How to use Diffie Hellman Key Exchange with SNMP4J?

As prerequisite please read RFC 2786 which proves necessary information about the exchange workflow and corresponding terms. The RFC is also included in the `spec` directory of the SNMP4J distribution ZIP file.

The following code snippets provide some sample code taken from the SNMP4J-CLT tool illustrating how DH key exchange can be implemented using SNMP4J:

DH Kickstart Init

```
private int usmDHKickstartInit() {
    List<Object> userNames = settings.get("user");
    String privateKeysFilename = (String) ArgumentParser.getValue(settings, "privateKeysFile", 0);
    String dhParametersValArg = (String) ArgumentParser.getValue(settings, "dhp", 0);
    String dhPropertiesPrefix = (String) ArgumentParser.getValue(settings, "dhx", 0);
    String authProtocol = (String) ArgumentParser.getValue(settings, SnmpConfigurator.O_AUTH_PROTOCOL, 0);
    String privProtocol = (String) ArgumentParser.getValue(settings, SnmpConfigurator.O_PRIV_PROTOCOL, 0);
    OID authOID = SnmpConfigurator.getAuthProtocolOid(authProtocol);
    OID privOID = SnmpConfigurator.getPrivProtocolOid(privProtocol);
    try {
        if (privateKeysFilename == null) {
            System.err.println("Mandatory private keys file parameter not provided.");
            return 1;
        }
        else {
            File privateKeysFile = new File(privateKeysFilename);
            if (privateKeysFile.exists()) {
                System.err.println("Private keys file '" + privateKeysFile + "' must not exist");
                return 2;
            }
            else {
                Properties privateKeysProps = new Properties();
                Properties publicKeysProps = new Properties();
                OctetString dhParametersVal = OctetString.fromHexString(dhParametersValArg);
                DHParameters dhParameters = DHParameters.getDHParametersFromBER(dhParametersVal);
                FileOutputStream privateKeysOS = new FileOutputStream(privateKeysFile);
                for (Object userName : userNames) {
                    KeyPair keyPair = DHOperations.generatePublicKey(dhParameters);
                    OctetString z = DHOperations.derivePublicKey(keyPair);
                    privateKeysProps.put(dhPropertiesPrefix+DHOperations.DH_PRIVATE_KEY_PROPERTY +userName,
                            DHOperations.derivePrivateKey(keyPair).toString(16));
                    privateKeysProps.put(dhPropertiesPrefix+DHOperations.DH_PUBLIC_KEY_PROPERTY +userName,
                            z.toString(16));
                    publicKeysProps.put(dhPropertiesPrefix+DHOperations.DH_PUBLIC_KEY_PROPERTY +userName,
                            z.toString(16));
                    if (authOID !=  null) {
                        privateKeysProps.put(dhPropertiesPrefix+DHOperations.DH_AUTH_PROTOCOL_PROPERTY
+userName,
                                authOID.toDottedString());
                        publicKeysProps.put(dhPropertiesPrefix+DHOperations.DH_AUTH_PROTOCOL_PROPERTY +userName,
                                authOID.toDottedString());
                    }
                    if (privOID !=  null) {
                        privateKeysProps.put(dhPropertiesPrefix+DHOperations.DH_PRIV_PROTOCOL_PROPERTY
+userName,
                                privOID.toDottedString());
                        publicKeysProps.put(dhPropertiesPrefix+DHOperations.DH_PRIV_PROTOCOL_PROPERTY +userName,
                                privOID.toDottedString());
                    }
                }
                Date now = new Date();
                privateKeysProps.store(privateKeysOS, "Diffie Hellman private keys generated "+now);
                privateKeysOS.flush();
                privateKeysOS.close();
                publicKeysProps.store(System.out, "Diffie Hellman public keys generated "+now);
                return 0;
            }
        }
    } catch (FileNotFoundException e) {
        e.printStackTrace();
```

```
        return 3;
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
        return 4;
    } catch (IOException e) {
        e.printStackTrace();
        return 5;
    } catch (InvalidAlgorithmParameterException e) {
        e.printStackTrace();
        return 6;
    }
}
```

DH Kickstart

```
private int usmDHKickstartRun() {
    OctetString dhKickstart = new OctetString(DHOperations.DH_KICKSTART_SEC_NAME);
    target.setSecurityName(dhKickstart);
    snmp.getUSM().addUser(dhKickstart, new UsmUser(dhKickstart,
            SnmpConstants.usmNoAuthProtocol, null,
            SnmpConstants.usmNoPrivProtocol, null));
    String dhParametersValArg = (String) ArgumentParser.getValue(settings, "dhp", 0);
    OctetString dhParametersVal = OctetString.fromHexString(dhParametersValArg);
    List<Object> userNames = settings.get("user");
    String privateKeysFilename = (String) ArgumentParser.getValue(settings, "privateKeysFile", 0);
    File privateKeysFile = new File(privateKeysFilename);
    if (!privateKeysFile.exists()) {
        System.err.println("Private key file '"+privateKeysFile+"' does not exist");
        return 1;
    }
    else if (!privateKeysFile.canRead()) {
        System.err.println("Private key file '"+privateKeysFile+"' cannot be read");
        return 2;
    }
    String dhPropertiesPrefix = (String) ArgumentParser.getValue(settings, "dhx", 0);
    if (dhPropertiesPrefix == null) {
        System.err.println("Mandatory parameter 'dhx' with DH properties prefix is not set for DH kickstart
run");
        return 3;
    }
    Properties privateKeysProps = new Properties();
    Map<OctetString, OctetString[]> privateKeyMap = new HashMap<>();
    try {
        DHParameters dhParameters = DHParameters.getDHParametersFromBER(dhParametersVal);
        FileInputStream privateKeysIS = new FileInputStream(privateKeysFile);
        privateKeysProps.load(privateKeysIS);
        for (Object key : privateKeysProps.keySet()) {
            if (key.toString().startsWith(dhPropertiesPrefix+DHOperations.DH_PRIVATE_KEY_PROPERTY)) {
                String userName = key.toString().substring(dhPropertiesPrefix.length()+
                        DHOperations.DH_PRIVATE_KEY_PROPERTY.length());
                if (userName.length() > 0 && (userNames.contains(userName) || (userNames.size() == 0))) {
                    String privateKeyHexString = privateKeysProps.getProperty(key.toString());
                    OctetString privateKeyOctets = OctetString.fromString(privateKeyHexString,16);
                    String publicKeyHexString = privateKeysProps.getProperty(dhPropertiesPrefix+
                            DHOperations.DH_PUBLIC_KEY_PROPERTY +userName);
                    String authProto = privateKeysProps.getProperty(dhPropertiesPrefix+
                            DHOperations.DH_AUTH_PROTOCOL_PROPERTY +userName);
                    String privProto = privateKeysProps.getProperty(dhPropertiesPrefix+
                            DHOperations.DH_PRIV_PROTOCOL_PROPERTY +userName);
                    OctetString publicKeyOctets = OctetString.fromString(publicKeyHexString, 16);
                    privateKeyMap.put(publicKeyOctets,
                            new OctetString[] { privateKeyOctets, new OctetString(userName),
                                    (authProto == null) ? null : new OctetString(authProto),
                                    (privProto == null) ? null : new OctetString(privProto) });
                }
            }
        }
```

```
        Map<OctetString, OctetString[]> publicKeysMap =
                DHOperations.getDHKickstartPublicKeys(snmp, pduFactory, target, privateKeyMap.keySet());
        for (Entry<OctetString, OctetString[]> publicKeyEntry : publicKeysMap.entrySet()) {
           OctetString[] privateKeyInfo = privateKeyMap.get(publicKeyEntry.getKey());
           OctetString publicKeyOctets = publicKeyEntry.getValue()[0];
           if (privateKeyInfo != null) {
                KeyPair keyPair = DHOperations.createKeyPair(publicKeyEntry.getKey(), privateKeyInfo[0],
dhParameters);
                KeyAgreement keyAgreement = DHOperations.getInitializedKeyAgreement(keyPair);
                byte[] sharedKey =
                        DHOperations.computeSharedKey(keyAgreement, publicKeyOctets.getValue(),
                                dhParameters);
                OID authProto = new OID(SnmpConstants.usmHMACMD5AuthProtocol);
                if (privateKeyInfo[2] != null) {
                    authProto = new OID(privateKeyInfo[2].toString());
                }
                OID privProto = new OID(SnmpConstants.usmDESPrivProtocol);
                if (privateKeyInfo[3] != null) {
                    privProto = new OID(privateKeyInfo[3].toString());
                }
                System.out.println(privateKeyInfo[1]+":="+new OctetString(sharedKey).toString(16)+"["+
                        authProto+";"+privProto+"]");
           }
        }
        return 0;
    } catch (FileNotFoundException e) {
        e.printStackTrace();
        // should not happen
        return 1;
    } catch (IOException e) {
        e.printStackTrace();
        return 3;
    }
}
```

DH Key Change

```
public static boolean diffieHellmannKeyChange(Snmp session,
                                              Target target,
                                              PDUFactory pduFactory,
                                              byte[] engineID,
                                              OctetString user,
                                              OID usmUserKeyChangeOID,
                                              OID protocolOID,
                                              OID rowIndex,
                                              OctetString newKey)
        throws UsmException, IOException, TimeoutException {
    if ((target == null) || (engineID == null))
        throw new UsmException("Cannot determine SNMP engine ID", USM_ENGINE_ID);

    if (rowIndex == null) {
        rowIndex = getRowIndex(engineID, user);
    }
    if (usmNoPrivProtocol.equals(protocolOID) || usmNoAuthProtocol.equals(protocolOID)) {
        PDU pdu = pduFactory.createPDU(target);
        pdu.setType(PDU.SET);
        VariableBinding vb = new VariableBinding();
        OID usmSecurityProtocolOID = new OID(protocolOID);
        usmSecurityProtocolOID.append(rowIndex);
        vb.setOid(usmSecurityProtocolOID);
        vb.setVariable(protocolOID);
        ResponseEvent responseEvent = session.send(pdu, target);
        PDU response = responseEvent.getResponse();
        if (response.getErrorStatus() != SnmpConstants.SNMP_ERROR_SUCCESS) {
            throw new UsmException("Cannot set security protocol '"+protocolOID+"': " +
                    response.getErrorStatusText(), USM_SNMP_ERROR,
                    response.getErrorStatus());
```

```
        }
        return true;
    }
    SecurityProtocol securityProtocol = SecurityProtocols.getInstance().getSecurityProtocol(protocolOID);
    if (securityProtocol == null) {
        throw new UsmException("Given security protocol "+protocolOID+" is unknown",
USM_SECURITY_PROTOCOL_UNKNOWN);
    }
    // get DH public key and parameters
    PDU paramPDU = pduFactory.createPDU(target);
    VariableBinding[] vbs = new VariableBinding[2];
    vbs[0] = new VariableBinding();
    vbs[0].setOid(usmDHParameters);
    OID usmUserDHUserKeyChangeOID = new OID(usmUserKeyChangeOID);
    usmUserDHUserKeyChangeOID.append(rowIndex);
    vbs[1] = new VariableBinding();
    vbs[1].setOid(usmUserDHUserKeyChangeOID);
    paramPDU.addAll(vbs);
    // send it
    paramPDU.setType(PDU.GET);
    ResponseEvent responseEvent = session.send(paramPDU, target);
    PDU response = responseEvent.getResponse();
    if ((response.getErrorStatus() != PDU.noError) ||
            (response.size() != 2)) {
        if (response.getType() == PDU.REPORT) {
            throw new UsmException("Cannot get usmDHParameters value - " +
                    "received REPORT PDU instead: " + response,
                    USM_REPORT, response);
        } else {
            throw new UsmException("Cannot get usmDHParameters and usmUserDHUserPrivKeyChange public value: " +
                    response.getErrorStatusText(), USM_SNMP_ERROR, response.getErrorStatus());
        }
    }
    Variable dhParametersVal = response.get(0).getVariable();
    Variable y = response.get(1).getVariable();
    if ((y.getSyntax() != SMIConstants.SYNTAX_OCTET_STRING) ||
            (dhParametersVal.getSyntax() != SMIConstants.SYNTAX_OCTET_STRING)) {
        throw new UsmException("Set privacy failed: usmDHParameters.0 or usmUserDHUserPrivKeyChange." +
                rowIndex.toString() + " do not exist.",
                USM_USER_NOT_FOUND);
    }
    try {
        DHParameters dhParameters = DHParameters.getDHParametersFromBER((OctetString) dhParametersVal);
        KeyPair keyPair = DHOperations.generatePublicKey(dhParameters);
        KeyAgreement keyAgreement = DHOperations.getInitializedKeyAgreement(keyPair);
        byte[] sharedKey = DHOperations.computeSharedKey(keyAgreement, ((OctetString)y).toByteArray(),
dhParameters);
        OctetString z = DHOperations.derivePublicKey(keyPair);

        PDU pdu = pduFactory.createPDU(target);

        vbs = new VariableBinding[1];
        vbs[0] = new VariableBinding();
        vbs[0].setOid(usmUserDHUserKeyChangeOID);
        vbs[0].setVariable(new OctetString(((OctetString)y).getValue(),z.getValue()));
        pdu.addAll(vbs);
        pdu.setType(PDU.SET);
        responseEvent = session.send(pdu, target);
        response = responseEvent.getResponse();
        if (response == null) {
            pdu = pduFactory.createPDU(target);
            vbs[0].setVariable(new Null());
            pdu.add(vbs[0]);
            pdu.setType(PDU.GET);
            responseEvent = session.send(pdu, target);
            response = responseEvent.getResponse();
            if (response == null)
                throw new TimeoutException("Timeout on getting usmUserDHUserPrivKeyChange." + rowIndex.
toString() +
                        " after failed privacy key change");
            if (response.getErrorStatus() != PDU.noError) {
```

```
                throw new UsmException("Cannot get usmUserDHUserPrivKeyChange." + rowIndex.toString() +
                        " after failed privacy key change: " +
                        response.getErrorStatusText(), USM_SNMP_ERROR, response.getErrorStatus());
            }
            OctetString cp = (OctetString)
                    response.get(0).getVariable();
            if (Arrays.equals(((OctetString) y).toByteArray(), cp.getValue()))
                throw new UsmException("Privacy key change failed for unknown reason.", USM_KEY_CHANGE_FAILED);
        } else if (response.getErrorStatus() == PDU.inconsistentValue) {
            return false;
        } else if (response.getErrorStatus() != PDU.noError) {
            throw new UsmException("Setting new privacy password failed: " +
                    response.getErrorStatusText(), USM_SNMP_ERROR, response.getErrorStatus());
        }
        else {
            byte[] usmKey = DHOperations.deriveKey(sharedKey, securityProtocol.getMaxKeyLength());
            newKey.setValue(usmKey);
        }
        return true;
    }
    catch (IOException iox) {
        // TODO
        iox.printStackTrace();
        return false;
    } catch (InvalidAlgorithmParameterException | NoSuchAlgorithmException e) {
        e.printStackTrace();
    }
    return false;
}
```