## 

# Security Primer

A consistent set of fundamental security concepts apply in any distributed messaging system. The following concerns offer increasing message protection:

* Authentication: Identifying the message sender. Mutual authentication involves authenticating both the sender and the message receiver, to prevent possible man-in-the-middle attacks.
* Authorization: After authenticating the message sender, authorization determines what system features and functionality they are entitled to execute.
* Integrity: Messages should be digitally signed to ensure they have not been altered between sender and receiver.
* Confidentiality: Sensitive messages or specific message parts should be encrypted to ensure they cannot be openly viewed on the wire.

Authentication is mandatory for KlWCF. Message integrity and confidentiality are optional features. No capabilities for authorization are built into the WCF communication infrastructure.

The first step to securing a WCF service is defining the security policy. Once the policy is defined, the configuration for authentication, authorization, and message protection is achieved declaratively or through the runtime object model. The choice of binding determines what security features are supported.

There are six basic WCF security modes:

* None: Turns security off.
* Transport: Uses transport security for mutual authentication and message protection.
* Message: Uses message security for mutual authentication and message protection.
* Both: Allows for transport and message-level security (only MSMQ supports this).
* TransportWithMessageCredential: Credentials are passed with the message and message protection and server authentication are provided by the transport layer.
* TransportCredentialOnly: Client credentials are passed with the transport layer and no message protection is applied.

The KlWCF security mechanism uses the message based mode for all bindings except streaming channels

Each transport protocol (TCP, IPC, MSMQ, or HTTP) has its own mechanism for passing credentials and handling message protection. Message security supports passing credentials as part of the SOAP message using interoperable standards, and also makes it possible to protect the message independent of transport all the way through to the ultimate message receiver. Transport message protection is only good from point to point.

Transport security is not viable for the KlWCF since encryption cannot be disabled for that mode. The aim of the security infrastructure is to provide authentication. We would like to support the ability to validate or encrypt messages as well. Transport security always encrypts and this is deemed a performance risk. The security mechanism uses message mode for these reasons.

#### Certificates

Certificates are digital documents typically used for authentication and exchange of information on networks. X.509 digital certificates are commonly used to authenticate clients and servers, encode messages, and digitally sign messages. A digital certificate is a part of a public key infrastructure (PKI), which is a system of digital certificates, certificate authorities, and other registration authorities that verify and authenticate the validity of each party involved in an electronic transaction through the use of public key cryptography. A certificate authority issues certificates and each certificate has a set of fields that contain data, such as subject (the entity to which the certificate is issued), validity dates, issuer, and public key.

Certificates are issued with a planned lifetime, which is defined by start time and an expiration date. Once issued, a certificate is considered valid from the start time until its expiration date. The certificates used by KlWCF will a lifetime of 100 years, effectively obviating the need to reissue expired certificates.

A certification authority is an entity that issues digital certificates for use by other parties. Digital certificates are used to authenticate an entity by relying on a chain of trust. The chain is formed from top to bottom, with a root authority at the top.

A mechanism exists in Windows by which certificates can be revoked. Windows maintains a list of Certificate Revocations which is a file created and signed by a CA, containing serial numbers of certificates that have been issued by that CA and are revoked. Certificates can be revoked for a variety of reasons, including a breach of trust in the root certificate. For chains of trust not requiring access to the internet, revocation requires access to the machine on which the certificate is stored.

The KlWCF security solution will not rely on a chain of trust originating from a third party source. Essentially, we will trust ourselves to be a root authority. This has a cost benefit, allows control of expiration, and is easier to maintain. There are potential downsides of this policy. It could violate the security policy of an institution in which we are deploying KlWCF. Also, a web viewer would report a un-trusted provider when making direct calls to our WCF services.

#### Certificate Stores

Windows maintains certificates in certificate stores. These stores are represented by containers in the file system, or in the registry. Certificate stores are associated either with the computer object or they are owned by a distinct user who has a security context and profile on that computer. Services can also have certificate stores. A computer has read and writes permissions to the personal certificate store that belongs to the computer. Certificates can be issued for a user, a device, or a service. Personal certificates establish identity. Intermediate certificates authenticate other certificates. Root certificates establish the identity of other computers. The logical certificate stores used by the KlWCF are listed below:

* Personal: Certificates associated with private keys to which you have access. These are the certificates that have been issued to you, or to the computer or service for which you are managing certificates.
* Trused Root: Implicitly trusted certification authorities.
* Enterprise Trust: A mechanism for trusting self-signed root certificates from other organizations and limiting the purposes for which these certificates are trusted.
* Trusted People: Certificates issued to people or end entities that are explicitly trusted. Most often these are self-signed certificates or certificates explicitly trusted in an application

Since the KlWCF is deployed on the intranet, it is not a requirement to purchase a Certificate from a trusted authority. By placing the base certificate in EnterpriseTrust, a chain of trust is established which is suitable for securing WCF communication.

For an enterprise to establish their own Enterprise CA it must self-sign a CA Root Certificate, which is used to issue digital certificates to users, machines and services.  The self-signed Root CA Certificate is not automatically trusted by main-stream operating systems and browsers. This inability to trust causes warning messages to be displayed in depending on use. If the KlWCF were to expose browser based capability, the existing security infrastructure would need to be modified since the browser would not trust the self signed root certificate.

#### Makecert

Makecert is used to generate and install certificates in the Certificate Store. A self-signed base certificate is stored in the EnterpriseTrust store of the LocalMachine. That is used to sign the client and server certificates.



The commands to generate and inject certificates are displayed below. The private keys are not stored in the local file system. They must be exported and secured with a password. This feature enables the certificates to be deployed only with the password chosen at export time.

makecert -n "CN=klBase" -sk klBaseKey -pe -sr localmachine -sky exchange -ss TRUST -r klBase.cer

makecert -n "CN=klServer" -sk klServerKey -pe -sr localmachine -ss MY -sky exchange -ic klBase.cer -is TRUST klServer.cer

makecert -n "CN=klClient" -sk klClientKey -pe -sr localmachine -ss MY -sky exchange -ic klBase.cer -is TRUST klClient.cer

The options for the example above are described below.

* -pe Marks the generated private key as exportable. This allows the private key to be included in the certificate.
* -ss Specifies the subject's certificate store name that stores the output certificate.
* -sk Specifies the subject's key container location, which contains the private key.
* -sky Specifies the subject's key type, which must be signature, exchange
* -r Creates a self-signed certificate.
* -ic Specifies the issuer's certificate file.



The base certificate is stored in the Enterprise Trust store of both the client and server. The client certificate is stored in the trusted people store of the server and the personal store of the client. The public key of the server is located in the personal store of the client.

#### TransferMode

Windows Communication Foundation (WCF) transports support two modes of transferring messages in each direction:

* Buffered transfers hold the entire message in a memory buffer until the transfer is complete.
* Streamed transfers only buffer the message headers and expose the message body as a stream, from which smaller portions can be read at a time.

Streamed transfers can improve the scalability of a service by eliminating the need for large memory buffers. Whether changing the transfer mode actually improves scalability in practice depends on the size of the messages being transferred. Improvements in scalability should be most evident when large messages use streamed instead of buffered transfers.

#### Secure Session

A feature of message security is the ability to establish a secure session to reduce the overhead of one-off key exchange and validation. By default, secure sessions are enabled for message security. A security context token (SCT) is generated through an initial exchange between caller and service. This token is used to authorize and secure subsequent message exchanges. If the caller plans to make several calls to a service, secure sessions are more efficient. For a single call, however, you can disable this feature by setting establish SecurityContext to false;

binding.Security.Message.EstablishSecurityContext = establishSecurityContext;

KlDPS has a configuration setting in the Service Registry for WShttp bindings to enable this option.

#### Negotiation

To support mutual authentication and message protection, services must provide credentials to the caller. When transport security is used, service credentials are negotiated through the transport protocol. Service credentials for message security can also be negotiated when Windows credentials are used; otherwise a service certificate must be specified in the <behaviors> section under <serviceCredentials>. In this case, the caller must have access to the public key portion of the service certificate to encrypt messages sent to the service. This can be specified out of band, or negotiated with an initial handshake.  
  
The default behavior for message security supports negotiation. That means that the service is dynamically asked for the correct token before any messages are exchanged. The negotiation process depends on the credential type. For certificate credential types the client must have access to the service public key to encrypt messages. Negotiation is optional, and binding.Security.Message.NegotiateServiceCredential is set to false for WShttp bindings in the KlWCF.

#### Protection Level

By default, all secure WCF bindings will encrypt and sign messages. The protection levels defined for WCF are None, Sign, and EncryptAndSign. Encryption cannot be disabled for transport security. THe KlWCF security solution does not use Transport security for this reason. For message security, the binding can be configured to sign but not encrypt WCF messages. The default behavior for KlWCF is to set the default protection to be None, and use certificates for authentication. Currently the protection level is set in code, and there is no parameter in the registry which enables setting it in the field. If signing and encryption are desired features, this can be easily exposed in the registry.

# System Overview

The security model is implemented using certificates stored in the Certificate Store on the client and server machines. A trusted root certificate is used to generate client and server certificates which are used to negotiate authentication, message signing, and message encryption.

There are two mechanisms employed for securing WCF communications. They are differentiated based on the transfer mode of the binding. Transfer mode indicates whether the message is buffered or streamed. Four types of transfer mode are supported by WCF:

* Streamed - Both in and out messages are streamed
* Buffered - This is the default of buffering all data and sending it in one burst

For buffered bindings, credentials are set on the endpoint and a protection level is programmatically. This is accomplished by modification of ProxyBuilder , and BindingFactory to set service and client credentials on the WCF endpoints, and the addition of an endpoint behavior to set the protection level.

For Streamed communications, WCF is set up to provide authentication via client and server certificates. No message protection in the form of signing or encryption is provided. The authentication is accomplished by injecting a custom message header with a signed hash of the server certificate. Machines with the server certificate will be able to validate the signature and accept messages. An exception is thrown in the event that the signature cannot be validated.

The security mechanism depends upon a physical certificate, the WCF communication infrastructure, and the .NET Security module. It is assumed that certificates can be deployed to client and server machines. It is assumed that encryption of binary data is not a preferred approach. It is assumed that there is limited value in the signing of messages.

The main component in the security system is a set of physical certificates located in the LocalMachine certificate store. WCF communication has functionality to perform authentication using the certificate credentials, and to optionally sign and encrypt messages.

Helper classes are provided to extract certificate information, perform signing and verification, and encryption. For streaming endpoints a message inspector is provided to inject a certificate hash in the message header for authentication purposes. An endpoint behavior is provided to set the message protection level for Message based communication.

# Dependency Description

The component diagram below details the dependencies.

## 

RSA CryptoProvider performs asymmetric encryption and decryption using the implementation of the [RSA](http://msdn.microsoft.com/en-us/library/system.security.cryptography.rsa.aspx) algorithm.

## Error Handling

Client and server errors for authentication are logged using the default mechanism in ILog. For security reasons, there is no attempt to provide explicit details of the nature of an exception arising from bad credentials, or a certificate error. Service faults will be generated and logged

The .NET crypto module will raise a certificate not found error when trying to negotiate credentials and a certificate is missing on the client side. That error is consumed by the client application and displayed to the user.

## Module Interface

The interface for secure communications uses existing classes. The service side uses ServiceBootstrapperBase which inherits its interface from IServiceProxyBuilder. Client communications are set up in ClientServiceProxyBuilder, which also inherits from IServiceProxyBuilder, whose interface is presented below.

public interface IServiceProxyBuilder

{

TService Build<TService>();

TService Build<TService>(IExceptionHandler handler);

TService Build<TService>(String preferredUriScheme, String serviceNodeIdentifier);

TService Build<TService>(IExceptionHandler handler, String peferredUriScheme, String serviceNodeIdentifier);

}

#### Buffered Transfer Mode Module Interface

Buffered communications are secured using service and client credentials. These are set up in three different portions of the DPS communications framework;

* ServiceBootstrapperBase sets the server credentials
* ClientServiceProxyBuilder sets the client credentials
* The KlBindingFactory was modified to set the message based security policy for non-streamed bindings.

#### Streaming Transfer Mode Module Interface

The StreamingSecurityMessageInspector inherits its interface from IDispatchMessageInspectorand IClientMessageInspector which provides a mechanism to inspect and modify incoming and outgoing messages. A WCF MessageInspector is a kind of a "message filter" that we can implement on the service or on the consumer side, in order to intercept and inspect the messages coming in or going out of the service layer infrastructure.

In order to define a Message Inspector on the consumer side the IClientMessageInspector interface needs to be implemented, while on the service side the IDispatchMessageInspector interface needs to be implemented.

Here are their declarations:

public interface IClientMessageInspector

{

void AfterReceiveReply(ref Message reply, object correlationState);

object BeforeSendRequest(ref Message request, IClientChannel channel);

}

public interface IDispatchMessageInspector

{

object AfterReceiveRequest(ref Message request, IClientChannel channel, InstanceContext instanceContext);

void BeforeSendReply(ref Message reply, object correlationState);

}

The CryptoHelper interface contains methods to extract a certificate from the certificate store, sign, encrypt, and validate data. Key methods are listed below.

public class CryptoHelper

{

public static byte[] Sign(byte[] data, X509Certificate2 signingCert);

public static void ValidateCert(X509Certificate2 cert);

public static X509Certificate2 FindCertificate(StoreLocation location, StoreName name, X509FindType findType, string findValue);

public static byte[] VerifyAndRemoveSignature(byte[] data);

}

CryptoHelper is used in StreamingSecurityMessageInspector implementation of IDispatchMessageInspector::BeforeSendReply to

* locate the server certificate
* sign the certificate hash

An example implementation is included below for clarity.

public void BeforeSendReply(ref System.ServiceModel.Channels.Message reply, object correlationState)

{

X509Certificate2 myCert = CryptoHelper.FindCertificate(StoreLocation.LocalMachine, StoreName.My, X509FindType.FindBySubjectDistinguishedName, "CN=KlServer");

byte[] myHash = myCert.GetCertHash();

byte[] mySignedHash = CryptoHelper.Sign(myHash, myCert);

char[] keyChars = new char[mySignedHash.Length];

for (int i = 0; i < mySignedHash.Length; i++)

keyChars[i] = (char)mySignedHash[i];

reply.Headers.Add((new CustomSecurityHeader(new string(keyChars))));

MessageBuffer buffer = reply.CreateBufferedCopy(Int32.MaxValue);

reply = buffer.CreateMessage();

}

The signature is validated on the client side is a similar manor:

public void AfterReceiveReply(ref Message reply, object correlationState)

{

MessageBuffer buffer = reply.CreateBufferedCopy(Int32.MaxValue);

reply = buffer.CreateMessage();

X509Certificate2 myCert = CryptoHelper.FindCertificate(StoreLocation.LocalMachine, StoreName.My, X509FindType.FindBySubjectDistinguishedName, "CN=KlServer");

byte[] myHash = myCert.GetCertHash();

Int32 headerPosition = reply.Headers.FindHeader(CustomHeaderNames.CustomHeaderName, CustomHeaderNames.CustomHeaderNamespace);

XmlDictionaryReader reader = reply.Headers.GetReaderAtHeader(headerPosition);

CustomSecurityHeader header = CustomSecurityHeader.ReadHeader(reader);

string mySignedHashString = header.Key;

char[] mysignedHashCharArray = mySignedHashString.ToArray();

byte[] mySignedHash = new byte[mySignedHashString.Length];

for (int i = 0; i < mySignedHashString.Length; i++)

mySignedHash[i] = (byte)mysignedHashCharArray[i];

byte[] myDecodedHash = CryptoHelper.VerifyAndRemoveSignature(mySignedHash);

for (int i = 0; i < myHash.Length; i++)

{

if (myDecodedHash[i] != myHash[i])

{

throw new Exception("Access Denied");

}

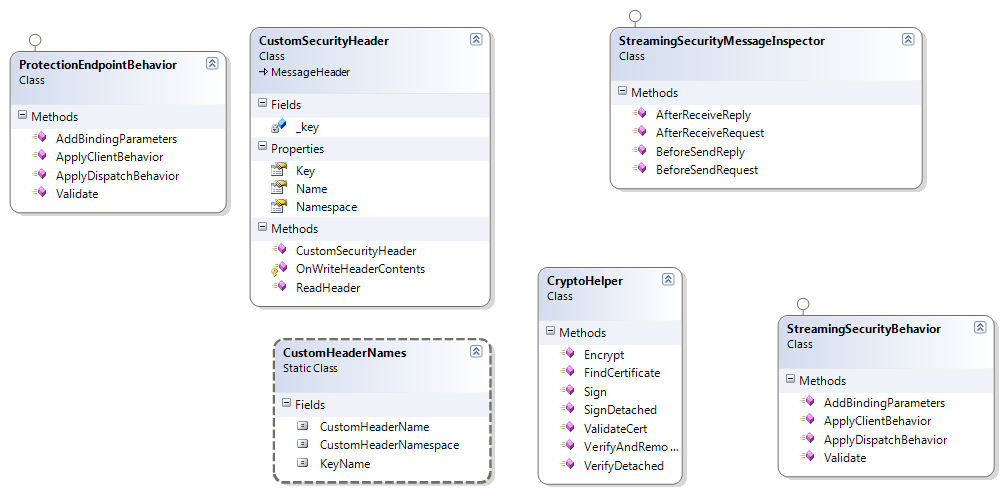
}

}

# Detailed Design

The security module was integrated into the existing WCF communications infrastructure without significant structural modifications

A class diagram for helper classes used in the security infrastructure is presented below.

CryptoHelper is used by the StreamingSecurityMessageInspector for streamed message transfer. Streamed mode is used by our TCP streaming bindings to transfer ingested image files into the KlWCF. CryptoHelper provides the ability to locate certificates, and perform signing, encryption, and validation using the .NET PKI.

The StreamingSecurityMessageInspector, CustomHeaderNames, and CustomSecurityHeader function to authenticate streamed transfer mode bindings.

ProtectionEndpointBehavior is a WCF endpoint behavior that sets the message protection level for buffered communication.