گزارش آزمایش چهارم

زیرساخت کلید عمومی

زیرساخت کلید عمومی (PKI)، یک سیستم برای ایجاد، ذخیره‌سازی و توزیع امضاهای دیجیتال می‌باشد که به منظور بررسی یک کلید عمومی منحصربه‌فرد متعلق به نهاد خاص مورد استفاده قرار می‌گیرد. زیرساخت کلید عمومی، گواهی‌های دیجیتال را که حاصل نگاشت کلیدهای عمومی به هویت افراد است، ایجاد کرده و این گواهی‌ها را در یک مخزن مرکزی، به‌طور امن نگهداری می‌کند و اگر لازم باشد، آنها را باطل می‌کند.

زیرساخت کلید عمومی از موارد زیر تشکیل شده‌است:

- مرکز صدور گواهی (CA) که گواهی های دیجیتال را صادر و تأیید و امضا می‌کند.
- مرکز ثبت نام گواهی (RA) که هویت کاربران متقاضی گواهی های دیجیتال از CA را بررسی می‌کند و این درخواست را در صورت احراز هویت کاربر برای CA ارسال می کند.
- VA برای بررسی اعتبار گواهی ها
- central directory. به‌طور مثال، مکانی امن برای ذخیره و نمایه سازی کلیدها
- سیستم مدیریت گواهی که کارهایی مثل دسترسی به گواهی های ذخیره شده یا صدور آن ها را انجام می دهد.
- خط مشی ها و سیاست ها که نیازمندی های این زیرساخت را بیان می کند و به دیگران اجازه می دهد نحوه اعتماد در این زیرستخت را درک کنند.

گواهی X.509

استاندارد X.509 استانداردی است که فرمت گواهی‌های استفاده شده در زیرساخت کلید عمومی (PKI) را تعریف می‌کند. این استاندارد مانند یک زبان خاص است که برای ایجاد این اسناد دیجیتال برای اتصال هویت‌ها به کلیدهای عمومی استفاده می‌شود. گواهی X.509 مانند یک گذرنامه دیجیتال است. این گواهی حاوی اطلاعات زیر است:

کلید عمومی: این کلید برای رمزگذاری استفاده می‌شود. هر کسی می‌تواند آن را ببیند، اما فقط کلید خصوصی مربوطه می‌تواند پیام‌های رمزگذاری شده با آن را رمزگشایی کند.

اطلاعات موضوع (Subject): این اطلاعات درباره مالک گواهی می باشد مانند نام دامنه وب‌سایت، نام سازمان یا نام یک فرد.

اطلاعات صادرکننده (Issuer): این اطلاعات، مرجع صدور گواهی (CA) مورد اعتماد را که گواهی را صادر کرده است، شناسایی می‌کند.

دوره اعتبار (Validity Period): این دوره، چارچوب زمانی است که گواهی در آن معتبر تلقی می‌شود.

امضای دیجیتال: این یک امضای الکترونیکی ویژه است که توسط CA با استفاده از کلید خصوصی آن ایجاد می‌شود. این امضا، صحت گواهی را تأیید می‌کند و تمام اطلاعات را به هم پیوند می‌دهد.

چگونه X.509 با PKI کار می‌کند:

صدور گواهی: هنگامی که یک وب‌سایت یا کاربر به گواهی نیاز دارند، آن را از یک CA درخواست می‌کنند. CA هویت آنها را تأیید می‌کند و در صورت موفقیت، یک گواهی X.509 حاوی اطلاعات ذکر شده در بالا صادر می‌کند.

تأیید و ارتباط امن:

وب‌سایت‌ها: وب‌سایت‌ها از گواهی‌های X.509 خود برای شناسایی خود به مرورگر شما استفاده می‌کنند. مرورگر شما اعتبار گواهی را با CA صادر کننده بررسی می‌کند (مانند تأیید گذرنامه با کشور صادر کننده). اگر همه چیز معتبر باشد، یک اتصال امن (مانند HTTPS) برقرار می‌شود.

امنیت ایمیل: در ایمیل‌های امن (S/MIME)، فرستندگان از گواهی‌های X.509 خود برای «امضای» ایمیل‌هایشان استفاده می‌کنند. این کار هویت آنها را تأیید می‌کند و اطمینان می‌دهد که پیام دستکاری نشده است. گیرنده سپس می‌تواند از کلید عمومی فرستنده برای تأیید امضا استفاده کند.

مزایای گواهی‌های X.509:

استانداردسازی: X.509 فرمت مشترکی را برای گواهی‌ها ارائه می‌دهد و سازگاری در سیستم‌ها و برنامه‌های کاربردی مختلف را تضمین می‌کند.

امنیت: قابلیت‌های امضای دیجیتال و رمزگذاری گواهی‌های X.509، ارتباط امن و تأیید هویت را تضمین می‌کند.

چارچوب اعتماد: X.509 به CAهای مورد اعتماد متکی است که به برقراری اعتماد در تعاملات آنلاین کمک می‌کند.

تایید اعتبار گواهی SSL

دو روش اصلی برای اعتبارسنجی گواهی SSL وجود دارد:

1. اعتبارسنجی مبتنی بر مرورگر:

این روش رایج ترین روش برای کاربران عادی است. مرورگر وب به طور خودکار اکثر مراحل اعتبارسنجی را هر زمان که از وب سایتی با HTTPS بازدید می کنیم، انجام می دهد. در اینجا خلاصه ای ساده آورده شده است:

بررسی HTTPS: مرورگر اطمینان حاصل می کند که وب سایت از HTTPS به جای HTTP ساده استفاده می کند. "S" نشان دهنده یک اتصال امن با استفاده از SSL/TLS است.

تأیید زنجیره گواهی: مرورگر گواهی SSL وب سایت را دریافت می کند. بررسی می کند که آیا گواهی توسط یک مرجع صدور گواهی معتبر (CA) امضا شده است. مرورگرها لیستی از CA های معتمد را به صورت پیش فرض دارند.

اعتبارسنجی زنجیره گواهی (اختیاری): در برخی موارد، گواهی ممکن است توسط یک CA واسطه صادر شود که سپس توسط یک CA ریشه معتمد امضا می شود. مرورگر کل زنجیره گواهی ها را بررسی می کند تا مطمئن شود که همه آنها معتبر و به درستی به هم مرتبط هستند.

بررسی لغو گواهی (اختیاری): برخی از مرورگرها ممکن است یک بررسی اضافی در برابر لیست لغو گواهی (CRL) یا پروتکل وضعیت گواهی آنلاین (OCSP) انجام دهند تا ببینند آیا گواهی توسط CA به دلایل امنیتی (مانند کلید به خطر افتاده) لغو شده است یا خیر.

تایید اعتبار تاریخ انقضا: مرورگر اطمینان می دهد که گواهی در محدوده تاریخ اعتبار خود (منقضی نشده باشد) قرار دارد.

شاخص های امنیتی: در صورت موفقیت آمیز بودن همه بررسی ها، مرورگر یک نماد قفل یا "https" را در نوار آدرس نمایش می دهد که نشان دهنده یک اتصال امن است. در برخی موارد، ممکن است اطلاعات اعتبارسنجی بیشتری در مورد مالک گواهی نیز نمایش دهد.

۲. اعتبارسنجی دستی (برای کاربران پیشرفته):

این روش برای کاربران فنی تری است که می خواهند جزئیات بیشتری در مورد گواهی بدست آورند. برای اعتبارسنجی دستی می توان از ابزارها و تکنیک های مختلفی استفاده کرد:

مشاهده جزئیات گواهی: اکثر مرورگرها این امکان را می دهند که با کلیک کردن روی نماد قفل یا "https" در نوار آدرس، جزئیات گواهی SSL وب سایت را مشاهده کنیم. این کار اطلاعاتی مانند صادرکننده گواهی، دوره اعتبار و موضوع (هویت وب سایت) را نمایش می دهد.

تأیید کننده های آنلاین گواهی: ابزارهای آنلاین ارائه شده توسط مقامات صدور گواهی SSL یا فروشندگان امنیتی وجود دارند که به ما این امکان را می دهند که برای اعتبارسنجی، آدرس وب سایتی را وارد کنیم یا یک فایل گواهی را بارگذاری کنیم. این ابزارها می توانند اطلاعات دقیق تری در مورد گواهی و زنجیره اعتماد آن ارائه دهند.

ابزارهای خط فرمان (برای کارشناسان): کاربران پیشرفته می توانند از ابزارهای خط فرمانی مانند OpenSSL برای دانلود و تجزیه و تحلیل گواهی SSL وب سایت استفاده کنند. این روش نیاز به دانش فنی برای تفسیر داده های گواهی دارد.

پیاده سازی

در این پروژه با استفاده از زبان پایتون یک پیاده سازی ساده از زیر ساخت کلید عمومی ارائه شده است که در آن یک CA ریشه برای صدور گواهی و مدیریت گواهی های باطل شده، یک RA برای احراز هویت کاربر و ارجاع درخواست گواهی آن ها و یک VA برای بررسی اعتبار گواهی های ارسالی و 2 تا کلاینت می باشد که با یکدیگر ارتباط می گیرند.

در ابتدا برای همه موجودیت ها یک کلید خصوصی ایجاد می شود که در تصویر زیر برای CA ریشه ایجاد شده است.

```
def generate_private_key():
    root_private_key = rsa.generate_private_key(
        public_exponent=65537,
        key_size=4096,
        backend=default_backend()
    )

    with open('ca_private_key.pem', 'wb') as f:
        f.write(root_private_key.private_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PrivateFormat.PKCS8,
            encryption_algorithm=serialization.NoEncryption()
        ))

    return root_private_key
```

برای CA ریشه یک گواهی ایجاد می شود که توسط خودش امضا می شود.

```
def generate_root_certificate(ca_private_key):
    ca_public_key = ca_private_key.public_key()
    builder = x509.CertificateBuilder()
    builder = builder.subject_name(x509.Name([
        x509.NameAttribute(x509.oid.NameOID.COUNTRY_NAME, u'US'),
        x509.NameAttribute(x509.oid.NameOID.STATE_OR_PROVINCE_NAME, u'California'),
        x509.NameAttribute(x509.oid.NameOID.LOCALITY_NAME, u'San Francisco'),
        x509.NameAttribute(x509.oid.NameOID.ORGANIZATION_NAME, u'My Company'),
        x509.NameAttribute(x509.oid.NameOID.COMMON_NAME, u'Root CA'),
    ]))
    builder = builder.issuer_name(x509.Name([
        # its issuer is itself
        x509.NameAttribute(x509.oid.NameOID.COUNTRY_NAME, u'US'),
        x509.NameAttribute(x509.oid.NameOID.STATE_OR_PROVINCE_NAME, u'California'),
        x509.NameAttribute(x509.oid.NameOID.LOCALITY_NAME, u'San Francisco'),
        x509.NameAttribute(x509.oid.NameOID.ORGANIZATION_NAME, u'My Company'),
        x509.NameAttribute(x509.oid.NameOID.COMMON_NAME, u'Root CA'),
    ]))
    builder = builder.not_valid_before(datetime.utcnow())
    builder = builder.not_valid_after(datetime.utcnow() + timedelta(days=10*365))  # Valid for 10 years
    builder = builder.serial_number(x509.random_serial_number())
    builder = builder.public_key(ca_public_key)
    builder = builder.add_extension(
        x509.BasicConstraints(ca=True, path_length=None), critical=True,
    )
    # Self-sign our certificate
    ca_certificate = builder.sign(
        private_key=ca_private_key, algorithm=hashes.SHA256(),
        backend=default_backend()
    )
    with open('ca_cert.pem', 'wb') as f:
        f.write(ca_certificate.public_bytes(serialization.Encoding.PEM))
    return ca_certificate
```

اما در سایر موارد یعنی کلاینت ها و VA و RA ابتدا یک فایل CSR ایجاد می شود و این فایل تقاضای امضای گواهی برای CA ریشه ارسال شده و توسط آن امضا می شود.

تصویر زیر ایجاد فایل CSR برای RA را نشان می دهد.

```python
def generate_csr(private_key):
    csr = x509.CertificateSigningRequestBuilder() \
        .subject_name(x509.Name([
        x509.NameAttribute(NameOID.COUNTRY_NAME, u'US'),
        x509.NameAttribute(NameOID.STATE_OR_PROVINCE_NAME, u'California'),
        x509.NameAttribute(NameOID.LOCALITY_NAME, u'San Francisco'),
        x509.NameAttribute(NameOID.ORGANIZATION_NAME, u'My RA'),
        x509.NameAttribute(NameOID.ORGANIZATIONAL_UNIT_NAME, u'IT'),
        x509.NameAttribute(NameOID.COMMON_NAME, u'RA'),
    ])) \
        .add_extension(
        x509.SubjectAlternativeName([x509.DNSName(u'RA')]),
        critical=False,
    ) \
        .sign(private_key, hashes.SHA256())

    with open('ra_csr.pem', 'wb') as f:
        f.write(csr.public_bytes(serialization.Encoding.PEM))

    return csr
```

تصویر زیر تابعی را نشان می دهد که از طریق آن CA ریشه سایر گواهی ها را امضا می کند.

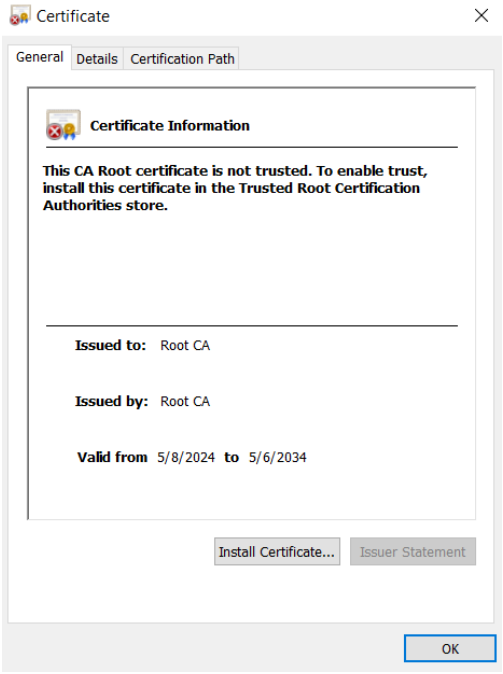
```python
def sign_certificate(csr, ca_private_key, ca_cert, output_cert_path):
    # Generate the certificate signing the CSR with the CA's private key
    certificate = x509.CertificateBuilder()\
        .subject_name(csr.subject)\
        .issuer_name(ca_cert.subject)\
        .public_key(csr.public_key())\
        .serial_number(x509.random_serial_number())\
        .not_valid_before(ca_cert.not_valid_before_utc)\
        .not_valid_after(ca_cert.not_valid_after_utc)\
        .add_extension(x509.BasicConstraints(ca=False, path_length=None), critical=True)\
        .sign(ca_private_key, hashes.SHA256(), default_backend())

    with open(output_cert_path, 'wb') as f:
        f.write(certificate.public_bytes(serialization.Encoding.PEM))

    return certificate
```

در این پیاده سازی کلاینت برای اینکه بتواند گواهی امضا شده توسط CA ریشه را بدست بیاورد، ابتدا به RA درخواست میزند و توسط RA احراز هویت می شود سپس در صورت تائید هویت آن گواهی اش توسط CA ریشه امضا می شود.

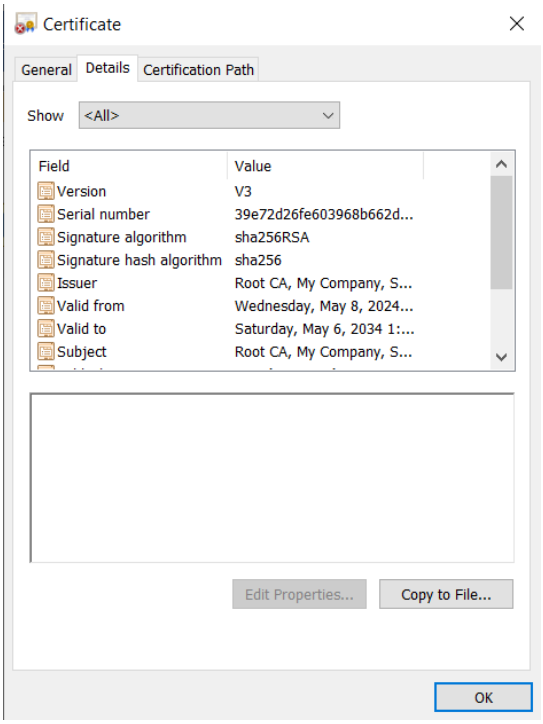
```python
def client_socket():
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect(("localhost", 12345))

    # Send data to RA server
    username = input("Enter username: ")
    password = input("Enter password: ")
    cert_path = 'client_cert.pem'
    csr_path = 'client_csr.pem'
    data = f"{username}:{password}:{cert_path}:{csr_path}".encode("utf-8")
    sock.send(data)

    # Receive data from RA server
    data = sock.recv(1024)

    if data.decode("utf-8") == "success":
        print("Authentication successful")
    else:
        print("Authentication failed")
    sock.close()
```

```python
def RA_socket():
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.bind(("localhost", 12345))
    sock.listen(1)

    while True:
        conn, addr = sock.accept()
        data = conn.recv(1024)
        data_str = data.decode("utf-8")
        username, password, cert_path, csr_path = data_str.split(":")
        if user_authentication(username, password):
            CA_sign_certificate(csr_path, cert_path)
            conn.sendall(b"success")
        else:
            conn.sendall(b"failure")
        conn.close()
```

فایل .cert گواهی CA ریشه به صورت زیر می باشد.

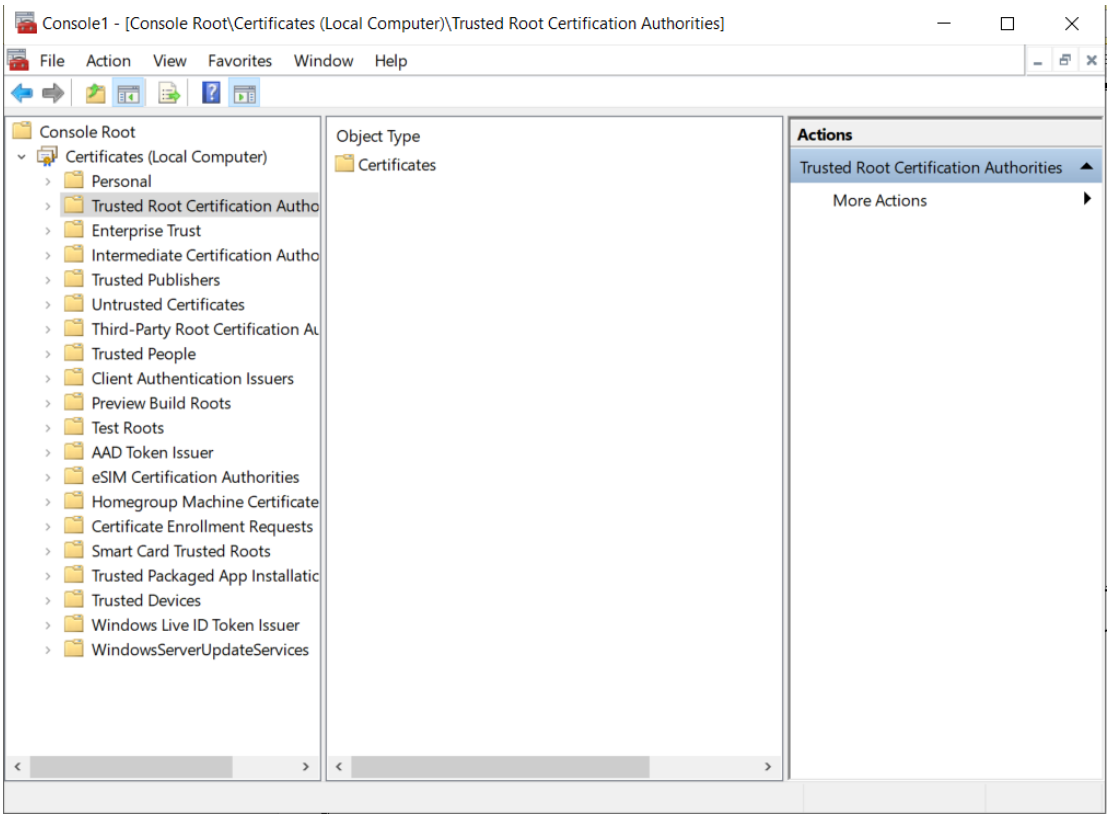


همانطور که مشاهده می شود نام سازمان مالک گواهی و سازمانی که گواهی را صادر کرده و مدت زمان اعتبار آن آورده شده با بررسی بخش جزئیات می توانیم اطلاعات دقیق تری مانند سریال آن، الگوریتم مورد استفاده برای امضا و هش و ... ازگواهی بدست بیاوریم.

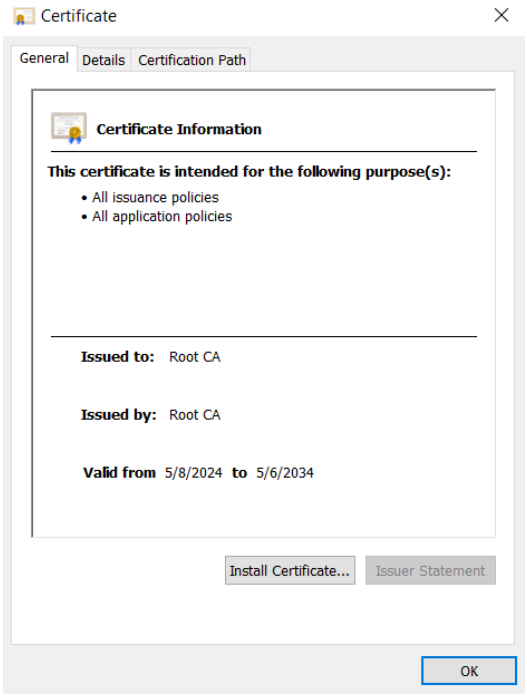


از آنجا که این گواهی در لیست گواهی های مورد اعتماد سیستم قرار ندارد و هیچ CA ای که برای سیستم معتبر است آن را امضا نکرده است، این گواهی معتبر نیست.

با استفاده از ایجاد تغییراتی در فولدر زیر و افزودن گواهی خود به این فولدر سیستم گواهی را به عنوان گواهی ای معتبر شناسایی میکند.



حالا مشاهده می شود که گواهی به گواهی مورد اعتماد برای سیستم تبدیل شده است.



به منظور نگهداری گواهی های باطل شده CA مطابق تابع زیر یک لیست CRL می سازد.

```
def create_crl(ca_private_key, ca_cert):
    now = datetime.utcnow()
    next_update = now + timedelta(days=7)

    crl_builder = x509.CertificateRevocationListBuilder().issuer_name(
        ca_cert.subject
    ).last_update(
        now
    ).next_update(
        next_update
    )

    crl = crl_builder.sign(ca_private_key, hashes.SHA256(), default_backend())

    with open('crl.pem', 'wb') as f:
        f.write(crl.public_bytes(serialization.Encoding.PEM))

    return crl
```

و برای به روزرسانی این لیست از تابع زیر استفاده می کند که در آن لیست قبلی را بارگذاری کرده و تاریخ به روزرسانی بعدی و گواهی های باطل شده جدید را در آن قرار می دهد.

```
def update_crl(ca_private_key, revoked_cert_path):
    cert_to_revoke_data = open(revoked_cert_path, 'rb').read()
    cert_to_revoke = x509.load_pem_x509_certificate(cert_to_revoke_data, backend=default_backend())
    with open('crl.pem', 'rb') as f:
        crl = load_pem_x509_crl(f.read(), default_backend())
    # generate a new crl object
    builder = x509.CertificateRevocationListBuilder()
    builder = builder.issuer_name(crl.issuer)
    builder = builder.last_update(crl.last_update_utc)
    builder = builder.next_update(datetime.now() + timedelta(1, 0, 0))
    # add crl certificates from file to the new crl object
    for i in range(0, len(crl)):
        builder = builder.add_revoked_certificate(crl[i])
    # see if the cert to be revoked already in the list
    ret = crl.get_revoked_certificate_by_serial_number(cert_to_revoke.serial_number)

    # if not, then add new revoked cert
    if not isinstance(ret, x509.RevokedCertificate):
        revoked_cert = x509.RevokedCertificateBuilder() \
            .serial_number(cert_to_revoke.serial_number) \
            .revocation_date(datetime.now()).build(backend=default_backend())

        builder = builder.add_revoked_certificate(revoked_cert)

    # sign and save to new crl file
    cert_revocation_list = builder.sign(private_key=ca_private_key, algorithm=hashes.SHA256(), backend=default_backend())

    with open('crl.crl', 'wb') as f:
        f.write(cert_revocation_list.public_bytes(serialization.Encoding.PEM))

    with open('crl.pem', 'wb') as f:
        f.write(cert_revocation_list.public_bytes(serialization.Encoding.PEM))
```

برای ارتباط دو کلاینت با یکدیگر از ارتباط SSL استفاده می کنند.

در مثال زیر کلاینت 1 ارتباط را آغاز کرده و گواهی خود را ارسال می کند و کلاینت 2 آن را دریافت می کند و اگر اعتبارسنجی به درستی صورت گرفت، گواهی خود را نیز ارسال می کند. (البته در این پیاده سازی از آنجا که در دریافت گواهی اشکالی وجود دارد گواهی مجددا بارگذاری می شود تا در روند پروژه و بررسی سایر قسمت ها مشکلی پیش نیاید)

```python
# initiate communication
def client1_initiate_ssl(client1_certificate):
    context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)
    context.load_verify_locations('ca_cert.pem')
    context.load_cert_chain(certfile='client1_cert.pem', keyfile='client1_private_key.pem')
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    ssl_sock = context.wrap_socket(sock, server_hostname='Client2')
    ssl_sock.connect(('localhost', 12345))
    client1_certificate_bytes = client1_certificate.public_bytes(Encoding.PEM)
    ssl_sock.send(client1_certificate_bytes)
    client2_certificate = ssl_sock.recv(1024)
    ssl_sock.close()
```

```python
# listen
def client2_listen_ssl(client2_certificate):
    context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
    context.load_verify_locations('ca_cert.pem')
    context.load_cert_chain(certfile='client2_cert.pem', keyfile='client2_private_key.pem')
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.bind(('localhost', 12345))
    sock.listen(1)
    while True:
        conn, addr = sock.accept()
        ssl_conn = context.wrap_socket(conn, server_side=True)
        client1_certificate_bytes = ssl_conn.recv(1024)
        # client1_certificate = x509.load_pem_x509_certificate(client1_certificate_bytes, default_backend())
        with open('client1_cert.pem', 'rb') as cert_file:
            client1_certificate = load_pem_x509_certificate(
                cert_file.read(),
                backend=default_backend()
            )
        if va_validate_certificate(client1_certificate):
            print('Certificate Validated Successfully')
            client2_certificate_bytes = client2_certificate.public_bytes(serialization.Encoding.PEM)
            ssl_conn.send(client2_certificate_bytes)
        else:
            print('Certificate Validation Failed')

        ssl_conn.close()
```

کلاینت 2 برای اعتبارسنجی گواهی کلاینت 1، گواهی را با ارتباط SSL به VA ارسال کرده و VA نیز توسط توابع زیر اعتبار آن را بررسی می کند.

```python
def validate_certificate(certificate, crl):
    if certificate.issuer != crl.issuer:
        return 'False'

    revoked_certificate = crl.get_revoked_certificate_by_serial_number(certificate.serial_number)
    if revoked_certificate is not None:
        return 'False'

    return 'True'

def va(certificate_to_validate, crl_path):
    crl = load_crl(crl_path)
    if not validate_certificate(certificate_to_validate, crl):
        print("Certificate is revoked")
    else:
        print("Certificate is valid")
```