# NativeSPSigningEncryption

The SP supports various profiles during which it will generate and send messages to other systems, and there are settings to control the use of digital signatures, encryption, and how they interact with the use of TLS in SOAP exchanges.

Relative to an IdP, there are fewer scenarios in which it typically becomes necessary to worry about these settings. Usually you can accept the defaults and things will work most of the time. The settings are provided for handling more unusual situations.

## Version 2.6+

The `signing` and `encryption` settings are typically set in the `<ApplicationDefaults>`, `<ApplicationOverride>`, and `<RelyingParty>` elements and allow for per-application, per-IdP behavior.

It's also possible in more unusual cases to set them in the `<SSO>`, `<Logout>`, and `<NameIDMgmt>` elements, allowing for per-protocol behavior. They can also be set in advanced configurations that define individual `<SessionInitiator>`, `<LogoutInitiator>`, `<md:AssertionConsumerService>`, and `<md:SingleLogoutService>` endpoints. In these cases, they override any values set by application or IdP.

The five possible values are:

- `conditional`
- `true`
- `false`
- `front`
- `back`

The default value is *usually* `conditional` but is somewhat context-dependent, and defaults to `false` (with a caveat) for SAML 2.0 SSO initiation. The caveat with SAML 2.0 authentication is that omitting the setting defaults to a softer false that really means "don't sign unless the IdP's metadata includes the `WantAuthnRequestsSigned` flag and the SP can do so". Unless explicitly disabled, the metadata will typically cause the SP to sign if it can do so.

The goal going forward is for the default behavior to be "what's expected" in any given case.

As in earlier versions, the "true" and "false" settings explicitly turn signing and encryption on or off unilaterally. The "front" and "back" settings explicitly turn them on based on whether the message is being sent through the client or directly to the peer. The "front channel" applies to cases like Browser SSO requests, while the "back channel" applies to queries. Logout can happen over either channel.

The newly-defined "conditional" setting is a new tool to allow for more automated decision-making based on the profile/protocol, channel, and characteristics of the communication path. Generally, conditional implies "true" for front-channel messages except for SSO, and it implies "false" for back-channel messages **unless** the back-channel endpoint does not support TLS or runs on the default TLS port.

As an example, a SOAP message to `https://idp.example.com:8443/idp/profile/SAML/SOAP/AttributeQuery` **will not** be signed, nor its content encrypted, while the same message sent to `https://idp.example.com/idp/profile/SAML/SOAP/AttributeQuery` **will** be signed and its content encrypted.

The intent of this algorithm is to effect an automatic transition to use of message signing and encryption when back-channel communication takes place over the default TLS port, eliminating the need for browser-facing ports to support client TLS, making load balancing simpler. The algorithm can always be overridden in individual situations, but the default behavior automates a general move from an older paradigm to a newer one with hopefully little work by deployers apart from simply maintaining current software.

This approach mirrors a similar algorithm used by the Shibboleth V3 IdP software, though SOAP communication from it is much more rare.

### Failures

Failure while signing or encrypting is usually handled differently. When signing fails, it's common for the message to be sent unsigned. When encryption fails, the message isn't sent *unless* the encryption was triggered through the use of the "conditional" setting. The most common source of failure in either case is the inability to locate a key, but in the case of encryption, that key belongs to the peer, and is generally obtained from SAML metadata.

The rationale behind proceeding in a conditional encryption case is that ultimately the data encrypted belongs to the peer, and if it can't be bothered to support encryption, then it doesn't get encryption.

### Transport Authentication

An additional setting (`requireTransportAuth`) exists to control whether to allow for a back-channel peer to authenticate itself to the SP using a message signature, or whether the TLS connection must be authenticated. It defaults to "conditional", requiring the server's TLS certificate to be trusted (generally through metadata) in the cases in which signing and encryption would **not** be used by default.

This change in default behavior is intended to facilitate the move to bidirectional use of signatures. Note that when transport authentication is not required, that does not open the system to any exploit. It does mean that messages will be sent to the peer before it's known whether the response is authentic, but if the response isn't signed and trusted, then it will be rejected anyway.

## Version 2.5 and Earlier

Prior to Version 2.6, the `signing` and `encryption` settings can be used in the `<ApplicationDefaults>`, `<ApplicationOverride>`, and `<Relying Party>` elements and allow for per-application, per-IdP behavior.

The four possible values are:

- `true`
- `false`
- `front`
- `back`

and the default setting is `false`.

The "true" and "false" settings obviously explicitly turn signing and encryption on or off unilaterally. The "front" and "back" settings explicitly turn them on based on whether the message is being sent through the client or directly to the peer. The "front channel" applies to cases like Browser SSO requests, while the "back channel" applies to queries. Logout can happen over either channel.

## Failures

Failure while signing or encrypting is usually handled differently. When signing fails, it's common for the message to be sent unsigned. When encryption fails, the message isn't sent. The most common source of failure in either case is the inability to locate a key, but in the case of encryption, that key belongs to the peer, and is generally obtained from SAML metadata.

## Transport Authentication

An additional setting (`requireTransportAuth`) exists to control whether to allow for a back-channel peer to authenticate itself to the SP using a message signature, or whether the TLS connection must be authenticated. It defaults to "true", requiring the server's TLS certificate to be trusted (generally through metadata). Prior to V2.6, this setting has to be explicitly turned off to allow for signature-based communication in both directions, which is the main reason why the back channel has traditionally only worked when a separate port and certificate are deployed by the peer.