Create an Endpoint

Getting Started

1. Create new F# Console Application

```
// Learn more about F# at http://fsharp.net
// See the 'F# Tutorial' project for more help.

[<EntryPoint>]
let main argv =
    printfn "%A" argv
           0 // return an integer exit code
```

Program.fs

2. Add the following NuGet packages using Package Manager:

   Install-Package Microsoft.Owin
   Install-Package Microsoft.Owin.Hosting
   Install-Package Microsoft.AspNet.WebApi.Owin
   Install-Package Microsoft.owin.Host.HttpListener

3. Add the following references to the project:

   System.Configuration
   System.Configuration.Install
   System.ServiceProcess
   System.Runtime.Serialization

4. Rename “Program.fs” to “EndPoint.fs”
5. Modify the content of EndPoint.fs to be:

```
namespace Example

open System
open Owin
open System.Web.Http
open System.Configuration
open Microsoft.Owin.Hosting

type Startup() =
    member this.Configuration(app: IAppBuilder) =
        try
            let config = new HttpConfiguration()
            config.MapHttpAttributeRoutes()
            //app.UseCors(CorsOptions.AllowAll) |> ignore
            app.UseWebApi(config) |> ignore
        with ex ->
            printfn "%A" ex
            //sprintf "%A" ex |> log

type EndPointState =
     | Running of IDisposable
     | NotRunning

type AgentSignal =
     | Start
     | Stop


type EndPoint() =
    let agent = MailboxProcessor.Start(fun inbox ->
        let rec loop state = async {
            let! msg = inbox.Receive()
            match msg, state with
            | Start, NotRunning ->
                let baseAddress = ConfigurationManager.AppSettings.["baseAddress"]
                let server = WebApp.Start<Startup>(baseAddress)
                //sprintf "Endpoint listening at %s" baseAddress |> log
                return! loop (Running(server))
            | Start, Running(_)
            | Stop, NotRunning ->
                return! loop state
            | Stop, Running(server) ->
                server.Dispose()
                return! loop NotRunning
        }
        loop NotRunning)

    member this.StartEndPoint() =
        agent.Post(Start)

    member this.StopEndPoint() =
        agent.Post(Stop)
```

EndPoint.fs

6. Add this entry to the App.config file:

```
<appSettings>
  <add key="baseAddress" value="http://localhost:4445"/>
</appSettings>
```

7. Add a file “SvcHost.fs” to the project:

```
namespace Example

open System
open System.Configuration
open System.ServiceProcess
open System.Configuration.Install
open System.ComponentModel

module constants =
    let serviceName = "MyService"

type Svc() =
    inherit ServiceBase(ServiceName = constants.serviceName)

    let endPoint = new EndPoint()

    override this.OnStart(args) =
        endPoint.StartEndPoint()
        //sprintf "%s service host started" constants.serviceName |> log

    override this.OnStop() =
        endPoint.StopEndPoint()
        Async.CancelDefaultToken()
        //sprintf "%s service host stopped" constants.serviceName |> log

[<RunInstaller(true)>]
type SvcHost() =
    inherit Installer()

    let defaultServiceName = constants.serviceName
    let serviceInstaller =
        new ServiceInstaller
            (   DisplayName = defaultServiceName,
                ServiceName = defaultServiceName,
                StartType = ServiceStartMode.Manual )

    do
        new ServiceProcessInstaller(Account = ServiceAccount.LocalSystem)
        |> base.Installers.Add |> ignore

        serviceInstaller
        |> base.Installers.Add |> ignore

    override this.Install(state) =
        match this.Context.Parameters.ContainsKey("ServiceName") with
        | true ->
            serviceInstaller.DisplayName <- this.Context.Parameters.["ServiceName"]
            serviceInstaller.ServiceName <- this.Context.Parameters.["ServiceName"]
        | false -> ()
        base.Install(state)

    override this.Uninstall(state) =
        match this.Context.Parameters.ContainsKey("ServiceName") with
        | true ->
            serviceInstaller.DisplayName <- this.Context.Parameters.["ServiceName"]
            serviceInstaller.ServiceName <- this.Context.Parameters.["ServiceName"]
        | false -> ()
        base.Uninstall(state)

module Main =

    // ServiceBase.Run [| new Svc() :> ServiceBase |]

    let endPoint = new EndPoint()
    endPoint.StartEndPoint()
    Console.ReadLine() |> ignore
```

SvcHost.fs

8. Add a file “Controllers.fs” to the project:

```
namespace Example

open System.Web.Http

type PingController() =
    inherit ApiController()

    [<Route("api/ping")>]
    [<HttpGet>]
    member this.Get() =
        // sprintf "%A" this.Request |> log
        "Hello World!"
```

Controllers.fs

9. At this point, you can Ctrl-F5 to build and run the project.
10. Open Fiddler’s Composer
    a. GET
    b. http://localhost:4445/api/ping
    c. Headers:

       Accept: application/json
       Content-Type: application/json;charset=UTF-8

    d. Execute

Improvement #1 – Install and run as a Windows service

1. Modify the SvcHost.fs file by so that the application will run as a service instead of a console application:

```
namespace Example

open System
open System.Configuration
open System.ServiceProcess
open System.Configuration.Install
open System.ComponentModel

module constants =
    let serviceName = "MyService"

type Svc() =
    inherit ServiceBase(ServiceName = constants.serviceName)

    let endPoint = new EndPoint()
    override this.OnStart(args) =
        endPoint.StartEndPoint()
        //sprintf "%s service host started" constants.serviceName |> log

    override this.OnStop() =
        endPoint.StopEndPoint()
        Async.CancelDefaultToken()
        //sprintf "%s service host stopped" constants.serviceName |> log

[<RunInstaller(true)>]
type SvcHost() =
    inherit Installer()

    let defaultServiceName = constants.serviceName
    let serviceInstaller =
        new ServiceInstaller
            (   DisplayName = defaultServiceName,
                ServiceName = defaultServiceName,
                StartType = ServiceStartMode.Manual )

    do
        new ServiceProcessInstaller(Account = ServiceAccount.LocalSystem)
        |> base.Installers.Add |> ignore

        serviceInstaller
        |> base.Installers.Add |> ignore

    override this.Install(state) =
        match this.Context.Parameters.ContainsKey("ServiceName") with
        | true ->
            serviceInstaller.DisplayName <- this.Context.Parameters.["ServiceName"]
            serviceInstaller.ServiceName <- this.Context.Parameters.["ServiceName"]
        | false -> ()
        base.Install(state)

    override this.Uninstall(state) =
        match this.Context.Parameters.ContainsKey("ServiceName") with
        | true ->
            serviceInstaller.DisplayName <- this.Context.Parameters.["ServiceName"]
            serviceInstaller.ServiceName <- this.Context.Parameters.["ServiceName"]
        | false -> ()
        base.Uninstall(state)

module Main =

    ServiceBase.Run [| new Svc() :> ServiceBase |]

    //let endPoint = new EndPoint()
    //endPoint.StartEndPoint()
    //Console.ReadLine() |> ignore
```

SvcHost.fs

2. Install the service from command prompt (run as Administrator):

```
sc create "MyService" binPath= "[Path to the executable your project built]"
```

3. Start the service:

```
net start MyService
```

4. Try it in Fiddler

Improvement #2 – Returning something more interesting

1. Add a new file “Contracts.fs” to the project:

```
namespace Example

open System
open System.Runtime.Serialization

module Outputs =

    [<CLIMutable>]
    [<DataContract>]
    type Product =
        {
            [<field: DataMember(Name = "Descr")>]
            Descr: string

            [<field: DataMember(Name = "Price")>]
            Price: decimal
        }

    [<CLIMutable>]
    [<DataContract>]
    type Person =
        {
            [<field: DataMember(Name = "Id")>]
            Id: int

            [<field: DataMember(Name = "Name")>]
            Name: string

            [<field: DataMember(Name = "Age")>]
            Age: int

            [<field: DataMember(Name = "Products")>]
            Products: Product list
        }
```

Contracts.fs

2. Make additions highlighted in yellow to Controllers.fs

```
namespace Example

open System.Web.Http
open Example.Outputs
```

```
type PingController() =
    inherit ApiController()

    [<Route("api/ping")>]
    [<HttpGet>]
    member this.Get() =
        // sprintf "%A" this.Request |> log
        "Hello World!"

type PersonController() =
    inherit ApiController()

    [<Route("api/people/{personId}")>]
    [<HttpGet>]
    member this.Get(personId) =
        {    Id = personId
             Person.Name = "Bob"
             Age = 40
             Products = [{Descr = "Best Product"; Price = 34.95M }]}
```

3. Run the application and make the following call through Fiddler:

   Request:

```
GET http://localhost:4445/api/people/123 HTTP/1.1
Accept: application/json
Content-Type: application/json;charset=UTF-8
Host: localhost:4445
```

   Response:

```
HTTP/1.1 200 OK
Content-Length: 84
Content-Type: application/json; charset=utf-8
Server: Microsoft-HTTPAPI/2.0
Date: Tue, 10 Nov 2015 22:48:08 GMT

{"Id":123,"Name":"Bob","Age":40,"Products":[{"Descr":"Best Product","Price":34.95}]}
```

Improvement #3 – Accepting more interesting input

4. Make additions to “Controllers.fs”:

```
namespace Example

open System.Web.Http
open System.Net
open System.Net.Http
open Example.Outputs

type PingController() =
    inherit ApiController()

    [<Route("api/ping")>]
    [<HttpGet>]
    member this.Get() =
        // sprintf "%A" this.Request |> log
        "Hello World!"

type PersonController() =
    inherit ApiController()

    [<Route("api/people/{personId}")>]
    [<HttpGet>]
    member this.Get(personId) =
        {   Id = personId
            Person.Name = "Bob"
            Age = 40
            Products = [{Descr = "Best Product"; Price = 34.95M }]}

    [<Route("api/people/{personId}")>]
    [<HttpPost>]
    member this.Post([<FromBody>] personInput:Person) =
        // TODO: Validate input and send a command to the domain
        this.Request.CreateResponse(HttpStatusCode.Created)
```

Controllers.js

Improvement #4 – Let's get the caller's IP address

We'll capture the caller's IP address and compare it to the local IP address (both IPv4 and IPv6). If local, we'll let the caller through. Otherwise, we'll reject the request.

5. Add a new reference to the project:

   System.ServiceModel

6. Change the base address in the App.config file to the machine name (instead of localhost)
7. Add a file to the project called "Handlers.fs"

```
namespace Example

open System.Net
open System.Net.Http
open System.ServiceModel.Channels
open System.Threading.Tasks
open Microsoft.Owin
```

```
module Handlers =

    let getIpAddress (request: HttpRequestMessage) =
        let owinCtx =
            match request.Properties.ContainsKey("MS_OwinContext") with
            | true ->
                Some(request.Properties.["MS_OwinContext"] :?> OwinContext)
            | false ->
                None
        match owinCtx with
        | None ->
            match request.Properties.ContainsKey(RemoteEndpointMessageProperty.Name) with
            | true ->
                let remote =
                    request.Properties.[RemoteEndpointMessageProperty.Name] :?>
                        RemoteEndpointMessageProperty
                remote.Address
            | false -> ""
        | Some(owinCtx) -> owinCtx.Request.RemoteIpAddress

    let private getLocalIpAddresses () =
        let ipv6 =
            Dns.GetHostEntry(Dns.GetHostName()).AddressList
            |> Array.filter(fun a ->
                a.AddressFamily = System.Net.Sockets.AddressFamily.InterNetworkV6)
            |> List.ofArray
            |> List.head
        let ipv4 =
            Dns.GetHostEntry(Dns.GetHostName()).AddressList
            |> Array.filter(fun a ->
                a.AddressFamily = System.Net.Sockets.AddressFamily.InterNetwork)
            |> List.ofArray
            |> List.head

        (ipv6.ToString(), ipv4.ToString())

    let ipHandler =
        { new DelegatingHandler() with
            member x.SendAsync(request, cancellationToken) =
                let (localIpv6, localIpv4) = getLocalIpAddresses()
                match getIpAddress request with
                | ip when ip = localIpv6 ->
                    printfn "%A" ip
                    base.SendAsync(request, cancellationToken)
                | ip when ip = localIpv4 ->
                    base.SendAsync(request, cancellationToken)
                | ip ->
                    printfn "%A" ip
                    let tcs = new TaskCompletionSource<HttpResponseMessage>()
                    tcs.SetResult(new HttpResponseMessage(HttpStatusCode.Forbidden))
                    tcs.Task }
```

Handlers.fs

8. Make the following addition to “EndPoint.fs”:

```
namespace Example

open System
open Owin
open System.Web.Http
open System.Configuration
open Microsoft.Owin.Hosting

type Startup() =
    member this.Configuration(app: IAppBuilder) =
        try
            let config = new HttpConfiguration()
            config.MapHttpAttributeRoutes()
            config.MessageHandlers.Add(Handlers.ipHandler)
            //app.UseCors(CorsOptions.AllowAll) |> ignore
            app.UseWebApi(config) |> ignore
        with ex ->
            printfn "%A" ex
            //sprintf "%A" ex |> log

type EndPointState =
    | Running of IDisposable
    | NotRunning

type AgentSignal =
    | Start
    | Stop


type EndPoint() =
    let agent = MailboxProcessor.Start(fun inbox ->
        let rec loop state = async {
            let! msg = inbox.Receive()
            match msg, state with
            | Start, NotRunning ->
                let baseAddress = ConfigurationManager.AppSettings.["baseAddress"]
                let server = WebApp.Start<Startup>(baseAddress)
                //sprintf "Endpoint listening at %s" baseAddress |> log
                return! loop (Running(server))
            | Start, Running(_)
            | Stop, NotRunning ->
                return! loop state
            | Stop, Running(server) ->
                server.Dispose()
                return! loop NotRunning
        }
        loop NotRunning)

    member this.StartEndPoint() =
        agent.Post(Start)

    member this.StopEndPoint() =
        agent.Post(Stop)
```

EndPoint.fs

Improvement #5 – Let's switch to HTTPS!

9. Make a certificate:
   a. Run Developer Command Prompt as Administrator
   b. `makecert -r -pe -n CN=Example -ss my sha1 -sr localmachine -sky exchange`
10. Get the thumbprint of the cert:
    a. Inetmgr
    b. Select the server node and double-click "Server Certificates"
    c. Double-click "Example" certificate
    d. Click on "Details"
    e. Click on "Thumbprint"
    f. Copy and paste into Notepad
    g. Carefully remove spaces

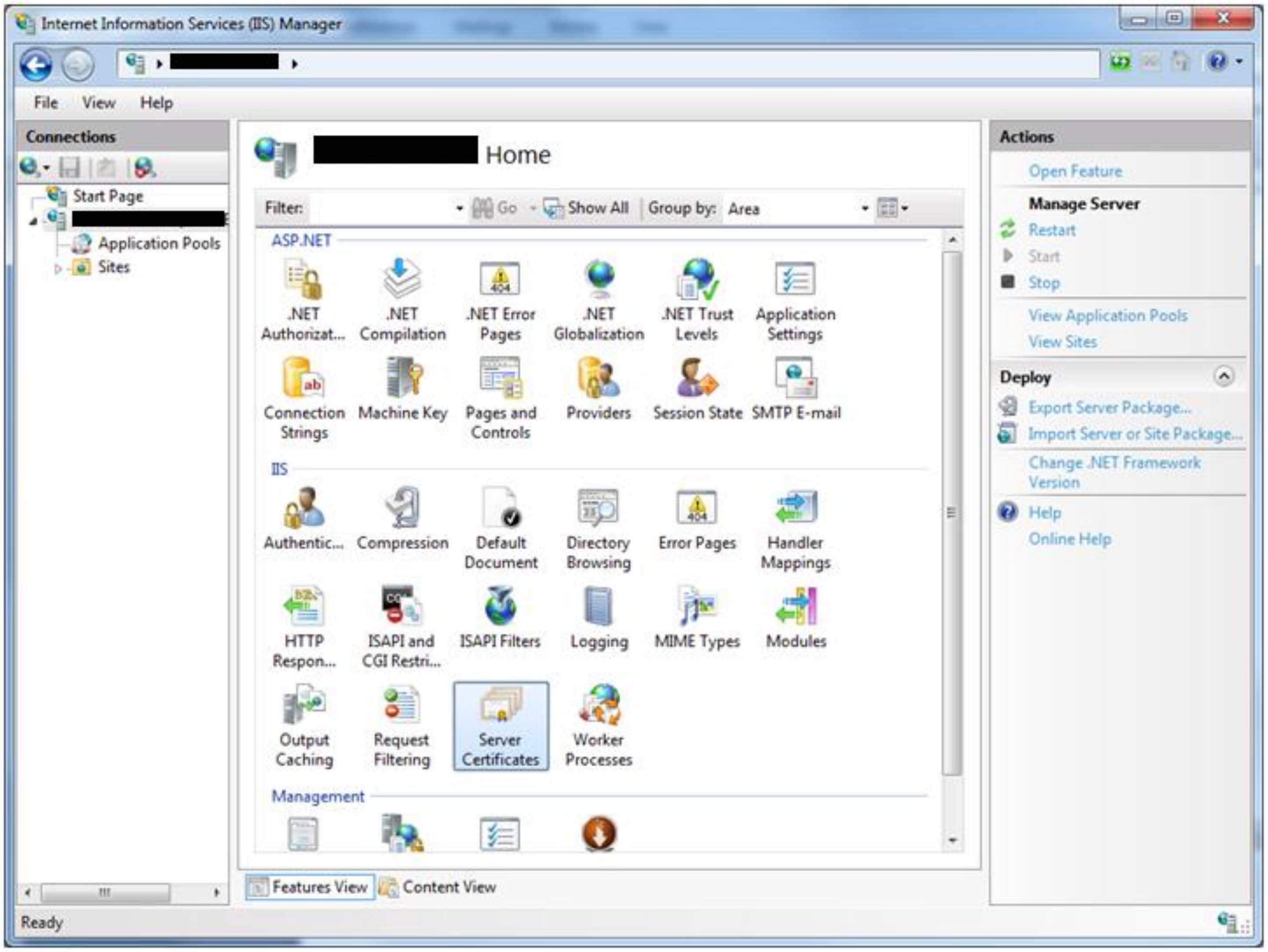
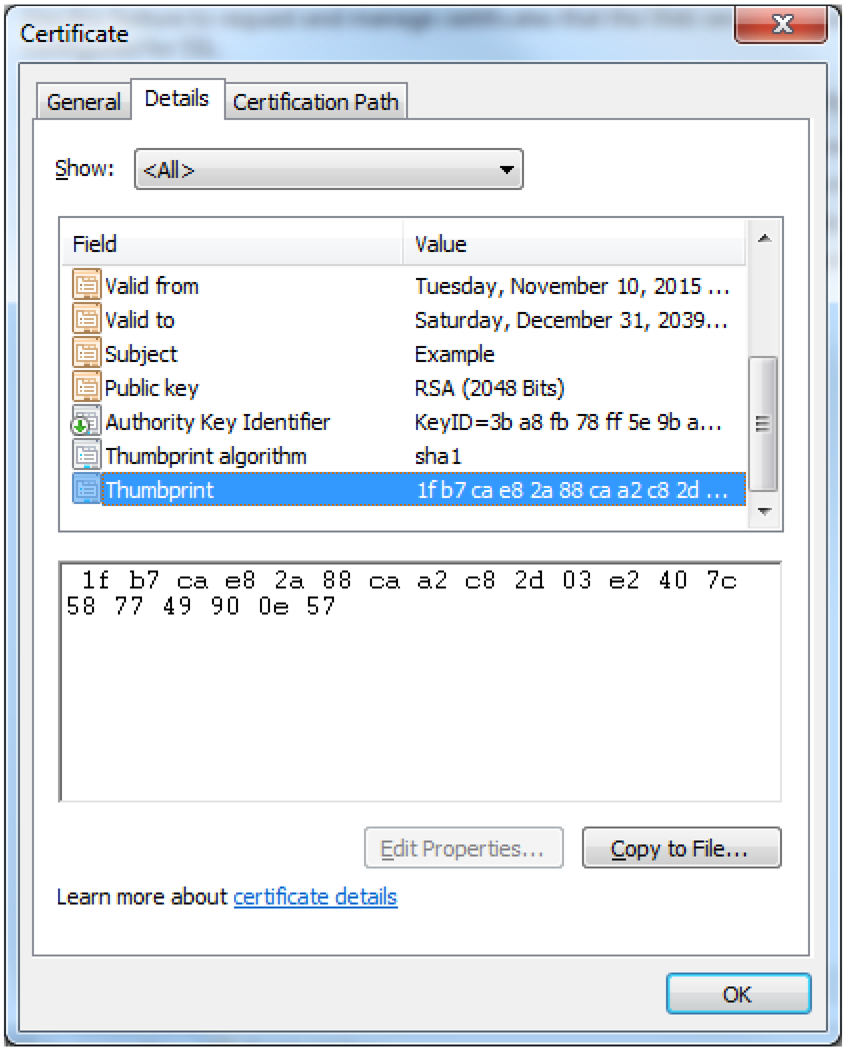


Figure 1 (at step 20.b)



Show: <All>

Field	Value
Valid from	Tuesday, November 10, 2015 ...
Valid to	Saturday, December 31, 2039...
Subject	Example
Public key	RSA (2048 Bits)
Authority Key Identifier	KeyID=3b a8 fb 78 ff 5e 9b a...
Thumbprint algorithm	sha1
Thumbprint	1f b7 ca e8 2a 88 ca a2 c8 2d ...

```
 1f b7 ca e8 2a 88 ca a2 c8 2d 03 e2 40 7c
58 77 49 90 0e 57
```

Learn more about certificate details

Figure 2 (at step 20.e)

11. Get the App Id of the application
    a. Open the .sln file in Notepad
    b. Copy GUID and curly braces from GlobalSection where it is repeated several times.
12. Assert the certificate with the application
    a. Open VS Command Prompt as Administrator
    b. `netsh http add sslcert ipport=0.0.0.0:4445 certhash=[Thumbprint goes here] appid=[App Id goes here]`

NOTE: If you need to disassociate a cert:

```
netsh http delete sslcert ipport=0.0.0.0:4445
```

NOTE: If you need to delete a cert:

mmc on Command line

File > Add/Remove Snap-in…
Certificates
Computer
Under Personal/Certificates you'll find the ones we have created here and should be able to delete them.

13. Change the App.config to have a baseAddress starting with "https" instead of "http"
14. Try the api/ping request with just http. It will likely hang or timeout with a bad request. Try again with https in the URL. You may be prompted about the certificate being bad.

Improvement #6 – Let's issue a token to the caller

15. Add a new file to the project "CryptoHelpers.fs"

```
namespace Example

open System
open System.Text
open System.Security.Cryptography
open System.Security.Cryptography.X509Certificates

module CryptoHelpers =

    let subjectName = "CN=Example"

    let enumerate (collection:System.Collections.IEnumerable) =
        let en = collection.GetEnumerator()
        let rec loop (en:System.Collections.IEnumerator)(i:int) result =
            match en.MoveNext() with
            | true ->
                loop en (i+1) (en.Current :?> 'a :: result)
            | false -> result
        loop en 0 List.empty

    let getX509Certificate =
        let store = new X509Store(StoreName.My, StoreLocation.LocalMachine)
        store.Open(OpenFlags.ReadOnly)
        let cert =
            let certs =
                store.Certificates
                |> enumerate
                |> List.filter(fun (f:X509Certificate2) ->
                    f.SubjectName.Name.Equals(subjectName, StringComparison.OrdinalIgnoreCase))
            match certs |> List.isEmpty with
            | true -> None
            | false -> Some(certs.Head)
        store.Close()
        cert

    let encrypt (cert:X509Certificate2) (plainToken:string) =
        let crypto = cert.PublicKey.Key :?> RSACryptoServiceProvider
        crypto.Encrypt(Encoding.UTF8.GetBytes(plainToken),true)
        |> Convert.ToBase64String
    let decrypt (cert:X509Certificate2) (encryptedToken:string) =
        let crypto = cert.PrivateKey :?> RSACryptoServiceProvider
        let v =
            encryptedToken
            |> Convert.FromBase64String
        crypto.Decrypt(v, true)
        |> Encoding.UTF8.GetString

    let createToken (userName, ipAddress) =
        let txt = sprintf "%s;%s" userName ipAddress
        match getX509Certificate with
        | None -> String.Empty
        | Some(cert) -> encrypt cert txt

    let parseToken (token:string) =
        try
            match getX509Certificate with
            | None -> (String.Empty,String.Empty)
            | Some(cert) ->
                match decrypt cert token with
                | null -> (String.Empty, String.Empty)
                | decrypted ->
                    let parts = decrypted.Split([|';'|])
                    parts.[0], parts.[1]
        with ex ->
            printfn "%A" ex
            new Exception("Bad Token") |> raise
```

CryptoHelpers.fs

NOTE: We deliberate obscure the exception so that information about implementation is not passed to the caller.

16. Add a new contract to the Contracts.fs file:

```
namespace Example

open System
open System.Runtime.Serialization

module Outputs =

    [<CLIMutable>]
    [<DataContract>]
    type Product =
        {
            [<field: DataMember(Name = "Descr")>]
            Descr: string

            [<field: DataMember(Name = "Price")>]
            Price: decimal
        }

    [<CLIMutable>]
    [<DataContract>]
    type Person =
        {
            [<field: DataMember(Name = "Id")>]
            Id: int

            [<field: DataMember(Name = "Name")>]
            Name: string

            [<field: DataMember(Name = "Age")>]
            Age: int

            [<field: DataMember(Name = "Products")>]
            Products: Product list
        }

    [<CLIMutable>]
    [<DataContract>]
    type Login =
        {
            [<field: DataMember(Name = "Username")>]
            Username: string

            [<field: DataMember(Name = "Pwd")>]
            Pwd: string
        }
```

Contracts.fs

17. Add a controller to the Controllers.fs file:

```
namespace Example

open System.Web.Http
open System.Net
open System.Net.Http
open Example.Outputs
open Example.CryptoHelpers

type PingController() =
    inherit ApiController()

    [<Route("api/ping")>]
    [<HttpGet>]
    member this.Get() =
        // sprintf "%A" this.Request |> log
        "Hello World!"

type PersonController() =
    inherit ApiController()

    [<Route("api/people/{personId}")>]
    [<HttpGet>]
    member this.Get(personId) =
        {   Id = personId
            Person.Name = "Bob"
            Age = 40
            Products = [{Descr = "Best Product"; Price = 34.95M }]}

    [<Route("api/people/{personId}")>]
    [<HttpPost>]
    member this.Post([<FromBody>] personInput:Person) =
        this.Request.CreateResponse(HttpStatusCode.Created)

type LogInController() =
    inherit ApiController()

    [<Route("api/login")>]
    [<HttpPost>]
    member this.Post([<FromBody>] loginInput: Login) =
        printfn "%A" loginInput
        match loginInput with
        | { Username = "John"; Pwd = "Password1" } ->
            createToken (loginInput.Username, (Handlers.getIpAddress this.Request))
        | _ -> ""
```

Controller.fs

18. Try the login operation now:

Request:

```
POST https://[your machine name here]:4445/api/login HTTP/1.1
Accept: application/json
Content-Type: application/json;charset=UTF-8
Host: [your machine name here]:4445
Content-Length: 38

    {"Username":"John", "Pwd":"Password1"}
```

Response:

```
HTTP/1.1 200 OK
Content-Length: 346
Content-Type: application/json; charset=utf-8
Server: Microsoft-HTTPAPI/2.0
Date: Wed, 11 Nov 2015 01:20:21 GMT

"dx6cVx0NTMMLxK2GI6kJnhZBTe6+QujtWkQJpzTY61qGIHX60jNtvFgGh3JcUKp04Av3+vD2OVW8T0k4p5WDfKD+yFfYOqNwoBIT
+OAjg2CEIVT9X4+Ahzy5ZngYJK3VaBXfVKTkgNahpwVdaeYr30adphxqwqOdmVGyUGI0D3/ZzMiic33vGrqAvNwjrXbRyCMoqvvho5
RBX5RkxtuWv+Dx3J3apVlJ7CXZRGI6DbKrPkjlNyX9f6cCxxH5NVPsFGl9UzLbBiJTyaNIWCNQXHLMNIOQi5JUYOqrMj6GSckxvNRj
0Hcn7+LZctUmgdmj7OB6hH/LybdcPXTonXLKDQ=="
```

This token is what we will use when making authenticated calls going forward.

Improvement #7 – Let's make sure request have valid tokens

19. Add a new DelegatingHandler to the Handlers.fs file:

```
namespace Example

open System.Net
open System.Net.Http
open System.ServiceModel.Channels
open System.Threading.Tasks
open Microsoft.Owin

module Handlers =

    let getIpAddress (request: HttpRequestMessage) =
        let owinCtx =
            match request.Properties.ContainsKey("MS_OwinContext") with
            | true ->
                Some(request.Properties.["MS_OwinContext"] :?> OwinContext)
            | false ->
                None
        match owinCtx with
        | None ->
            match request.Properties.ContainsKey(RemoteEndpointMessageProperty.Name) with
            | true ->
                let remote =
                    request.Properties.[RemoteEndpointMessageProperty.Name] :?>
                        RemoteEndpointMessageProperty
                remote.Address
            | false -> ""
        | Some(owinCtx) -> owinCtx.Request.RemoteIpAddress

    let private getLocalIpAddresses () =
        let ipv6 =
            Dns.GetHostEntry(Dns.GetHostName()).AddressList
            |> Array.filter(fun a ->
                a.AddressFamily = System.Net.Sockets.AddressFamily.InterNetworkV6)
            |> List.ofArray
            |> List.head
        let ipv4 =
            Dns.GetHostEntry(Dns.GetHostName()).AddressList
            |> Array.filter(fun a ->
                a.AddressFamily = System.Net.Sockets.AddressFamily.InterNetwork)
            |> List.ofArray
            |> List.head

        (ipv6.ToString(), ipv4.ToString())

    let ipHandler =
        { new DelegatingHandler() with
            member x.SendAsync(request, cancellationToken) =
                let (localIpv6, localIpv4) = getLocalIpAddresses()
                match getIpAddress request with
                | ip when ip = localIpv6 ->
                    printfn "%A" ip
                    base.SendAsync(request, cancellationToken)
                | ip when ip = localIpv4 ->
                    base.SendAsync(request, cancellationToken)
                | ip ->
                    printfn "%A" ip
                    let tcs = new TaskCompletionSource<HttpResponseMessage>()
                    tcs.SetResult(new HttpResponseMessage(HttpStatusCode.Forbidden))
                    tcs.Task }

    let private tokenHeaderName = "X-Token"

    let tokenInspector =
        { new DelegatingHandler() with
            member x.SendAsync(request, cancellationToken) =
                match request.RequestUri.ToString().Contains("api/login") with
                | false ->
                    let (userName,ipAddress) =
                        match request.Headers.Contains(tokenHeaderName) with
                        | true ->
                            request.Headers.GetValues(tokenHeaderName)
                            |> Seq.head
                            |> CryptoHelpers.parseToken
                        | false -> ("","")
                    match getIpAddress request with
                    | x when x = ipAddress ->
                        base.SendAsync(request, cancellationToken)
                    | _ ->
                        request.CreateErrorResponse(HttpStatusCode.BadRequest, "Invalid token")
                        |> Task.FromResult
                | true -> base.SendAsync(request,cancellationToken)  }
```

Handlers.fs

20. Wire up the tokenInspector:

```
namespace Example

open System
open Owin
open System.Web.Http
open System.Configuration
open Microsoft.Owin.Hosting

type Startup() =
    member this.Configuration(app: IAppBuilder) =
        try
            let config = new HttpConfiguration()
            config.MapHttpAttributeRoutes()
            config.MessageHandlers.Add(Handlers.ipHandler)
            config.MessageHandlers.Add(Handlers.tokenInspector)
            //app.UseCors(CorsOptions.AllowAll) |> ignore
            app.UseWebApi(config) |> ignore
        with ex ->
            printfn "%A" ex
            //sprintf "%A" ex |> log

type EndPointState =
    | Running of IDisposable
    | NotRunning

type AgentSignal =
    | Start
    | Stop


type EndPoint() =
    let agent = MailboxProcessor.Start(fun inbox ->
        let rec loop state = async {
            let! msg = inbox.Receive()
            match msg, state with
            | Start, NotRunning ->
                let baseAddress = ConfigurationManager.AppSettings.["baseAddress"]
                let server = WebApp.Start<Startup>(baseAddress)
                //sprintf "Endpoint listening at %s" baseAddress |> log
                return! loop (Running(server))
            | Start, Running(_)
            | Stop, NotRunning ->
                return! loop state
            | Stop, Running(server) ->
                server.Dispose()
                return! loop NotRunning
        }
        loop NotRunning)

    member this.StartEndPoint() =
        agent.Post(Start)

    member this.StopEndPoint() =
        agent.Post(Stop)
```

EndPoint.fs

21. Build, run and let's try the ping method again, but this time with a token:

Request:

```
GET https://[your machine name here]:4445/api/ping HTTP/1.1
Accept: application/json
Content-Type: application/json;charset=UTF-8
Host: [your machine name here]:4445
Content-Length: 38
X-Token:
dx6cVx0NTMMLxK2GI6kJnhZBTe6+QujtWkQJpzTY61qGIHX60jNtvFgGh3JcUKp04Av3+vD2OVW8T0k4p5WDfKD+yFfYO
qNwoBIT+OAjg2CEIVT9X4+Ahzy5ZngYJK3VaBXfVKTkgNahpwVdaeYr30adphxqwqOdmVGyUGI0D3/ZzMiic33vGrqAvNwjr
XbRyCMoqvvho5RBX5RkxtuWv+Dx3J3apVIJ7CXZRGI6DbKrPkjINyX9f6cCxxH5NVPsFGI9UzLbBiJTyaNIWCNQXHLMNIO
Qi5JUYOqrMj6GSckxvNRj0Hcn7+LZctUmgdmj7OB6hH/LybdcPXTonXLKDQ==

        {"Username":"John", "Pwd":"Password1"}
```

Response:

```
HTTP/1.1 200 OK
Content-Length: 14
Content-Type: application/json; charset=utf-8
Server: Microsoft-HTTPAPI/2.0
Date: Wed, 11 Nov 2015 01:30:12 GMT

        "Hello World!"
```