**Messaging for In-App and Web History with User Verification:** Securely connect your customers to their own messaging history when you create JSON keysets with User Verification. This feature increases the Messaging for Web conversation’s history from six hours to an unlimited display time frame. Messaging for In-App continues to display messaging history for an unlimited time frame. To maximize data privacy, securely connect your customers to their messaging history and persist conversations across devices with token-based or credential-based user verification. Persist conversations across devices with token-based user verification. Add token-based user verification to Messaging for In-App or Messaging for Web on an external site. If you use Messaging for Web on a Salesforce site, choose between token-based user verification or credential-based user verification.

Admins with Customize Application permission can set up User Verification for a mobile app or external website. User Verification doesn’t work with experience sites.

**Prechat form Details:**

<https://help.salesforce.com/s/articleView?id=release-notes.rn_miaw_pre_chat_frequency.htm&release=244&type=5>

Click **Edit Pre-Chat** in your deployment settings page and select **Every Conversation** or **Every Session** from the Pre-Chat Display Frequency dropdown.

If you select **Every Conversation**, the display frequency of the pre-chat form depends on whether you add user verification to your messaging channel. If you add user verification, a Messaging for In-App and Web conversation has no end. Your customer sees the pre-chat form only when messaging you for the first time. If you don’t add user verification, a customer can click **End Chat** to end a conversation. They see the pre-chat form the first time that they message, and each subsequent time that they end a conversation and start a new one.

If you select **Every Session**, the display frequency of the pre-chat form also depends on whether you add user verification to your messaging channel:

* In Messaging for In-App without user verification, the messaging session ends when the agent clicks **End Chat**. Your customer sees the pre-chat form the first time that they message you, and each time they begin a new session after an agent ended the previous one.
* In Messaging for Web without user verification, the messaging session ends when the customer or agent clicks **End Chat.** Your customer sees the pre-chat form the first time that they message you, and each time they begin a new session after a previous session is ended.
* In Messaging for In-App with user verification, the messaging session ends when the agent clicks **End Chat**. Your customer must be logged in to see the messaging conversation, including the pre-chat form. Your customer sees the pre-chat form the first time that they message you, and the next time they log in and begin a new session after a previous session was ended by an agent.
* In Messaging for Web with User Verification, the messaging session ends when the agent clicks **End Chat**. Your customer sees the pre-chat form the first time that they message you, and the next time they begin a new session after the previous session was ended by an agent.

Important Token-based user verification is available for a mobile app, an external website, or some Aura sites (Build Your Own Aura, Help Center, and Customer Service templates). When you’re using the Messaging for In-App SDK for mobile apps, we don’t support verified users alongside unverified users. Your mobile app implementation must be designed for either verified users or unverified users.

Step1: Install JDK11 or above.

Step2: Download Maven and set path of maven in environment variables.

Step3: Clone the project git clone.

* https://github.com/Salesforce-Async-Messaging/key-command-line-utility.git

Step4: Run this command to build project.

* cd key-command-line-utility/
* mvn clean install

Step 5: Download openssl

# Step 6: Setup-> Certificate and Key Management🡪 Create Self Signed certificate.

Step7: Setup🡪 Certificate and Key Management🡪 Click on “Export to Keystore”. We need to pass some password for exporting.

Step 8: Get certificate name from keystore. Copy alias name of certificate after executing below command. When this command is executed then we need to pass password which we mentioned in Step7

keytool -list -v -keystore {keystorefile.jks}

Step 9: Convert keystore jks file to .p12 file

keytool -importkeystore -srckeystore {keystorefile.jks from step7} -srcalias {certificate alias name from step8} -destkeystore {filename with .p12 extension} -srcstoretype JKS -deststoretype PKCS12 -srcstorepass {password which we provided in step7} -deststorepass {new password which will require later, you can provide same password from Step7} -destkeypass {new password which will require later, you can provide same password from Step7} -noprompt

Step10: Get private ket from .p12 file from step9. We need to provide the password which we provided in step9.

openssl pkcs12 -in {.p12 file path} -nocerts -nodes -out {private key file name with .pem extension}

Step 11: Get certificate from .p12 file. We need to provide password which we provided in step9.

openssl pkcs12 -in {.p12 file path} -clcerts -nokeys -out {certificate file name with .crt extension}

Step 12: Get public key from certificate which we got in Step11.

openssl x509 -pubkey -noout -in {certificate file path on Step11} > {public key file name with .key extension}

Step 13: Get JKS file from public key and certificate. We need to go to project which we cloned.

java -jar target/key-command-line-utility.jar --generateJwk -kid {provide any key id, same we need to use while generating JWT token, use some complex id}-alg RS256 -publicKeyFile {public key file path } -publicCertFile {certificate file path}

copy the content and store in file with .json extension

Step 14: Setup🡪 Messaging for In-App and Web User Verification 🡪 Click on “New Key”🡪 Note down key name somewhere as after saving, you will not see key on UI but created successfully.

Step 15: Setup 🡪 Messaging for In-App and Web User Verification 🡪 Click on “New Keyset” 🡪 Select the key which we created in step14 and provide any value in “JSON Web Key Issuer” but note down somewhere as we need to use while generating JWT token in “Iss” field. Once you will click on “save” then in UI, you will not see any keyset but that is created successfully.

Step 16: Setup🡪 Messaging Settings 🡪 Click on edit button of your channel 🡪 Click on “Add User Verification” checkbox and “Authorization Token Expiration Time for Verified Users” field default value is 60 minutes and make sure when we generate JWT token then we should use same expiration.

Step 17: Setup🡪 Messaging Settings 🡪 click on your channel 🡪 Click on “New” button from “User Verification Configuration”🡪 Select the keyset which created on Step15 and provide any name in configuration.

## Generate JWT token: We need to generate jwt token using header, payload, and private key.

Header:

{

"kid":{KID which we provided in step 13},

"typ": "JWT",

"alg": "RS256"

}

{

"kid": "123456",

"typ": "JWT",

"alg": "RS256"

}

Payload:

{

"sub”: {Customer identification that can be email in our case},

"iss”: { JSON Web Key Issuer which we provided in step15},

"exp”: {Expiration time in milisec},

"iat": {Issuer time in milisec}

}

{

"sub": "user-123",

"iss": "example.com",

"exp": 1674164345,

"iat": 1674158345

}

Add below code in script:

window.addEventListener("onEmbeddedMessagingReady", () => {

console.log("Received the onEmbeddedMessagingReady event.");

// Send your identity token to Salesforce.

embeddedservice\_bootstrap.userVerificationAPI.setIdentityToken({

identityTokenType: "JWT",

identityToken: {JWT Token Provide}

});

});

window.addEventListener("onEmbeddedMessagingIdentityTokenExpired", () => {

console.log("Received the onEmbeddedMessagingIdentityTokenExpired event.");

// Refresh the identity token and send it to Salesforce.

embeddedservice\_bootstrap.userVerificationAPI.setIdentityToken({

identityTokenType: "JWT",

identityToken:{JWT token provide},

});

});

//Call below code when external site logout

embeddedservice\_bootstrap.userVerificationAPI

.clearSession(true)

.then(() => {

// TO DO: Handle success

})

.catch((error) => {

// TO DO: Handle error

})

.finally(() => {

// TO DO: Handle regardless of result

});