



نام و نام خانوادگی: تینا توکلی

شماره دانشجویی: ۹۹۲۲۷۶۲۲۲۰

شماره تمرین: ۰۴

(۱) بررسی زیرساخت کلید عمومی (PKI)

زیرساخت کلید عمومی (PKI) سیستمی است که از رمزنگاری نامتقارن برای تأمین امنیت ارتباطات دیجیتال استفاده می‌کند. این سیستم از اجزای مختلفی تشکیل شده است که با یکدیگر کار می‌کنند تا هویت کاربران را تأیید، داده‌ها را رمزگذاری و امضاهای دیجیتال را ایجاد کنند.

اجزای اصلی PKI عبارتند از:

- مرکز صدور گواهی (CA): سازمانی است که گواهی‌های دیجیتال را صادر می‌کند. گواهی‌های دیجیتال حاوی اطلاعاتی مانند کلید عمومی کاربر، هویت کاربر و تاریخ انقضا گواهی هستند.
- دفتر ثبت نام: سازمانی است که هویت متقاضیان گواهی دیجیتال را تأیید می‌کند.
- ذخیره‌سازی کلید: مکانی برای ذخیره امن کلیدهای خصوصی کاربران.
- نرم‌افزار رمزنگاری: نرم‌افزاری که برای رمزگذاری و رمزگشایی داده‌ها با استفاده از کلیدهای رمزنگاری استفاده می‌شود.

مزایای استفاده از PKI عبارتند از:

- محرمانگی: از رمزنگاری برای محافظت از داده‌ها در برابر دسترسی غیرمجاز استفاده می‌کند.
- احراز هویت: هویت کاربران را تأیید می‌کند.
- عدم انکار: اطمینان می‌دهد که فرستنده پیام نمی‌تواند ارسال پیام را انکار کند.
- integridad de datos (یکپارچگی پیام): اطمینان می‌دهد که داده‌ها در حین انتقال دستکاری نشده‌اند.
- برخی از چالش‌های استفاده از PKI عبارتند از:
- مدیریت کلید: مدیریت ایمن کلیدهای خصوصی کاربران ضروری است.
- اعتماد به CA: کاربران باید به CA که گواهی‌های دیجیتال را صادر می‌کند اعتماد کنند.
- هزینه: پیاده‌سازی و نگهداری PKI می‌تواند پرهزینه باشد.

در اینجا چند نکته برای بررسی PKI خود آورده شده است:

- از یک CA معتبر استفاده کنید: CAی را انتخاب کنید که به آن اعتماد دارید و سابقه اثبات شده‌ای در صدور گواهی‌های دیجیتال امن دارد.
- از کلیدهای قوی استفاده کنید: از کلیدهای رمزنگاری با طول کافی استفاده کنید تا از آنها در برابر حملات هک محافظت کنید.
- کلیدهای خود را به صورت ایمن ذخیره کنید: کلیدهای خصوصی خود را در مکانی امن ذخیره کنید که فقط شما به آنها دسترسی دارید.
- نرم‌افزار رمزنگاری خود را به‌روز نگه دارید: از آخرین نسخه نرم‌افزار رمزنگاری استفاده کنید تا از آسیب‌پذیری‌های امنیتی شناخته‌شده محافظت کنید.
- به طور مرتب PKI خود را ممیزی کنید: PKI خود را به طور مرتب برای بررسی وجود هرگونه آسیب‌پذیری امنیتی ممیزی کنید (ذخیره سابقه).

۲) گواهی‌های دیجیتال:

گواهی‌های دیجیتال ستون فقرات امنیت زیرساخت کلید عمومی (PKI) هستند. آنها هویت افراد و دستگاه‌ها را تأیید می‌کنند و امکان رمزگذاری و امضای دیجیتال امن را فراهم می‌کنند. در حالی که گواهی‌های X.۵۰۹ استاندارد صنعت برای چندین دهه بوده‌اند، انواع جدیدی از گواهی‌ها در حال ظهور هستند که مزایای امنیتی و انعطاف‌پذیری بیشتری را ارائه می‌دهند. در اینجا به برخی از انواع رایج گواهی‌های دیجیتال و نحوه مقایسه آنها با X.۵۰۹ می‌پردازیم:

۱. گواهی‌های X.509:

- قدیمی‌ترین و رایج‌ترین نوع گواهی دیجیتال.
- توسط استانداردهای ITU-T X.۵۰۹ تعریف شده است.
- برای طیف گسترده‌ای از برنامه‌ها، از جمله تجارت الکترونیکی، بانکداری آنلاین و ایمیل امن استفاده می‌شود.
- مزایا: به طور گسترده شناخته شده و پشتیبانی می‌شوند، ساختار نسبتاً ساده‌ای دارند.
- معایب: می‌توانند پیچیده باشند و مدیریت آنها دشوار باشد، ممکن است برای برخی برنامه‌ها به اندازه کافی انعطاف‌پذیر نباشند.

۲. گواهی‌های Elliptic Curve (ECC):

- از رمزنگاری منحنی بیضوی استفاده می‌کنند که امن‌تر و کارآمدتر از رمزنگاری کلید عمومی سنتی است.
- به طور فزاینده‌ای برای برنامه‌هایی که به عملکرد و امنیت بالا نیاز دارند، مانند اینترنت اشیا (IoT) و بلاک چین استفاده می‌شوند.
- مزایا: امنیت و کارایی بالا، مقیاس‌پذیری خوب.

- معایب: پذیرش کمتری نسبت به گواهی های X.۵۰۹ دارند، ممکن است با برخی از سیستم های قدیمی سازگار نباشند.

۳. گواهی های TLS/SSL:

- به طور خاص برای تأمین امنیت ارتباطات وب طراحی شده اند.
- برای محافظت از حریم خصوصی و امنیت داده ها در هنگام مرور وب، بانکداری آنلاین و موارد دیگر استفاده می شود.
- مزایا: نصب و پیکربندی آسان، به طور گسترده توسط مرورگرها و وب سرورها پشتیبانی می شوند.
- معایب: فقط برای ارتباطات وب قابل استفاده هستند، به اندازه سایر انواع گواهی ها انعطاف پذیر نیستند.

۴. گواهی های SSH:

- برای تأمین امنیت اتصالات SSH استفاده می شود که برای مدیریت از راه دور سرورها و دستگاه های شبکه استفاده می شود.
- احراز هویت قوی و رمزگذاری را برای اتصالات SSH فراهم می کند.
- مزایا: امنیت بالا برای اتصالات از راه دور، به طور گسترده توسط مدیران سیستم استفاده می شود.
- معایب: پیکربندی آنها می تواند پیچیده باشد، به اندازه سایر انواع گواهی ها انعطاف پذیر نیستند.

۵. گواهی های SMIME:

- برای امضای دیجیتال و رمزگذاری ایمیل استفاده می شود.
- تأیید هویت فرستنده و یکپارچگی ایمیل را فراهم می کند.
- مزایا: امنیت و قابلیت اطمینان را برای ارتباطات ایمیل افزایش می دهد.
- معایب: استفاده از آنها ممکن است برای برخی از کاربران دشوار باشد، به طور گسترده توسط همه ارائه دهندگان ایمیل پشتیبانی نمی شوند.

انتخاب نوع مناسب گواهی دیجیتال:

- بهترین نوع گواهی دیجیتال برای شما به نیازهای خاص شما بستگی دارد.
- سطح امنیتی مورد نیاز خود را در نظر بگیرید. اگر به امنیت بالایی نیاز دارید، ممکن است بخواهید از گواهی های ECC یا TLS/SSL استفاده کنید.
- برنامه ای را که برای آن به گواهی نیاز دارید در نظر بگیرید. اگر برای مرور وب به گواهی نیاز دارید، گواهی TLS/SSL بهترین انتخاب است. اگر برای امضای دیجیتال ایمیل به گواهی نیاز دارید، گواهی SMIME بهترین انتخاب است.
- با الزامات سازگاری بررسی کنید. مطمئن شوید که نوع گواهی که انتخاب می کنید با سیستم ها و برنامه های شما سازگار است.
- هزینه را در نظر بگیرید. قیمت گواهی ها می تواند بسته به نوع و ارائه دهنده متفاوت باشد.

۳) نحوه ی تأیید اعتبار گواهی SSL:

تأیید اعتبار گواهی SSL فرآیندی است که به شما اطمینان می دهد گواهی معتبر، صادر شده توسط یک مرجع صدور گواهی (CA) قابل اعتماد و مرتبط با نام دامنه وب سایتی است که به آن مراجعه می کنید. مراحل کلیدی involved in the process of validating an SSL certificate:

۱. بررسی زنجیره گواهی:

- مرورگر شما زنجیره گواهی را که شامل گواهی خود سایت و همچنین گواهی های میانی CA های صادر کننده آن است، بازیابی می کند.
- هر گواهی باید توسط یک CA معتبر امضا شده باشد و تاریخ صدور آن معتبر باشد.
- هر گونه گواهی نامعتبر یا منقضی شده در زنجیره، باعث عدم اعتبار گواهی کل می شود.

۲. بررسی نام دامنه:

- نام دامنه ذکر شده در گواهی باید با نام دامنه وب سایتی که به آن مراجعه می کنید مطابقت داشته باشد.
- این امر تضمین می کند که گواهی برای وب سایت مورد نظر شما صادر شده است و نه برای وب سایت دیگری.

۳. بررسی وضعیت CRL:

- لیست لغو گواهی (CRL) فهرستی از گواهی های SSL است که به دلیل ناامنی یا سایر مسائل، توسط CA ها باطل شده اند.
- مرورگر شما باید CRL صادر کننده را بررسی کند تا ببیند آیا گواهی در لیست سیاه قرار گرفته است یا خیر.

۴. بررسی تاریخ انقضا:

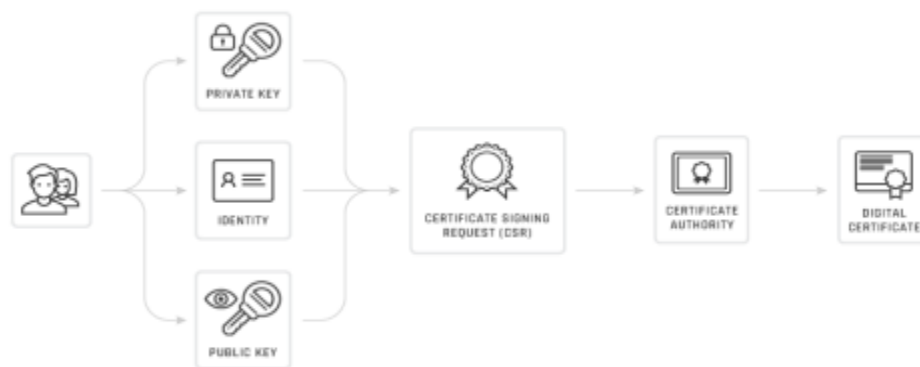
- تاریخ انقضا گواهی باید در آینده باشد.
- گواهی های منقضی شده دیگر معتبر نیستند و باید تمدید شوند.

ابزارهای تأیید اعتبار SSL:

علاوه بر بررسی های داخلی مرورگر، می توانید از ابزارهای مختلف آنلاین برای تأیید اعتبار گواهی SSL یک وب سایت استفاده کنید. برخی از ابزارهای محبوب عبارتند از:

- <https://www.sslshopper.com> :SSL Checker
- <https://www.ssllabs.com/ssltest/index.html> :Qualys SSL Server Test
- <https://www.digicert.com/help> :Digicert SSL Checker

تأیید اعتبار گواهی SSL بخش مهمی از اطمینان از امنیت آنلاین است.



مراحل اعتبار سنجی گواهی SSL در شکل بالا به اینگونه است:

۱. درخواست گواهی SSL:

- مالک وبسایت باید از یک مرکز صدور گواهی معتبر (CA) درخواست گواهی SSL کند.
- CA هویت مالک وبسایت را تأیید کرده و اطلاعات مربوط به وبسایت را جمع‌آوری می‌کند.

۲. صدور گواهی SSL:

- در صورت تأیید هویت و صحت اطلاعات، CA گواهی SSL را صادر می‌کند.
- گواهی SSL شامل اطلاعاتی مانند نام دامنه، کلید عمومی و تاریخ انقضا است.

۳. نصب گواهی SSL:

- مالک وبسایت باید گواهی SSL را بر روی سرور وب خود نصب کند.
- نصب گواهی SSL نیاز به پیکربندی سرور وب دارد.

۴. بررسی گواهی SSL توسط مرورگر:

- هنگامی که کاربری از وبسایت بازدید می‌کند، مرورگر گواهی SSL ارائه شده توسط سرور را بررسی می‌کند.
- مرورگر زنجیره گواهی، نام دامنه، وضعیت CRL و تاریخ انقضا را بررسی می‌کند.

۵. نمایش نماد قفل در مرورگر:

- اگر گواهی SSL معتبر باشد، مرورگر نماد قفل را در نوار آدرس نمایش می‌دهد.
- این نماد به کاربر نشان می‌دهد که اتصال به وبسایت امن است.

۶. تأیید اطلاعات گواهی SSL:

- کاربر می‌تواند با کلیک بر روی نماد قفل، اطلاعات مربوط به گواهی SSL را مشاهده کند.
- این اطلاعات شامل نام دامنه، CA صادر کننده، تاریخ انقضا و اثر انگشت گواهی است.

البته

- گواهی‌های SSL باید به طور منظم تمدید شوند تا تاریخ انقضای آنها به پایان نرسد.
- استفاده از گواهی‌های SSL معتبر از وبسایت‌های معتبر CA ضروری است.
- عدم وجود نماد قفل در مرورگر یا نمایش اخطار عدم اعتبار گواهی SSL نشان‌دهنده ناامن بودن وبسایت است.

مراحل در تصویر:

تصویر ارائه شده مراحل زیر از فرآیند اعتبار سنجی گواهی SSL را نشان می‌دهد:

- **A:** مالک وبسایت از CA درخواست گواهی SSL می‌کند.
- **B:** CA هویت مالک وبسایت را تأیید می‌کند.
- **C:** CA گواهی SSL را صادر می‌کند.
- **D:** مالک وبسایت گواهی SSL را بر روی سرور وب نصب می‌کند.
- **E:** مرورگر گواهی SSL را بررسی می‌کند.
- **F:** مرورگر نماد قفل را در نوار آدرس نمایش می‌دهد.

همانطور که در بالا هم ذکر شده تأیید اعتبار گواهی SSL اقدامی ضروری برای حفاظت از اطلاعات و حفظ امنیت در دنیای دیجیتال است. با اتخاذ این رویکرد، می‌توانید از صحت و اعتبار هویت وبسایت‌ها اطمینان حاصل کنیم.

۴) پیاده سازی PKI از پایه بدون استفاده از نرم افزارهای PKI موجود:

مراحل:

۱. طراحی و پیاده سازی رمزنگاری نامتقارن:

- الگوریتم‌های رمزنگاری نامتقارن مناسب مانند RSA یا Elliptic Curve Cryptography (ECC) را انتخاب کنید.
- توابع لازم برای رمزگذاری، رمزگشایی، امضا و تأیید امضا را با استفاده از یک زبان برنامه نویسی مناسب مانند Python یا Java پیاده سازی کنید.

۲. ایجاد ساختار گواهی دیجیتال:

- ساختاری برای گواهی دیجیتال تعریف کنید که شامل اطلاعاتی مانند نام دامنه، کلید عمومی، تاریخ انقضا و امضای CA باشد.
- تابعی برای ایجاد گواهی دیجیتال بر اساس ساختار تعریف شده پیاده سازی کنید.

۳. پیاده سازی مرجع صدور گواهی (CA):

- یک کلید خصوصی برای CA ایجاد کنید.
- تابعی برای صدور گواهی دیجیتال برای کاربران نهایی با استفاده از کلید خصوصی CA و ساختار گواهی دیجیتال پیاده سازی کنید.

۴. پیاده سازی مرجع ثبت (RA):

- یک رابط کاربری برای دریافت درخواست‌های گواهی از کاربران نهایی ایجاد کنید.
- تابعی برای تأیید هویت کاربران نهایی و صحت اطلاعات درخواست گواهی پیاده سازی کنید.
- درخواست‌های گواهی تأیید شده را به CA ارسال کنید.

۵. پیاده سازی لیست لغو گواهی (CRL):

- یک ساختار برای ذخیره اطلاعات گواهی‌های ابطال شده مانند شناسه گواهی و تاریخ ابطال تعریف کنید.

- تابعی برای ابطال گواهی دیجیتال و اضافه کردن آن به CRL پیاده سازی کنید.

- CRL را به طور منظم به روز کنید.

۶. پیاده سازی تأیید اعتبار گواهی:

- تابعی برای بررسی اعتبار گواهی دیجیتال با استفاده از CRL و کلید عمومی CA پیاده سازی کنید.

- در صورت معتبر بودن گواهی، اطلاعات آن را نمایش دهید.

- در صورت نامعتبر بودن گواهی، دلیل عدم اعتبار را نمایش دهید.

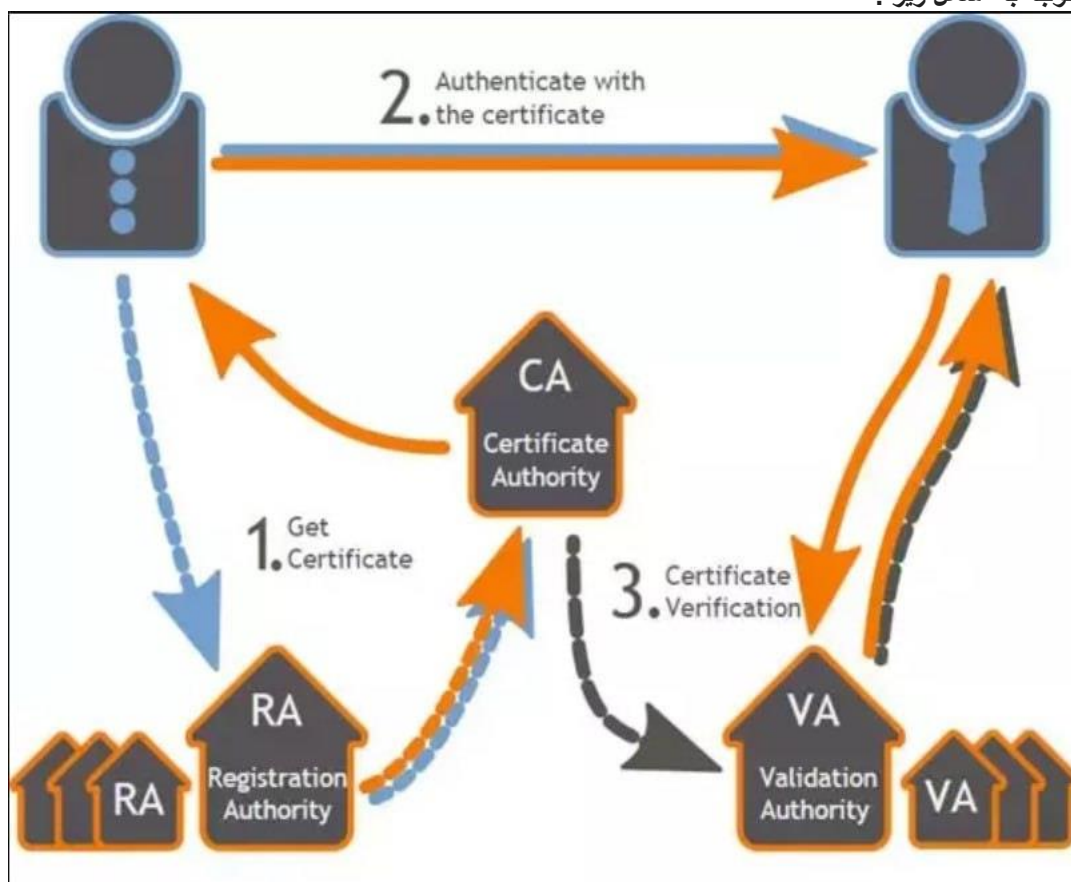
۷. پیاده سازی کانال‌های امن TLS/SSL:

- از TLS/SSL برای محافظت از ارتباطات بین CA و RA و بین CA و کاربران نهایی استفاده کنید.

- گواهی‌های سرور را برای CA و RA پی‌گیری کنید.

- تنظیمات TLS/SSL را در مرورگرهای وب و کلاینت‌های دیگر پی‌گیری کنید.

مراحل با توجه به شکل زیر :



در ابتدا سرور درخواستی به RA می‌دهد مبنی بر دریافت سرتفیکیت و RA این درخواست را از CA می‌گیرد تا سرتفیکیت را دریافت کند و وقتی دریافت شد (سرتفیکیت امضا شده توسط کلید خصوصی CA)، CA آن را برای

سرور میفرستد

حال سرور سرتیفیکیت خود را برای client خود ارسال میکند .

Client سرتیفیکیت گرفته شده را برای VA میفرستد تا از صحت آن باخبر شود و VA این کار با ارتباط با CA انجام میدهد.

پیاده سازی کلاس RA:

راه اندازی اولیه ('init'):

- `self.name = name`: در سازنده کلاس (init)، آرگومان ارائه شده "name" به ویژگی نمونه "self.name" اختصاص داده می شود. این ویژگی نام شی مرجع ثبت را ذخیره می کند.

تولید یک جفت کلید (روش "تولید_جفت_کلید"):

- `return rsa.generate_private_key(...)`: این روش یک جفت کلید RSA جدید را با استفاده از کتابخانه

`'cryptography'` ایجاد می کند. پارامترهای خاص مورد استفاده عبارتند از:

- `public_exponent=65537`: این یک مقدار توان عمومی رایج است که در رمزنگاری RSA استفاده می شود.

- `key_size=2048`: اندازه کلید را بر حسب بیت مشخص می کند که یک انتخاب استاندارد برای امنیت در

بسیاری از برنامه ها است.

- `backend=default_backend()`: این نشان می دهد که پشتیبان رمزنگاری پیش فرض باید برای تولید کلید

استفاده شود.

کلید خصوصی تولید شده توسط روش برگردانده می شود.

ایجاد درخواست امضای گواهی (CSR) (روش «تولید_گواهی_امضای_درخواست»):

- `subject = x509.Name(...)`: این خط با استفاده از ماژول `'x509'` یک موضوع برای CSR ایجاد می کند.

موضوع شناسایی نهاد (در این مورد، مرجع ثبت) است که گواهی برای آن صادر می شود. در اینجا، فقط نام رایج

(`self.name`) تنظیم شده است.

- `builder = x509.CertificateSigningRequestBuilder()`: یک شی سازنده درخواست امضای گواهی با استفاده

از ماژول `'x509'` ایجاد می شود. این شیء برای ساخت CSR استفاده خواهد شد.

- `builder = builder.subject_name(subject)`: موضوعی که قبلا ایجاد شده است به عنوان موضوع CSR تنظیم

می شود.

- `builder = builder.add_extension(...)`: یک پسوند به CSR اضافه می شود. این پسوند مشخص می کند که گواهی درخواستی گواهینامه CA نیست (`ca=False`) و محدودیت طول مسیر خاصی ندارد (`path_length=None`). آرگومان `critical=True` نشان می دهد که این پسوند برای اعتبار گواهی ضروری است. - `csr = builder.sign(...)`: CSR در نهایت با استفاده از `ra_private_key` (کلید خصوصی مرجع ثبت)، الگوریتم هش SHA-256 و پیش فرض رمزنگاری امضا شده است. سپس CSR امضا شده برگردانده می شود.

ارسال CSR به CA و دریافت گواهی :-

- `csr_pem = csr.public_bytes(...)`: این خط شی CSR را به یک رشته بایت کدگذاری شده با PEM تبدیل می کند. PEM (Privacy Enhanced Mail) یک قالب رایج برای ذخیره گواهی ها و CSR ها است. - `csr = x509.load_pem_x509_csr(...)`: CSR رمزگذاری شده با PEM سپس برای پردازش بیشتر با استفاده از ماژول `x509` در یک شی CSR بارگذاری می شود. - `ca = CertificateAuthority()`: در اینجا نمونه ای از کلاس `CertificateAuthority` ایجاد می شود. با این حال، کد ارائه شده این کلاس یا روش «تولید گواهی» آن را تعریف نمی کند. این روش احتمالاً مسئول تعامل با یک مرجع گواهی برای دریافت گواهی امضا شده بر اساس CSR است.

پیاده سازی کلاس CA:

کلاس CertificateAuthority یک کلاس پایتون است که برای ایجاد و مدیریت یک (CA) Authority Certificate به کار می رود.

دارای متدهای زیر است:

- **init**: این متد سازنده (constructor) کلاس است که در زمان ایجاد یک آبجکت از این کلاس اجرا می شود.
 - یک آرگومان اختیاری `name` دارد که به صورت پیش فرض مقدار "My CA" را به `name attribut` اختصاص می دهد.
 - بررسی می کند که آیا فایل `ca-cert.pem` وجود دارد یا خیر. اگر وجود داشته باشد، گواهی CA موجود را از این فایل بارگذاری می کند.
 - بررسی می کند که آیا فایل `server-private-key.pem` وجود دارد یا خیر. اگر وجود داشته باشد، کلید خصوصی CA موجود را از این فایل بارگذاری می کند.
- **generate_key_pair**: این متد یک جفت کلید (`private key`, `public key`) از نوع RSA با سایز ۲۰۴۸ بیت و `exponent` عمومی ۶۵۵۳۷ تولید می کند.
- **generate_self_certificate**: این متد یک گواهی self-signed برای CA با مشخصات زیر ایجاد می کند:

- موضوع (subject) گواهی، نام CA را که از آرگومان name در سازنده کلاس مقداردهی شده، قرار می دهد.
- صادر کننده (issuer) گواهی نیز همین موضوع را قرار می دهد (یعنی CA خودش برای خودش گواهی صادر می کند).
- کلید عمومی گواهی را از private_key که به عنوان آرگومان ورودی داده می شود، استخراج می کند.
- یک شماره سریال تصادفی برای گواهی ایجاد می کند.
- تاریخ اعتبار گواهی را از زمان حال تا ۳۶۵۰ روز دیگر (تقریباً ۱۰ سال) تنظیم می کند.
- یک extension از نوع BasicConstraints به گواهی اضافه می کند که نشان می دهد این گواهی برای امضای گواهی های دیگر (CA است) و طول مسیر صدور گواهی محدودیتی ندارد (None).
- در نهایت، گواهی را با استفاده از private key امضا می کند و آن را در فایل ca-cert.pem ذخیره می کند.
- **generate_certificate**: این متد یک گواهی برای یک موجودیت (entity) بر اساس CSR (Certificate Signing Request) آن موجودیت ایجاد می کند:
 - موضوع (subject) گواهی را از subject موجود در CSR استخراج می کند.
 - صادر کننده (issuer) گواهی را از subject گواهی CA (که در ca_cert attribut ذخیره شده) قرار می دهد.
 - کلید عمومی گواهی را از CSR استخراج می کند.
 - یک شماره سریال تصادفی برای گواهی ایجاد می کند.
 - تاریخ اعتبار گواهی را از زمان حال تا ۳۶۵ روز دیگر (تقریباً ۱ سال) تنظیم می کند.
 - شبیه به متد generate_self_certificate یک extension از نوع BasicConstraints به گواهی اضافه می کند.
 - در نهایت، گواهی را با استفاده از private key متعلق به CA امضا می کند و آن را برمیگرداند.
- در کل این کلاس، امکان ایجاد یک CA، تولید جفت کلید برای CA، صدور گواهی self-signed برای CA و همچنین صدور گواهی برای موجودیت های دیگر بر اساس CSR آنها را فراهم می کند.

پیاده سازی کلاس CRL:

این کلاس یک لیست ابطال گواهی (CRL) را مدیریت می کند.

راه اندازی اولیه (روش __init__):

```
- def __init__(self):
    این متد سازنده کلاس را تعریف می کند که هر زمان که یک شیء جدید
    CertificateRevocationList ایجاد شود فراخوانی می شود.
```

- `self.revoked_certs` = `{}` این خط یک متغیر نمونه به نام `revoked_certs` در داخل شی ایجاد می کند. این به عنوان یک فرهنگ لغت خالی `{}` برای ذخیره گواهی های باطل شده راه اندازی شده است. فرهنگ لغت از شماره سریال گواهی به عنوان کلید و تاریخ و زمان ابطال به عنوان مقدار استفاده می کند.

لغو گواهینامه (روش `revoke_certificate`):

- `def revoke_certificate(self, Certificate):` این روش یک شیء `Certificate` را به عنوان ورودی می گیرد که احتمالاً حاوی اطلاعاتی در مورد گواهی است که باید باطل شود.

- `serial_number = Certificate.serial_number` این خط شماره سریال را از شی گواهی ارائه شده استخراج می کند. شماره سریال یک شناسه منحصر به فرد برای هر گواهی است.

- `self.revoked_certs[serial_number] = datetime.utcnow()` این خط شماره سریال گواهی ابطال شده را به فرهنگ لغت `revoked_certs` به همراه تاریخ و زمان فعلی به دست آمده با استفاده از `datetime.utcnow()` اضافه می کند. این نشان دهنده زمانی است که گواهی باطل شده است.

تولید CRL (روش `generate_crl`):

- `def generate_crl(self, ca_key, ca_cert):` این روش CRL واقعی را تولید می کند که یک لیست امضا شده از گواهی های باطل شده است. دو استدلال نیاز دارد:

- `ca_key`: این کلید خصوصی مرجع صدور گواهی (CA) است که گواهی ها را صادر کرده است.

- `ca_cert`: این خود گواهی CA است.

- اطلاعات صادرکننده:

- `issuer = x509.Name([x509.NameAttribute(x509.NameOID.COMMON_NAME, "My `CA"))` این بلوک با استفاده از کتابخانه `x509` یک شی صادرکننده ایجاد می کند. صادرکننده CA را که CRL را صادر کرده است شناسایی می کند. در اینجا، با استفاده از ویژگی نام مشترک، روی "CA" تنظیم شده است.

- CRL Builder:

- `builder = x509.CertificateRevocationListBuilder()` این خط یک شی سازنده با استفاده از کتابخانه `x509` برای ساخت CRL ایجاد می کند.

- `builder = builder.issuer_name(issuer)` این خط اطلاعات صادر کننده را برای CRL با استفاده از شیء "صادرکننده" ایجاد شده قبلی تنظیم می کند.

- `builder = builder.last_update(datetime.utcnow())` این خط فیلد 'CRL' last_update را روی تاریخ و زمان فعلی تنظیم می کند، که نشان می دهد آخرین بار چه زمانی CRL به روز شده است.

- `builder = builder.next_update(datetime.utcnow() + timedelta(days=30))` این خط فیلد

"next_update" را تنظیم می کند و مشخص می کند که CRL بعدی چه زمانی صادر می شود. در اینجا، از تاریخ و زمان فعلی روی ۳۰ روز تنظیم شده است.

- افزودن گواهی های باطل شده:

- حلقه "for" از طریق فرهنگ لغت "Revoked_certs" تکرار می شود:

- `builder.add_revoked_certificate(...)` این خط با استفاده از روش

"add_revoked_certificate" اطلاعات هر گواهی ابطال شده را به سازنده CRL اضافه می کند.

- `x509.RevokedCertificateBuilder` این یک سازنده برای ورودی های گواهی ابطال شده ایجاد می

کند.

- `serial_number(serial_number)` شماره سریال گواهی ابطال شده را تنظیم می کند.

- `revocation_date(revocation_date)` تاریخ و زمان ابطال گواهی را تعیین می کند.

- `build(default_backend())` این ورودی گواهی ابطال شده را با استفاده از باطن پیش فرض ایجاد می

کند.

- امضای CRL:

- `crl = builder.sign(private_key=ca_key, algorithm=hashes.SHA256(),`

`backend=default_backend())` این خط CRL کامل را با استفاده از کلید خصوصی ca_key (`CA`)، الگوریتم

هش SHA-256 امضا می کند. (`hashes.SHA256()`)، و باطن پیش فرض. امضا، اعتبار CRL را تضمین می کند و از دستکاری جلوگیری می کند.

- برگرداندن CRL:

- `return crl` این خط شی CRL تولید شده و امضا شده را برمی گرداند.

این کد کلاسی را برای مدیریت فهرست ابطال گواهی (CRL) تعریف می کند. این اجازه می دهد تا گواهی ها را به لیست ابطال اضافه کنید، زمان های ابطال آنها را ردیابی کنید، و یک CRL امضا شده با استفاده از خصوصی CA ایجاد کنید.

The screenshot shows a Python IDE with a project named '4-security'. The file explorer on the left lists various files including certificates and keys. The main editor shows the code in 'CA.py', which defines a Certificate Authority and a Registration Authority. The output window at the bottom shows the command executed and the successful completion message.

```

146 # Example Usage
147 if __name__ == "__main__":
148     ca = CertificateAuthority()
149     ca_key = ca.generate_key_pair()
150     ca_cert = ca.generate_self_certificate(ca_key)
151
152     ra = RegistrationAuthority()
153     ra_key = ra.generate_key_pair()
154     csr = ra.generate_certificate_signing_request(ra_key)
155
156     signed_cert = ra.send_csr_to_ca_and_get_certificate(csr, ca_cert)
157     print('CA and RA and CRL done successfully')
158
159 if __name__ == "__main__":

```

```

Run: C:\hw4-security\venv\Scripts\python.exe C:\hw4-security\CA.py
CA and RA and CRL done successfully
Process finished with exit code 0

```

تصویر ۱ - خروجی در ارتباط با پیاده سازی بخش RA,CA,CRL

پیاده سازی کلاس VA:

این کد تابعی را برای تأیید اعتبار یک گواهی در برابر کلید عمومی صادر کننده آن و به صورت اختیاری در برابر فهرست ابطال گواهی (CRL) ارائه می دهد.

- «from cryptography import x509»: ماژول «x509» را برای کار با گواهی های X.509 وارد می کند.
- «from cryptography.hazmat.primitives.asymmetric import padding»: ماژول «padding» را برای مدیریت طرح های padding رمزگذاری نامتقارن وارد می کند (در این مورد).
- «from cryptography.x509 import load_pem_x509_certificate, load_pem_x509_crl»: توابع را برای بارگیری گواهی ها و CRL از فایل های رمزگذاری شده با PEM وارد می کند.
- «from cryptography.hazmat.backends import default_backend»: پشتیبان رمزنگاری پیش فرض را برای انجام عملیات رمزنگاری وارد می کند.

عملکرد «تأیید_گواهی» اصلی :

- این تابع زنجیره گواهی ("cert_pem")، گواهی صادر کننده ("issuer_cert_pem") و CRL اختیاری ("crl_pem") را به عنوان آرگومان در نظر گرفت.
- سعی کرده گواهینامه و گواهی صادرکننده (در صورت ارائه) را با استفاده از «load_pem_x509_certificate» بارگیری کند.

- امضای گواهی را در برابر کلید عمومی صادرکننده با استفاده از `issuer_public_key.verify` تأیید کرد.
- با استفاده از CRL (در صورت ارائه) با تکرار گواهی های باطل شده و مقایسه شماره های سریال، ابطال را بررسی کرد.

کلاس CertificateValidator:

- این رویکرد refactored از یک کلاس CertificateValidator برای کپسوله کردن منطق اعتبار سنجی استفاده می کند.
- سازنده (`__init__`) زنجیره گواهی ("`cert_pem`")، گواهی صادر کننده ("`issuer_cert_pem`") و CRL اختیاری ("`crl_pem`") را به عنوان آرگومان می گیرد و آنها را به عنوان ویژگی ذخیره می کند.
- روش `"Validate_certificate"` مراحل اعتبارسنجی واقعی را انجام می دهد:
- گواهی را با استفاده از `'load_pem_x509_certificate'` بارگیری می کند.
- اگر گواهی صادرکننده ارائه شود (`'issuer_cert_pem'`)، سعی می کند آن را بارگیری کند و با استفاده از کلید عمومی صادرکننده تأیید امضا را انجام می دهد.
- اگر یک CRL ارائه شده باشد (`'crl_pem'`)، CRL را بارگیری می کند و بررسی می کند که آیا شماره سریال گواهی در لیست گواهی های باطل شده وجود دارد یا خیر.
- اگر همه چک ها تأیید شوند، متد `«True»` را برمی گرداند که گواهی معتبر را نشان می دهد و در غیر این صورت `«نادرست»` است.

اجرای اصلی (main):

- این بلوک زنجیره گواهی و CRL را از فایل های PEM بارگیری می کند.
- زنجیره گواهی را بر اساس جداکننده `«-----END CERTIFICATE-----»` تقسیم می کند.
- از طریق هر گواهی در زنجیره تکرار می شود:
- داده های PEM گواهی را استخراج می کند و کاراکترهای خط جدید پیشرو/آخر را مدیریت می کند.
- بر اساس موقعیت زنجیره ای گواهی صادر کننده مناسب را تعیین می کند.
- یک نمونه CertificateValidator با گواهی PEM، گواهی صادرکننده (در صورت وجود) و CRL (در صورت وجود) ایجاد می کند.
- متد `"validate_certificate"` را روی شی اعتبار دهنده فراخوانی می کند.
- نتیجه اعتبارسنجی ("`معتبر`" یا "`لغو/بررسی امضا انجام نشد`") برای هر گواهی چاپ می شود.

این کد مکانیزمی را برای اعتبارسنجی زنجیره گواهی ارائه می‌کند و اطمینان می‌دهد که گواهی توسط یک صادرکننده مورد اعتماد امضا شده است و توسط مرجع صدور گواهی (CA) باطل نشده است. بررسی CRL با تأیید اینکه گواهی به خطر نیفتاده است، یک لایه امنیتی اضافی اضافه می‌کند.

The screenshot shows a Python IDE with a project named 'v4-security'. The file explorer on the left lists various files including certificates and private keys. The main editor displays the 'va.py' file, which contains the 'CertificateValidator' class. The class has an '.__init__' method that initializes 'cert_pem', 'issuer_cert_pem', and 'crl_pem'. It also has a 'validate_certificate' method that loads the certificate and checks its issuer. The output window at the bottom shows the execution of 'va.py', which prints 'Certificate 1 is valid.' and 'Certificate 2 is valid.', and then 'Process finished with exit code 0'.

```

class CertificateValidator:
    def __init__(self, cert_pem, issuer_cert_pem=None, crl_pem=None):
        self.cert_pem = cert_pem
        self.issuer_cert_pem = issuer_cert_pem
        self.crl_pem = crl_pem

    def validate_certificate(self):
        """Validates a certificate against its issuer's public key and op
        cert = load_pem_x509_certificate(self.cert_pem, default_backend())

        if self.issuer_cert_pem:
            try:
                issuer_cert = load_pem_x509_certificate(self.issuer_cert_
            except ValueError:
                print("Invalid issuer certificate PEM format.")

```

```

Run: C:\hw4-security\venv\Scripts\python.exe C:\hw4-security\va.py
Certificate 1 is valid.
Certificate 2 is valid.
Process finished with exit code 0

```

تصویر ۲- خروجی کد VA

پیاده سازی کد server:

- کلاس سرور: این کلاس یک سرور امن را تعریف می‌کند که ارتباط TLS را با کلاینت‌ها برقرار می‌کند.
- '.__init__' روش: شی سرور را با جزئیات میزبان، پورت، سوکت و اتصال راه‌اندازی می‌کند.
- روش‌های 'send_msg' و 'rcv_msg': ارسال و دریافت پیام‌ها را از طریق اتصال برقرار شده مدیریت می‌کند و پس از راه‌اندازی TLS، ارتباط امن را تضمین می‌کند.
- روش 'start_tls':
- یک زمینه SSL با احراز هویت مشتری مورد نیاز ایجاد می‌کند ('ssl.Purpose.CLIENT_AUTH').

- گواهی سرور و کلید خصوصی را از فایل های PEM بارگیری می کند ("server-cert.pem" -server- "private-key.pem").
- گواهی CA را از یک فایل ("ca-cert.pem") PEM برای تأیید مشتری بارگیری می کند.
- حالت تأیید را روی «ssl.CERT_REQUIRED» تنظیم می کند و ارائه گواهی مشتری را اجرا می کند.
- به اتصالات گوش می دهد و یکی را می پذیرد.
- اتصال پذیرفته شده را با زمینه SSL می پیچد تا مذاکره TLS را آغاز کند.
- در صورت handshake موفق، «درست»، در صورت شکست «نادرست» (با خطای چاپ) را برمی گرداند.
- روش generate_key_pair: یک جفت کلید خصوصی جدید RSA را با استفاده از پشتیبان رمزنگاری پیش فرض ایجاد می کند.
- روش request_certificate_from_ra (بهبود):
- یک شی "RegistrationAuthority" را نمونه سازی می کند.
- یک جفت کلید با استفاده از 'generate_key_pair' ایجاد می کند.
- روش «RegistrationAuthority» را برای ایجاد یک درخواست امضای گواهی (CSR) با استفاده از جفت کلید تولید شده فراخوانی می کند.
- یک شی "CertificateAuthority" را به نمایش می گذارد .
- برای ارسال CSR و دریافت گواهی سرور (که به طور بالقوه شامل CA) می شود، با «مرجع ثبت» تعامل دارد.
- گواهی دریافت شده را در صورت موفقیت در "server-cert.pem" با فرمت PEM ذخیره می کند.
- در هنگام دریافت گواهی موفقیت آمیز، «درست»، در غیر این صورت «نادرست» را برمی گرداند.
- روش send_certificate_to_client:
- گواهی سرور را از «server-cert.pem» می خواند.
- داده های گواهی را از طریق اتصال ایمن به مشتری ارسال می کند.


```
client.py
project
va.py client.py CA.py SERVER.py
SERVER client
C:\hw4-security\venv\Scripts\python.exe C:\hw4-security\SERVER.py
Certificate obtained and created successfully.
TLS handshake completed.

Process finished with exit code 0
|
```

تصویر ۳- خروجی سرور

پیاده سازی کد CLIENT:

یک کلاس «Client» را تعریف می کند که با استفاده از امنیت لایه حمل و نقل (TLS) یک اتصال امن با یک سرور برقرار می کند. تمرکز بر روی دست دادن TLS سمت مشتری و مدیریت گواهی است.

- روش `__init__`:

- شی کلاینت را با هاست، پورت راه اندازی می کند و یک سوکت TCP ایجاد می کند.

- به سرور متصل می شود.

- روش `send_msg`:

- ارسال پیام به سرور

- روش `rcv_msg`:

- یک پیام از سرور با حداکثر حجم ۴۰۹۶ بایت دریافت می کند.

- روش `start_tls`: (کلید برای دست دادن TLS و بازیابی گواهی)

- TLS handshake را آغاز می کند:

- یک زمینه SSL با استفاده از `ssl.create_default_context()` ایجاد می کند.

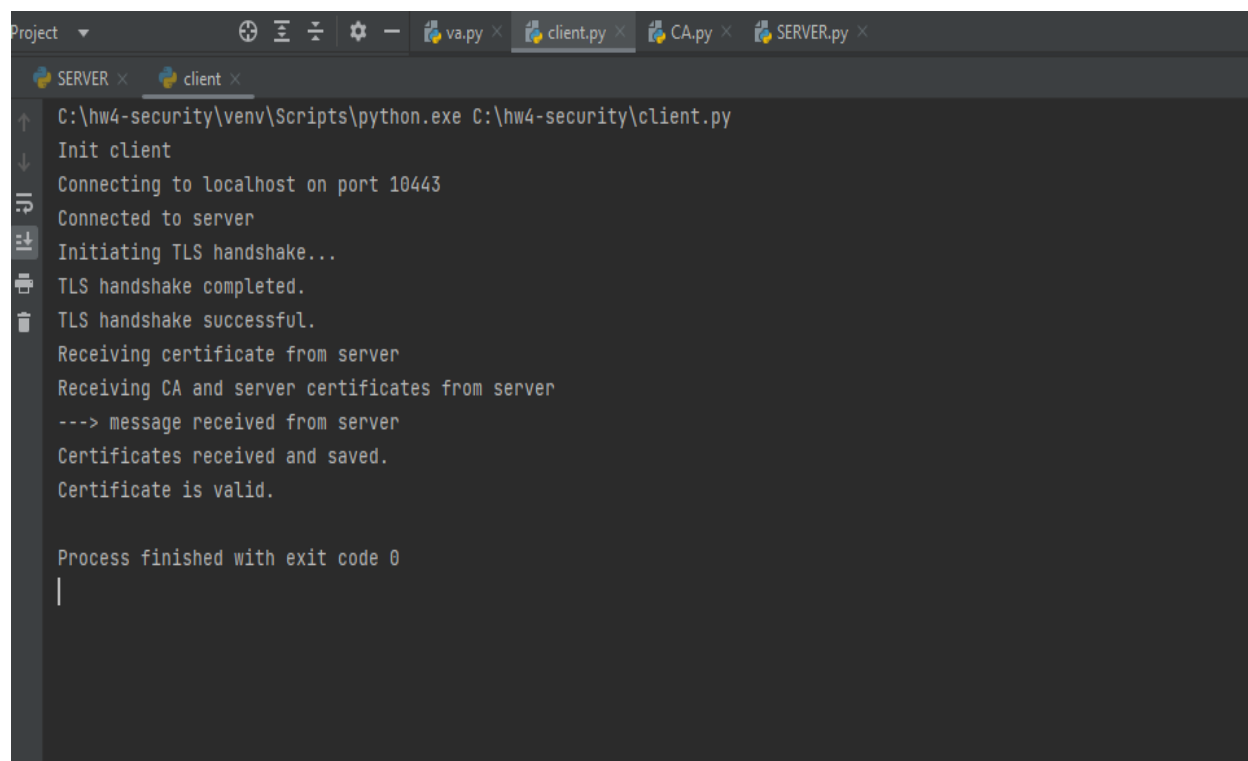
- برای اطمینان از هویت سرور، هدف تأیید را روی «SERVER_AUTH» تنظیم می کند.

- گواهی مرجع گواهی معتبر (CA) را از فایل «ca-cert.pem» با استفاده از

«context.load_verify_locations()» بارگیری می کند.

- حالت تأیید را روی «ssl.CERT_REQUIRED» تنظیم می کند تا اعتبار گواهی سرور را اعمال کند.

- سوکت TCP را با زمینه SSL با استفاده از `context.wrap_socket()` می پیچد.
- «Server_hostname» را برای اطمینان از اعتبارسنجی نام میزبان در برابر گواهی سرور تنظیم می کند.
- استثنای `ssl.SSLError` را کنترل می کند:
- در صورت عدم موفقیت در TLS handshake، پیام خطا را چاپ می کند.
- در صورت موفقیت آمیز بودن handshake «درست»، در غیر این صورت «نادرست» را برمی گرداند.
- روش دریافت_گواهی: (کلید برای دریافت و اعتبارسنجی گواهی):
- منتظر پاسخ گواهی سرور است.
- داده های گواهی را با استفاده از `rcv_msg()` دریافت می کند.
- گواهی دریافت شده را در فایل `server-cert.pem` ذخیره می کند.



```

Project ▾
va.py × client.py × CA.py × SERVER.py ×
SERVER × client ×
C:\hw4-security\venv\Scripts\python.exe C:\hw4-security\client.py
Init client
Connecting to localhost on port 10443
Connected to server
Initiating TLS handshake...
TLS handshake completed.
TLS handshake successful.
Receiving certificate from server
Receiving CA and server certificates from server
--> message received from server
Certificates received and saved.
Certificate is valid.

Process finished with exit code 0
|

```

تصویر ۴ - خروجی کلاینت