Web サービスの保護

Version 2024.1
2024-06-03

InterSystems Corporation　1 Memorial Drive　Cambridge MA 02142　www.intersystems.com

Web サービスの保護
InterSystems IRIS Data Platform　Version 2024.1　2024-06-03

インターシステムズ社の製品に関するサポートやご質問は、以下にお問い合わせください：

InterSystems Worldwide Response Center (WRC)
Tel:　+1-617-621-0700
Tel:　+44 (0) 844 854 2917
Email:　support@InterSystems.com

目次

1

SOAP による Web サービスの保護

InterSystems IRIS は、セキュリティを Web サービスおよび Web クライアントに追加する方法を記述する WS-Security、WS-Policy、WS-SecureConversation、および WS-ReliableMessaging の各仕様の一部をサポートしています。このトピックでは、ツールの概要を示し、サポートされる規格をリストします。

認証を必要とする Web サービスが InterSystems IRIS Web クライアントで使用されており、これを使用する特別な理由がない場合は、従来の WS-Security ログイン機能を使用できます。"WS-Security ログイン機能の使用法" を参照してください。

1.1 SOAP セキュリティに関連する InterSystems IRIS のツール

InterSystems IRIS には、Web サービスおよび Web クライアントのセキュリティに関連する以下のツールが用意されています。

- 着信メッセージで受信した証明書とシグニチャを検証する際に InterSystems IRIS が使用する信頼された証明書を用意する機能。
- X.509 証明書を表現する機能。所有している証明書および通信先のエンティティの証明書を IRISSYS データベースに格納できます。所有している証明書では、発信メッセージに署名する必要がある場合、対応する秘密鍵も格納できます。

  IRISSYS データベースにおいて、X.509 証明書は、InterSystems IRIS 資格情報セット内 (具体的には、%SYS.X509Credentials のインスタンス内) に格納されます。このクラスのメソッドを使用して、証明書 (および、オプションで該当する場合には関連付けられた秘密鍵ファイル) をデータベースにロードします。このメソッドを直接実行するか、または管理ポータルを使用できます。

  資格情報セットを所有するユーザおよびこれを使用できるユーザを指定できます。

  また、%SYS.X509Credentials クラスは、エイリアス、サムプリント、サブジェクト・キー識別子などによって、証明書にアクセスするメソッドも提供します。セキュリティ上の理由により、%SYS.X509Credentials クラスは、通常のオブジェクトおよび SQL テクニックを使用してアクセスすることはできません。
- SSL (Secure Sockets Layer) および TLS (Transport Layer Security) のサポート管理ポータルを使用して InterSystems IRIS SSL/TLS 構成を定義し、この構成を使用して、X.509 証明書によって、InterSystems IRIS Web サービスまたは Web クライアントから、あるいはこれらへの通信を保護できます。

  SSL/TLS 構成については、インターシステムズの "TLS ガイド" を参照してください。
- WS-Policy のサポート。InterSystems IRIS は、WS-Policy の情報を InterSystems IRIS Web サービスまたは Web クライアントにアタッチする機能を提供します。ポリシーは、以下のような項目を指定できます。
  - WS-SecureConversation の使用。

- SSL/TLS の使用。
- 使用する WS-Security 機能または必要な WS-Security 機能。
- 使用する WS-Addressing ヘッダまたは必要な WS-Addressing ヘッダ。WS-Addressing ヘッダについては、"Web サービスおよび Web クライアントの作成" を参照してください。これらのヘッダを手動で追加する方法についても説明しています。
- MTOM (Message Transmission Optimization Mechanism) パッケージの使用。MTOM については、"Web サービスおよび Web クライアントの作成" を参照してください。MTOM パッケージを手動で使用する方法についても説明しています。

ポリシーは、個別の構成クラスで作成されます。そのクラスでは、XData ブロックを使用して、ポリシー (XML ドキュメント) を格納し、ポリシーのアタッチ先のサービスまたはクライアントの部分を指定します。ポリシーは、サービス全体、クライアント全体、または特定のメソッド (あるいは、特定の要求メッセージもしくは応答メッセージ) にアタッチできます。

スタジオ・ウィザードを使用して、この構成クラスを作成できます。ウィザードは、一連の事前定義されたポリシーおよび豊富なオプションを提供します。ポリシーが X.509 証明書を必要とするときには、必ずウィザードによって、IRISSYS に格納されている証明書から選択できるようになります。同様に、ポリシーが SSL/TLS を必要とする場合は、必ず既存の SSL/TLS 構成の選択も可能になります (該当する場合)。

必要に応じて、後でポリシーを直接編集できます。ポリシーは、構成クラスをコンパイルしたときに有効になります。

- WS-Security 要素の直接作成および直接作業のサポート。InterSystems IRIS には、`<UsernameToken>` や `<Signature>` などの WS-Security ヘッダ要素を表すための一連の XML 対応クラスが用意されています。これらの専用クラスは、これらの要素の作成や変更に使用するメソッド、および要素間の参照を提供します。

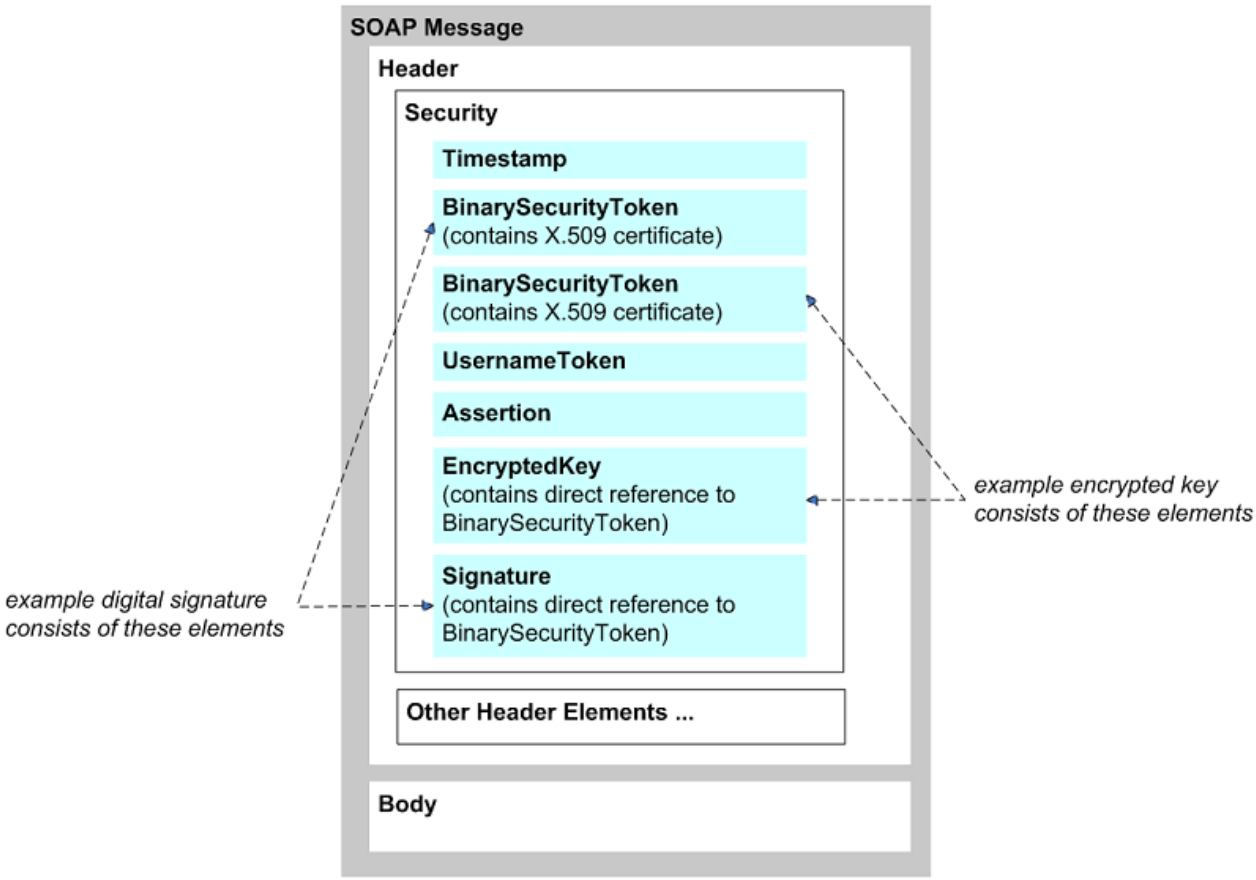
  WS-Policy サポートを使用している場合、InterSystems IRIS は、これらのクラスを自動的に使用します。WS-Security サポートを直接使用する場合、これらのクラスのインスタンスを作成してセキュリティ・ヘッダに挿入するコードを記述します。

  いずれの場合も、InterSystems IRIS Web サービスまたは Web クライアントが SOAP メッセージを WS-Security 要素と共に受信すると、これらの要素を表すためにこれらのクラスのインスタンスが作成されます。また、%SYS.X509Credentials のインスタンスも作成し、着信メッセージで受信されたすべての証明書を格納します。
- WS-SecureConversation 要素の直接作成および直接作業のサポート。InterSystems IRIS には、これらの要素を表すための一連の XML 対応クラスが用意されています。Web サービスでコールバック・メソッドを定義し、Web サービスが安全な通信の要求に応答する方法を制御します。

WS-Policy を使用するか、または WS-Security および WS-SecureConversation を直接使用することができます。WS-Policy を使用する場合、必要に応じて WS-Security ツールが自動的に使用されます。WS-Security または WS-SecureConversation を直接使用する場合は、さらにコーディングする必要があります。

1.2 WS-Security ヘッダの概要

SOAP メッセージは、WS-Security ヘッダ要素内 (SOAP `<Header>` 要素の `<Security>` サブ要素) にセキュリティ要素を保持します。以下の例は、考えられるコンポーネントの一部を示しています。



これらの要素は、以下のとおりです。

- タイムスタンプ・トークン (<Timestamp>) には <Created> 要素と <Expires> 要素があり、このメッセージの有効期間の範囲が指定されます。タイムスタンプは、厳密に言うと、セキュリティ要素ではありません。ただし、タイムスタンプが署名されている場合、リプレイ攻撃を阻止するためにタイムスタンプを使用できます。
- バイナリ・セキュリティ・トークン (<BinarySecurityToken>) は、受信者がシグニチャの検証または暗号化要素の解読をできるようにするための情報を含むバイナリ・エンコード・トークンです。これらは、シグニチャ要素、暗号化要素、およびアサーション要素で使用することができます。
- ユーザ名トークン (<UsernameToken>) によって、Web クライアントによる Web サービスへのログインが可能になります。このトークンには、Web サービスで要求されるユーザ名とパスワードが組み込まれており、既定ではクリア・テキストで組み込まれています。パスワードを保護するいくつかのオプションがあります。
- アサーション要素 (<Assertion>) には、作成した SAML アサーションが組み込まれます。このアサーションは、署名済みでも未署名でもかまいません。

  アサーション要素には、サブジェクト確認要素 (<SubjectConfirmation>) を含めることができます。この要素は、Holder-of-key メソッドまたは Sender-vouches メソッドを使用できます。前者の場合、アサーションは、他の目的で使用できるキー・マテリアルを保持します。
- 暗号化キー要素 (<EncryptedKey>) には、キーが格納され、暗号化メソッドが指定されているほか、他の同様の情報が含まれています。この要素には、メッセージがどのように暗号化されたのかが記述されます。"暗号化の概要" を参照してください。
- シグニチャ要素 (<Signature>) は、メッセージの部分に署名します。非公式のフレーズ メッセージの部分への署名は "デジタル・シグニチャの概要" で説明されているように、シグニチャ要素がメッセージのこれらの部分に適用されることを意味します。

図ではこれは示されていませんが、シグニチャ要素には、メッセージの署名済み部分を指す <Reference> 要素が格納されています。

ここで示すように、暗号化キー要素には、前に同じメッセージに含まれていたバイナリ・セキュリティ・トークンへの参照が一般的に含まれており、そのトークンには、受信者が暗号化キーの解読に使用できる情報が含まれています。ただし、トークンへの参照をメッセージの別の場所で持つのではなく、解読に必要な情報を <EncryptedKey> に含めることができます。InterSystems IRIS は、これについて複数のオプションをサポートしています。

同様に、デジタル・シグニチャは、一般的に 2 つの部分から構成されます。すなわち、X.509 証明書を使用するバイナリ・セキュリティ・トークンと、このバイナリ・セキュリティ・トークンを直接参照するシグニチャ要素です(バイナリ・セキュリティ・トークンではなく、代わりに Holder-of-key メソッドを持つ署名済みの SAML アサーションを使用する方法です)。シグニチャが <Signature> 要素のみから構成されることも可能です。この場合、この要素には、受信者がシグニチャを検証できる情報が含まれます。InterSystems IRIS は、これについても複数のオプションをサポートしています。

1.3 SOAP セキュリティ標準

このセクションでは、InterSystems IRIS Web サービスおよび Web クライアントの WS-Security、WS-Policy、WS-SecureConversation、および WS-ReliableMessaging に対するサポートの詳細を示します。

"XML 標準" および "SOAP 標準" も参照してください。

1.3.1 InterSystems IRIS における WS-Security のサポート

InterSystems IRIS は、OASIS で作成された WS-Security 1.1 (http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-pr-SOAPMessageSecurity-01.pdf) の以下の部分をサポートします。

- WS-Security ヘッダ (http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf)
- X.509 トークン・プロファイル 1.1 (http://www.oasis-open.org/committees/download.php/16785/wss-v1.1-spec-os-x509TokenProfile.pdf)
- XML 暗号化 (https://www.w3.org/TR/xmlenc-core/)。以下のアルゴリズムから選択できます。
  - ブロック暗号化 (データ暗号化) :AES-128 (既定)、AES-192、または AES-256
  - 鍵転送 (キー暗号化) :RSA-OAEP (既定) または RSA-v1.5
- Exclusive XML Canonicalization を使用した XML シグニチャ (https://www.w3.org/TR/xmldsig-core/)。以下のアルゴリズムから選択できます。
  - ダイジェスト・メソッド :SHA1 (既定)、SHA256、SHA384、または SHA512
  - シグニチャ・アルゴリズム :RSA-SHA1、RSA-SHA256 (既定)、RSA-SHA384、RSA-SHA512、HMACSHA256、HMACSHA384、または HMACSHA512

    既定のシグニチャ・アルゴリズムは変更できないことに注意してください。変更するには、管理ポータルにアクセスし、**[システム管理]**、**[セキュリティ]**、**[システムセキュリティ]**、**[システムワイドセキュリティパラメータ]** の順にクリックします。既定のシグニチャ・アルゴリズムを指定するオプションには、**[既定のシグニチャ・ハッシュ]** というラベルが付いています。

  暗号化または署名において、バイナリ・セキュリティ・トークンに X.509 証明書が含まれる場合、InterSystems IRIS は X509v3 トークン・タイプの X.509 証明書トークン・プロファイルに従います。キー・マテリアルが SAML アサーションを使用する場合、InterSystems IRIS は、WS-Security SAML トークン・プロファイル仕様に従います。

デジタル・シグニチャの適用先のメッセージ部分を指定できます。

- UsernameToken Profile 1.1 (http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-pr-UsernameTokenProfile-01.pdf)
- SAML バージョン2.0 を基盤としたほとんどの WS-Security SAML トークン・プロファイル 1.1 (http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-SAMLTokenProfile.pdf)。例外は、InterSystems IRIS SOAP サポートには SAML 1.0 または 1.1 を参照する機能がないことです。

  発信 SOAP メッセージの場合、InterSystems IRIS Web サービスおよび Web クライアントは、SAML アサーション・トークンに署名できます。ただし、実際の SAML アサーションの定義は、アプリケーションで行う必要があります。

  着信 SOAP メッセージの場合、InterSystems IRIS Web サービスおよび Web クライアントは、SAML アサーション・トークンを処理して、シグニチャを検証できます。SAML アサーションの詳細はアプリケーションで検証できなければなりません。

  SAML の完全サポートは実装されていません。InterSystems IRIS の SAML サポートは、ここに掲載した詳細のみに適用されます。

1.3.2 InterSystems IRIS における WS-Policy のサポート

WS-Policy 1.2 (https://www.w3.org/Submission/WS-Policy/) と WS-Policy 1.5 (https://www.w3.org/TR/ws-policy) の両方のフレームワークが、関連付けられている特定のポリシー・タイプと共にサポートされます。

- WS-SecurityPolicy 1.1 (http://www.oasis-open.org/committees/download.php/16569/)
- WS-SecurityPolicy 1.2 (http://docs.oasis-open.org/ws-sx/ws-securitypolicy/v1.2/ws-securitypolicy.html)
- Web Services Addressing 1.0 – Metadata (https://www.w3.org/TR/ws-addr-metadata)
- Web Services Addressing 1.0 – WSDL Binding (https://www.w3.org/TR/ws-addr-wsdl)
- WS-MTOMPolicy (https://www.w3.org/Submission/WS-MTOMPolicy/)

`<PolicyReference>` は 2 つの場所のみでサポートされています。構成要素内の `<Policy>` 要素の代わりとして、または `<Policy>` 要素の唯一の子としてです。

WS-SecurityPolicy 1.2 は、以下のようにサポートされます。WS-SecurityPolicy 1.1 の同等の部分もサポートされます。

- 4.1.1 SignedParts は、以下の例外付きでサポートされます。
  - 本文はサポートされます。
  - ヘッダはサポートされます。
  - 添付はサポートされません。
- 4.1.2 SignedElements はサポートされません。
- 4.2.1 EncryptedParts は、以下の例外付きでサポートされます。
  - 本文はサポートされます。
  - ヘッダはサポートされません。
  - 添付はサポートされません。
- 4.2.2 EncryptedElements はサポートされません。
- 4.3.1 RequiredElements はサポートされません。
- 4.2.1 RequiredParts はサポートされます。

  - ヘッダはサポートされます。

- 5.1 sp:IncludeToken はサポートされます。
- 5.2 Token Issuer および Required Claims はサポートされません。
- 5.3 派生キー・プロパティは、X509Token および SamlToken に対してのみサポートされます。
- 5.4.1 UsernameToken はサポートされます。
- 5.4.2 IssuedToken はサポートされません。
- 5.4.3 X509Token はサポートされます。
- 5.4.4 KerberosToken はサポートされません。
- 5.4.5 SpnegoContextToken はサポートされません。
- 5.4.6 SecurityContextToken はサポートされません。
- 5.4.7 SecureConversationToken はサポートされます。
- 5.4.8 SamlToken はサポートされます。
- 5.4.9 RelToken はサポートされません。
- 5.4.10 HttpsToken は、TransportBinding アサーションに対してのみサポートされます。
- 5.4.11 KeyValueToken はサポートされます。
- 6.1 [Algorithm Suite] は、部分的にサポートされます。
  - Basic256、Basic192、Basic128 はサポートされます。
  - Basic256Rsa15、Basic192Rsa15、Basic128Rsa15 はサポートされます。
  - Basic256Sha256、Basic192Sha256、Basic128Sha256 はサポートされます。
  - Basic256Sha256Rsa15、Basic192Sha256Rsa15、Basic128Sha256Rsa15 はサポートされます。
  - TripleDes、TripleDesRsa15、TripleDesSha256、TripleDesSha256Rsa15 はサポートされません。
  - InclusiveC14N、SOAPNormalization10、STRTransform10 はサポートされません。
  - XPath10、XPathFilter20、AbsXPath はサポートされません。
- 6.2 [Timestamp] はサポートされます。
- 6.3 [Protection Order] はサポートされます。
- 6.4 [Signature Protection] はサポートされます。
- 6.5 [Token Protection] はサポートされます。
- 6.6 [Entire Header and Body Signatures] はサポートされます。
- 6.7 [Security Header Layout] はサポートされます。
- 7.1 6.1 ごとの AlgorithmSuite アサーション
- 7.2 6.7 ごとの Layout アサーション
- 7.3 TransportBinding は、HttpsToken でのみサポートされます。
- 7.4 SymmetricBinding はサポートされます。
- 7.5 AsymmetricBinding は、以下のようにサポートされます。
  - セクション 5.4 でサポートされるトークンに対してのみ。

  - セクション 6 のプロパティに対してのみ。

- 8.1 SupportingTokens アサーションはサポートされます。
- 8.2 SignedSupportingTokens アサーションはサポートされます。
- 8.3 EndorsingSupportingTokens アサーションはサポートされます。
- 8.4 SignedEndorsingSupportingTokens アサーションはサポートされます。
- 8.5 Encrypted SupportingTokens アサーションはサポートされます。
- 8.6 SignedEncrypted SupportingTokens アサーションはサポートされます。
- 8.7 EndorsingEncrypted SupportingTokens アサーションはサポートされます。
- 8.8 SignedEndorsingEncrypted SupportingTokens アサーションはサポートされます。
- 9.1 Wss10 アサーションは、以下の例外付きでサポートされます。
  - sp:MustSupportRefKeyIdentifier はサポートされます。
  - sp:MustSupportRefIssuerSerial はサポートされます。
  - sp:MustSupportRefExternalURI はサポートされません。
  - sp:MustSupportRefEmbeddedToken はサポートされません。
- 9.2 Wss11 アサーションは、以下の例外付きでサポートされます。
  - sp:MustSupportRefKeyIdentifier はサポートされます。
  - sp:MustSupportRefIssuerSerial はサポートされます。
  - sp:MustSupportRefExternalURI はサポートされません。
  - sp:MustSupportRefEmbeddedToken はサポートされません。
  - sp:MustSupportRefKeyThumbprint はサポートされます。
  - sp:MustSupportRefKeyEncryptedKey はサポートされます。
  - sp:RequireSignatureConfirmation はサポートされます。
- 10.1 Trust13 アサーションは、以下の例外付きでサポートされます。
  - sp:MustSupportClientChallenge はサポートされません。
  - sp:MustSupportServerChallenge はサポートされません。
  - sp:RequireClientEntropy はサポートされます。
  - sp:RequireServerEntropy はサポートされます。
  - sp:MustSupportIssuedTokens はサポートされません。今のところは無視されます。
  - sp:RequireRequestSecurityTokenCollection はサポートされません。
  - sp:RequireAppliesTo はサポートされません。
- Trust10 アサーション (http://specs.xmlsoap.org/ws/2005/07/securitypolicy/ws-securitypolicy.pdf を参照)

  注釈　Trust10 アサーションは簡易な形でのみサポートされています。InterSystems IRIS は、エラーのスローを回避するために Trust10 アサーションを Trust13 アサーションに変換します。

1.3.3 InterSystems IRIS における WS-SecureConversation のサポート

InterSystems IRIS は、以下のように WS-SecureConversation 1.3 (http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.3/ws-secureconversation.pdf) の一部をサポートしています。

- SCT Binding (WS-Trust の Issuance Binding に基づいて SecureConversationTokens を発行) および WS-Trust キャンセル・バインディングをサポートしています (http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.4/os/ws-secureconversation-1.4-spec-os.html の Canceling Contexts を参照)。
- 使用されるサービスが独自のセキュリティ・トークン・サービスとして動作する場合をサポートします。
- トークンに対する単純な要求および単純な応答のみをサポートします。

InterSystems IRIS は、WS-Trust 1.3 (http://docs.oasis-open.org/ws-sx/ws-trust/v1.3/ws-trust.pdf) の必要なサポート部分もサポートしています。WS-Trust のサポートは、WS-SecureConversation が必要とするバインディングに限定されます。これは一般的な実装ではありません。

1.3.4 InterSystems IRIS における WS-ReliableMessaging のサポート

InterSystems IRIS は、HTTP を介したメッセージを同期するために、WS-ReliableMessaging 1.1 および 1.2 をサポートします。応答メッセージでは、匿名の応答のみがサポートされています。同期メッセージのみがサポートされているので、キューは実行されません。

http://docs.oasis-open.org/ws-rx/wsrmp/200702/wsrmp-1.1-spec-os-01.html および http://docs.oasis-open.org/ws-rx/wsrm/200702 を参照してください。

2

設定およびその他の一般的なアクティビティ

参照のため、このトピックでは、Web サービスの保護に適用される共通のアクティビティについて説明します。

2.1 設定タスクの実行

SOAP セキュリティに関連するほとんどのタスクでは、まず以下のタスクを実行する必要があります。

- InterSystems IRIS が使用する信頼された証明書の用意
- InterSystems IRIS 資格情報セットの作成

これらのタスクは、"XML ツールの使用法" で説明しているいくつかのタスクの前提条件でもあります。

SSL/TLS 構成を作成する必要がある場合もあります。詳細は、インターシステムズの "TLS ガイド" を参照してください。

2.1.1 InterSystems IRIS が使用する信頼された証明書の用意

InterSystems IRIS は、信頼された証明書の独自のコレクションを使用して、着信 SOAP メッセージ (または XML ドキュメント) のユーザ証明書とシグニチャを検証します。また、発信 SOAP メッセージのコンテンツを暗号化するときまたは XML ドキュメントを暗号化するときにもこれらの証明書を使用します。このコレクションは、この InterSystems IRIS インストールのすべてのネームスペースで使用できます。このコレクションを作成するには、以下の 2 つのファイルを作成し、それらをシステム・マネージャのディレクトリに配置します。

- iris.cer ― ルート証明書、つまり PEM エンコード形式の信頼された CA X.509 証明書を格納します。InterSystems IRIS で何らかの WS-Policy または WS-Security 機能を使用する場合は、このファイルが必要です。
- iris.crl ― PEM エンコード形式の X.509 証明書取り消しリストを格納します。このファイルはオプションです。

特定の InterSystems IRIS 資格情報セットで使用する別のルート証明書を用意することができます。次のサブセクションを参照してください。

これらのファイルの作成に関する情報は、このドキュメントの対象ではありません。証明書の内容と証明書取り消しリストを規定する X.509 の詳細は、RFC5280 (https://www.ietf.org/rfc/rfc5280.txt) を参照してください。ファイル形式の一種である PEM エンコードの詳細は、RFC1421 (https://www.ietf.org/rfc/rfc1421.txt) を参照してください。

注意　これらの証明書は他のすべての証明書の信頼の基礎になるので、実稼働環境で使用する証明書は必ず信頼できるソースから取得してください。

このコレクションは、SSL では使用されません。

2.1.2 InterSystems IRIS 資格情報セットの作成と編集

このセクションでは、X.509 証明書のコンテナである InterSystems IRIS 資格情報セットを作成し、編集する方法について説明します。これには、以下の 2 つの一般的なシナリオがあります。

- 証明書を所有している。この場合、秘密鍵も持っています。以下のときにこの証明書を使用します。
  - 発信メッセージに署名するとき (秘密鍵ファイルもロードする場合)。
  - 所持している公開鍵で暗号化されたメッセージを解読するとき。
- 証明書を所有していない。この場合、証明書の所有者から証明書を取得していますが、秘密鍵ファイルを持っていません。以下のときにこの証明書を使用します。
  - 証明書の所有者に送信するメッセージを暗号化するとき。
  - 証明書の所有者が作成したデジタル・シグニチャを検証するとき。

2.1.2.1 InterSystems IRIS 資格情報セットの作成

InterSystems IRIS 資格情報セットを作成する手順は以下のとおりです。

1. 以下のファイルを取得します。
   - PEM でエンコードされた X.509 形式の個人の X.509 証明書

     これは、所有している証明書または SOAP メッセージの交換相手のエンティティから取得した証明書のいずれかとなります。
   - (オプション) PEM でエンコードされた PKCS#1 形式の関連付けられている秘密鍵

     これは、証明書を所有している場合にのみ適用されます。発信メッセージに署名しない場合、秘密鍵ファイルをロードする必要はありません。
   - (オプション) ルート証明書、つまりこの資格情報セットに使用する、PEM エンコード形式の信頼された CA X.509 証明書が含まれているファイル。

   これらのファイルの作成に関する情報は、このドキュメントの対象ではありません。
2. 管理ポータルで **[システム管理]→[セキュリティ]→[X.509 証明書]** を選択します。
3. **[新規資格情報の作成]** をクリックします。
4. 以下の値を指定します。
   - **[エイリアス]** ー この資格情報セットの識別に使用する文字列を指定します。これは一意にする必要があり、大文字と小文字は区別されます。このプロパティは必須です。
   - **[X.509 証明書を含むファイル]** ー **[参照]** をクリックして、証明書ファイルに移動します。このプロパティは必須です。
   - **[関連づけられた秘密鍵を含むファイル]** ー **[参照]** をクリックして、証明書ファイルに移動します。
   - **[秘密鍵パスワード]** および **[秘密鍵パスワード(確認)]** ー 秘密鍵のパスワードを指定します。パスワードを指定しない場合は、資格情報セットを取得するときにパスワードを指定する必要があります。

     これらのフィールドは、**[関連づけられた秘密鍵を含むファイル]** の値を指定した場合にのみ表示されます。
   - **[信頼された証明書機関 X.509 証明書を含むファイル]** ー この資格情報セットにより信頼される CA の X.509 証明書のパスおよびファイル名。証明書は PEM 形式にする必要があります。このパスは、絶対パス、または管理者ディレクトリを基準とした相対パスで指定します。

例外として、この資格情報セットを使用する場合、InterSystems IRIS は、前述したように、iris.cer ではなく、この信頼された証明書を使用します。この例外は、デジタル・シグニチャにメッセージ内のバイナリ・セキュリティ・トークンへの直接参照が含まれている場合です。この場合、シグニチャを検証するために必要な公開鍵がメッセージに含まれているため、InterSystems IRIS はこの資格情報セットを検索しません。InterSystems IRIS は代わりに、iris.cer に含まれている信頼された証明書を使用します。

- **[許可されたユーザ]** ― この資格情報セットを使用できる InterSystems IRIS ユーザのコンマ区切りリストを指定します。このプロパティが NULL の場合、あらゆるユーザがこの資格情報セットを使用できます。
- **[Intended peer(s)]** ― この資格情報セットを使用できるシステムの DNS 名の、コンマ区切りのリストを指定します。この資格情報セットに対する相手の有効性は、ユーザのコードで資格情報オブジェクトの CheckPeerName() メソッドを使用して確認する必要があります。

5. **[保存]** をクリックします。

   この操作によって、証明書ファイルと秘密鍵ファイル (存在する場合) の両方がデータベースにコピーされます。**[信頼された証明書機関 X.509 証明書を含むファイル]** を指定した場合、そのファイルはデータベースにコピーされません。

管理ポータルを使用するのではなく、%SYS.X509Credentials クラスのメソッドを使用できます。以下はその例です。

ObjectScript

```
Set credset=##class(%SYS.X509Credentials).%New()
Set credset.Alias="MyCred"
Do credset.LoadCertificate("c:\mycertbase64.cer")
Do credset.LoadPrivateKey("c:\mycertbase64.key")
Set sc=credset.Save()
If sc Do $system.Status.DisplayError(sc)
```

注釈　このデータにアクセスするために、標準のオブジェクトと SQL メソッドは使用しないでください。Save()、Delete()、および LoadPrivateKey() メソッドの使用には、**%Admin_Secure:USE** 特権が必要です。

詳細は、%SYS.X509Credentials のクラス・リファレンスを参照してください。

2.1.2.2 InterSystems IRIS 資格情報セットの編集

InterSystems IRIS 資格情報セットを作成したら、次のように編集できます。

1. 管理ポータルで **[システム管理]→[セキュリティ]→[X.509 証明書]** を選択します。
2. 資格情報セットのテーブルでは、エイリアスの列にある値が識別子として機能します。編集する資格情報セットで **[編集]** をクリックします。
3. 必要に応じて編集します。これらのフィールドの詳細は、前のセクションを参照してください。
4. **[保存]** をクリックすると、変更内容が保存されます。

資格情報セットのエイリアスと証明書はいずれも変更できません。資格情報に関連付けた秘密鍵の追加、変更、および削除も不可能です。このような変更を加えるには、新しい資格情報セットを作成します。

2.2 プログラムによる資格情報セットの取得

暗号化または署名を実行する際は、使用する証明書を指定する必要があります。そのためには、InterSystems IRIS 資格情報セットを選択します。

ウィザードを使用してポリシーを作成する場合、ウィザード内で資格情報セットを選択するか、または Web サービスあるいは Web クライアント内でプログラムによって資格情報セットを取得して使用できます。WS-Security ヘッダを手動で作成する場合は、資格情報セットをプログラムで取得して使用する必要があります。

参考のため、このセクションでは、以下の一般的なアクティビティについて説明します。

- 格納された資格情報セットを取得する方法
- 着信メッセージから資格情報セットを取得する方法

2.2.1 格納された資格情報セットの取得

%SYS.X509Credentials のインスタンスを取得するには、GetByAlias() クラス・メソッドを呼び出します。このメソッドは、証明書およびその他の情報を格納している InterSystems IRIS 資格情報セットを返します。以下はその例です。

ObjectScript

```
set credset=##class(%SYS.X509Credentials).GetByAlias(alias,password)
```

- alias は、証明書のエイリアスです。
- pwd は、秘密鍵パスワードです。これは、証明書を所有している場合にのみ適用されます。これは、関連付けられている秘密鍵が暗号化されていて、その秘密鍵のファイルをロードしたときにパスワードをロードしなかった場合にのみ必要です。

  証明書を所有していない場合、いずれの形式でも秘密鍵にはアクセスできません。

  パスワード引数を指定しない場合、%SYS.X509Credentials インスタンスは秘密鍵にアクセスできないので、暗号化のみで使用できます。

このメソッドを実行するには、当該の資格情報セットの OwnerList に記載されたユーザとしてログインするか、OwnerList が NULL である必要があります。

暗号化に証明書を使用する場合、FindByField()、GetBySubjectKeyIdentifier()、GetByThumbprint() など、他のクラス・メソッドを使用して、InterSystems IRIS 資格情報セットを取得できます。%SYS.X509Credentials については、クラス・ドキュメントを参照してください。GetByAlias() は、秘密鍵へのアクセスを提供する唯一のメソッドであるため、このクラスで署名用の証明書を取得する際に使用できる唯一のメソッドです。

2.2.2 着信メッセージからの証明書の取得

デジタル署名が行われた SOAP メッセージを受信する場合、%SYS.X509Credentials のインスタンス内において関連付けられた証明書を使用できます。その証明書を取得できます。そのためには、以下の操作を実行します。

1. まず、Web サービスまたは Web クライアントの SecurityIn プロパティを使用して WS-Security ヘッダ要素にアクセスします。これにより %SOAP.Security.Header のインスタンスが返されます。
2. 次のいずれかの操作を行います。
   - %SOAP.Security.Header インスタンスの中で、セキュリティ・ヘッダ要素の最初の <Signature> 要素を参照する Signature プロパティにアクセスします。
   - %SOAP.Security.Header インスタンスの FindElement() メソッドを使用して、%SOAP.Security.Header インスタンスの最初の <Signature> 要素にアクセスします。

   どちらの場合も、デジタル・シグニチャを含む %XML.Security.Signature のインスタンスが返されます。
3. シグニチャ・オブジェクトの X509Credentials プロパティにアクセスします。
4. 返されたオブジェクトのタイプが %SYS.X509Credentials のインスタンスかどうかを確認します。

ObjectScript

```
if $CLASSNAME(credset)'="%SYS.X509Credentials" {set credset=""}
```

着信メッセージに署名済みの SAML アサーションが含まれている場合、X509Credentials プロパティは、他のクラスのインスタンスであるため、%SYS.X509Credentials インスタンスへのアクセスには使用できません。

以下はその例です。

ObjectScript

```
set credset=..SecurityIn.Signature.X509Credentials
if $CLASSNAME(credset)'="%SYS.X509Credentials" {set credset=""}
//if credset is not null, then use it...
```

2.3 クライアントで SSL/TLS 構成を使用するように指定する方法

Web サービスが HTTP over SSL/TLS (HTTPS) の使用を必要とする場合、Web クライアントは、適切な InterSystems IRIS SSL/TLS 構成を使用する必要があります。

ウィザードを使用してポリシーを作成する場合、ウィザード内で SSL/TLS 構成を選択するか、Web サービスまたは Web クライアント内で使用する構成をプログラムによって指定できます。WS-Security ヘッダを手動で作成する場合、使用する構成をプログラムで指定する必要があります。

使用する SSL/TLS 構成を指定するには、Web クライアントの SSLConfiguration プロパティを SSL/TLS 構成名に等しくなるように設定します。以下はその例です。

ObjectScript

```
set client=##class(proxyclient.classname).%New()
set client.SSLConfiguration="mysslconfig"
//invoke web method of client
```

プロキシ・サーバ経由でクライアントを接続している場合は、Web クライアントで HttpProxySSLConnect プロパティを 1 に設定する必要もあります。

3
ポリシーの作成と使用

このトピックでは、InterSystems IRIS で WS-Policy サポートを使用する方法を説明します。WS-Policy により、使用する WS-Security ヘッダまたは必要な WS-Security ヘッダを指定できます。また、WS-Addressing ヘッダおよび MTOM の使用を指定することもできます ("Web サービスおよび Web クライアントの作成" を参照)。Web サービスまたは Web クライアントを直接編集するのではなく、個別のクラスでポリシーを作成します。ほとんどの場合、低レベルのプログラミングは必要ありません。

3.1 概要

InterSystems IRIS では、Web サービスまたは Web クライアントのポリシー (またはポリシーのコレクション) は、別々の構成クラス (%SOAP.Configuration のサブクラス) に含まれています。ポリシーは、クラスがコンパイルされたときに有効になります。

Web サービスの WSDL が既に定義されている場合は、構成クラスを生成できます。最初に Web サービスを作成する場合は、Web サービス/クライアント構成ウィザードを使用して、事前定義されたポリシーの選択および構成を行い、Web サービスに適用できます。クラスを手動で作成することもできます。

通常、コーディングの必要はありません。ただし、場合によってはその要素をポリシーにハードコードするのではなく、詳細をプログラムで指定してもかまいません。

3.1.1 構成クラスの影響

構成クラスをコンパイルすると、Web サービスまたはクライアントの以降の動作は次のような影響を受けます。

- Web サービスまたはクライアントでは、ポリシーの詳細に従って、発信メッセージに追加のヘッダ要素が含まれます。
- Web サービスまたはクライアントは、ポリシーに基づいて着信 SOAP メッセージを検証します。これには、必要に応じた着信メッセージの解読も含まれます。
- Web サービスまたはクライアントは、オプションで、発信メッセージを必要に応じて暗号化します。
- Web サービスでは、WSDL が自動的に影響を受けます。具体的には、<wsp:Policy> 要素が追加され、ネームスペース宣言に次の行が含まれるようになります。

```
xmlns:wsp="http://www.w3.org/ns/ws-policy"
```

重要　構成クラスが複数のネームスペースにマップされている場合は、これらの各ネームスペースでコンパイルする必要があります。

3.1.2 WS-Security、WS-Addressing、および MTOM サポートとの関係

WS-Policy に対する InterSystems IRIS のサポートは、WS-Security、WS-Addressing、および MTOM に対する InterSystems IRIS のサポートに基づいています。以下の点に注意してください。

- ポリシーにセキュリティ・ポリシーが含まれていない場合、InterSystems IRIS は、Web サービスまたは Web クライアントの SecurityOut プロパティを使用します (Web サービスまたは Web クライアントにセキュリティ・ヘッダ要素を手動で追加するには、別途説明しているように、SecurityOut プロパティに追加します)。
- ポリシーにセキュリティ ポリシーが含まれている場合、InterSystems IRIS は、そのポリシーに関連する要素を除いて、Web サービスまたは Web クライアントの SecurityOut プロパティを無視します。

  例えば、X.509 相互証明書セキュリティ・ポリシーを使用する場合、使用する InterSystems IRIS 資格情報セットをポリシー内で直接指定することも、%SYS.X509Credentials のインスタンスを作成し、バイナリ・セキュリティ・トークンに格納して SecurityOut プロパティに追加することもできます。資格情報セットをポリシー内で直接指定しない場合、InterSystems IRIS は SecurityOut プロパティからバイナリ・セキュリティ・トークンを取得し、それを使用します。ただし、SecurityOut プロパティの他の要素はこのシナリオには適用されないため、InterSystems IRIS はそれらを無視します。
- ポリシーで WS-Addressing を必要とする場合、InterSystems IRIS は WSADDRESSING クラス・パラメータを無視します。

  ただし、AddressingOut プロパティが設定されていると、InterSystems IRIS は、指定の WS-Addressing ヘッダを使用します。それ以外の場合、WS-Addressing ヘッダの既定のセットが使用されます。
- ポリシーで MTOM を必要とする場合、InterSystems IRIS は MTOMREQUIRED クラス・パラメータおよび MTOMRequired プロパティを無視します。

3.1.3 Web サービスと Web クライアントの関係

Web サービスにポリシーを添付する場合、すべてのクライアントは、そのポリシーに従うことが可能でなければなりません。Web サービス・ポリシーに代替ポリシーが何も含まれていない場合、クライアントには、Web サービスと同じポリシーが必要です。このとき、必要に応じて、サーバ側の証明書の代わりにクライアント側の証明書を使用します。

同様に、Web クライアントにポリシーを添付する場合、サービスは、そのポリシーに従うことが可能でなければなりません。

実際には、InterSystems IRIS でサービスとクライアントの両方を作成する場合、最も簡単な手順は次のとおりです。

1. Web サービス・クラスを作成します。
2. Web サービス構成クラスを、サービス・ポリシーと共に作成します。
3. SOAP ウィザードを使用して、クライアント構成クラスなどのクライアント・クラスを生成します。

   これを実行した後、生成されたクライアント・クラスを検証し、必要に応じて変更を加えます。

   通常、このためにラッパ・クラスも作成します。

   これらのタスクについては、"Web サービスおよび Web クライアントの作成" を参照してください。
4. 生成された構成クラスを検証し、必要に応じて変更を加えます。"生成されたポリシーの編集" を参照してください。

構成クラスの詳細は、"WS-Policy 構成クラスの詳細" を参照してください。

3.2 ポリシーの作成と添付

ポリシーを作成し、これを Web サービスまたは Web クライアントにアタッチするには、構成クラスを作成してコンパイルします。このクラスを作成するには、以下のような方法があります。

- Web サービス/クライアント構成ウィザードの使用。このオプションは、Web サービスまたは Web クライアントのクラスが既に存在している場合に適用されます。
- SOAP ウィザードの使用このオプションは、WSDL から開始する場合に適用されます。

  ウィザードで構成クラスが生成されるのは、WSDL に WS-Policy 要素が含まれている場合のみです。

  詳細は、"Web サービスおよび Web クライアントの作成" を参照してください。
- GeneratePolicyFromWSDL() メソッドの使用によって、WSDL から構成クラスのみを生成します。このオプションは、Web サービスまたは Web クライアントのクラスが既に存在していて、これを再生成しない場合に適用されます。
- 既存の Web サービスまたは Web クライアントの構成クラスを手動で作成します。詳細は、次のトピックを参照してください。

WSDL からポリシー・クラスを生成する場合、次のセクションで説明するように、ポリシー・クラスの編集が必要になる場合があります。

3.2.1 Web サービス/クライアント構成ウィザードの使用法

スタジオの Web サービス/クライアント構成ウィザードを使用して、ポリシーを作成および添付することができます。このウィザードを使用する手順は次のとおりです。

1. **[ファイル]→[新規作成]** をクリックします。
2. **[一般]** タブで、**[Web サービス/クライアント構成]** をクリックします。
3. **[OK]** をクリックします。

   スタジオにダイアログ・ボックスが表示されます。
4. ポリシーの適用対象とする Web サービスまたは Web クライアントを選択します。ウィザードには、コンパイルされるサービスおよびクライアントのみが一覧表示されます。

   そのためには、必要に応じて **[サービス]** または **[クライアント]** ドロップダウン・メニューをクリックし、Web サービス・クラスまたは Web クライアント・クラスをクリックします。
5. **[次へ]** をクリックします。

   スタジオに別のダイアログ・ボックスが表示されます。
6. オプションで、このダイアログ・ボックスの 2 番目のフィールドに表示されている構成クラス名を編集します。

   注釈　このクラスが既に存在する場合、ウィザードは、既存のコンテンツを無視し、それを上書きします。ウィザードの最後に到達しない限り、新しいクラスは作成されません。

   既定の構成クラス名は、Web サービスまたはクライアント・クラス名に `Config` を追加したものです。
7. このページの残りの詳細は、"セキュリティ・ポリシーの説明" および "ポリシー・オプションのリファレンス" を参照してください。
8. **[完了]** をクリックします。

   ウィザードにより、クラスが作成され、保存されます。

9. 生成された構成クラスを確認します。構成クラスには、発信メッセージにタイムスタンプを追加するポリシー式も含まれています。

```
<sp:IncludeTimestamp/>
```

10. オプションで、構成クラスを編集し、再保存します。

    例えば、代替ポリシーを追加したり、ウィザードによって作成されたポリシーを調整したりするには、この操作を行います。

クラスは、自動的にはコンパイルされません。また、ポリシーは、クラスがコンパイルされないと有効になりません。

Tipヒント 構成クラスをコンパイルした後、それを無効にする場合は、XData ブロックをコメントアウトし、再コンパイルします。

3.2.2 WSDL からのポリシーの生成

クライアント・クラスは既にあるものの、対応する構成クラスがない場合があります。これは、WSDL からクライアント・クラスを生成し、その後 WS-Policy 情報を含むように WSDL が変更された場合などに発生します。このような場合、以下のように、%SOAP.WSDL.Reader のユーティリティ・メソッドを使用して、構成クラスのみを生成できます。

1. %SOAP.WSDL.Reader のインスタンスを作成します。
2. そのインスタンスのプロパティを規定どおりに設定します。%SOAP.WSDL.Reader については、クラス・ドキュメントを参照してください。

   Process() メソッドは使用しないでください。
3. インスタンスの GeneratePolicyFromWSDL() メソッドを呼び出します。

   このメソッドには、以下のシグニチャがあります。

```
method GeneratePolicyFromWSDL(wsdlURL As %String,
    clientWebServiceClass As %String,
    policyConfigClass As %String) as %Status
```

   以下はその説明です。

   - wsdlURL は、ポリシーを含む WSDL の URL です。WSDL が 1 つのポートのみを指定することが前提となっています。
   - clientWebServiceClass は、Web クライアント・クラスの名前です。この Web クライアントが指定の WSDL と一致することを確認する必要があります。
   - policyConfigClass は、作成される構成クラスの名前です。

Web サービスの WSDL が指定するポリシーを含む Web サービス・クライアントの構成クラスが作成 (または上書き) されます。WSDL にポリシーがない場合は、空の構成クラスが作成されます。インスタンスの CompileClasses プロパティが 1 の場合は、この構成クラスがコンパイルされます。

3.3 生成されたポリシーの編集

構成クラスを WSDL から生成し、WSDL が InterSystems IRIS のこのインスタンスの外側にある場合、構成クラスを編集して、使用する証明書および SSL/TLS 構成に関する情報を組み込む必要があります。または、この情報を実行時に指定できます。

また、[**保護セッション（安全な通信）を確立する**] を選択した場合は、安全な通信の存続時間のオプションを指定するポリシーを編集できます。

以下のテーブルに詳細を示します。

生成されたポリシーに含まれている要素	以下のことを実行します。
<sp:HttpsToken>	クライアントに添付されたポリシーの場合、次のいずれかを実行します。 ・ "InterSystems 拡張属性の追加" の説明に従って、この要素を編集します。 ・ "クライアントで SSL/TLS 構成を使用するように指定する方法" の説明に従って、SSL/TLS 構成の名前を指定します。 サービスに添付されたポリシーの場合、変更は必要ありません。
<sp:InitiatorToken>	クライアントに添付されたポリシーの場合、次のいずれかを実行します。 ・ "InterSystems 拡張属性の追加" の説明に従って、この要素にある <sp:X509Token> 要素を編集します。 ・ "実行時の証明書の追加" の説明に従って、資格情報セットを取得し、そこに格納されている証明書を追加します。 どちらの場合も、これは、クライアントが所有する資格情報セットである必要があります。 サービスに添付されたポリシーの場合、変更は必要ありません。
<sp:RecipientToken>	以下のいずれかを行います。 ・ "InterSystems 拡張属性の追加" の説明に従って、この要素にある <sp:X509Token> 要素を編集します。 ・ "実行時の証明書の追加" の説明に従って、資格情報セットを取得し、そこに格納されている証明書を追加します。 どちらの場合も、これは、サービスが所有する資格情報セットである必要があります。
<sp:SecureConversationToken>	必要に応じて、"InterSystems 拡張属性の追加" の説明に従って、cfg:Lifetime 属性を追加します。既定の存続時間は 5 分です。

3.4 セキュリティ・ポリシーの説明

ウィザードの主な目的は、構成可能なセキュリティ・ポリシーを用意することです。[**セキュリティ・ポリシー**] の選択肢は、以下のとおりです。

- セキュリティ・ポリシーなし
- SSL/TLS 接続セキュリティ
- SSL/TLS 経由のユーザ名認証
- SSL/TLS 経由の X.509 証明書認証
- 対称鍵による認証

- 保証証明書を使用した対称鍵
- X.509 相互証明書セキュリティ
- SSL/TLS 経由の SAML 承認
- X.509 証明書を持つ SAML

以下のセクションは、すべてのポリシー・オプションを示しています。

3.4.1 SSL/TLS 接続セキュリティ

このポリシーは、Web クライアントと Web サービス間で HTTP over SSL/TLS (HTTPS) を使用することを要求します。これにより、データ・ストリームの機密性と整合性、サーバの認証、およびクライアントのオプションの認証が提供されます。

3.4.2 SSL/TLS 経由のユーザ名認証

このポリシーは、クライアントに、(ユーザ名およびパスワードと共に) `<UsernameToken>` を送信することを要求します。また、HTTP over SSL/TLS (HTTPS) も要求します。これにより、データ・ストリームの機密性と整合性の維持、サーバの認証、およびクライアントのオプションの認証が可能となります。

実行時、Web クライアントは、既定のパスワード・タイプを使用してユーザ名トークンを作成および追加する必要があります。"ユーザ名トークンの追加" を参照してください。

3.4.3 SSL/TLS 経由の X.509 証明書認証

このポリシーは、クライアントに、署名済みの本文とタイムスタンプ、およびシグニチャを検証できる X.509 証明書と共にメッセージを送信することを要求します。WS-Addressing ヘッダが含まれている場合は、それも署名されます。また、HTTP over SSL/TLS (HTTPS) も要求します。これにより、データ・ストリームの機密性と整合性の維持、サーバの認証、およびクライアントのオプションの認証が可能となります。

3.4.4 対称鍵による認証

このポリシーは、メッセージの署名と暗号化の両方に使用する、1 つの共有秘密鍵を必要とします。この対称鍵は、実行時に生成され、サービスの証明書の公開鍵を使用して暗号化されます。

サービスは、認証のために、暗号化されたユーザ名および既定のパスワード・タイプのパスワードを必要に応じて要求することができます。このオプションを選択した場合、クライアントは、"ユーザ名トークンの追加" の説明に従って、実行時に `<UsernameToken>` を追加する必要があります。手動で `<UsernameToken>` を暗号化する必要はありません。InterSystems IRIS によって自動的に暗号化されます。

3.4.5 保証証明書を使用した対称鍵

このポリシーは、メッセージの署名と暗号化の両方に使用する、1 つの共有秘密鍵を必要とします。この対称鍵は、実行時に生成され、サービスの証明書の公開鍵を使用して暗号化されます。

サービスは、認証のために、暗号化されたユーザ名および既定のパスワード・タイプのパスワードを必要に応じて要求することができます。このオプションを選択した場合、クライアントは、"ユーザ名トークンの追加" の説明に従って、実行時に `<UsernameToken>` を追加する必要があります。手動で `<UsernameToken>` を暗号化する必要はありません。InterSystems IRIS によって自動的に暗号化されます。

このメカニズムは、クライアント保証証明書を使用して、メッセージ・シグニチャに関連付けられたトークンを拡張します。

3.4.6 X.509 相互証明書セキュリティ

このポリシーは、すべての相手に、メッセージ本文とタイムスタンプ、および WS-Addressing ヘッダ（含まれている場合）に署名することを要求します。また、オプションで、相手の証明書の公開鍵を使用してメッセージ本文を暗号化します。

サービスは、認証のために、暗号化されたユーザ名および既定のパスワード・タイプのパスワードを必要に応じて要求することができます。このオプションを選択した場合、クライアントは、"ユーザ名トークンの追加" の説明に従って、実行時に `<UsernameToken>` を追加する必要があります。手動で `<UsernameToken>` を暗号化する必要はありません。InterSystems IRIS によって自動的に暗号化されます。

3.4.7 SSL/TLS 経由の SAML 承認

このポリシーは、クライアントに、X.509 証明書または公開鍵を含む SAML トークンを送信することを要求します。対応する秘密鍵は、メッセージ本文とタイムスタンプ、および WS-Addressing ヘッダ（含まれている場合）に署名します。また、HTTP over SSL/TLS（HTTPS）も要求します。これにより、データ・ストリームの機密性と整合性の維持、サーバの認証、およびクライアントのオプションの認証が可能となります。

3.4.8 X.509 証明書を持つ SAML

このポリシーは、SAML トークンを送信することをクライアントに要求します。このポリシーはまた、メッセージ本文とタイムスタンプ、および WS-Addressing ヘッダ（含まれている場合）に署名します。また、オプションで、相手の証明書の公開鍵を使用してメッセージ本文を暗号化します。

3.5 ポリシー・オプションのリファレンス

事前定義のポリシーには、同じオプションが多数あります。以下のリストは、すべてのオプションの詳細をウィザードに表示される順序に従って示しています。

保護セッション（安全な通信）を確立する

該当する場所：すべてのセキュリティ・ポリシー。

このオプションを選択した場合、Web サービスおよび Web クライアントは、共通の秘密セキュリティ・コンテキストを確立して使用します。そして両者で同じ対称鍵を生成し、これを署名、暗号化、シグニチャ検証、および解読に使用できます。

派生キーを要求する

該当する場所：すべてのセキュリティ・ポリシー。

このオプションは、前のオプションも選択した場合にのみ適用されます。**[派生キーを要求する]** を選択した場合、元のセッション・キーではなく、派生キーがメッセージで使用されます。

信頼性の高いメッセージ配信

該当する場所：すべてのセキュリティ・ポリシー。

このオプションは、メッセージを送信する場合に WS-ReliableMessaging プロトコルを使用する必要があることを指定します。このプロトコルを使用すると、ソフトウェア・コンポーネント、システム、またはネットワークで障害が発生してもSOAP メッセージが確実に配信されます。

注釈　InterSystems IRIS Web サービスでパラメータを指定することで、WS-ReliableMessaging の動作を調整できます。"Web サービスで信頼性の高いメッセージングを処理する方法を制御する" を参照してください。

SSL 構成

該当する場所：SSL/TLS を使用するすべてのポリシー。クライアントのみに適用されます。

クライアントが使用する SSL/TLS 構成。

すべての適用可能なポリシーにおいて、ウィザード内で SSL/TLS 構成を選択するか (その選択をハードコード)、またはプログラムで選択し、Web サービスまたは Web クライアントに追加できます。"SSL/TLS 構成を使用するように指定する方法" を参照してください。

SSL/TLS 接続でクライアント証明書を要求する

該当する場所：SSL/TLS を使用するすべてのポリシー。

SSL/TLS 接続でクライアントが自らを認証することを要求する場合、オプションでこれを選択します。[SSL 構成] の説明を参照してください。

SOAP 本文の暗号化

該当する場所：SSL/TLS を使用しないすべてのポリシー。

(オプション) 相手の証明書の公開鍵を使用して SOAP 本文を暗号化します。

署名の前に暗号化

該当する場所：SSL/TLS を使用しないすべてのポリシー。

(オプション) 署名の前にメッセージを暗号化することを要求します。このオプションを選択しない場合、メッセージは、署名されてから暗号化されます。

トークンの保護 – シグニチャは、シグニチャの生成に使用されたトークンを対象とする必要があります

該当する場所：SSL/TLS を使用しないすべてのポリシー。

(オプション) 関連付けられた秘密鍵がメッセージに署名する証明書を送信するバイナリ・セキュリティ・トークンにメッセージ・シグニチャを適用するよう要求します。

X.509 証明書

該当する場所：SSL/TLS 経由の X.509 証明書認証。クライアントのみに適用されます。

メッセージに署名するときにクライアントが使用する InterSystems IRIS 資格情報セット。シグニチャは、関連付けられている証明書で秘密鍵を使用します。サブセクション "資格情報セット" を参照してください。

暗号化されたユーザ名トークンを含める

該当する場所：複数のポリシー。

(オプション) `<UsernameToken>` を含む暗号化要素を送信することをクライアントに要求します。

このオプションを選択すると、"ユーザ名トークンの追加" の説明に従って、実行時に `<UsernameToken>` を追加する必要があります。手動で `<UsernameToken>` を暗号化する必要はありません。InterSystems IRIS によって自動的に暗号化されます。

保護トークン

該当する場所：対称鍵を使用するポリシー。クライアントのみに適用されます。

対称鍵を生成するためにクライアントが使用する InterSystems IRIS 資格情報セット。これは、サービスの証明書である必要があります。対称鍵は、証明書の公開鍵を使用して生成されます。"資格情報セット" を参照してください。

保証トークン

該当する場所：保証証明書を使用した対称鍵。クライアントのみに適用されます。

保護トークンに署名するためにクライアントが使用する InterSystems IRIS 資格情報セット。これは、クライアントの証明書である必要があります。サブセクション "資格情報セット" を参照してください。

イニシエータ・トークン

該当する場所：X.509 相互証明書セキュリティおよび X.509 証明書を持つ SAML。クライアントのみに適用されます。

Web クライアントが使用する InterSystems IRIS 資格情報セット。クライアントは、メッセージの署名に、関連付けられている秘密鍵を使用し、イニシエータ・トークンをサービスに送信して、サービスを有効化し、シグニチャの検証と応答の暗号化を行います。サブセクション "資格情報セット" を参照してください。

受信者トークン

該当する場所：X.509 相互証明書セキュリティおよび X.509 証明書を持つ SAML。

Web サービスが使用する InterSystems IRIS 資格情報セット。クライアントは、受信者トークン内の証明書の公開鍵を使用して、発信メッセージ本文を暗号化します。サービスは、関連付けられた秘密鍵をメッセージの署名に使用します。サブセクション "資格情報セット" を参照してください。

アルゴリズム・スイート

該当する場所：すべてのセキュリティ・ポリシー。

Web サーバおよび Web クライアントが使用するアルゴリズム・スイート。詳細は、"WS-SecurityPolicy 仕様" のセクション 6.1 を参照してください。

厳密なセキュリティ・ヘッダ・レイアウト

該当する場所：すべてのセキュリティ・ポリシー。

(オプション) セキュリティ・ヘッダ要素の厳密なレイアウトを強制します。

WS-Addressing の有効化

該当する場所：すべてのポリシー。

このオプションを選択すると、Web サービスまたは Web クライアントは、発信メッセージに WS-Addressing ヘッダ要素を含め、着信メッセージに WS-Addressing ヘッダ要素があると想定します。

[WS-Addressing 有効] を使用した場合、InterSystems IRIS は、WS-Addressing ヘッダ要素の既定セットを使用します。含まれる要素の詳細は、"WS-Addressing ヘッダ要素の追加と使用" を参照してください。

[WS-Addressing 有効] を選択しても、"WS-Addressing ヘッダ要素の追加と使用" の説明に従って、WS-Addressing ヘッダ要素を手動で作成し、これをアタッチできます。このようにした場合、Web サービスまたは Web クライアントは、既定のヘッダ要素ではなく作成したヘッダ要素を使用します。

バイナリ・データの転送の最適化 (MTOM)

該当する場所：すべてのポリシー。

このオプションを選択すると、Web サービスまたは Web クライアントは、発信メッセージに MTOM パッケージを使用し、着信メッセージに MTOM パッケージがあると想定します。

3.5.1 資格情報セット

多くのポリシーにおいて、以下のように InterSystems IRIS 資格情報セットを選択できます。

- [X.509 証明書] ドロップダウン・リストから資格情報セットを選択します。
- 証明書のフィールドの値を指定して、資格情報セットを選択します。そのためには、[フィールド] ドロップダウン・リストからフィールドを選択し、入力ボックスに値を入力します。

  使用できるフィールドは次のとおりです。

  - Alias
  - SubjectKeyIdentifier
  - Thumbprint
  - SerialNumber
  - IssuerDN
  - IssuerName ― IssuerDN フィールドの検索文字列として動作します。システムは、IssuerDN フィールドに指定の文字列が含まれている最初の資格情報セットを選択します (一方、IssuerDN を使用する場合は、正確に一致しなければなりません)。
  - SubjectDN
  - SubjectName ― SubjectDN フィールドの検索文字列として動作します。システムは、SubjectDN フィールドに指定の文字列が含まれている最初の資格情報セットを選択します (一方、SubjectDN を使用する場合は、正確に一致しなければなりません)。

または、プログラムで資格情報セットを選択できます。"プログラムによる資格情報セットの取得" を参照してください。この場合、"実行時の証明書の追加" の説明に従って、証明書をバイナリ・セキュリティ・トークンでパッケージ化し、これを発信メッセージに追加する必要もあります。

3.6 実行時の証明書の追加

Web サービスまたは Web クライアントが、プログラムで証明書を選択して含める必要がある場合は、以下の手順を実行します。

1. "プログラムによる資格情報セットの取得" の説明に従って、%SYS.X509Credentials のインスタンスを取得します。

   以下はその例です。

   ObjectScript

   ```
   set credset=##class(%SYS.X509Credentials).GetByAlias(alias,password)
   ```

   または以下のようにします。

   ObjectScript

   ```
   set credset=..SecurityIn.Signature.X509Credentials
   ```

2. この資格情報セットの証明書を含む %SOAP.Security.BinarySecurityToken のインスタンスを作成します。以下はその例です。

ObjectScript

```
set bst=##class(%SOAP.Security.BinarySecurityToken).CreateX509Token(credset)
```

credentials は、前の手順で取得した資格情報セットです。

これにより、シリアル化された Base 64 のエンコード形式の証明書を保持する <BinarySecurityToken> 要素を表すオブジェクトが返されます。

3. Web クライアントまたは Web サービスの SecurityOut プロパティの AddSecurityElement() メソッドを呼び出します。このメソッドの引数として、前に作成したバイナリ・セキュリティ・トークンを使用します。以下はその例です。

ObjectScript

```
do ..SecurityOut.AddSecurityElement(bst)
```

重要　2 つのバイナリ・セキュリティ・トークン (暗号化用と署名用) が必要な場合があります。[**署名の前に暗号化**] オプションを選択したかどうかに応じて、これらを適切な順序で追加してください。ポリシーがメッセージを暗号化してからこれに署名する場合、暗号化に使用するバイナリ・セキュリティ・トークンを追加してから署名に使用するバイナリ・セキュリティ・トークンを追加してください。逆に、ポリシーが署名してから暗号化する場合は、署名に使用するバイナリ・セキュリティ・トークンを先にする必要があります。

以下に、Web サービスの Web メソッドの例を示します。

ObjectScript

```
//get credentials
set x509alias = "something"
set pwd = "password"
set credset = ##class(%SYS.X509Credentials).GetByAlias(x509alias,pwd)

//get certificate and add it as binary security token
set cert = ##class(%SOAP.Security.BinarySecurityToken).CreateX509Token(credset)
do ..SecurityOut.AddSecurityElement(cert)
```

Web クライアントの場合、通常はプロキシ・クライアントを編集しないため、コードは多少異なります。

ObjectScript

```
set client=##class(proxyclient.classname).%New()
//get credentials
set x509alias = "something"
set pwd = "password"
set credset = ##class(%SYS.X509Credentials).GetByAlias(x509alias,pwd)

//get certificate and add it as binary security token
set cert = ##class(%SOAP.Security.BinarySecurityToken).CreateX509Token(credset)
do client.SecurityOut.AddSecurityElement(cert)
//invoke web method of client
```

3.7 実行時のポリシーの指定

InterSystems IRIS Web クライアントでは、実行時に使用するポリシーを指定できます。これにより、ポリシー構成クラスがオーバーライドされます。実行時にポリシーを指定するには、Web クライアント・インスタンスの PolicyConfiguration プロパティを設定します。この値は、以下の形式にする必要があります。

```
Configuration class name:Configuration name
```

ここで、Configuration class name は、このトピックで前述したように、ポリシー構成クラスの完全なパッケージおよびクラス名であり、Configuration name は、そのクラス内のポリシーの <configuration> 要素の name 属性です。

3.8 サポートされていないポリシーに対するコンパイル・エラーの抑制

既定では、構成クラスのコンパイル時に、InterSystems IRIS でサポートされていないポリシー式が構成に含まれていると、InterSystems IRIS はエラーを発行します。そのようなエラーを抑制するには、構成クラスで次のように設定します。

Class Member

```
Parameter REPORTANYERROR=0;
```

SOAP ウィザードを使用して WSDL から Web クライアントまたは Web サービスを生成するときに、InterSystems IRIS によって構成クラスも生成される場合は、そのクラスにこのパラメータ設定が含まれます。

サポートされている代替ポリシーが 1 つあれば、サポートされていない代替ポリシーは無視できます。

4

WS-Policy 構成クラスの詳細

参考のため、このトピックでは、InterSystems IRIS が WS-Policy 情報を格納するために使用する構成クラスについて詳しく説明します。

4.1 構成クラスの基本

WS-Policy 構成クラスを手動で作成するには、%SOAP.Configuration のサブクラスを作成します。このクラスで、次のとおり XData ブロックを追加します。

```
XData service
{
<cfg:configuration xmlns:cfg="http://www.intersystems.com/configuration" name="service">
...
```

XData ブロックの一般的な構造は、次のとおりです。

```
XData service
{
<cfg:configuration ...>
   <service ...>
      <method ...>
         <request ...>
         <response ...>
...
```

要素 <service>、<method>、<request>、および <response> にはそれぞれ、そのレベルで適用されるポリシー情報を含めることができます。<service> 要素は必須ですが、他の要素はオプションです。

ポリシー情報を含める場合は、ポリシー式 (<wsp:Policy> 要素) か、ポリシー参照 (同じ構成クラス内の別の XData ブロックに含まれているポリシーをポイントする <wsp:PolicyReference> 要素) のいずれかになります。以下のセクションで詳細を説明します。

<PolicyReference> は 2 つの場所のみでサポートされています。構成要素内の <Policy> 要素の代わりとして、または <Policy> 要素の唯一の子としてです。

4.2 InterSystems 拡張属性の追加

前述の cfg:wsdlElement 属性に加えて、ポリシー要素内の以下の要素で InterSystems 拡張属性の追加が必要になる場合があります。

- `<sp:X509Token>` (`<sp:InitiatorToken>` または `<sp:RecipientToken>` 内)

  この要素では、`cfg:FindField` 属性および `cfg:FindValue` 属性の値を指定します。これらの属性は、このトークンで使用される InterSystems IRIS 資格情報セットを指定するものです。

  - `cfg:FindField` 属性は、検索するフィールドの名前を指定します。通常、これは `Alias` です。
  - `cfg:FindValue` 属性は、そのフィールドの値を指定します。`cfg:FindField` が `Alias` である場合、これは、InterSystems IRIS 資格情報セットの名前です。

  以下はその例です。

  XML

```
<sp:X509Token IncludeToken="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy/IncludeToken/Never"

              cfg:FindField="Alias"
              cfg:FindValue="servercred">
  <wsp:Policy>
    <sp:WssX509V3Token11/>
  </wsp:Policy>
</sp:X509Token>
```

- `<sp:HttpsToken>`

  この要素では、`cfg:SSLConfiguration` 属性の値を指定します。これは、InterSystems IRIS SSL/TLS 構成の名前と同じである必要があります。以下はその例です。

  XML

```
<sp:HttpsToken cfg:SSLConfiguration="mysslconfig">
    <wsp:Policy/>
</sp:HttpsToken>
```

  Web クライアントの場合のみ、この属性を指定してください。

- `<sp:SecureConversationToken>`

  この要素では、`cfg:Lifetime` 属性を指定できます。これは、安全な通信の存続時間と同じにする必要があります。単位は、時間または小数の時間です。既定の存続時間は 5 分です。この存続時間を 15 分に指定すると仮定します。このためには、以下のように `<sp"SecureConversationToken>` を編集します。

```
<sp:SecureConversationToken cfg:Lifetime=".25"

sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/IncludeToken/AlwaysToRecipient">

    <wsp:Policy>
      <sp:MustNotSendAmend/>
      <sp:MustNotSendRenew/>
  ...
  </wsp:Policy>
</sp:SecureConversationToken>
```

  Web クライアントの場合のみ、この属性を指定してください。

接頭語 `cfg` の詳細は、次のセクションを参照してください。

Web サービス/クライアント構成ウィザードを使用してポリシーを作成する場合、これらの属性の値は、InterSystems IRIS によって自動的に設定されます。Web クライアントまたはサービスの生成時に構成クラスを生成する場合は、これらの属性を編集する必要があることもあります。

4.3 構成 XData ブロックの詳細

このセクションでは、Web サービスまたはクライアント構成クラスの XData ブロックの内容について説明します。

<configuration>、<service>、<method>、<request>、および <response> 要素はすべて、次のネームスペース内に存在していなければなりません。

```
"http://www.intersystems.com/configuration"
```

このトピックでは、接頭語 cfg がそのネームスペースを示します。

"InterSystems 拡張属性の追加" も参照してください。

4.3.1 〈configuration〉

<configuration> 要素は、XData ブロックのルート要素です。この要素には次の項目が含まれます。

属性または要素	目的
name	(オプション) この構成の名前。指定する場合は、XData ブロックの名前と一致させる必要があります。
<service>	(オプション) ポリシーを InterSystems IRIS Web サービスまたは Web クライアントに関連付けます。

4.3.2 〈service〉

<service> 要素は、ポリシーを InterSystems IRIS Web サービスまたは Web クライアントに関連付けます。この要素には次の項目が含まれます。

属性または要素	目的
classname	(必須) InterSystems IRIS Web サービスまたは Web クライアントのパッケージとクラスの完全な名前。
<wsp:Policy>	(0 または 1 を含む) この Web サービスまたはクライアントに (バインディング・レベルで) 適用するポリシーを指定します。WS-Policy 1.2 または WS-Policy 1.5 のポリシー式を指定します。 <wsp:Policy>、<wsp:PolicyReference> を指定するか、どちらも指定しません。
<wsp:PolicyReference>	(0 または 1 を含む) この Web サービスまたはクライアントに (バインディング・レベルで) 適用するポリシー参照を指定します。これを指定する場合、policyID 属性は、同じ構成クラス内の別の XData ブロックで定義されているローカル・ポリシーへの参照である必要があります。使用例は、"ポリシー参照を使用した構成" を参照してください。 <wsp:Policy>、<wsp:PolicyReference> を指定するか、どちらも指定しません。
<method>	(0 以上を含む) 指定の Web サービスまたはクライアントの特定の Web メソッドにポリシーを関連付けます (処理レベルで適用)。<service> 要素には、<method> 要素を任意の数だけ含めることができます。

以下はその例です。

XML

```
<cfg:configuration
xmlns:cfg="http://www.intersystems.com/configuration"
xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702"
xmlns:wsap="http://www.w3.org/2006/05/addressing/wsdl"
xmlns:wsp="http://www.w3.org/ns/ws-policy"
name="service">
  <cfg:service classname="DemoPolicies.NoSecurity">
    <wsp:Policy>
      <wsap:UsingAddressing/>
    </wsp:Policy>
  </cfg:service>
</cfg:configuration>
```

<service> の子である <wsp:Policy> または <wsp:PolicyReference> 内で、cfg:wsdlElement 属性を指定できます。この属性は、このポリシー要素の添付先となる WSDL のパートを指定するものです。このコンテキストでは、この属性に次のいずれの値も指定できます。

- "service" ― このポリシー要素を WSDL <service> 要素に添付します。
- "port" ― このポリシー要素を WSDL <port> 要素に添付します。
- "binding" (既定) ― このポリシー要素を WSDL <binding> 要素に添付します。
- "portType" ― このポリシー要素を WSDL <portType> 要素に添付します。

4.3.3 〈method〉

<method> 要素は、親の <service> 要素によって指定された Web サービスまたはクライアント内の特定の Web メソッドにポリシーを関連付けます。<method> 要素には次の項目が含まれます。

属性または要素	目的
name	Web メソッドの名前。
<wsp:Policy>	(0 または 1 を含む) この Web サービスまたはクライアントに (処理レベルで) 適用するポリシーを指定します。WS-Policy 1.2 または WS-Policy 1.5 のポリシー式を指定します。 <wsp:Policy>、<wsp:PolicyReference> を指定するか、どちらも指定しません。
<wsp:PolicyReference>	(0 または 1 を含む) この Web メソッドのオプションの参照として WS-Policy 1.2 または WS-Policy 1.5 のポリシー式を指定します。policyID 属性は、同じ構成クラス内の別の XData ブロックで定義されているローカル・ポリシーへの参照です。使用例は、"ポリシー参照を使用した構成" を参照してください。 <wsp:Policy>、<wsp:PolicyReference> を指定するか、どちらも指定しません。
<request>	(0 または 1 を含む) ポリシーを Web メソッドの要求メッセージに関連付けます。
<response>	(0 または 1 を含む) ポリシーを Web メソッドの応答メッセージに関連付けます。

<method> の子である <wsp:Policy> または <wsp:PolicyReference> 内で、cfg:wsdlElement attribute 属性を指定できます。この属性は、このポリシー要素の添付先となる WSDL のパートを指定するものです。このコンテキストでは、この属性に次のいずれの値も指定できます。

- "binding" (既定) ― このポリシー要素を WSDL <binding> 要素に添付します。
- "portType" ― このポリシー要素を WSDL <portType> 要素に添付します。

4.3.4 〈request〉

<request> 要素は、親の <method> 要素が参照する Web メソッドの要求メッセージにポリシーを関連付けます。<request> 要素には次の項目が含まれます。

属性または要素	目的
<wsp:Policy>	(0 または 1 を含む) 要求メッセージに適用するポリシーを指定します。WS-Policy 1.2 または WS-Policy 1.5 のポリシー式を指定します。<wsp:Policy>、<wsp:PolicyReference> を指定するか、どちらも指定しません。
<wsp:PolicyReference>	(0 または 1 を含む) この要求メッセージのオプションの参照として WS-Policy 1.2 または WS-Policy 1.5 のポリシー式を指定します。policyID 属性は、同じ構成クラス内の別の XData ブロックで定義されているローカル・ポリシーへの参照です。使用例は、"ポリシー参照を使用した構成" を参照してください。<wsp:Policy>、<wsp:PolicyReference> を指定するか、どちらも指定しません。

<request> の子である <wsp:Policy> または <wsp:PolicyReference> 内で、cfg:wsdlElement 属性を指定できます。この属性は、このポリシー要素の添付先となる WSDL のパートを指定するものです。このコンテキストでは、この属性に次のいずれの値も指定できます。

- "binding" (既定) ― このポリシー要素を WSDL <binding> 要素に添付します。
- "portType" ― このポリシー要素を WSDL <portType> 要素に添付します。
- "message" ― このポリシー要素を WSDL <message> 要素に添付します。

4.3.5 〈response〉

<response> 要素は、親の <method> 要素が参照する Web メソッドの応答メッセージにポリシーを関連付けます。<response> 要素には次の項目が含まれます。

属性または要素	目的
<wsp:Policy>	(0 または 1 を含む) 応答メッセージに適用するポリシーを指定します。WS-Policy 1.2 または WS-Policy 1.5 のポリシー式を指定します。<wsp:Policy>、<wsp:PolicyReference> を指定するか、どちらも指定しません。
<wsp:PolicyReference>	(0 または 1 を含む) この応答メッセージのオプションの参照として WS-Policy 1.2 または WS-Policy 1.5 のポリシー式を指定します。policyID 属性は、同じ構成クラス内の別の XData ブロックで定義されているローカル・ポリシーへの参照です。使用例は、"ポリシー参照を使用した構成" を参照してください。<wsp:Policy>、<wsp:PolicyReference> を指定するか、どちらも指定しません。

<response> の子である <wsp:Policy> または <wsp:PolicyReference> 内で、cfg:wsdlElement 属性を指定できます。この属性は、このポリシー要素の添付先となる WSDL のパートを指定するものです。このコンテキストでは、この属性に次のいずれの値も指定できます。

- "binding" (既定) ― このポリシー要素を WSDL <binding> 要素に添付します。
- "portType" ― このポリシー要素を WSDL <portType> 要素に添付します。

- "message" ― このポリシー要素を WSDL <message> 要素に添付します。

4.4 カスタム構成の例

このセクションでは、カスタム構成クラスの例をいくつか紹介します。

4.4.1 代替ポリシーを使用した構成

次の構成クラスには、2 つの代替ポリシー、すなわち WS-Addressing ヘッダを使用するものと使用しないものが含まれています。

Class Definition

```
/// PolicyAlternatives.DivideWSConfig
Class PolicyAlternatives.DivideWSConfig Extends %SOAP.Configuration
{

XData service
{
<cfg:configuration xmlns:cfg="http://www.intersystems.com/configuration"
xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702"
xmlns:wsap="http://www.w3.org/2006/05/addressing/wsdl"
xmlns:wsp="http://www.w3.org/ns/ws-policy"
name="service">
  <cfg:service classname="PolicyAlternatives.DivideWS">
   <wsp:Policy>
     <wsp:ExactlyOne>
       <wsp:All>
         <wsap:UsingAddressing/>
       </wsp:All>
       <wsp:All>
         <wsp:Policy/>
       </wsp:All>
     </wsp:ExactlyOne>
    </wsp:Policy>
  </cfg:service>
</cfg:configuration>
}

}
```

WS-Addressing を必要とするポリシーが添付されている Web クライアントで使用されると、この Web サービスは、WS-Addressing ヘッダを持つメッセージで応答します。WS-Addressing を使用しないポリシーのクライアントで使用されると、この Web サービスは、WS-Addressing ヘッダを持たないメッセージで応答します。

別のシナリオとしては、あるポリシーでは SSL/TLS を必要とし、別なポリシーではメッセージの暗号化を使用する (X.509 相互証明書セキュリティ ポリシーなど) といったものがあります。

4.4.2 ポリシー参照を使用した構成

次の構成クラスには、2 つの XData ブロックが含まれています。1 つには、ID 属性が mypolicy であるポリシーが含まれており、もう 1 つには、Web サービスの構成が含まれています。この構成は、他の XData ブロックにあるポリシーを参照します。

Class Definition

```
Class DemoPolicies.WithReferenceConfig Extends %SOAP.Configuration
{

XData service
{
<cfg:configuration
xmlns:cfg="http://www.intersystems.com/configuration"
```

```
xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702"
xmlns:wsp="http://www.w3.org/ns/ws-policy"
name="service">
  <cfg:service classname="DemoPolicies.WithReference">
    <wsp:PolicyReference URI="#mypolicy">
    </wsp:PolicyReference>
  </cfg:service>
</cfg:configuration>
}

XData Policy1
{
<wsp:Policy
  xmlns:wsp="http://www.w3.org/ns/ws-policy"
  xmlns:wsu="http://schemas.xmlsoap.org/ws/2003/06/utility"
  xmlns:wsap="http://www.w3.org/2006/05/addressing/wsdl"
  xmlns:wsoma="http://schemas.xmlsoap.org/ws/2004/09/policy/optimizedmimeserialization"
  wsu:Id="mypolicy">
 <wsap:UsingAddressing/>
 <wsoma:OptimizedMimeSerialization/>
</wsp:Policy>
}

}
```

この例では、ポリシー式は `Policy1` という名前の XData ブロックに含まれています。このブロックの名前は、WSDL または SOAP の操作に影響しません。

5
セキュリティ要素の手動追加

このトピックでは、InterSystems IRIS Web サービスおよび InterSystems IRIS Web クライアントによって送信されるメッセージにセキュリティ要素を手動で追加する一般的な方法について説明します。

以下の各トピックでは、特定のセキュリティ・タスクの詳細について説明します。

5.1 セキュリティ・ヘッダ要素の追加

WS-Security ヘッダ要素にセキュリティ要素を追加するには、Web クライアントまたは Web サービスで以下の一般的な手順を実行します。

1. 該当するクラスのインスタンスを作成します。そのためには、そのクラスに応じて Create() または CreateX509() という名前のメソッドを使用します。このインスタンスは、<Username> や <EncryptedKey> などの WS-Security ヘッダ要素の 1 つを表します。
2. Web クライアントまたは Web サービスの SecurityOut プロパティを更新して、各インスタンスを WS-Security ヘッダ要素に追加します。これには、AddSecurityElement() メソッドを呼び出します。
3. SOAP メッセージを送信します。メッセージにWS-Security ヘッダが組み込まれ、ヘッダには追加した要素が含まれます。
4. 以降の発信メッセージは、次のようになります。
   - Web クライアントの場合、SecurityOut プロパティは変更されず、このインスタンスからの以降の発信メッセージには、追加したセキュリティ・ヘッダが含まれます。これが適切でない場合は、SecurityOut プロパティを NULL に設定します。
   - Web サービスの場合、SecurityOut プロパティは、最初の発信 SOAP メッセージの後で、自動的に NULL に設定されます。

5.2 ヘッダ要素の順序

複数のセキュリティ要素をヘッダに追加する場合、セキュリティ・ヘッダ要素を適切な順序で追加することが重要になります。同じメッセージ要素の暗号化と署名の両方を実行する場合、これは特に重要です。つまり、暗号化処理と署名処理を実行するのと同じ順序でこれらを追加します。

ヘッダ要素の順序は、メッセージの処理が行われる順序を示します。WS-Security 1.1 仕様では、このことを以下のように規定しています。

```
As elements are added to a <wsse:Security> header block, they SHOULD be prepended to
the existing elements. As such, the <wsse:Security> header block represents the signing and
encryption steps the message producer took to create the message. This prepending rule
ensures that the receiving application can process sub-elements in the order they appear in the
<wsse:Security> header block, because there will be no forward dependency among the sub-
elements.
```

ヘッダ要素を追加すると、InterSystems IRIS は、前に追加した要素の先頭に各要素を追加します。ただし、以下の例外があります。

- `<Timestamp>` 要素を含める場合は、この要素が最初になるよう強制されます。
- `<BinarySecurityToken>` 要素を含める場合は、この要素が `<Timestamp>` 要素（含まれている場合）の後、または最初になるよう強制されます。
- AddSecurityElement() を使用して暗号化されたセキュリティ・ヘッダ要素を追加する場合は、2 番目の引数を指定して、挿入する `<EncryptedData>` 要素が、関連付けられた `<EncryptedKey>` 要素の後になるよう強制します。

同じメッセージ要素の暗号化と署名の両方を実行する場合、セキュリティ・ヘッダ要素を適切な順序で追加することが特に重要です。つまり、暗号化処理と署名処理を実行するのと同じ順序でこれらを追加します。

6
タイムスタンプおよびユーザ名トークンの追加

このトピックでは、タイムスタンプおよびユーザ・トークンについて説明します。

6.1 概要

タイムスタンプは、WS-Security ヘッダの <Timestamp> セキュリティ要素です。タイムスタンプは、厳密にはセキュリティ要素ではありません。ただし、リプレイ攻撃を阻止するためにタイムスタンプを使用できます。タイムスタンプは、カスタム・ログにも役に立ちます。

ユーザ名トークンは、WS-Security ヘッダの <UsernameToken> セキュリティ要素で、ユーザ名が保持されます。また、対応するパスワードも保持されます (オプションにてダイジェスト形式で)。通常、これを認証に使用します。つまり、InterSystems IRIS Web クライアントが、パスワードを必要とする Web サービスを使用できるようにします。

注意　WS-Security ヘッダ要素は、既定ではクリア・テキストで送信されます。<UsernameToken> のパスワードを保護するには、SSL/TLS を使用するか、<UsernameToken> を暗号化するか (別途説明)、これらの手法を組み合わせて使用する必要があります。

6.2 タイムスタンプの追加

WS-Security ヘッダ要素にタイムスタンプを追加するには、Web クライアントまたは Web サービスで次の操作を実行します。

1. %SOAP.Security.Timestamp の Create() クラス・メソッドを呼び出します。このメソッドは、オプションの引数 (秒単位の有効期間) を 1 つ取ります。既定の有効期間は 300 秒です。以下はその例です。

   ```
   set ts=##class(%SOAP.Security.Timestamp).Create()
   ```

   このメソッドは、%SOAP.Security.Timestamp のインスタンスを作成し、その Created、Expires、および TimestampAtEnd プロパティの値を設定して、インスタンスを返します。このインスタンスは、<Timestamp> ヘッダ要素を表します。

2. Web クライアントまたは Web サービスの SecurityOut プロパティの AddSecurityElement() メソッドを呼び出します。このメソッドの引数として、作成した %SOAP.Security.Timestamp インスタンスを使用します。以下はその例です。

   ObjectScript

   ```
   do client.SecurityOut.AddSecurityElement(ts)
   ```

3. SOAP メッセージを送信します。"セキュリティ・ヘッダ要素の追加" の一般的な手順を参照してください。

<Timestamp> 要素を含める場合、InterSystems IRIS は、この要素が <Security> 内で最初になるように強制します。

6.3 ユーザ名トークンの追加

ユーザ名トークンを追加するには、Web クライアントで次の操作を実行します。

1. 必要に応じて、%soap.inc インクルード・ファイルを組み込みます。このファイルには、使用する可能性のあるマクロが定義されています。
2. %SOAP.Security.UsernameToken の Create() クラス・メソッドを呼び出します。以下はその例です。

ObjectScript

```
set user="SYSTEM"
set pwd="_SYS"
set utoken=##class(%SOAP.Security.UsernameToken).Create(user,pwd)
```

メソッドには、オプションの 3 番目の引数 (type) があり、これはユーザ名トークンにパスワードを含める方法を指定します。これは次のいずれかである必要があります。

- $$$SOAPWSPasswordText – プレーン・テキストでパスワードを含めます。これが既定値です。
- $$$SOAPWSPasswordDigest – パスワードを含めず、代わりにパスワードのダイジェストを含めます。ダイジェスト、Nonce、および Created タイム・スタンプは、WS-Security 1.1 で指定されたように導出されます。

  重要　このオプションは、SOAP 対応のサードパーティ製サーバと対話する SOAP クライアントでのみ利用できます。PasswordDigest 認証では、パスワードをサーバに平文で格納する必要がありますが、これは最近のセキュリティ環境では許容されません。PasswordDigest アルゴリズムはレガシー機能であると考える必要があります。<UsernameToken> のパスワードを保護するには、SSL/TLS を使用するか、<UsernameToken> を暗号化するか、これらの手法を組み合わせて使用する必要があります。

- $$$SOAPWSPasswordNone – パスワードを含めません。

このメソッドは、%SOAP.Security.UsernameToken のインスタンスを作成し、その Username プロパティおよび Password プロパティを設定して、インスタンスを返します。このオブジェクトは、<UsernameToken> ヘッダ要素を表します。

注釈　スタジオ・ウィザードによって作成されたポリシーで必要な <UsernameToken> を作成するためにこの手順を使用する場合、ウィザードは他のトークン・タイプを使用するポリシーを生成しないため、既定タイプ $$$SOAPWSPasswordText を使用する必要があります。ただし、HashPassword アサーションを使用するポリシー ($$$SOAPWSPasswordDigest を使用するポリシー) を手動で作成することができます。

3. Web クライアントまたは Web サービスの SecurityOut プロパティの AddSecurityElement() メソッドを呼び出します。このメソッドの引数として、作成した %SOAP.Security.UsernameToken インスタンスを使用します。以下はその例です。

ObjectScript

```
do client.SecurityOut.AddSecurityElement(utoken)
```

4. SOAP メッセージを送信します。"セキュリティ・ヘッダ要素の追加" の一般的な手順を参照してください。

6.4 タイムスタンプおよびユーザ名トークンの例

この例では、パスワード認証を必要とする Web サービス、およびその要求メッセージ内でタイムスタンプとユーザ名トークンを送信する Web クライアントを示します。

注意　　この例では、ユーザ名とパスワードがクリア・テキストで送信されます。

個々の環境でこの例のとおり機能するようにするには、まず次の手順を実行します。

- Web サービスが属している Web アプリケーションの場合、パスワード認証のみをサポートするように、そのアプリケーションを構成します。
  1. 管理ポータルのホーム・ページで、**[システム管理]→[セキュリティ]→[アプリケーション]→[Web アプリケーション]** を選択します。
  2. Web アプリケーションを選択します。
  3. **[パスワード]** オプションのみを選択してから、**[保存]** を選択します。
- 既定値を使用しない場合は、適切な InterSystems IRIS ユーザ名とパスワードを使用するように、クライアントを編集します。

Web サービスは、次のとおりです。

Class Definition

```
Class Tokens.DivideWS Extends %SOAP.WebService
{

Parameter SECURITYIN = "REQUIRE";

///  Name of the Web service.
Parameter SERVICENAME = "TokensDemo";

///  SOAP namespace for the Web service
Parameter NAMESPACE = "http://www.myapp.org";

///  Divide arg1 by arg2 and return the result. In case of error, call ApplicationError.
Method Divide(arg1 As %Numeric = 2, arg2 As %Numeric = 8) As %Numeric [ WebMethod ]
{
  Try {
       Set ans=arg1 / arg2
       }Catch{
       Do ..ApplicationError("division error")
       }
  Quit ans
}

///  Create our own method to produce application specific SOAP faults.
Method ApplicationError(detail As %String)
{
    //details not shown here
}

}
```

次のクライアント側のクラスは、プロキシ・クライアント (ここには示されていません) を呼び出し、ユーザ名トークンを追加します。

Class Definition

```
Include %systemInclude

Class TokensClient.UseClient
{

ClassMethod Test() As %Numeric
{
```

```
  Set client=##class(TokensClient.TokensDemoSoap).%New()

  Do ..AddSecElements(.client)
  Set ans=client.Divide(1,2)

  Quit ans
}

ClassMethod AddSecElements(ByRef client As %SOAP.WebClient)
{
   Set utoken=##class(%SOAP.Security.UsernameToken).Create("_SYSTEM","SYS")
   Do client.SecurityOut.AddSecurityElement(utoken)

   Set ts=##class(%SOAP.Security.Timestamp).Create()
   Do client.SecurityOut.AddSecurityElement(ts)
   Quit
}

}
```

このクライアントからのサンプル・メッセージは、次のようになります。

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope [parts omitted]>
  <SOAP-ENV:Header>
   <Security xmlns="[parts omitted]oasis-200401-wss-wssecurity-secext-1.0.xsd">
      <Timestamp xmlns="[parts omitted]oasis-200401-wss-wssecurity-utility-1.0.xsd">
         <Created>2010-03-12T20:18:03Z</Created>
         <Expires>2010-03-12T20:23:03Z</Expires>
      </Timestamp>
      <UsernameToken>
         <Username>_SYSTEM</Username>
         <Password
          Type="[parts omitted]#PasswordText">
            SYS
         </Password>
      </UsernameToken>
      </Security>
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
      [omitted]
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

7

SOAP 本文の暗号化

このトピックでは、InterSystems IRIS Web サービスおよび Web クライアントによって送信される SOAP メッセージの本文を暗号化する方法について説明します。

"セキュリティ・ヘッダ要素の暗号化" のトピックおよび "暗号化および署名への派生キー・トークンの使用" のトピックでは、セキュリティ・ヘッダ要素を暗号化する方法および SOAP 本文を暗号化する他の方法について説明しています。

7.1 暗号化の概要

SOAP メッセージの暗号化に対する InterSystems IRIS サポートは、WS-Security 1.1 を基盤としています。また、WS-Security は、XML 暗号化仕様に従います。後者の仕様に従って、XML ドキュメントを暗号化する方法は、以下のとおりです。

1. 一時的に使用する対称鍵を生成します。
2. これを使用して、ドキュメント (または、ドキュメントの選択部分) を暗号化します。

   ドキュメントのこの部分を、暗号化されたコンテンツを含む `<EncryptedData>` 要素に置き換えます。
3. ドキュメントの送信先エンティティの公開鍵を使用して、対称鍵を暗号化します。

   公開鍵は、そのエンティティからの要求メッセージに含まれている X.509 証明書から取得できます。または、事前に取得することができます。
4. 暗号化された対称鍵を、同じドキュメントの `<EncryptedKey>` 要素内に含めます。`<EncryptedKey>` 要素は、この要素の解読に使用する鍵を受信者が判別できる情報を直接または間接的に提供します。

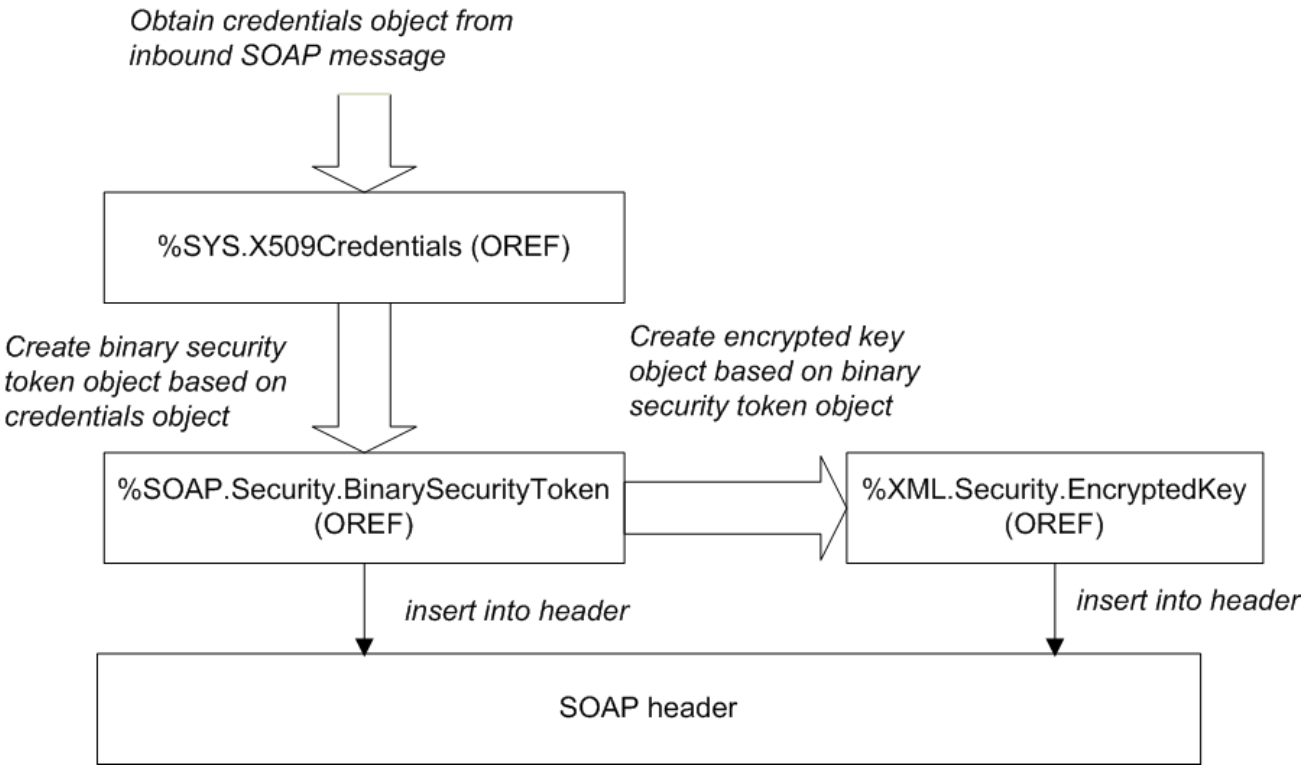
   この情報は、`<EncryptedKey>` 要素内に含めることができます。または、`<EncryptedKey>` 要素は、X.509 証明書または署名済みの SAML アサーションを格納しているバイナリ・セキュリティ・トークンへの直接参照を含めることができます。後者の場合、`<Signature>` 要素を追加する前にセキュリティ・トークンをメッセージに追加する必要があります。

ドキュメントには、ドキュメントのさまざまな暗号化部分に適用可能な複数の `<EncryptedKey>` 要素を含めることができます。

別のトピックで、SOAP メッセージの各部分を暗号化するその他の方法について説明します。メッセージ自体の詳細はさまざまですが、一般的なプロセスは同じで、XML 暗号化仕様に従います。つまり、対称鍵の生成および使用、対称鍵の暗号化、暗号化された対称鍵のメッセージへの組み込みを行います。

7.2 SOAP 本文の暗号化

SOAP メッセージの本文を暗号化するには、次に説明する基本的な手順を使用するか、またはサブセクションで説明するバリエーションを使用します。まず、次の図にプロセスを示します。



詳細なプロセスは以下のとおりです。

1. 必要に応じて、%soap.inc インクルード・ファイルを組み込みます。このファイルには、使用する可能性のあるマクロが定義されています。
2. SOAP メッセージを受信するエンティティの公開鍵を含む資格情報セットを取得します。一般的には、受信した着信メッセージから取得します。"プログラムによる資格情報セットの取得" を参照してください。

   例：

   ObjectScript

   ```
   set credset=..SecurityIn.Signature.X509Credentials
   ```

   "プログラムによる資格情報セットの取得" の説明に従って、返されたオブジェクトのタイプが %SYS.X509Credentials のインスタンスかどうか確認します。
3. その資格情報セットに関連付けられている証明書を含むバイナリ・セキュリティ・トークンを作成します。これには、%SOAP.Security.BinarySecurityToken の CreateX509Token() クラス・メソッドを呼び出します。以下はその例です。

   ObjectScript

   ```
   set bst=##class(%SOAP.Security.BinarySecurityToken).CreateX509Token(credset)
   ```

このメソッドは、<BinarySecurityToken> ヘッダ要素を表す %SOAP.Security.BinarySecurityToken のインスタンスを返します。

4. WS-Security ヘッダ要素にこのトークンを追加します。そのためには、Web クライアントまたは Web サービスの SecurityOut プロパティの AddSecurityElement() メソッドを呼び出します。このメソッドの引数には、作成したトークンを使用します。以下はその例です。

   ObjectScript

   ```
   do ..SecurityOut.AddSecurityElement(bst)
   ```

5. バイナリ・セキュリティ・トークンに基づいて、暗号化キーを作成します。これには、%XML.Security.EncryptedKey の CreateX509() クラス・メソッドを呼び出します。以下はその例です。

   ObjectScript

   ```
   set enckey=##class(%XML.Security.EncryptedKey).CreateX509(bst)
   ```

   このメソッドは、対称鍵を生成し、これを使用して SOAP 本文を暗号化し、<EncryptedKey> ヘッダ要素を表す %XML.Security.EncryptedKey のインスタンスを返します。このヘッダ要素には、指定されたバイナリ・セキュリティ・トークン内にある公開鍵で暗号化された対称鍵が含まれます。

6. 必要に応じて、別のアルゴリズムを使用するように暗号化キーのインスタンスを変更します。"ブロック暗号化アルゴリズムの指定" および "鍵転送アルゴリズムの指定" を参照してください。

7. WS-Security ヘッダ要素に <EncryptedKey> 要素を追加します。そのためには、Web クライアントまたは Web サービスの SecurityOut プロパティの AddSecurityElement() メソッドを呼び出します。追加する要素について、%XML.Security.EncryptedKey のインスタンスを指定します。

   以下はその例です。

   ObjectScript

   ```
   do ..SecurityOut.AddSecurityElement(enckey)
   ```

   この手順では、<Body> 要素の子として <EncryptedData> 要素も追加します。

8. SOAP メッセージを送信します。SOAP 本文は暗号化され、WS-Security ヘッダはメッセージに含まれます。

   WS-Security ヘッダには、<BinarySecurityToken> 要素および <EncryptedKey> 要素が含まれます。

   "セキュリティ・ヘッダ要素の追加" の一般的な手順を参照してください。

7.2.1 バリエーション : 証明書を指定する情報の使用

<BinarySecurityToken> には、シリアル化された Base 64 のエンコード形式の証明書が含まれています。このトークンを省略し、代わりに証明書を指定する情報を使用できます。受信者は、この情報を使用して、適切な場所から証明書を取得します。そのためには、前述の手順を使用しますが、以下の変更点があります。

- 手順 3 と 4 をスキップします。つまり、<BinarySecurityToken> を追加しないでください。
- 手順 5 (暗号化キーの作成) で、CreateX509() の最初の引数として手順 1 の資格情報セットを使用します (バイナリ・セキュリティ・トークンは使用しません)。以下はその例です。

  ObjectScript

  ```
  set enckey=##class(%XML.Security.EncryptedKey).CreateX509(credset,,referenceOption)
  ```

  3 番目の引数 (referenceOption) では、<Signature> 要素が証明書を使用する方法を指定できます。

(このバリエーションで実行しているように) 最初の引数として資格情報セットを指定する場合、referenceOption の既定値は $$$SOAPWSReferenceThumbprint です。必要に応じて、このサブセクションの説明に従って、値を指定します。$$$SOAPWSReferenceDirect 以外の任意の値を使用できます。

7.2.1.1 X.509 証明書の参照オプション

"WS-Security ヘッダの概要" のセクションでは、証明書が SOAP メッセージで使用される 1 つの方法を示しています。そこでの例では、デジタル・シグニチャは、以下の 2 つのヘッダ要素から構成されています。

- シリアル化された Base 64 のエンコード形式の証明書を保持する <BinarySecurityToken>。
- シグニチャを保持し、バイナリ・セキュリティ・トークンへの直接参照を含む <Signature> 要素。

参照の形式は、この他にも考えられます。例えば、<Signature> は、代わりに証明書のサムプリントを含むことができ、この場合、メッセージに <BinarySecurityToken> は不要です。

暗号化キー、デジタル・シグニチャ、または SAML アサーションを作成する場合、referenceOption 引数を指定することにより、新しく作成された要素が、資格情報に含まれている証明書 (つまり、キー・マテリアル) を使用する方法を制御できます。

参考として、この引数は、以下のいずれかの値を持つことができます。これらの値は、%soap.inc インクルード・ファイルで定義されているマクロです。

$$$SOAPWSReferenceDirect

要素にバイナリ・セキュリティ・トークンへの直接参照が組み込まれます。具体的には、<KeyInfo> 要素は、URI 属性が <BinarySecurityToken> に対するローカル参照である <Reference> 要素を下位に持つ <SecurityTokenReference> 下位要素を使用して作成されます。このオプションを使用するには、必ず WS-Security ヘッダにセキュリティ・トークンも追加しなければなりません。詳細は、関連するセクションを参照してください。

$$$SOAPWSReferenceThumbprint

この要素には、X.509 証明書の SHA-1 サムプリントが組み込まれます。

$$$SOAPWSReferenceKeyIdentifier

この要素には、X.509 証明書の SubjectKeyIdentifier が組み込まれます。

$$$SOAPWSReferenceIssuerSerial

この要素には、<X509IssuerSerial> 要素を格納する <X509Data> の子を持つ <SecurityTokenReference> の子が指定された <KeyInfo> 要素が組み込まれます。

$$$KeyInfoX509Certificate

この要素には、<X509Certificate> 要素を格納する <X509Data> の子を持つ <KeyInfo> 要素が組み込まれます。WS-Security 仕様では、これを <Signature> 要素と <EncryptedKey> 要素に使用することは推奨されていませんが、<Assertion> 要素では使用できます。

$$$KeyInfoX509IssuerSerial

この要素には、<X509IssuerSerial> 要素を格納する <X509Data> の子を持つ <KeyInfo> 要素が組み込まれます。WS-Security 仕様では、これを <Signature> 要素と <EncryptedKey> 要素に使用することは推奨されていませんが、<Assertion> 要素では使用できます。

$$$KeyInfoX509SKI

この要素には、<X509SKI> 要素を格納する <X509Data> の子を持つ <KeyInfo> 要素が組み込まれます。WS-Security 仕様では、これを <Signature> 要素と <EncryptedKey> 要素に使用することは推奨されていませんが、<Assertion> 要素では使用できます。

$$$KeyInfoX509SubjectName

この要素には、<X509SubjectName> 要素を格納する <X509Data> の子を持つ <KeyInfo> 要素が組み込まれます。WS-Security 仕様では、これを <Signature> 要素と <EncryptedKey> 要素に使用することは推奨されていませんが、<Assertion> 要素では使用できます。

$$$KeyInfoRSAKey

この要素には、<RSAKeyValue> 要素を格納する <KeyValue> の子を持つ <KeyInfo> 要素が組み込まれます。WS-Security 仕様では、これを <Signature> 要素と <EncryptedKey> 要素に使用することは推奨されていませんが、<Assertion> 要素では使用できます。

7.2.2 バリエーション :署名済みの SAML アサーションの使用

署名済みの SAML アサーションの証明書に格納されている公開鍵を使用して暗号化するには、以下の手順を実行します。

1. 前述の手順の手順 1-4 をスキップします。
2. Holder-of-key メソッドを使用する <SubjectConfirmation> 要素を使用して、署名済みの SAML アサーションを作成します。"SAML トークンの作成と追加" を参照してください。
3. <EncryptedKey> 要素を作成します。そのためには、CreateX509() クラス・メソッドの最初の引数として署名済みの SAML アサーションを使用します。以下はその例です。

   ObjectScript

   ```
   set enckey=##class(%XML.Security.EncryptedKey).CreateX509(signedassertion)
   ```

4. 前述の手順の手順 5 から続行します。

7.3 メッセージの暗号化の例

この例では、Web クライアント (ここでは示されていません) は署名済みの要求メッセージを送信し、Web サービスは暗号化された応答を送信します。

Web サービスは、クライアント証明書の要求メッセージのシグニチャから公開鍵を取得し、それを使用して、その応答内に <EncryptedKey> 要素を追加します。この応答は暗号化されます。<EncryptedKey> 要素は、クライアントの公開鍵で暗号化され、この要素には、応答メッセージ本文を暗号化するのに使用される対称鍵が含まれます。

Web サービスは、次のとおりです。

Class Definition

```
Class XMLEncr.DivideWS Extends %SOAP.WebService
{

Parameter SECURITYIN = "REQUIRE";

Parameter SERVICENAME = "XMLEncryptionDemo";
```

```
Parameter NAMESPACE = "http://www.myapp.org";

Method Divide(arg1 As %Numeric = 2, arg2 As %Numeric = 8) As %Numeric [ WebMethod ]
{
  Do ..EncryptBody()
  Try {
      Set ans=arg1 / arg2
      } Catch {
      Do ..ApplicationError("division error")
      }
  Quit ans
}

Method EncryptBody()
{
  //Retrieve X.509 certificate from the signature of the inbound request
  Set clientsig = ..SecurityIn.Signature
  Set clientcred = clientsig.X509Credentials
  set bst=##class(%SOAP.Security.BinarySecurityToken).CreateX509Token(clientcred)
  do ..SecurityOut.AddSecurityElement(bst)

  //generate a symmetric key, encrypt that with the public key of
  //the certificate contained in the token, and create an
  //<EncryptedKey> element with a direct reference to the token (default)
  Set enc=##class(%XML.Security.EncryptedKey).CreateX509(bst)

  //add the <EncryptedKey> element to the security header
  Do ..SecurityOut.AddSecurityElement(enc)
}

///  Create our own method to produce application specific SOAP faults.
Method ApplicationError(detail As %String)
{
    //details omitted
}

}
```

このサービスは、次のような応答メッセージを送信します。

```
<?xml version="1.0" encoding="UTF-8" ?>
   <SOAP-ENV:Envelope [parts omitted]>
      <SOAP-ENV:Header>
         <Security xmlns="[parts omitted]oasis-200401-wss-wssecurity-secext-1.0.xsd">
            <BinarySecurityToken wsu:Id="SecurityToken-4EC1997A-AD6B-48E3-9E91-8D50C8EA3B53"
                                 EncodingType="[parts omitted]#Base64Binary"
                                 ValueType="[parts omitted]#X509v3">
                          MIICnDCCAYQ[parts omitted]ngHKNhh
            </BinarySecurityToken>
            <EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#">
               <EncryptionMethod Algorithm="[parts omitted]xmlenc#rsa-oaep-mgf1p">
                  <DigestMethod xmlns="http://www.w3.org/2000/09/xmldsig#"
                                Algorithm="http://www.w3.org/2000/09/xmldsig#sha1">
               </DigestMethod>
               </EncryptionMethod>
               <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
                  <SecurityTokenReference
                        xmlns="[parts omitted]oasis-200401-wss-wssecurity-secext-1.0.xsd">
                     <Reference URI="#SecurityToken-4EC1997A-AD6B-48E3-9E91-8D50C8EA3B53"
                                ValueType="[parts omitted]#X509v3"></Reference>
                  </SecurityTokenReference>
               </KeyInfo>
               <CipherData>
                  <CipherValue>WtE[parts omitted]bSyvg==</CipherValue>
               </CipherData>
               <ReferenceList>
                  <DataReference URI="#Enc-143BBBAA-B75D-49EB-86AC-B414D818109F"></DataReference>
               </ReferenceList>
            </EncryptedKey>
         </Security>
      </SOAP-ENV:Header>
      <SOAP-ENV:Body>
         <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#"
                        Id="Enc-143BBBAA-B75D-49EB-86AC-B414D818109F"
                        Type="http://www.w3.org/2001/04/xmlenc#Content">
            <EncryptionMethod Algorithm="[parts omitted]#aes128-cbc"></EncryptionMethod>
            <CipherData>
               <CipherValue>MLwR6hvKE0gon[parts omitted]8njiQ==</CipherValue>
            </CipherData>
         </EncryptedData>
      </SOAP-ENV:Body>
   </SOAP-ENV:Envelope>
```

7.4 ブロック暗号化アルゴリズムの指定

既定では、$$$SOAPWSaes128cbc によってメッセージ自体が暗号化されます。別のアルゴリズムを指定できます。そのためには、%XML.Security.EncryptedKey のインスタンスの Algorithm プロパティを設定します。

指定可能な値は、$$$SOAPWSaes128cbc (既定)、$$$SOAPWSaes192cbc、および $$$SOAPWSaes256cbc です。

以下はその例です。

ObjectScript

```
set enckey.Algorithm=$$$SOAPWSaes256cbc
```

この情報は、別のシナリオで <EncryptedKey> を作成する場合にも適用可能です。これについては、別途説明します。

7.5 鍵転送アルゴリズムの指定

鍵転送アルゴリズムは、対称鍵用に使用される公開鍵暗号化アルゴリズムです (https://www.w3.org/TR/xmlenc-core/ を参照)。既定では、これは $$$SOAPWSrsaoaep です。代わりに $$$SOAPWSrsa15 を使用することもできます。そのためには、前の手順で作成した %XML.Security.EncryptedKey のインスタンスの SetEncryptionMethod() メソッドを呼び出します。このメソッドの引数は、$$$SOAPWSrsaoaep (既定) と $$$SOAPWSrsa15 のいずれかになります。

以下はその例です。

ObjectScript

```
do enckey.SetEncryptionMethod($$$SOAPWSrsa15)
```

この情報は、別のシナリオで <EncryptedKey> を作成する場合にも適用可能です。これについては、別途説明します。

8

セキュリティ・ヘッダ要素の暗号化

このトピックでは、InterSystems IRIS Web サービスおよび Web クライアントによって送信されるメッセージの WS-Security ヘッダ内の要素を暗号化する方法について説明します(ここで説明するツールは、単独でまたはセキュリティ・ヘッダ要素と組み合わせて、SOAP 本文の暗号化にも使用できます)。

通常、暗号化と署名の両方を実行します。このトピックでは、説明を簡単にするために、暗号化のみについて説明します。暗号化と署名の組み合わせに関する詳細は、"暗号化と署名の組み合わせ" を参照してください。

"暗号化および署名への派生キー・トークンの使用" のトピックでは、さらに別の方法で SOAP メッセージの部分を暗号化する方法について説明します。

8.1 セキュリティ・ヘッダ要素の暗号化

前のトピックで示した暗号化手法と異なり、WS-Security ヘッダ要素を暗号化するプロセスは、<EncryptedData> 要素を対応する <EncryptedKey> 要素に接続する方法を指定する必要があります。

セキュリティ・ヘッダ要素を暗号化するには、以下の手順を実行します。

1. 必要に応じて、%soap.inc インクルード・ファイルを組み込みます。このファイルには、使用する可能性のあるマクロが定義されています。
2. ヘッダ要素または暗号化される要素を作成します。以下はその例です。

   ObjectScript

   ```
   set userToken=##class(%SOAP.Security.UsernameToken).Create("_SYSTEM","SYS")
   ```

3. SOAP メッセージを受信するエンティティの公開鍵を含む資格情報セットを取得します。"プログラムによる資格情報セットの取得" を参照してください。

   例：

   ObjectScript

   ```
   set credset=..SecurityIn.Signature.X509Credentials
   ```

   "プログラムによる資格情報セットの取得" の説明に従って、返されたオブジェクトのタイプが %SYS.X509Credentials のインスタンスかどうか確認します。
4. 資格情報セットに基づいて、暗号化キーを作成します。そのためには、%XML.Security.EncryptedKey の CreateX509() クラス・メソッドを呼び出し、オプションで 2 番目の引数を指定します。以下はその例です。

ObjectScript

```
set enckey=##class(%XML.Security.EncryptedKey).CreateX509(credset,$$$SOAPWSEncryptNone)
```

このメソッドは、対称鍵を生成し、<EncryptedKey> ヘッダ要素を表す %XML.Security.EncryptedKey のインスタンスを返します。このヘッダ要素には、指定された資格情報セット内にある公開鍵で暗号化された対称鍵が含まれます。

2 番目の引数は、(鍵のその他の使用に加えて) この鍵が SOAP 本文を暗号化するかどうかを指定します。値 $$$SOAPWSEncryptNone は、この鍵が SOAP 本文の暗号化に使用されないことを意味します。この引数を省略すると、SOAP 本文も暗号化されます。

5. 必要に応じて、別のアルゴリズムを使用するように暗号化キーのインスタンスを変更します。"ブロック暗号化アルゴリズムの指定" および "鍵転送アルゴリズムの指定" を参照してください。

6. 各セキュリティ・ヘッダ要素を暗号化するには、その要素に基づいて <EncryptedData> 要素を作成します。そのためには、%XML.Security.EncryptedData の Create() クラス・メソッドを呼び出します。この手順では、暗号化するセキュリティ・ヘッダ要素である 2 番目の引数のみ指定します。以下はその例です。

   ObjectScript

   ```
   set encdata=##class(%XML.Security.EncryptedData).Create(,userToken)
   ```

7. <EncryptedKey> の場合、参照を <EncryptedData> 要素に追加します。各 <EncryptedData> 要素に対して、以下の手順を実行します。

   a. %XML.Security.DataReference の Create() クラス・メソッドを呼び出し、暗号化されたデータのインスタンスを引数として指定します。

   b. 暗号化キー・インスタンスの AddReference() メソッドを呼び出して、データ参照を引数として指定します。

   以下はその例です。

   ObjectScript

   ```
   set dataref=##class(%XML.Security.DataReference).Create(encdata)
   do enckey.AddReference(dataref)
   ```

   この手順は、暗号化キー・インスタンスを更新し、暗号化されたデータのインスタンスへのポインタを組み込みます。

   この <EncryptedKey> が SOAP 本文も暗号化する場合、<Body> の <EncryptedData> 要素への参照も自動的に組み込まれます。

8. WS-Security ヘッダ要素に <EncryptedKey> 要素を追加します。そのためには、Web クライアントまたは Web サービスの SecurityOut プロパティの AddSecurityElement() メソッドを呼び出します。追加する要素について、%XML.Security.EncryptedKey のインスタンスを指定します。

   以下はその例です。

   ObjectScript

   ```
   do ..SecurityOut.AddSecurityElement(enckey)
   ```

9. 暗号化されたセキュリティ・ヘッダ要素を WS-Security ヘッダ要素に追加します。そのためには、Web クライアントまたは Web サービスの SecurityOut プロパティの AddSecurityElement() メソッドを呼び出します。この場合、以下の 2 つの引数を指定します。

   a. 組み込むセキュリティ・ヘッダ要素 (その要素に基づく %XML.Security.EncryptedData のインスタンスではない)。

b. 暗号化キー・インスタンス。2 番目の引数は、最初の引数が指定するアイテムの配置場所を指定します。引数が A,B の場合、InterSystems IRIS は、A が B の後であることを確認します。これを指定して、受信側が暗号化キーを最初に処理し、これに依存する暗号化されたセキュリティ・ヘッダ要素を後で処理するようにします。

以下はその例です。

ObjectScript

```
 do ..SecurityOut.AddSecurityElement(userToken,enckey)
```

10. SOAP メッセージを送信します。"セキュリティ・ヘッダ要素の追加" の一般的な手順を参照してください。

8.2 基本的な例

以下の例は、Web クライアントを呼び出し、暗号化された <UsernameToken> を送信します。この例では、本文は暗号化されません。

ObjectScript

```
  Set client=##class(XMLEncrSecHeader.Client.XMLEncrSecHeaderSoap).%New()

  // Create UsernameToken
  set user="_SYSTEM"
  set pwd="SYS"
  set userToken=##class(%SOAP.Security.UsernameToken).Create(user,pwd)

  //get credentials for encryption
  set cred = ##class(%SYS.X509Credentials).GetByAlias("servernopassword")

  //get EncryptedKey element and add it
  set encropt=$$$SOAPWSEncryptNone   ; means do not encrypt body
  set enckey=##class(%XML.Security.EncryptedKey).CreateX509(cred,encropt)

  //create EncryptedData and add a reference to it from EncryptedKey
  set encdata=##class(%XML.Security.EncryptedData).Create(,userToken)
  set dataref=##class(%XML.Security.DataReference).Create(encdata)
  do enckey.AddReference(dataref)

  //add EncryptedKey to security header
  do client.SecurityOut.AddSecurityElement(enckey)
  //add UsernameToken and place it after EncryptedKey
  do client.SecurityOut.AddSecurityElement(userToken,enckey)

  Quit client.Divide(1,2)
```

このクライアントは、次のようなメッセージを送信します。

```
<?xml version="1.0" encoding="UTF-8" ?>
   <SOAP-ENV:Envelope [parts omitted]>
      <SOAP-ENV:Header>
         <Security xmlns="[parts omitted]oasis-200401-wss-wssecurity-secext-1.0.xsd">
            <EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#">
               <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p">
                  <DigestMethod
                    xmlns="http://www.w3.org/2000/09/xmldsig#"
                    Algorithm="http://www.w3.org/2000/09/xmldsig#sha1">
                  </DigestMethod>
               </EncryptionMethod>
               <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
                  <SecurityTokenReference
                       xmlns="[parts omitted]oasis-200401-wss-wssecurity-secext-1.0.xsd">
                    <KeyIdentifier EncodingType="[parts omitted]#Base64Binary"
                          ValueType="[parts omitted]#ThumbprintSHA1">[omitted]</KeyIdentifier>
                  </SecurityTokenReference>
               </KeyInfo>
               <CipherData>
                  <CipherValue>pftET8jFDEjNC2x[parts omitted]xEjNC2==</CipherValue>
               </CipherData>
               <ReferenceList>
```

```
                <DataReference URI="#Enc-61000920-44DE-471E-B39C-6D08CB17FDC2">
                </DataReference>
             </ReferenceList>
          </EncryptedKey>
          <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#"
                 Id="Enc-61000920-44DE-471E-B39C-6D08CB17FDC2"
                 Type="http://www.w3.org/2001/04/xmlenc#Element">
             <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc">
             </EncryptionMethod>
             <CipherData>
                <CipherValue>wW3ZM5tgPD[parts omitted]tgPD==</CipherValue>
             </CipherData>
          </EncryptedData>
       </Security>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
       [omitted]
    </SOAP-ENV:Body>
 </SOAP-ENV:Envelope>
```

単純なバリエーションとして、前のセクションの手順を考えてみます。手順 4 で以下を実行し、他には変更しないものとします。

ObjectScript

```
 set enckey=##class(%XML.Security.EncryptedKey).CreateX509(credset)
```

この場合、クライアントからのメッセージには、暗号化された本文および暗号化された <UsernameToken> が含まれています。

```
<?xml version="1.0" encoding="UTF-8" ?>
   <SOAP-ENV:Envelope [parts omitted]>
      <SOAP-ENV:Header>
         <Security xmlns="[parts omitted]oasis-200401-wss-wssecurity-secext-1.0.xsd">
            <EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#">
               <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p">
                  <DigestMethod xmlns="http://www.w3.org/2000/09/xmldsig#"
                        Algorithm="http://www.w3.org/2000/09/xmldsig#sha1">
                  </DigestMethod>
               </EncryptionMethod>
               <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
                  <SecurityTokenReference
                     xmlns="[parts omitted]oasis-200401-wss-wssecurity-secext-1.0.xsd">
                     <KeyIdentifier EncodingType="[parts omitted]#Base64Binary"
                         ValueType="[parts omitted]#ThumbprintSHA1">
                               5a[parts omitted]dM1r6cM=
                     </KeyIdentifier>
                  </SecurityTokenReference>
               </KeyInfo>
               <CipherData>
                  <CipherValue>TB8uavpr[parts omitted]nZBiMCcg==</CipherValue>
               </CipherData>
               <ReferenceList>
                  <DataReference URI="#Enc-43FE435F-D1D5-4088-A343-0E76D154615A"></DataReference>
                  <DataReference URI="#Enc-55FE109A-3C14-42EB-822B-539E380EDE48"></DataReference>
               </ReferenceList>
            </EncryptedKey>
            <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#"
                 Id="Enc-43FE435F-D1D5-4088-A343-0E76D154615A"
                 Type="http://www.w3.org/2001/04/xmlenc#Element">
               <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc">
               </EncryptionMethod>
               <CipherData>
                  <CipherValue>G+X7dqI[parts omitted]nojroQ==</CipherValue>
               </CipherData>
            </EncryptedData>
         </Security>
      </SOAP-ENV:Header>
      <SOAP-ENV:Body>
         <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#"
                  Id="Enc-55FE109A-3C14-42EB-822B-539E380EDE48"
                  Type="http://www.w3.org/2001/04/xmlenc#Content">
            <EncryptionMethod Algorithm="[parts omitted]aes128-cbc"></EncryptionMethod>
            <CipherData>
               <CipherValue>YJbzyi[parts omitted]NhJoln==</CipherValue>
            </CipherData>
         </EncryptedData>
      </SOAP-ENV:Body>
   </SOAP-ENV:Envelope>
```

前の例と比べると、この場合、<EncryptedKey> 要素には、2 つの <EncryptedData> 要素への参照が含まれています。一方は、セキュリティ・ヘッダの <EncryptedData> 要素で、これには <UsernameToken> が含まれています。この参照は、手動で作成および追加されました。もう一方は、SOAP 本文の <EncryptedData> 要素です。この参照は、自動で追加されました。

9

デジタル・シグニチャの追加

このトピックでは、InterSystems IRIS Web サービスおよび Web クライアントによって送信される SOAP メッセージにデジタル・シグニチャを追加する方法について説明します。

通常、暗号化と署名の両方を実行します。このトピックでは、説明を簡単にするために、署名のみについて説明します。暗号化と署名の組み合わせに関する詳細は、"暗号化と署名の組み合わせ" のトピックを参照してください。

"暗号化および署名への派生キー・トークンの使用" のトピックでは、さらに別の方法で SOAP メッセージにデジタル・シグニチャを追加する方法について説明します。

9.1 デジタル・シグニチャの概要

デジタル・シグニチャを使用して、メッセージの変更を検出したり、リストされているエンティティによってメッセージの特定の部分が実際に生成されたことを検証したりできます。従来の手書きの署名と同様に、デジタル・シグニチャは、ドキュメントの作成者のみが作成できるドキュメントへの追加物で、簡単に偽造することはできません。

SOAP メッセージのデジタル・シグニチャに対する InterSystems IRIS サポートは、WS-Security 1.1 を基盤としています。また、WS-Security は、XML シグニチャ仕様に従います。後者の仕様に従って、XML ドキュメントに署名する方法は、以下のとおりです。

1. ダイジェスト関数を使用して、ドキュメントの複数の部分のハッシュ値を計算します。
2. ダイジェスト値を連結します。
3. 秘密鍵を使用して、連結されたダイジェストを暗号化します(これは、ユーザのみが実行できる計算です)。
4. 以下の情報を含む `<Signature>` 要素を作成します。
   - 署名済み部分への参照 (このシグニチャを適用するメッセージの部分を示します)。
   - 暗号化されたダイジェスト値。
   - 暗号化されたダイジェスト値の解読に使用する公開鍵を受信者が特定できる情報。

     この情報は、`<Signature>` 要素内に含めることができます。または、`<Signature>` 要素は、X.509 証明書または署名済みの SAML アサーションを格納しているバイナリ・セキュリティ・トークンへの直接参照を含めることができます。後者の場合、`<Signature>` 要素を追加する前にセキュリティ・トークンをメッセージに追加する必要があります。

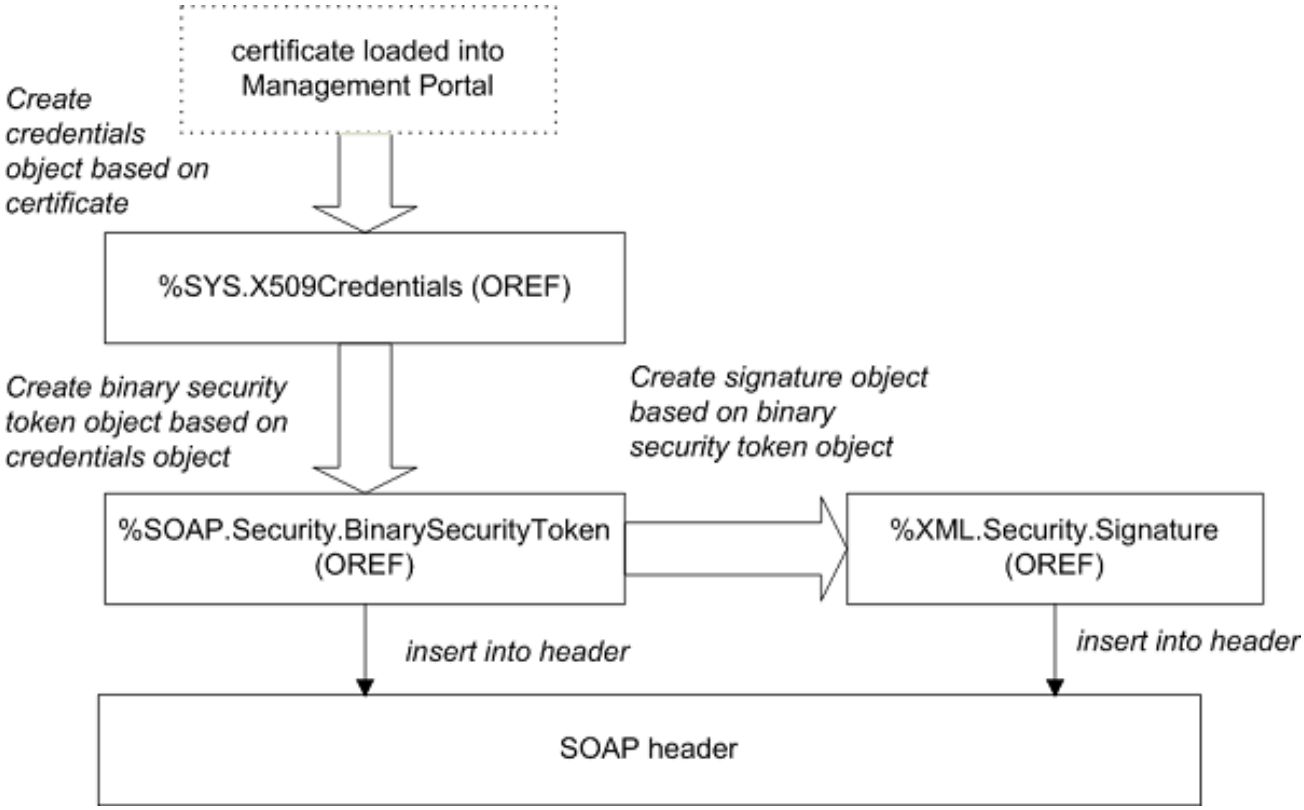
     この情報を使用すると、ユーザが公開鍵と秘密鍵のペアの所有者であることを受信者が検証することもできます。

"暗号化および署名への派生キー・トークンの使用" のトピックでは、さらに別の方法で SOAP メッセージにデジタル・シグニチャを追加する方法について説明します。メッセージ自体の詳細はさまざまですが、一般的なプロセスは同じで、XML 署名仕様に従います。つまり、署名済みの部分のダイジェストを生成し、そのダイジェストを暗号化してから、受信者がシグニチャを検証して暗号化されたダイジェストを解読できる情報を含む <Signature> 要素を組み込みます。

9.2 デジタル・シグニチャの追加

SOAP メッセージをデジタル署名するには、次に説明する基本的な手順を使用するか、またはこのトピックの次の部分で説明するバリエーションを使用します。

まず、次の図にプロセスを示します。



詳細なプロセスは以下のとおりです。

1. 必要に応じて、%soap.inc インクルード・ファイルを組み込みます。このファイルには、使用する可能性のあるマクロが定義されています。
2. 任意のセキュリティ・ヘッダ要素に署名する場合、そのセキュリティ・ヘッダ要素を作成します。以下はその例です。

   ObjectScript

   ```
   set utoken=##class(%SOAP.Security.UsernameToken).Create("_SYSTEM","SYS")
   ```

3. "プログラムによる資格情報セットの取得" の説明に従って、%SYS.X509Credentials のインスタンスを作成します。この InterSystems IRIS 資格情報セットには、独自の証明書を格納する必要があります。また、秘密鍵のパスワードを入力する必要があります (まだロードされていない場合)。以下はその例です。

ObjectScript

```
Set x509alias = "servercred"
Set pwd = "mypassword"
Set credset = ##class(%SYS.X509Credentials).GetByAlias(x509alias,mypassword)
```

4. その資格情報セットに関連付けられている証明書を含むバイナリ・セキュリティ・トークンを作成します。これには、%SOAP.Security.BinarySecurityToken の CreateX509Token() クラス・メソッドを呼び出します。以下はその例です。

   ObjectScript

   ```
   set bst=##class(%SOAP.Security.BinarySecurityToken).CreateX509Token(credset)
   ```

   このメソッドは、<BinarySecurityToken> ヘッダ要素を表す %SOAP.Security.BinarySecurityToken のインスタンスを返します。

5. WS-Security ヘッダ要素にこのトークンを追加します。そのためには、Web クライアントまたは Web サービスの SecurityOut プロパティの AddSecurityElement() メソッドを呼び出します。このメソッドの引数には、作成したトークンを使用します。以下はその例です。

   ObjectScript

   ```
   do ..SecurityOut.AddSecurityElement(bst)
   ```

6. バイナリ・セキュリティ・トークンに基づいて、<Signature> 要素を作成します。これには、%XML.Security.Signature の CreateX509() クラス・メソッドを呼び出します。以下はその例です。

   ObjectScript

   ```
   set dsig=##class(%XML.Security.Signature).CreateX509(bst)
   ```

   このメソッドは、<Signature> ヘッダ要素を表す %XML.Security.Signature のインスタンスを返します。<Signature> 要素は、メッセージの既定の部分に適用されます。また、異なる部分を指定できます。

   最後に、このメソッドには、以下のシグニチャがあります。

   ```
   classmethod CreateX509(credentials As %SYS.X509Credentials = "",
                          signatureOptions As %Integer,
                          referenceOption As %Integer,
                          Output status As %Status) as %XML.Security.Signature
   ```

   以下はその説明です。

   - credentials は、%SYS.X509Credentials インスタンス、%SAML.Assertion インスタンス、または %SOAP.Security.BinarySecurityToken インスタンスのいずれかです。
   - signatureOptions は、署名する部分を指定します。このオプションは、"メッセージの特定部分へのデジタル・シグニチャの適用" で説明します。
   - referenceOption は、作成する参照のタイプを指定します。詳細は、"X.509 資格情報の参照オプション" を参照してください。
   - status は、メソッドが成功したかどうかを示します。

7. WS-Security ヘッダ要素にデジタル・シグニチャを追加します。そのためには、Web クライアントまたは Web サービスの SecurityOut プロパティの AddSecurityElement() メソッドを呼び出します。引数に、前の手順で作成したシグニチャ・オブジェクトを指定します。以下はその例です。

   ObjectScript

   ```
   do ..SecurityOut.AddSecurityElement(dsig)
   ```

8. SOAP メッセージを送信します。"セキュリティ・ヘッダ要素の追加" の一般的な手順を参照してください。

9.2.1 例

この例では、その応答メッセージに署名する Web サービスを示します。

個々の環境でこの例のとおり機能するようにするには、まず次の手順を実行します。

- サーバ用の証明書を作成します。
- サーバ側で InterSystems IRIS にこの証明書をロードし、`servercred` という名前で資格情報を作成します。このとき、(Web サービスがその応答メッセージに署名する際、パスワードを指定しないで済むように) 秘密鍵ファイルもロードし、パスワードを指定します。

  Web サービスは、この正確な名前で InterSystems IRIS 資格情報セットを参照します。

Class Definition

```
Class DSig.DivideWS Extends %SOAP.WebService
{

///  Name of the Web service.
Parameter SERVICENAME = "DigitalSignatureDemo";

///  SOAP namespace for the Web service
Parameter NAMESPACE = "http://www.myapp.org";

/// use in documentation
Method Divide(arg1 As %Numeric = 2, arg2 As %Numeric = 8) As %Numeric [ WebMethod ]
{
 Do ..SignResponses()
 Try {
  Set ans=arg1 / arg2
  }Catch{
    Do ..ApplicationError("division error")
  }
 Quit ans
}

/// use in documentation
/// signs and includes a binary security token
Method SignResponses()
{
 //Add timestamp because that's commonly done
 Set ts=##class(%SOAP.Security.Timestamp).Create()
 Do ..SecurityOut.AddSecurityElement(ts)

 //access previously stored server certificate & private key file
 //no need to use private key file password, because that has been saved
 Set x509alias = "servercred"
 Set cred = ##class(%SYS.X509Credentials).GetByAlias(x509alias)
 set bst=##class(%SOAP.Security.BinarySecurityToken).CreateX509Token(cred)
 do ..SecurityOut.AddSecurityElement(bst)

 //Create WS-Security Signature object
 Set signature=##class(%XML.Security.Signature).CreateX509(bst)

 //Add WS-Security Signature object to the outbound message
 Do ..SecurityOut.AddSecurityElement(signature)
 Quit
}


///  Create our own method to produce application specific SOAP faults.
Method ApplicationError(detail As %String)
{
    Set fault=##class(%SOAP.Fault).%New()
    Set fault.faultcode=$$$FAULTServer
    Set fault.detail=detail
    Set fault.faultstring="Application error"
    // ReturnFault must be called to send the fault to the client.
    // ReturnFault will not return here.
    Do ..ReturnFault(fault)
}
```

```
}
```

9.3 証明書とシグニチャを使用するその他の方法

前のセクションで説明した基本的な手順では、シリアル化された Base 64 エンコード形式を含む <BinarySecurityToken> を使用します。証明書を組み込む代わりに、証明書を識別する情報を使用できます。または、署名済みの SAML アサーション内に証明書を入れることもできます。このセクションでは、これらのバリエーションについて説明します。

9.3.1 バリエーション : 証明書を指定する情報の使用

証明書をメッセージに組み込む代わりに、証明書を識別する情報を使用できます。受信者はこの情報を使用して、適切な場所から証明書を取得します。そのためには、前のセクションの手順を使用しますが、以下の変更点があります。

- 手順 4 と 5 をスキップします。つまり、<BinarySecurityToken> を追加しないでください。
- 手順 6 (シグニチャの作成) で、CreateX509() の最初の引数として手順 1 の資格情報セットを使用します (バイナリ・セキュリティ・トークンは使用しません)。以下はその例です。

  ObjectScript

  ```
  set dsig=##class(%XML.Security.Signature).CreateX509(credset,,referenceOption)
  ```

  3 番目の引数 (referenceOption) では、<Signature> 要素が証明書を使用する方法を指定できます。

  (このバリエーションで実行しているように) 最初の引数として資格情報セットを指定する場合、referenceOption の既定値は $$$SOAPWSReferenceThumbprint です。必要に応じて、"X.509 資格情報の参照オプション" の説明に従って、値を指定します。$$$SOAPWSReferenceDirect 以外の任意の値を使用できます。

9.3.1.1 例

この例は、このトピックの前の例のバリエーションです。

Class Member

```
Method SignResponses()
{
 //Add timestamp because that's commonly done
 Set ts=##class(%SOAP.Security.Timestamp).Create()
 Do ..SecurityOut.AddSecurityElement(ts)

 //access previously stored server certificate & private key file
 //no need to use private key file password, because that has been saved
 Set x509alias = "servercred"
 Set cred = ##class(%SYS.X509Credentials).GetByAlias(x509alias)

 //Create WS-Security Signature object
 Set signature=##class(%XML.Security.Signature).CreateX509(cred)

 //Add WS-Security Signature object to the outbound message
 Do ..SecurityOut.AddSecurityElement(signature)
 Quit
}
```

この場合、この Web サービスは、次のような応答メッセージを送信します。

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope [parts omitted]>
  <SOAP-ENV:Header>
    <Security xmlns="[parts omitted]oasis-200401-wss-wssecurity-secext-1.0.xsd">
```

```
      <Timestamp xmlns="[parts omitted]oasis-200401-wss-wssecurity-utility-1.0.xsd"
                 wsu:Id="Timestamp-48CEE53E-E6C3-456C-9214-B7D533B2663F">
        <Created>2010-03-19T14:35:06Z</Created>
        <Expires>2010-03-19T14:40:06Z</Expires>
      </Timestamp>
      <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
        <SignedInfo>
          <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
          </CanonicalizationMethod>
          <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha256"></SignatureMethod>

            <Reference URI="#Timestamp-48CEE53E-E6C3-456C-9214-B7D533B2663F">
              <Transforms>
                <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"></Transform>
              </Transforms>
              <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></DigestMethod>
                <DigestValue>waSMFeYMruQn9XHx85HqunhMGIA=</DigestValue>
            </Reference>
            <Reference URI="#Body-73F08A5C-0FFD-4FE9-AC15-254423DBA6A2">
              <Transforms>
                <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"></Transform>
              </Transforms>
              <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></DigestMethod>
                <DigestValue>wDCqAzy5bLKKF+Rt0+YV/gxTQws=</DigestValue>
            </Reference>
          </SignedInfo>
          <SignatureValue>j6vtht/[parts omitted]trCQ==</SignatureValue>
          <KeyInfo>
            <SecurityTokenReference
            xmlns="[parts omitted]oasis-200401-wss-wssecurity-secext-1.0.xsd">
              <KeyIdentifier EncodingType="[parts omitted]#Base64Binary"
                             ValueType="[parts omitted]#ThumbprintSHA1">
                   WeCnU2sMyOXfHH8CHTLjNTQQnGQ=
              </KeyIdentifier>
            </SecurityTokenReference>
          </KeyInfo>
        </Signature>
      </Security>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body wsu:Id="Body-73F08A5C-0FFD-4FE9-AC15-254423DBA6A2">
      [omitted]
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

9.3.2 バリエーション :署名済みの SAML アサーションの使用

署名済みの SAML アサーションの証明書を使用するデジタル・シグニチャを追加するには、以下の手順を実行します。

1. 必要に応じて、%soap.inc インクルード・ファイルを組み込みます。このファイルには、使用する可能性のあるマクロが定義されています。
2. 任意のセキュリティ・ヘッダ要素に署名する場合、そのセキュリティ・ヘッダ要素を作成します。以下はその例です。

   ObjectScript

   ```
   set utoken=##class(%SOAP.Security.UsernameToken).Create("_SYSTEM","SYS")
   ```

3. Holder-of-key メソッドを使用する <SubjectConfirmation> 要素を使用して、署名済みの SAML アサーションを作成します。"SAML トークンの作成と追加" を参照してください。
4. <Signature> 要素を作成します。そのためには、CreateX509() クラス・メソッドの最初の引数として署名済みの SAML アサーションを使用します。以下はその例です。

   ObjectScript

   ```
   set signature=##class(%XML.Security.EncryptedKey).CreateX509(signedassertion)
   ```

5. WS-Security ヘッダ要素にデジタル・シグニチャを追加します。そのためには、Web クライアントまたは Web サービスの SecurityOut プロパティの AddSecurityElement() メソッドを呼び出します。引数に、前の手順で作成したシグニチャ・オブジェクトを指定します。以下はその例です。

ObjectScript

```
do ..SecurityOut.AddSecurityElement(dsig)
```

6. SOAP メッセージを送信します。"セキュリティ・ヘッダ要素の追加" の一般的な手順を参照してください。

9.4 メッセージの特定部分へのデジタル・シグニチャの適用

既定では、デジタル・シグニチャを作成し、WS-Security ヘッダ要素に追加する場合、シグニチャは、SOAP 本文、ヘッダの <Timestamp> 要素 (ある場合)、およびすべての WS-Addressing ヘッダ要素に適用されます。

シグニチャを適用する部分を指定するには、1 つのバリエーションで前に説明した手順のいずれかを使用します。つまり、シグニチャを作成するとき、2 番目の引数 (signatureOptions) を使用して、署名するメッセージ部分を指定します。この引数は、以下のマクロ (%soap.inc ファイルに保存) のいずれかのバイナリの組み合わせとして指定します。

- $$$SOAPWSIncludeNone
- $$$SOAPWSIncludeDefault ($$$SOAPWSIncludeSoapBody + $$$SOAPWSIncludeTimestamp + $$$SOAPWSIncludeAddressing に相当)
- $$$SOAPWSIncludeSoapBody
- $$$SOAPWSIncludeTimestamp
- $$$SOAPWSIncludeAddressing
- $$$SOAPWSIncludeAction
- $$$SOAPWSIncludeFaultTo
- $$$SOAPWSIncludeFrom
- $$$SOAPWSIncludeMessageId
- $$$SOAPWSIncludeRelatesTo
- $$$SOAPWSIncludeReplyTo
- $$$SOAPWSIncludeTo
- $$$SOAPWSIncludeRMHeaders ("WS-ReliableMessaging の使用" を参照)

マクロを組み合わせるときには、プラス記号 (+) とマイナス記号 (-) を使用します。以下はその例です。

```
$$$SOAPWSIncludeSoapBody+$$$SOAPWSIncludeTimestamp
```

注釈　これらのオプションは、CreateX509() メソッドと Create() メソッドの両方に適用されます。後者については、"暗号化および署名への派生キー・トークンの使用" を参照してください。

以下はその例です。

ObjectScript

```
set ts=##class(%SOAP.Security.Timestamp).Create()
do ..SecurityOut.AddSecurityElement(ts)
set x509alias = "servercred"
set cred = ##class(%SYS.X509Credentials).GetByAlias(x509alias)

set parts=$$$SOAPWSIncludeSoapBody + $$$SOAPWSIncludeTimestamp
set signature=##class(%XML.Security.Signature).CreateX509(cred,parts)
```

9.5 ダイジェスト・メソッドの指定

既定では、シグニチャのダイジェスト値は、SHA-1 アルゴリズムを使用して計算され、セキュリティ・ヘッダの <Signature> 要素は以下の例のようになります。

```
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></DigestMethod>
<DigestValue>waSMFeYMruQn9XHx85HqunhMGIA=</DigestValue>
```

シグニチャには異なるダイジェスト・メソッドを指定することもできます。そのためには、%XML.Security.Signature のインスタンスの SetDigestMethod() メソッドを呼び出します。引数には、以下のマクロ (%soap.inc ファイルに保存) のいずれかを使用します。

- $$$SOAPWSsha1 (既定)
- $$$SOAPWSsha256
- $$$SOAPWSsha384
- $$$SOAPWSsha512

以下はその例です。

```
do sig.SetDigestMethod($$$SOAPWSsha256)
```

9.6 シグニチャ・メソッドの指定

既定では、シグニチャ値は、RSA-SHA256 アルゴリズムを使用して計算され、セキュリティ・ヘッダの <Signature> 要素は以下の例のようになります。

```
<SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha256"></SignatureMethod>
...
<SignatureValue>J+gACmdjkJxaq2hJqA[parts omitted]</SignatureValue>
```

シグニチャ・メソッドには異なるアルゴリズムを指定することもできます。そのためには、%XML.Security.Signature のインスタンスの SetSignatureMethod() メソッドを呼び出します。引数には、以下のマクロ (%soap.inc ファイルに保存) のいずれかを使用します。

- $$$SOAPWSrsasha1
- $$$SOAPWSrsasha256 (既定)
- $$$SOAPWSrsasha384
- $$$SOAPWSrsasha512
- $$$SOAPWShmacsha256
- $$$SOAPWShmacsha384
- $$$SOAPWShmacsha512

以下はその例です。

```
do sig.SetSignatureMethod($$$SOAPWSrsasha512)
```

既定のシグニチャ・アルゴリズムは変更できないことに注意してください。変更するには、管理ポータルにアクセスし、[システム管理]、[セキュリティ]、[システムセキュリティ]、[システムワイドセキュリティパラメータ] の順にクリックします。既定のシグニチャ・アルゴリズムを指定するオプションには、[既定のシグニチャ・ハッシュ] というラベルが付いています。

9.7 〈KeyInfo〉 の正規化メソッドの指定

既定では、<KeyInfo> 要素が Exclusive XML Canonicalization によって正規化され、<KeyInfo> 要素は以下を含みます。

```
<CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
```

その代わりに、Inclusive XML Canonicalization によってこの要素を正規化する場合は、以下の手順を実行します。

```
 Set sig.SignedInfo.CanonicalizationMethod.Algorithm=$$$SOAPWSc14n
```

sig は、%XML.Security.Signature のインスタンスです。

この場合、<KeyInfo> は以下の要素で構成されます。

```
<CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315">
```

9.8 シグニチャの確認の追加

WS-Security 1.1 <SignatureConfirmation> 機能は、受信した SOAP メッセージが、Web クライアントによって送信された元の要求への応答で生成されたことを、Web クライアントが確認できるようにします。クライアント要求は通常、署名されますが、署名は必須ではありません。このメカニズムでは、Web サービスは、セキュリティ・ヘッダ要素に <SignatureConfirmation> 要素を追加し、Web クライアントは、その <SignatureConfirmation> 要素を検証できます。

Web サービスの場合、セキュリティ・ヘッダ要素に <SignatureConfirmation> 要素を追加するには、次の操作を実行します。

1. Web サービスの WSAddSignatureConfirmation() メソッドを呼び出します。引数に、セキュリティ・ヘッダ要素の主要なシグニチャを指定します。以下はその例です。

   ```
    do ..WSAddSignatureConfirmation(sig)
   ```

2. 通常どおり、SOAP メッセージを送信します。"セキュリティ・ヘッダ要素の追加" の一般的な手順を参照してください。

このメソッドでは、発信メッセージに WS-Security 1.1 <SignatureConfirmation> 要素が追加されます。SecurityIn で受信された各 <Signature> の SecurityOut プロパティに <SignatureConfirmation> 要素が追加されます。

SecurityIn にシグニチャが含まれていない場合、<SignatureConfirmation> 要素は、WS-Security 1.1 によって要求されているとおり Value 属性なしで追加されます。

<SignatureConfirmation> 要素を検証する方法については、"シグニチャの確認のチェック" を参照してください。

10

暗号化および署名への派生キー・トークンの使用

InterSystems IRIS は、WS-SecureConversation 1.4 で定義されているように <DerivedKeyToken> 要素をサポートします。前の 3 つのトピックで説明した方法の代替として、暗号化および署名に <DerivedKeyToken> 要素を作成して使用できます。

通常、暗号化と署名の両方を実行します。このトピックでは、説明を簡単にするために、これらのタスクを別々に説明します。暗号化と署名の組み合わせに関する詳細は、"暗号化と署名の組み合わせ" を参照してください。

10.1 概要

<DerivedKeyToken> 要素は、同じ対称鍵を生成するために、送信者と受信者が独立して使用できる情報を保持するためのものです。両者は、この対称鍵を使用して、SOAP メッセージの指定の部分の暗号化、署名、または両方のアクションを実行できます。

<DerivedKeyToken> を生成して使用するには、以下の手順を実行します。

1. 一時的に使用する対称鍵を生成します。
2. メッセージの送信先のエンティティの公開鍵を使用して、対称鍵を暗号化します。これにより、<EncryptedKey> 要素が作成されます。

   公開鍵は、そのエンティティからの要求メッセージに含まれている X.509 証明書から取得できます。または、事前に取得することができます。
3. P_SHA1 アルゴリズムを使用して、元の対称鍵から新しい対称鍵を計算します。

   これにより、<EncryptedKey> 要素を参照する <DerivedKeyToken> 要素が作成されます。
4. 新しい対称鍵を使用して暗号化または署名を行います。

   分析するデータを最小化するために、これらのアクティビティには異なる対称鍵を使用することをお勧めします。
5. <EncryptedKey> 要素と <DerivedKeyToken> をメッセージに含めます。

InterSystems IRIS では、ある派生キー・トークンが別の派生キー・トークンに基づくこともできます。

10.2 <DerivedKeyToken> の作成と追加

参考として、このセクションでは、後述の各セクションで必要な一般的なアクティビティについて説明します。また、<DerivedKeyToken> を作成し、これを WS-Security ヘッダに追加する方法について説明します。以下の手順または各サブセクションで説明するバリエーションを使用できます。

1. 必要に応じて、%soap.inc インクルード・ファイルを組み込みます。このファイルには、使用する可能性のあるマクロが定義されています。
2. メッセージの送信先のエンティティの資格情報セットを取得します。"プログラムによる資格情報セットの取得" を参照してください。

   以下はその例です。

   ObjectScript

   ```
   Set x509alias = "servernopassword"
   Set credset = ##class(%SYS.X509Credentials).GetByAlias(x509alias)
   ```

3. 資格情報セットに基づいて、暗号化キーを作成します。そのためには、%XML.Security.EncryptedKey の CreateX509() クラス・メソッドを呼び出し、2 番目の引数に $$$SOAPWSEncryptNone を指定します。以下はその例です。

   ObjectScript

   ```
   set enckey=##class(%XML.Security.EncryptedKey).CreateX509(credset,$$$SOAPWSEncryptNone)
   ```

   このメソッドは、対称鍵を生成し、<EncryptedKey> ヘッダ要素を表す %XML.Security.EncryptedKey のインスタンスを返します。このヘッダ要素には、指定された資格情報セット内にある公開鍵で暗号化された対称鍵が含まれます。

   派生キーの基盤として使用する暗号化キーを作成する場合、常に $$$SOAPWSEncryptNone または "" を CreateX509() の 2 番目の引数として指定します。
4. 必要に応じて、別のアルゴリズムを使用するように暗号化キーのインスタンスを変更します。"ブロック暗号化アルゴリズムの指定" および "鍵転送アルゴリズムの指定" を参照してください。
5. WS-Security ヘッダ要素に <EncryptedKey> 要素を追加します。そのためには、Web クライアントまたは Web サービスの SecurityOut プロパティの AddSecurityElement() メソッドを呼び出します。追加する要素について、%XML.Security.EncryptedKey のインスタンスを指定します。

   以下はその例です。

   ObjectScript

   ```
   do ..SecurityOut.AddSecurityElement(enckey)
   ```

6. 暗号化キーに基づいて、派生キー・トークンを作成します。これには、%SOAP.WSSC.DerivedKeyToken の Create() メソッドを呼び出します。このメソッドは、以下の 2 つの引数を取ります。
   a. 基盤として使用する暗号化キー。
   b. 派生キーがその暗号化キーを参照する方法を指定する参照オプション。この基本的な手順では、$$$SOAPWSReferenceEncryptedKey を使用します。

   以下はその例です。

ObjectScript

```
set refopt=$$$SOAPWSReferenceEncryptedKey
set dkenc=##class(%SOAP.WSSC.DerivedKeyToken).Create(enckey,refopt)
```

このメソッドは、<DerivedKeyToken> 要素を表す %SOAP.WSSC.DerivedKeyToken のインスタンスを返します。

7. WS-Security ヘッダ要素に <DerivedKeyToken> 要素を追加します。そのためには、Web クライアントまたは Web サービスの SecurityOut プロパティの AddSecurityElement() メソッドを呼び出します。追加する要素に対して、%SOAP.WSSC.DerivedKeyToken のインスタンスを指定します。

   以下はその例です。

   ObjectScript

   ```
   do ..SecurityOut.AddSecurityElement(dkenc)
   ```

8. SOAP メッセージを送信します。"セキュリティ・ヘッダ要素の追加" の一般的な手順を参照してください。

10.2.1 バリエーション : 暗黙的な <DerivedKeyToken> の作成

暗黙的な <DerivedKeyToken> も作成できます。これは、<DerivedKeyToken> を参照する簡単な方法です。この方法は、以下のようになります。

- <DerivedKeyToken> は、メッセージに含まれません。
- <DerivedKeyToken> を使用する要素内において、<SecurityTokenReference> 要素は、Nonce 属性を指定します。この属性には、<DerivedKeyToken> に使用される Nonce の値が含まれます。これは、派生キー・トークンが暗黙的に使用され、参照されたトークンから派生していることをメッセージの受信者に示します。

暗黙的 <DerivedKeyToken> を作成するには、前述した一般的な手順を使用します。ただし、2 つ変更点があります。

1. 派生キー・トークンのインスタンスに対して、Implied プロパティを 1 に設定します。

   以下はその例です。

   ObjectScript

   ```
   set dkt.Implied=1
   ```

2. WS-Security ヘッダ要素に <DerivedKeyToken> 要素を追加しないでください。

<DerivedKeyToken> がメッセージに含まれている場合とまったく同じようにこのトークンを使用します。

10.2.2 バリエーション : <EncryptedKey> の SHA1 ハッシュの参照

このバリエーション (Web サービス上でのみ使用可能) では、送信者は、メッセージに <EncryptedKey> 要素を含めませんが、代わりにこのキーの SHA1 ハッシュを参照します。Web サービスは、着信メッセージで受信した <EncryptedKey> 要素を参照できます。

前述の一般的な手順を使用しますが、以下の変更点があります。

- 手順 2-4 はオプションです。
- 手順 5 を省略します (<EncryptedKey> は追加しないでください)。

- 手順 6 で、Create() を使用して派生キー・トークンを作成するとき、クライアントから受信した <EncryptedKey> を使用するために、最初の引数を省略します。または、<EncryptedKey> を作成済みの場合、これを最初の引数として使用します。

  2 番目の引数に $$$SOAPWSReferenceEncryptedKeySHA1 を指定します。

  例えば、Web クライアントから受信したメッセージの最初の <EncryptedKey> 要素を使用するには、以下のようにします。

  ObjectScript

```
set refopt=$$$SOAPWSReferenceEncryptedKeySHA1
set dkenc=##class(%SOAP.WSSC.DerivedKeyToken).Create(,refopt)
```

10.3 暗号化への <DerivedKeyToken> の使用

暗号化に <DerivedKeyToken> を使用するには、以下の手順を実行します。

1. 1 つ以上のセキュリティ・ヘッダ要素を暗号化する場合、そのセキュリティ・ヘッダ要素を作成します。
2. "<DerivedKeyToken> の作成と追加" の説明に従って、<DerivedKeyToken> を作成し、これを WS-Security ヘッダに追加します。

   この手順では、<DerivedKeyToken> の基盤になっている <EncryptedKey> 要素も作成および追加されることに注意してください。
3. 暗号化する各要素に対して、その要素に基づいて <EncryptedData> 要素を作成します。そのためには、%XML.Security.EncryptedData の Create() クラス・メソッドを呼び出します。この手順では、以下の引数を指定します。

   a. 派生キー・トークン。

   b. 暗号化するアイテム。本文を暗号化するには、この引数を省略します。

   c. <EncryptedData> 要素が <DerivedKeyToken> を参照する方法を指定するマクロ。このシナリオで現在サポートされている値は $$$SOAPWSReferenceDerivedKey のみです。

   例えば、<UsernameToken> を暗号化するには、以下のようにします。

   ObjectScript

```
set refopt=$$$SOAPWSReferenceDerivedKey
set encryptedData=##class(%XML.Security.EncryptedData).Create(dkenc,userToken,refopt)
```

   または、本文を暗号化するには、以下のようにします。

   ObjectScript

```
set refopt=$$$SOAPWSReferenceDerivedKey
set encryptedData=##class(%XML.Security.EncryptedData).Create(dkenc,,refopt)
```

4. <ReferenceList> 要素を作成します。そのためには、%XML.Security.ReferenceList クラスの %New() メソッドを呼び出します。以下はその例です。

   ObjectScript

```
set reflist=##class(%XML.Security.ReferenceList).%New()
```

5. この <ReferenceList> 内で、<EncryptedData> 要素を指す <ReferenceList> を作成します。そのためには、各 <EncryptedData> に対して、以下の手順を実行します。

   a. %XML.Security.DataReference の Create() クラス・メソッドを呼び出し、暗号化されたデータのインスタンスを引数として指定します。このメソッドは %XML.Security.DataReference のインスタンスを返します。

   b. 参照リストのインスタンスの AddReference() メソッドを呼び出し、データ参照のインスタンスを引数として指定します。

   以下はその例です。

   ObjectScript

   ```
   set dataref=##class(%XML.Security.DataReference).Create(encdata)
   do reflist.AddReference(dataref)
   set dataref2=##class(%XML.Security.DataReference).Create(encdata2)
   do reflist.AddReference(dataref2)
   ```

6. WS-Security ヘッダ要素に <ReferenceList> 要素を追加します。そのためには、Web クライアントまたは Web サービスの SecurityOut プロパティの AddSecurityElement() メソッドを呼び出します。追加する要素に対して、参照リストのインスタンスを指定します。

   以下はその例です。

   ObjectScript

   ```
   do ..SecurityOut.AddSecurityElement(reflist)
   ```

7. セキュリティ・ヘッダ要素を暗号化した場合、これらを WS-Security ヘッダ要素に追加します。そのためには、Web クライアントまたは Web サービスの SecurityOut プロパティの AddSecurityElement() メソッドを呼び出します。この場合、以下の 2 つの引数が必要です。

   a. セキュリティ・ヘッダ要素 (この要素から生成した %XML.Security.EncryptedData ではありません)。

   b. 参照リストのインスタンス。2 番目の引数は、最初の引数が指定するアイテムの配置場所を指定します。引数が A,B の場合、InterSystems IRIS は、A が B の後であることを確認します。これを指定して、受信側が参照リストを最初に処理し、これに依存する暗号化されたセキュリティ・ヘッダ要素を後で処理するようにします。

   以下はその例です。

   ObjectScript

   ```
   do client.SecurityOut.AddSecurityElement(userToken,reflist)
   ```

   SOAP 本文のみを暗号化した場合、<EncryptedData> 要素が <Body> の子として自動的に含まれます。

8. SOAP メッセージを送信します。"セキュリティ・ヘッダ要素の追加" の一般的な手順を参照してください。

例えば、以下のクライアント側のコードは、SOAP 本文と <UsernameToken> の両方を暗号化します。

ObjectScript

```
// Create UsernameToken
set userToken=##class(%SOAP.Security.UsernameToken).Create("_SYSTEM","SYS")

// get credentials for encryption
set cred = ##class(%SYS.X509Credentials).GetByAlias("servercred")

// get EncryptedKey element to encrypt <UsernameToken)
// $$$SOAPWSEncryptNone means that this key does not encrypt the body
set enckey=##class(%XML.Security.EncryptedKey).CreateX509(cred,$$$SOAPWSEncryptNone)
//add to WS-Security Header
do client.SecurityOut.AddSecurityElement(enckey)
```

```
  // get derived key to use for encryption
  // second argument specifies how the derived key
  // refers to the key on which it is based
  set dkenc=##class(%SOAP.WSSC.DerivedKeyToken).Create(enckey,
     $$$SOAPWSReferenceEncryptedKey)
  //add to WS-Security Header
  do client.SecurityOut.AddSecurityElement(dkenc)

  // create <EncryptedData> element to contain <UserToken>
  set encdata=##class(%XML.Security.EncryptedData).Create(dkenc,userToken,
     $$$SOAPWSReferenceDerivedKey)

  // create <EncryptedData> element to contain SOAP body
  set encdata2=##class(%XML.Security.EncryptedData).Create(dkenc,"",
     $$$SOAPWSReferenceDerivedKey)

  // create <ReferenceList> with <DataReference> elements that
  // point to these two <EncryptedData> elements
  set reflist=##class(%XML.Security.ReferenceList).%New()
  set dataref=##class(%XML.Security.DataReference).Create(encdata)
  do reflist.AddReference(dataref)
  set dataref2=##class(%XML.Security.DataReference).Create(encdata2)
  do reflist.AddReference(dataref2)

  // add <ReferenceList> to WS-Security header
  do client.SecurityOut.AddSecurityElement(reflist)
  // add encrypted <UserName> to security header;
  // 2nd argument specifies position
  do client.SecurityOut.AddSecurityElement(userToken,reflist)

  // encrypted SOAP body is handled automatically
```

このクライアントは、次のようなメッセージを送信します。

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope [parts omitted]>
  <SOAP-ENV:Header>
    <Security xmlns="[parts omitted]oasis-200401-wss-wssecurity-secext-1.0.xsd">
      <EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#"
                    Id="Id-658202BF-239A-4A8C-A100-BB25579F366B">
        <EncryptionMethod Algorithm="[parts omitted]#rsa-oaep-mgf1p">
          <DigestMethod xmlns="http://www.w3.org/2000/09/xmldsig#"
                        Algorithm="http://www.w3.org/2000/09/xmldsig#sha1">
          </DigestMethod>
        </EncryptionMethod>
        <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
          <SecurityTokenReference xmlns="[parts omitted]oasis-200401-wss-wssecurity-secext-1.0.xsd">
            <KeyIdentifier EncodingType="[parts omitted]#Base64Binary"
                           ValueType="[parts omitted]#ThumbprintSHA1">5afOHv1w7WSXwDyz6F3WdM1r6cM=
            </KeyIdentifier>
          </SecurityTokenReference>
        </KeyInfo>
        <CipherData>
          <CipherValue>tFeKrZKw[parts omitted]r+bx7KQ==</CipherValue>
        </CipherData>
      </EncryptedKey>
      <DerivedKeyToken xmlns="[parts omitted]ws-secureconversation/200512"
                       xmlns:wsc="[parts omitted]ws-secureconversation/200512"
                       wsu:Id="Enc-943C6673-E3F3-48E4-AA24-A7F82CCF6511">
        <SecurityTokenReference xmlns="[parts omitted]oasis-200401-wss-wssecurity-secext-1.0.xsd">
           <Reference URI="#Id-658202BF-239A-4A8C-A100-BB25579F366B"></Reference>
        </SecurityTokenReference>
        <Nonce>GbjRvVNrPtHs0zo/w9Ne0w==</Nonce>
      </DerivedKeyToken>
      <ReferenceList xmlns="http://www.w3.org/2001/04/xmlenc#">
        <DataReference URI="#Enc-358FB189-81B3-465D-AFEC-BC28A92B179C"></DataReference>
        <DataReference URI="#Enc-9EF5CCE4-CF43-407F-921D-931B5159672D"></DataReference>
      </ReferenceList>
      <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#"
                     Id="Enc-358FB189-81B3-465D-AFEC-BC28A92B179C"
                     Type="http://www.w3.org/2001/04/xmlenc#Element">
        <EncryptionMethod Algorithm="[parts omitted]#aes256-cbc"></EncryptionMethod>
        <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
           <SecurityTokenReference xmlns="[parts omitted]oasis-200401-wss-wssecurity-secext-1.0.xsd">
            <Reference URI="#Enc-943C6673-E3F3-48E4-AA24-A7F82CCF6511"></Reference>
           </SecurityTokenReference>
        </KeyInfo>
        <CipherData>
          <CipherValue>e4//6aWGqo1dIQ7ZAF[parts omitted]KZcj99N78A==</CipherValue>
        </CipherData>
      </EncryptedData>
    </Security>
  </SOAP-ENV:Header>
```

```
  <SOAP-ENV:Body>
    <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#"
                   Id="Enc-9EF5CCE4-CF43-407F-921D-931B5159672D"
                   Type="http://www.w3.org/2001/04/xmlenc#Content">
       <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes256-cbc">
       </EncryptionMethod>
       <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
         <SecurityTokenReference xmlns="[parts omitted]oasis-200401-wss-wssecurity-secext-1.0.xsd">
           <Reference URI="#Enc-943C6673-E3F3-48E4-AA24-A7F82CCF6511"></Reference>
         </SecurityTokenReference>
       </KeyInfo>
       <CipherData>
         <CipherValue>Q3XxuNjSan[parts omitted]x9AD7brM4</CipherValue>
       </CipherData>
    </EncryptedData>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

別の例として、以下の Web サービスは、着信メッセージで <EncryptedKey> を受信し、これを使用して、応答の部分を暗号化するために使用する <DerivedKeyToken> を生成します。

ObjectScript

```
  // create <DerivedKeyToken> based on first <EncryptedKey> in inbound message;
  // refer to it with SHA1 thumbprint
  set refopt=$$$SOAPWSReferenceEncryptedKeySHA1
  set dkenc=##class(%SOAP.WSSC.DerivedKeyToken).Create(,refopt)
  do ..SecurityOut.AddSecurityElement(dkenc)

  // create <EncryptedData> element to contain SOAP body
  set encdata=##class(%XML.Security.EncryptedData).Create(dkenc,"",
     $$$SOAPWSReferenceDerivedKey)

  // create <ReferenceList> with <DataReference> elements that
  // point to the <EncryptedData> elements
  set reflist=##class(%XML.Security.ReferenceList).%New()
  set dataref=##class(%XML.Security.DataReference).Create(encdata)
  do reflist.AddReference(dataref)

  // add <ReferenceList> to WS-Security header
  do ..SecurityOut.AddSecurityElement(reflist)
```

この Web サービスは、次のようなメッセージを送信します。

```
<SOAP-ENV:Envelope [parts omitted]>
   <SOAP-ENV:Header>
      <Security xmlns="[parts omitted]oasis-200401-wss-wssecurity-secext-1.0.xsd">
         <DerivedKeyToken xmlns="[parts omitted]ws-secureconversation/200512"
                          xmlns:wsc="[parts omitted]ws-secureconversation/200512"
                          wsu:Id="Enc-D69085A9-9608-472D-85F3-44031586AB35">
            <SecurityTokenReference xmlns="[parts omitted]oasis-200401-wss-wssecurity-secext-1.0.xsd"

                                    s01:TokenType="[parts omitted]#EncryptedKey"
                                    xmlns:s01="h[parts omitted]oasis-wss-wssecurity-secext-1.1.xsd">
               <KeyIdentifier EncodingType="[parts omitted]#Base64Binary"
                              [parts omitted]#EncryptedKeySHA1">
                     U8CEWXdUPsIk/r8JT+2KdwU/gSw=
               </KeyIdentifier>
            </SecurityTokenReference>
            <Nonce>nJWyIJUcXXLd4k1tbNg10w==</Nonce>
         </DerivedKeyToken>
         <ReferenceList xmlns="http://www.w3.org/2001/04/xmlenc#">
            <DataReference URI="#Enc-0FF09175-B594-4198-9850-57D40EB66DC3"></DataReference>
         </ReferenceList>
      </Security>
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
      <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#"
                     Id="Enc-0FF09175-B594-4198-9850-57D40EB66DC3"
                     Type="http://www.w3.org/2001/04/xmlenc#Content">
         <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes256-cbc">
         </EncryptionMethod>
         <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
           <SecurityTokenReference xmlns="[parts omitted]oasis-200401-wss-wssecurity-secext-1.0.xsd">

              <Reference URI="#Enc-D69085A9-9608-472D-85F3-44031586AB35"></Reference>
            </SecurityTokenReference>
         </KeyInfo>
         <CipherData>
            <CipherValue>NzI94WnuQU4uBO[parts omitted]xHZpJSA==</CipherValue>
```

```
        </CipherData>
      </EncryptedData>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

10.4 署名への <DerivedKeyToken> の使用

署名に <DerivedKeyToken> を使用するには、以下の手順を実行します。

1. 任意のセキュリティ・ヘッダ要素に署名する場合、そのセキュリティ・ヘッダ要素を作成します。
2. "<DerivedKeyToken> の作成と追加" の説明に従って、<DerivedKeyToken> を作成し、これを WS-Security ヘッダに追加します。

   この手順では、<DerivedKeyToken> の基盤になっている <EncryptedKey> 要素も作成および追加されることに注意してください。
3. 派生キー・トークンに基づいて、<Signature> 要素を作成します。そのためには、%XML.Security.Signature の Create() クラス・メソッドを呼び出します。以下はその例です。

   ObjectScript

   ```
   set dsig=##class(%XML.Security.Signature).Create(dkt)
   ```

   このメソッドは、<Signature> ヘッダ要素を表す %XML.Security.Signature のインスタンスを返します。シグニチャ値は、HMAC-SHA1 ダイジェスト・アルゴリズムを使用し、<DerivedKeyToken> によって暗黙的に使用される対称鍵を使用して計算されます。

   <Signature> 要素は、メッセージの既定の部分に適用されます。また、異なる部分を指定できます。
4. WS-Security ヘッダ要素にデジタル・シグニチャを追加します。そのためには、Web クライアントまたは Web サービスの SecurityOut プロパティの AddSecurityElement() メソッドを呼び出します。引数に、前の手順で作成したシグニチャ・オブジェクトを指定します。以下はその例です。

   ObjectScript

   ```
   do ..SecurityOut.AddSecurityElement(dsig)
   ```

例えば、以下のクライアント側のコードは、SOAP 本文に署名します。

ObjectScript

```
// get credentials
set cred = ##class(%SYS.X509Credentials).GetByAlias("servercred")

// get EncryptedKey element that does not encrypt the body
set enckey=##class(%XML.Security.EncryptedKey).CreateX509(cred,$$$SOAPWSEncryptNone)
//add to WS-Security Header
do client.SecurityOut.AddSecurityElement(enckey)

// get derived key & add to header
set dksig=##class(%SOAP.WSSC.DerivedKeyToken).Create(enckey,$$$SOAPWSReferenceEncryptedKey)
//add to WS-Security Header
do client.SecurityOut.AddSecurityElement(dksig)

// create a signature and add it to the security header
set sig=##class(%XML.Security.Signature).Create(dksig,,$$$SOAPWSReferenceDerivedKey)
do client.SecurityOut.AddSecurityElement(sig)
```

このクライアントは、次のようなメッセージを送信します。

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope [parts omitted]>
  <SOAP-ENV:Header>
    <Security xmlns="[parts omitted]oasis-200401-wss-wssecurity-secext-1.0.xsd">
      <EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#"
                  Id="Id-6188CA15-22BF-41EB-98B1-C86D4B242C9F">
        <EncryptionMethod Algorithm="[parts omitted]#rsa-oaep-mgf1p">
          <DigestMethod xmlns="http://www.w3.org/2000/09/xmldsig#"
              Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></DigestMethod>
        </EncryptionMethod>
        <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
          <SecurityTokenReference
                xmlns="[parts omitted]oasis-200401-wss-wssecurity-secext-1.0.xsd">
            <KeyIdentifier EncodingType="[parts omitted]#Base64Binary"
                           ValueType="[parts omitted]#ThumbprintSHA1">5afOHv1w7WSXwDyz6F3WdM1r6cM=
            </KeyIdentifier>
          </SecurityTokenReference>
        </KeyInfo>
        <CipherData>
          <CipherValue>VKyyi[parts omitted]gMVfayVYxA==</CipherValue>
          </CipherData>
      </EncryptedKey>
      <DerivedKeyToken xmlns="[parts omitted]ws-secureconversation/200512"
                       xmlns:wsc=[parts omitted]ws-secureconversation/200512"
                       wsu:Id="Enc-BACCE807-DB34-46AB-A9B8-42D05D0D1FFD">
        <SecurityTokenReference
             xmlns="[parts omitted]oasis-200401-wss-wssecurity-secext-1.0.xsd">
           <Reference URI="#Id-6188CA15-22BF-41EB-98B1-C86D4B242C9F"></Reference>
        </SecurityTokenReference>
        <Offset>0</Offset>
        <Length>24</Length>
        <Nonce>IgSfZJ1jje710zadbPXf1Q==</Nonce>
      </DerivedKeyToken>
      <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
        <SignedInfo>
          <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
          </CanonicalizationMethod>
          <SignatureMethod Algorithm="[parts omitted]#hmac-sha1"></SignatureMethod>
          <Reference URI="#Body-B08978B3-8BE8-4365-A352-1934D7C33D2D">
             <Transforms>
               <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"></Transform>
             </Transforms>
             <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></DigestMethod>
             <DigestValue>56gxpK1mSVW7DN5LUYRvqDbMt0s=</DigestValue>
           </Reference>
        </SignedInfo>
        <SignatureValue>aY4dKX17zDS2SF+BXlVTHcEituc=</SignatureValue>
        <KeyInfo>
          <SecurityTokenReference
                xmlns="[parts omitted]oasis-200401-wss-wssecurity-secext-1.0.xsd">
            <Reference URI="#Enc-BACCE807-DB34-46AB-A9B8-42D05D0D1FFD"></Reference>
          </SecurityTokenReference>
        </KeyInfo>
      </Signature>
    </Security>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body wsu:Id="Body-B08978B3-8BE8-4365-A352-1934D7C33D2D">
    [omitted]
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

11

暗号化と署名の組み合わせ

同じメッセージ内で暗号化と署名を実行できます。ほとんどの場合、前述の各トピックで説明した方法を単純に組み合わせることができます。ここでは、複数のシナリオについて説明します。

11.1 非対称鍵を使用して署名してから暗号化する方法

非対称鍵を使用して署名してから暗号化するには、以下の手順を実行します。

1. ″デジタル・シグニチャの追加″ の各手順を実行します。
2. ″セキュリティ・ヘッダ要素の暗号化″ の各手順を実行します。

   または、″SOAP 本文の暗号化″ の各手順を実行します。

11.2 非対称鍵を使用して暗号化してから署名する方法

非対称鍵を使用して SOAP 本文のみを暗号化してからデジタル・シグニチャを追加するには、以下の手順を実行します。

1. ″SOAP 本文の暗号化″ の各手順を実行します。
2. ″デジタル・シグニチャの追加″ の各手順を実行します。

非対称鍵を使用して任意のセキュリティ・ヘッダ要素を暗号化してからデジタル・シグニチャを追加するには、最上位レベルの `<ReferenceList>` 要素を使用する必要があります (この要素は、このドキュメントの他の部分では必要ありませんでした)。この場合、以下の手順を実行します。

1. ″セキュリティ・ヘッダ要素の暗号化″ の手順 1 ～ 4 を実行します。
2. 各セキュリティ・ヘッダ要素を暗号化するには、その要素に基づいて `<EncryptedData>` 要素を作成します。そのためには、%XML.Security.EncryptedData の Create() クラス・メソッドを呼び出します。この手順では、以下の 3 つの引数すべてを指定します。

   a. 前の手順で作成した暗号化キーのインスタンス。

   b. 暗号化するセキュリティ・ヘッダ要素。

c. <EncryptedData> が暗号化キーのインスタンスを使用する方法を指定する $$$SOAPWSReferenceEncryptedKey。

以下はその例です。

ObjectScript

```
set refopt=$$$SOAPWSReferenceEncryptedKey
set encdata=##class(%XML.Security.EncryptedData).Create(enckey,userToken,refopt)
```

3. <ReferenceList> 要素を作成します。そのためには、%XML.Security.ReferenceList クラスの %New() メソッドを呼び出します。以下はその例です。

ObjectScript

```
set reflist=##class(%XML.Security.ReferenceList).%New()
```

4. この <ReferenceList> 内で、<EncryptedData> 要素を指す <Reference> を作成します。そのためには、各 <EncryptedData> に対して、以下の手順を実行します。

a. %XML.Security.DataReference の Create() クラス・メソッドを呼び出し、暗号化されたデータのインスタンスを引数として指定します。このメソッドは %XML.Security.DataReference のインスタンスを返します。

b. 参照リストのインスタンスの AddReference() メソッドを呼び出し、データ参照のインスタンスを引数として指定します。

以下はその例です。

ObjectScript

```
set dataref=##class(%XML.Security.DataReference).Create(encdata)
do reflist.AddReference(dataref)
```

5. WS-Security ヘッダ要素に <ReferenceList> 要素を追加します。そのためには、Web クライアントまたは Web サービスの SecurityOut プロパティの AddSecurityElement() メソッドを呼び出します。追加する要素に対して、参照リストのインスタンスを指定します。以下はその例です。

ObjectScript

```
do ..SecurityOut.AddSecurityElement(reflist)
```

注釈　<ReferenceList> 要素は、その他のアイテムを追加する前に追加する必要があります。

6. WS-Security ヘッダ要素に <EncryptedKey> 要素を追加します。AddSecurityElement() を使用します。以下はその例です。

ObjectScript

```
do ..SecurityOut.AddSecurityElement(enckey)
```

7. 暗号化されたセキュリティ・ヘッダ要素を WS-Security ヘッダ要素に追加します。そのためには、Web クライアントまたは Web サービスの SecurityOut プロパティの AddSecurityElement() メソッドを呼び出します。この場合、以下の 2 つの引数を指定します。

a. 組み込むセキュリティ・ヘッダ要素 (その要素に基づく %XML.Security.EncryptedData のインスタンスではない)。

b. 暗号化キー・インスタンス。2 番目の引数は、最初の引数が指定するアイテムの配置場所を指定します。引数が A,B の場合、InterSystems IRIS は、A が B の後であることを確認します。これを指定して、受信側が暗号化キーを最初に処理し、これに依存する暗号化されたセキュリティ・ヘッダ要素を後で処理するようにします。

以下はその例です。

ObjectScript

```
 do ..SecurityOut.AddSecurityElement(userToken,enckey)
```

または、暗号化されたセキュリティ・ヘッダ要素が <Signature> の場合、代わりに AddSecurityElement() を使用します。

8. "デジタル・シグニチャの追加" の各手順を実行します。
9. SOAP メッセージを送信します。"セキュリティ・ヘッダ要素の追加" の一般的な手順を参照してください。

11.3 対称鍵を使用して署名してから暗号化する方法

対称鍵を使用して署名してから暗号化するには、以下の手順を実行します。

1. "暗号化への <DerivedKeyToken> の使用" の各手順を実行します。
2. "署名への <DerivedKeyToken> の使用" の各手順を実行します。

11.3.1 <DerivedKeyToken> 要素の使用

以下の例では、対称鍵を使用して署名してから暗号化します。メッセージ受信者の公開鍵を使用して <EncryptedKey> 要素を作成し、これを使用して 2 つの <DerivedKeyToken> 要素 (署名用と暗号化用) を生成します。

ObjectScript

```
// create UsernameToken
 set userToken=##class(%SOAP.Security.UsernameToken).Create("_SYSTEM","SYS")

 //get credentials of message recipient
 set x509alias = "servernopassword"
 set cred = ##class(%SYS.X509Credentials).GetByAlias(x509alias)

 //get EncryptedKey element
 set enc=##class(%XML.Security.EncryptedKey).CreateX509(cred,$$$SOAPWSEncryptNone)
 do client.SecurityOut.AddSecurityElement(enc)

 // get derived keys
 set dkenc=##class(%SOAP.WSSC.DerivedKeyToken).Create(enc,$$$SOAPWSReferenceEncryptedKey)
 do client.SecurityOut.AddSecurityElement(dkenc)
 set dksig=##class(%SOAP.WSSC.DerivedKeyToken).Create(enc,$$$SOAPWSReferenceEncryptedKey)
 do client.SecurityOut.AddSecurityElement(dksig)

 // create and add signature
 set sig=##class(%XML.Security.Signature).Create(dksig,,$$$SOAPWSReferenceDerivedKey)
 do client.SecurityOut.AddSecurityElement(sig)

 // ReferenceList to encrypt Body and Username. Add after signing
 set reflist=##class(%XML.Security.ReferenceList).%New()
 set refopt=$$$SOAPWSReferenceDerivedKey
 set encryptedData=##class(%XML.Security.EncryptedData).Create(dkenc,userToken,refopt)
 set dataref=##class(%XML.Security.DataReference).Create(encryptedData)
 do reflist.AddReference(dataref)
 set encryptedData=##class(%XML.Security.EncryptedData).Create(dkenc,"",refopt)
 set dataref=##class(%XML.Security.DataReference).Create(encryptedData)
 do reflist.AddReference(dataref)
 do client.SecurityOut.AddSecurityElement(reflist)
```

```
  // Add UsernameToken; force after ReferenceList so that it can decrypt properly
  do client.SecurityOut.AddSecurityElement(userToken,reflist)
```

このクライアントは、次のようなメッセージを送信します。

```
<SOAP-ENV:Envelope [parts omitted]'>
   <SOAP-ENV:Header>
      <Security xmlns="[parts omitted]oasis-200401-wss-wssecurity-secext-1.0.xsd">
         <EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#"
                       Id="Id-A0CBB4B7-18A8-40C1-A2CD-C0C383BF9531">
            <EncryptionMethod Algorithm="[parts omitted]#rsa-oaep-mgf1p">
               <DigestMethod xmlns="http://www.w3.org/2000/09/xmldsig#"
                             Algorithm="[parts omitted]#sha1"></DigestMethod>
            </EncryptionMethod>
            <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
             <SecurityTokenReference xmlns="[parts omitted]oasis-200401-wss-wssecurity-secext-1.0.xsd">

                  <KeyIdentifier EncodingType="[parts omitted]#Base64Binary"
                                 ValueType="[parts omitted]#ThumbprintSHA1">
              5afOHv1w7WSXwDyz6F3WdM1r6cM=</KeyIdentifier>
               </SecurityTokenReference>
            </KeyInfo>
            <CipherData>
               <CipherValue>fR4hoJy4[parts omitted]Gmq1xg==</CipherValue>
            </CipherData>
         </EncryptedKey>
         <DerivedKeyToken xmlns="http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512"
                          xmlns:wsc="http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512"
                          wsu:Id="Enc-43F73EB2-77EC-4D72-9DAD-17B1781BC49C">
           <SecurityTokenReference xmlns="[parts omitted]oasis-200401-wss-wssecurity-secext-1.0.xsd">

               <Reference URI="#Id-A0CBB4B7-18A8-40C1-A2CD-C0C383BF9531"></Reference>
            </SecurityTokenReference>
            <Nonce>Q1wDt0PSSLmARcy+Pg49Sg==</Nonce>
         </DerivedKeyToken>
         <DerivedKeyToken xmlns="http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512"
                          xmlns:wsc="http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512"
                          wsu:Id="Enc-ADE64310-E695-4630-9DA6-A818EF5CEE9D">
           <SecurityTokenReference xmlns="[parts omitted]oasis-200401-wss-wssecurity-secext-1.0.xsd">

               <Reference URI="#Id-A0CBB4B7-18A8-40C1-A2CD-C0C383BF9531"></Reference>
            </SecurityTokenReference>
            <Offset>0</Offset>
            <Length>24</Length>
            <Nonce>PvaakhgdxoBVLR6I1j6KGA==</Nonce>
         </DerivedKeyToken>
         <ReferenceList xmlns="http://www.w3.org/2001/04/xmlenc#">
            <DataReference URI="#Enc-F8013636-5339-4C25-87CD-C241330865F5"></DataReference>
            <DataReference URI="#Enc-CDF877AC-8347-4903-97D9-E8238C473DC4"></DataReference>
         </ReferenceList>
         <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#"
                        Id="Enc-F8013636-5339-4C25-87CD-C241330865F5"
                        Type="http://www.w3.org/2001/04/xmlenc#Element">
            <EncryptionMethod Algorithm="[parts omitted]#aes256-cbc"></EncryptionMethod>
            <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
             <SecurityTokenReference xmlns="[parts omitted]oasis-200401-wss-wssecurity-secext-1.0.xsd">

                  <Reference URI="#Enc-43F73EB2-77EC-4D72-9DAD-17B1781BC49C"></Reference>
               </SecurityTokenReference>
            </KeyInfo>
            <CipherData>
               <CipherValue>ebxkmD[parts omitted]ijtJg==</CipherValue>
            </CipherData>
         </EncryptedData>
         <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
            <SignedInfo>
               <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
               </CanonicalizationMethod>
               <SignatureMethod Algorithm="[parts omitted]#hmac-sha1"></SignatureMethod>
               <Reference URI="#Body-C0D7FF05-EE59-41F6-939D-7B2F2B883E5F">
                  <Transforms>
                     <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"></Transform>
                  </Transforms>
                  <DigestMethod Algorithm="[parts omitted]#sha1"></DigestMethod>
                  <DigestValue>vic7p2selz4Wvm1nAX67p0xF1VI=</DigestValue>
               </Reference>
            </SignedInfo>
            <SignatureValue>TxIBa4a8wX5oFN+eyjjsUuLdn7U=</SignatureValue>
            <KeyInfo>
             <SecurityTokenReference xmlns="[parts omitted]oasis-200401-wss-wssecurity-secext-1.0.xsd">

                  <Reference URI="#Enc-ADE64310-E695-4630-9DA6-A818EF5CEE9D"></Reference>
```

```
            </SecurityTokenReference>
         </KeyInfo>
      </Signature>
   </Security>
</SOAP-ENV:Header>
<SOAP-ENV:Body wsu:Id="Body-C0D7FF05-EE59-41F6-939D-7B2F2B883E5F">
   <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#"
                  Id="Enc-CDF877AC-8347-4903-97D9-E8238C473DC4"
                  Type="http://www.w3.org/2001/04/xmlenc#Content">
      <EncryptionMethod Algorithm="[parts omitted]#aes256-cbc"></EncryptionMethod>
      <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
        <SecurityTokenReference xmlns="[parts omitted]oasis-200401-wss-wssecurity-secext-1.0.xsd">

            <Reference URI="#Enc-43F73EB2-77EC-4D72-9DAD-17B1781BC49C"></Reference>
         </SecurityTokenReference>
      </KeyInfo>
      <CipherData>
         <CipherValue>vYtzDsv[parts omitted]GohGsL6</CipherValue>
      </CipherData>
   </EncryptedData>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

11.4 対称鍵を使用して暗号化してから署名する方法

対称鍵を使用して暗号化してから署名するには、以下の手順を実行します。

1. ″署名への <DerivedKeyToken> の使用″ の各手順を実行します。
2. ″暗号化への <DerivedKeyToken> の使用″ の各手順を実行します。

11.5 セキュリティ・ヘッダ要素の順序

通常、処理を実行する順序で、セキュリティ要素をセキュリティ・ヘッダに追加する必要があります。メッセージ受信者は、転送参照なしで、メッセージを最初から最後まで処理できる必要があります。

以下のテーブルは、非対称鍵を使用したときに、結果として得られるセキュリティ・ヘッダ要素の順序のリストを示しています (このシナリオでは、非対称鍵バインディングを使用します)。

署名してから暗号化	暗号化してから署名
1. その他のヘッダ要素 2. <EncryptedKey> 3. <Signature>	1. その他のヘッダ要素 2. <EncryptedKey> 3. <Signature> 4. <ReferenceList>

以下のテーブルは、対称鍵を使用したときに、結果として得られるセキュリティ・ヘッダ要素の順序のリストを示しています (このシナリオでは、対称鍵バインディングを使用します)。

署名してから暗号化	暗号化してから署名
1. その他のヘッダ要素 2. <EncryptedKey> 3. <DerivedKeyToken> 4. <DerivedKeyToken> 5. <ReferenceList> 6. <Signature>	1. その他のヘッダ要素 2. <EncryptedKey> 3. <DerivedKeyToken> 4. <DerivedKeyToken> 5. <Signature> 6. <ReferenceList>

12

着信メッセージの検証と解読

このトピックでは、InterSystems IRIS Web サービスまたは Web クライアントが受信したメッセージのセキュリティ要素を検証する方法 (および暗号化された内容を自動的に解読する方法) について説明します。

12.1 概要

InterSystems IRIS Web サービスおよび Web クライアントでは、着信 SOAP メッセージの WS-Security ヘッダ要素の検証、および着信メッセージの自動解読を実行できます。

InterSystems IRIS Web サービスおよび Web クライアントでは、署名済みの SAML アサーション・トークンを処理して、シグニチャを検証することもできます。ただし、SAML アサーションの詳細の検証は、アプリケーションで行う必要があります。

セキュリティ・ポリシーを使用している場合、前述のすべてのアクティビティは自動的に行われます。

すべてのシナリオにおいて、InterSystems IRIS は、ルート権限証明書のコレクションを使用します。"設定およびその他の一般的なアクティビティ" を参照してください。

12.2 WS-Security ヘッダの検証

着信 SOAP メッセージに記述された WS-Security ヘッダ要素を検証するには、次の操作を実行します。

1. Web サービスまたは Web クライアントで、SECURITYIN パラメータを設定します。以下の値のいずれかを使用します。
   - `REQUIRE` — Web サービスまたは Web クライアントは、WS-Security ヘッダ要素を検証し、不一致がある場合、またはこの要素が見つからない場合はエラーを出力します。
   - `ALLOW` — Web サービスまたは Web クライアントは、WS-Security ヘッダ要素を検証します。

   いずれの場合も、Web サービスや Web クライアントで、ヘッダ要素 `<Timestamp>`、`<UsernameToken>`、`<BinarySecurityToken>`、`<Signature>`、および `<EncryptedKey>` が検証されます。また、ヘッダの SAML アサーションに WS-Security シグニチャが存在する場合は、これも検証されます。必要に応じてメッセージの解読も実行されます。

   検証に失敗すると、エラーが返されます。

テストおよびトラブルシューティングに使用する SECURITYIN パラメータには、この他に以下の 2 つの値も設定できます。

- `IGNORE` ― Web サービスまたは Web クライアントは、"CSP 認証と WS-Security" の説明のように `<UsernameToken>` 以外の WS-Security ヘッダ要素を無視します。

  後方互換性のために、この値が既定値です。

- `IGNOREALL` ― Web サービスまたは Web クライアントは、WS-Security ヘッダ要素をすべて無視します。

使用例は、"メッセージの暗号化の例" を参照してください。

注釈　SECURITYIN パラメータは、関連付けられた (そしてコンパイルされた) 構成クラスにセキュリティ・ポリシーがある場合、無視されます。

12.3 WS-Security ヘッダの SAML アサーションへのアクセス

WS-Security ヘッダ要素に `<Assertion>` 要素が含まれている場合、InterSystems IRIS Web サービスまたは Web クライアントでは、その SAML アサーションのシグニチャが自動的に検証されます (署名されている場合)。

注釈　検証には、信頼された証明書が必要です。InterSystems IRIS が署名を検証できるのは、中間証明書 (もしあれば) を含め、署名者独自の証明書から、InterSystems IRIS が信頼する認証機関 (CA) の自己署名証明書までの署名者の証明書チェーンを検証できる場合です。

ただし、アサーションが自動的に検証されることはありません。コードでアサーションを取得し、これを検証する必要があります。

SAML アサーションにアクセスするには、セキュリティ・ヘッダ要素の `<Assertion>` 要素を見つけます。そのためには、次のように、サービスまたはクライアントの **SecurityIn** プロパティの FindElement() メソッドを使用します。

ObjectScript

```
Set assertion=..SecurityIn.FindElement("Assertion")
```

これにより **%SAML.Assertion** のインスタンスが返されます。必要に応じてこのオブジェクトのプロパティを検証します。

12.4 インスタンス認証と WS-Security

InterSystems IRIS Web サービスでは、IRIS サーバと Web サービス・コードという 2 つの別個のメカニズムが適用されていることを認識しておくと役立ちます。

- 管理ポータルで、Web アプリケーションに許可されていて **`%Service_WebGateway`** サービスへのアクセスを制御する認証モードを指定します (詳細は、"タイムスタンプおよびユーザ名トークンの例" を参照してください。その他の背景情報は、"Web アプリケーション" を参照してください。)**[パスワード]** オプションを選択すると、Web アプリケーションで InterSystems IRIS のユーザ名とパスワードのペアを受け入れることができます。これをインスタンス認証と呼びます。
- これとは別に、Web サービスで InterSystems IRIS のユーザ名とパスワードのペアを要求することもできます。

この両機能は以下のように連携して機能します。

1. メッセージの受信時に、Web サービスで、<Security> と呼ばれるヘッダ要素の存在がチェックされます。ただし、この要素の内容は検証されません。
2. <Security> ヘッダ要素が存在せず、SECURITYIN パラメータが REQUIRE である場合、Web サービスは、フォルトを出力して終了します。
3. <Security> ヘッダ要素に <UsernameToken> 要素がある場合は、以下のように処理されます。
   - Web アプリケーションに [パスワード] オプションを選択すると、Web サービスは <UsernameToken> 要素を読み取ってそこからユーザ名とパスワードを取得し、Web アプリケーションにログインします。

     Web サービスは、SECURITYIN パラメータの任意の値 (IGNOREALL を除く) に対してこれを実行します。

     ユーザ名は、$USERNAME 特殊変数と Web サービスの Username プロパティで使用できます。パスワードは使用できません。
   - [パスワード] オプションを選択しない場合、ログインは行われません。

注釈　SECURITYIN パラメータは、関連付けられた (そしてコンパイルされた) 構成クラスにセキュリティ・ポリシーがある場合、無視されます。

12.5 セキュリティ・ヘッダ要素の取得

場合によっては、WS-Security ヘッダ要素のカスタム処理を追加する必要があることもあります。そのためには、Web サービスまたは Web クライアントの SecurityIn プロパティを使用します。サービスまたはクライアントが WS-Security ヘッダ要素を受信した場合、このプロパティは、このヘッダ要素を含む %SOAP.Security.Header のインスタンスになります。以下はその例です。

ObjectScript

```
Set secheader=myservice.SecurityIn
```

そして、そのインスタンスの次のメソッドのいずれかを使用して、ヘッダ要素を取得します。

FindByEncryptedKeySHA1()

```
method FindByEncryptedKeySHA1(encryptedKeySHA1 As %Binary) as %SOAP.Security.Element
```

指定の EncryptedKeySHA1 引数に対応する <EncryptedKey> 要素からキーを返します。または、一致するものがない場合は、空の文字列を返します。

FindElement()

```
method FindElement(type As %String, ByRef pos As %String) as %SOAP.Security.Element
```

位置 pos の後で、指定されたタイプの最初のセキュリティ要素を返します。一致するものがない場合、メソッドは空の文字列を返します (pos を 0 として返します)。

type には、"Timestamp"、"BinarySecurityToken"、"UsernameToken"、"Signature"、または "EncryptedKey" を指定します。

FindLastElement()

```
method FindLastElement(type As %String, ByRef pos As %String) as %SOAP.Security.Element
```

指定のタイプの最後のセキュリティ要素を返します。一致するものがない場合、メソッドは空の文字列を返します(pos を 0 として返します)。

type の詳細は、FindElement() のエントリを参照してください。

これらすべてのメソッドは、要素タイプに応じて、%SOAP.Security.Element のインスタンスまたは以下のサブクラスのインスタンスのいずれかを返します。

要素タイプ	使用されるサブクラス
"Timestamp"	%SOAP.Security.Timestamp
"BinarySecurityToken"	%SOAP.Security.BinarySecurityToken
"UsernameToken"	%SOAP.Security.UsernameToken
"Signature"	%XML.Security.Signature

詳細は、クラスリファレンスを参照してください。

12.6 シグニチャの確認のチェック

WS-Security 1.1 <SignatureConfirmation> 機能は、受信した SOAP メッセージが、Web クライアントによって送信された元の要求への応答で生成されたことを、Web クライアントが確認できるようにします。クライアント要求は通常、署名されますが、署名は必須ではありません。このメカニズムでは、Web サービスは、セキュリティ・ヘッダ要素に <SignatureConfirmation> 要素を追加し、Web クライアントは、その <SignatureConfirmation> 要素を検証できます。

Web クライアントの場合、Web サービスから受信した応答内の <SignatureConfirmation> 要素を検証するには、Web クライアントの WSCheckSignatureConfirmation() メソッドを呼び出します。このメソッドは、<SignatureConfirmation> 要素が有効な場合は true を、それ以外の場合は false を返します。

Web サービスが送信するメッセージにシグニチャの確認を追加する方法については、"シグニチャの確認の追加" を参照してください。

13
安全な通信の作成

InterSystems IRIS は、WS-SecureConversation 1.3 仕様に従って、安全な通信をサポートします。これを行う最も簡単な方法は、セキュリティ・ポリシーを作成し、Web サービス/クライアント構成ウィザードの **[保護セッション（安全な通信）を確立する]** オプションを使用することです。別のオプションは、このトピックで説明するように、手動で安全な通信を作成することです。

13.1 概要

安全な通信では、Web クライアントは、最初の要求を Web サービスに対して行い、`<SecurityContextToken>` を含むメッセージを受信します。この要素には、両者が使用できる対称鍵に関する情報が含まれています。この情報は、この両者のみが知っている共有秘密鍵を参照します。両者は、トークンが無効になるまで、またはクライアントがトークンをキャンセルするまで、この対称鍵をその後のメッセージ交換に使用できます。

これらのタスクに `<SecurityContextToken>` を直接使用する（これはお勧めしません）のではなく、両者は、このトークンから `<DerivedKeyToken>` を生成し、これを暗号化、署名、解読、およびシグニチャ検証に使用する必要があります。

共有秘密鍵は、以下のいずれかの方法で指定できます。

- 両者がランダム・エントロピー値を指定する場合、両者が共同で指定。これは、一般的なシナリオです。
- クライアントがランダム・クライアント・エントロピー値を指定する場合、クライアントが指定。
- サービスがランダム・サービス・エントロピー値を指定する場合、サービスが指定。

13.2 安全な通信の開始

Web クライアントは、安全な通信を開始します。InterSystems IRIS でこれを行うには、Web クライアント内で以下の手順を実行します。

1. SOAP 本文を暗号化します。クライアントが送信した要求には、保護する必要がある情報が含まれています。この情報は、SOAP 本文内に保持されます。

   必要に応じて、別の方法で要求メッセージを保護します。

2. %SOAP.WST.Entropy の CreateBinarySecret() メソッドを呼び出します。このメソッドは、ランダム・クライアント・エントロピーを表すクラスのインスタンスを返します。このメソッドは、1 つの引数 (バイト単位のエントロピーのサイズ) を取ります。

   以下はその例です。

   ObjectScript

   ```
   set clientEntropy=##class(%SOAP.WST.Entropy).CreateBinarySecret(32)
   ```

   このインスタンスは、<Entropy> 要素およびこれに含まれる <BinarySecret> を表します。

3. %SOAP.WST.RequestSecurityToken の CreateIssueRequest() クラス・メソッドを呼び出します。このメソッドは、安全な通信を要求するためにクライアントが使用する、このクラスのインスタンスを返します。このメソッドには、以下の引数があります。

   a. Interval。要求されたトークンの存続期間。既定は 300 秒です。存続期間を指定しない場合、空の文字列を使用します。

   b. clientEntropy。前の手順で作成したクライアント・エントロピー・オブジェクト (該当する場合)。

   c. requireServerEntropy。サーバが <SecurityContextToken> を作成するときにサーバ・エントロピーを使用する必要があるかどうかを指定するブーリアン値。既定値は False です。

   ```
   set RST=##class(%SOAP.WST.RequestSecurityToken).CreateIssueRequest(300,clientEntropy,1)
   ```

4. オプションで、%SOAP.WST.RequestSecurityToken インスタンスの ComputedKeySize プロパティを指定します。

5. Web クライアントの StartSecureConversation() メソッドを呼び出します。このメソッドは、両者が使用できる <SecurityContextToken> を要求する Web サービスにメッセージを送信します。このメソッドは、1 つの引数 (前の手順の %SOAP.WST.RequestSecurityToken のインスタンス) を取ります。

   このメソッドは、ステータス・コードを返すので、コードでこのステータス・コードを確認する必要があります。応答が成功を示す場合、クライアントの SecurityContextToken プロパティには、クライアントから返された <SecurityContextToken> を表す %SOAP.WSSC.SecurityContextToken のインスタンスが含まれます。この要素には、両者が暗号化、署名、解読、およびシグニチャ検証に使用できる対称鍵に関する情報が含まれています。

6. <SecurityContextToken> を使用して、必要に応じてセキュリティ・ヘッダを再指定します。"〈SecurityContext-Token〉 の使用" を参照してください。

以下に例を示します。

ObjectScript

```
//encrypt the SOAP body because it contains part of the shared secret key
Set x509alias = "servernopassword"
Set cred = ##class(%SYS.X509Credentials).GetByAlias(x509alias)
set enckey=##class(%XML.Security.EncryptedKey).CreateX509(cred)
do client.SecurityOut.AddSecurityElement(enckey)

// if client entropy to be passed
set entropy=##class(%SOAP.WST.Entropy).CreateBinarySecret(32)
// request with 300 second lifetime and computed key using both client
//and server entropy.
set RST=##class(%SOAP.WST.RequestSecurityToken).CreateIssueRequest(300,entropy,1)
set sc=client.StartSecureConversation(RST)  ; sends a SOAP message
```

13.3 InterSystems IRIS Web サービスによる WS-SecureConversation のサポートの有効化

安全な通信は、Web クライアントが Web サービスにメッセージを送信して安全な通信を要求したときに開始されます。応答で、Web サービスは、両者が使用できる <SecurityContextToken> を送信します。

InterSystems IRIS Web サービスがこのトークンを使用して応答できるようにするには、Web サービスの OnStartSecureConversation() メソッドをオーバーライドします。このメソッドには、以下のシグニチャがあります。

```
Method OnStartSecureConversation(RST As %SOAP.WST.RequestSecurityToken) As
                        %SOAP.WST.RequestSecurityTokenResponseCollection
```

このメソッドは、以下を実行します。

1. SOAP 本文を暗号化します。OnStartSecureConversation() が送信したメッセージには、保護する必要がある情報が含まれています。この情報は、SOAP 本文内に保持されます。

   必要に応じて、別の方法でメッセージを保護します。

2. オプションで、%SOAP.WST.Entropy の CreateBinarySecret() メソッドを呼び出します。このメソッドは、ランダム・サーバ・エントロピーを表すクラスのインスタンスを返します。このメソッドは、1 つの引数 (バイト単位のエントロピーのサイズ) を取ります。

   以下はその例です。

   ObjectScript

   ```
   set serverEntropy=##class(%SOAP.WST.Entropy).CreateBinarySecret(32)
   ```

   このインスタンスは、<Entropy> 要素およびこれに含まれる <BinarySecret> を表します。

3. OnStartSecureConversation() が受信する %SOAP.WST.RequestSecurityToken インスタンスの CreateIssueResponse() メソッドを呼び出します。CreateIssueResponse() メソッドは以下の引数を取ります。

   a. $THIS。現在の Web サービスのインスタンスを表します。

   b. keysize。目的の鍵のサイズ (バイト単位)。この引数は、サーバ・エントロピーとクライアント・エントロピーの両方が指定される場合にのみ使用されます。クライアント・エントロピーの鍵とサーバ・エントロピーの鍵が指定された場合、既定では小さいほうの鍵のサイズになります。

   c. requireClientEntropy。要求にクライアント・エントロピーを含めることを Web サービスが要求するかどうかに応じて、True または False のいずれかです。これが False の場合、要求にクライアント・エントロピーを含めないでください。

   d. serverEntropy。手順 2 で作成したサーバ・エントロピー・オブジェクト (該当する場合)。

   e. error。出力パラメータとして返されるステータス・コード。

   f. lifetime。安全な通信の存続時間を指定する秒単位の整数。

   以下はその例です。

   ```
   set responseCollection=RST.CreateIssueResponse($this,,1,serverEntropy,.error)
   ```

4. 前の手順の error 出力パラメータをチェックします。エラーが発生した場合、コードで Web サーバの SoapFault プロパティを設定し、空の文字列を返す必要があります。

5. 成功の場合、手順 2 で作成された %SOAP.WST.RequestSecurityTokenResponseCollection のインスタンスを返します。

   このインスタンスは、<RequestSecurityTokenResponseCollection> 要素を表します。これには、両者が使用できる <SecurityContextToken> が含まれています。

以下に例を示します。

Class Member

```
Method OnStartSecureConversation(RST As %SOAP.WST.RequestSecurityToken)
As %SOAP.WST.RequestSecurityTokenResponseCollection
{
 // encrypt the SOAP body sent by this messsage
 //because it contains part of the shared secret key
 Set x509alias = "clientnopassword"
 Set cred = ##class(%SYS.X509Credentials).GetByAlias(x509alias)
 set enckey=##class(%XML.Security.EncryptedKey).CreateX509(cred)
 do ..SecurityOut.AddSecurityElement(enckey)

 //Supply the server entropy
 set serverEntropy=##class(%SOAP.WST.Entropy).CreateBinarySecret(32)
 // Get the response collection for computed key
 set responseCollection=RST.CreateIssueResponse($this,,1,serverEntropy,.error)

 If error'="" {
    set ..SoapFault=##class(%SOAP.WST.RequestSecurityTokenResponse).MakeFault("InvalidRequest")
    Quit ""
    }

 Quit responseCollection
}
```

注釈　OnStartSecureConversation() メソッドは最初に定義され、<SecurityContextToken> がポリシーによって指定された場合にのみ、これを返します。"ポリシーの作成と使用" を参照してください。

13.4 <SecurityContextToken> の使用

Web サービスが <SecurityContextToken> を使用して応答すると、クライアントのインスタンスおよびサービスのインスタンスは、同じ対称鍵にアクセスできます。この鍵の情報は、両方のインスタンスの SecurityContextToken プロパティに格納されています。推奨される手順は、以下のとおりです。

1. クライアントで、SecurityOut プロパティを NULL に設定し、要求メッセージで使用されたセキュリティ・ヘッダを削除します。

   この手順は、Web サービスでは必要ありません。Web サービスは、各呼び出し後にセキュリティ・ヘッダを自動的にクリアするためです。

2. 必要に応じて、WS-Security ヘッダ要素に <SecurityContextToken> を追加します。そのためには、Web クライアントまたは Web サービスの SecurityOut プロパティの AddSecurityElement() メソッドを呼び出します。以下はその例です。

   ObjectScript

   ```
   set SCT=..SecurityContextToken
   do ..SecurityOut.AddSecurityElement(SCT)
   ```

   次の手順で派生キー・トークンを作成するときに $$$SOAPWSReferenceSCT 参照オプションを使用する場合、この手順は必須です。それ以外の場合、この手順は必要ありません。

3. <SecurityContextToken> に基づいて、新しい <DerivedKeyToken> を作成します。そのためには、以下のように %SOAP.WSSC.DerivedKeyToken の Create() メソッドを呼び出します。

ObjectScript

```
set dkenc=##class(%SOAP.WSSC.DerivedKeyToken).Create(SCT,refOpt)
```

このシナリオでは、Create() に最初の引数を指定する必要があり、また、refOpt は以下の参照値のいずれかにする必要があります。

- `$$$SOAPWSReferenceSCT` (ローカル参照) ー 参照の `URI` 属性は、# で始まり、`<SecurityContextToken>` 要素の `wsu:Id` 値を指します。これは、メッセージに含まれている必要があります。
- `$$$SOAPWSReferenceSCTIdentifier` (リモート参照) ー 参照の `URI` 属性には、`<SecurityContextToken>` 要素の `<Identifier>` 値が含まれています。これは、メッセージに含まれている必要はありません。

以下はその例です。

ObjectScript

```
set dkenc=##class(%SOAP.WSSC.DerivedKeyToken).Create(SCT,$$$SOAPWSReferenceSCT)
```

4. 必要に応じて新しい `<DerivedKeyToken>` を使用し、セキュリティ・ヘッダを指定します。"暗号化および署名への派生キー・トークンの使用" を参照してください。
5. SOAP メッセージを送信します。"セキュリティ・ヘッダ要素の追加" の一般的な手順を参照してください。

以下に例を示します。

ObjectScript

```
//initiate conversation -- not shown here
//clear SecurityOut so that we can respecify the security header elements for
//the secure conversation
set client.SecurityOut=""

//get SecurityContextToken
set SCT=client.SecurityContextToken
do client.SecurityOut.AddSecurityElement(SCT)

// get derived keys
set dksig=##class(%SOAP.WSSC.DerivedKeyToken).Create(SCT,$$$SOAPWSReferenceSCT)
do client.SecurityOut.AddSecurityElement(dksig)

// create and add signature
set sig=##class(%XML.Security.Signature).Create(dksig,,$$$SOAPWSReferenceDerivedKey)
do client.SecurityOut.AddSecurityElement(sig)


//invoke web methods
```

13.5 安全な通信の終了

安全な通信は、`<SecurityContextToken>` の有効期限が切れるとすぐに終了しますが、Web クライアントは、有効期限が切れていない `<SecurityContextToken>` をキャンセルして、通信を終了することも可能です。

安全な通信を終了するには、Web クライアントの CancelSecureConversation() メソッドを呼び出します。以下はその例です。

```
set status=client.CancelSecureConversation()
```

メソッドはステータス値を返します。

このメソッドは、クライアントの SecurityOut プロパティを空の文字列に設定することに注意してください。

14

WS-ReliableMessaging の使用

″はじめに″ で説明したように、InterSystems IRIS は、WS-ReliableMessaging 仕様の一部をサポートします。この仕様は、メッセージのシーケンスを確実に順番に送信するメカニズムを提供します。このサポートを使用する最も簡単な方法は、セキュリティ・ポリシーを作成し、Web サービス/クライアント構成ウィザードの **[信頼性の高いメッセージ配信]** オプションを使用することです。別のオプションは、このトピックで説明するように、信頼性の高いメッセージングを手動で使用することです。

14.1 Web クライアントからメッセージのシーケンスを送信する

InterSystems IRIS Web クライアントから WS-ReliableMessaging をサポートする Web サービスにメッセージのシーケンスを確実に送信するには、以下の手順を実行します。

1. 必要に応じて、Web クライアントのセキュリティ・ヘッダ要素を指定します。

   WS-SecureConversation を使用している場合は、安全な通信を開始します。

2. **%SOAP.RM.CreateSequence** の Create() クラス・メソッドを呼び出します。クラスのインスタンスを返します。

   このメソッドには、以下のシグニチャがあります。

   ```
   classmethod Create(addressingNamespace As %String,
                      oneWay As %Boolean = 0,
                      retryInterval As %Float = 1.0,
                      maxRetryCount As %Integer = 8,
                      expires As %xsd.duration,
                      SSLSecurity As %Boolean = 0) as %SOAP.RM.CreateSequence
   ```

   以下はその説明です。

   - addressingNamespace は、WS-Addressing をサポートするために使用されるネームスペースです。既定は `"http://www.w3.org/2005/08/addressing"` です。
   - oneWay は、要求シーケンスのみが作成される場合に True になります。
   - retryInterval は、再試行される前に待機する間隔の秒数です。
   - maxRetryCount は、アクティビティが発生しなかった場合の再試行の最大数です。
   - expires は、要求されるシーケンスの送信の有効期間を指定するXML 形式の有効期間です。
   - SSLSecurity は、Web クライアントが SSL を使用して Web サービスに接続するかどうかを指定します。

3. Web クライアントの %StartRMSession() メソッドを呼び出し、**%SOAP.RM.CreateSequence** のインスタンスを引数として渡します。

%SOAP.RM.CreateSequence のインスタンスは一度しか使用できません。つまり、このインスタンスを使用して、後で別のセッションを作成することはできません。

4. 必要に応じて Web メソッドを呼び出します。

   毎回同じ Web クライアント・インスタンスを使用します。

5. メッセージの送信を完了したときに Web クライアントの %CloseRMSession() メソッドを呼び出します。

重要　　また、次のセクションで説明するように、WS-ReliableMessaging ヘッダに署名してください。

14.2 WS-ReliableMessaging ヘッダに署名する

以下のいずれかの方法で、WS-ReliableMessaging ヘッダに署名できます。

14.2.1 SecurityContextToken を使用してヘッダに署名する

WS-SecureConversation を使用している場合は、Web クライアントの SecurityContextToken プロパティには対称鍵が含まれ、これを WS-ReliableMessaging ヘッダ要素の署名に使用できます。そのためには、%SOAP.RM.CreateSequence のインスタンスの AddSTR() メソッドを呼び出し、SecurityContextToken プロパティを引数として渡します。

```
do createsequence.AddSTR(client.SecurityContextToken)
```

この処理は、%StartRMSession() を呼び出す前に実行します。

14.2.2 メッセージに署名するときにヘッダに署名する

メッセージの残りに署名するときと同じ方法で WS-ReliableMessaging ヘッダにも署名できます。このためには、%XML.Signature の Create() メソッドまたは CreateX509() メソッドを呼び出す場合に、値 $$$SOAPWSIncludeRMHeaders を signatureOptions 引数に追加します。$$$SOAPWSIncludeRMHeaders マクロは、%soap.inc ファイルに含まれます。

14.3 WS-ReliableMessaging をサポートするように Web サービスを変更する

WS-ReliableMessaging をサポートするように InterSystems IRIS Web サービスを変更するには、以下が実行されるように Web メソッドを変更します。

- 着信要求メッセージに WS-ReliableMessaging ヘッダが含まれることを確認します。
- WS-ReliableMessaging ヘッダが署名されていることを確認します。

  InterSystems IRIS は、シグニチャが有効かどうかを自動的に確認します。"着信メッセージの検証と解読" を参照してください。
- 次のセクションの説明に従って、Web サービス・クラスのパラメータを指定して、Web サービスの動作を微調整できます。

ただし、セキュリティ・ポリシーを作成し、Web サービス/クライアント構成ウィザードの [**信頼性の高いメッセージ配信**] オプションを使用する方が簡単です。

14.4 Web サービスで信頼性の高いメッセージングを処理する方法を制御する

Web サービス・クラスの以下のパラメータを指定して、Web サービスの動作を微調整できます。

RMINORDER

WS-ReliableMessaging の InOrder ポリシー・アサーションに対応します。0 (False)、もしくは 1 (True) に指定します。詳細は、Web Services Reliable Messaging Policy 1.1 の仕様を参照してください。

既定では、このパラメータが指定されていない場合、InterSystems IRIS Web サービスはメッセージの順序に関する SOAP フォルトを発行しません。

RMDELIVERYASSURANCE

WS-ReliableMessaging の DeliveryAssurance ポリシー・アサーションに対応します。`"ExactlyOnce"`、`"AtLeastOnce"`、または `"AtMostOnce"` に指定します。詳細は、Web Services Reliable Messaging Policy 1.1 の仕様を参照してください。

既定では、このパラメータが指定されていない場合、InterSystems IRIS Web サービスはこのポリシー・アサーションに従った配信の失敗に関する SOAP フォルトを発行しません。

RMINACTIVITYTIMEOUT

Web サービスによって受信されるシーケンスでアクティビティがない場合のタイムアウトを秒数で指定します。既定値は 10 分です。

[**信頼性の高いメッセージ配信**] オプションを使用するセキュリティ・ポリシーを使用する Web サービス・クラスで同じパラメータを指定できます。

また、Web サービスの %OnCreateRMSession() コールバック・メソッドを実装することもできます。このメソッドは、%SOAP.RM.CreateSequenceResponse が返される前に、WS-ReliableMessaging セッションの開始時に呼び出されます。この時点では、応答引数は完全に作成されており、まだ返されていません。このコールバックによって、任意の必須セキュリティ・ヘッダ要素を Web サービスの **SecurityOut** プロパティに追加可能になります。WS-Policy が使用されている場合は、WS-Policy サポートによってこの操作が自動的に実行されます。メソッド・シグニチャについては、**%SOAP.WebService** のクラス・リファレンスを参照してください。

15

SAML トークンの作成と追加

このトピックでは、WS-Security ヘッダ要素に SAML トークンを追加する方法を説明します。

%SAML.Assertion および関連クラスのクラス・リファレンスも参照してください。

SAML の完全サポートは実装されていません。InterSystems IRIS の SAML サポートは、"InterSystems IRIS における WS-Security のサポート" に掲載した詳細のみに適用されます。

15.1 概要

InterSystems IRIS SOAP サポートでは、WS-Security ヘッダ要素に SAML トークンを追加できます。

必要に応じて、この SAML トークンを署名または暗号化のキー・マテリアルとして使用できます。このようにすると、InterSystems IRIS は WS-Security SAML トークン・プロファイル仕様に準拠します。キー・マテリアルは、Holder-of-key (HOK) メソッドと <SubjectConfirmationData> を持つ SAML アサーションの <SubjectConfirmation> 要素、または <KeyInfo> 下位要素を持つ <KeyInfoConfirmationData> から取得されます。

また、Sender-vouches (SV) メソッドを持つ <SubjectConfirmation> を追加することもできます。この場合、サブジェクトには、キーは含まれません。この場合にアサーションを保護するには、メッセージ・シグニチャから SAML トークンへのセキュリティ・トークン参照を取得します。

15.2 基本的な手順

SAML トークンを作成し、これを発信 SOAP メッセージに追加するには、次に説明する基本的な手順を使用するか、またはサブセクションで説明するバリエーションを使用します。

1. 必要に応じて、%soap.inc インクルード・ファイルを組み込みます。このファイルには、使用する可能性のあるマクロが定義されています。
2. "プログラムによる資格情報セットの取得" の説明に従って、%SYS.X509Credentials のインスタンスを作成します。

   この InterSystems IRIS 資格情報セットには、独自の証明書を含める必要があります。以下はその例です。

   ObjectScript

```
Set x509alias = "servercred"
Set pwd = "mypassword"
Set credset = ##class(%SYS.X509Credentials).GetByAlias(x509alias,pwd)
```

3. 指定の資格情報セットに関連付けられている証明書が含まれるバイナリ・セキュリティ・トークンを作成します。これには、%SOAP.Security.BinarySecurityToken の CreateX509Token() クラス・メソッドを呼び出します。以下はその例です。

   ObjectScript

   ```
   set bst=##class(%SOAP.Security.BinarySecurityToken).CreateX509Token(credset)
   ```

   credset は、前の手順で作成した InterSystems IRIS 資格情報セットです。

4. WS-Security ヘッダ要素にこのトークンを追加します。そのためには、Web クライアントまたは Web サービスの SecurityOut プロパティの AddSecurityElement() メソッドを呼び出します。このメソッドの引数には、作成したトークンを使用します。以下はその例です。

   ObjectScript

   ```
   do ..SecurityOut.AddSecurityElement(bst)
   ```

5. バイナリ・セキュリティ・トークンに基づいて、署名済みの SAML アサーションを作成します。そのためには、%SAML.Assertion の CreateX509() クラス・メソッドを呼び出します。以下はその例です。

   ObjectScript

   ```
   set assertion=##class(%SAML.Assertion).CreateX509(bst)
   ```

   このメソッドは %SAML.Assertion のインスタンスを返します。InterSystems IRIS は、このインスタンスの Signature、SAMLID、および Version の各プロパティを自動的に設定します。

   このインスタンスは、<Assertion> 要素を表します。

6. %SAML.Assertion のインスタンスの以下の基本プロパティを指定します。

   - IssueInstant では、このアサーションの発行日時を指定します。
   - Issuer では、%SAML.NameID のインスタンスを作成します。必要に応じてこのインスタンスのプロパティを指定し、アサーションの Issuer プロパティをこのインスタンスに設定します。

7. "SAML 文の追加" の説明に従って、SAML 文を追加します。
8. "<Subject> 要素の追加" の説明に従って、SAML アサーションに <Subject> 要素を追加します。
9. オプションで、"<SubjectConfirmation> 要素の追加" の説明に従って、<Subject> に <SubjectConfirmation> 要素を追加します。

   Holder Of Key メソッドまたは Sender Voucher メソッドのいずれかを持つサブジェクトを確認できます。

10. "<Conditions> 要素の追加" の説明に従って、SAML <Conditions> 要素を指定します。
11. オプションで、"<Advice> 要素の追加" の説明に従って、<Advice> 要素を追加します。
12. Web クライアントまたは Web サービスの SecurityOut プロパティの AddSecurityElement() メソッドを呼び出します。このメソッドの引数には、作成した SAML トークンを使用します。
13. オプションで、SOAP メッセージ・シグニチャから SAML アサーションに参照を追加することで、SAML アサーションに署名します。

    シグニチャが %XML.Security.Signature オブジェクトである場合、次のように SAML アサーションに署名します。

    ObjectScript

    ```
    Set str=##class(%SOAP.Security.SecurityTokenReference).GetSAMLKeyIdentifier(assertion)
    Set ref=##class(%XML.Security.Reference).CreateSTR(str.GetId())
    Do signature.AddReference(ref)
    ```

この手順は、特に、Sender Vouches メソッドを持つ <SubjectConfirmation> を追加する場合にお勧めします。

14. SOAP メッセージを送信します。"セキュリティ・ヘッダ要素の追加" の一般的な手順を参照してください。

15.2.1 バリエーション :<BinarySecurityToken> を使用しない手順

<BinarySecurityToken> には、シリアル化された Base 64 のエンコード形式の証明書が含まれています。このトークンを省略し、代わりに証明書を指定する情報を使用できます。受信者は、この情報を使用して、適切な場所から証明書を取得します。そのためには、前述の手順を使用しますが、以下の変更点があります。

- 手順 2 と 3 をスキップします。つまり、<BinarySecurityToken> の作成および追加は行わないでください。
- 手順 4 で、CreateX509() の最初の引数として資格情報セットを使用します (バイナリ・セキュリティ・トークンは使用しません)。以下はその例です。

ObjectScript

```
set assertion=##class(%SAML.Assertion).CreateX509(credset,referenceOption)
```

referenceOption には、必要に応じて、"X.509 資格情報の参照オプション" の説明に従って、値を指定します。$$$SOAPWSReferenceDirect 以外の任意の値を使用します。

(このバリエーションで実行しているように) 最初の引数として資格情報セットを指定する場合、既定の参照オプションは、証明書のサムプリントです。

15.2.2 バリエーション :署名なしの SAML アサーションの作成

署名なしの SAML アサーションを作成するには、前述の手順を使用しますが、以下の変更点があります。

- 手順 1、2 および 3 をスキップします。つまり、<BinarySecurityToken> の作成および追加は行わないでください。
- 手順 4 で、CreateX509() の代わりに Create() メソッドを使用します。このメソッドは、引数を取りません。以下はその例です。

ObjectScript

```
set assertion=##class(%SAML.Assertion).Create()
```

このメソッドは %SAML.Assertion のインスタンスを返します。InterSystems IRIS は、このインスタンスの SAMLID プロパティおよび Version プロパティを自動的に設定します。Signature プロパティは NULL です。

15.3 SAML 文の追加

SAML 文を %SAML.Assertion のインスタンスに追加するには、次の操作を実行します。

1. 該当する文クラスの 1 つまたは複数のインスタンスを作成します。
   - %SAML.AttributeStatement
   - %SAML.AuthnStatement
   - %SAML.AuthzDecisionStatement
2. 必要に応じてこれらのインスタンスのプロパティを指定します。

%SAML.AttributeStatement の場合、Attribute プロパティは %SAML.Attribute、または %SAML.EncryptedAttribute のインスタンスです。

%SAML.Attribute の AttributeValue プロパティには属性値が格納されます。これは %SAML.AttributeValue インスタンスのリストです。

%SAML.Attribute インスタンスに属性値を追加する手順は以下のとおりです。

a. %SAML.AttributeValue のインスタンスを作成します。

b. %SAML.AttributeValue のメソッドを使用して属性を XML、文字列、または単一の子要素のいずれかとして指定します。

c. それらの属性値インスタンスを含むリストを作成します。

d. 属性値オブジェクトの AttributeValue プロパティをこのリストに設定します。

または AttributeValueOverride プロパティを直接指定します。この値には、完全一致文字列 (XML 混在コンテンツの文字列) を使用します。

3. それらの文インスタンスを含むリストを作成します。
4. アサーション・オブジェクトの Statement プロパティをこのリストに設定します。

15.4 <Subject> 要素の追加

<Subject> 要素を %SAML.Assertion のインスタンスに追加するには、次の操作を実行します。

1. %SAML.Subject の新しいインスタンスを作成します。
2. 必要に応じて件名のプロパティを設定します。
3. アサーション・オブジェクトの Subject プロパティをこのインスタンスに設定します。

15.5 <SubjectConfirmation> 要素の追加

<SubjectConfirmation> 要素を %SAML.Assertion のインスタンスに追加するには、次のサブセクションのいずれかで説明している手順に従います。

15.5.1 Holder-of-key メソッドを持つ <SubjectConfirmation>

Holder-of-key メソッドを持つ <SubjectConfirmation> を追加するには、次の手順を実行します。

1. "プログラムによる資格情報セットの取得" の説明に従って %SYS.X509Credentials のインスタンスを作成します。

   また、アサーションの署名に使用した資格情報セットと同じものを使用することもできます。

2. オプションで、指定の資格情報セットに関連付けられている証明書が含まれるバイナリ・セキュリティ・トークンを作成して追加します。

   トークンを作成するには、%SOAP.Security.BinarySecurityToken の CreateX509Token() クラス・メソッドを呼び出します。以下はその例です。

ObjectScript

```
set bst=##class(%SOAP.Security.BinarySecurityToken).CreateX509Token(credset)
```

credset は、前の手順で作成した資格情報セットです。

WS-Security ヘッダ要素にこのトークンを追加するには、Web クライアントまたは Web サービスの SecurityOut プロパティの AddSecurityElement() メソッドを呼び出します。このメソッドの引数には、作成したトークンを使用します。

3. SAML アサーション・オブジェクトの Subject プロパティの AddX509Confirmation() メソッドを呼び出します。

```
method AddX509Confirmation(credentials As %SYS.X509Credentials,
                           referenceOption As %Integer) as %Status
```

credentials には、バイナリ・セキュリティ・トークンまたは資格情報セットを使用します。ここでトークンを使用する場合は、referenceOption を指定しないでください。後者の場合、"X.509 資格情報の参照オプション" の説明に従って、値を指定します。

<SubjectConfirmation> 要素は、X.509 KeyInfo 要素に従います。

15.5.2 Sender-vouches メソッドを持つ 〈SubjectConfirmation〉

Sender-vouches メソッドを持つ <SubjectConfirmation> を追加するには、次の手順を実行します。

1. SAML アサーション・オブジェクトの Subject プロパティの NameID プロパティを設定します。
2. SAML アサーション・オブジェクトの Subject プロパティの AddConfirmation() メソッドを呼び出します。

```
method AddConfirmation(method As %String) as %Status
```

method では、$$$SAMLSenderVouches、$$$SAMLHolderOfKey、または $$$SAMLBearer を指定します。

この場合、必ず SAML アサーションに署名して、その SAML アサーションを保護します。

15.5.3 〈EncryptedKey〉 を持つ 〈SubjectConfirmation〉

<EncryptedKey> 要素を含んだ <SubjectConfirmationData> を持つ <SubjectConfirmation> を追加するには、次の手順を実行します。

1. "プログラムによる資格情報セットの取得" の説明に従って %SYS.X509Credentials のインスタンスを作成します。

   また、アサーションの署名に使用した資格情報セットと同じものを使用することもできます。
2. SAML アサーション・オブジェクトの Subject プロパティの NameID プロパティを設定します。
3. SAML アサーション・オブジェクトの Subject プロパティの AddEncryptedKeyConfirmation() メソッドを呼び出します。

```
method AddEncryptedKeyConfirmation(credentials As %X509.Credentials) as %Status
```

引数には、先ほど作成した %SYS.X509Credentials のインスタンスを使用します。

15.5.4 Holder-of-key として BinarySecret を持つ 〈SubjectConfirmation〉

Holder-of-key として BinarySecret を持つ <SubjectConfirmation> を追加するには、次の手順を実行します。

1. SAML アサーションに署名するときに、以下のように署名を作成します。

```
set sig=##class(%XML.Security.Signature).Create(assertion,$$$SOAPWSIncludeNone,$$$SOAPWSSAML)
```

assertion は、SAML アサーションです。このシナリオでは、Create() メソッドを使用することに注意してください。$$$SOAPWSSAML 参照オプションは、SAML アサーションへの参照を作成します。

2. BinarySecret を作成します。これには、%SOAP.WST.BinarySecret の Create() メソッドを呼び出します。

```
set binsec=##class(%SOAP.WST.BinarySecret).Create()
```

3. SAML アサーション・オブジェクトの Subject プロパティの AddBinarySecretConfirmation() メソッドを呼び出します。

```
set status=assertion.Subject.AddBinarySecretConfirmation(binsec)
```

binsec の場合は、前の手順で作成した BinarySecret を使用します。

これにより、<BinarySecret> を含んだ <KeyInfo> を含む <SubjectConfirmationData> を含んだ <SubjectConfirmation> が追加されます。

15.6 〈Conditions〉 要素の追加

<Conditions> 要素を %SAML.Assertion のインスタンスに追加するには、次の操作を実行します。

1. %SAML.Conditions のインスタンスを作成します。
2. 必要に応じてこのインスタンスのプロパティを指定します。
3. アサーション・オブジェクトの Conditions プロパティをこのインスタンスに設定します。

15.7 〈Advice〉 要素の追加

<Advice> 要素を %SAML.Assertion のインスタンスに追加するには、次の操作を実行します。

1. 次のクラスの 1 つまたは複数のインスタンスを作成します。
   - %SAML.AssertionIDRef
   - %SAML.AssertionURIRef
   - %SAML.EncryptedAssertion
2. 必要に応じてこれらのインスタンスのプロパティを指定します。
3. それらの Advice インスタンスを含むリストを作成します。
4. アサーション・オブジェクトの Advice プロパティをこのリストに設定します。

16

セキュリティの問題のトラブルシューティング

このトピックでは、InterSystems IRIS で SOAP セキュリティに関する問題の原因を特定する際に役立つ情報を提供します。

セキュリティに関係のない問題については、"InterSystems IRIS での SOAP の問題のトラブルシューティング" を参照してください。

16.1 トラブルシューティングに必要な情報

SOAP の問題をトラブルシューティングするには、通常、以下の情報が必要です。

- WSDL およびこれが参照するすべての外部ドキュメント。
- (メッセージ関連の問題の場合) 何らかの形式のメッセージ・ロギングおよびトレース。以下のオプションがあります。

オプション	SSL/TLS で使用可能かどうか	HTTP ヘッダを表示するかどうか	コメント
InterSystems IRIS SOAP ログ	はい	いいえ	セキュリティ・エラーの場合、このログは、SOAP フォルトに含まれるものよりも詳細を表示します。
Web ゲートウェイ・トレース	はい	はい	MTOM (MIME 添付) を使用する SOAP メッセージの問題の場合、HTTP ヘッダを表示することは重要です。
サードパーティのトレース・ツール	いいえ	ツールに依存	トレース・ツールの中には、送信された実際のパケットなど、下位レベルの詳細を表示するものもあります。これらの情報は、トラブルシューティングの際に重要になる場合があります。

  これらのオプションについては、"InterSystems IRIS での SOAP の問題のトラブルシューティング" を参照してください。

- まれな状況として SOAP クライアントで HTTP 認証を使用する場合は、認証のログ記録を有効にすることができます。"ログイン資格情報の提供" を参照してください。

また、フォルトを正しく処理して最適な情報を受け取るようにすることは非常に有効です。"SOAP フォルトの処理" を参照してください。

16.2 考えられるエラー

このセクションでは、InterSystems IRIS Web サービスおよび Web クライアントで考えられるセキュリティ関連のエラーについて説明します。

- InterSystems IRIS Web サービスまたは Web クライアントを生成したばかりの場合、まだ WS-Security ヘッダを認識するように構成されていないことがあります。この場合、Web メソッドを実行しようとすると、以下のような汎用エラーが発生します。

  ```
  <ZSOAP>zInvokeClient+269^%SOAP.WebClient.1
  ```

  以下を Web サービスまたは Web クライアントに追加して、再コンパイルします。

  Class Member

  ```
  Parameter SECURITYIN="REQUIRE";
  ```

  この汎用エラーは、Web メソッドを誤って呼び出すことによって発生する可能性もあります (例えば、Web メソッドに返り値がないときに返り値を参照した場合)。

  この項目は、WS-Policy を使用している場合には適用されません。

- Web メソッドを実行しようとすると、以下のようなセキュリティ・エラーが発生する場合もあります。

  ```
  ERROR #6454: No supported policy alternative in configuration
  Policy.Client.Demo1SoapConfig:service
  ```

  "セキュリティ・エラー発生時に確認する項目" を参照してください。

- 着信メッセージが検証に失敗する場合があります。この場合、SOAP ログにこのエラーが示されます。以下はその例です。

  ```
  08/05/2011 14:40:11 **********************
  Input to Web client with SOAP action = http://www.myapp.org/XMLEncr.DivideWS.Divide
  <?xml version='1.0' encoding='UTF-8' standalone='no' ?>
  <SOAP-ENV:Envelope xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance' xmlns:s='http://www.w3.org/2001/XMLSchema'
  xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd" >
    <SOAP-ENV:Body>
      <SOAP-ENV:Fault>
        <faultcode>wsse:FailedAuthentication</faultcode>
        <faultstring>The security token could not be authenticated or authorized</faultstring>
        <detail></detail>
      </SOAP-ENV:Fault>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
  ```

  "セキュリティ・エラー発生時に確認する項目" を参照してください。

16.3 セキュリティ・エラー発生時に確認する項目

着信メッセージが検証に失敗した場合、または "サポートされている代替ポリシーがない" というエラーを InterSystems IRIS が発行した場合、以下の項目を確認すると役立ちます。

- 格納された InterSystems IRIS 資格情報セットを取得する場合、その名前が正しく入力されていることを確認します。
- InterSystems IRIS 資格情報セットを取得した後、オブジェクトのタイプをチェックして、%SYS.X509Credentials であることを確認します。
- 適切な証明書が使用されていることを確認します。

  証明書を暗号化に使用する場合、メッセージの送信先のエンティティの証明書を使用します。暗号化は、この証明書の公開鍵を使用します。

  証明書を署名に使用する場合、所有している証明書を使用し、関連付けられた秘密鍵を使用して署名します。この場合、秘密鍵をロード済みであることおよびその秘密鍵ファイルのパスワードが正しく指定されていることを確認します。
- 証明書が InterSystems IRIS で信頼されている認証機関によって署名されていることを確認します。
- WS-Policy を使用している場合は必ず、生成された構成クラスを編集して、使用する InterSystems IRIS 資格情報セットを指定します。"生成されたポリシーの編集" を参照してください。
- Web サービスが <UsernameToken> を要求する場合、InterSystems IRIS Web クライアントがこのトークンを送信していること、およびこのトークンに正しい情報が含まれていることを確認します。InterSystems IRIS は、送信する <UsernameToken> を自動的に指定できません。これは、実行時に行う必要があります。"タイムスタンプおよびユーザ名トークンの追加" を参照してください。

  Web サービスまたは Web クライアントで必要なセキュリティ・ポリシーのうち、少なくとも 1 つが InterSystems IRIS でサポートされていることを確認します。"SOAP セキュリティ標準" を参照してください。
- 認証に失敗した場合、<UsernameToken> のユーザを特定し、そのユーザが属するロールを調べます。

A
セキュリティ要素の詳細

このトピックでは、SOAP メッセージの一般的なセキュリティ要素、特に InterSystems IRIS Web サービスおよび Web クライアントで送信可能なバリエーションについて説明します。この情報は、継続して SOAP の作業に携わっていない人が覚えている知識を確認するためのものです。ここで提供する詳細は、トラブルシューティングにも役立ちます。

A.1 <BinarySecurityToken>

<BinarySecurityToken> の目的は、メッセージ受信者が使用するために、メッセージの他の要素で使用されるセキュリティ資格情報を保持することです。セキュリティ資格情報は、シリアル化されたエンコード形式で保持されます。以下は、部分的な例です。

```
<BinarySecurityToken wsu:Id="SecurityToken-4EC1997A-AD6B-48E3-9E91-8D50C8EA3B53"
                     EncodingType="[parts omitted]#Base64Binary"
                     ValueType="[parts omitted]#X509v3">
              MIICnDCCAYQ[parts omitted]ngHKNhh
</BinarySecurityToken>
```

A.1.1 詳細

この要素の各部分は以下のとおりです。

- Id は、このトークンの一意の識別子で、このメッセージの他の要素がこのトークンを参照できるように組み込まれます。InterSystems IRIS では、必要に応じて、これが自動的に生成されます。
- EncodingType は、<BinarySecurityToken> の値を生成するために使用されたエンコードのタイプを示します。InterSystems IRIS では、<BinarySecurityToken> で使用されるエンコードは Base 64 エンコードのみです。
- ValueType は、トークンに含まれている値のタイプを示します。InterSystems IRIS では、サポートされている値のタイプは X.509 証明書のみです。
- <BinarySecurityToken> 要素内に含まれる値は、シリアル化されたエンコード証明書です。この例では、値 MIICnDCCAYQ[parts omitted]ngHKNhh がセキュリティ資格情報です。

このトークンが暗号化アクションに関連付けられている場合、含まれる証明書はメッセージ受信者の証明書です。このトークンが署名に関連付けられている場合、含まれる証明書はメッセージ送信者の証明書です。

A.1.2 メッセージ内での位置

<BinarySecurityToken> は、<Security> 内で、この要素を参照する他の要素よりも前に組み込まれる必要があります。

A.2 〈EncryptedKey〉

<EncryptedKey> の目的は、メッセージの他の要素によって使用される対称鍵を保持することです。対称鍵は、暗号化された形式で保持されます。以下は、部分的な例です。

```
<EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#">
   <EncryptionMethod Algorithm="[parts omitted]xmlenc#rsa-oaep-mgf1p">
      <DigestMethod xmlns="http://www.w3.org/2000/09/xmldsig#"
                    Algorithm="http://www.w3.org/2000/09/xmldsig#sha1">
      </DigestMethod>
   </EncryptionMethod>
   <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
      <SecurityTokenReference xmlns="[parts omitted]oasis-200401-wss-wssecurity-secext-1.0.xsd">
         <Reference URI="#SecurityToken-4EC1997A-AD6B-48E3-9E91-8D50C8EA3B53"
                    ValueType="[parts omitted]#X509v3">
         </Reference>
      </SecurityTokenReference>
   </KeyInfo>
   <CipherData>
      <CipherValue>WtE[parts omitted]bSyvg==</CipherValue>
   </CipherData>
   <ReferenceList>
      <DataReference URI="#Enc-143BBBAA-B75D-49EB-86AC-B414D818109F"></DataReference>
   </ReferenceList>
</EncryptedKey>
```

A.2.1 詳細

この要素の各部分は以下のとおりです。

- <EncryptionMethod> は、対称鍵の暗号化に使用されたアルゴリズムを示します。

  InterSystems IRIS では、鍵転送アルゴリズムを指定できます (<EncryptionMethod> の Algorithm 属性によって示されます)。"鍵転送アルゴリズムの指定" を参照してください。

- <KeyInfo> は、この対称鍵の暗号化に使用された鍵を特定します。InterSystems IRIS では、<KeyInfo> に以下のいずれかの形式の <SecurityTokenReference> が含まれます。

  - 前述の例で示したように、WS-Security ヘッダ内で前に位置する <BinarySecurityToken> への参照。
  - 証明書を一意に特定する情報。これは、メッセージ受信者が所有していることを想定しています。例えば、<SecurityTokenReference> には、以下のように証明書の SHA1 サムプリントを記述できます。

    ```
    <SecurityTokenReference xmlns="[parts omitted]oasis-200401-wss-wssecurity-secext-1.0.xsd">
       <KeyIdentifier EncodingType="[parts omitted]#Base64Binary"
                      ValueType="[parts omitted]#ThumbprintSHA1">
          maedm8CNoh4zH8SMoF+3xV1MYtc=
       </KeyIdentifier>
    </SecurityTokenReference>
    ```

  いずれの場合も、この <EncryptedKey> 要素に保持される対称鍵の暗号化には、対応する公開鍵が使用されています。

  暗号化で最上位レベルの <ReferenceList> 要素が使用される場合、この要素は省略されます。"〈ReferenceList〉" を参照してください。

- <CipherData> は、<CipherValue> 要素の値として、暗号化された対称鍵を保持します。この例では、値 WtE[parts omitted]bSyvg== が暗号化された対称鍵です。
- <ReferenceList> は、このメッセージの <EncryptedKey> 要素に保持された対称鍵を使用して暗号化された部分を示します。具体的には、<DataReference> の URI 属性は、メッセージの他の部分の <EncryptedData> 要素の Id 属性を指します。

  使用する手法に応じて、この要素が含まれない場合があります。代わりに、最上位レベルの <ReferenceList> 要素を使用して、<EncryptedData> および対応する <EncryptedKey> をリンクすることもできます。"〈ReferenceList〉" を参照してください。

A.2.2 メッセージ内での位置

<EncryptedKey> 要素は、<Security> 内で、この要素が使用するすべての <BinarySecurityToken> よりも後、かつ、この要素を参照するすべての <EncryptedData> 要素および <DerivedKeyToken> 要素よりも前に組み込まれる必要があります。

A.3 〈EncryptedData〉

<EncryptedData> の目的は、暗号化されたデータを保持することです。以下は、部分的な例です。

```
<EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#"
               Id="Enc-143BBBAA-B75D-49EB-86AC-B414D818109F"
               Type="http://www.w3.org/2001/04/xmlenc#Content">
   <EncryptionMethod Algorithm="[parts omitted]#aes128-cbc"></EncryptionMethod>
   <CipherData>
      <CipherValue>MLwR6hvKE0gon[parts omitted]8njiQ==</CipherValue>
   </CipherData>
</EncryptedData>
```

A.3.1 詳細

この要素の各部分は以下のとおりです。

- Id は、この要素の一意の識別子です。InterSystems IRIS では、これが自動的に生成されます。
- <EncryptionMethod> は、このデータの暗号化に使用されたアルゴリズムを示します。

  InterSystems IRIS では、このアルゴリズムを指定できます。"ブロック暗号化アルゴリズムの指定" を参照してください。
- <CipherData> は、<CipherValue> 要素の値として、暗号化されたデータを保持します。この例では、値 MLwR6hvKE0gon[parts omitted]8njiQ== が暗号化されたデータです。
- (この例には含まれていません) <KeyInfo> は、対称鍵を特定します。この場合、<KeyInfo> には <SecurityTokenReference> 要素が含まれ、この要素には対称鍵への参照が以下のいずれかの形式で含まれます。
  - WS-Security ヘッダ内で前に位置する <DerivedKeyToken> への参照。

- 暗黙的 <DerivedKeyToken> への参照。以下はその例です。

```
<KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
   <SecurityTokenReference xmlns="[parts omitted]oasis-200401-wss-wssecurity-secext-1.0.xsd"
                s01:Nonce="mMDk0zn8V7WTsFaIjUJ7zg=="
                xmlns:s01="http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512">
      <Reference URI="#Id-93F97220-568E-47FC-B3E1-A2CF3F70B29B"></Reference>
   </SecurityTokenReference>
</KeyInfo>
```

この場合、<Reference> の URI 属性は、<DerivedKeyToken> の生成に使用された <EncryptedKey> 要素を指し、Nonce 属性は、使用された Nonce 値を示します。

いずれの場合も、この <EncryptedData> 要素に保持されるデータの暗号化には、この派生キーが使用されています。

暗号化で最上位レベルの <ReferenceList> 要素が使用される場合に <KeyInfo> 要素が記述されます。"〈ReferenceList〉" を参照してください。

A.3.2 メッセージ内での位置

<EncryptedData> 要素は、<Security> 内で、関連付けられている <EncryptedKey> よりも後に組み込まれる必要があります。

<EncryptedData> 要素は SOAP 本文 (<Body> 要素) の子にすることもできます。

A.4 〈Signature〉

<Signature> の目的は、メッセージの受信者が検証できるデジタル・シグニチャを保持することです。デジタル・シグニチャを使用して、メッセージの変更を検出したり、リストされているエンティティによってメッセージの特定の部分が実際に生成されたことを検証したりできます。従来の手書きの署名と同様に、デジタル・シグニチャは、ドキュメントの作成者のみが作成できるドキュメントへの追加物で、簡単に偽造することはできません。

以下は、部分的な例です。

```
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
   <SignedInfo>
      <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
      </CanonicalizationMethod>
      <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha256"></SignatureMethod>
      <Reference URI="#Timestamp-48CEE53E-E6C3-456C-9214-B7D533B2663F">
         <Transforms>
            <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"></Transform>
         </Transforms>
         <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></DigestMethod>
         <DigestValue>waSMFeYMruQn9XHx85HqunhMGIA=</DigestValue>
      </Reference>
      <Reference URI="#Body-73F08A5C-0FFD-4FE9-AC15-254423DBA6A2">
         <Transforms>
            <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"></Transform>
         </Transforms>
         <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></DigestMethod>
         <DigestValue>wDCqAzy5bLKKF+Rt0+YV/gxTQws=</DigestValue>
      </Reference>
   </SignedInfo>
   <SignatureValue>j6vtht/[parts omitted]trCQ==</SignatureValue>
   <KeyInfo>
      <SecurityTokenReference xmlns="[parts omitted]oasis-200401-wss-wssecurity-secext-1.0.xsd">
         <Reference URI="#SecurityToken-411A262D-990E-49F3-8D12-7D7E56E15081"
                    ValueType="[parts omitted]oasis-200401-wss-x509-token-profile-1.0#X509v3">
         </Reference>
      </SecurityTokenReference>
   </KeyInfo>
</Signature>
```

A.4.1 詳細

この要素の各部分は以下のとおりです。

- <SignedInfo> は、このシグニチャによって署名されるメッセージの部分を示し、その部分が署名の前にどのように処理されたかを示します。

  InterSystems IRIS では、ダイジェスト・メソッドを指定できます (<DigestMethod> の Algorithm 属性によって示されます)。"ダイジェスト・メソッドの指定" を参照してください。

  シグニチャの計算に使用するアルゴリズムを指定することも可能です (<SignatureMethod> の Algorithm 属性 によって示されます)。"シグニチャ・メソッドの指定" を参照してください。

- <SignatureValue> は、実際のシグニチャを保持します。この場合、シグニチャは、6vtht/[parts omitted]trCQ== です。

  この値は、署名された部分の連結されたダイジェストを暗号化することによって計算されます。暗号化は、送信者の秘密鍵を使用して実行されます。

- <KeyInfo> は、シグニチャの作成に使用された鍵を特定します。InterSystems IRIS では、<KeyInfo> に以下のいずれかの形式の <SecurityTokenReference> が含まれます。

  - 前述の例で示したように、WS-Security ヘッダ内で前に位置する <BinarySecurityToken> への参照。この場合、シグニチャの作成には対応する秘密鍵が使用されています。
  - メッセージ受信者が以前に受信して格納した証明書を特定する情報。例えば、<SecurityTokenReference> には、以下のように証明書の SHA1 サムプリントを記述できます。

    ```
    <SecurityTokenReference xmlns="[parts omitted]oasis-200401-wss-wssecurity-secext-1.0.xsd">
       <KeyIdentifier EncodingType="[parts omitted]#Base64Binary"
                      ValueType="[parts omitted]#ThumbprintSHA1">
          maedm8CNoh4zH8SMoF+3xV1MYtc=
       </KeyIdentifier>
    </SecurityTokenReference>
    ```

    前の場合と同様に、シグニチャの作成には対応する秘密鍵が使用されています。

  - WS-Security ヘッダ内で前に位置する <DerivedKeyToken> への参照。以下はその例です。

    ```
    <SecurityTokenReference xmlns="[parts omitted]oasis-200401-wss-wssecurity-secext-1.0.xsd">
       <Reference URI="#Enc-BACCE807-DB34-46AB-A9B8-42D05D0D1FFD"></Reference>
    </SecurityTokenReference>
    ```

    この場合、シグニチャは、そのトークンが示す対称鍵によって作成されています。

  - 暗黙的 <DerivedKeyToken> への参照。以下はその例です。

    ```
    <SecurityTokenReference xmlns="[parts omitted]oasis-200401-wss-wssecurity-secext-1.0.xsd"
                  s01:Nonce="mMDk0zn8V7WTsFaIjUJ7zg=="
                  xmlns:s01="http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512">
       <Reference URI="#Id-93F97220-568E-47FC-B3E1-A2CF3F70B29B"></Reference>
    </SecurityTokenReference>
    ```

    この場合、<Reference> の URI 属性は、<DerivedKeyToken> の生成に使用された <EncryptedKey> 要素を指し、Nonce 属性は、使用された Nonce 値を示します。

    前の場合と同様に、データの暗号化には派生キーが使用されています。

A.4.2 メッセージ内での位置

<Signature> 要素は、<Security> 内で、この要素が使用する <BinarySecurityToken> または <DerivedKeyToken> よりも後に組み込まれる必要があります (これらがある場合)。

A.5 〈DerivedKeyToken〉

<DerivedKeyToken> の目的は、同じ対称鍵を生成するために、送信者と受信者の両者が独立して使用できる情報を保持することです。両者は、この対称鍵を使用して、SOAP メッセージの関連付けられた部分の暗号化、解読、署名、およびシグニチャ検証を実行できます。

以下は、部分的な例です。

```
<DerivedKeyToken xmlns="[parts omitted]ws-secureconversation/200512"
                 xmlns:wsc="[parts omitted]ws-secureconversation/200512"
                 wsu:Id="Enc-943C6673-E3F3-48E4-AA24-A7F82CCF6511">
   <SecurityTokenReference xmlns="[parts omitted]oasis-200401-wss-wssecurity-secext-1.0.xsd">
      <Reference URI="#Id-658202BF-239A-4A8C-A100-BB25579F366B"></Reference>
   </SecurityTokenReference>
   <Nonce>GbjRvVNrPtHs0zo/w9Ne0w==</Nonce>
</DerivedKeyToken>
```

A.5.1 詳細

この要素の各部分は以下のとおりです。

- Id は、この要素の一意の識別子です。InterSystems IRIS では、これが自動的に生成されます。
- <EncryptionMethod> は、このデータの暗号化に使用されたアルゴリズムを示します。

  InterSystems IRIS では、このアルゴリズムを指定できます。″ブロック暗号化アルゴリズムの指定″ を参照してください。
- <SecurityTokenReference> は、派生キーを計算するときに基盤として使用する対称鍵を示します。これには、<EncryptedKey> または同じメッセージの別の <DerivedKeyToken> のいずれかを指す URI 属性を持つ <Reference> 要素を含めることができます。

  または、<SecurityTokenReference> には、使用された <EncryptedKey> の SHA1 ハッシュを参照する <KeyIdentifier> を含めることができます。以下はその例です。

  ```
  <SecurityTokenReference xmlns="[parts omitted]oasis-200401-wss-wssecurity-secext-1.0.xsd"
                                 s01:TokenType="[parts omitted]#EncryptedKey"
                                 xmlns:s01="h[parts omitted]oasis-wss-wssecurity-secext-1.1.xsd">
     <KeyIdentifier EncodingType="[parts omitted]#Base64Binary"
                                [parts omitted]#EncryptedKeySHA1">
                       U8CEWXdUPsIk/r8JT+2KdwU/gSw=
     </KeyIdentifier>
  </SecurityTokenReference>
  ```
- <Nonce> は、この派生キーの鍵導出関数で seed に使用された Base 64 のエンコード値です。

<DerivedKeyToken> 要素の計算には、RFC 2246 の TLS 用に定義されたメカニズムのサブセットを使用します。

<DerivedKeyToken> 要素と関連付けられた <EncryptedData> 要素または <Signature> 要素との関係は、以下のように処理されます。

- 各 <DerivedKeyToken> には、一意の識別子付きの Id 属性が含まれます。

- 指定された <DerivedKeyToken> が示す対称鍵で暗号化された各 <EncryptedData> 要素内には、その <DerivedKeyToken> を指している URI 属性を持つ <SecurityTokenReference> 要素があります。
- 指定された <DerivedKeyToken> が示す対称鍵によって計算された値を持つ各 <Signature> 要素内には、その <DerivedKeyToken> を指している URI 属性を持つ <SecurityTokenReference> 要素があります。
- <EncryptedData> 要素と <Signature> 要素のそれぞれには、一意の識別子付きの Id 属性も含まれます。
- WS-Security ヘッダには、<EncryptedData> 要素と <Signature> 要素を参照する <ReferenceList> 要素が含まれます。

A.5.2 メッセージ内での位置

<DerivedKeyToken> は、<Security> 内で、この要素を参照するすべての <EncryptedData> 要素および <Signature> 要素よりも前に組み込まれる必要があります。

A.6 <ReferenceList>

このセクションでは、メッセージ・ヘッダで <Security> の子として使用される場合の <ReferenceList> 要素について説明します。<ReferenceList> がこのように使用される場合、署名の前に暗号化を実行できます。以下にこの要素の例を示します。

```
<ReferenceList xmlns="http://www.w3.org/2001/04/xmlenc#">
   <DataReference URI="#Enc-358FB189-81B3-465D-AFEC-BC28A92B179C"></DataReference>
   <DataReference URI="#Enc-9EF5CCE4-CF43-407F-921D-931B5159672D"></DataReference>
</ReferenceList>
```

A.6.1 詳細

各 <DataReference> 要素において、URI 属性は、メッセージの他の部分の <EncryptedData> 要素の Id 属性を指します。

最上位レベルの <ReferenceList> 要素を使用する場合、<EncryptedKey> と <EncryptedData> では以下のように詳細が異なります。

シナリオ	<EncryptedKey>	<EncryptedData>
<EncryptedKey> に <EncryptedData> へのポインタが含まれる	<KeyInfo> が組み込まれる（関連付けられたすべての <EncryptedData> 要素に対して同一になる）	<KeyInfo> は組み込まれない
最上位レベルの <ReferenceList> に <EncryptedData> へのポインタが含まれる	<KeyInfo> は組み込まれない	<KeyInfo> が組み込まれる（各 <EncryptedData> 要素で異なる場合がある）

A.6.2 メッセージ内での位置

<ReferenceList> 要素は、<Security> 内で、関連付けられている <EncryptedKey> よりも後に組み込まれる必要があります。