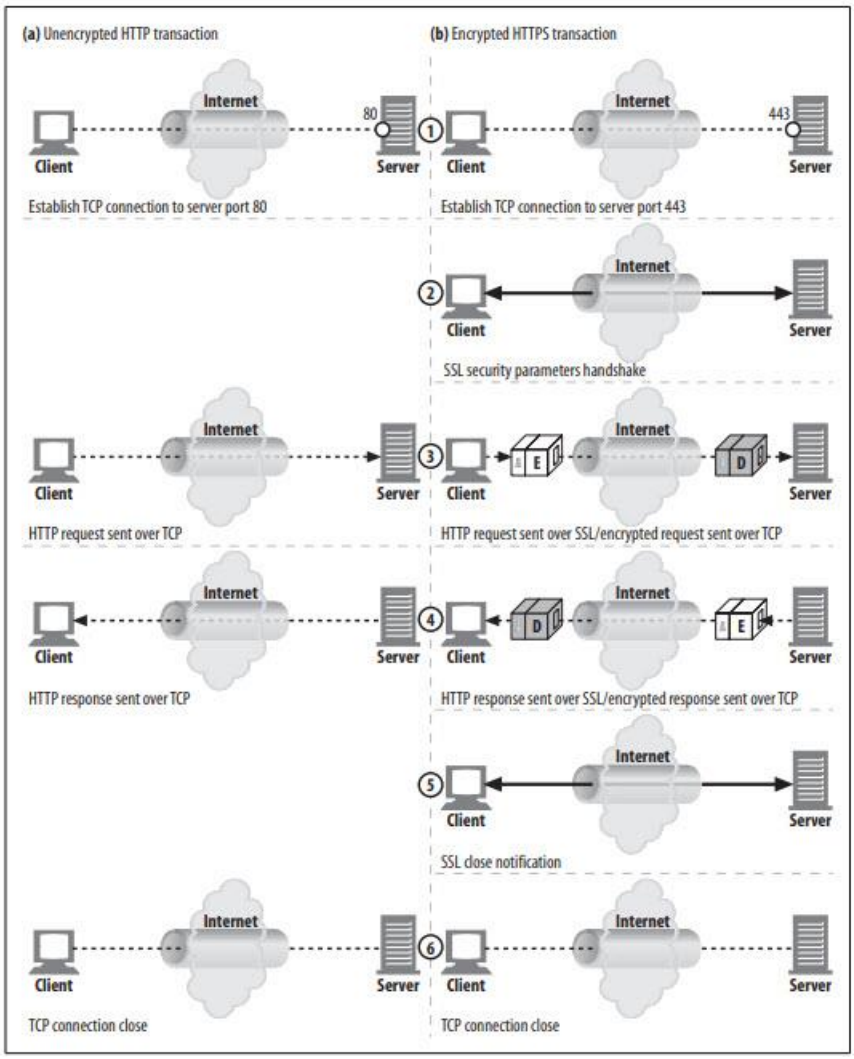
از آنجایی که ترافیک SSL یک پروتکل باینری است که کاملاً با HTTP متفاوت است، ترافیک در پورت‌های مختلف انجام می‌شود (SSL معمولاً روی پورت ۴۴۳ حمل می‌شود). اگر ترافیک SSL و HTTP هر دو به پورت ۸۰ می‌رسید، اکثر وب سرورها ترافیک SSL باینری را به عنوان HTTP اشتباه تفسیر می‌کردند و اتصال را

می بستند. لایه‌بندی یکپارچه‌تر سرویس‌های امنیتی در HTTP نیاز به پورت‌های مقصد متعدد را از بین می‌برد، اما در عمل مشکلات جدی ایجاد نمی‌کند.

بیایید کمی دقیق تر به نحوه تنظیم اتصالات SSL با سرورهای امن نگاه کنیم.

Secure Transport Setup

در HTTP رمزگذاری نشده، یک کلاینت یک اتصال TCP را به پورت ۸۰ روی سرور وب باز می‌کند، یک پیام درخواست ارسال می‌کند، یک پیام پاسخ دریافت می‌کند و اتصال را می‌بندد. این دنباله در بخش a شکل زیر ترسیم شده است.



این روش در HTTPS به دلیل لایه امنیتی SSL کمی پیچیده‌تر است. در HTTPS، کلاینت ابتدا یک اتصال به پورت ۴۴۳ (درگاه پیش‌فرض برای HTTP ایمن) در سرور وب باز می‌کند. هنگامی که اتصال

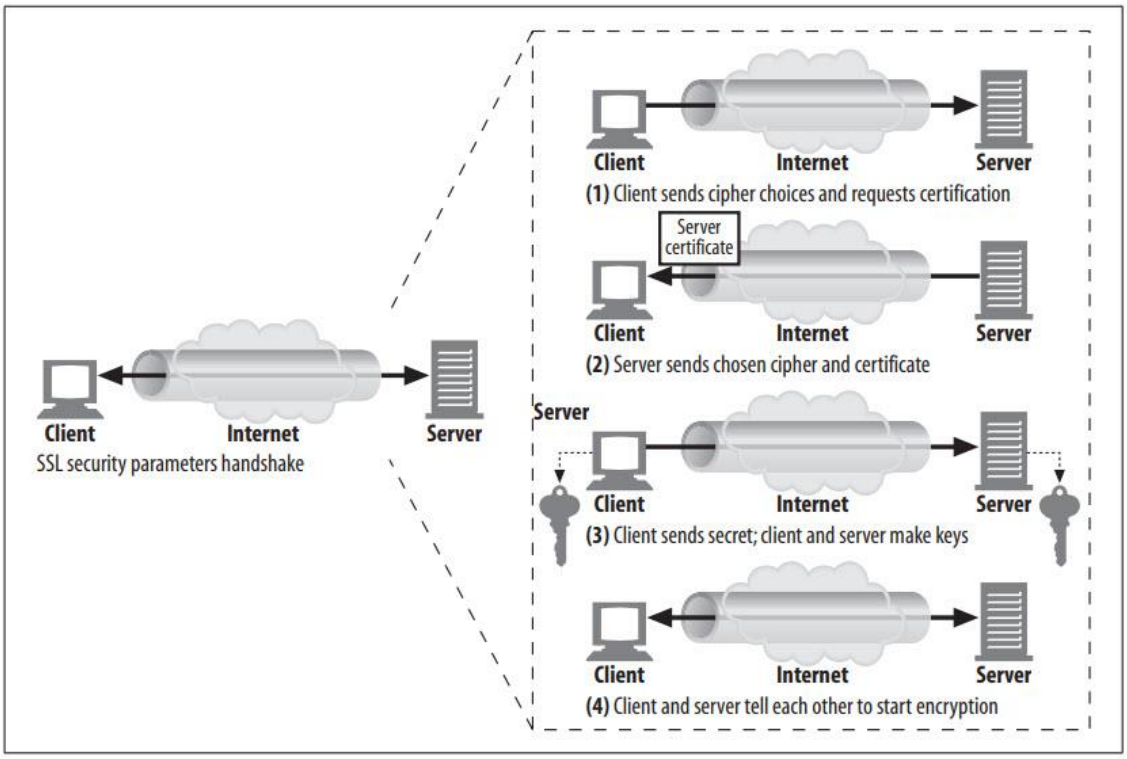
TCP برقرار شد، سرویس گیرنده و سرور لایه SSL را مقداردهی اولیه می‌کنند و در مورد پارامترهای رمزنگاری مذاکره نموده و کلیدها را مبادله می‌کنند. هنگامی که دست دادن کامل شد، مقداردهی اولیه SSL انجام می‌شود و کلاینت می‌تواند پیام‌های درخواستی را به لایه امنیتی ارسال کند. این پیام‌ها قبل از ارسال به TCP رمزگذاری می‌شوند. این روش در بخش b شکل بالا نشان داده شده است.

SSL Handshake

قبل از اینکه بتوانید پیام‌های HTTP رمزگذاری شده ارسال کنید، سرویس گیرنده و سرور باید یک SSL Handshake انجام دهند:

- تبادل شماره نسخه پروتکل
- رمزی را انتخاب کنید که هر طرف بداند.
- هویت هر طرف را تأیید کنید.
- کلیدهای جلسه موقت را برای رمزگذاری کانال ایجاد کنید.

قبل از اینکه هر داده HTTP رمزگذاری شده در سراسر شبکه پخش شود، SSL قبلاً یک دسته از داده‌های Handshake را برای برقراری ارتباط ارسال کرده است. ماهیت SSL Handshake در شکل زیر نشان داده شده است.



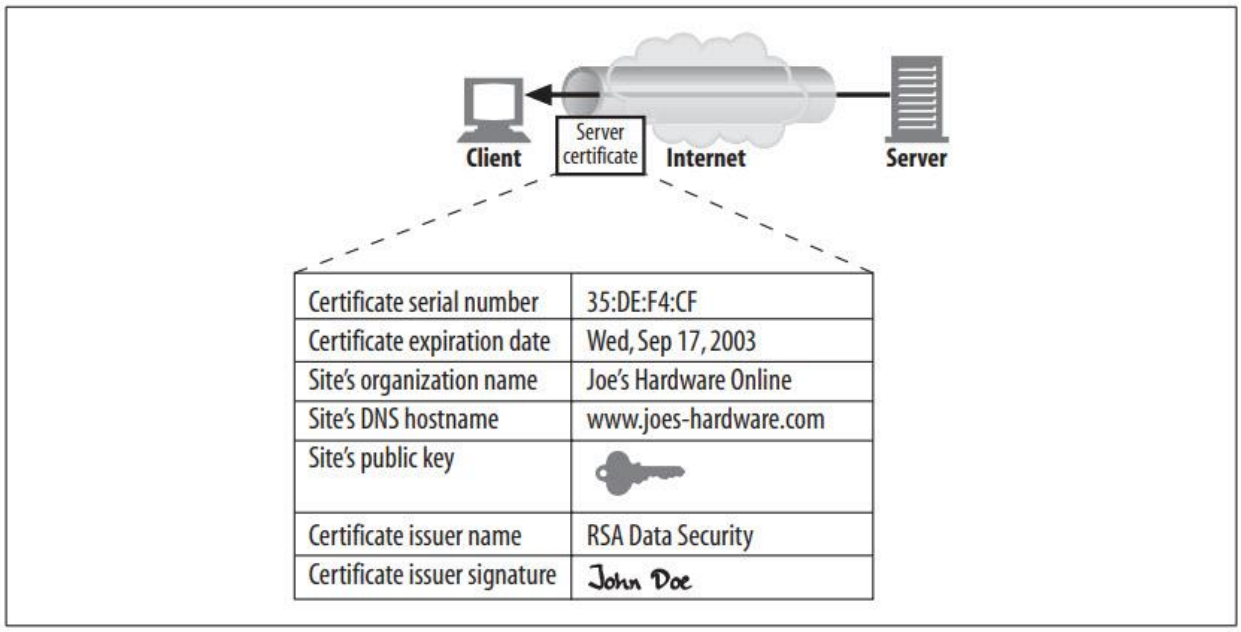
این یک نسخه ساده شده از SSL Handshake است. بسته به نحوه استفاده از SSL، Handshake می‌تواند پیچیده‌تر باشد، اما این ایده کلی است.

Server Certificates

SSL از احراز هویت متقابل پشتیبانی می‌کند، گواهی‌نامه‌های سرور را به کلاینت‌ها حمل می‌کند و گواهی‌نامه‌های کلاینت را به سرورها باز می‌گرداند. اما امروزه، گواهی کلاینت‌ها معمولا برای مرور استفاده نمی‌شود. اغلب کاربران حتی گواهی کلاینت شخصی ندارند. یک سرور وب می‌تواند برای دریافت گواهی درخواست نماید، اما این امر به ندرت در عمل رخ می‌دهد.

از سوی دیگر، تراکنش‌های امن HTTPS همیشه نیاز به گواهی سرور دارند. زمانی که شما یک تراکنش ایمن را بر روی یک سرور وب انجام می‌دهید، مانند ارسال اطلاعات کارت اعتباری خود، شما می‌خواهید بدانید که با سازمانی که فکر می‌کنید با آن صحبت می‌کنید، صحبت می‌کنید. گواهی‌های سرور، که توسط یک مقام شناخته‌شده امضا شده‌اند، به شما کمک می‌کنند پیش از ارسال کارت اعتباری یا اطلاعات شخصی خود، میزان اعتماد خود را به سرور ارزیابی کنید.

گواهی سرور یک گواهی مشتق شده از X.509 v3 است که نام سازمان، آدرس، نام دامنه سرور DNS و سایر اطلاعات را نشان می‌دهد (شکل زیر را ببینید). شما و نرم‌افزار کلاینت‌تان می‌توانید گواهی را بررسی کنید تا مطمئن شوید که همه چیز در حال به‌روزرسانی است.



Site Certificate Validation

در خود SSL نیازی نیست که شما گواهی سرور وب را بررسی کنید، اما اکثر مرورگرهای مدرن چک‌های منطقی ساده‌ای بر روی گواهی‌ها انجام می‌دهند و ابزار انجام چک‌های کامل‌تر را در اختیار شما قرار می‌دهند. یک الگوریتم برای تایید گواهی سرور وب، پیشنهاد شده توسط نت‌اسکیپ، اساس اکثر تکنیک‌های تایید مرورگر را تشکیل می‌دهد. مراحل عبارتند از:

Data Chek

اول، مرورگر تاریخ شروع و پایان گواهی را بررسی می‌کند تا اطمینان حاصل شود که گواهی هنوز معتبر است. اگر گواهی منقضی شده یا هنوز فعال نشده باشد، اعتبار گواهی با شکست مواجه می‌شود و مرورگر خطایی را نشان می‌دهد.

Signer trust check

هر گواهی توسط یک مرجع صدور گواهی یا همان CA که ضامن سرور است، امضا می‌شود. سطوح مختلفی از گواهی وجود دارد که هر کدام نیازمند سطوح متفاوتی از تایید پس‌زمینه هستند. به عنوان مثال، اگر برای گواهی سرور تجارت الکترونیک درخواست می‌کنید، معمولاً باید مدرک قانونی ثبت نام به عنوان یک تجارت را ارائه دهید.

هر کسی می‌تواند گواهی تولید کند، اما برخی از CA سازمان‌های معروفی هستند که رویه‌های کاملاً درک شده برای تأیید هویت و رفتار تجاری خوب متقاضیان گواهی را دارند. به همین

دلیل، مرورگرها فهرستی از مقامات امضاکننده مورد اعتماد را ارسال می کنند. اگر یک مرورگر گواهی امضا شده توسط مقامات ناشناخته (و احتمالاً مخرب) دریافت کند، مرورگر معمولاً یک هشدار نمایش می‌دهد. مرورگرها همچنین ممکن است هر گواهینامه‌ای را با مسیر امضای معتبر به یک CA قابل اعتماد بپذیرند. به عبارت دیگر، اگر یک CA مورد اعتماد گواهینامه‌ای را برای Sam's Signing Shop یک گواهی سایت را امضا کند، مرورگر ممکن است گواهی را به عنوان منشاء از یک مسیر CA معتبر بپذیرد.

Signature check

هنگامی که مرجع امضا به عنوان قابل‌اعتماد در نظر گرفته می‌شود، مرورگر یکپارچگی گواهی را با استفاده از کلید عمومی مرجع امضا برای امضا و مقایسه آن با مبلغ چک بررسی می‌کند.

Site identity check

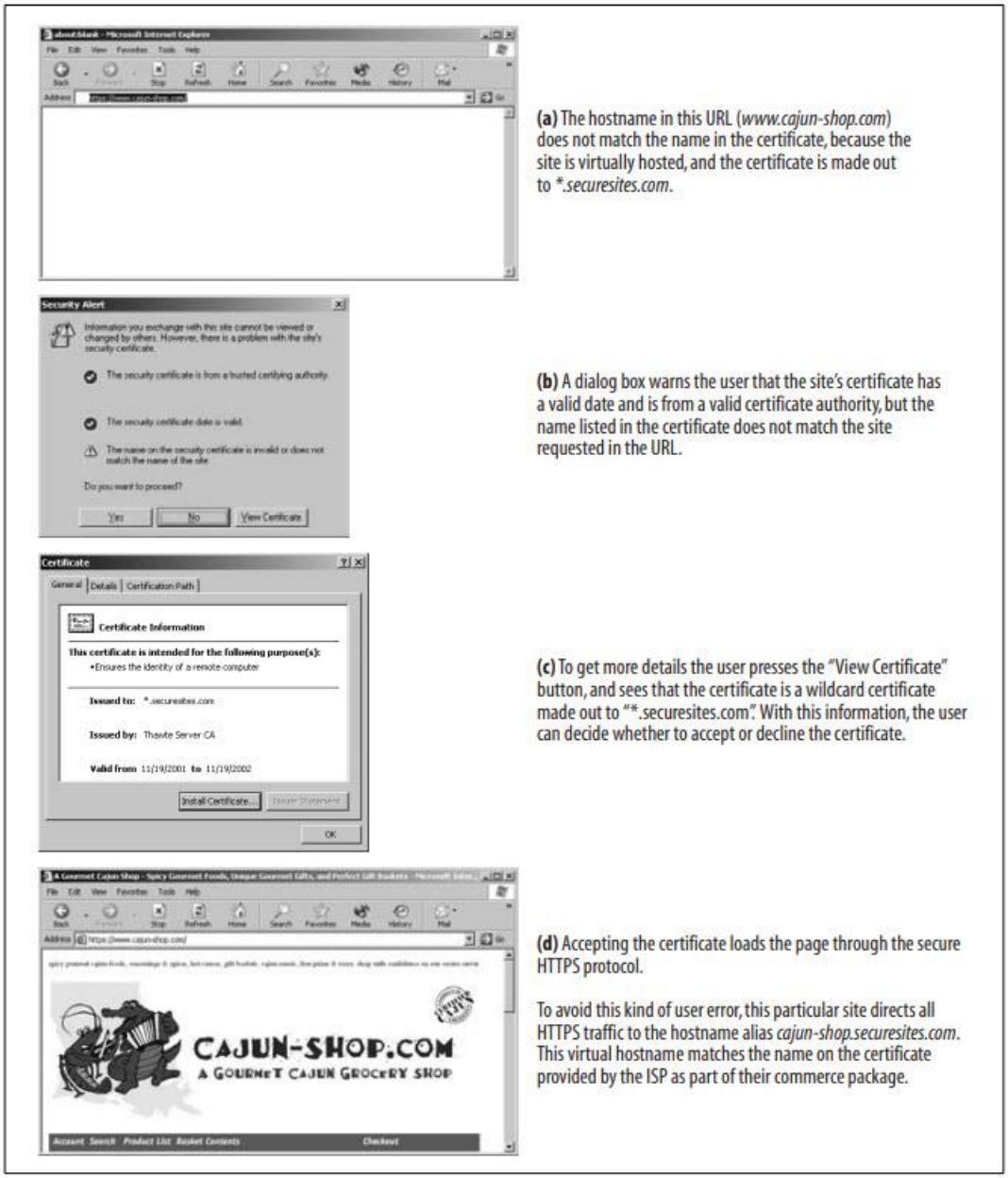
اغلب مرورگرها برای جلوگیری از کپی کردن گواهی شخص دیگری یا رهگیری ترافیک خود، سعی می‌کنند تایید کنند که نام دامنه در گواهی با نام دامنه سرور که با آن صحبت کرده‌اند مطابقت دارد. گواهی‌های سرور معمولاً شامل یک نام دامنه واحد هستند، اما برخی از CA ها گواهی‌هایی را ایجاد می‌کنند که حاوی فهرست‌هایی از نام سرور یا نام دامنه‌های با حروف عام است، برای کلاسترها یا مجموعه سرورها. اگر نام میزبان با هویت موجود در گواهی مطابقت نداشته باشد، سرویس گیرندگان کاربر محور باید یا به کاربر اطلاع دهند یا اتصال را با یک خطای گواهی بد خاتمه دهند.

Virtual Hosting and Certificates

گاهی اوقات رسیدگی به ترافیک ایمن در سایت‌هایی که به صورت مجازی میزبانی می‌شوند دشوار است (چند نام میزبان در یک سرور). برخی از برنامه‌های وب سرور محبوب فقط از یک گواهی پشتیبانی می‌کنند. اگر کاربری برای نام میزبان مجازی که کاملاً با نام گواهی مطابقت ندارد وارد شود، یک کادر هشدار نمایش داده می‌شود.

به عنوان مثال، سایت تجارت الکترونیکی Cajun-Shop.com را در نظر بگیرید. ارائه دهنده میزبانی سایت نام رسمی cajun-shop.securesites.com را ارائه کرده است. وقتی کاربران به https://www.cajun-shop.com مراجعه می‌کنند، نام میزبان رسمی فهرست شده در گواهی سرور (*.securesites.com) با نام میزبان مجازی که کاربر در آن مرور کرده است (www.cajun-shop.com) مطابقت ندارد و در این حالت هشدار مشابه شکل زیر ظاهر می‌شود.

برای جلوگیری از این مشکل، صاحبان Cajun-Shop.com همه کاربران را هنگام شروع تراکنش‌های امن به cajunshop.securesites.com هدایت می‌کنند. مدیریت گواهی برای سایت‌های میزبان مجازی می‌تواند کمی مشکل باشد.



A Real HTTPS Client

SSL یک پروتکل باینری پیچیده است. مگر اینکه متخصص رمزنگاری باشید، نباید مستقیماً ترافیک خام SSL را ارسال کنید. خوشبختانه چندین کتابخانه تجاری و منبع باز وجود دارد تا برنامه نویسی کلاینت‌ها و سرورهای SSL را آسان‌تر کند.

OpenSSL

OpenSSL محبوب‌ترین پیاده سازی متن باز SSL و TLS است. پروژه OpenSSL یک تلاش داوطلبانه مشترک برای توسعه یک جعبه ابزار قوی، تجاری و با امکانات کامل است که پروتکل‌های SSL و TLS را پیاده‌سازی می‌کند و همچنین یک کتابخانه رمزنگاری همه‌منظوره با قدرت کامل است. می‌توانید اطلاعاتی درباره OpenSSL دریافت کرده و نرم افزار را از http://www.openssl.org دانلود کنید.

همچنین ممکن است نام SSLeay (تلفظ S-S-L-e-a-y) را بشنوید. OpenSSL جانشین کتابخانه SSLeay است و رابط کاربری بسیار مشابهی دارد. SSLeay در اصل توسط Eric A. Young (the “eay” of SSLeay) توسعه داده شد.

A Simple HTTPS Client

در این بخش، از بسته OpenSSL برای نوشتن یک کلاینت HTTPS بسیار ابتدایی استفاده می‌کنیم. این سرویس گیرنده، یک اتصال SSL با یک سرور برقرار می‌کند، برخی از اطلاعات شناسایی را از سرور سایت چاپ می‌کند، یک درخواست HTTP GET را در کانال امن ارسال می‌کند، یک پاسخ HTTP را دریافت می‌کند و پاسخ را چاپ می‌کند.

برنامه C نشان داده شده در زیر یک اجرای OpenSSL از سرویس گیرنده HTTPS ساده است. برای ساده نگه داشتن برنامه، منطق مدیریت خطا و پردازش گواهی گنجانده نشده است.

از آنجایی که مدیریت خطا از این برنامه نمونه حذف شده است، باید از آن فقط برای مقدار توضیحی استفاده کنید. نرم افزار در شرایط خطای عادی از کار می‌افتد یا در غیر این صورت بد رفتار می‌کند.

```
/**************************************************************
 * https_client.c --- very simple HTTPS client with no error checking
 * usage: https_client servername
 **************************************************************/
#include <stdio.h>
#include <memory.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <openssl/crypto.h>
#include <openssl/x509.h>
#include <openssl/pem.h>
#include <openssl/ssl.h>
#include <openssl/err.h>
void main(int argc, char **argv)
{
 SSL *ssl;
 SSL_CTX *ctx;
 SSL_METHOD *client_method;
        X509 *server_cert;
        int sd,err;
```

```
char *str,*hostname,outbuf[4096],inbuf[4096],host_header[512];
struct hostent *host_entry;
struct sockaddr_in server_socket_address;
struct in_addr ip;

/*=========================================*/
/* (1) initialize SSL library */
/*=========================================*/

SSLeay_add_ssl_algorithms( );
client_method = SSLv2_client_method( );
SSL_load_error_strings( );
ctx = SSL_CTX_new(client_method);
 printf("(1) SSL context initialized\n\n");

/*=============================================*/
/* (2) convert server hostname into IP address */
/*=============================================*/

hostname = argv[1];
host_entry = gethostbyname(hostname);
bcopy(host_entry->h_addr, &(ip.s_addr), host_entry->h_length);
        printf("(2) '%s' has IP address '%s'\n\n", hostname, inet_ntoa(ip));
```

```
/*==================================================*/
/* (3) open a TCP connection to port 443 on server */
/*==================================================*/

sd = socket (AF_INET, SOCK_STREAM, 0);
memset(&server_socket_address, '\0', sizeof(server_socket_address));
server_socket_address.sin_family = AF_INET;
server_socket_address.sin_port = htons(443);
memcpy(&(server_socket_address.sin_addr.s_addr),
host_entry->h_addr, host_entry->h_length);
err = connect(sd, (struct sockaddr*) &server_socket_address,
sizeof(server_socket_address));
if (err < 0) { perror("can't connect to server port"); exit(1); }
printf("(3) TCP connection open to host '%s', port %d\n\n",
hostname, server_socket_address.sin_port);

/*=====================================================*/
/* (4) initiate the SSL handshake over the TCP connection */
/*=====================================================*/

ssl = SSL_new(ctx); /* create SSL stack endpoint */
SSL_set_fd(ssl, sd); /* attach SSL stack to socket */
        err = SSL_connect(ssl); /* initiate SSL handshake */
        printf("(4) SSL endpoint created & handshake completed\n\n");
```

```
/*==========================================*/
/* (5) print out the negotiated cipher chosen */
/*==========================================*/

printf("(5) SSL connected with cipher: %s\n\n", SSL_get_cipher(ssl));

/*==========================================*/
/* (6) print out the server's certificate */
/*==========================================*/

server_cert = SSL_get_peer_certificate(ssl);
printf("(6) server's certificate was received:\n\n");
str = X509_NAME_oneline(X509_get_subject_name(server_cert), 0, 0);
printf(" subject: %s\n", str);
str = X509_NAME_oneline(X509_get_issuer_name(server_cert), 0, 0);
printf(" issuer: %s\n\n", str);
/* certificate verification would happen here */
X509_free(server_cert);
```

```
/*********************************************************/
/* (7) handshake complete --- send HTTP request over SSL */
/*********************************************************/

sprintf(host_header,"Host: %s:443\r\n",hostname);
strcpy(outbuf,"GET / HTTP/1.0\r\n");
strcat(outbuf,host_header);
strcat(outbuf,"Connection: close\r\n");
strcat(outbuf,"\r\n");
err = SSL_write(ssl, outbuf, strlen(outbuf));
shutdown (sd, 1); /* send EOF to server */
printf("(7) sent HTTP request over encrypted channel:\n\n%s\n",outbuf);

/*********************************************************/
/* (8) read back HTTP response from the SSL stack */
/*********************************************************/

err = SSL_read(ssl, inbuf, sizeof(inbuf) - 1);
inbuf[err] = '\0';
printf ("(8) got back %d bytes of HTTP response:\n\n%s\n",err,inbuf);
```

```
/*********************************************/
/* (9) all done, so close connection & clean up */
/*********************************************/

SSL_shutdown(ssl);
close (sd);
SSL_free (ssl);
SSL_CTX_free (ctx);
printf("(9) all done, cleaned up and closed connection\n\n");
}
```

این مثال بر روی Sun Solaris کامپایل و اجرا می‌شود، اما نحوه عملکرد برنامه‌های SSL بر روی بسیاری از پلتفرم‌های سیستم عامل را نشان می‌دهد. کل این برنامه، از جمله تمام رمزگذاری و مدیریت کلید و گواهی، به لطف ویژگی‌های قدرتمند ارائه شده توسط OpenSSL، در یک برنامه C سه صفحه‌ای قرار می‌گیرد.

اجازه دهید بخش به بخش برنامه را مرور کنیم:

بالای برنامه شامل فایل‌های پشتیبانی مورد نیاز برای پشتیبانی از شبکه TCP و SSL است.

Section 1 با فراخوانی SSL_CTX_new زمینه محلی را ایجاد می‌کند که پارامترهای Handshake و سایر وضعیت‌های اتصال SSL را ردیابی می‌کند.

Section 2 نام میزبان ورودی (ارائه شده به عنوان آرگومان خط فرمان) را با استفاده از تابع gethostbyname یونیکس به آدرس IP تبدیل می‌کند. پلتفرم‌های دیگر ممکن است راه‌های دیگری برای ارائه این قابلیت داشته باشند.

Section 3 با ایجاد یک سوکت محلی، تنظیم اطلاعات آدرس راه دور و اتصال به سرور راه دور، اتصال TCP را به پورت ۴۴۳ روی سرور باز می‌کند.

هنگامی که اتصال TCP برقرار شد، لایه SSL را با استفاده از SSL_new و SSL_set_fd به اتصال TCP متصل می‌کنیم و با فراخوانی SSL_connect، Handshake یا دست دادن SSL را با سرور

انجام می‌دهیم. وقتی Section 4 انجام شد، ما یک کانال SSL کارآمد ایجاد شده با رمزهای انتخاب شده و گواهی‌های مبادله شده خواهیم داشت.

Section 5 ارزش رمزگذاری انبوه (Bulk-Encryption Cipher) انتخاب شده را چاپ می‌کند.

Section 6 برخی از اطلاعات موجود در گواهی X.509 را که از سرور ارسال شده است چاپ می‌کند، از جمله اطلاعات مربوط به دارنده گواهی و سازمانی که گواهی را صادر کرده است. کتابخانه OpenSSL کار خاصی با اطلاعات گواهی سرور انجام نمی‌دهد. یک برنامه SSL واقعی، مانند یک مرورگر وب، برخی از بررسی‌های عمومی گواهی را انجام می‌دهد تا مطمئن شود که به درستی امضا شده است و از میزبان مناسب ارائه شده است.

در این مرحله، اتصال SSL ما برای انتقال امن داده آماده استفاده است. در Section 7، درخواست ساده HTTP GET / HTTP/1.0 را از طریق کانال SSL با استفاده از SSL_write ارسال می‌کنیم، سپس نیمه خروجی اتصال را می‌بندیم.

در Section 8، ما پاسخ اتصال را با استفاده از SSL_read می‌خوانیم و آن را روی صفحه چاپ می‌کنیم. از آنجایی که لایه SSL تمام رمزگذاری و رمزگشایی را انجام می‌دهد، ما فقط می‌توانیم دستورات HTTP معمولی را بنویسیم و بخوانیم.

در نهایت در Section 9 پاکسازی را انجام می‌دهیم.

جهت کسب اطلاعات بیشتر در مورد کتابخانه‌های OpenSSL به http://www.openssl.org مراجعه کنید.

Executing Our Simple OpenSSL Client

در زیر خروجی سرویس گیرنده HTTP ساده ما را هنگامی که به سمت یک سرور امن اشاره می‌کنیم نشان می‌دهد. در این مورد، ما به کلاینت به صفحه اصلی کارگزاری آنلاین مورگان استنلی اشاره کردیم. لازم به ذکر است که شرکت‌های تجارت آنلاین به طور گسترده از HTTPS استفاده می‌کنند.

```
% https_client clients1.online.msdw.com

(1) SSL context initialized
(2) 'clients1.online.msdw.com' has IP address '63.151.15.11'
(3) TCP connection open to host 'clients1.online.msdw.com', port 443
(4) SSL endpoint created & handshake completed
(5) SSL connected with cipher: DES-CBC3-MD5
(6) server's certificate was received:

subject: /C=US/ST=Utah/L=Salt Lake City/O=Morgan Stanley/OU=Online/CN=clients1.online.msdw.com
issuer: /C=US/O=RSA Data Security, Inc./OU=Secure Server Certification Authority

(7) sent HTTP request over encrypted channel:

GET / HTTP/1.0
Host: clients1.online.msdw.com:443
Connection: close

(8) got back 615 bytes of HTTP response:

HTTP/1.1 302 Found
Date: Sat, 09 Mar 2002 09:43:42 GMT
Server: Stronghold/3.0 Apache/1.3.14 RedHat/3013c (Unix) mod_ssl/2.7.1 OpenSSL/0.9.6
Location: https://clients.online.msdw.com/cgi-bin/ICenter/home
Connection: close
Content-Type: text/html; charset=iso-8859-1
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>302 Found</TITLE>
</HEAD><BODY>
<H1>Found</H1>
The document has moved <A HREF="https://clients.online.msdw.com/cgi-bin/ICenter/
home">here</A>.<P>
<HR>
<ADDRESS>Stronghold/3.0Apache/1.3.14 RedHat/3013c Server at clients1.online.msdw.com
Port 443</ADDRESS>
</BODY></HTML>

(9) all done, cleaned up and closed connection
```

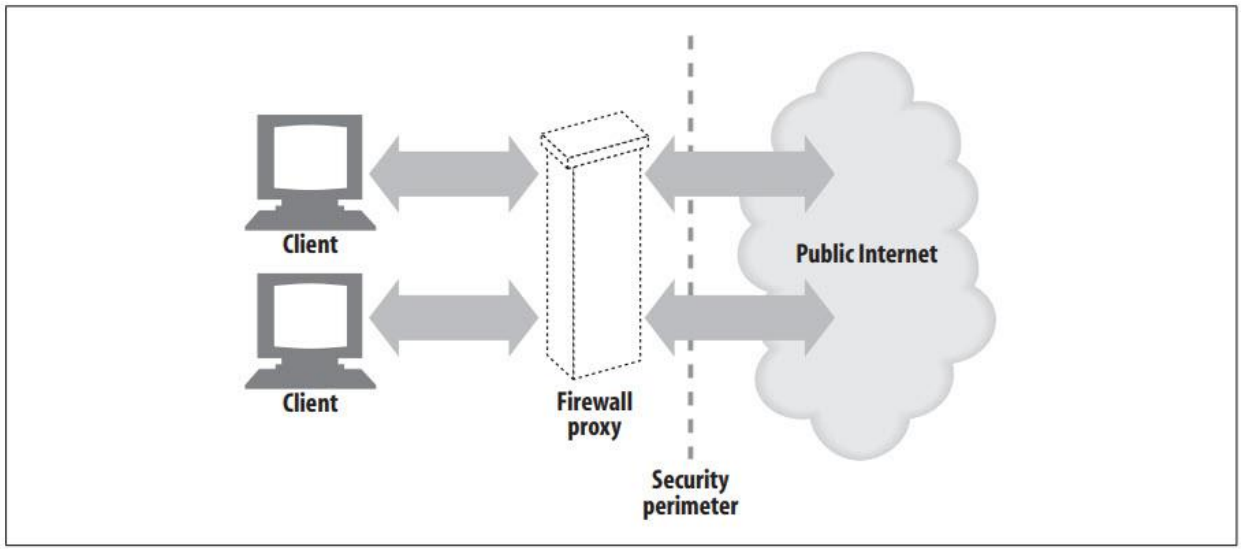
به محض تکمیل چهار بخش اول، کلاینت یک اتصال SSL باز دارد. سپس می‌تواند وضعیت اتصال و پارامترهای انتخابی را جویا شود و گواهی‌های سرور را بررسی کند.

در این مثال، کلاینت و سرور در مورد رمزگذاری انبوه DES-CBC3-MD5 مذاکره کردند. همچنین می‌توانید ببینید که گواهی سایت سرور متعلق به سازمان "Morgan Stanley" در "سالت لیک سیتی، یوتا، ایالات متحده آمریکا" است. این گواهی توسط RSA Data Security اعطا شده است و نام میزبان "clients1.online.msdw.com" است که با درخواست ما مطابقت دارد.

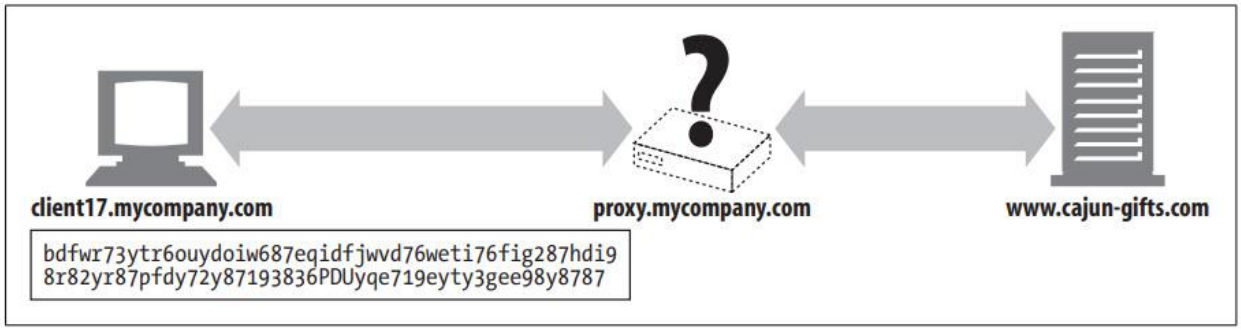
هنگامی که کانال SSL ایجاد شد و کلاینت در مورد گواهی سایت احساس راحتی کرد، درخواست HTTP خود را از طریق کانال امن ارسال می‌کند. در مثال ما، کلاینت یک درخواست HTTP ساده «GET / HTTP/1.0» ارسال می‌کند و یک پاسخ 302 Redirect دریافت می‌کند و از کاربر درخواست می‌کند که یک URL متفاوت واکشی کند.

Tunneling Secure Traffic Through Proxies

کلاینت‌ها اغلب از سرورهای پروکسی وب برای دسترسی به سرورهای وب از طرف خود استفاده می‌کنند. به عنوان مثال، بسیاری از شرکت‌ها یک پروکسی را در محیط امنیتی شبکه شرکت و اینترنت عمومی قرار می‌دهند (شکل زیر). پروکسی تنها وسیله‌ای است که توسط روترهای فایروال، مجاز به تبادل ترافیک HTTP است و ممکن است از بررسی ویروس یا سایر کنترل‌های محتوا استفاده کند.



اما هنگامی که کلاینت با استفاده از کلید عمومی سرور شروع به رمزگذاری داده‌ها در سرور می‌کند، پروکسی دیگر توانایی خواندن هدر HTTP را ندارد! و اگر پروکسی نتواند هدر HTTP را بخواند، نمی‌داند درخواست را کجا ارسال کند (شکل زیر).



برای اینکه HTTPS با پراکسی‌ها کار کند، چند تغییر لازم است تا به پراکسی بگوید کجا باید متصل شود. یکی از تکنیک‌های محبوب پروتکل HTTPS SSL Tunneling است.

با استفاده از پروتکل HTTPS Tunneling، کلاینت ابتدا میزبان امن و پورتی را که می‌خواهد به آن متصل شود را به پروکسی می‌گوید. این کار را قبل از شروع رمزگذاری در متن ساده انجام می‌دهد، بنابراین پروکسی می‌تواند این اطلاعات را بخواند.

HTTP برای ارسال اطلاعات نقطه پایانی متن ساده از یک متد افزودنی جدید به نام CONNECT استفاده می‌شود. متد CONNECT به پراکسی می‌گوید که یک اتصال به هاست و شماره پورت مورد نظر را باز کند و پس از انجام این کار، داده‌ها را مستقیماً بین کلاینت و سرور تونل کند. متد CONNECT یک دستور متنی تک خطی است که نام میزبان و پورت سرور مبدا امن را ارائه می‌کند که با یک دو نقطه از هم جدا شده‌اند. host:port با یک فاصله و یک رشته نسخه HTTP و همچنین CRLF دنبال می‌شود. پس از آن یک سری خط هدر درخواست HTTP صفر یا بیشتر بوده و به دنبال آن یک خط خالی وجود دارد. پس از خط خالی، اگر Handshake برای برقراری اتصال موفقیت آمیز بود، انتقال داده‌های SSL می‌تواند آغاز شود. به عنوان مثال:

CONNECT home.netscape.com:443

HTTP/1.0 User-agent: Mozilla/1.1N

<raw SSL-encrypted data would follow here…>

پس از خط خالی در درخواست، کلاینت منتظر پاسخ از طرف پروکسی خواهد بود. پروکسی درخواست را ارزیابی می‌کند و مطمئن می‌شود که معتبر است و کاربر مجاز به درخواست چنین اتصالی است. اگر همه چیز مرتب باشد، پروکسی به سرور مقصد متصل می‌شود و در صورت موفقیت آمیز بودن، یک پاسخ 200 Connection Established را برای کلاینت ارسال می‌کند.

HTTP/1.0 200 Connection established

Proxy-agent: Netscape-Proxy/1.1

For More Information

امنیت و رمزنگاری موضوعات بسیار مهم و بسیار پیچیده‌ای هستند. اگر می‌خواهید درباره امنیت HTTP، رمزنگاری دیجیتال، گواهی‌های دیجیتال و زیرساخت کلید عمومی اطلاعات بیشتری کسب کنید، در اینجا چند نقطه شروع وجود دارد.

HTTP Security

Web Security, Privacy & Commerce/ Simson Garfinkel, O'Reilly & Associates, Inc

http://www.ietf.org/rfc/rfc2818.txt

http://www.ietf.org/rfc/rfc2817.txt

SSL and TLS

http://www.ietf.org/rfc/rfc2246.txt

http://developer.netscape.com/docs/manuals/security/sslin/contents.htm

http://www.netscape.com/eng/ssl3/draft302.txt

http://developer.netscape.com/tech/security/ssl/howitworks.html

http://www.openssl.org

Public-Key Infrastructure

http://www.ietf.org/html.charters/pkix-charter.html

http://www.ietf.org/rfc/rfc2459.txt

Digital Cryptography

Applied Cryptography/Bruce Schneier, John Wiley & Sons

The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography/Simon Singh, Anchor Books