Apache NiFi 3

Apache NiFi Toolkit

Date of Publish: 2018-11-15

https://docs.hortonworks.com/

Contents

Security Configuration

NiFi provides several different configuration options for security purposes. The most important properties are those under the "security properties" heading in the nifi.properties file. In order to run securely, the following properties must be set:

Property Name	Description
nifi.security.truststorePasswd	The password for the Truststore.
nifi.security.keystore	Filename of the Keystore that contains the server's private key.
nifi.security.keystoreType	The type of Keystore. Must be either PKCS12 or JKS. JKS is the preferred type, PKCS12 files will be loaded with BouncyCastle provider.
nifi.security.keystorePasswd	The password for the Keystore.
nifi.security.keyPasswd	The password for the certificate in the Keystore. If not set, the value of nifi.security.keystorePasswd will be used.
nifi.security.truststore	Filename of the Truststore that will be used to authorize those connecting to NiFi. A secured instance with no Truststore will refuse all incoming connections.
nifi.security.truststoreType	The type of the Truststore. Must be either PKCS12 or JKS. JKS is the preferred type, PKCS12 files will be loaded with BouncyCastle provider.

Once the above properties have been configured, we can enable the User Interface to be accessed over HTTPS instead of HTTP. This is accomplished by setting the nifi.web.https.host and nifi.web.https.port properties. The nifi.web.https.host property indicates which hostname the server should run on. If it is desired that the HTTPS interface be accessible from all network interfaces, a value of 0.0.0.0 should be used. To allow admins to configure the application to run only on specific network interfaces, nifi.web.http.network.interface* or nifi.web.https.network.interface* properties can be specified.

Note: It is important when enabling HTTPS that the nifi.web.http.port property be unset. NiFi only supports running on HTTP or HTTPS, not both simultaneously.

NiFi's web server will REQUIRE certificate based client authentication for users accessing the User Interface when not configured with an alternative authentication mechanism which would require one way SSL (for instance LDAP, OpenId Connect, etc). Enabling an alternative authentication mechanism will configure the web server to WANT certificate base client authentication. This will allow it to support users with certificates and those without that may be logging in with credentials.

Now that the User Interface has been secured, we can easily secure Site-to-Site connections and inner-cluster communications, as well. This is accomplished by setting the nifi.remote.input.secure and nifi.cluster.protocol.is.secure properties, respectively, to true. These communications will always REQUIRE two way SSL as the nodes will use their configured keystore/truststore for authentication.

TLS Generation Toolkit

In order to facilitate the secure setup of NiFi, you can use the tls-toolkit command line utility to automatically generate the required keystores, truststore, and relevant configuration files. This is especially useful for securing multiple NiFi nodes, which can be a tedious and error-prone process.

Wildcard certificates (i.e. two nodes node1.nifi.apache.org and node2.nifi.apache.org being assigned the same certificate with a CN or SAN entry of *.nifi.apache.org) are not officially supported and not recommended. There are numerous disadvantages to using wildcard certificates, and a cluster working with wildcard certificates has occurred in previous versions out of lucky accidents, not intentional support. Wildcard SAN entries are acceptable if each cert maintains an additional unique SAN entry and CN entry.

Potential issues with wildcard certificates

- In many places throughout the codebase, cluster communications use certificate identities many times to identify a node, and if the certificate simply presents a wildcard DN, that doesn't resolve to a specific node
- Admins may need to provide a custom node identity in authorizers.xml for *.nifi.apache.org because all proxy actions only resolve to the cert DN
- Admins have no traceability into which node performed an action because they all resolve to the same DN
- Admins running multiple instances on the same machine using different ports to identify them can accidentally put node1 hostname with node2 port, and the address will resolve fine because it's using the same certificate, but the host header handler will block it because the node1 hostname is (correctly) not listed as an acceptable host for node2 instance
- If the wildcard certificate is compromised, all nodes are compromised

Note: JKS keystores and truststores are recommended for NiFi. This tool allows the specification of other keystore types on the command line but will ignore a type of PKCS12 for use as the truststore because that format has some compatibility issues between BouncyCastle and Oracle implementations.

The tls-toolkit command line tool has two primary modes of operation:

1. Standalone - generates the certificate authority, keystores, truststores, and nifi.properties files in one command.
2. Client/Server mode - uses a Certificate Authority Server that accepts Certificate Signing Requests from clients, signs them, and sends the resulting certificates back. Both client and server validate the other's identity through a shared secret.

Standalone

Standalone mode is invoked by running ./bin/tls-toolkit.sh standalone -h which prints the usage information along with descriptions of options that can be specified.

You can use the following command line options with the tls-toolkit in standalone mode:

- -a,--keyAlgorithm <arg> Algorithm to use for generated keys (default: RSA)
- --additionalCACertificate <arg> Path to additional CA certificate (used to sign toolkit CA certificate) in PEM format if necessary
- -B,--clientCertPassword <arg> Password for client certificate. Must either be one value or one for each client DN (auto-generate if not specified)
- -c,--certificateAuthorityHostname <arg> Hostname of NiFi Certificate Authority (default: localhost)
- -C,--clientCertDn <arg> Generate client certificate suitable for use in browser with specified DN (Can be specified multiple times)
- -d,--days <arg> Number of days issued certificate should be valid for (default: 1095)
- -f,--nifiPropertiesFile <arg> Base nifi.properties file to update (Embedded file identical to the one in a default NiFi install will be used if not specified)
- -g,--differentKeyAndKeystorePasswords Use different generated password for the key and the keystore
- -G,--globalPortSequence <arg> Use sequential ports that are calculated for all hosts according to the provided hostname expressions (Can be specified multiple times, MUST BE SAME FROM RUN TO RUN)
- -h,--help Print help and exit
- -k,--keySize <arg> Number of bits for generated keys (default: 2048)
- -K,--keyPassword <arg> Key password to use. Must either be one value or one for each host (auto-generate if not specified)
- -n,--hostnames <arg> Comma separated list of hostnames
- --nifiDnPrefix <arg> String to prepend to hostname(s) when determining DN (default: CN=)

- --nifiDnSuffix <arg> String to append to hostname(s) when determining DN (default: , OU=NIFI)
- -o,--outputDirectory <arg> The directory to output keystores, truststore, config files (default: ../bin)
- -O,--isOverwrite Overwrite existing host output
- -P,--trustStorePassword <arg> Keystore password to use. Must either be one value or one for each host (auto-generate if not specified)
- -s,--signingAlgorithm <arg> Algorithm to use for signing certificates (default: SHA256WITHRSA)
- -S,--keyStorePassword <arg> Keystore password to use. Must either be one value or one for each host (auto-generate if not specified)
- --subjectAlternativeNames <arg> Comma-separated list of domains to use as Subject Alternative Names in the certificate
- -T,--keyStoreType <arg> The type of keystores to generate (default: jks)

Hostname Patterns:

- Square brackets can be used in order to easily specify a range of hostnames. Example: [01-20]
- Parentheses can be used in order to specify that more than one NiFi instance will run on the given host(s). Example: (5)

Examples:

Create 4 sets of keystore, truststore, nifi.properties for localhost along with a client certificate with the given DN:

```
bin/tls-toolkit.sh standalone -n 'localhost(4)' -C 'CN=username,OU=NIFI'
```

Create keystore, truststore, nifi.properties for 10 NiFi hostnames in each of 4 subdomains:

```
bin/tls-toolkit.sh standalone -n 'nifi[01-10].subdomain[1-4].domain'
```

Create 2 sets of keystore, truststore, nifi.properties for 10 NiFi hostnames in each of 4 subdomains along with a client certificate with the given DN:

```
bin/tls-toolkit.sh standalone -n 'nifi[01-10].subdomain[1-4].domain(2)' -C
 'CN=username,OU=NIFI'
```

Client/Server

Client/Server mode relies on a long-running Certificate Authority (CA) to issue certificates. The CA can be stopped when you're not bringing nodes online.

Server

The CA server is invoked by running ./bin/tls-toolkit.sh server -h which prints the usage information along with descriptions of options that can be specified.

You can use the following command line options with the tls-toolkit in server mode:

- -a,--keyAlgorithm <arg> Algorithm to use for generated keys (default: RSA)
- --configJsonIn <arg> The place to read configuration info from (defaults to the value of configJson), implies useConfigJson if set (default: configJson value)
- -d,--days <arg> Number of days issued certificate should be valid for (default: 1095)
- -D,--dn <arg> The dn to use for the CA certificate (default: CN=YOUR_CA_HOSTNAME,OU=NIFI)
- -f,--configJson <arg> The place to write configuration info (default: config.json)
- -F,--useConfigJson Flag specifying that all configuration is read from configJson to facilitate automated use (otherwise configJson will only be written to)
- -g,--differentKeyAndKeystorePasswords Use different generated password for the key and the keystore
- -h,--help Print help and exit
- -k,--keySize <arg> Number of bits for generated keys (default: 2048)
- -p,--PORT <arg> The port for the Certificate Authority to listen on (default: 8443)

- -s,--signingAlgorithm <arg> Algorithm to use for signing certificates (default: SHA256WITHRSA)
- -T,--keyStoreType <arg> The type of keystores to generate (default: jks)
- -t,--token <arg> The token to use to prevent MITM (required and must be same as one used by clients)

Client

The client can be used to request new Certificates from the CA. The client utility generates a keypair and Certificate Signing Request (CSR) and sends the CSR to the Certificate Authority. The client is invoked by running ./bin/tls-toolkit.sh client -h which prints the usage information along with descriptions of options that can be specified.

You can use the following command line options with the tls-toolkit in client mode:

- -a,--keyAlgorithm <arg> Algorithm to use for generated keys (default: RSA)
- -c,--certificateAuthorityHostname <arg> Hostname of NiFi Certificate Authority (default: localhost)
- -C,--certificateDirectory <arg> The directory to write the CA certificate (default: .)
- --configJsonIn <arg> The place to read configuration info from, implies useConfigJson if set (default: configJson value)
- -D,--dn <arg> The DN to use for the client certificate (default: CN=<localhost name>,OU=NIFI) (this is auto-populated by the tool)
- -f,--configJson <arg> The place to write configuration info (default: config.json)
- -F,--useConfigJson Flag specifying that all configuration is read from configJson to facilitate automated use (otherwise configJson will only be written to)
- -g,--differentKeyAndKeystorePasswords Use different generated password for the key and the keystore
- -h,--help Print help and exit
- -k,--keySize <arg> Number of bits for generated keys (default: 2048)
- -p,--PORT <arg> The port to use to communicate with the Certificate Authority (default: 8443)
- --subjectAlternativeNames <arg> Comma-separated list of domains to use as Subject Alternative Names in the certificate
- -T,--keyStoreType <arg> The type of keystores to generate (default: jks)
- -t,--token <arg> The token to use to prevent MITM (required and must be same as one used by CA)

After running the client you will have the CA's certificate, a keystore, a truststore, and a config.json with information about them as well as their passwords.

For a client certificate that can be easily imported into the browser, specify: -T PKCS12.

Using An Existing Intermediate Certificate Authority (CA)

In some enterprise scenarios, a security/IT team may provide a signing certificate that has already been signed by the organization's certificate authority (CA). This intermediate CA can be used to sign the node (sometimes referred to as leaf) certificates that will be installed on each NiFi node, or the client certificates used to identify users. In order to inject the existing signing certificate into the toolkit process, follow these steps:

1. Generate or obtain the signed intermediate CA keys in the following format (see additional commands below):
    - Public certificate in PEM format: nifi-cert.pem
    - Private key in PEM format: nifi-key.key
2. Place the files in the toolkit working directory. This is the directory where the tool is configured to output the signed certificates. This is not necessarily the directory where the binary is located or invoked.
    - For example, given the following scenario, the toolkit command can be run from its location as long as the output directory -o is ../hardcoded/, and the existing nifi-cert.pem and nifi-key.key will be used.
        - e.g. $ ./toolkit/bin/tls-toolkit.sh standalone -o ./hardcoded/ -n 'node4.nifi.apache.org' -P thisIsABadPassword -S thisIsABadPassword -O will result in a new directory at ./hardcoded/node4.nifi.apache.org with a keystore and truststore containing a certificate signed by ./hardcoded/nifi-key.key
    - If the -o argument is not provided, the default working directory (.) must contain nifi-cert.pem and nifi-key.key

- e.g. $ cd ./hardcoded/ && ../toolkit/bin/tls-toolkit.sh standalone -n 'node5.nifi.apache.org' -P thisIsABadPassword -S thisIsABadPassword -O

```
        # Example directory structure *before* commands above are run

## 0s @ 18:07:58 $ tree -L 2
.
### hardcoded
#   ### CN=myusername.hardcoded_OU=NiFi.p12
#   ### CN=myusername.hardcoded_OU=NiFi.password
#   ### nifi-cert.pem
#   ### nifi-key.key
#   ### node1.nifi.apache.org
#   ### node2.nifi.apache.org
#   ### node3.nifi.apache.org
### toolkit
    ### LICENSE
    ### NOTICE
    ### README
    ### bin
    ### conf
    ### docs
    ### lib
```

Additional Commands

The nifi-cert.pem and nifi-key.key files should be ASCII-armored (Base64-encoded ASCII) files containing the CA public certificate and private key respectively. Here are sample files of each to show the expected format:

nifi-cert.pem

```
        # The first command shows the actual content of the encoded file,
 and the second parses it and shows the internal values

.../certs $ more nifi-cert.pem
-----BEGIN CERTIFICATE-----
MIIDZTCCAk2gAwIBAgIKAWTeM3kDAAAAADANBgkqhkiG9w0BAQsFADAxMQ0wCwYD
VQQLDAROSUZJMSAwHgYDVQQDDBduaWZpLWNhLm5pZmkuYXBhY2hlLm9yZzAeFw0x
ODA3MjgwMDA0MzJaFw0yMTA3MjcwMDA0MzJaMDExDTALBgNVBAsMBE5JRkkxIDAe
BgNVBAMMF25pZmktY2EubmlmaS5hcGFjaGUub3JnMIIBIjANBgkqhkiG9w0BAQEF
AAOCAQ8AMIIBCgKCAQEAqkVrrC+AkFbjnCpupSy84tTFDsRVUIWYj/k2pVwC145M
3bpr0pRCzLuzovAjFCmT5L+isTvNjhionsqif07Ebd/M2psYE/Rih2MULsX6KgRe
1nRUiBeKF08hlmSBMGDFPj39yDzE/V9edxV/KGjRqVgw/Qy0vwaS5uWdXnLDhzoV
4/Mz7lGmYoMasZ1uexlH93jjBl1+EFL2Xoa06oLbEojJ9TKaWhpG8ietEedf7WM0
zqBEz2kHo9ddFk9yxiCkT4SUKnDWkhwc/o6us1vEXoSw+tmufHY/A3gVihjWPIGz
qyLFl9JuN7CyJepkVVqTdskBG7S85G/kBlizUj5jOwIDAQABo38wfTAOBgNVHQ8B
Af8EBAMCAf4wDAYDVR0TBAUwAwEB/zAdBgNVHQ4EFgQUKiWBKbMMQ1zUabD4gI7L
VOWOcy0wHwYDVR0jBBgwFoAUKiWBKbMMQ1zUabD4gI7LVOWOcy0wHQYDVR0lBBYw
FAYIKwYBBQUHAwIGCCsGAQUFBwMBMA0GCSqGSIb3DQEBCwUAA4IBAQAxfHFIZLOw
mwIqnSI/ir8f/uzDMq06APHGdhdeIKV0HR74BtK95KFg42zeXxAEFeic98PC/FPV
tKpm2WUa1slMB+oP27cRx5Znr2+pktaqnM7f2JgMeJ8bduNH3RUkr9jwgkcJRwyC
I4fwHC9k18aizNdOf2q2UgQXxNXaLYPe17deuNVwwrflMgeFfVrwbT2uPJTMRi1D
FQyc6haF4vsOSSRzE6OyDoc+/1PpyPW75OeSXeVCbc3AEAvRuTZMBQvBQUqVM51e
MDG+K3rCeieSBPOnGNrEC/PiA/CvaMXBEog+xPAw1SgYfuCz4rlM3BdRa54z3+oO
lc8xbzd7w8Q3
-----END CERTIFICATE-----
.../certs $ openssl x509 -in nifi-cert.pem -text -noout
Certificate:
    Data:
```

```
        Version: 3 (0x2)
        Serial Number:
            01:64:de:33:79:03:00:00:00:00
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: OU=NIFI, CN=nifi-ca.nifi.apache.org
        Validity
            Not Before: Jul 28 00:04:32 2018 GMT
            Not After : Jul 27 00:04:32 2021 GMT
        Subject: OU=NIFI, CN=nifi-ca.nifi.apache.org
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:aa:45:6b:ac:2f:80:90:56:e3:9c:2a:6e:a5:2c:
                    bc:e2:d4:c5:0e:c4:55:50:85:98:8f:f9:36:a5:5c:
                    02:d7:8e:4c:dd:ba:6b:d2:94:42:cc:bb:b3:a2:f0:
                    23:14:29:93:e4:bf:a2:b1:3b:cd:8e:18:a8:9e:ca:
                    a2:7f:4e:c4:6d:df:cc:da:9b:18:13:f4:62:87:63:
                    14:2e:c5:fa:2a:04:5e:d6:74:54:88:17:8a:17:4f:
                    21:96:64:81:30:60:c5:3e:3d:fd:c8:3c:c4:fd:5f:
                    5e:77:15:7f:28:68:d1:a9:58:30:fd:0c:b4:bf:06:
                    92:e6:e5:9d:5e:72:c3:87:3a:15:e3:f3:33:ee:51:
                    a6:62:83:1a:b1:9d:6e:7b:19:47:f7:78:e3:06:5d:
                    7e:10:52:f6:5e:86:b4:ea:82:db:12:88:c9:f5:32:
                    9a:5a:1a:46:f2:27:ad:11:e7:5f:ed:63:34:ce:a0:
                    44:cf:69:07:a3:d7:5d:16:4f:72:c6:20:a4:4f:84:
                    94:2a:70:d6:92:1c:1c:fe:8e:ae:b3:5b:c4:5e:84:
                    b0:fa:d9:ae:7c:76:3f:03:78:15:8a:18:d6:3c:81:
                    b3:ab:22:c5:97:d2:6e:37:b0:b2:25:ea:64:55:5a:
                    93:76:c9:01:1b:b4:bc:e4:6f:e4:06:58:b3:52:3e:
                    63:3b
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Key Usage: critical
                Digital Signature, Non Repudiation, Key Encipherment, Data
Encipherment, Key Agreement, Certificate Sign, CRL Sign
            X509v3 Basic Constraints:
                CA:TRUE
            X509v3 Subject Key Identifier:
                2A:25:81:29:B3:0C:43:5C:D4:69:B0:F8:80:8E:CB:54:E5:8E:73:2D
            X509v3 Authority Key Identifier:

keyid:2A:25:81:29:B3:0C:43:5C:D4:69:B0:F8:80:8E:CB:54:E5:8E:73:2D

            X509v3 Extended Key Usage:
                TLS Web Client Authentication, TLS Web Server Authentication
    Signature Algorithm: sha256WithRSAEncryption
         31:7c:71:48:64:b3:b0:9b:02:2a:9d:22:3f:8a:bf:1f:fe:ec:
         c3:32:ad:3a:00:f1:c6:76:17:5e:20:a5:74:1d:1e:f8:06:d2:
         bd:e4:a1:60:e3:6c:de:5f:10:04:15:e8:9c:f7:c3:c2:fc:53:
         d5:b4:aa:66:d9:65:1a:d6:c9:4c:07:ea:0f:db:b7:11:c7:96:
         67:af:6f:a9:92:d6:aa:9c:ce:df:d8:98:0c:78:9f:1b:76:e3:
         47:dd:15:24:af:d8:f0:82:47:09:47:0c:82:23:87:f0:1c:2f:
         64:d7:c6:a2:cc:d7:4e:7f:6a:b6:52:04:17:c4:d5:da:2d:83:
         de:d7:b7:5e:b8:d5:70:c2:b7:e5:32:07:85:7d:5a:f0:6d:3d:
         ae:3c:94:cc:46:2d:43:15:0c:9c:ea:16:85:e2:fb:0e:49:24:
         73:13:a3:b2:0e:87:3e:ff:53:e9:c8:f5:bb:e4:e7:92:5d:e5:
         42:6d:cd:c0:10:0b:d1:b9:36:4c:05:0b:c1:41:4a:95:33:9d:
         5e:30:31:be:2b:7a:c2:7a:27:92:04:f3:a7:18:da:c4:0b:f3:
         e2:03:f0:af:68:c5:c1:12:88:3e:c4:f0:30:d5:28:18:7e:e0:
         b3:e2:b9:4c:dc:17:51:6b:9e:33:df:ea:0e:95:cf:31:6f:37:
         7b:c3:c4:37
```

nifi-key.key

```
         # The first command shows the actual content of the encoded file,
 and the second parses it and shows the internal values

.../certs $ more nifi-key.key
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAqkVrrC+AkFbjnCpupSy84tTFDsRVUIWYj/k2pVwC145M3bpr
0pRCzLuzovAjFCmT5L+isTvNjhionsqif07Ebd/M2psYE/Rih2MULsX6KgRe1nRU
iBeKF08hlmSBMGDFPj39yDzE/V9edxV/KGjRqVgw/Qy0vwaS5uWdXnLDhzoV4/Mz
7lGmYoMasZ1uexlH93jjBl1+EFL2Xoa06oLbEojJ9TKaWhpG8ietEedf7WM0zqBE
z2kHo9ddFk9yxiCkT4SUKnDWkhwc/o6us1vEXoSw+tmufHY/A3gVihjWPIGzqyLF
l9JuN7CyJepkVVqTdskBG7S85G/kBlizUj5jOwIDAQABAoIBAAdWRnV89oVBuT0Z
dvsXGmyLzpH8U9DMcO6DRp+Jf3XaY+WKCutgCCDaVbtHrbtIr17EAzav5QOifGGb
SbVCp6Q0aJdi5360oSpEUrJRRZ5Z4dxL1vimSwUGG+RnIEn9YYJ1GWJve+2PFnr7
KieLnL03V6UPzxoMJnhcnJNdTp+dBwzSazVQwye2csSJlVMk49t2lxBwce7ohuh+
9fL7G3HU5S9d08QT1brknMHahcw1SYyJd0KSjRJCB6wAxnAZmJYJ1jQCI8YICq0j
RX2rhxEXuEMXQcaiFQXzCrmQEXreKUISDvNeu/h7YU9UvJWPZSFGnEGgnMP2XvQm
EjK3rQECgYEA5+OkpLsiLNMHGzj72PiBkq82sTLQJ2+8udYp6PheOGkhjjXoBse5
YynyHlQt6CnVpJQ33mQUkJ+3ils0SMFtmI3rz3udzleek1so2L2J3+CI4kt7fFCb
FFbVXv+dLNrm+tOw68J48asyad8kEnHYq9Us+/3MLDmFJYTthkgzCpECgYEAu/ml
lQaWaZAQcQ8UuVeasxMYoN8zMmzfrkxc8AfNwKxF9nc44ywo4nJr+u/UVRGYpRgM
rdll5vz0Iq68qk03spaW7vDJn8hJQhkReQw1it9Fp/51r9MHzGTVarORJGa2oZ0g
iNe8LNizD3bQ19hEvju9mn0x9Q62Q7dapVpffwsCgYEAtC1TPpQQ59dIjERom5vr
wffWfTTIO/w8HgFkKxrgyuAVLJSCJtKFH6H1+M7bpKrsz6ZDCs+kkwMm76ASLf3t
lD2h3mNkqHG4SzLnuBD90jB666pO1rci6FjYDap7i+DC3F4j9+vxYYXt9Aln09UV
z94hx+LaA/rlk9OHY3EyB6ECgYBA/cCtNNjeaKv2mxM8PbjD/289d85YueHgfpCH
gPs3iZiq7W+iw8ri+FKzMSaFvw66zgTcOtULtxulviqG6ym9umk29dOQRgxmKQqs
gnckq6uGuOjxwJHqrlZHjQw6vLSaThxIk+aAzu+iAh+U8TZbW4ZjmrOiGdMUuJlD
oGpyHwKBgQCRjfqQjRelYVtU7j6BD9BDbCfmipwaRNP0CuAGOVtS+UnJuaIhsXFQ
QGEBuOnfFijIvb7YcXRL4plRYPMvDqYRNObuI6A+1xNtr000nxa/HUfzKVeI9Tsn
9AKMWnXS8ZcfStsVf3oDFffXYRqCaWeuhpMmg9TwdXoAuwfpE5GCmw==
-----END RSA PRIVATE KEY-----
.../certs $ openssl rsa -in nifi-key.key -text -noout
Private-Key: (2048 bit)
modulus:
    00:aa:45:6b:ac:2f:80:90:56:e3:9c:2a:6e:a5:2c:
    bc:e2:d4:c5:0e:c4:55:50:85:98:8f:f9:36:a5:5c:
    02:d7:8e:4c:dd:ba:6b:d2:94:42:cc:bb:b3:a2:f0:
    23:14:29:93:e4:bf:a2:b1:3b:cd:8e:18:a8:9e:ca:
    a2:7f:4e:c4:6d:df:cc:da:9b:18:13:f4:62:87:63:
    14:2e:c5:fa:2a:04:5e:d6:74:54:88:17:8a:17:4f:
    21:96:64:81:30:60:c5:3e:3d:fd:c8:3c:c4:fd:5f:
    5e:77:15:7f:28:68:d1:a9:58:30:fd:0c:b4:bf:06:
    92:e6:e5:9d:5e:72:c3:87:3a:15:e3:f3:33:ee:51:
    a6:62:83:1a:b1:9d:6e:7b:19:47:f7:78:e3:06:5d:
    7e:10:52:f6:5e:86:b4:ea:82:db:12:88:c9:f5:32:
    9a:5a:1a:46:f2:27:ad:11:e7:5f:ed:63:34:ce:a0:
    44:cf:69:07:a3:d7:5d:16:4f:72:c6:20:a4:4f:84:
    94:2a:70:d6:92:1c:1c:fe:8e:ae:b3:5b:c4:5e:84:
    b0:fa:d9:ae:7c:76:3f:03:78:15:8a:18:d6:3c:81:
    b3:ab:22:c5:97:d2:6e:37:b0:b2:25:ea:64:55:5a:
    93:76:c9:01:1b:b4:bc:e4:6f:e4:06:58:b3:52:3e:
    63:3b
publicExponent: 65537 (0x10001)
privateExponent:
    07:56:46:75:7c:f6:85:41:b9:3d:19:76:fb:17:1a:
    6c:8b:ce:91:fc:53:d0:cc:70:ee:83:46:9f:89:7f:
    75:da:63:e5:8a:0a:eb:60:08:20:da:55:bb:47:ad:
    bb:48:af:5e:c4:03:36:af:e5:03:a2:7c:61:9b:49:
    b5:42:a7:a4:34:68:97:62:e7:7e:b4:a1:2a:44:52:
    b2:51:45:9e:59:e1:dc:4b:d6:f8:a6:4b:05:06:1b:
```

```
    e4:67:20:49:fd:61:82:75:19:62:6f:7b:ed:8f:16:
    7a:fb:2a:27:8b:9c:bd:37:57:a5:0f:cf:1a:0c:26:
    78:5c:9c:93:5d:4e:9f:9d:07:0c:d2:6b:35:50:c3:
    27:b6:72:c4:89:95:53:24:e3:db:76:97:10:70:71:
    ee:e8:86:e8:7e:f5:f2:fb:1b:71:d4:e5:2f:5d:d3:
    c4:13:d5:ba:e4:9c:c1:da:85:cc:35:49:8c:89:77:
    42:92:8d:12:42:07:ac:00:c6:70:19:98:96:09:d6:
    34:02:23:c6:08:0a:ad:23:45:7d:ab:87:11:17:b8:
    43:17:41:c6:a2:15:05:f3:0a:b9:90:11:7a:de:29:
    42:12:0e:f3:5e:bb:f8:7b:61:4f:54:bc:95:8f:65:
    21:46:9c:41:a0:9c:c3:f6:5e:f4:26:12:32:b7:ad:
    01
prime1:
    00:e7:e3:a4:a4:bb:22:2c:d3:07:1b:38:fb:d8:f8:
    81:92:af:36:b1:32:d0:27:6f:bc:b9:d6:29:e8:f8:
    5e:38:69:21:8e:35:e8:06:c7:b9:63:29:f2:1e:54:
    2d:e8:29:d5:a4:94:37:de:64:14:90:9f:b7:8a:5b:
    34:48:c1:6d:98:8d:eb:cf:7b:9d:ce:57:9e:93:5b:
    28:d8:bd:89:df:e0:88:e2:4b:7b:7c:50:9b:14:56:
    d5:5e:ff:9d:2c:da:e6:fa:d3:b0:eb:c2:78:f1:ab:
    32:69:df:24:12:71:d8:ab:d5:2c:fb:fd:cc:2c:39:
    85:25:84:ed:86:48:33:0a:91
prime2:
    00:bb:f9:a5:95:06:96:69:90:10:71:0f:14:b9:57:
    9a:b3:13:18:a0:df:33:32:6c:df:ae:4c:5c:f0:07:
    cd:c0:ac:45:f6:77:38:e3:2c:28:e2:72:6b:fa:ef:
    d4:55:11:98:a5:18:0c:ad:d9:65:e6:fc:f4:22:ae:
    bc:aa:4d:37:b2:96:96:ee:f0:c9:9f:c8:49:42:19:
    11:79:0c:35:8a:df:45:a7:fe:75:af:d3:07:cc:64:
    d5:6a:b3:91:24:66:b6:a1:9d:20:88:d7:bc:2c:d8:
    b3:0f:76:d0:d7:d8:44:be:3b:bd:9a:7d:31:f5:0e:
    b6:43:b7:5a:a5:5a:5f:7f:0b
exponent1:
    00:b4:2d:53:3e:94:10:e7:d7:48:8c:44:68:9b:9b:
    eb:c1:f7:d6:7d:34:c8:3b:fc:3c:1e:01:64:2b:1a:
    e0:ca:e0:15:2c:94:82:26:d2:85:1f:a1:f5:f8:ce:
    db:a4:aa:ec:cf:a6:43:0a:cf:a4:93:03:26:ef:a0:
    12:2d:fd:ed:94:3d:a1:de:63:64:a8:71:b8:4b:32:
    e7:b8:10:fd:d2:30:7a:eb:aa:4e:d6:b7:22:e8:58:
    d8:0d:aa:7b:8b:e0:c2:dc:5e:23:f7:eb:f1:61:85:
    ed:f4:09:67:d3:d5:15:cf:de:21:c7:e2:da:03:fa:
    e5:93:d3:87:63:71:32:07:a1
exponent2:
    40:fd:c0:ad:34:d8:de:68:ab:f6:9b:13:3c:3d:b8:
    c3:ff:6f:3d:77:ce:58:b9:e1:e0:7e:90:87:80:fb:
    37:89:98:aa:ed:6f:a2:c3:ca:e2:f8:52:b3:31:26:
    85:bf:0e:ba:ce:04:dc:3a:d5:0b:b7:1b:a5:be:2a:
    86:eb:29:bd:ba:69:36:f5:d3:90:46:0c:66:29:0a:
    ac:82:77:24:ab:ab:86:b8:e8:f1:c0:91:ea:ae:56:
    47:8d:0c:3a:bc:b4:9a:4e:1c:48:93:e6:80:ce:ef:
    a2:02:1f:94:f1:36:5b:5b:86:63:9a:b3:a2:19:d3:
    14:b8:99:43:a0:6a:72:1f
coefficient:
    00:91:8d:fa:90:8d:17:a5:61:5b:54:ee:3e:81:0f:
    d0:43:6c:27:e6:8a:9c:1a:44:d3:f4:0a:e0:06:39:
    5b:52:f9:49:c9:b9:a2:21:b1:71:50:40:61:01:b8:
    e9:df:16:28:c8:bd:be:d8:71:74:4b:e2:99:51:60:
    f3:2f:0e:a6:11:34:e6:ee:23:a0:3e:d7:13:6d:af:
    4d:34:9f:16:bf:1d:47:f3:29:57:88:f5:3b:27:f4:
    02:8c:5a:75:d2:f1:97:1f:4a:db:15:7f:7a:03:15:
    f7:d7:61:1a:82:69:67:ae:86:93:26:83:d4:f0:75:
    7a:00:bb:07:e9:13:91:82:9b
```

1. To convert from DER encoded public certificate (cert.der) to PEM encoded (cert.pem):
   - If the DER file contains both the public certificate and private key, remove the private key with this command:
     - perl -pe 'BEGIN{undef $/;} s|-----BEGIN PRIVATE KEY-----.*?-----END PRIVATE KEY-----|Removed private key|gs' cert.der > cert.pem
   - If the DER file only contains the public certificate, use this command:
     - openssl x509 -inform der -in cert.der -out cert.pem
2. To convert from a PKCS12 keystore (keystore.p12) containing both the public certificate and private key into PEM encoded files ($PASSWORD is the keystore password):
   - openssl pkcs12 -in keystore.p12 -out cert.der -nodes -password "pass:$PASSWORD"
   - openssl pkcs12 -in keystore.p12 -nodes -nocerts -out key.key -password "pass:$PASSWORD"
   - Follow the steps above to convert cert.der to cert.pem
3. To convert from a Java Keystore (keystore.jks) containing private key into PEM encoded files ($P12_PASSWORD is the PKCS12 keystore password, $JKS_PASSWORD is the Java keystore password you want to set, and $ALIAS can be any value - the NiFi default is nifi-key):
   - keytool -importkeystore -srckeystore keystore.jks -destkeystore keystore.p12 -srcstoretype JKS -deststoretype PKCS12 -destkeypass "$P12_PASSWORD" -deststorepass "$P12_PASSWORD" -srcstorepass "$JKS_PASSWORD" -srcalias "$ALIAS" -destalias "$ALIAS"
   - Follow the steps above to convert from keystore.p12 to cert.pem and key.key
4. To convert from PKCS #8 PEM format to PKCS #1 PEM format:
   - If the private key is provided in PKCS #8 format (the file begins with -----BEGIN PRIVATE KEY----- rather than -----BEGIN RSA PRIVATE KEY-----), the following command will convert it to PKCS #1 format, move the original to nifi-key-pkcs8.key, and rename the PKCS #1 version as nifi-key.key:
     - openssl rsa -in nifi-key.key -out nifi-key-pkcs1.key && mv nifi-key.key nifi-key-pkcs8.key && mv nifi-key-pkcs1.key nifi-key.key

Signing with Externally-signed CA Certificates

To sign generated certificates with a certificate authority (CA) generated outside of the TLS Toolkit, ensure the necessary files are in the right format and location (see above). For example, an organization Large Organization has an internal CA (CN=ca.large.org, OU=Certificate Authority). This root CA is offline and only used to sign other internal CAs. The Large IT team generates an intermediate CA (CN=nifi_ca.large.org, OU=NiFi, OU=Certificate Authority) to be used to sign all NiFi node certificates (CN=node1.nifi.large.org, OU=NiFi, CN=node2.nifi.large.org, OU=NiFi, etc.).

To use the toolkit to generate these certificates and sign them using the intermediate CA, ensure that the following files are present:

- nifi-cert.pem - the public certificate of the intermediate CA in PEM format
- nifi-key.key - the Base64-encoded private key of the intermediate CA in PKCS #1 PEM format

If the intermediate CA was the root CA, it would be self-signed - the signature over the certificate would be issued from the same key. In that case (the same as a toolkit-generated CA), no additional arguments are necessary. However, because the intermediate CA is signed by the root CA, the public certificate of the root CA needs to be provided as well to validate the signature. The --additionalCACertificate parameter is used to specify the path to the signing public certificate. The value should be the absolute path to the root CA public certificate.

Example:

```
            # Generate cert signed by intermediate CA (which is signed by
 root CA) -- WILL FAIL

$ ./bin/tls-toolkit.sh standalone -n 'node1.nifi.apache.org' \
-P passwordpassword \
-S passwordpassword \
```

```
-o /opt/certs/externalCA \
-O

2018/08/02 18:48:11 INFO [main]
 org.apache.nifi.toolkit.tls.standalone.TlsToolkitStandaloneCommandLine: No
 nifiPropertiesFile specified, using embedded one.
2018/08/02 18:48:12 INFO [main]
 org.apache.nifi.toolkit.tls.standalone.TlsToolkitStandalone: Running
 standalone certificate generation with output directory /opt/certs/
externalCA
2018/08/02 18:48:12 INFO [main] org.apache.nifi.toolkit.tls.util.TlsHelper:
 Verifying the certificate signature for CN=nifi_ca.large.org,
 OU=Certificate Authority
2018/08/02 18:48:12 INFO [main] org.apache.nifi.toolkit.tls.util.TlsHelper:
 Attempting to verify certificate CN=nifi_ca.large.org, OU=NiFi,
 OU=Certificate Authority signature with CN=nifi_ca.large.org, OU=NiFi,
 OU=Certificate Authority
2018/08/02 18:48:12 WARN [main] org.apache.nifi.toolkit.tls.util.TlsHelper:
 Certificate CN=nifi_ca.large.org, OU=NiFi, OU=Certificate Authority
 not signed by CN=nifi_ca.large.org, OU=NiFi, OU=Certificate Authority
 [certificate does not verify with supplied key]
Error generating TLS configuration. (The signing certificate was not signed
 by any known certificates)

# Provide additional CA certificate path for signature verification of
 intermediate CA

$ ./bin/tls-toolkit.sh standalone -n 'node1.nifi.apache.org' \
-P passwordpassword \
-S passwordpassword \
-o /opt/certs/externalCA \
--additionalCACertificate /opt/certs/externalCA/root.pem \
-O

2018/08/02 18:48:44 INFO [main]
 org.apache.nifi.toolkit.tls.standalone.TlsToolkitStandaloneCommandLine: No
 nifiPropertiesFile specified, using embedded one.
2018/08/02 18:48:44 INFO [main]
 org.apache.nifi.toolkit.tls.standalone.TlsToolkitStandalone: Running
 standalone certificate generation with output directory /opt/certs/
externalCA
2018/08/02 18:48:44 INFO [main] org.apache.nifi.toolkit.tls.util.TlsHelper:
 Verifying the certificate signature for CN=nifi_ca.large.org, OU=NiFi,
 OU=Certificate Authority
2018/08/02 18:48:44 INFO [main] org.apache.nifi.toolkit.tls.util.TlsHelper:
 Attempting to verify certificate CN=nifi_ca.large.org, OU=NiFi,
 OU=Certificate Authority signature with CN=ca.large.org, OU=Certificate
 Authority
2018/08/02 18:48:44 INFO [main] org.apache.nifi.toolkit.tls.util.TlsHelper:
 Certificate was signed by CN=ca.large.org, OU=Certificate Authority
2018/08/02 18:48:44 INFO [main]
 org.apache.nifi.toolkit.tls.standalone.TlsToolkitStandalone: Using existing
 CA certificate /opt/certs/externalCA/nifi-cert.pem and key /opt/certs/
externalCA/nifi-key.key
2018/08/02 18:48:44 INFO [main]
 org.apache.nifi.toolkit.tls.standalone.TlsToolkitStandalone: Writing new
 ssl configuration to /opt/certs/externalCA/node1.nifi.apache.org
2018/08/02 18:48:44 INFO [main]
 org.apache.nifi.toolkit.tls.standalone.TlsToolkitStandalone: Successfully
 generated TLS configuration for node1.nifi.apache.org 1 in /opt/certs/
externalCA/node1.nifi.apache.org
2018/08/02 18:48:44 INFO [main]
 org.apache.nifi.toolkit.tls.standalone.TlsToolkitStandalone: No
 clientCertDn specified, not generating any client certificates.
```

```
2018/08/02 18:48:44 INFO [main]
 org.apache.nifi.toolkit.tls.standalone.TlsToolkitStandalone: tls-toolkit
 standalone completed successfully
```

Encrypted Passwords in Configuration Files

In order to facilitate the secure setup of NiFi, you can use the encrypt-config command line utility to encrypt raw configuration values that NiFi decrypts in memory on startup. This extensible protection scheme transparently allows NiFi to use raw values in operation, while protecting them at rest. In the future, hardware security modules (HSM) and external secure storage mechanisms will be integrated, but for now, an AES encryption provider is the default implementation.

This is a change in behavior; prior to 1.0, all configuration values were stored in plaintext on the file system. POSIX file permissions were recommended to limit unauthorized access to these files

If no administrator action is taken, the configuration values remain unencrypted.

Encrypt-Config Tool

The encrypt-config command line tool (invoked as ./bin/encrypt-config.sh or bin\encrypt-config.bat) reads from a nifi.properties file with plaintext sensitive configuration values, prompts for a master password or raw hexadecimal key, and encrypts each value. It replaces the plain values with the protected value in the same file, or writes to a new nifi.properties file if specified.

The default encryption algorithm utilized is AES/GCM 128/256-bit. 128-bit is used if the JCE Unlimited Strength Cryptographic Jurisdiction Policy files are not installed, and 256-bit is used if they are installed.

You can use the following command line options with the encrypt-config tool:

- -h,--help Prints this usage message
- -v,--verbose Sets verbose mode (default false)
- -n,--niFiProperties <arg> The nifi.properties file containing unprotected config values (will be overwritten)
- -l,--loginIdentityProviders <arg> The login-identity-providers.xml file containing unprotected config values (will be overwritten)
- -a,--authorizers <arg> The authorizers.xml file containing unprotected config values (will be overwritten)
- -f,--flowXml <arg> The flow.xml.gz file currently protected with old password (will be overwritten)
- -b,--bootstrapConf <arg> The bootstrap.conf file to persist master key
- -o,--outputNiFiProperties <arg> The destination nifi.properties file containing protected config values (will not modify input nifi.properties)
- -i,--outputLoginIdentityProviders <arg> The destination login-identity-providers.xml file containing protected config values (will not modify input login-identity-providers.xml)
- -u,--outputAuthorizers <arg> The destination authorizers.xml file containing protected config values (will not modify input authorizers.xml)
- -g,--outputFlowXml <arg> The destination flow.xml.gz file containing protected config values (will not modify input flow.xml.gz)
- -k,--key <arg> The raw hexadecimal key to use to encrypt the sensitive properties
- -e,--oldKey <arg> The old raw hexadecimal key to use during key migration
- -p,--password <arg> The password from which to derive the key to use to encrypt the sensitive properties
- -w,--oldPassword <arg> The old password from which to derive the key during migration
- -r,--useRawKey If provided, the secure console will prompt for the raw key value in hexadecimal form
- -m,--migrate If provided, the nifi.properties and/or login-identity-providers.xml sensitive properties will be re-encrypted with a new key

- -x,--encryptFlowXmlOnly If provided, the properties in flow.xml.gz will be re-encrypted with a new key but the nifi.properties and/or login-identity-providers.xml files will not be modified
- -s,--propsKey <arg> The password or key to use to encrypt the sensitive processor properties in flow.xml.gz
- -A,--newFlowAlgorithm <arg> The algorithm to use to encrypt the sensitive processor properties in flow.xml.gz
- -P,--newFlowProvider <arg> The security provider to use to encrypt the sensitive processor properties in flow.xml.gz

As an example of how the tool works, assume that you have installed the tool on a machine supporting 256-bit encryption and with the following existing values in the nifi.properties file:

```
# security properties #
nifi.sensitive.props.key=thisIsABadSensitiveKeyPassword
nifi.sensitive.props.algorithm=PBEWITHMD5AND256BITAES-CBC-OPENSSL
nifi.sensitive.props.provider=BC
nifi.sensitive.props.additional.keys=

nifi.security.keystore=/path/to/keystore.jks
nifi.security.keystoreType=JKS
nifi.security.keystorePasswd=thisIsABadKeystorePassword
nifi.security.keyPasswd=thisIsABadKeyPassword
nifi.security.truststore=
nifi.security.truststoreType=
nifi.security.truststorePasswd=
```

Enter the following arguments when using the tool:

```
encrypt-config.sh
-b bootstrap.conf
-k 0123456789ABCDEFFEDCBA98765432100123456789ABCDEFFEDCBA9876543210
-n nifi.properties
```

As a result, the nifi.properties file is overwritten with protected properties and sibling encryption identifiers (aes/gcm/256, the currently supported algorithm):

```
# security properties #
nifi.sensitive.props.key=n2z+tTTbHuZ4V4V2||uWhdasyDXD4ZG2lMAes/
vqh6u4vaz4xgL4aEbF4Y/dXevqk3ulRcOwf1vc4RDQ==
nifi.sensitive.props.key.protected=aes/gcm/256
nifi.sensitive.props.algorithm=PBEWITHMD5AND256BITAES-CBC-OPENSSL
nifi.sensitive.props.provider=BC
nifi.sensitive.props.additional.keys=

nifi.security.keystore=/path/to/keystore.jks
nifi.security.keystoreType=JKS
nifi.security.keystorePasswd=oBjT92hIGRElIGOh||MZ6uYuWNBrOA6usq/
Jt3DaD2e4otNirZDytac/w/KFe0HOkrJR03vcbo
nifi.security.keystorePasswd.protected=aes/gcm/256
nifi.security.keyPasswd=ac/BaE35SL/esLiJ||
+ULRvRLYdIDA2VqpE0eQXDEMjaLBMG2kbKOdOwBk/hGebDKlVg==
nifi.security.keyPasswd.protected=aes/gcm/256
nifi.security.truststore=
nifi.security.truststoreType=
nifi.security.truststorePasswd=
```

Additionally, the bootstrap.conf file is updated with the encryption key as follows:

```
# Master key in hexadecimal format for encrypted sensitive configuration
 values
nifi.bootstrap.sensitive.key=0123456789ABCDEFFEDCBA98765432100123456789ABCDEFFEDCBA98765
```

Sensitive configuration values are encrypted by the tool by default, however you can encrypt any additional properties, if desired. To encrypt additional properties, specify them as comma-separated values in the nifi.sensitive.props.additional.keys property.

If the nifi.properties file already has valid protected values, those property values are not modified by the tool.

When applied to login-identity-providers.xml and authorizers.xml, the property elements are updated with an encryption attribute:

Example of protected login-identity-providers.xml:

```
   <!-- LDAP Provider -->
   <provider>
       <identifier>ldap-provider</identifier>
       <class>org.apache.nifi.ldap.LdapProvider</class>
       <property name="Authentication Strategy">START_TLS</property>
       <property name="Manager DN">someuser</property>
       <property name="Manager Password" encryption="aes/
gcm/128">q4r7WIgN0MaxdAKM||SGgdCTPGSFEcuH4RraMYEdeyVbOx93abdWTVSWvh1w+klA</
property>
       <property name="TLS - Keystore"></property>
       <property name="TLS - Keystore Password" encryption="aes/
gcm/128">Uah59TWX+Ru5GY5p||B44RT/LJtC08QWA5ehQf01JxIpf0qSJUzug25UwkF5a50g</
property>
       <property name="TLS - Keystore Type"></property>
       ...
   </provider>
```

Example of protected authorizers.xml:

```
   <!-- LDAP User Group Provider -->
   <userGroupProvider>
       <identifier>ldap-user-group-provider</identifier>
       <class>org.apache.nifi.ldap.tenants.LdapUserGroupProvider</class>
       <property name="Authentication Strategy">START_TLS</property>
       <property name="Manager DN">someuser</property>
       <property name="Manager Password" encryption="aes/
gcm/128">q4r7WIgN0MaxdAKM||SGgdCTPGSFEcuH4RraMYEdeyVbOx93abdWTVSWvh1w+klA</
property>
       <property name="TLS - Keystore"></property>
       <property name="TLS - Keystore Password" encryption="aes/
gcm/128">Uah59TWX+Ru5GY5p||B44RT/LJtC08QWA5ehQf01JxIpf0qSJUzug25UwkF5a50g</
property>
       <property name="TLS - Keystore Type"></property>
       ...
   </userGroupProvider>
---
```

Sensitive Property Key Migration

In order to change the key used to encrypt the sensitive values, indicate migration mode using the -m or --migrate flag, provide the new key or password using the -k or -p flags as usual, and provide the existing key or password using -e or -w respectively. This will allow the toolkit to decrypt the existing values and re-encrypt them, and update bootstrap.conf with the new key. Only one of the key or password needs to be specified for each phase (old vs. new), and any combination is sufficient:

- old key # new key
- old key # new password
- old password # new key
- old password # new password

Existing Flow Migration

This tool can also be used to change the value of nifi.sensitive.props.key for an existing flow. The tool will read the existing flow.xml.gz and decrypt any sensitive component properties using the original key, then re-encrypt the sensitive properties with the new key, and write out a new version of the flow.xml.gz, or overwrite the existing one.

The current sensitive properties key is not provided as a command-line argument, as it is read directly from nifi.properties. As this file is a required parameter, the -x/--encryptFlowXmlOnly flags tell the tool not to attempt to encrypt the properties in nifi.properties, but rather to only update the nifi.sensitive.props.key value with the new key. The exception to this is if the nifi.properties is already encrypted, the new sensitive property key will also be encrypted before being written to nifi.properties.

The following command would migrate the sensitive properties key in place, meaning it would overwrite the existing flow.xml.gz and nifi.properties:

```
./encrypt-config.sh -f /path/to/flow.xml.gz -n ./path/to/nifi.properties -s
 newpassword -x
```

The following command would migrate the sensitive properties key and write out a separate flow.xml.gz and nifi.properties:

```
./encrypt-config.sh -f ./path/to/src/flow.xml.gz -g /path/to/dest/
flow.xml.gz -n /path/to/src/nifi.properties -o /path/to/dest/nifi.properties
 -s newpassword -x
```

Password Key Derivation

Instead of providing a 32 or 64 character raw hexadecimal key, you can provide a password from which the key will be derived. As of 1.0.0, the password must be at least 12 characters, and the key will be derived using SCrypt with the parameters:

- pw - the password bytes in UTF-8
- salt - the fixed salt value (NIFI_SCRYPT_SALT) bytes in UTF-8
- N - 216
- r - 8
- p - 1
- dkLen - determined by the JCE policies available

As of August 2016, these values are determined to be strong for this threat model but may change in future versions.

Note: While fixed salts are counter to best practices, a static salt is necessary for deterministic key derivation without additional storage of the salt value.

Secure Prompt

If you prefer not to provide the password or raw key in the command-line invocation of the tool, leaving these arguments absent will prompt a secure console read of the password (by default) or raw key (if the -r flag is provided at invocation).

Administrative Tools

The admin toolkit contains command line utilities for administrators to support NiFi maintenance in standalone and clustered environments. These utilities include:

- Notify - The notification tool allows administrators to send bulletins to the NiFi UI using the command line.
- Node Manager - The node manager tool allows administrators to perform a status check on a node as well as to connect, disconnect, or remove nodes that are part of a cluster.
- File Manager - The file manager tool allows administrators to backup, install or restore a NiFi installation from backup.

The admin toolkit is bundled with the nifi-toolkit and can be executed with scripts found in the bin folder.

Prerequisites for Running Admin Toolkit in a Secure Environment

For secured nodes and clusters, two policies should be configured in advance:

- Access the controller - A user that will have access to these utilities should be authorized in NiFi by creating an "access the controller" policy (/controller) with both view and modify rights.
- Proxy user request - If not previously set node's identity (the DN value of the node's certificate) should be authorized to proxy requests on behalf of a user

When executing either the notify or node manager tools in a secured environment the proxyDN flag option should be used in order to properly identify the user that was authorized to execute these commands. In non-secure environments, or if running the status operation on the Node Manager tool, the flag is ignored.

Notify

Notify allows administrators to send messages as bulletins to NiFi. Notify is supported on NiFi version 1.2.0 and higher. The notification tool is also available in a notify.bat file for use on Windows machines.

To send notifications:

```
notify.sh -d {$NIFI_HOME} -b {nifi bootstrap file path} -m {message} [-l
 {level}] [-v]
```

To show help:

```
notify.sh -h
```

The following are available options:

- -b,--bootstrapConf <arg> Existing Bootstrap Configuration file (required)
- -d,--nifiInstallDir <arg> NiFi Root Folder (required)
- -h,--help Help Text (optional)
- -l,--level <arg> Status level of bulletin - INFO, WARN, ERROR
- -m,--message <arg> Bulletin message (required)
- -p,--proxyDN <arg> Proxy or User DN (required for secured nodes)
- -v,--verbose Verbose messaging (optional)

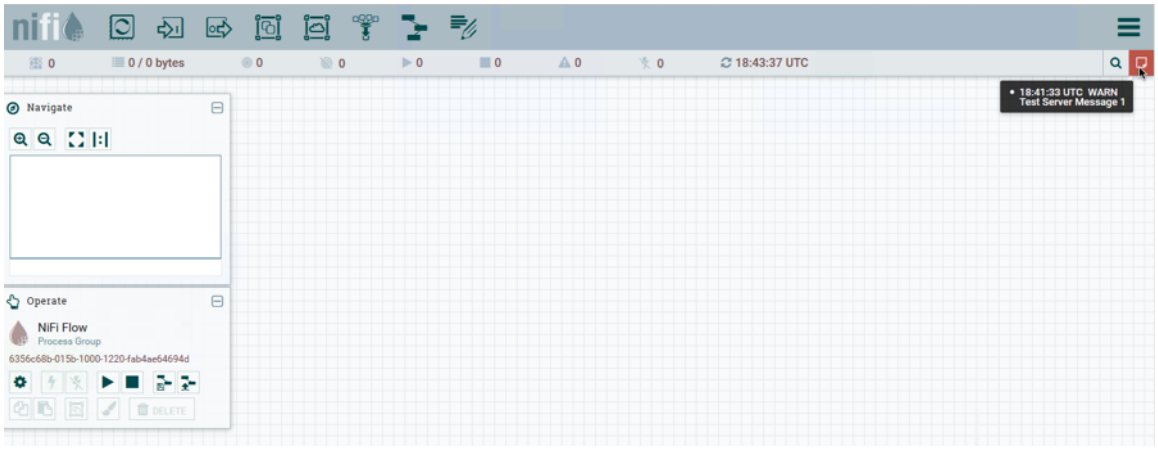
Example usage on Linux:

```
./notify.sh -d /usr/nifi/nifi_current -b /usr/nifi/nifi_current/conf/
bootstrap.conf -m "Test Message Server 1" -l "WARN" -p "ydavis@nifi" -v
```

Example usage on Windows:

```
notify.bat -v -d "C:\\Program Files\\nifi\\nifi-1.2.0-SNAPSHOT" -b "C:\
\Program Files\\nifi\\nifi-1.2.0-SNAPSHOT\\conf\\bootstrap.conf" -m "Test
 Message Server 1" -v
```

Executing the above command line should result in a bulletin appearing in NiFi:



Node Manager

Node manager supports connecting, disconnecting and removing a node when in a cluster (an error message displays if the node is not part of a cluster) as well as obtaining the status of a node. When nodes are disconnected from a cluster and need to be connected or removed, a list of urls of connected nodes should be provided to send the required command to the active cluster. Node Manager supports NiFi version 1.0.0 and higher. Node Manager is also available in node-manager.bat file for use on Windows machines.

To connect, disconnect, or remove a node from a cluster:

```
node-manager.sh -d {$NIFI_HOME} -b { nifi bootstrap file path}
-o {remove|disconnect|connect|status} [-u {url list}] [-p {proxy name}] [-v]
```

To show help:

```
node-manager.sh -h
```

The following are available options:

- -b,--bootstrapConf <arg> Existing Bootstrap Configuration file (required)
- -d,--nifiInstallDir <arg> NiFi Root Folder (required)
- -h,--help Help Text (optional)
- -o, --operation <arg> Operations supported: status, connect (cluster), disconnect (cluster), remove (cluster)
- -p,--proxyDN <arg> Proxy or User DN (required for secured nodes doing connect, disconnect and remove operations)
- -u,--clusterUrls <arg> Comma delimited list of active urls for cluster (optional). Not required for disconnecting a node yet will be needed when connecting or removing from a cluster
- -v,--verbose Verbose messaging (optional)

Example usage on Linux:

```
# disconnect without cluster url list
./node-manager.sh
-d /usr/nifi/nifi_current
```

```
-b /usr/nifi/nifi_current/conf/bootstrap.conf
-o disconnect
-p ydavis@nifi
-v
```

```
#with url list
./node-manager.sh
-d /usr/nifi/nifi_current
-b /usr/nifi/nifi_current/conf/bootstrap.conf
-o connect
-u 'http://nifi-server-1:8080,http://nifi-server-2:8080'
-v
```

Example usage on Windows:

```
node-manager.bat
-d "C:\\Program Files\\nifi\\nifi-1.2.0-SNAPSHOT"
-b "C:\\Program Files\\nifi\\nifi-1.2.0-SNAPSHOT\\conf\\bootstrap.conf"
-o disconnect
-v
```

Expected behavior

Status:

To obtain information on UI availability of a node, the status operation can be used to determine if the node is running. If the -u (clusterUrls) option is not provided the current node url is checked otherwise the urls provided will be checked.

Disconnect:

When a node is disconnected from the cluster, the node itself should appear as disconnected and the cluster should have a bulletin indicating the disconnect request was received. The cluster should also show n-1/n nodes available in the cluster. For example, if 1 node is disconnected from a 3-node cluster, then "2 of 3" nodes should show on the remaining nodes in the cluster. Changes to the flow should not be allowed on the cluster with a disconnected node.

Connect:

When the connect command is executed to reconnect a node to a cluster, upon completion the node itself should show that it has rejoined the cluster by showing n/n nodes. Previously it would have shown Disconnected. Other nodes in the cluster should receive a bulletin of the connect request and also show n/n nodes allowing for changes to be allowed to the flow.

Remove:

When the remove command is executed the node should show as disconnected from a cluster. The nodes remaining in the cluster should show n-1/n-1 nodes. For example, if 1 node is removed from a 3-node cluster, then the remaining 2 nodes should show "2 of 2" nodes. The cluster should allow a flow to be adjusted. The removed node can rejoin the cluster if restarted and the flow for the cluster has not changed. If the flow was changed, the flow template of the removed node should be deleted before restarting the node to allow it to obtain the cluster flow (otherwise an uninheritable flow file exception may occur).

File Manager

The File Manager utility allows system administrators to take a backup of an existing NiFi installation, install a new version of NiFi in a designated location (while migrating any previous configuration settings) or restore an installation from a previous backup. File Manager supports NiFi version 1.0.0 and higher and is available in file-manager.bat file for use on Windows machines.

To show help:

```
file-manager.sh -h
```

The following are available options:

- -b,--backupDir <arg> Backup NiFi Directory (used with backup or restore operation)
- -c,--nifiCurrentDir <arg> Current NiFi Installation Directory (used optionally with install or restore operation)
- -d,--nifiInstallDir <arg> NiFi Installation Directory (used with install or restore operation)
- -h,--help Print help info (optional)
- -i,--installFile <arg> NiFi Install File (used with install operation)
- -m,--moveRepositories Allow repositories to be moved to new/restored nifi directory from existing installation, if available (used optionally with install or restore operation)
- -o,--operation <arg> File operation (install | backup | restore)
- -r,--nifiRollbackDir <arg> NiFi Installation Directory (used with install or restore operation)
- -t,--bootstrapConf <arg> Current NiFi Bootstrap Configuration File (used optionally)
- -v,--verbose Verbose messaging (optional)
- -x,--overwriteConfigs Overwrite existing configuration directory with upgrade changes (used optionally with install or restore operation)

Example usage on Linux:

```
# backup NiFi installation
# option -t may be provided to ensure backup of external boostrap.conf file
./file-manager.sh
-o backup
-b /tmp/nifi_bak
-c /usr/nifi_old
-v
```

```
# install NiFi using compressed tar file into /usr/nifi directory (should
 install as /usr/nifi/nifi-1.3.0).
# migrate existing configurations with location determined by external
 bootstrap.conf and move over repositories from nifi_old
# options -t and -c should both be provided if migration of configurations,
 state and repositories are required
./file-manager.sh
-o install
-i nifi-1.3.0.tar.gz
-d /usr/nifi
-c /usr/nifi/nifi_old
-t /usr/nifi/old_conf/bootstrap.conf
-v
-m
```

```
# restore NiFi installation from backup directory and move back repositories
# option -t may be provided to ensure bootstrap.conf is restored to the file
 path provided, otherwise it is placed in the
# default directory under the rollback path (e.g. /usr/nifi_old/conf)
./file-manager.sh
-o restore
-b /tmp/nifi_bak
-r /usr/nifi_old
-c /usr/nifi
-m
-v
```

Expected Behavior

Backup:

During the backup operation a backup directory is created in a designated location for an existing NiFi installation. Backups will capture all critical files (including any internal or external configurations, libraries, scripts and documents) however it excludes backing up repositories and logs due to potential size. If configuration/library files are external from the existing installation folder the backup operation will capture those as well.

Install:

During the install operation File Manager will perform installation using the designated NiFi binary file (either tar.gz or zip file) to create a new installation or migrate an existing nifi installation to a new one. Installation can optionally move repositories (if located within the configuration folder of the current installation) to the new installation as well as migrate configuration files to the newer installation.

Restore:

The restore operation allows an existing installation to revert back to a previous installation. Using an existing backup directory (created from the backup operation) the FileManager utility will restore libraries, scripts and documents as well as revert to previous configurations.

Note: If repositories were changed due to the installation of a newer version of NiFi these may no longer be compatible during restore. In that scenario exclude the -m option to ensure new repositories will be created or, if repositories live outside of the NiFi directory, remove them so they can be recreated on startup after restore.

ZooKeeper Migrator

You can use the NiFi ZooKeeper Migrator to perform the following tasks:

- Moving ZooKeeper information from one ZooKeeper cluster to another
- Migrating ZooKeeper node ownership

For example, you may want to use the ZooKeeper Migrator when you are:

- Upgrading from NiFi 0.x to NiFi 1.x in which embedded ZooKeepers are used
- Migrating from an embedded ZooKeeper in NiFi 0.x or 1.x to an external ZooKeeper
- Upgrading from NiFi 0.x with an external ZooKeeper to NiFi 1.x with the same external ZooKeeper
- Migrating from an external ZooKeeper to an embedded ZooKeeper in NiFi 1.x

zk-migrator.sh Command Line Parameters

You can use the following command line options with the ZooKeeper Migrator:

- -a,--auth <username:password> Allows the specification of a username and password for authentication with ZooKeeper. This option is mutually exclusive with the -k,--krb-conf option.
- -f,--file <filename> The file used for ZooKeeper data serialized as JSON. When used with the -r,--receive option, data read from ZooKeeper will be stored in the given filename. When used with the -s,--send option, the data in the file will be sent to ZooKeeper.
- -h,--help Prints help, displays available parameters with descriptions
- --ignore-source Allows the ZooKeeper Migrator to write to the ZooKeeper and path from which the data was obtained.
- -k,--krb-conf <jaas-filename> Allows the specification of a JAAS configuration file to allow authentication with a ZooKeeper configured to use Kerberos. This option is mutually exclusive with the -a,--auth option.

- -r,--receive Receives data from ZooKeeper and writes to the given filename (if the -f,--file option is provided) or standard output. The data received will contain the full path to each node read from ZooKeeper. This option is mutually exclusive with the -s,--send option.
- -s,--send Sends data to ZooKeeper that is read from the given filename (if the -f,--file option is provided) or standard input. The paths for each node in the data being sent to ZooKeeper are absolute paths, and will be stored in ZooKeeper under the path portion of the -z,--zookeeper argument. Typically, the path portion of the argument can be omitted, which will store the nodes at their absolute paths. This option is mutually exclusive with the -r,--receive option.
- --use-existing-acl Allows the Zookeeper Migrator to write ACL values retrieved from the source Zookeeper server to destination server. Default action will apply Open rights for unsecured destinations or Creator Only rights for secured destinations.
- -z,--zookeeper <zookeeper-endpoint> The ZooKeeper server(s) to use, specified by a connect string, comprised of one or more comma-separated host:port pairs followed by a path, in the format of host:port[,host2:port… ,hostn:port]/znode/path.

Migrating Between Source and Destination ZooKeepers

Before you begin, confirm that:

- You have installed the destination ZooKeeper cluster.
- You have installed and configured a NiFi cluster to use the destination ZooKeeper cluster.
- If you are migrating ZooKeepers due to upgrading NiFi from 0.x to 1.x,, you have already followed appropriate NiFi upgrade steps.
- You have configured Kerberos as needed.
- You have not started processing any dataflow (to avoid duplicate data processing).
- If one of the ZooKeeper clusters you are using is configured with Kerberos, you are running the ZooKeeper Migrator from a host that has access to NiFi's ZooKeeper client jaas configuration file.

ZooKeeper Migration Steps

1. Collect the following information:

Required Information	Description
Source ZooKeeper hostname (sourceHostname)	The hostname must be one of the hosts running in the ZooKeeper ensemble, which can be found in <NiFi installation dir>/conf/ zookeeper.properties. Any of the hostnames declared in the server.N properties can be used.
Destination ZooKeeper hostname (destinationHostname)	The hostname must be one of the hosts running in the ZooKeeper ensemble, which can be found in <NiFi installation dir>/conf/ zookeeper.properties. Any of the hostnames declared in the server.N properties can be used.
Source ZooKeeper port (sourceClientPort)	This can be found in <NiFi installation dir>/conf/ zookeeper.properties. The port is specified in the clientPort property.
Destination ZooKeeper port (destinationClientPort)	This can be found in <NiFi installation dir>/conf/ zookeeper.properties. The port is specified in the clientPort property.
Export data path	Determine the path that will store a json file containing the export of data from ZooKeeper. It must be readable and writable by the user running the zk-migrator tool.

Source ZooKeeper Authentication Information	This information is in <NiFi installation dir>/conf/state-management.xml. For NiFi 0.x, if Creator Only is specified in state-management.xml, you need to supply authentication information using the -a,--auth argument with the values from the Username and Password properties in state-management.xml. For NiFi 1.x, supply authentication information using the -k,--krb-conf argument. If the state-management.xml specifies Open, no authentication is required.
Destination ZooKeeper Authentication Information	This information is in <NiFi installation dir>/conf/state-management.xml. For NiFi 0.x, if Creator Only is specified in state-management.xml, you need to supply authentication information using the -a,--auth argument with the values from the Username and Password properties in state-management.xml. For NiFi 1.x, supply authentication information using the -k,--krb-conf argument. If the state-management.xml specifies Open, no authentication is required.
Root path to which NiFi writes data in Source ZooKeeper (sourceRootPath)	This information can be found in <NiFi installation dir>/conf/state-management.xml under the Root Node property in the cluster-provider element. (default: /nifi)
Root path to which NiFi writes data in Destination ZooKeeper (destinationRootPath)	This information can be found in <NiFi installation dir>/conf/state-management.xml under the Root Node property in the cluster-provider element.

2. Stop all processors in the NiFi flow. If you are migrating between two NiFi installations, the flows on both must be stopped.
3. Export the NiFi component data from the source ZooKeeper. The following command reads from the specified ZooKeeper running on the given hostname:port, using the provided path to the data, and authenticates with ZooKeeper using the given username and password. The data read from ZooKeeper is written to the file provided.
   - For NiFi 0.x
     - For an open ZooKeeper:
       - zk-migrator.sh -r -z sourceHostname:sourceClientPort/sourceRootPath/components -f /path/to/export/zk-source-data.json
     - For a ZooKeeper using username:password for authentication:
       - zk-migrator.sh -r -z sourceHostname:sourceClientPort/sourceRootPath/components -a <username:password> -f /path/to/export/zk-source-data.json
   - For NiFi 1.x
     - For an open ZooKeeper:
       - zk-migrator.sh -r -z sourceHostname:sourceClientPort/sourceRootPath/components -f /path/to/export/zk-source-data.json
     - For a ZooKeeper using Kerberos for authentication:
       - zk-migrator.sh -r -z sourceHostname:sourceClientPort/sourceRootPath/components -k /path/to/jaasconfig/jaas-config.conf -f /path/to/export/zk-source-data.json
4. (Optional) If you have used the new NiFi installation to do any processing, you can also export its ZooKeeper data as a backup prior to performing the migration.
   - For an open ZooKeeper:
     - zk-migrator.sh -r -z destinationHostname:destinationClientPort/destinationRootPath/components -f /path/to/export/zk-destination-backup-data.json
   - For a ZooKeeper using Kerberos for authentication:
     - zk-migrator.sh -r -z destinationHostname:destinationClientPort/destinationRootPath/components -k /path/to/jaasconfig/jaas-config.conf -f /path/to/export/zk-destination-backup-data.json

5. Migrate the ZooKeeper data to the destination ZooKeeper. If the source and destination ZooKeepers are the same, the --ignore-source option can be added to the following examples.
   - For an open ZooKeeper:
     - zk-migrator.sh -s -z destinationHostname:destinationClientPort/destinationRootPath/components -f /path/to/export/zk-source-data.json
   - For a ZooKeeper using Kerberos for authentication:
     - zk-migrator.sh -s -z destinationHostname:destinationClientPort/destinationRootPath/components -k /path/to/jaasconfig/jaas-config.conf -f /path/to/export/zk-source-data.json
6. Once the migration has completed successfully, start the processors in the NiFi flow. Processing should continue from the point at which it was stopped when the NiFi flow was stopped.