

T. H. E. Solution LLC

Taming the development of intelligent products

An Embeddable Standards Compliant Web Service Server
A Secure Implementation

Terry H. Ess
T. H. E. Solution LLC

INTRODUCTION

The quote that by 200x there will be more smart devices on the Internet than people has become widely accepted. There are a number of TCP/IP high-level protocols that have good potential for connecting smart devices depending on the application and platform. One of the most general (and complex) is the evolving series of transport and content protocols that are bundled under the banner web service. Roughly a year and a half ago our investigation found that we could implement a WS-I Basic Profile 1.0 compliant web service even on a pretty constrained platform (see www.the-solution-llc.com/embeddableserver.pdf). The weak point was the very limited security provided. Since then web service security has been addressed by both the WS-I Basic Profile 1.1 and WS-I Basic Security Profile 1.0. Regretfully, it is probably not practical to implement these on a limited resource platform. In fact both provide security that is significantly more flexible and complete than we generally need. So what would it take to make a fairly general-purpose server that would meet realistic security requirements and still be embeddable? This article will explore exactly that.

REQUIREMENTS

The design of the reference implementations was driven by five primary requirements;

1. Load characteristics assumptions.
2. The desire to support the general-purpose application level communications required for remote command and control.
3. The need to provide an interface that meets a “web service standard” likely to be acceptable for use by an IT organization.
4. The desire to make an implementation that can be easily ported to modest platforms. This includes a small footprint and a very limited set of platform resource dependencies.
5. The need to provide strong security that is sustainable on modest platforms.

These are exactly the same requirements we used in the original investigation with one exception, the wording “strong security”. Obviously this eliminates the simplistic measures included in the initial implementation. What does this mean? For most device communications we do not need data confidentiality, we require mutual authentication and data integrity assurance. These services must use commonly accepted strong cryptographic functions in a “correct” manner.

DESIGN

We can meet our requirements on limited resource platform by extending the original implementation with relatively minor application level modifications that provide mutual authentication and data integrity assurance. The external world knows the service from a formal definition, the WSDL document. The revised WSDL is provided in Appendix A. Beyond incorporating elements needed for security, it has a more generalized message structure so all key parameters are data within the XML command/response document. This is an important change if we use data integrity assurances (and not document level assurance).

Basic access security remains the same as in the original implementation:

1. The TCP/IP socket session will be the level at which access control is exercised.
2. User authentication and different authorization levels matched to command actions, get data, set data and start/top actions, provide access security.
3. At the beginning of each session the user must authenticate itself using a connection message.

Everything beyond this has been changed to meet the revised security requirements. The changes are summarized below:

1. Mutual authentication
 - a. Connection message – the pass element data is a shared secret, a password matched to an authorization level, encrypted using the password as the base key.
 - b. Connection response
 - i. The session element data is the session base key encrypted using the password as the base key.
 - ii. The pass element data is the password encrypted using the session base key.
2. Value integrity
 - a. The integrity element data is the hash of all data in the command/response (except of course the integrity element itself) encrypted using the session base key.
 - b. Using this in every command/response assures us that even if a socket is hijacked no harm will occur. This approach was used in the reference implementation.

The cryptography resources used in the implementation are summarized below:

1. SHA1 is used for hashing.
2. RC4 is used for encrypting/decrypting.
 - a. The base key is appended to an initialization vector (IV) and hashed to produce a 128 bit key.
 - b. At the start of a connection the base key is a shared secret (password).
 - c. Most of a session uses a randomly generated session number as the base key.
 - d. Each encryption uses a new initialization vector.

Both SHA1 and RC4 are fast and compact. Since RC4 is a streaming symmetric-key cipher, we do not need to worry about block sizes, padding etc. The use of a 128 bit key that is new for

each encryption avoids the known weakness of the RC4 cipher. The encoding and decoding process used in the implementation are summarized in flow diagrams provided in Figures 1 and 2.

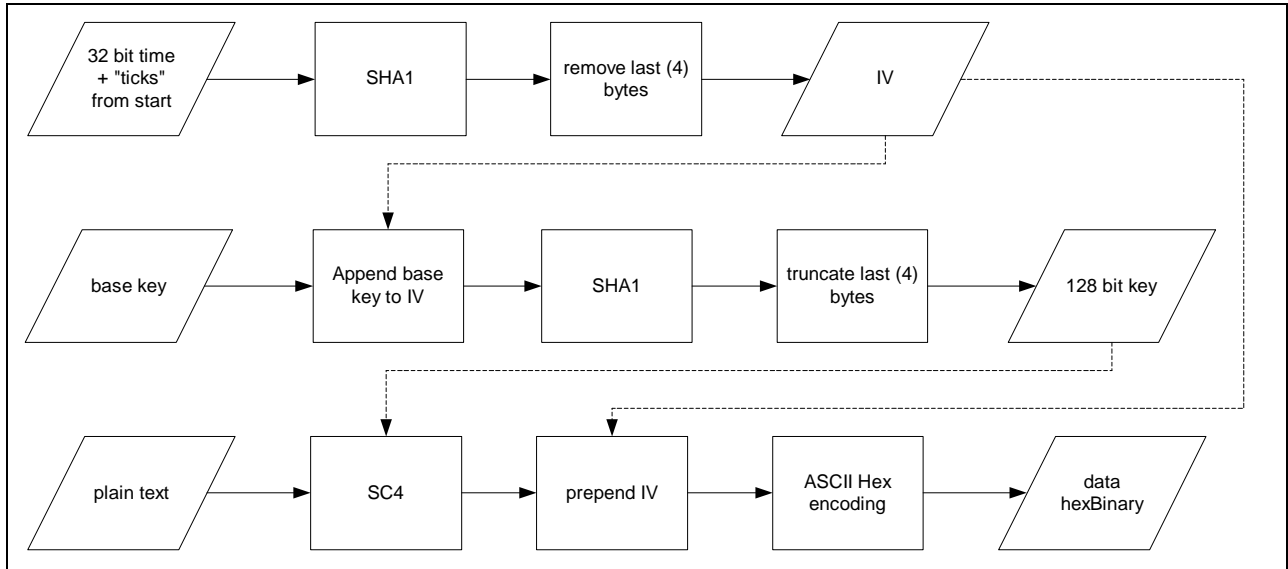


Figure 1, Encoding

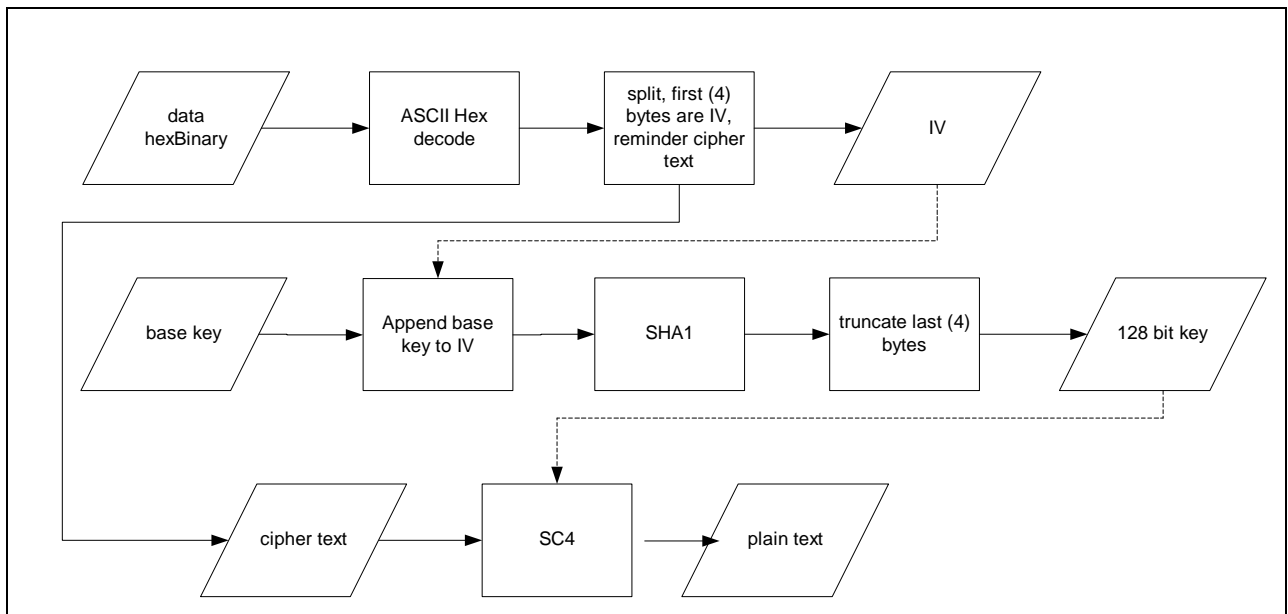


Figure 2, Decoding

RESULTS

A server reference implementation for a software based TCP/IP stack was developed as a Windows console application following the “embeddable” guidelines used in the original

implementation. It was tested against a client program written in C# .NET (Visual Studio 2003, .NET Framework 1.1). Access to the web service was automatically generated from the WSDL and the client program simply called stubs. The server implementation was just under 1000 lines of C++. A version that off-loaded the TCP/IP stack to a chip would take approximately 1400 lines of C++. This would take less than 20 Kbytes of ROM on a M68HC12 micro-controller, including library support and a small real time executive. The primary problem for very limited platforms is data space. Approximately 4 Kbytes of RAM would be required for each active client connection in the reference implementation. This could undoubtedly be driven down with careful design but it will be significant even then. In spite of this, it can easily fit in a wide variety of devices implemented on modest platforms if the number of simultaneous connections required is small.

There is a significant price for using the web service approach; the amount of data moved is significantly larger. Where a proprietary protocol running on top of TCP/IP could use less than ten bytes to perform a “get short” the initial implementation web service used over 500 bytes. The revised implementation is even worse at 650 bytes. The application level security also comes at a price. Both the client and server required software be created to perform the encryption/decryption etc. On the server side this included the actual SHA1 and RC4 implementations. Open source versions with no strings attached are readily available for these (and many other crypto functions) so this represents no real problem. Outside of the cryptographic code, this required less than 100 lines of code (C++ or C#) on both the server and client. So the application level security only adds a couple of man-days of effort to the implementation.

Because of the very generalized command/response structure, extending the reference implementation to different domains should be trivial. Performance was acceptable using both LAN and dial-up connections. The difference between them was noticeable by a human but not huge. Both of these were significantly slower than a “localhost” connection, indicating that the problem is not the security overhead but the wordiness of the web service and the realities of a TCP/IP based transport.

APPENDIX A, Revised Server Reference Implementation WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
  targetNamespace="http://device/def/"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://device/def/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <types>
    <xs:schema targetNamespace="http://device/def/" xmlns="http://device/def/">

      <xs:complexType name="ConnectParameters">
        <xs:sequence>
          <xs:element name="user" type="xs:string"/>
          <xs:element name="pass" type="xs:hexBinary"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="ConnectResponse">
        <xs:sequence>
          <xs:element name="Session" type="xs:hexBinary"/>
          <xs:element name="pass" type="xs:hexBinary"/>
        </xs:sequence>
      </xs:complexType>
      <xs:element name="Connect" type="ConnectParameters"/>
      <xs:element name="ConnectResp" type="ConnectResponse"/>

      <xs:complexType name="CommandParameters">
        <xs:sequence>
          <xs:element name="action" type="xs:string"/>
          <xs:element name="type" type="xs:string"/>
          <xs:element name="id" type="xs:short" minOccurs="0" maxOccurs="1"/>
          <xs:element name="value" type="xs:short" minOccurs="0" maxOccurs="1"/>
          <xs:element name="integrity" type="xs:hexBinary"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="CommandResponse">
        <xs:sequence>
          <xs:element name="action" type="xs:string"/>
          <xs:element name="type" type="xs:string"/>
          <xs:element name="id" type="xs:short" minOccurs="0" maxOccurs="1"/>
          <xs:element name="value" type="xs:short" minOccurs="0" maxOccurs="1"/>
          <xs:element name="result" type="xs:boolean"/>
          <xs:element name="integrity" type="xs:hexBinary"/>
        </xs:sequence>
      </xs:complexType>
      <xs:element name="Command" type="CommandParameters"/>
      <xs:element name="CommandResp" type="CommandResponse"/>

    </xs:schema>
  </types>
  <message name="ConnectResponseMessage">
    <part element="tns:ConnectResp" name="parameters"/>
  </message>
  <message name="ConnectMessage">
    <part element="tns:Connect" name="parameters"/>
  </message>

  <message name="CommandResponseMessage">
    <part element="tns:CommandResp" name="parameters"/>
  </message>
  <message name="CommandMessage">
    <part element="tns:Command" name="parameters"/>
  </message>

  <portType name="DeviceIO">
    <operation name="Connect">
      <input message="tns:ConnectMessage"/>
      <output message="tns:ConnectResponseMessage"/>
    </operation>
  </portType>
</definitions>
```

```
</operation>

<operation name="Command">
  <input message="tns:CommandMessage" />
  <output message="tns:CommandResponseMessage" />
</operation>
</portType>

<binding name="DeviceIOBinding" type="tns:DeviceIO">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="Connect">
    <soap:operation soapAction="DeviceIO#Connect" />
    <input>
      <soap:body use="literal" />
    </input>
    <output>
      <soap:body use="literal" />
    </output>
  </operation>

  <operation name="Command">
    <soap:operation soapAction="DeviceIO#Command" />
    <input>
      <soap:body use="literal" />
    </input>
    <output>
      <soap:body use="literal" />
    </output>
  </operation>
</binding>

<service name="Service">
  <port binding="tns:DeviceIOBinding" name="DevicePort">
    <soap:address location="http://localhost:2000/DeviceD" />
  </port>
</service>
</definitions>
```