شرکت خدمات انفورماتیک

به نام خدا

Crypto Mentoring-Academy 1403

گزارش تولید و اعتبارسنجی امضای دیجیتال

محقق و توسعه دهنده:

ریحانه سراج

۱۴ بهمن ۱۴۰۳

فهرست مطالب

مقدمه

در دنیای دیجیتال امروز، امنیت اطلاعات یکی از مهم‌ترین مسائل مورد توجه است و امضای دیجیتال یکی از روش‌های پیشرفته برای تضمین صحت و اصالت داده‌ها در ارتباطات آنلاین می‌باشد. در این پروژه، یک سامانه نرم‌افزاری برای تولید، ارسال و اعتبارسنجی امضای دیجیتال طراحی و پیاده‌سازی شده است. این سامانه از دو بخش اصلی Client و Server تشکیل شده است.

در بخش Client، فرآیند امضای دیجیتال به‌صورت خودکار و با استفاده از کلید خصوصی ذخیره‌شده در یک فایل KeyStore (از نوع PKCS12 یا PFX) انجام می‌شود. داده‌ای که قرار است امضا شود، یک رشته (String) است که پس از امضا، به همراه امضای دیجیتال ایجاد شده به سرور ارسال می‌شود. در سمت Server، داده‌های دریافتی از سمت Client مورد بررسی قرار می‌گیرند و اعتبار سنجی میشوند. در صورتی که تمام مراحل اعتبار سنجی در سمت سرور تایید گردد، داده امضا شده به همراه امضای دیجیتال در پایگاه داده به فرمت DER ذخیره و برای پردازش‌های آتی آماده می‌شود. اما اگر امضای دیجیتال معتبر نباشد یا گواهی الکترونیکی معتبر نباشد، پیغام خطا به Client بازگشت داده می‌شود.

در این پروژه برای تولید زوج کلید و صدور گواهی الکترونیکی از ابزار OpenSSL استفاده شده است. در ادامه نیز با استفاده از زبان برنامه‌نویسی جاوا و توابع و کلاس‌های مرتبط با پکیج java.security، انجام عملیات امضای دیجیتال و اعتبارسنجی گواهی‌ها صورت گرفته است.

ایجاد فایل PKCS12

برای ایجاد فایل p12 از OpenSSL استفاده کردیم که مراحل ایجاد ان به شرح زیر است.

تولید گواهی و کلید خصوصی ریشه

در قدم اول برای rootCA کلید خصوصی[1] ایجاد می‌کنیم. این کلید خصوصی برای امضای گواهی‌ها و تایید هویت گواهی ریشه مورد استفاده قرار می‌گیرد که در دستور زیر کلید خصوصی RSA با طول ۴۰۹۶ بیت ایجاد می‌شود و در rootCA.key ذخیره می‌شود.

```
(base) rey@DESKTOP-DTPDD7C:~$ openssl genrsa -out rootCA.key 4096
Generating RSA private key, 4096 bit long modulus (2 primes)
.....................................................++++
.....................................................................................++++
e is 65537 (0x010001)
```

در قدم دوم برای root CA گواهی ایجاد می‌کنیم. بدین صورت که برای ایجاد گواهی از استاندارد X509 استفاده میشود این گواهی self signed است زیرا که گواهی توسط خود root CA امضا می‌شود. در ادامه‌ی این دستور، مسیر کلید خصوصی که برای امضای این گواهی نیاز است درج می‌شود همچنین مدت اعتبار گواهی به صورت پیش فرض دو ساله در نظر گرفته شد. در قسمت subj جزئیات هویتی گواهی که شامل کد کشور، استان، شهر، Organization name و Common name است، درج می‌شود.

```
(base) rey@DESKTOP-DTPDD7C:~$ openssl req -x509 -new -nodes -key rootCA.key -sha256 -days 730 -out rootCA.crt -subj"/C=I
R/ST=THR/L=City/O=Organization/OU=RootCA/CN=RootCA"-addext "keyUsage=keyCertSign, cRLSign"-addext "basicConstraints=CA:T
RUE"\-addext "subjectKeyIdentifier=hash"
```

سایر دستورات بکار گرفته شده نیز به شرح زیر است:

- -addext "keyUsage= keyCertSign, cRLSign"

  مشخص می‌کند که این گواهی برای امضای گواهی‌ها و لیست‌های ابطال گواهی (CRL) استفاده می‌شود.

- -addext "basicConstraints= CA:TRUE"

  نشان می‌دهد که این گواهی متعلق به یک مرجع صدور گواهی (CA) است.

- -addext "subjectKeyIdentifier= hash"

  مقدار SKI (Subject Key Identifier) را برای شناسایی کلید ایجاد می‌کند.

[1] Private Key

بررسی صحت ایجاد فایل rootCA.crt

```
(base) rey@DESKTOP-DTPDD7C:~$ cat rootCA.crt
-----BEGIN CERTIFICATE-----
MIIF4zCCA8ugAwIBAgIUUuVCwu0UMLdgYZi99VYLtCaMqNEwDQYJKoZIhvcNAQEL
BQAwYzELMAkGA1UEBhMCSVIxDDAKBgNVBAgMA1RIUjENMAsGA1UEBwwEQ2l0eTEV
MBMGA1UECgwMT3JnYW5pemF0aW9uMQ8wDQYDVQQLDAZSb290Q0ExDzANBgNVBAMM
BlJvb3RDQTAeFw0yNTAxMjcxODIxMTlaFw0yNzAxMjcxODIxMTlaMGMxCzAJBgNV
BAYTAklSMQwwCgYDVQQIDANUSFIxDTALBgNVBAcMBENpdHkxFTATBgNVBAoMDE9y
Z2FuaXphdGlvbjEPMA0GA1UECwwGUm9vdENBMQ8wDQYDVQQDDAZSb290Q0EwggIi
MA0GCSqGSIb3DQEBAQUAA4ICDwAwggIKAoICAQDLqdQyb7GtXRE/tF7HtdfFOsYz
3oZ66ZkEzTWzGUQyHKlaNyn8CkNz9kXThYUotieyXScisQxkVBHw5QyVk78/fHFZ
zt4U7reA4i6ZL/U29ocequO/brZpOyvCdVtTPx75WjhtZcBVSmoCqDQS8uFsUnCF
RdCJf3VR4cfFaPMuM/lC9bHwFKXguTXkr1v4SmDX22dWEuqxvMjwpTKuQ2pfxWRK
LmY9aqaSFUDaPAXOGmUbgdIALpGnKPDBDhLKPCPyIsAtL0LkLBP0IEsSsd9dTGjR
b6/vEbJ5HXB1y0OcyW+9tp+O55TwlMDHM9RAux8GNj8ecSlLjd9cK5Jo1VYYEQPr
0Q5wOmBCSeYXlF+FRljuvwD0qLICGgY8PEeqkjue1rmJ/dIrU+GN7vEceDh0MS+j
bAlrJhksP2Nn01GWV8RxYlEuKm0GB7AB/F45Ldljpgk/9n/2Ye+vnoj9YE8G/1Uw
Zq6pN8THYk7SLH5tCNbVCJSexoWvUW400tDhpluh2tzxsrHEmJ4sjCFHRzPDLVin
AqwWyCY+OXd3F6Bue8yhwcgP1G+xrlUwhMKRLh+ll7/qtW3xF1puBTXIXSsP7PnK
eSotUMYa7mqz2k6m1V2JlNKJDTqXa4tH8u2unWUH18vf1x7/0fk4gcm7Kws3D+19
UAUbmNzktvBVtN+xvwIDAQABo4GOMIGLMB0GA1UdDgQWBBSZJQvgaHdB/3BhLxHR
HXAyMpkP2DAfBgNVHSMEGDAWgBSZJQvgaHdB/3BhLxHRHXAyMpkP2DAPBgNVHRMB
Af8EBTADAQH/MAsGA1UdDwQEAwIBBjAMBgNVHRMEBTADAQH/MB0GA1UdDgQWBBSZ
JQvgaHdB/3BhLxHRHXAyMpkP2DANBgkqhkiG9w0BAQsFAAOCAgEAcfdDmv38gciH
hvyuD3fOTlHRBDe4zc6+e6PJD5zRlH/AEvBvuGSl85r7ghwB0R325j3b8U0gYF1C
dSpc3SH7b7HzSrNJXinvt5wkgEL5fanzh3UOpHROrwlHJjtuvJ5+ElI5mM8C6rRR
lbfUul6ld6F9YFMcDuaMyL960v9ZuAJvHPfUdfaC63NZZl0N4LvaheEPnjHNgF8w
on8EgBdFCdVp2R9IpwhwfS8TBVvOdVxDVAg9OU6zbdpigQZmh9lQ8OrZHFsFL+i+
CmwGWW2q/s4/UaKCLgmd/Q3NrBrKUrvJLU9vnuNrrOGDGNFlKT7iZn67l6aaL9qm
P62AEVU1FwQa2iNhsixtDCH46/s4jOLvLXEyA9Ro4296IJpVwfhARMoYhsJHyijt
xTFKkEbdEj+GHL2eIf70cZIFs3liAwM1SE33iJRio+jLOjVZDqvlIaPr5+9AbNO5
k4qQGQfTT+sdRKHlBSf1WFFn3868M5pTPhnAxWRwj53fp68KgjfmMlyFJEgfz9+1
M99dD37Z8GwmrDEvN4UF6Xa+Nar75nlXAexjs2R06ksJARRMyEZFZ57wyARcNg5s
sjO2pZ5zgb9kriVRtGp/DLqEUZh8Kqal0jQqnb82K8LlMcknz2BxwwsejqNcwXAn
de/VSWN4klZV5iW7dgFI/C4OgGvfI0c=
-----END CERTIFICATE-----
```

تولید گواهی و کلید خصوصی میانی

بدین ترتیب در ادامه تولید کلید خصوصی و گواهی CA میانی را داریم.

```
(base) rey@DESKTOP-DTPDD7C:~$ openssl genrsa -out intermediateCA.key 4096
Generating RSA private key, 4096 bit long modulus (2 primes)
.++++
.....................................................................................++++
e is 65537 (0x010001)
```

```
(base) rey@DESKTOP-DTPDD7C:~$ openssl req -new -key intermediateCA.key -out intermediateCA.csr -subj
"/C=IR/ST=THR/L=City/O=Organization/OU=IntermediateCA/CN=IntermediateCA" -addext "keyUsage=keyCertSig
n, cRLSign" -addext "basicConstraints=CA:TRUE" -addext "subjectKeyIdentifier=hash"
```

در کنار دستورات مشابه قبلی یک درخواست گواهی Certificate Signing Request(CSR) ایجاد می‌شود برای CA میانی و آن را در فایل intermediateCA.csr ذخیره می‌کنیم. که CSR برای گرفتن گواهی میانی که توسط گواهی ریشه امضا شود استفاده می‌شود.

دستور زیر گواهی میانی را با استفاده از گواهی و کلید خصوصی ریشه امضا می‌کند.

```
(base) rey@DESKTOP-DTPDD7C:~$ openssl x509 -req -in intermediateCA.csr -CA rootCA.crt -CAkey rootCA.k
ey -CAcreateserial -out intermediateCA.crt -days 730 -sha256 -extfile <(echo -e "authorityKeyIdentifi
er=keyid,issuer\nbasicConstraints=CA:TRUE, pathlen:0\nkeyUsage=keyCertSign, cRLSign\nsubjectKeyIdenti
fier=hash")
Signature ok
subject=C = IR, ST = THR, L = City, O = Organization, OU = IntermediateCA, CN = IntermediateCA
Getting CA Private Key
```

در قسمت extfile تنظیمات بیشتری برای گواهی درج شده از جمله:

- authorityKeyIdentifier= keyid, issuer

شناسایی کلید مرجع و صادرکننده

- basicConstraints= CA:TRUE, pathlen:0

گواهی میانی می‌تواند گواهی صادر کند اما نه بیشتر از یک سطح

- keyUsage= keyCertSign, cRLSign

قابلیت امضای گواهی‌ها و CRL

- subjectKeyIdentifier=hash

ایجاد شناسه کلید برای گواهی

در ادامه برای اطمینان از ایجاد و صحت فایل intermediateCA.crt از این دستور استفاده می‌کنیم.

```
(base) rey@DESKTOP-DTPDD7C:~$ ls -l intermediateCA.crt
-rw-r--r-- 1 rey rey 2061 Jan 27 22:31 intermediateCA.crt
(base) rey@DESKTOP-DTPDD7C:~$ cat intermediateCA.crt
-----BEGIN CERTIFICATE-----
MIIFxDCCA6ygAwIBAgIUfR4GuZrtX3DM07O7yyPVPVNqFVQwDQYJKoZIhvcNAQEL
BQAwYzELMAkGA1UEBhMCSVIxDDAKBgNVBAgMA1RIUjENMAsGA1UEBwwEQ2l0eTEV
MBMGA1UECgwMT3JnYW5pemF0aW9uMQ8wDQYDVQQLDAZSb290QOExDzANBgNVBAMM
BlJvb3RDQTAeFw0yNTAxMjcxOTAxMzFaFw0yNzAxMjcxOTAxMzFaMHMxCzAJBgNV
BAYTAklSMQwwCgYDVQQIDANUSFIxDTALBgNVBAcMBENpdHkxFTATBgNVBAoMDE9y
Z2FuaXphdGlvbjEXMBUGA1UECwwOSW50ZXJtZWRpYXRlQ0ExFzAVBgNVBAMMDklu
dGVybWVkaWF0ZUNBMIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAnFdY
E9MbGldG+FSkGXkMzU0OmAkR9aMjYA0hL129aOiM7oRjK0WZMInejK1KHtu/g0DX
sIdnTdN/5nkU86PHQQRnq0b2Cevem/AsJEB8UQuQH0oyeT3fQNzSv/Y7v8KrLSst
iWzE6lWBjxi+kHa+v12i9rCiXlX7OHK4YdHIIyOmzpmHkFwhQbQrv/GNvPrNr13s
4DeL5ZRsI2lqgbKb48oSb0sbQyTuxGxkQ4rpK6QAXXs27YgXq8looklQuHohBwZC
5E3AHK7qOeB7C5ajtL0Skem0GJHFfJxggSF06lH6xmsR+05/aj598MTJg2i4zJpT
mrZg+r3RKft/StG4cQ6jfwqsZ0Stz4Gj4SfK8VkCQlNK7QE3zPUi/l/EThVYsiPg
MF9DaIO7Sq9TbHpP/Fq49yfMOCFZeiAyOTWzI9VH2S9izDMgoaY0F5Wl45hfNjeJ
22L/NzFJXZy/mOS75BBED/KL9rFVE1nz9MPm4+nQsfmK9Tjm/u7VH6Sw6vhqD/oT
9CHWUZhsov5QCdqEklmeluspjRqdzA03hptHw5cWszs5dUYFM1+bUbAtT7raPZ/k
39+ey0RcFjsy9aDWBZ6MIESciFGNhYRYAVC7z4/ktGwK+ub4gcVpva1Fqv2s4NnW
L3NjCQoByNVQETWdOUum/O+8e9EvSgG7gpcgXZsCAwEAAaNgMF4wHwYDVR0jBBgw
FoAUmSUL4Gh3Qf9wYS8R0R1wMjKZD9gwDwYDVR0TBAgwBgEB/wIBADALBgNVHQ8E
BAMCAQYwHQYDVR0OBBYEFIsIidw0MRXQ/gGM/gyTNOqWFwQTMA0GCSqGSIb3DQEB
CwUAA4ICAQAJwPSU1sqzvhCxsYdDB8uWf7YEpMTlKdwWvnYfMntpn9sHWMC3Sac3
j+AWYwefxkcA24EnYlgb4CZA+36OCcygG0lyrppDESavNx7kP9QujBAyayJUixVS
JO7X9YkUVGuuSz3xC304NZDuBNVBDLX0pPg4klyQByYY+ew1oP5g/Tf7SAHzg2Sk
cA48/ZL3mfyYW1DjuDOG51RTBMAyPYVAatVPG2Fg9I1qBP889EQHhv0YHsAltjaN
iZohzPi5KJILBUXzvRM4PfmNessw7s6uYoMwzfw06gG93sJFfkeWUmW2VEnhiVBx
/wVq/LCGHRpeyrbTYT1HpS/fQmsNtdF7LRhLmM0+qCcmbKUowF7nkgaze7k3S+b9
fvzEVVos1VSY5/C+clqn9Y7PH+PLpf4eHHisr72/pvpV0aNcG92Yjnr4JYm9R3Vb
uOPJ0FouaSpwA7xNUAcm3ochtOmUDGuKdE6SZQNAt0zuu0G9LOCHfZfv08MxXK6q
+WUXNF+v+W5yd2gBzWVBXG8KALsDxJrjIydaMPqxaNInb45IcLxjdfX/ujghnk/M
nVCA5bIa03IaSm+zK4F+616GEYUfeSnm5E3Blx6Q++rTxC2fm/fPhysumNhDwALB
dSUW3QNmaKmdSrw6q6O3bnAuwbni/R0WKK2IvPY4Dy3CHxYaWAdSOA==
-----END CERTIFICATE-----
```

بررسی صحت فایل intermediateCA.csr، که این فایل شامل اطلاعات هویتی است.

```
(base) rey@DESKTOP-DTPDD7C:~$ ls -l intermediateCA.csr
-rw-r--r-- 1 rey rey 1813 Jan 27 21:56 intermediateCA.csr
(base) rey@DESKTOP-DTPDD7C:~$ cat intermediateCA.csr
-----BEGIN CERTIFICATE REQUEST-----
MIIFAzCCAusCAQAwczELMAkGA1UEBhMCSVIxDDAKBgNVBAgMA1RIUjENMAsGA1UE
BwwEQ2l0eTEVMBMGA1UECgwMT3JnYW5pemF0aW9uMRcwFQYDVQQLDA5JbnRlcmll
ZGlhdGVDQTEXMBUGA1UEAwwOSW50ZXJtZWRpYXRlQ0EwggIiMA0GCSqGSIb3DQEB
AQUAA4ICDwAwggIKAoICAQCcV1gT0xsaV0b4VKQZeQzNTQ6YCRH1oyNgDSEvXb1o
6IzuhGMrRZkwid6MrUoe27+DQNewh2dN03/meRTzo8dBBGerRvYJ696b8CwkQHxR
C5AfSjJ5Pd9A3NK/9ju/wqstKy2JbMTqVYGPGL6Qdr6/XaL2sKJeVfs4crhh0cgj
I6bOmYeQXCFBtCu/8Y28+s2vXezgN4vllGwjaWqBspvjyhJvSxtDJO7EbGRDiukr
pABdezbtiBerzWiiSVC4eiEHBkLkTcAcruo54HsLlqO0vRKR6bQYkcV8nGCBIXTq
UfrGaxH7Tn9qPn3wxMmDaLjMmlOatmD6vdEp+39K0bhxDqN/CqxnRK3PgaPhJ8rx
WQJCU0rtATfM9SL+X8ROFViyI+AwX0Nog7tKr1Nsek/8Wrj3J8w4IVl6IDI5NbMj
lUfZL2LMMyChpjQXlaXjmF82N4nbYv83MUldnL+Y5LvkEEQP8ov2sVUTWfPOw+bj
6dCx+Yrl0Ob+7tUfpLDq+GoP+hP0IdZRmGyi/lAJ2oSSWZ6W6ymNGp3MDTeGm0fD
lxazOzllRgUzX5tRsClPuto9n+Tf357LRFwWOzLloNYFnowgRJyIUY2FhFgBULvP
j+S0bAr65viBxWm9rUWq/azg2dYvc2MJCgHI1VARNZ05S6b877x70S9KAbuClyBd
mwIDAQABoEswSQYJKoZIhvcNAQkOMTwwOjALBgNVHQ8EBAMCAQYwDAYDVR0TBAUw
AwEB/zAdBgNVHQ4EFgQUiwiJ3DQxFdD+AYz+DJM06pYXBBMwDQYJKoZIhvcNAQEL
BQADggIBAE7AptUem4c5cGQGSELofJnkIX16RMfUmhdjU4bEBZK3TGZ61J9jYARH
b2OGFrq8b3J66Is6Aw2Q8ylG08Iq43KXQq+Q7Kjm8stGep8LGHh7psHxMvncKIJi
n228ddQuSHhXkeskmPfV6QNBzLCvJQvvMAMU1e+B9QReiyv/RN3MsOYipZ3OfhLG
C15xU52lf7dTD47ymwdZ6yZhFLDyUWTOj5mom4iltxQbvYM+/ZjA7RX/ocgligiO
0fveaCLLRZBxNvhF/hvPyGs2KXuEBpK2C/RADGbaHRisLqGwPLzPrFcGQ8Vpq7a7
UzRMQ6KUrEwa+utQb2ijMgT54LkWGEnqrKfdziI8GlZf2f7So/F0KXzsNMLWuHXK
wKO+kXslELA9HQRubfAHIUi6tSDQfElPSAIPJsIdD26+PqXY199qmFLnPGirj3VX
b/XrluTH2oKDCKGdXQmXH58Riz/CdqPQ0PrHctRFL2aBB5L4SLJf97TfQHz8OhDU
cRURkQkJwnkoOaWEqsZ+oDwWVV/GN/58OOK5VqZlxFCZW8omuN6DPRKu5P7aMzeN
zrVN6Y/k6cG/4I9RMZrXPCqBKJ/pl9HBnN5Cff0CL4jOgalTBf+/hNgf5uKID0+l
mc5fissCcFl2TsFa5FLUkcAzgXF9Bfm/8RXfflmBsq0/JYuMrjBS
-----END CERTIFICATE REQUEST-----
```

تولید گواهی و کلید خصوصی کاربر نهایی

پس از صحت سنجی فایل های ایجاد شده CA میانی تولید کلید خصوصی و گواهی را برای کاربر را داریم. دستور زیر یک کلید خصوصی ۲۰۴۸ بیتی برای کاربر تولید می‌کند.

```
(base) rey@DESKTOP-DTPDD7C:~$ openssl genrsa -out client.key 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
...................+++++
.....................................................+++++
e is 65537 (0x010001)
```

این دستور یک CSR برای کلاینت تولید می‌کند. و در ادامه دستوری برای بررسی صحت فایل client.csr را داریم.

```
(base) rey@DESKTOP-DTPDD7C:~$ openssl req -new -key client.key -out client.csr \
(base) rey@DESKTOP-DTPDD7C:~$ ls -l client.csr
-rw-r--r-- 1 rey rey 1082 Jan 27 22:53 client.csr
(base) rey@DESKTOP-DTPDD7C:~$ cat client.csr
-----BEGIN CERTIFICATE REQUEST-----
MIIC5TCCAc0CAQAwYzELMAkGA1UEBhMCSVIxDDAKBgNVBAgMA1RIUjENMAsGA1UE
BwwEQ2l0eTEVMBMGA1UECgwMT3JnYW5pemF0aW9uMQ8wDQYDVQQLDAZDbGllbnQx
DzANBgNVBAMMBkNsaWVudDCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEB
AOHlteVhgiDqExMlwYQlmumJbEEr9w8TVV6Q0FTXjG80bN0DjzAXS+BoE6ehovku
9RmFTe3cn6CAcEJ9KDCCuIe57uAiwiZv6m48LzS9GWPOFUaeRBkvG/D0IbbfUdny
XSbD8a2qJjkRqmi4ztclNWHUtJ2c8f3Gq44RTRqEmg36u+g5uQd6DmiABwTqGbPw
I9VAmrp5EuWsEz3Mkn7vDamb3evCVCVi4vdwwWcJobNezbCNvmRP7tbghkH+eSaE
HmPG/PIhOVsReS61uwUyQXe9oterV124tGUtl9y93LNREkTJSzMqzGPPms9YLMGi
I3QVOlx5yE7YDL+xjA4nOpUCAwEAAaA9MDsGCSqGSIb3DQEJDjEuMCwwCwYDVR0P
BAQDAgbAMB0GA1UdDgQWBBQD63OH+nrO3d4blkVEAfpkLr1Z8zANBgkqhkiG9w0B
AQsFAAOCAQEAS6So/5YS6vAVcIbqBuyyis2KYiOeNN+JRZazc9Oqyxd/oDpdtbK/
rBMyb17ShCTZVu7dQMNlVT2j7COkXByzhziu0j9KiGzk/Ju+QNfphAK54Xixz/uK
uLCmCXtckhyGgXc+0/F7pla0yR3DiaetwKI2m5ZJPwG5foSnFA1R2x+69g6WKJYJ
boA9BbuIpThICChX+cKHWNDSpsmnYG7Uh6vLaFkqICAag0MrVYSvXhqsQi2RSARb
PGVxZe2wemgiSuDrdgNX3lgR0EdKz6izAIMflqnh7HWfWQcQvzTqR5Uq9ye8Uqhi
FhvDYdRc+iLg7c99j1M2ttNQpoBSVZJvVQ==
-----END CERTIFICATE REQUEST-----
```

در این قسمت امضای گواهی کاربر توسط گواهی میانی را داریم که پارامترهای ان مشابه قسمت قبل است ولی با این تفاوت که CA در قسمت basicConstraints را مساوی با FALSE قرار دادیم زیرا که کاربر این گواهی CA نیست

و کاربر نهایی [۲]است و همچنین پارامترهای keyUsage برای کاربر نهایی nonrepudiation و digitalSignature است.

```
(base) rey@DESKTOP-DTPDD7C:~$ openssl x509 -req -in client.csr -CA intermediateCA.crt -CAkey intermed
iateCA.key -CAcreateserial -out client.crt -days 730 -sha256 -extfile <(echo "authorityKeyIdentifier=
keyid,issuer
basicConstraints=CA:FALSE
keyUsage=digitalSignature, nonRepudiation
subjectKeyIdentifier=hash")
Signature ok
subject=C = IR, ST = THR, L = City, O = Organization, OU = Client, CN = Client
Getting CA Private Key
```

در مراحل پایانی برای ایجاد زنجیره گواهی، دستور زیر سه گواهی (کاربر، میانی و ریشه) را به ترتیب در یک فایل certificate_chain.pem ترکیب می‌کند و یک زنجیره گواهی [۳]کامل (ریشه ⟵ میانی ⟵ کاربر) ایجاد می‌کند. در این زنجیره ترتیب مهم است زیرا گواهی‌های میانی و ریشه باید پس از گواهی کاربر باشند.

```
(base) rey@DESKTOP-DTPDD7C:~$ cat client.crt intermediateCA.crt rootCA.crt > certificate_chain.pem
```

برای ذخیره کلید خصوصی کاربر و زنجیره گواهی در فایل PKCS12 به فرمت p12، باید از دستور زیر استفاده کرد. در هنگام اجرای این دستور از ما خواسته میشود که رمز عبوری برای این فایل درج کنیم که رمز عبوری درج شده برای این فایل ۱۲۳۴۵ است.

```
(base) rey@DESKTOP-DTPDD7C:~$ openssl pkcs12 -export -out client.p12 -inkey client.key -in client.crt
 -certfile certificate_chain.pem -name "ClientCertificate"
```

ساخت فایل CRL

برای مدیریت گواهی‌ها و لیست ابطال گواهی‌ها (CRL)، از فایل تنظیمات openssl.cnf استفاده می‌شود که شامل پارامترهای مربوط به مدیریت گواهی‌ها و CRL است. برای ایجاد یک یک لیست ابطال گواهی برای rootCA و و همچنین یک لیست ابطال گواهی برای intermediateCA، ابتدا تنظیمات مربوط به گواهی ریشه در فایل openssl_root.cnf انجام می‌شود. همچنین، دو فایل index.txt و serial برای مدیریت وضعیت گواهی‌ها (لغو یا صادر شدن) ضروری هستند و در این فایل‌ها اطلاعات مربوط به گواهی‌ها ثبت می‌شود.

[2] End Entity
[3] Certificate Chain

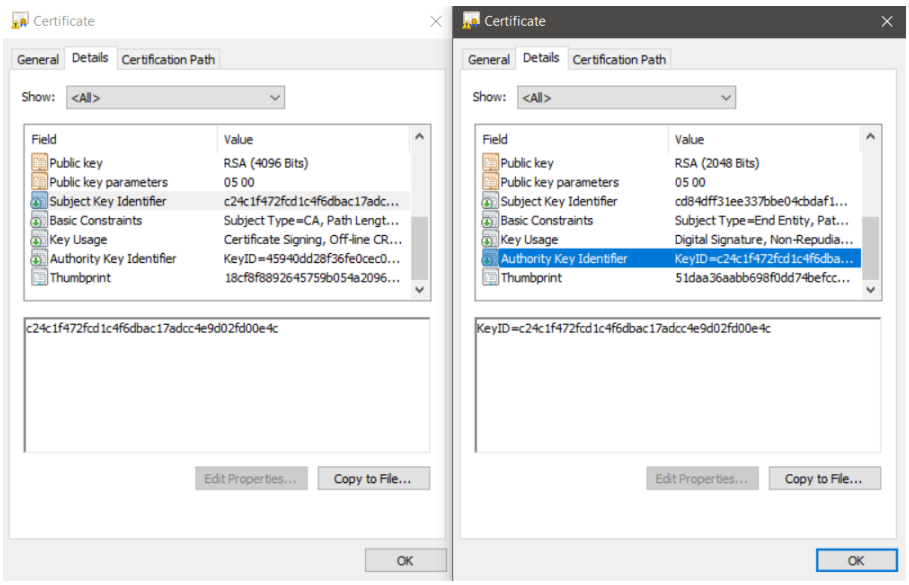
برای ایجاد CRL از دستور زیر استفاده می‌شود:

```
(base) rey@DESKTOP-DTPDD7C:~$ openssl ca -config openssl_root.cnf -gencrl -out rootCA/rootCA.crl
```

این دستور باعث تولید دو فایل CRL خواهد شد که شامل لیستی از گواهی‌های لغو شده توسط هر CA هستند. این فایل‌ها در سمت سرور برای اعتبارسنجی گواهی‌ها مورد استفاده قرار می‌گیرند.

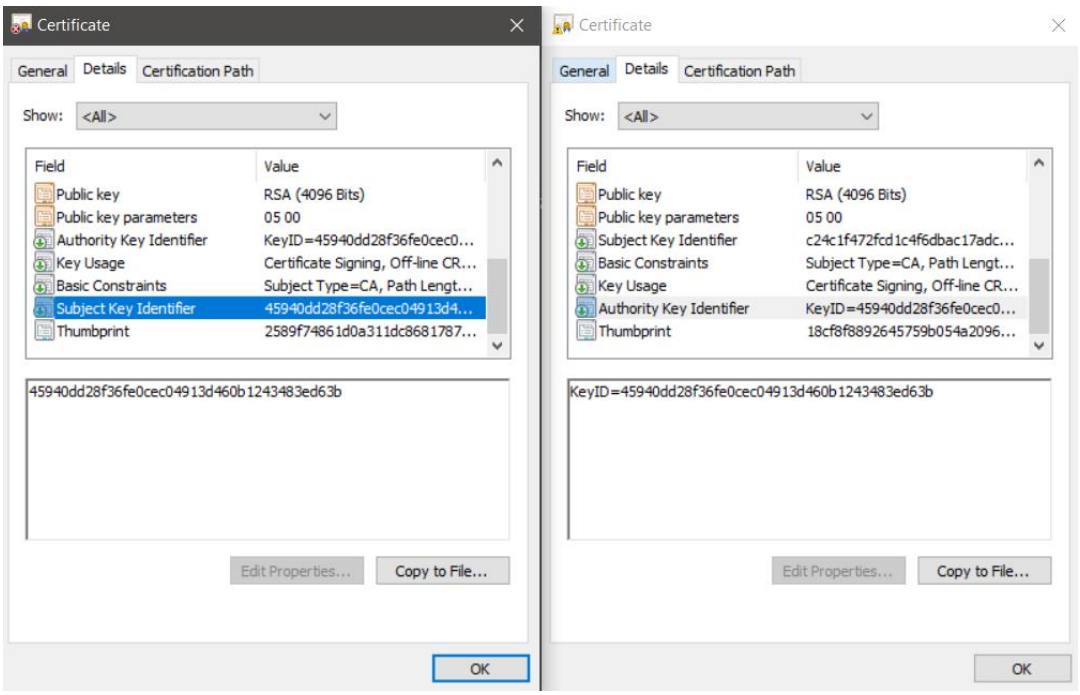
اعتبار سنجی گواهی‌های تولید شده از طریق Key Chaining

در ادامه می‌توانیم زنجیره گواهی را از طریق روش key chaining اعتبا سنجی کنیم.



شکل۱

همان طور که در تصویر۱ مشاهده می‌کنید Authority Key Identifier در گواهی client با Subject Key Identifier در گواهی CA بالا دستی ان یعنی IntermediateCA برابر است بدین ترتیب در تصویر ۲ نیز Authority Key Identifier در گواهی IntermediateCA با Subject Key Identifier در گواهی rootCA برابر است. قابل ذکر است که Authority Key Identifier و Subject Key Identifier در گواهی rootCA مساوی است.

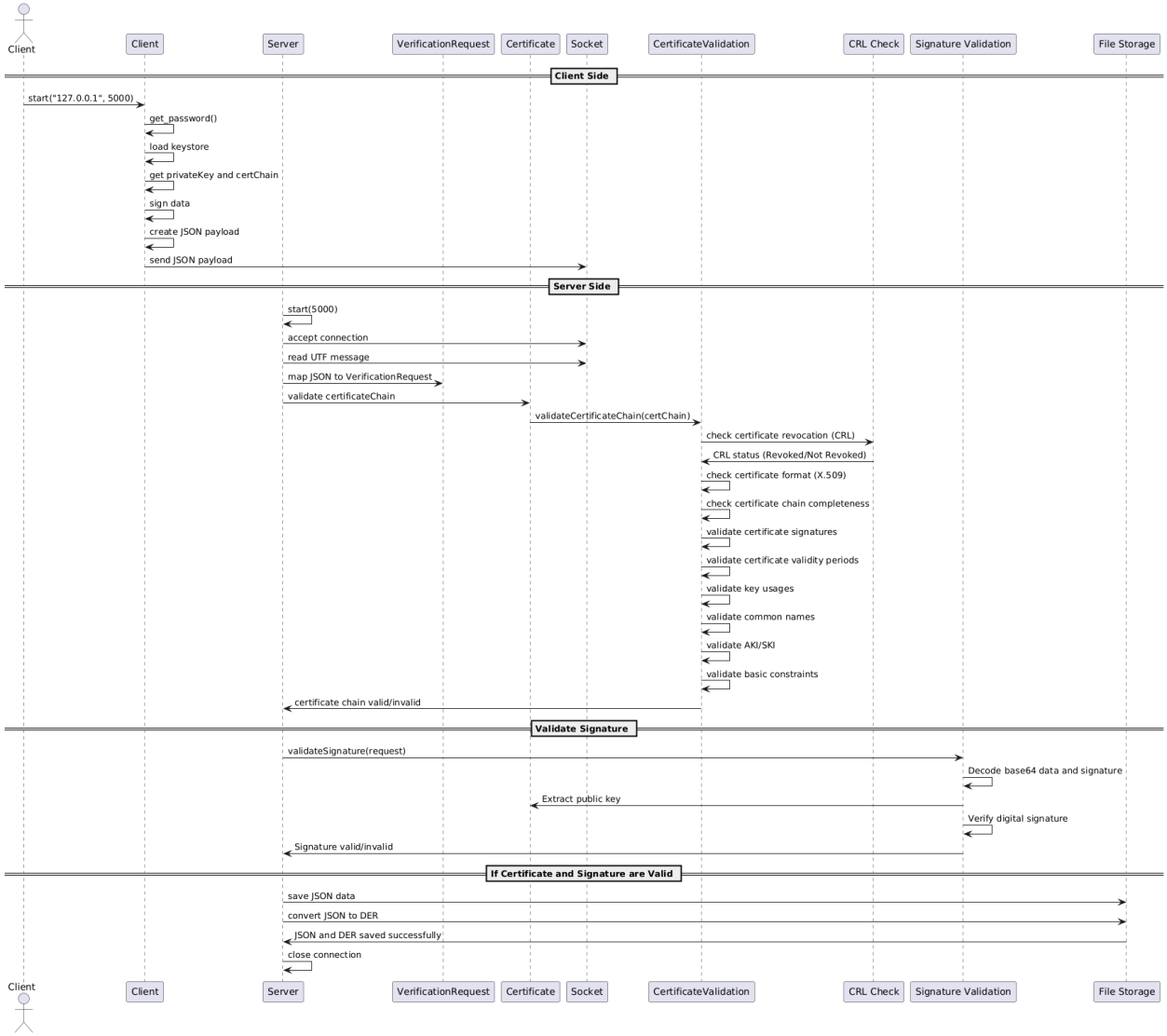


شکل۲

نمودار توالی و بررسی آن

نمودار توالی[۴] یکی از نمودارهای زبان مدل‌سازی یکپارچه است که روندی در یک پروژه را از زمانی که برنامه شروع می‌شود و تا زمانی که برنامه به پایان می‌رسد را مرحله به مرحله نمایش می‌دهد. در شکل ۳ تصویر نمودار طراحی شده را مشاهده می‌کنید. پروژه‌ی ما به دو بخش اصلی client و server تقسیم شده است که در ادامه قصد دارم مراحل نمایش داده شده در این نمودار را برای هر دو بخش پروژه به صورت کامل مجزا توضیح دهم.

[4] Sequence Diagram


Client
Client
Server
VerificationRequest
Certificate
Socket
CertificateValidation
CRL Check
Signature Validation
File Storage
Client Side
start("127.0.0.1", 5000)
get_password()
load keystore
get privateKey and certChain
sign data
create JSON payload
send JSON payload
Server Side
start(5000)
accept connection
read UTF message
map JSON to VerificationRequest
validate certificateChain
validateCertificateChain(certChain)
check certificate revocation (CRL)
CRL status (Revoked/Not Revoked)
check certificate format (X.509)
check certificate chain completeness
validate certificate signatures
validate certificate validity periods
validate key usages
validate common names
validate AKI/SKI
validate basic constraints
certificate chain valid/invalid
Validate Signature
validateSignature(request)
Decode base64 data and signature
Extract public key
Verify digital signature
Signature valid/invalid
If Certificate and Signature are Valid
save JSON data
convert JSON to DER
JSON and DER saved successfully
close connection
Client
Client
Server
VerificationRequest
Certificate
Socket
CertificateValidation
CRL Check
Signature Validation
File Storage

شکل۳

بخش Client

پس از ایجاد فایل PKCS12 در سمت کاربر نهایی داده‌ای داریم که بایستی در سمت سرور اعتبارسنجی ان انجام شود که در صورت داشتن امضا و گواهی معتبر، امضا و داده‌ی امضای شده برای پردازش‌های اتی به فرمت JSON یا DER در پایگاه داده‌های سرور ذخیره شود. لازم به ذکر است که از کلاس Socket برای ارتباط با سرور استفاده می‌شود.

برنامه نویسی این پروژه به زبان جاوا بوده و به طبع از کتابخانه‌های جاوا به خصوص کتابخانه‌یJava_Security بهره برده شده است. در این کد که مراحل ان را در نمودار توالی مشاهده می‌کنید در ابتدا یک شی از کلاس Client ساخته می‌شود و ارتباط با سرور با آدرس "۱۲۷.۰.۰.۱" و پورت "۵۰۰۰" برقرار می‌شود پس از ان از کاربر می‌خواهد که رمز عبور فایل client.p12 را که در قسمت قبل مراحل ساخت ان توضیح داده شده بود، وارد کند. در ادامه برای ذخیره سازی امن کلیدهای رمزنگاری از کلاس KeyStore استفاده می‌شود که این کلاس در جاوا این اجازه را به ما می‌دهد که یک KeyStore با فرمت خاص (PKCS12) ایجاد کنیم که برای بارگذاری و استفاده از کلید خصوصی و گواهی‌نامه‌ها از فایل client.p12 مورد استفاده قرار می‌گیرد. پس از بارگذاری کلید خصوصی کاربر نهایی و زنجیره گواهی‌ها استخراج می‌شود.

در ادامه این متن نمونه "This is some data to sign" تعریف و داده‌ها توسط کلاس Signature در جاوا و با استفاده از الگوریتم SHA256withRSA توسط کلید خصوصی امضا می‌شوند. این الگوریتم ترکیبی از الگوریتم SHA-256 برای درهم سازی داده‌ها و RSA برای امضای دیجیتال است. داده‌ها و گواهی‌ها پس از امضا به فرمت Base64 تبدیل می‌شوند تا برای ارسال در قالب JSON مناسب شوند. این تبدیل به عنوان یک روش عمومی برای کدگذاری داده‌ها به کار می‌رود. در نهایت، داده اصلی، داده‌ی امضا شده و گواهی‌ها در یک ساختار JSON قرار می‌گیرند که از سوی سرور می‌تواند پردازش شود.

بخش Server

در قسمت سرور ابتدا متد start را فراخوانی می‌کنیم. این متد مسئول راه اندازی سرور و منتظر ماندن برای اتصال کلاینت‌ها است. در این متد پورت ۵۰۰۰ به عنوان پارامتر ورودی به سرور داده می‌شود و سرور در این پورت منتظر اتصال می‌ماند. اگر کلاینتی در این پورت به سرور متصل شود، ارتباط برقرار می‌شود. در قسمت accept connection کاربر به سرور متصل می‌شود و اتصال برقرار می‌شود. در قسمت بعد سرور منتظر دریافت پیام از کاربر است که این پیام به صورت رشته UTF-8 توسط سرور دریافت می‌شود. در کد، با استفاده از متد inputStream.readUTF میتوان داده‌های دریافتی را به فرمت رشته UTF-8 خواند. در ادامه پیام دریافتی از کلاینت که به صورت JSON است، به یک شی از کلاس `VerificationRequest` تبدیل می‌شود. این تبدیل توسط کلاس `ObjectMapper` انجام می‌شود که وظیفه تبدیل داده‌های JSON به شی را بر عهده دارد. پس از ان سرور کار اصلی خود یعنی فرآیند اعتبارسنجی زنجیره گواهی‌ها را اغاز می‌کند. زنجیره گواهی‌ها به ترتیب اعتبار و صحت ان‌ها بررسی می‌شود. اگر هر کدام از مراحل اعتبار سنجی به درستی انجام نشود، خطا چاپ می‌شود و فرآیند ادامه نمی‌یابد. برای بارگذاری و تجزیه گواهی‌ها از کلاس CertificateFactory جاوا با فرمت X.509 استفاده می‌شود.

- قسمت اول اعتبار سنجی مربوط به بررسی وضعیت لغو هر کدام از گواهی‌ها در زنجیره گواهی است. برای این کار فایل CRL مناسب برای گواهی مورد نظر بارگذاری می‌شود.
- در قسمت دوم ان اگر گواهی لغو شده باشد، خطای "Certificate has been revoked" چاپ می‌شود.
- در قسمت سوم بررسی می‌شود که فرمت تمام گواهی‌های موجود در زنجیره گواهی از نوع X.509 باشد. این فرمت استاندارد برای گواهی‌های دیجیتال است که در آن داده‌ها به صورت خاصی قالب‌بندی شده‌اند. تمامی گواهی‌ها باید در فرمت X.509 باشند تا فرایند ادامه پیدا کند.
- در قسمت چهارم زنجیره‌ی گواهی‌ها بررسی می‌شود که ایا گواهی ریشه تا گواهی انتهایی به درستی متصل شده‌اند یا خیر. به عبارت دیگر، گواهی‌های پایین دستی باید توسط گواهی بالا دستی خود (Issuer) تایید شوند.
- در قسمت پنجم، امضای گواهی‌ها اعتبارسنجی می‌شود. برای هر گواهی در زنجیره گواهی، امضای گواهی صادرکننده (Issuer) آن بررسی می‌شود. این بررسی نشان می‌دهد که گواهی به طور صحیح توسط یک مرجع معتبر صادر شده است.
- در قسمت ششم تاریخ اعتبار هر گواهی بررسی می‌شود تا مطمئن شویم که گواهی‌ها در محدوده تاریخ‌های اعتبارشان قرار دارند. این بررسی با استفاده از متد `checkValidity` انجام می‌شود. کلاس

X509Certificate در جاوا از متدهای getNotBefore و getNotAfter برای بررسی دوره اعتبار استفاده می‌کند.

- در قسمت هفتم الحاقیه Key Usages در گواهی‌ها اعتبارسنجی می‌شود. گواهی‌های End Entity باید قابلیت امضای دیجیتال (DigitalSignature) داشته باشند و گواهی‌های میانی و ریشه باید دو پارامتر (KeyCertSign) و (CrlSign) CRL را داشته باشند.
- در قسمت هشتم نام‌های عمومی[5] در زنجیره گواهی‌ها بررسی می‌شود تا اطمینان حاصل شود که گواهی‌ها به درستی به یکدیگر مرتبط هستند. به طور خاص، نام گواهی‌ها باید به گونه‌ای باشد که همخوانی داشته باشد و زنجیره از گواهی‌های زیرین تا گواهی ریشه به درستی تایید شود. در واقع بررسی میشود که Issuer و Subject در زنجیره گواهی‌ها باهم مطابقت داشته باشند.
- در قسمت نهم تایید صحت ارتباط شناسه کلید موضوع [6]و شناسه کلید مرجع [7]را داریم که ان را Key Chaning می‌نامند. در این بخش، شناسه‌های کلید مربوط به گواهی‌ها استخراج شده و با یکدیگر مقایسه می‌شوند تا اطمینان حاصل شود که این دو شناسه هم‌خوانی دارند. اگر شناسه‌ها مطابقت نداشته باشند، به معنای خطا در ارتباط بین گواهی‌ها است. بدین صورت که ابتدا شناسه‌های AKI و SKI از هر گواهی استخراج می‌شود و بررسی می‌شود که آیا این دو شناسه مطابقت دارند یا خیر. برای هر جفت گواهی (مانند گواهی مشتری و گواهی میانه)، این مقایسه انجام می‌شود.
- در قسمت دهم بررسی ویژگی‌های "Basic Constraints" گواهی‌ها است که مشخص می‌کند ایا یک گواهی قادر به عمل به عنوان گواهی صادر کننده است یا خیر. همچنین این قسمت مشخص میکند که سلسه مراتب گواهی ها تا کجا میتواند ادامه یابد. در کد، برای بررسی این محدودیت‌ها از متد `getBasicConstraints` استفاده می‌شود که می‌تواند اطلاعاتی راجع به اینکه گواهی می‌تواند به‌عنوان CA عمل کند یا نه، ارائه دهد. اگر مقدار برگشتی از این متد مثبت باشد، به این معنی است که گواهی قادر به امضای گواهی‌های دیگر است و به‌عنوان یک گواهی صادرکننده شناخته می‌شود. در صورتی که مقدار آن منفی باشد، گواهی به‌عنوان یک "End Entity" یا موجودیت نهایی شناخته می‌شود که حق امضای گواهی‌های دیگر را ندارد.

[5] Common Name(CN)
[6] Subject Key Identifier(SKI)
[7] Authority Key Identifier(AKI)

اعتبارسنجی امضا

اگر تمام ۱۰ مورد ذکر شده اعتبار سنجی شود و صحت ان تایید شود در قسمت بعد صحت امضا را بررسی می‌کنیم. ورودی اصلی متد validateSignature یک شیء `VerificationRequest` است که حاوی داده‌ها و امضای دیجیتال است که باید بررسی شوند. داده‌ها و امضای دیجیتال در فرمت Base64 رمزگذاری شده‌اند. بنابراین، برای استفاده در صحت سنجی امضا، ابتدا باید این داده‌ها از فرمت Base64 به باینری (Byte) تبدیل شوند. در ادامه برای تایید امضا به کلید عمومی [8]مرتبط با ان گواهی نیاز داریم پس گواهی را نیز به فرمت باینری تبدیل میکنیم. این گواهی به یک شیء `X509Certificate` تبدیل شده و از آن کلید عمومی استخراج می‌شود. پس از استخراج کلید عمومی، فرآیند تایید امضا با استفاده از آن انجام می‌شود. در بخش نهایی اعتبارسنجی امضا، نتیجه بررسی امضا مشخص می‌شود. اگر امضا با داده‌ها همخوانی داشته باشد، نتیجه مثبت است و امضا تایید می‌شود. در غیر این صورت، امضا معتبر نیست.

ذخیره سازی

در بخش انتهایی اگر گواهی‌ها و امضا معتبر باشد داده‌ها به فرمت JSON و DER ذخیره می‌شوند. فایل با فرمت JSON دارای داده اصلی، امضا و زنجیره گواهی است. ان را به فرمت DER نیز ذخیره می‌کنیم که فقط دارای داده اصلی و داده‌ی امضا شده است. که سرور بتواند برای پردازش‌های اتی از ان استفاده کند. در انتها نیز اتصال به سرور متوقف می‌شود.

[8] Public Key

نمونه کد اجرایی

```
Server started
Waiting for client ...
Client accepted
Received message: {"data": "VGhpcyBpcyBzb21lIGRhdGEgdG8gc2lnbg==", "signature": "AlwJkwsQqxEH50mU+w/luAZtrlOS4D0aPHBI
Starting the certificate chain validation process...
Validating certificate against CRL...
Using intermediate CRL file: C:\Users\seraj\Documents\workspace-spring-tool-suite-4-4.26.0.RELEASE\CRL folders for se
Certificate is not revoked according to the CRL.
Validating certificate against CRL...
Using intermediate CRL file: C:\Users\seraj\Documents\workspace-spring-tool-suite-4-4.26.0.RELEASE\CRL folders for se
Certificate is not revoked according to the CRL.
Validating certificate against CRL...
Using root CRL file: C:\Users\seraj\Documents\workspace-spring-tool-suite-4-4.26.0.RELEASE\CRL folders for security p
Certificate is not revoked according to the CRL.
1. The certificates provided are not in the CRL list.
2. Certificate format checked: All certificates are in X.509 format.
3. The certificate chain is complete and ends with the root certificate.
4. Signatures of all certificates in the chain were verified.
5. The validity dates of all certificates have been checked and are valid.
6. Key Usage All certificates are valid.
7.Common Name (CN) validation passed: Name chaining is correct.
8.AKI and SKI validation passed between client and intermediate certificates.
8.AKI and SKI validation passed between intermediate and root certificates.
8.AKI and SKI validation passed between root and root certificates.
9. Basic Constraints Certificates Checked and are valid.
The certificate chain is valid.
Starting the signature verification process...
The signature is valid.
JSON file saved successfully C:\Users\seraj\Documents\workspace-spring-tool-suite-4-4.26.0.RELEASE\PKCS12 for securit
 DER file saved successfully: C:\Users\seraj\Documents\workspace-spring-tool-suite-4-4.26.0.RELEASE\PKCS12 for securi
Closing connection
```

شکل ۴

نتیجه گیری

در پایان این پروژه، با توجه به پیاده‌سازی سناریوی عملیاتی تولید و اعتبارسنجی امضای دیجیتال در یک سامانه نرم‌افزاری، مفاهیم و الزامات اساسی رمزنگاری کلید عمومی (Public Key Cryptography) و زیرساخت کلید عمومی (PKI) به‌طور کامل پیاده‌سازی و آزمون شد. در این راستا، استفاده از KeyStore برای ذخیره‌سازی کلید خصوصی و گواهی‌های الکترونیکی مبتنی بر استاندارد X509، به همراه الگوریتم RSA-SHA256 برای عملیات امضا، اعتبارسنجی و مدیریت زنجیره گواهی‌ها به‌طور عملیاتی بررسی گردید.

نتایج حاصل از پیاده‌سازی و تست‌های انجام شده نشان‌دهنده عملکرد صحیح سیستم در اعتبارسنجی امضای دیجیتال و صحت عملکرد زنجیره گواهی‌ها (Certificate Path) در سمت سرور است. علاوه بر این، استفاده از ابزارهایی نظیر OpenSSL در مراحل صدور گواهی و تولید فایل‌های KeyStore به‌طور مؤثر در محیط عملیاتی انجام شد.