

# Proposed Security Approach for message Exchanges between AZTech™ centers using TMDD and WSDL messaging

Revision 05 DC Kelley [davidkelley@itsware.net](mailto:davidkelley@itsware.net)

This short white paper outlines some key design requirements expressed in the stakeholder meeting that need to be incorporated in the different computer systems that will be exchanging XML messages and allowing access to ITS devices under their control. The purpose of this paper is to establish a common understanding of the overall threat model and to narrow the scope of solutions that would meet these needs and be practical. This paper does not propose a single vendor solution. It does propose several *check-off* items that many commercial vendors can provide and which, if followed, will provide adequate<sup>1</sup> security measures for the program.

There are many footnotes provided here that may be of interest to those that want to learn more about this topic.

## Summary:

Choose any one of the popular PKI 2-part encryption methods (X.509<sup>2</sup> certificates, etc.) which will be used by all participating centers. The message generating center will create an encrypted message using his (private) key that involves the complete message, as well as encrypting an inner portion with selected elements (detailed below). The message receiver will decode the message using his key.<sup>3</sup> Keys for outsiders can also be provided.<sup>4</sup> Open and decoded past messages passing about inside each system will not lead to comprising the security system. Due to the aspects of this process the message can not be spoofed or replayed by others. Insiders in each center decoding the message, including its password area, must still compromise the encryption method to some degree in order to use a replay attach or to create imposter messages. The validity interval for this type of attach would be under 10 minutes in any event. Details of this are outlined in the pages that follow, but this is believed to be simple to implement, low cost, and robust.

---

<sup>1</sup> All such systems can be broken with time and effort, particularly those where the message structure and content are more or less known, such as in this case. If you are interested, search for “plain text attack” on any cryptology web site. For the purposes of this paper we need to decide how secure the message needs to be expressed in how long a skilled attacker might take to break into it armed with all the right tools. I have arbitrarily picked at least one hour, for someone intimately aware of what the message should look like and what much of its content is and whose only problem is to decode it. For someone to determine *how* it was encoded (a much harder problem than decoding *one* message *one* time, in fact it means they have compromised your encoding keys), I have picked at least 8 hours as the time.

<sup>2</sup> You do not need to know what is inside an X.509 certificate, but if you want to learn more review <http://www.ietf.org/rfc/rfc3280.txt> where you will see that many different approaches can be used. The following link is fairly negative on X.509 <http://www.cs.auckland.ac.nz/~pgut001/pubs/x509guide.txt>. For a nice terse answer to what this thing is, try: <http://java.sun.com/j2se/1.3/docs/guide/security/cert3.html>. This is a Java site and I should point out that similar tools come with any copy of Win Server 2003.

<sup>3</sup> He can use a public key provided by the sender, or have his own.

<sup>4</sup> It is common, and more secure, for *each* trading partner to have their own key from *each* partner that sent data. Given the small number of centers, this approach is recommended. Key distribution is not covered, but should be secure. You do not want the attacker to compromise the system here. It has been mentioned that some other centers, such as public media points, may also be given keys to decode things. In such a case a single key shared with all users of this class may be practical, but you need to consider whether you will be at risk. If the data is to be decoded only (nothing is ever sent or controlled from this center) then the risks are modest.

## The Basics

The subject of cryptography is very diverse and much of it has nothing to do with the problem at hand. I would be remiss if I did not cite one of the more popular books on this subject for those with an interest.<sup>5</sup> While this book contains some high order math, it is still quite readable by anyone who wants a summary of the issues and terminology in the field today. Another document that AZTech™ implementors and vendors will want to review is the OASIS SOAP security document.<sup>6</sup> I will make passing reference to the specific sections of these two documents in the footnotes that builders may want to consult for details. There is also a degree of noise out there in the marketplace about the strength of the various encryption solutions, and from what I can tell from your needs, the debates about the relative strength of one hashing algorithm over another are not germane to any of your requirements.<sup>7</sup>

I wanted to place a terse explanation of the Public Key Infrastructure (PKI)<sup>8</sup> system in the document, but so far I have not found the ideal reference. The critical concept we care about is that with one of two keys<sup>9</sup>, you can encrypt your message and send it in a secure form to another center where that party can use the other key and decrypt it back to its original form. The process allows the receiving party to confirm that the message was unchanged and came from who they think it did, hence building trust. It is generally immune from various attacks including someone at the receiving side (an insider) using the key they have to fake a message. You can buy keys from third parties such as Verisign, or build your own<sup>10</sup> with readily available tools. Keys are typically exchanged in a format called X.509 certificates. The keys are used when the message content is hashed to produce the encrypted data stream that is actually sent.

It should be noted that the ITS center-to-center document on using SOAP and WSDL (2306) provides no specific guidance on this topic, presuming you can overlay it as needed.

---

<sup>5</sup> Practical Cryptography by Niels Ferguson and Bruce Schneier and printed by Wiley.

<sup>6</sup> The formal title is: Web Services Security: SOAP Message Security 1.1 (WS-Security 2004) Published Feb 1, 2006. It is an OASIS published standard and can be obtained off the web at <http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf>. Be sure to get the newest one. I had to read this document several times before I could state that I find nothing wrong with it and much right with it. It does not provide a solution to the current problem, but like this paper, recommends things to do and things to avoid. We will probably be in conformance with it at the end of the day.

<sup>7</sup> In particular, the debates about the strength of the X.509 certificates and that the SHA1 algorithm can be compromised to reduce its protection level from  $2^{80}$  to  $2^{63}$  are not really an issue for your program, chose an X.509 vendor and get on with it.

<sup>8</sup> Here is an overview: <http://en.wikipedia.org/wiki/Pki>

<sup>9</sup> You might use one key for each end point of delivery. Keep in mind that the public/private key is simply the most common internet tool used for distributing data from one center to many other centers.

<sup>10</sup> The primary benefit of using someone like Verisign is that you are “chained” to a point of public trust. In your semi-closed system that has no financial transactions, this is of less value.

## The Threat Model

The threat model<sup>11</sup> we are defending against in this case is an attack from someone *inside* the centers attacking the systems with either replay attacks<sup>12</sup> or spoofed<sup>13</sup> messages. This person could be a disgruntled employee who has general access to one or more of the systems from the inside. We wish to prevent this person from harming the system. We also want to be able to detect and reject such messages. Put yourself in this role for a moment. Ask yourself if we are building a system that you would find difficult to exploit, given that you know its secrets and have access to it. If you can not see a way to exploit it without great expense, then perhaps we have done the job.

This attacker would be aware of the structure of the XML messages as well as the contents of many of the fields found within them (for example, all messages coming to his location may be addressed to center-a). This goes a long way to reducing the complexity of decoding a message because the attacker can more or less tell by these fields if the crack is right. It does not help in creating a message because of the message count value and time stamp issues. This may mean we are still slightly venerable to an attacker intercepting an incoming message within his own center, replacing critical data in an otherwise valid message during a short validity interval, and passing it on.<sup>14</sup>

This attacker would also be able to see the decrypted messages between the centers in plain text under many conditions.<sup>15</sup> In all likelihood, they would also be able to see the user-ID and passwords fields in the message in their XML forms in all likelihood. They would see the base-64 text of any non-human-readable security string that was passed at the same time. All of these artifacts are inputs to the tools used by a determined person to crack the system. And again, short of compromising the generating key, we expect to be protected against a skillful attacker who has obtained all this data.

Let us spend a moment more on the data itself. This data is frankly not of great value over any span of time. It is transitional at best and nearly worthless an hour after the event. Unlike a financial transaction or some sort of secret, there is no design need to ensure it will take years of effort to decode it. Our design requirements are more to ensure that it comes from who we think it does (*trust* and *authority* in cryptographic terms).

---

<sup>11</sup> Security systems are often classified in terms of outlining the threats which they defend against. Here, we have chosen the single worst case event as our basis for development. Other events do exist, such as an outsider trying to penetrate the system and doing the same things, but for the purposes of this study, this one threat pretty well sums it all up.

<sup>12</sup> Retransmitting a captured message, sometimes with changes just to confuse things. Many systems, including what is proposed here, have some short susceptibility period when a valid message might be re-used in some way to attack the system.

<sup>13</sup> A general term used in cryptography to represent any fake message.

<sup>14</sup> Because of the socket nature of the encrypting, the attacker would have to have compromised the generating key of the sender to do this, and would have to do it within the time interval limits matching the message sequence values expected. These concepts are explained a bit further on.

<sup>15</sup> We presume that user displays and screens will hide such information unless it serves a purpose to display it, thereby preventing casual observation of this information. We presume that the attacker knows where to dig to get it out. We suggest hiding this in a 2-level decoding of the XML message later in the paper, but for the threat model we presume it has been compromised. The first defense in all these systems is not to make it easy for the bad guy to gather the data.

The computers that operate between the centers<sup>16</sup> are presumed to be equipped with a reasonably accurate source of Coordinated Universal Time (UTC) time. Time is presumed to be monotonic. Time is often used in the cryptographic process as a way to validate that a message is correct. We will do so here as well. One of the common tricks of the trade in doing this is to establish a Time Validity Interval (TVI) which all users adhere to.<sup>17</sup> A value of ten minutes is proposed for this system as a useful limit and one which is sufficiently long such that no issue of overlapping TVIs will occur.<sup>18</sup> Jumping into the design for a moment; within the TVI the inner encoding of a user ID, password, time, etc. would remain constant and valid for the ten minute period. The outer encoding of the message in its socket would still be encoded in the normal way for each message complete with the precise time it occurred. Frankly, I have chosen this inner system to be an extreme nuisance to an attacker who is inside the other system, and not as a cryptographically strong solution.

Biometric measures, hardware dongles, and other specialist devices that would not make much sense in a central server computer have not been considered.

### Some Details

If you look at the XML Schema you will see a common recurring pattern in the XML as follows:

```
<user-id> Security-user-name</user-id>
<password> Security-password</password>
```

Directly following these we expect to be adding two new fields<sup>19</sup> to support the security model

```
<hashTime> Valid XML time and date </hashTime>
<hash> Some Base 64 string here </hash>
```

The first three new elements will be signed using the X.509 methods defined in XMLSEC and the signed hash value will appear in the final tag. Again, the SOAP security document has several examples of this in use. Because the time value here need only be updated once per 10 minutes (the TVI chosen), the processing load to re-calculate the hash value would be very low.

This information will be kept in the tag area called *Authorization Set* which is defined as follows (taken from a working draft) and used uniformly throughout the AZTech™ messages.

```
AuthorizationSet ::= SEQUENCE {
    user-id Security-user-name, -- #3108
```

---

<sup>16</sup> Not every end user computer, just the primate nodes that are exchanging data.

<sup>17</sup> The precise use of the term varies in different systems, so be careful. In many systems the actual key used changes on the TVI rate based on some known but hidden formula. The TVI rate chosen can vary from 24 hours to fractions of a second. Here, we want to use the term in the more relaxed sense that another center's credentials would remain valid until the next TVI.

<sup>18</sup> In some military warfare systems the TVI is very short and there is a problem with some users being rejected because they are still using the last value which must be overcome.

<sup>19</sup> In fact, these may be added at the end of the message, as ITS style suggests that as locally added data, it is better placed there.

```

password Security-password,      -- #3109
hashtime DateTimePair,
-- XML style date and time
hash     IA5String (SIZE(256)),
-- the security hash determined by
-- the x.509 cert
...
}

```

In its XML form this looks like the fragment below (for a graphical view, click [here](#) – this is part of the final schema documentation).

```

<xs:complexType name="AuthorizationSet" >
  <xs:sequence>
    <xs:element name="user-id" type="Security-user-name" />
    <xs:element name="password" type="Security-password" />
    <xs:element name="hashtime" type="DateTimePair" />
    <xs:element name="hash" >
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:length value="256"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

```

For the *hash* string we use the method provided by the SOAP Security standard (see footnote 19), particularly the one mentioned in Section 11 (Page 54) of the document in the examples.<sup>20</sup>

This part of the message is then, along with the rest of the message, encoded at the message set level by the Web Services Security level. For example, you can see it used as the third element in the DMS control request message [here](#).

By the way, one of the benefits of this overall method is that X.509 certificate for a given user (perhaps a recently terminated employee) can be quickly revoked and sent to other centers in a matter of minutes without changing the center-to-center certificate process in use. In such an event the user would no longer validate at the inner level, but the message (which is associated with the sending center and not the specific user) would.

Observe that the *user-id* value is, and may be, different than the *contact person name* used in the message. This allows you to have an entirely separate password set between the users and the messages or to share them as you see fit. You might choose to assign such user passwords to classes of users if you wanted to (don't it weakens your system).

You also need to decide which X.509 algorithms you want to support. This will determine to some degree what vendors you can use. The X.509 standard is itself a holder for certificates, some are good and some are not. You need to ask your vendor what specific "certificate algorithm identifiers" they support and make a choice.

---

<sup>20</sup> We may add some attributes to this element definition to better fit with XMLSEC and WSDL practices.

Here is a useful link on the pros and cons of getting your certificates from a commercial vendor such as VeriSign, versus doing it yourselves.

<http://www.davidpashley.com/articles/cert-authority.html>

### **A Socket Style Solution**

We will be using a socket style solution. The term here is used as socket in an internet and port type meaning. In this context, the message in its entirety fits within some form of socket and is encoded as a whole. The commercial solutions of SSL<sup>21</sup>, IPsec, VPN and SOAP services are all *sockets* in this context and any of these would work for this application if the encoding chosen is of moderate strength. Viewed as a *black box function*, you hand a completed XML message over to this layer and it pops out on the other end decoded and validated. You probably will get a time stamp and an IP value back as well.

### **Two Part Process Details**

The XML messages will be floating about in each center in three different formats which will be briefly mentioned here. In general, it will be difficult for the attacker to gather all of these, but it is presumed a determined attacker can do so.

**1**      ***Completely Encrypted message*** - This is the format an outside observer would likely capture if they could monitor the systems and if they could penetrate the basic network security. Before further decoding, the message exists in this form somewhere in the receiving and sending nodes.

**2**      ***Completely Decoded message*** - This is the format that exists on the other side of the decoding function and may be relatively secure. It exists on both the receiving and sending nodes, although the receiving node is of interest here. The userID-Password pair and any security string would be clearly visible in this message. In order to reach this point, the message was determined by the security layer to be valid. The cryptographic key decoding worked, the message count was greater than the last message, the time stamp is within the TVI, and the user security string checked out (user ID, password, and security string all match as expected). Invalid messages are rejected. Obtaining and storing copies of this “complete” message to compare with the encrypted version can lead to compromise, and should be made hard to do.

**3**      ***Downstream Decoded message*** - This is the format that exists beyond the decoding function as the now-validated message to send to other devices and user consoles. There is little reason to send the security strings (all content appearing inside the *AuthorizationSet* tags) with the user password to this point. This is true regardless of whether the message is pure XML or rendered into some displayed format. Suppressing these two elements from this point onward is a useful and easy security measure. All that need be known at this point is that the message is valid, and therefore the user is considered a “good one” and should be processed. Bear in

---

<sup>21</sup> If you are going to use SSL (which in fact uses PKI) use version 3, version 2 has many holes.

mind that if your application needs to learn about the user at this point in the message processing, the *Contact Details* tag section still has this information.

### **How to break this Approach**

While not exhaustive, the basic technique I would use is as follows. Get myself inside the IP realm of the target center so that I can originate messages from a valid IP or spoof this part of the process. If I cannot get inside the network, perhaps I can establish a Trojan node inside that will relay things for me. Observe and capture as many encoded and decoded messages as practical so that I have established a rich inventory of valid messages. I can use this to neck down the search paths I need when I finally break the system. From this, concentrate on the valid user data when possible as that part will likely be easier to compromise. I will also have to compromise the generating key being used on the overall message. From this, and with a ton of work, I can possibly decode both keys used. I would have a much greater chance of success in simply decoding individual messages, but this will not help me much in obtaining the key used. However, it may help me learn the algorithms that were selected, if I do not already know. If I am successful in actually decoding the encoding keys<sup>22</sup> used by a given center, I will have some window of opportunity as long as that key lasts. However in order to use these key to make a valid message, I will also need the correct time and message count. It is more likely an attacker will launch the message from inside, injecting messages that are already presumed decoded so this way the inner decoding is the only protection. If the attacker can get on the inside and break a user code sequence (protecting the inside network is a design point beyond the scope of this paper), then the 10 minute validity window will limit the attack unless they have compromised that key and can therefore regenerate valid looking information. If I am really lucky, I can replace the inner part of the message with my own key as this will likely check out as valid and the center may not bother to note that that the key I supplied (while valid) is not associated with the center which the message purports to come from.

### **Recommendations**

Now that the reader has reviewed all this, and read through footnotes for further information, perhaps some practical advice can be given on how to proceed and set up a suitable system for use on this project.

It has been mentioned that AZTech™ maintains an existing certificate issued by VeriSign that may form a suitable starting point to anchor the certificates of the proposed system.<sup>23</sup> It has also been mentioned that it is likely some centers will push data to RADS which will act as an information clearing house for other centers. These two items are used to presume an initial solution for common security.

---

<sup>22</sup> You will need at least two keys in this scenario, and perhaps three. You will need the “inner” key used to sign the User-ID (frankly any simple 509 certificate systems would be fine here) and then you need to be able to encrypt the outgoing message in order to fake a center. You may also need to decrypt the received message (using a third key for the receiving center) depending on where in this overall system you are able to inject your attacking message.

<sup>23</sup> This is properly called “chaining” and the basic concept is that you can extend trust (as well as the right to revoke trust) from one trusted player to another.

The first problem to be solved is to establish a center, preferably one, which will handle issuing the X.509 certificates to the other centers. I recommend you use the RADS center to perform this for others, and it is my understanding that the tools they currently use can easily perform this. You could chain this to the VeriSign CA<sup>24</sup> if you wish. The only value I see in that is that the owner of that certificate could perhaps shut the entire distributed system down quickly should the need occur (more on this in a moment).

Initially it will be easier if this center issues simple two-way certificates for everyone allowing secure publishing of a message and secure decoding of the message where all end receivers share the same common certificate for that center. Each center, and each user profile in the system, would have its own well-guarded encoding certificate as well. I suggest you establish a common expiration time for these, perhaps in the range of a couple of weeks. It is essential that the private side of these certificates be guarded against disclosure.

In time, each center will want to have a bit more control over this process and this is where certificate chaining may be very useful to you. In this scenario, the RADS center still has the mundane role of providing completed certificates to everyone in the deployment, but the certificates for *your* center's data/control messages are also derived from a certificate which *you* have provided as input to the RADS center as part of the process. This additional step complicates things to a modest degree, but it also allows each center to subsequently revoke any certificate issued from its provided credentials, hence a high degree of control is maintained. For centers that do not wish to become involved in this process, the first system described still works.

The next level of complexity is that rather than *sharing* the decoding side of the certificates (all centers with the common key from center X can decode and validate that a message came from center X), *each* center can provide unique and different keys (each chained to whatever tree of CAs you wish to use) to *each* other center to control what they can decode. Frankly, I do not see sufficient value in this step for this deployment and it complicates the certificates creation and distribution process considerably.<sup>25</sup> I would recommend you do not do this. You can achieve nearly the same security goals by simply not distributing your common keys to any party you do not want having it.

One final remark on the certificates themselves, you should certainly start with a two-part system allowing message encoding and then validation (the same sort of thing commonly found in the internet). However you do not need to end there. The X.509 system supports a rich set of permissions and attributes if you want to use them. You can, for example, use these certificate attributes to determine that a specific user and/or center is authorized to perform operation-X but not operation-Y as needed. You should discuss what level of detail might be needed long term, then build certificates that support this regardless of the time frame needed to implement it.<sup>26</sup>

---

<sup>24</sup> CA = Certificate Authority - the issuer of a certificate, which may in fact be based on other CAs too

<sup>25</sup> For X centers you would then need  $X^2$  certificates, rather than X certificates. With many end users as well as many centers this will quickly become difficult to manage.

How do you actually build a certificate? Well, in this case you simply download some inexpensive or free software for your favorite platform, and go to it. The Microsoft Server 2003 release comes with a certificate [builder](#) and services as part of the basic system. Similar tools are available on the Linux platform as well. I am told the that the tool [TinyCA](#) may be one that RADS wants to use, due in part to the open SSL backend.

Once everyone in the party has working X.509 certificates , you need to decide on some basic transport issues. I presume you will use Secure Sockets Layer (SSL) and WS-Security driven primarily based on what the group preferences in serve software are. My limited understanding of the AXIS-2 library that is used is that it fully supports both of these as well as an XML Encryption<sup>27</sup> library. I suggest you start by placing one of the WSDL messages *on-line* with some “hello world” dummy replies by using the X.509 certificates you have developed. Use this to successfully exchange *well-formed* messages between each other<sup>28</sup>, providing not only a level of confidence in the security system but a reference node for centers and vendors in the deployment to share and confirm successful inter-operations before integrating with each other.

**Addendum for programmers:**

The WSDL for the web services in this project can be downloaded at:  
<http://www.consystem.com/c2cxml/rtm/aztech.wsdl.xml>

The XSD (XML Schema file) for the message definitions can be downloaded at:  
<http://www.consystem.com/c2cxml/rtm/aztech.xsd.xml>

---

<sup>26</sup> The cost to change a certificate system which people have come to trust in daily usage is much higher than the cost to design in some attributes no one intends to use from the start.

<sup>27</sup> The term *XML Encryption* is actually yet another W3C standards effort, but here it refers to a library of routines found on the server that can easily decode the AuthorizationSet data frame used in the XSD schema that we have proposed for AZTech™ . Both JAXB/AXIS and Microsoft-NET have supported this for over 2 years now.

<sup>28</sup> In fact, you may want to build such an endpoint with no security layer, simply as a starting point for message exchanging. Such an endpoint, once working, would validate the well formed structure of the messages from each center/vendor by comparing them with the WSDL and XSD files, regardless of the security layer.