Vitalware®
Vital Records Management

The World's Premier Vital Records Software

Vitalware Documentation

Encrypted Connections

Document Version 1

Vitalware Version 2.2.01

www.kesoftware.com

Contents

SECTION 1

Encrypted Connections

When using Vitalware over public networks it may be desirable to encrypt all data transferred between the Vitalware client and server. Vitalware 2.2.01 has been extended to support an encrypted connection between the client and server programs. The encrypted connection uses TLS v1.0 (Transport Layer Security) for the transmission of data, ensuring data integrity and security. The use of data encryption is optional and is not required for internal networks where the risk of unauthorised access to data is minimal. The Vitalware server may also be configured to accept connections only from clients who request data encryption. This provides system administrators with the ability to enforce data encryption, or not, as required.

How it works

The TLS v1.0 protocol uses Public Key Infrastructure (PKI). A key is a sequence of bytes, normally 40, 56, 64, 128 or 265 bits in length, which is used by a cipher (an encryption algorithm) to encrypt data. Using the same cipher with different keys will produce different output. Hence, the key is used to "lock" the encrypted data. In order to unlock the data, that is decrypt it, the right key is required. With public/private key infrastructure two keys are generated, a public key and a private key. Data encrypted with the public key requires the private key in order to be decrypted and data encrypted with the private key requires the public key in order to be decrypted. In other words the keys are symmetrical.

The private key must be kept safe to ensure data privacy. If someone has both the private and public keys, they can decrypt the data and so compromise data security. The public key may be made available to anyone without compromising security as the private key is required to decrypt data encrypted using the public key. The public key is wrapped in a digital certificate, which consists of:

- **Public Key**
- **Subject** - details about the owner of the certificate.
- **Serial Number** - unique number used to identify the certificate.
- **Issuer** - details about who verified and issued the certificate.
- **Valid Dates** - start and end dates for which the certificate is valid.
- **Key Usage** - purpose(s) for which the public key may be used.
- **Thumbprint** - a check-sum to ensure the certificate has not been modified.

In order for a digital certificate to be valid it is necessary to verify the details of the Issuer. A small number of companies are allowed to issue valid and verifiable certificates. When you want a public digital certificate you approach one of these companies and they verify your details before issuing your public digital certificate. They *sign* your certificate with their own private key, making them the Issuer. In order to read your digital certificate you need the Issuer's public key. The key is embedded in their digital certificate which is available freely. These Issuer public digital certificates are known as Certificate Authorities (CA). In order to verify your certificate it is necessary to determine who the Issuer is and locate their CA certificate. Using the public key in their certificate your certificate can now be decrypted and the check-sum verified to ensure it has not been altered.

The private key is stored on the Vitalware server. Login access to the Vitalware server is generally restricted to user `vw`, hence the key is not available for general access. The public digital certificate is also stored on the Vitalware server. All CA certificates are stored with the Vitalware client.

When a connection is initiated the Vitalware server sends its public digital certificate to the Vitalware client. The client uses the CA certificates stored locally to verify that the certificate is valid. The server's public digital certificate contains the full host name of the Vitalware server machine. The Vitalware client checks the host name against the machine to ensure it has not connected to a rogue server.

Once the Vitalware client has verified the server's public digital certificate it sends a random number to the server encrypted using the public key in the server's digital certificate. As the server is the only machine with the private key it can decrypt the random number. The number is used as a key to a cipher (an encryption algorithm). The cipher uses the key to encrypt all data between the client and server. As the client and server are the only two machines which know the encryption key, data security and integrity is guaranteed.

The complete steps required to establish an encrypted connection are:

- The Vitalware client connects to the Vitalware server requesting a secure connection. The client provides a list of ciphers it supports.
- The Vitalware server selects the strongest cipher it supports from the client's list and notifies the client.
- The Vitalware server sends its public digital certificate to the client. The certificate contains the server's host name, the Issuer used to create the certificate and the server's public encryption key.
- The Vitalware client looks up the CAs on its machine and verifies that the server's certificate is valid.
- The Vitalware client generates a random number and encrypts it with the server's public encryption key. The random number is sent to the server.
- The Vitalware server decrypts the random number using its private encryption key (known only by the server).
- The random number is used as a key for the selected cipher. All data transferred is now encrypted using the agreed cipher.

Vitalware allows public digital certificates to be:

- **Self signed**

  A certificate that is verified by itself, that is the Issuer certificate is the same as the certificate itself. Self signed certificates allow institutions to generate their own digital certificates without the need to have them authenticated by an outside authority. In order for the certificate to be verified the self signed certificate must exist on both the client and server machines, the client version being the CA certificate.
- **Root signed**

  A certificate verified by one of a select set of "root" certificates. A root certificate is distributed as part of the public key infrastructure and can be installed on client machines to provide certificate verification.
- **Chain signed**

  A certificate is verified by its Issuer certificate. The issuer certificate itself is verified by its issuer certificate and so on until either a root or self signed certificate is found.

Vitalware allows both the client and server machines to define a list of ciphers they will support. When a connection is created the strongest (that is hardest to break) cipher supported by both the client and server is selected. System administrators may restrict the ciphers available on the server, forcing the client to use very strong encryption only (e.g. 256 bit ciphers).

Requirements

Support for encrypted connections between the Vitalware client and server requires the following software versions:

- Texpress 8.2.009 or later
- Vitalware 2.2.01 or later
- TexAPI 6.0.02 or later
- TexJDBC 0.9.6 or later

If any software is earlier than the version listed above, Vitalware will drop back to using unencrypted connections. In order to use encrypted connections your System Administrator must create the required keys (public/private) and public digital certificate and install them on the Vitalware server. The CA certificates may also need to be installed on the Vitalware client.

Vitalware server setup

The default installation of the Vitalware server does not have encrypted connections enabled. The following files need to exist to enable encrypted connections:

- `server.key` Vitalware server's private key
- `server.crt` Vitalware server's public digital certificate
- `ciphers` list of ciphers the server will support

These files must be placed in a directory called `certs` under the Texpress `etc` directory. For a standard Vitalware installation this corresponds to:

`$EMUHOME/texpress/8.2/etc/certs`.

If the files are not found, Vitalware will not attempt to use encrypted connections.

See *Generating certificates* (page 9) for details on how to create `server.key` and `server.crt`.

See *Configuring ciphers* (page 19) for details about configuring `ciphers`.

Important!

The permissions on the `server.key` file should restrict access to read-only by user `root`. All other permissions should be disabled, that is the file owner should be `root` and the permissions should be `r--------`. If these permissions are not set, it is possible that someone may access the file and so compromise the integrity of the system!

As described in *Requirements* (page 4), the Vitalware server will drop back to an unencrypted connection if versions earlier than 2.2.01 of the Vitalware client are used. If you want to enforce encrypted connections the `-s` option should be added to the `texserver` command configured in `inetd/xinetd/svcs`.

For example, the following `inetd` entry will accept encrypted connections only:

```
vwclient stream tcp nowait root /home/vw/client/bin/vwrun vwrun
texserver -avw -i -L -t60 -s
```

Vitalware client setup

The Vitalware client needs to verify the Vitalware server's public digital certificate for an encrypted connection to be established. In order to verify the certificate the client must be able to locate a valid CA (Certificate Authority) certificate for the Issuer of the server's certificate. In the case of a certificate chain this must be the Issuer of the first or "root" certificate. There are two locations used to hold CA certificates:

1. The first is on the Vitalware server and is used by programs using either TexAPI or `texapi.pm` (a perl based interface to TexAPI). These programs include web services.
2. The second is on the Vitalware client's machine and is used by the Windows client.

CA certificates stored on the Vitalware server must be placed in the `$EMUPATH/etc/certs` directory. The certificates should be stored in files with a `.crt` extension. CA bundles are also supported. A bundle is simply the concatenation of a number of certificates into one file. An optional `ciphers` file may also exist in the certificates directory. If it exists, it should list the ciphers the Vitalware client is willing to support.

See *Configuring ciphers* (page 19) for details about configuring `ciphers`.

CA certificates required by the Vitalware Windows client should be stored in the `certs` directory under the location where the Vitalware executable (`vw.exe`) is installed. As with CA certificates stored on the Vitalware server, the files must have a `.crt` extension and CA bundles are supported. A `ciphers` file may also be supplied defining the ciphers the client is willing to use.

TexJDBC setup

As with the Vitalware client, TexJDBC needs to be able to verify the Vitalware server's public digital certificate. The required CA certificates must be stored in an accessible Java Key Store (JKS). The system key store is located at `$JAVA_HOME/jre/lib/security/cacerts`. A key store may have a password associated with it. The password allows the integrity of the stored certificates to be checked when they are accessed. The password is not required to access the key store. The location of the key store used may be altered by setting the following system properties:

`javax.net.ssl.trustStore`

The location of the Java Key Store file containing the CA certificates to use for verifying the server's certificate.

`javax.net.ssl.trustStorePassword`

The password to use to check the integrity of the Java Key Store.

For example, if you want to use a key store located at `/home/vw/etc/certs/cacerts` with a password of `vwstore`, you could invoke java using:

```
java -Djavax.net.ssl.trustStore=/home/vw/etc/certs/cacerts -
Djavax.net.ssl.trustStorePassword=vwstore -jar application.jar
```

The location and password of the key store may also be specified using the `trustStore` and `trustStorePassword` connection properties:

```
Properties props = new Properties();
props.setProperty("trustStore", "/home/vw/etc/certs/cacerts");
props.setProperty("trustStorePassword", "vwstore");
...
Connection conn =
DriverManager.getConnection("jdbc:texpress:socket", props);
```

The `keytool` command should be used to import a CA certificate into a java key store:

```
keytool -importcert -alias alias -file certfile -storetype JKS -
keystore keystore
```

where:

alias	is an arbitrary unique name used to define the certificate within the key store
certfile	is the file containing the CA certificate
keystore	is the location of the key store file into which the certificate is imported

If *keystore* does not exist, a new key store is created. You will be prompted for the password if the key store already exists, otherwise you will be asked to set the password for the key store created.

To list the certificates in a key store use:

```
keytool -list -v -keystore keystore
```

where *keystore* is the location of the key store file.

SECTION 2

Generating certificates

In this section we will look at how to generate a private/public key pair and how the public digital signature is created for the public key. As mentioned earlier Vitalware provides support for the following public certificates:

- **Self signed**
- **Root signed**
- **Chain signed**

Each of these types will be examined and appropriate commands provided for OpenSSL (via the `openssl` command).

Self signed

A self signed certificate is one where the certificate Issuer is the same as the certificate Subject. In other words the certificate is used to verify itself. In order for the certificate to be trusted the certificate must be included with the client CA certificates. Self signed certificates are used when you only need one certificate (for example if you only have one Vitalware server and all clients connect to that server). As the certificate is self signed it has not been verified by an external agency and so should be used for internal use only. The steps required to generate the required files are:

Step 1: Generate a private key

Create your private key. The key is a 1024 bit RSA key stored in PEM (Privacy Enhanced Mail, a Base64 encoding of the key) format. It is readable as ASCII text:

```
openssl genrsa -out server.key 1024
Generating RSA private key, 1024 bit long modulus
..++++++
...++++++
e is 65537 (0x10001)
```

The file `server.key` now contains your private key. Remember to keep it safe!

Step 2: Generate public digital certificate

Using the private key generated in the first step the public digital certificate is generated. A number of questions will be asked as part of the creation process. It is important that the Common Name (CN) is set to the full host name of your Vitalware server machine (e.g. `vw.institution.org`). Support for wild card host names is provided by replacing any leading component of the name with an asterisk (e.g `*.institution.org`, or `*.org`):

```
openssl req -new -x509 -key server.key -out server.crt -days 1095
You are about to be asked to enter information that will be
incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name
or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:AU
State or Province Name (full name) [Some-State]:Victoria
Locality Name (eg, city) []:Melbourne
Organization Name (eg, company) [Internet Widgits Pty Ltd]:KE
Software Pty Ltd
Organizational Unit Name (eg, section) []:
Common Name (eg, YOUR name) []:*.mel.kesoftware.com
Email Address []:info@mel.kesoftware.com
```

The resulting public digital certificate will be stored in PEM format in `server.crt`.

You can view the contents of the public certificate using:

openssl x509 -text -in server.crt

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            f5:02:b4:7d:c3:5b:ad:a7
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=AU, ST=Victoria, L=Melbourne, O=KE Software Pty
Ltd, CN=*.mel.kesoftware.com/emailAddress=info@mel.kesoftware.com
        Validity
            Not Before: Nov 19 11:47:46 2010 GMT
            Not After : Nov 18 11:47:46 2013 GMT
        Subject: C=AU, ST=Victoria, L=Melbourne, O=KE Software Pty
Ltd, CN=*.mel.kesoftware.com/emailAddress=info@mel.kesoftware.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:c4:c9:0f:04:8f:cd:98:5f:d9:c6:3b:00:54:b2:
                    88:07:9b:06:4c:ea:f2:41:74:a3:68:7d:16:2a:de:
                    cf:bb:cf:73:d5:97:f2:d8:4e:38:b1:7d:a8:94:48:
                    5b:4a:fd:92:3b:45:8c:1b:ce:85:e5:18:2e:c1:60:
                    db:4d:09:32:46:72:b4:a3:f1:f8:ab:96:4a:db:a5:
                    4c:32:6d:83:ee:f9:02:4e:8f:f1:8b:ba:b4:62:b6:
                    29:00:97:fb:3b:06:73:a2:56:5f:04:2c:79:3e:2e:
                    f8:1b:eb:f5:8b:a6:cf:6b:56:bd:74:16:cb:53:a6:
                    91:dd:ec:af:7a:77:40:b0:e5
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:

F6:72:9C:A4:91:C2:E2:51:70:26:05:FE:06:C3:E4:E9:4F:AF:A0:D5
            X509v3 Authority Key Identifier:

keyid:F6:72:9C:A4:91:C2:E2:51:70:26:05:FE:06:C3:E4:E9:4F:AF:A0:D5
                DirName:/C=AU/ST=Victoria/L=Melbourne/O=KE
Software Pty
Ltd/CN=*.mel.kesoftware.com/emailAddress=info@mel.kesoftware.com
                serial:F5:02:B4:7D:C3:5B:AD:A7

            X509v3 Basic Constraints:
                CA:TRUE
    Signature Algorithm: sha1WithRSAEncryption
        4c:89:a2:57:d2:3b:3a:11:70:63:41:56:4e:b6:36:8e:28:c5:
        29:d7:7d:22:86:c4:43:90:4f:74:d1:31:32:7f:39:d8:f3:20:
        80:05:53:99:cd:17:28:b8:16:3b:a3:9a:84:ae:2c:08:f5:b0:
        11:6a:d5:ba:42:81:9d:e7:36:8f:39:9d:b4:15:13:52:23:fc:
        37:f6:5c:88:39:f9:9b:d1:e0:06:82:3f:e2:56:a3:f3:83:55:
        4d:8b:7c:69:a3:bc:fb:3a:66:18:f2:07:67:bc:39:54:28:c3:
        eb:3e:5c:d9:89:d8:ea:c7:d2:c4:fe:87:ee:24:e0:ce:c0:4f:
        d1:e7
-----BEGIN CERTIFICATE-----
MIIDtTCCAx6gAwIBAgIJAPUCtH3DW62nMA0GCSqGSIb3DQEBBQUAMIGZMQswCQYD
VQQGEwJBVTERMA8GA1UECBMIVmljdG9yaWExEjAQBgNVBAcTCU1lbGJvdXJuZTEc
```

```
MBoGA1UEChMTS0UgU29mdHdhcmUgUHR5IEx0ZDEdMBsGA1UEAxQUKi5tZWwua2Vz
b2Z0d2FyZS5jb20xJjAkBgkqhkiG9w0BCQEWF2luZm9AbWVsLmtlc29mdHdhcmUu
Y29tMB4XDTEwMTExOTExNDc0NloXDTEzMTExODExNDc0NlowgZkxCzAJBgNVBAYT
AkFVMREwDwYDVQQIEwhWaWN0b3JpYTESMBAGA1UEBxMJTWVsYm91cm5lMRwwGgYD
VQQKExNLRSBTb2Z0d2FyZSBQdHkgTHRkMR0wGwYDVQQDFBQqLm1lbC5rZXNvZnR3
YXJlLmNvbTEmMCQGCSqGSIb3DQEJARYXaW5mb0BtZWwua2Vzb2Z0d2FyZS5jb20w
gZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAMTJDwSPzZhf2cY7AFSyiAebBkzq
8kF0o2h9Firez7vPc9WX8thOOLF9qJRIW0r9kjtFjBvOheUYLsFg200JMkZytKPx
+KuWStulTDJtg+75Ak6P8Yu6tGK2KQCX+zsGc6JWXwQseT4u+Bvr9Yumz2tWvXQW
y1Omkd3sr3p3QLDlAgMBAAGjggEBMIH+MB0GA1UdDgQWBBT2cpykkcLiUXAmBf4G
w+TpT6+g1TCBzgYDVR0jBIHGMIHDgBT2cpykkcLiUXAmBf4Gw+TpT6+g1aGBn6SB
nDCBmTELMAkGA1UEBhMCQVUxETAPBgNVBAgTCFZpY3RvcmlhMRIwEAYDVQQHEwlN
ZWxib3VybmUxHDAaBgNVBAoTE0tFIFNvZnR3YXJlIFB0eSBMdGQxHTAbBgNVBAMU
FCoubWVsLmtlc29mdHdhcmUuY29tMSYwJAYJKoZIhvcNAQkBFhdpbmZvQG1lbC5r
ZXNvZnR3YXJlLmNvbYIJAPUCtH3DW62nMAwGA1UdEwQFMAMBAf8wDQYJKoZIhvcN
AQEFBQADgYEATImiV9I7OhFwY0FWTrY2jijFKdd9IobEQ5BPdNExMn852PMggAVT
mc0XKLgWO6OahK4sCPWwEWrVukKBnec2jzmdtBUTUiP8N/ZciDn5m9HgBoI/4laj
84NVTYt8aaO8+zpmGPIHZ7w5VCjD6z5c2YnY6sfSxP6H7iTgzsBP0ec=
-----END CERTIFICATE-----
```

Step 3: Installing the files

We now have the two files we require:

- `server.key` - private key (must be kept safe)
- `server.crt` - self signed public digital certificate

On the Vitalware server these two files should be placed in the directory `$TEXHOME/etc/certs` where `$TEXHOME` contains the location where Texpress is installed. Suitable permissions should be set on the private key file:

```
mv server.key server.crt $TEXHOME/etc/certs
chmod 644 $TEXHOME/etc/certs/server.crt
su root
Password:
chown root $TEXHOME/etc/certs/server.key
chmod 400 $TEXHOME/etc/certs/server.key
exit
```

Next, the public certificate should be stored on the Vitalware server for use by API based programs (TexAPI and texql.pm):

```
cp $TEXHOME/etc/certs/server.crt $EMUPATH/etc/certs
chmod 644 $EMUPATH/etc/certs/server.crt
```

Finally, on Vitalware Windows client machines the `server.crt` file must be placed in a directory called `certs` in the same location as the Vitalware executable (`vw.exe`).

Now that all the required files are in the right place it is possible to connect using encrypted connections. As mentioned earlier, the `-s` option for `texserver` (page 5) may be used to enforce secure connections.

Root signed

A root signed certificate is a public digital certificate created and verified by an external entity. You forward a certificate request to the external entity and they return the signed public digital certificate. Root entities distribute their CA certificates (really just a special form of self signed certificate) for all to use, allowing any certificate signed by them to be verified. Root signed certificates are used when you need a verifiable certificate for external use.

The steps required to generate the required files are:

Step 1: Generate a private key

Create your private key. The key is a 1024 bit RSA key stored in PEM (Privacy Enhanced Mail, a Base64 encoding of the key) format. It is readable as ASCII text:

```
openssl genrsa -out server.key 1024
Generating RSA private key, 1024 bit long modulus
..++++++
...++++++
e is 65537 (0x10001)
```

The file `server.key` now contains your private key.

Step 2: Generate a certificate signing request

We use the private key generated in the first step to create a certificate signing request (CSR). The file generated will contain the Subject information without an Issuer being assigned, that is a certificate that has not yet been signed. The resulting file, `server.csr` is then sent to an external entity for signing (e.g. Verisign).

```
openssl req -new -key server.key -out server.csr
You are about to be asked to enter information that will be
incorporated into your certificate request.
What you are about to enter is what is called a Distinguished Name
or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:AU
State or Province Name (full name) [Some-State]:Victoria
Locality Name (eg, city) []:Melbourne
Organization Name (eg, company) [Internet Widgits Pty Ltd]:KE
Software Pty Ltd
Organizational Unit Name (eg, section) []:
Common Name (eg, YOUR name) []:*.mel.kesoftware.com
Email Address []:info@mel.kesoftware.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
```

```
An optional company name []:
```

Once the external entity has verified the Subject information in the request they will generate a public digital certificate and return it to you. You should save the certificate in a file called `server.crt`.

You can view the contents of the certificate signing request using:

```
openssl req -in server.csr -noout -text
Certificate Request:
    Data:
        Version: 0 (0x0)
        Subject: C=AU, ST=Victoria, L=Melbourne, O=KE Software Pty
Ltd, CN=*.mel.kesoftware.com/emailAddress=info@mel.kesoftware.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:c4:c9:0f:04:8f:cd:98:5f:d9:c6:3b:00:54:b2:
                    88:07:9b:06:4c:ea:f2:41:74:a3:68:7d:16:2a:de:
                    cf:bb:cf:73:d5:97:f2:d8:4e:38:b1:7d:a8:94:48:
                    5b:4a:fd:92:3b:45:8c:1b:ce:85:e5:18:2e:c1:60:
                    db:4d:09:32:46:72:b4:a3:f1:f8:ab:96:4a:db:a5:
                    4c:32:6d:83:ee:f9:02:4e:8f:f1:8b:ba:b4:62:b6:
                    29:00:97:fb:3b:06:73:a2:56:5f:04:2c:79:3e:2e:
                    f8:1b:eb:f5:8b:a6:cf:6b:56:bd:74:16:cb:53:a6:
                    91:dd:ec:af:7a:77:40:b0:e5
                Exponent: 65537 (0x10001)
        Attributes:
            a0:00
    Signature Algorithm: sha1WithRSAEncryption
        ae:e0:68:b8:fe:56:53:5e:f4:f4:e0:8d:19:2c:62:ee:ee:83:
        01:d2:8d:55:d0:2d:18:b8:18:0a:f2:5b:c4:a5:da:75:fd:ca:
        87:69:cd:3f:2e:7c:9e:a2:c2:b7:b1:4a:bd:85:2e:24:84:8d:
        cc:81:64:9d:0c:a4:ad:c4:c5:54:4d:cf:22:dc:08:51:3f:ed:
        6d:45:d6:91:e3:a6:c0:7e:2e:f0:0f:9e:be:70:ef:6a:f8:2c:
        93:59:8d:90:ca:23:c4:07:f9:ae:2c:09:03:fd:cf:43:d6:b7:
        8c:2e:48:96:28:98:5c:c3:e8:66:55:b3:4a:8d:bb:c8:d0:bb:
        c8:41
```

Step 3: Installing the files

The process for installing the two files `server.key` (private key) and `server.crt` (public certificate) is exactly the same as for a self signed certificate (page 12).

Chain signed

A chain signed certificate is a certificate that is not self signed and is not root signed. In order for the certificate to be verified, the Issuer of the certificate is verified, then the Issuer of the Issuer certificate is verified and so on until either a self signed or root signed certificate is encountered. If the top certificate is root signed, then the chain signed certificate has the same level of verification as if the certificate had been root signed directly. If the top certificate is self signed, then the level of verification is the same as for any other self signed certificate. Chain signed certificates are used where you will be generating multiple certificates and you only want to distribute one CA certificate to verify them all. The only CA required is the top level self signed or root signed certificate.

The steps below outline how to produce a self signed CA certificate that can then be used to sign all other certificates generated. If you require a root signed CA certificate, you need to generate a certificate signing request (as per the previous section) and have the external entity generate the CA certificate.

Step 1: Generate a self signed CA certificate

The first step creates a self signed CA certificate. The CA certificate is the "root" certificate used to sign (and hence verify) all other certificates we generate. The public digital certificate of the CA certificate needs to be installed on client machines. We only need to generate the CA certificate once.

```
echo "01" > ca.srl
openssl req -new -x509 -nodes -extensions v3_ca -keyout ca.key -out ca.crt -
days 365
Generating a 1024 bit RSA private key
..............................++++++
.........++++++
writing new private key to 'ca.key'
-----
You are about to be asked to enter information that will be
incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name
or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:AU
State or Province Name (full name) [Some-State]:Victoria
Locality Name (eg, city) []:Melbourne
Organization Name (eg, company) [Internet Widgits Pty Ltd]:KE
Software Pty Ltd
Organizational Unit Name (eg, section) []:
Common Name (eg, YOUR name) []:KE Software CA certificate
Email Address []:info@mel.kesoftware.com
```

Step 2: Generate a private key

Once we have the CA certificate we can generate a new certificate. The first step is to generate the private key:

```
openssl genrsa -out server.key 1024
Generating RSA private key, 1024 bit long modulus
.++++++
..++++++
e is 65537 (0x10001)
```

Step 3: Generate a certificate signing request

We use the private key generated in the previous step to create a certificate signing request (CSR). The file generated will contain the Subject information without an Issuer being assigned, that is a certificate that has not been signed. Make sure `Common Name` is set to the host name of your Vitalware server.

```
openssl req -new -key server.key -out server.csr
You are about to be asked to enter information that will be
incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name
or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:AU
State or Province Name (full name) [Some-State]:Victoria
Locality Name (eg, city) []:Melbourne
Organization Name (eg, company) [Internet Widgits Pty Ltd]:KE
Software Pty Ltd
Organizational Unit Name (eg, section) []:
Common Name (eg, YOUR name) []:*.mel.kesoftware.com
Email Address []:info@mel.kesoftware.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

Step 4: Sign the certificate with your CA certificate

Using our CA certificate we sign the CSR to produce our public digital certificate:

```
openssl x509 -CA ca.crt -CAkey ca.key -CAserial ca.srl -req -in server.csr -
out server.crt -days 365
Signature ok
subject=/C=AU/ST=Victoria/L=Melbourne/O=KE Software Pty
Ltd/CN=*.mel.kesoftware.com/emailAddress=info@mel.kesoftware.com
Getting CA Private Key
```

Step 5: Creating the certificate chain

A certificate chain is simply the concatenation of all the signing public certificates from the certificate just generated to the root certificate (there may be any number of intermediate certificates). As we only have one CA in the chain in this example we do not need to concatenate the certificates, however if more than one CA has been used all certificates in the chain must be placed in one file. While not required for this example we will concatenate the certificates anyway (it does not hurt):

```
cat server.crt ca.crt > chain.crt
```

Step 6: Installing the files

First we install the private key and chain certificates on the Vitalware server:

```
mv server.key $TEXHOME/etc/certs/server.key
mv chain.crt $TEXHOME/etc/certs/server.crt
chmod 644 $TEXHOME/etc/certs/server.crt
su root
Password:
chown root $TEXHOME/etc/certs/server.key
chmod 400 $TEXHOME/etc/certs/server.key
exit
```

Next, the public CA certificate should be stored on the Vitalware server for use by API based programs (TexAPI and texql.pm):

```
cp ca.crt $EMUPATH/etc/certs
chmod 644 $EMUPATH/etc/certs/ca.crt
```

Finally, on Vitalware Windows client machines the `ca.crt` file must be placed in a directory called `certs` in the same location as the Vitalware executable (`vw.exe`).

Now that the CA certificate is installed on the Vitalware clients there is no need to add any further files when generating new certificates signed by the same CA certificate.

SECTION 3

Configuring ciphers

When an encrypted connection is formed the client and server negotiate to determine the highest level of encryption on which they can both agree. A list of ciphers may be set for the server and/or client allowing System Administrators to enforce a certain level of encryption. As the strongest cipher is chosen as part of the connection negotiation it is not necessary to restrict the list of ciphers used in most cases. The ciphers to use can be set at three levels:

- Server
- TexAPI/texapi.pm
- TexJDBC

Server

The list of ciphers the server will support is found in a file called `ciphers`. The file is located in the `$TEXHOME/etc/certs` directory. The format of the file is a colon separated list of cipher names. For details on what ciphers are supported and the exact format of the setting see the Ciphers section of the OpenSSL documentation.

For example, to enforce the use of MD5 ciphers the following cipher file could be used:

```
#
#  The allowable ciphers for use between the client and server are
#  defined by the last line in this file. See ciphers(1) in
OpenSSL
#  (http://www.openssl.org/docs/apps/ciphers.html) for the format
of
#  statement detailing the ciphers to use.
#
MD5
```

TexAPI / texapi.pm

To set the ciphers supported by the Windows client and client side programs using `texapi.pm` the `TEXCIPHERS` environment variable should be used. For example, to enforce the use of MD5 ciphers the following setting could be used:

```
TEXCIPHERS="MD5"
export TEXCIPHERS
```

It is also possible to set the ciphers when using TexAPI directly. The `Ciphers` member of the `TEXSESSINFO` structure may be used:

```
TEXSESSINFO info;
TEXSESSION  session;

...
info.Ciphers = "MD5";
...
TexSessConnect(&info, &session);
...

For texapi.pm the Ciphers key is used:

my $session = ke::texapi->new(
{
        ...
        Ciphers => 'MD5',
        ...
});
```

TexJDBC

When using TexJDBC the `ciphers` connection property may be set to restrict the ciphers used for a connection:

```
Properties props = new Properties();

props.setProperty("ciphers", "MD5");
...

Connection conn =
DriverManager.getConnection("jdbc:texpress:socket", props);
```

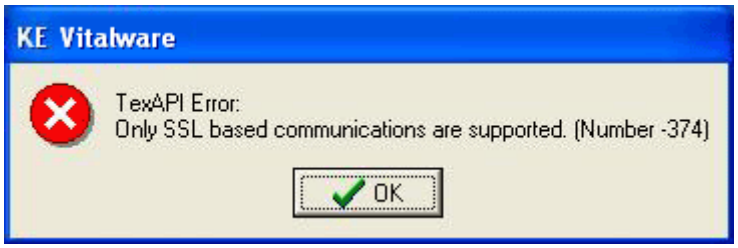
SECTION 4

Common Errors

When configuring the use of encrypted connections a number of common errors may occur. In this section a description of these errors is given, along with possible solutions.

Client does not support SSL

The System Administrator may configure the Vitalware server to accept encrypted connections only. The `-s` option for `texserver` will force the server to only accept connections where encryption is enabled. If the Vitalware client is a version prior to 2.2.01, then encryption is not supported and the following error message is displayed:



The solution is to upgrade the Vitalware client to version 2.2.01 or greater.

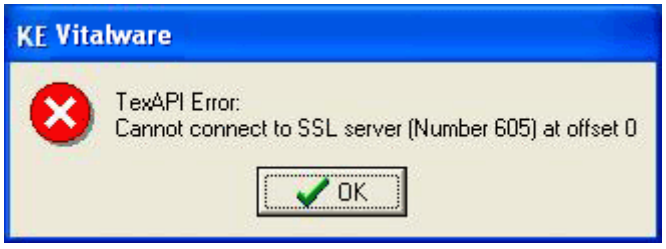
Server certificates not installed

If the System Administrator has turned on the `-s` option for `texserver`, thus forcing encrypted connections, and the server side certificates (`server.crt` and `server.key`) are not installed, the following error message is displayed:

The solution is to generate the private key (`server.key`) and public digital certificate (`server.crt`) and place them in the correct location (`$TEXHOME/etc/certs`).

Server/Client protocol error

The initiation of an encrypted connection involves a handshake between the client and server programs. If an error occurs as part of the handshake, the following message is displayed:



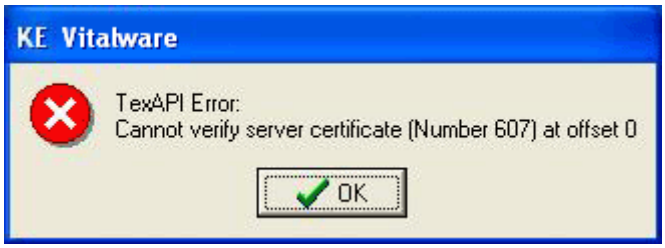
There are many reasons why a protocol error may occur. The most common are:

- The private key file `$TEXHOME/etc/certs/server.key` cannot be read. The error implies the contents of the private key file are corrupted.
- The public digital certificate file `$TEXHOME/etc/certs/server.crt` cannot be read. The error implies the contents of the public certificate file are corrupted.
- A private key/public certificate mismatch. The public digital key `server.crt` was not generated using the private key found in `server.key`. Either the private key or public certificate is incorrect.
- An acceptable cipher cannot be found. The client and server cannot agree on a cipher to use for the connection encryption. The server ciphers file should be altered to match a client cipher or vice versa.

Server side debugging may be required to determine the exact cause of the error. The Texpress debug flags `s15,16` will output the reason for the protocol error. For details on how to set Texpress debug flags please contact KE Software support.

Cannot verify certificate

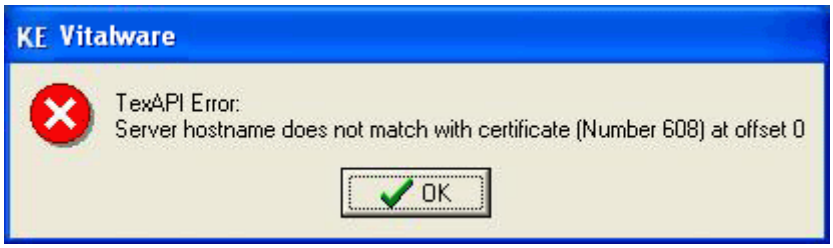
In order for the client to verify the server's certificate the client must have a copy of the server's top level public CA certificate. If the top level certificate is not installed on the client, the following error is displayed:



The solution is to install the top level CA certificate on the client. The certificate should be placed in a directory called `certs` located in the same place as the Vitalware client executable.

Bad host name

The *Common Name* field of the server's public digital certificate must contain the host name of the Vitalware server. Wild cards are supported using the asterisk character. If the *Common Name* field of the server's certificate does not match the host name to which the client is connected, the following error is displayed:



The solution is to fix up your DNS so that the host name of the Vitalware server matches the host name stored in the server's certificate. If this is not possible, a new server certificate should be generated with the correct host name.

Index