# Entity Authentication and Credential Verification for GENI Messaging Service

We have implemented features that enable publishing entities, users, or other XMPP clients to authenticate with the GENI messaging service (XMPP server) securely using GENI certificates in order to send XML routed messages to other XMPP servers in the federation. We have also extended the XMPP server (Openfire) code to enable verification of GENI credentials. This allows authenticated clients to publish/subscribe to specific pubsub nodes based on the rights in their credentials. We will demonstrate these features with two use-cases: (a) publishing of ORCA slice manifests by an ORCA service manager to an XMPP server. Entities interested in monitoring the slice manifest, as it evolves, can subscribe to the pubsub nodes based on their credentials, (b) publishing time using a simple Time measurement handler/MP, and subscribing to it - using authentication and credential verification.

Each publishing entity/ XMPP client is assumed to have obtained a GENI certificate issued by a GENI Clearing House (CH), the Certificate Authority (CA) in our case. The common name (CN) in the client certificate has to be a unique id that is used as the XMPP client user name, or Jabber ID (JID). In this work, we have used certificates generated by the OMNI/gcf tool from GENI project office. We have made minor modifications to the gcf tool to generate client certificates with unique CNs. The gcf tools can generate sample GENI certificates for the CH, XMPP servers and the users.

The rest of the document describes (a) how to make secure connections to an XMPP server using TLS, (b) how to authenticate an XMPP client using GENI certificates over the secure connection, and (c) how to use GENI credentials to publish/subscribe to specific pubsub nodes, once the client is authenticated.

## Connection Security

Connection security addresses the question – “Is the connection between the client and the XMPP server secure?” Connection security between an XMPP client and a server is implemented using Transport Layer Security (TLS), which encrypts the client-server connection. For secure connection with certificates to work, the XMPP server must have a server certificate (issued by any CA) and a private key inserted into the XMPP server keystore. Server and client TLS policies need to be enabled on the XMPP server. For certificate-based authentication in the next step, client certificate policy also needs to be enabled. The exact properties corresponding to these settings for the Openfire server are described in detail on the GENI-IMF wiki page.

When a publishing entity or a client connects to the XMPP server, the TLS negotiation goes through the following steps. The ciphers are exchanged and the server sends its certificate, which is verified by the client. Since client certificate policy is enabled, the server asks the client to present a client certificate, which is saved for later use during client authentication. The client sends the client certificate to the server, which, in our case, is the GENI user certificate generated by the gcf tool. Then the client encrypts a random number with the server’s public key and sends it to the server. The server decrypts the random number with its private key and generates a session key using that random number. The client also generates the session key from the same random number. This session key is used to encrypt further communication between the client and the server, ensuring connection security.

## Client Authentication

Once a secure communication channel is established, client authentication can proceed. Client authentication addresses the question – “Can a client authenticate with the XMPP server using authentication mechanisms advertised by the XMPP server?” There are several SASL client authentication mechanisms available for XMPP servers, like PLAIN, DIGEST-MD5 etc. For certificate-based authentication to work, the SASL EXTERNAL mechanism needs to be enabled on the XMPP server. Also, the client certificate verification policies must be enabled on the XMPP server. The certificate of the CA that issues certificates to the clients/publishing entities, which in our case is the CH certificate, needs to be inserted into the client truststore for the XMPP server. Details on these steps are available on the wiki.

During client authentication using SASL EXTERNAL, the server uses the client certificate obtained during TLS negotiation step and checks it against the client truststore. Since the JID of the client must also match the CN in the client certificate for authentication to work, we have extended our publishing clients to create client accounts on the XMPP server on-the-fly with username same as the CN on the client certificate (if it doesn’t exist already). After SASL EXTERNAL authentication is successful, the clients can securely send XML-routed messages to other XMPP servers in the federation.

## Credential Verification

Being able to connect securely and to authenticate with an XMPP server drives us to the next question – “Does the client have credentials to publish and subscribe to a pubsub node?” This leads to the issues of how to generate client credentials and how to verify the client credentials on the server when the client is trying to publish or subscribe to a pubsub node.

### Generating client credentials

Given the client certificate, the CH certificate, and the XMPP server certificate and private key, it is possible to generate GENI XMLSEC credentials using the gcf tools. This is a signed xmlsec credential that contains the owner’s certificate (the client in our case), the target’s certificate (the XMPP server in our case), and the rights the owner has on the target.

We have extended the gcf tools to generate custom pubsub credentials. The credential generation step also takes as arguments pubsub namespaces, on which the rights are given. For example, if one passes “measurement/polatis” in the credential generation step, the client will have the following rights - ‘pub\_ measurement/polatis’ and ‘sub\_ measurement/polatis’, which would enable the client to publish and subscribe to any pubsub node that has a node prefix ‘/measurements/polatis’. This means a publisher/subscriber client with these rights would be able to publish/subscribe to any pubsub node with ‘/measurements/polatis/\*’. These credentials are stored in a known webserver to be retrieved later in the credential verification step. The credential filename contains the JID/CN of the client. This enables the credential verification code on the server to pull up the credential for the client that is trying to publish/subscribe.

### Verifying credentials

We have extended the Openfire server’s pubsub code to enable credential verification. All the existing publish/subscribe policies were augmented with GENI credential verification, which now gates publish or subscribe by a client. The canPublish() and canSubscribe() methods for all publish and subscribe models supported by Openfire have been extended. Publish or subscribe is allowed only if the credential verification step succeeds.

To verify the credentials, the credential XMLSEC file is pulled from a known webserver. The rights are generated from the pubsub node that the client is trying to publish/subscribe to. For example, if the pubsub node is ‘/measurements/polatis/renci’ and a client is trying to publish to the node, the rights generated would be ‘pub\_measurements’, ‘pub\_measurements/polatis’, and ‘pub\_measurements/polatis/renci’. These rights are passed to the credential verification code, which would check whether any of these rights exist in the credential file pulled from the webserver. For example, if the client credential has ‘pub\_measurements’ rights, the verification will succeed. Of course, the check for rights is in addition to complete credential verification - checking whether the credential is valid in the first place, whether it is tampered with, whether the owner’s certificate CN matches the JID of the client, whether credential has expired etc.