AGIV Java Security Design

Highlights the architecture and design of the AGIV Java Security framework.
Version 1.0.1

Frank Cornelis

28 April 2013

Abstract

In this document we describe the architectural choices behind the AGIV Java Security framework.

1. Introduction

The `AGIVSecurity` component plays a central role within the framework. An `AGIVSecurity` instance corresponds with a certain user credential (e.g. username/password). This component basically decorates vanilla JAX-WS stubs with different JAX-WS SOAP handlers to add the required AGIV security behavior. Each provided JAX-WS SOAP handler implements a different part of a WS-* specification. The framework comes with different clients for the IP-STS, R-STS and secure conversations that are required by the `AGIVSecurity` component for acquiring the security tokens. The `AGIVSecurity` component can also handle HTTP/SOCKS network proxies.

1.1. Dependencies

During the AGIV Java Security framework development we tried to use as less dependencies as possible. This in order to avoid runtime conflicts with the different web service stacks as much as possible. We could have used for example the Apache CXF STSClient, but this would have introduced Apache CXF as dependency. Basically we only want to depend on the JAX-WS API. Of course certain operations, like the creation of the different WS-Security XML signatures, require us to include specific dependencies like Apache WSS4J. All used dependencies also run on Java 5.

For logging we use Apache Commons Logging.

The AGIV Java Security framework has been tested under Apache WSS4J version 1.6.4, 1.6.5, 1.6.6, 1.6.7, 1.6.8, 1.6.9 and 1.6.10. Apache WSS4J version 1.6.5 and higher is using Apache XML Security 1.5.1 which has a much stricter Id resolution policy. This might conflict at run-time with other libraries that use Apache XML Security to create/verify XML signatures. For this reason we bundle the AGIV Java Security framework with Apache WSS4J version 1.6.4.

1.2. Token Caching

The `AGIVSecurity` component takes care of the caching of the different tokens. A security token is represented in Java using the `SecurityToken` class.

Each `AGIVSecurity` instance caches its IP-STS token.

Each `AGIVSecurity` instance caches the R-STS tokens on a per web service location basis.

Each `AGIVSecurity` instance caches the secure conversation tokens (if applicable) on a per web service location basis.

The `AGIVSecurity` component will reacquire a token at a certain time before it expires. This time before expiration is configurable at the `AGIVSecurity` level.

1.3. Multi-threading

JAX-WS is inherently not multi-threaded by specification. However, an `AGIVSecurity` component instance can be shared between different threads. The multi-threaded support has been achieved without the usage of synchronization monitors by using `ConcurrentHashMap` as much as possible for the implementation of the different token caches.

The STS clients use JAX-WS themselves to request security tokens from the different STS based services. As such the STS clients are not thread-safe as per the JAX-WS specification. The `AGIVSecurity` component itself thus uses different STS client instances when acquiring a certain security token.

1.4. STS Clients

There is an STS client for each STS service:

- IP-STS to acquire a security token based on user credentials.
- R-STS to acquire security tokens based on IP-STS security tokens that apply to a certain AGIV web service.

- Secure conversation contexts that are required by WS-SecureConversation enabled AGIV web service.

Although JAX-WS 2.1 comes with WS-Addressing support, the different STS clients have their own WS-Addressing implementation. This was done in order to avoid a runtime conflict when Apache Axis2 is used on a Java 6 JRE.

Each STS client is specifically tailored for its corresponding AGIV security policy. The STS clients do not interpret the WS-Policy and WS-SecurityPolicy at runtime. As such the STS client implementation is strongly linked with this AGIV security policy. Changes in the AGIV security policy will most likely require changes within the different STS clients.

2. AGIV Security Protocol

In this section we analyze the AGIV Security Protocol. The analysis only highlights the most relevant cryptographic properties of the protocol. As such protocol details that do not impact the security properties are not mentioned.

We first define a simple formal protocol notation. The syntax is similar to that of the BAN logic. We do not formalize the believes and possessions of each party participating in a protocol run.

Principal A sends to principal B a message M:

$A \rightarrow B : M$

Principal A sends to principal B messages M_1 and M_2 :

$A \rightarrow B : M_1 , M_2$

Principal A sends to principal B a message M over an SSL secured connection:

$A \rightarrow_{SSL} B : M$

Message M is encrypted with symmetric key K:

$\{M\}_K$

Message M is encrypted with public key K^+ of principal A :

$\{M\}_{K^+_A}$

Message M is signed with symmetric key K:

$sign_K (M)$

The algorithm for such symmetric key signatures is HMAC-SHA1.

Message M is signed with the private key of principal A:

$sign_{K^-_A} (M)$

2.1. IP-STS with Username/password credentials

The relying party (RP) starts by sending a message to the IP-STS. This message contains the credentials and some entropy value.

RP $\rightarrow_{SSL}$ IP-STS : username, password, Ent $_{RP}$, AppliesTo $_{R\text{-}STS}$

The IP-STS answers with a token and some entropy value:

IP-STS $\rightarrow_{SSL}$ RP : Ent $_{IP\text{-}STS}$, Token $_{IP\text{-}STS}$

Where the token actually is:

Token $_{IP\text{-}STS}$ = {Token'} $_K$, {K} $_{K^+_{R\text{-}STS}}$

With K a symmetric AES key and $K^+_{R\text{-}STS}$ the public RSA key of R-STS. So only the R-STS can read the value of Token'. The relying party itself cannot read the content of Token'.

From the Ent $_{RP}$ and Ent $_{IP\text{-}STS}$ entropy values both parties calculate a secret.

$K_{POP\ IP\text{-}STS}$ = PSHA1(Ent $_{RP}$, Ent $_{IP\text{-}STS}$)

The PSHA1 algorithm is defined as part of the SSL specifications. This secret is later on used as proof-of-possession (POP) for the acquired IP-STS security token.

2.2. R-STS

The relying party sends the IP-STS security token together with a proof-of-possession signature to the R-STS.

RP $\rightarrow_{SSL}$ R-STS : Token $_{IP\text{-}STS}$, T $_{RP}$, sign $_{K_{POP\ IP\text{-}STS}}$ (T $_{RP}$), AppliesTo $_{SP}$

The relying party indicates that the requested security token should apply to a certain service provide (SP). The relying party is not signing the IP-STS security token, but the timestamp T $_{RP}$. This allows the R-STS to believe that the relying party recently said that it indeed wants to use the corresponding security token for accessing a certain web service.

The R-STS can decrypt the security token because the token has been encrypted with the private key of R-STS.

The R-STS answers with a new security token and a corresponding proof key.

R-STS $\rightarrow_{SSL}$ RP : Token $_{R\text{-}STS}$, K $_{R\text{-}STS}$

2.3. Secure Conversation

For AGIV web services that use WS-SecureConversation the relying party needs to create a secure conversation before being able to invoke a business web service.

2.3.1. Acquiring a secure conversation token

The relying party sends the following message to a service provider SP:

$RP \rightarrow_{SSL} SP$: $Token_{R\text{-}STS}$, T_{RP} , $sign_{K_{R\text{-}STS}}(T_{RP})$, Ent_{RP}

Where Ent_{RP} is a new entropy value generated by the relying party and T_{RP} is also a new timestamp value generated by the relying party.

The service provider answers with:

$SP \rightarrow_{SSL} RP$: $Token_{SP}$, Ent_{SP}

Where $Token_{SP}$ is the secure conversation token.

Both relying party and service provider calculate a secret out of the entropy values:

$K_{POP\ SP} = PSHA1(Ent_{RP}, Ent_{SP})$

This secret is later used as proof-of-possession (POP) for the acquired secure conversation token.

2.3.2. Using a secure conversation token

The relying party can invoke a business web service using the secure conversation token.

$RP \rightarrow_{SSL} SP$: M, $Token_{SP}$, T_{RP} , $sign_{K_{POP\ SP}}(T_{RP})$

Where M is the payload for the business web service request and T_{RP} is a new timestamp generated by the relying party.

The business web service answers with some response message M'.

$SP \rightarrow_{SSL} RP$: M'

2.3.3. Cancelling a secure conversation token

After invoking the business web service the relying party can cancel the secure conversation token.

$RP \rightarrow_{SSL} SP$: $Token_{SP}$, T_{RP} , $sign_{K_{POP\ SP}}(T_{RP})$

Where T_{RP} is a new timestamp generated by the relying party.

The service provider acknowledges the operation.

$SP \rightarrow_{SSL} RP$: ack

2.4. SAML secured web services

Not every AGIV web service requires a WS-SecureConversation to operate. In this case the web service simply requires the R-STS SAML token for authentication/authorization.

The relying party sends the following message to a service provider SP:

$RP \rightarrow_{SSL} SP$: M, $Token_{R\text{-}STS}$, T_{RP} , $sign_{K_{R\text{-}STS}}(T_{RP})$

Where M is the payload for the business web service request and T_{RP} is a new timestamp value generated by the relying party.

The business web service answers with some response message M'.

$SP \rightarrow_{SSL} RP : M'$

2.5. WS-Addressing

The WS-Addressing adds the following to each request message:

$RP \rightarrow_{SSL} SP : N_{RP} , A_{SP} , L_{SP} , \ldots$

Where N_{RP} is a nonce generated by the relying party, A_{SP} is the action that the relying party wants to invoke on the service provider, and L_{SP} is the location of the service provider.

The service provider relates the response with the request by echoing the nonce N_{RP} in the response message.

$SP \rightarrow_{SSL} RP: N_{RP} , \ldots$

This mechanism prevents replay attacks and ensures recognizability of the messages towards the service provider.

2.6. WS-Security timestamp

The service provider adds a timestamp to each response message.

$SP \rightarrow_{SSL} RP: T_{SP} , \ldots$

Besides message freshness, the yielded cryptographic properties of this timestamp are less clear given the replay attack countermeasure provided by the WS-Addressing and the fact that the security tokens have a limited lifetime anyway.

2.7. IP-STS with certificate credentials

The IP-STS service also support X509 certificates as user credential. The relying party (RP) starts by sending a message to the IP-STS. This message contains the certificate and a signature using the corresponding private key.

$RP \rightarrow_{SSL} IP\text{-}STS : Cert_{RP} , Ent_{RP} , AppliesTo_{R\text{-}STS} , sign_{K^{-}_{RP}} (L_{IP\text{-}STS} , T_{RP})$

The signature signs the location of the IP-STS service and a timestamp generated by the relying party. The message also contains some entropy value generated by the relying party.

The IP-STS answers with a token and some entropy value:

$IP\text{-}STS \rightarrow_{SSL} RP : Ent_{IP\text{-}STS} , Token_{IP\text{-}STS}$

Further behaviour is already described under *Section 2.1, “IP-STS with Username/password credentials”* .

A. AGIV Java Security Design License

B. Revision history

Table B.1. Revision history

Date	Author	Description
8 Jan 2012	Frank Cornelis	Initial version.
29 Jan 2012	Frank Cornelis	X509 credentials.
24 Mar 2012	Frank Cornelis	Optional WS-SecureConversation.
26 Aug 2012	Frank Cornelis	Apache WSS4J tests.
28 Apr 2013	Frank Cornelis	Apache WSS4J tests.