

# Towards Flexible Authorization Management

Patroklos G. Argyroudis

*Networks and Telecommunications Research Group  
Department of Computer Science  
University of Dublin, Trinity College  
argp@cs.tcd.ie*

Donal O'Mahony

*Centre for Telecommunications Value Chain Research  
Department of Computer Science  
University of Dublin, Trinity College  
omahony@cs.tcd.ie*

## Abstract

*During the last years there have been a lot of proposals in the literature for systems that attempt to manage the process of trust establishment. However, the engineering details related to the exchange and negotiation of authorization credentials have not received similar attention. Existing solutions like SSL/TLS and IPsec have limitations that minimize their applicability. In this paper we propose a new protocol, the Authorization eXchange Protocol (AXP), that provides a modular and extensible solution to this problem. It is situated between the application and the network layers acting as an authorization middleware component and handles the process of transmitting and receiving service access requests and replies, along with the credentials that are required to support them. In order to allow its use in securing delay sensitive applications, AXP has been designed to work over unreliable datagram transport protocols. We also present a case study and evaluate the performance of our proposal.*

## 1. Introduction

The problem of establishing and managing trust relationships has been extensively investigated in the literature. Authorization management systems like SPKI/SDSI [6, 8], KeyNote [4] and the attribute certificate (AC) extensions for X.509 [9] utilize public key cryptography in order to facilitate the establishment of security relationships and decide whether a principal is allowed to perform a specific action on a protected resource. Although these systems can be used to reach access control decisions, they provide no mechanism for exchanging the required authorization structures. Instead they rely on lower level protocols for this functionality. The two protocols that have been traditionally proposed for this purpose are SSL/TLS [7] and IPsec [11]. However, both of

these protocols have limitations that minimize their applicability in exchanging authorization credentials.

The Authorization eXchange Protocol (AXP) that we propose in this paper provides the functionality of negotiating and exchanging the authorization structures that are necessary to support access control in distributed networking environments. AXP has been designed in a modular way in order to support the credential formats used by different security management systems. Currently AXP supports X.509 identity and attribute certificates, KeyNote assertions, SDSI naming certificates, SPKI authorization certificates and the attribute and naming certificates of our own system,  $\text{\textit{ETHER}}$  [3]. In order to allow its use in securing delay sensitive application layer protocols, like the Real Time Protocol (RTP) [19] and the Media Gateway Control Protocol (MGCP) [2], AXP works over unreliable datagram protocols. In fact our prototype implementation is part of the ad hoc networking stack developed by the Networks and Telecommunications Research Group (NTRG) at the University of Dublin [15] that uses UDP [16] at the transport level.

The rest of this paper is structured as follows: In section 2 we briefly present the related work in this area and the remaining open problems. In section 3 we give an overview of the design requirements of AXP. In section 4 we describe our proposed protocol and its components. Section 5 discusses an application example of AXP and we conclude in section 6.

## 2. Related work

In this section we briefly present previous work on the problem of negotiating and exchanging security management data. We also explain the reasons they fail to provide a satisfactory solution.

The Secure Sockets Layer (SSL), the latest version of which is also known as Transport Layer Security (TLS), is by far the most widely deployed security

protocol [17]. TLS utilizes the X.509 identity-based security infrastructure in order to authenticate the peers that want to establish a secure channel. This is accomplished through the use of public key digital signatures and the corresponding X.509 public key certificates (PKCs). An X.509 PKC simply binds a (usually DNS) name with a public key, offering little help in supporting complex access control decisions. In order to allow the specification of authorization information, the X.509 attribute certificate profile has been recently proposed [9]. However, TLS provides no mechanism for exchanging and negotiating such authorization credentials or the associated access control policies. Furthermore, TLS is developed to protect TCP-based applications. Many delay sensitive applications, like Internet telephony for example, avoid the stream-based TCP due to its performance limitations and use UDP. Therefore, since TLS cannot be employed, one alternative is to develop a custom security solution. This usually requires a lot of effort and the end product cannot be reused with ease in other applications. The other alternative is to rely on IPsec.

IP-level Security (IPsec) consists of three different protocols that provide security services for any application that uses the Internet Protocol (IP) [11]. The Authentication Header (AH) and the Encapsulating Security Payload (ESP) are added to an IP datagram and provide authentication, integrity and confidentiality of the transmitted data. The Internet Key Exchange (IKE) protocol is used to negotiate the security association (SA) between two endpoints that need to communicate. A SA consists of the cryptographic keys and the negotiated algorithms supported by the peers needed to exchange data securely. Although IPsec supports X.509 PKCs and KeyNote assertions [5], it provides no extensible framework for adding support for additional security management systems. Furthermore, IPsec is implemented in the operating system kernel making it particularly inconvenient to deploy. Finally, IPsec has been criticized for being exceptionally complex and this fact hinders in depth security evaluations [17].

### 3. Requirements

During the initial investigation of the problem domain we have identified several specific requirements that must be addressed by AXP. We have divided these requirements into five general goals: Authorization mechanism independency, efficiency, modularity and extensibility, flexibility, and security.

In this section we analyze these goals into specific design requirements.

- *Authorization mechanism independency.* The protocol must provide the functionality of exchanging and negotiating authorization structures (like certificates, access control policies, access requests and replies) of many different security management systems by introducing an abstraction layer.
- *Efficiency.* AXP must not introduce significant performance penalties. In order to allow its use in securing delay sensitive applications, it must operate on top of unreliable datagram transport protocols, such as UDP. This requirement implies the need of a simple and lightweight session management mechanism that handles retransmission of unaccounted-for messages. An alternative that we have investigated in order to satisfy the efficiency requirement of AXP was a hybrid TCP/UDP solution. Key and session negotiation can take place over a TCP channel and bulk data transfer over a UDP channel. The security requirements of the separate datagram channel can be satisfied with the parameters negotiated over the TCP connection. However, after careful analysis we have rejected this hybrid approach since the synchronization of the two channels proved to be particularly complicated.
- *Modularity and extensibility.* AXP must be designed in a modular way that will facilitate the straightforward introduction of additions and functionality extensions. For example, the addition of support for new security management formats should be done with ease and without breaking protocol semantics or compatibility.
- *Flexibility.* The design of the protocol must allow higher level layers to specify application-related requirements. These can include for example the distribution method of access request supporting credentials (*push* or *pull*<sup>1</sup>) and whether identification of the

---

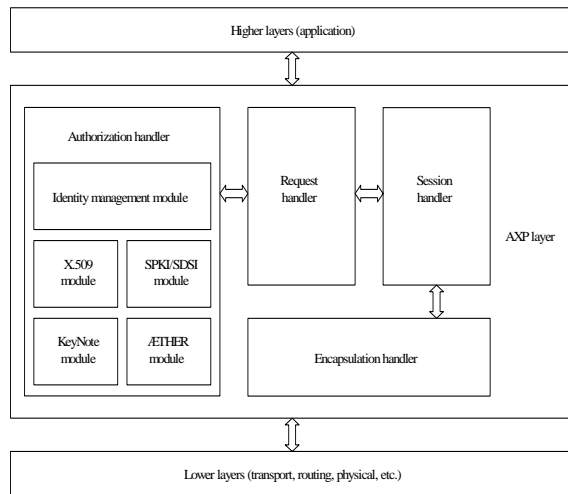
<sup>1</sup> In the *push* method the requesting party provides the supporting authorization structures to the verifier. In the *pull* method the verifier requests these from the issuer or a repository.

entity will occur by providing naming certificates.

- *Security*. AXP must satisfy the traditional security requirements of confidentiality and integrity for the exchanged data. Also, it must be as resistant as possible to denial-of-service attacks (availability).

#### 4. Design overview

In this section we present an overview of the design of AXP that has been engineered according to the previously identified requirements. The AXP layer is consisting of four different components that handle the provided functionality and are referred to as *handlers* (Figure 1).



**Figure 1. Architecture components of AXP**

- *Authorization handler*. This is the component of AXP responsible for handling the different modules that interface with the supported security management systems. There are modules for handling X.509 attribute certificates, SPKI authorizations, KeyNote assertions and ÆTHER attribute certificates. Moreover, the authorization handler has a special module for identity management since we treat identity information as a special kind of an authority attribute. This module can be enabled or disabled according to the session parameters negotiated by the *session handler* component and supports X.509 identity certificates and the naming certificates of SDSI and ÆTHER.

- *Request handler*. AXP also handles the transmission of access requests and decisions along with the supporting credentials and the relevant access control policies using a query/response model. The exact operation of the request handler is depended on the parameters negotiated during session establishment. Therefore, if the agreed method for credential distribution is *push*, the access request is accompanied with credentials for supporting it. If it is *pull*, the verifier retrieves them from a repository. Furthermore, this component maintains a cache of recently exchanged authorization information in order to optimize the trust establishment procedure. Cache maintenance depends on the decisions of the *authorization handler* and the session management performed by the *session handler*.

- *Session handler*. This component handles session establishment and maintenance, as well as the generation of key material used by the *encapsulation handler* for protecting the transmitted data. Based on requirements specified by the user (or the application programmer) the session handler negotiates the following parameters during session establishment:

- Utilized symmetric cipher and the associated key length for guaranteeing confidentiality. AXP supports 3DES and AES.
- A *message authentication code* (MAC) algorithm both communicating parties support.
- Whether the established session is going to be anonymous or identifiable. Based on this parameter the identity management module of the authorization handler is either disabled or enabled.
- The security management system that is going to be used for verifying and validating the credentials supporting an access request.

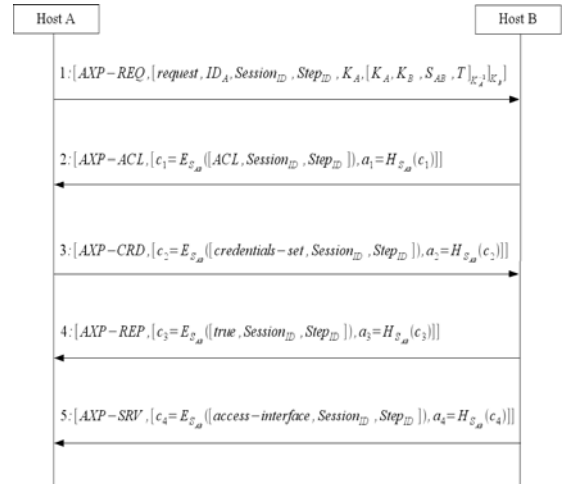
- The method of credentials distribution. As we mentioned earlier this can be either *push* or *pull*.
- Whether protection against denial-of-service attacks is enabled or not. If it is enabled the initiator of an AXP session is required to replay a stateless cookie sent by the responder in order to verify that it can receive datagrams at its claimed network address. This is implemented using the *cookie exchange* mechanism proposed by the Photuris protocol [10].

The session handler component is also responsible for managing reliability and replay protection. Reliability is implemented by using a retransmission mechanism that maintains a timer and keeps retransmitting a datagram until the expected reply is received. To detect replayed datagrams AXP uses the *replay window* mechanism specified as part of IPsec [11].

- *Encapsulation handler*. This component handles the encapsulation of AXP messages into a common encoding format. The encapsulated messages are authenticated using a MAC and encrypted using the symmetric cipher and the key material negotiated during session establishment by the *session handler*.

## 5. Case study

In this section we present an application example of AXP in order to demonstrate some related operation details. We assume that the session between the initiator and the responder has been established successfully and the parameter negotiation phase completed with both of them agreeing on the *push* method for credentials distribution, an identifiable session, the AES algorithm (with a 256 bits key length) for symmetric encryption, SHA-256 as the utilized MAC, no protection against denial-of-service attacks and the *ÆTHER* system for trust establishment. Based on these, a typical service access request session using AXP is illustrated in Figure 2.



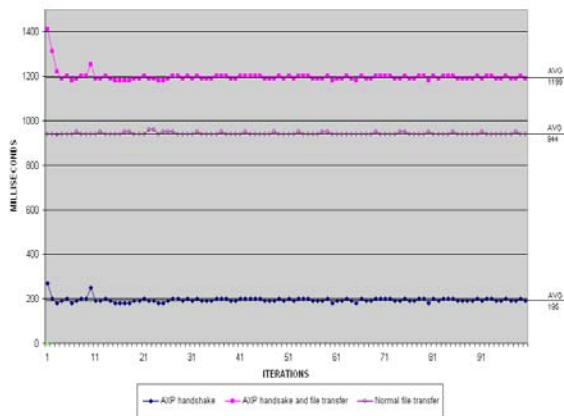
**Figure 2. An example AXP service access request session**

The initiator, host A, sends an access request to host B, the service provider. The notation we use is similar to [14]. The initial request message contains a packet identifier (AXP-REQ), a session identifier (Session<sub>ID</sub>), a protocol step identifier (Step<sub>ID</sub>), the service request and an identifier (ID<sub>A</sub>) of host A, i.e. its IP address. Furthermore, the public key of host A (K<sub>A</sub>), a digitally signed tuple containing the public keys of the two hosts, a generated symmetric key (S<sub>AB</sub>) and a timestamp (T) are also included. The entire message (except the packet identifier) is encrypted with the public key of host B (K<sub>B</sub>). Our key agreement protocol is based on [1]. Host B replies with the access control list (ACL) for the requested service (assuming that such a service is indeed provided), the same Session<sub>ID</sub> and an incremented Step<sub>ID</sub>. The message is encrypted using S<sub>AB</sub>, the negotiated session key. Furthermore, message authentication is performed by computing a MAC using c<sub>1</sub> (the ciphertext of the message) and S<sub>AB</sub>. Authentication and confidentiality is guaranteed for the remaining of the messages in this example using the same way<sup>2</sup>. Based on the received ACL, the initiator builds a reply with a set of credentials that can support its request. The service provider passes the set of credentials and the ACL to its *authorization handler* which invokes the previously agreed security management module (*ÆTHER*). The result is a boolean value that is transmitted to host A in step 4 (for the sake of the example we assume that the inference engine of *ÆTHER* reached a positive decision). At the end of step 4 we consider the AXP handshake to be over. In the final step host B transmits

<sup>2</sup> We use the *encrypt-then-authenticate* (ETA) mechanism of Krawczyk [12].

the access interface of the requested service. Please note that the exact resource descriptions for a service's access interface are outside the scope of AXP. Developed standards such as the Universal Remote Console (URC) [13] can be used for this purpose.

In order to investigate the overhead introduced by our protocol in both the handshake procedure and in network throughput we employed the above scenario and performed a simple file transfer of 1 MB between two communicating peers. Our test machines were i386 architecture machines running the Windows XP operating system and were interconnected by 100 Mbps ethernet. Specifically, host A that requested the file was an 1.8 GHz Pentium 4 and host B that provided the file was a 2.0 GHz Pentium 4. The experiment was performed with RSA keys of 1,024 bits size, with small public exponents ( $e$  was given the value 65,537) making the public key operations significantly faster than the private key operations. Moreover, we assumed that host B directly trusted the entity that certified the  $\text{\AE THER}$  attribute certificate of host A, thus requiring a single verification operation (for more details regarding the  $\text{\AE THER}$  trust establishment engine please see [3]). In order to have demonstrative results we have disabled the authorization cache of the *request handler*.



**Figure 3. File transfer timing measurements with and without AXP enabled**

The average time required for a full AXP handshake was 194.76 milliseconds, as it is illustrated in Figure 3. The whole transaction including the AXP handshake and the encrypted file transfer required 1199.12 milliseconds. In order to have a clear understanding of the overhead introduced by AXP in network throughput we run the same file transfer without any security mechanism. The average time taken was 943.82 milliseconds. Although the observed

overhead is significant (the difference is in the order of 27%), the use of the authorization cache in the *request handler* largely reduces the overhead of the handshake after the first reference to approximately 18%. These results do not prohibit the use of AXP in securing delay sensitive applications.

## 6. Conclusion

In this paper we have presented a new flexible protocol for negotiating and exchanging authorization information in networking environments. AXP supports several different security management systems and provides an easily extensible framework for adding new ones. Furthermore, it works over unreliable datagram transport protocols making it ideal for securing delay sensitive applications that have strict performance requirements. Our performance evaluation revealed that the overhead of AXP in network throughput is in the order of 27% without the caching of previous authorization decisions and in the order of 18% when caching is enabled. Another important advantage of AXP is that it is situated beneath the application level and above the network level, making its deployment transparent. The current prototype implementation of AXP is based on the C programming language and uses several open source cryptographic toolkits (OpenSSL [21], KeyNote [20] and SPKI/SDSI [18]) and is part of the ad hoc networking stack developed by the Networks and Telecommunications Research Group at the University of Dublin [15].

## 7. Acknowledgments

The first author is supported by the Irish Research Council for Science, Engineering and Technology (IRCSET), as part of the Embark Initiative, under contract number RS/2002/599-2. This material is based, in part, upon works supported by Science Foundation Ireland under grant number 03/CE3/I405.

## 8. References

- [1] M. Abadi, "Private Authentication", In *Proc. 2002 Workshop on Privacy Enhancing Technologies*, LNCS 2482, pp 27-40, 2003.
- [2] F. Andreasen, and B. Foster, "Media Gateway Control Protocol (MGCP)", Internet Engineering Task Force RFC 3435, 2003.
- [3] P.G. Argyroudis, and D. O'Mahony, " $\text{\AE THER}$ : an Authorization Management Architecture for Ubiquitous Computing", In *Proc. 2004 European PKI Workshop*, LNCS 3093, Springer-Verlag, pp 246-259, 2004.

- [4] M. Blaze, J. Feigenbaum, and A.D. Keromytis, "The KeyNote Trust Management System Version 2", Internet Engineering Task Force RFC 2704, 1999.
- [5] M. Blaze, J. Ioannidis, and A.D. Keromytis, "Compliance Checking and IPsec Policy Management", `draft-blaze-ipsec-trustmgt-00.txt`, 2000.
- [6] D. Clarke, J.-E. Elien, C. Ellison, M. Fredette, A. Morcos, and R.L. Rivest, "Certificate Chain Discovery in SPKI/SDSI", *Journal of Computer Security*, vol. 4, no. 9, pp 285-322, 2001.
- [7] T. Dierks, and C. Allen, "The TLS Protocol, Version 1.0", Internet Engineering Task Force RFC 2246, 1999.
- [8] C. Ellison, B. Frantz, B. Lampson, R.L. Rivest, B. Thomas, and T. Ylonen, "SPKI Certificate Theory", Internet Engineering Task Force RFC 2693, 1999.
- [9] S. Farrell, and R. Housley, "An Internet Attribute Certificate Profile for Authorization", Internet Engineering Task Force RFC 3281, 2002.
- [10] P. Karn, and W. Simpson, "Photuris: Session-key Management Protocol", Internet Engineering Task Force RFC 2522, 1999.
- [11] S. Kent, and R. Atkinson, "Security Architecture for the Internet Protocol", Internet Engineering Task Force RFC 2401, 1998.
- [12] H. Krawczyk, "The Order of Encryption and Authentication for Protecting Communications (or: How Secure is SSL?)", In Kilian, J., editor, *Advances in Cryptology – CRYPTO 2001*, LNCS 2139, Springer-Verlag, 2001.
- [13] B. LaPlant, S. Trewin, G. Zimmermann, and G. Vanderheiden, "The Universal Remote Console: a Universal Access Bus for Pervasive Computing", *IEEE Pervasive Computing*, vol. 3, no. 1, pp 76-80, 2004.
- [14] A. Menezes, P.v. Oorschot, and S.A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1996.
- [15] D. O'Mahony, and L. Doyle, *Mobile Computing: Implementing Pervasive Information and Communication Technologies*, chapter: An Adaptable Node Architecture for Future Wireless Networks. Kluwer Publishing, 2001.
- [16] J. Postel, "User Datagram Protocol", Internet Engineering Task Force RFC 768, 1980.
- [17] E. Rescorla, *SSL and TLS – Designing and Building Secure Systems*, Addison-Wesley, 2000.
- [18] R.L. Rivest, and B. Lampson, "SDSI – a Simple Distributed Security Infrastructure", Available at <http://theory.lcs.mit.edu/cis/sdsi.html>, 1996.
- [19] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: a Transport Protocol for Real-time Applications", Internet Engineering Task Force RFC 3550, 2003.
- [20] The KeyNote Trust Management System, see <http://www.cis.upenn.edu/~keynote/>.
- [21] The OpenSSL Project, see <http://www.openssl.org/>.