Security Architecture for Active Nets

AN Security Working Group

November 13, 2001

Abstract

This document describes the security architecture components of an Active Network node. It is intended as a guide for developing secure active applications and as a reference point for discussion of provision of security features in active networks.

1 Introduction

Active networking is a powerful and flexible new networking paradigm. While this new technology presents opportunities for innovative networking services, it also presents significant security challenges. This document describes an architecture for security in Active Networks. The features and services useful in a security architecture focused on authorization are discussed. The discussion describes the major components of such an architecture and their relationship to common packet processing flow. To be general to as many existing active network projects as possible, an attempt has been made to avoid assumptions of any particular computational model.

An active network transmits active packets which contain active code or references to code. An active node extracts the code from the packet and executes the code on the node with the packet as input. The result of the execution can be a modification of the state of the node, a modification of the packet, or the transmission of one or more packets. Legacy packets may be forwarded as before or may themselves be the recipient of custom network services.

The DARPA Active Network community has defined an architecture for an active node [7] that depicts a node as comprising a NodeOS and one or more Execution Environments. The Execution Environments (EE's) provide a programming interface or virtual machine that can be programmed or controlled by the active packets.

The NodeOS interface document [3, 12] of the community describes the resource abstractions of an active node operating system as thread pools (computation), memory pools (memory), channels (communication), files (persistent storage) and domains.[1] Domains are the primary

[1] "Domains" were termed "flows" in earlier versions of [3]. The term was changed because of the possible confusion of semantics with the usual use of "flow" in networking.

abstraction for resource management and are allocated a set of channels on which messages are transmitted, a memory pool and a thread pool. Channels are the primary abstraction for communication of packets into and out of the node. Some channels are anchored in an EE, with packets arriving on an in-channel and transmitted on an out-channel. Packets are assigned to an in-channel on the basis of a packet filter key provided in the creation of the domain. Channels not anchored in an EE are cut-through channels and involve no processing of active code. A cut-through channel can be created by a domain running in an EE from an existing in-channel and out-channel and continues to draw on the resource limits (e.g., memory, bandwidth) of the creating domain.

A domain may create a subdomain at any time. The principal assigned to a domain is established at creation time and governs the activities of the domain thereafter. The node begins operation with one or more domains assigned to EE's. An EE is permitted to start subdomains with more explicit packet filters when and as it wishes. Therefore, the EE makes its own decision as to whether the active code in a packet will run as part of the EE's domain, with the EE's principal privileges and resources, or in its own separate domain, with the packet's principal privileges and separately allocated resources.

The active packet arrives, is assigned by the node to an in-channel of a domain (either anchored in an EE or a cut-through channel) and is processed. If the active packet processing occurs in an EE, then the active code executes with access to the EE services and resources. The EE interface might pass the NodeOS interface through to the active code or might replace or augment that interface with its own services. The processing results in active packets being transmitted on an out-channel. The transmitted packets might be forwarded input packets, modified input packets, or newly inserted packets. The processing of the packet might also result in persistent changes in the node or EE state.

Active packets can be transmitted through the Internet by using the Active Network Encapsulation Protocol (ANEP). ANEP provides its own header format which includes a Type ID field that identifies the EE to which the packet should be transferred. The ANEP header also provides for options defined as type-length-value fields. Options defined in the ANEP protocol include source, destination, integrity and end-end authentication.

2 Security requirements

From the description of the active network, one can see that there are many entities in an active network that have assets that they would want to protect. The end user at the source and destination, the active node itself, the execution environments and the active code or domain all have security concerns.

The end user retains the concerns in traditional networks about the authenticity, integrity, and confidentiality of the packet's payload data as it traverses the network. As the active code may create persistent state in the active nodes it traverses, the end user will have the same concerns about data created in the infrastructure as well as concerns over access to that data.

The active node's security concerns are likely to be concentrated on authorization of use of the node's services and resources, in order to maintain availability of use. It will, of course, also be concerned about the integrity and confidentiality of its own state. The EE has the same concerns for its own services, resources and state.

The active code (standing as proxy to the end user who launched the packet) has security concerns that are related to access to its services (e.g., access to the domain in which it is executing) and access to sharable persistent state it creates.

2.1 Attacks

Attacks in an active network will arise from the same actions as in traditional networks. The following list of attacks, taken from RFC 2828 [14]. Security services of integrity, authorization, and confidentiality are needed to counter these threats. Authorization policy enforcement requires that authentication be provided as well.

1. attacks resulting in unauthorized disclosure:
   {countered by authorization and confidentiality services}
   - exposure by deliberate exposure or scavenging
   - interception by theft or wiretapping
   - intrusion by trespass or penetration
2. attacks resulting in deception:
   {countered by integrity, authentication, and non-repudiation services}
   - masquerade by spoofing or malicious logic (e.g., Trojan horse)
   - falsification by substitution or insertion
   - repudiation by false denial of origin
3. attacks resulting in disruption (denial of service):
   {countered by authorization and integrity services}
   - incapacitation by malicious logic
   - corruption by tampering or malicious logic
   - obstruction by interference or overload
4. attacks resulting in usurpation:
   {countered by authorization, integrity and confidentiality services}
   - misappropriation by theft of service or data
   - misuse by tampering, malicious logic or violation of permissions

This security architecture concentrates on the expressions and enforcement of authorization policy to counter these attacks. Confidentiality of the active packet payload is under the control of the sender and the active code, given access to cryptographic operations in each node.

2.2 Assets and Threat Sources

There are four parties that have interests to protect that are operating at once in a node. The first is the node itself, the second is the execution environment (EE), the third is the sender of the packet that is being processed and the fourth is the active code that is executing in the node.

The node is interested in protecting itself: protecting its resources against unauthorized usage, protecting the availability of its services, protecting the integrity of the state that will allow it to continue to offer services, and protecting its state against unauthorized exposure. The node can see these threats arising from the execution environment (that may consume resources, or modify the node state, or view sensitive node state, etc.), from the sender of the packet (who may send so many packets as to consume excessive resources) and from the active code (that may consume resources, or modify the node state, or view sensitive node state, etc. while executing in the EE). Through enforcement of appropriate security policies, the node expects to protect itself against these threats.

The EE is interested in protecting itself in the same ways. It sees the threats arising from other EE's, from the senders of packets, and from the active code it hosts. Through enforcement of appropriate security policies, the EE expects to protect itself against these threats. It also sees the possibility of threat from the node on which it executes. Some current research (e.g., [5, 13, 8, 1]) suggest techniques by which the EE could protect iteself from the node on which it executes. Baring access to implementations of this research, the EE cannot expect to protect itself from the node. EE's also have no control over where they are placed in the network. Any agent that distributes EE's to nodes must ensure that the EE's interests will be protected by the node.

The sender of the active packet is interested in protecting the data being transmitted in the packet: ensuring the integrity and confidentiality of the data in the packet and ensuring other attributes of the packet not represented by the bits in the packet such as the latency through the network. The sender of the packet sees threats directly to the data in the packet from other active code in the node, from the execution environment and from the node itself. Through suitable cryptographic protections, the sender can protect the data from source to destination. If, however, the node must share in the cryptographic association (i.e., the node must know the key that provides the protection), then the sender of the packet cannot protect the packet from the node or from the EE. The only hope is to rely on the active code to avoid nodes and EE's that are untrustworthy.

If the sender desires to ensure attributes of the packet other than the data it carries (i.e., the latency), then it must rely on the active code that it places in the packet to protect those attributes.

The active code has the same interests to protect. Its chief interest is the packet of which it is a component. Depending on the features of the EE in which it executes, it may be able to create state that can be shared with other active code or consumed by other active code, or it may be able to provide services to other active code. If so, then the active code wishes to protect itself: to protect its shared data and packet payload from unauthorized exposure or modification, to protect its services from unauthorized use, and to protect its resources against unauthorized consumption. The active code sees these threats arising from packets, from other active code, from the EE and from the node. Through suitable security policies, which it trusts the EE and node to enforce, it can protect itself against packets and other active code. Baring implementations of the research mentioned before ([5, 13, 8, 1]), the active code cannot protect itself against the EE and node on which it is executing. It can only protect itself by ensuring that it does not forward itself to untrustworthy nodes or EE's.

The following table summarizes this discussion:

This entity	sees a threat from this entity			
	Packet	Active Code	EE	Node
Sender of Packet		yes	yes	yes
Active Code	yes	yes	yes	yes
EE	yes	yes	yes	yes
Node	yes	yes	yes	

This entity	can protect itself from this entity			
	Packet	Active Code	EE	Node
Sender of Packet	NA	yes	not really*	not really*
Active Code	yes	yes	not really*	not really*
EE	yes	yes	yes	not really*
Node	yes	yes	yes and no	NA

The "not really" answers are places where executing code must trust the platform on which it executes. There is ongoing research ([5, 13, 8, 1]), addressing the issue of protecting code from untrusted platforms. But implementations are not yet widely available.

The "no" part of the "yes and no" answer relates to EE domains which act as a front for principals identified in those packets that are not assigned to their own individual subdomain. In those cases, the NodeOS must trust the EE to enforce the NodeOS authorization policy.

3 Authorization Support

The security architecture here concentrates on enforcement of authorization policy. There are several components needed to perform such enforcement. First, there must be a language in which the authorization policy is expressed. Second, there must be a way of representing the principal being authorized. Finally, there must be mechanisms to assure the authenticity of the claimed principal identities and the integrity of the packet.

3.1 Authorization Policy

Expression of the authorization policies of the various entities will naturally require a policy language. The language must be globally understandable, so that active code can express its policy at any of the nodes through which it travels – for access to the packet data, to any created persistent state, and to any services provided by the active code.

Simple access control lists (ACL's) may be sufficient for many applications. For ACL's to be useful as an expression for the active code policies, the identification of principals must be globally understood and unique.

For more complex authorization policies, a richer language is needed. PolicyMaker and Keynote are two related efforts that offer sophisticated policy expression capabilities. The Seraphim active capabilities [11] are Java code and so have full Turing expressibility.

Local node policies regarding principals may be configured locally or stored in a local policy server and retrieved as needed. Policy may also be distributed in-line in the packet itself. But while credentials attest to the attributes of a principal and therefore can be created without knowledge of the receiver, creating the node and EE authorization policies require knowledge of the intended target. Therefore, it is unlikely that the authorization policies of all nodes through which a packet may travel would be known at the origination point. In-line policies are likely to be added to the packet within security domains, perhaps as a function of a boundary controller. That means that in-line policies cannot be included in cryptographic authentications computable only at the origin (e.g., digital signatures). The binding between the policy and the packet must come from a reference in the in-line policy statement to a credential that is itself cryptographically bound to the packet.

3.2 Authentication

Hop-hop protections are needed to ensure the communication and signalling between neighboring active nodes. In cases where not all nodes are active nodes, the concept of "neighboring" may need to be configured or computed. Whether directly connected or configured, the neighbor nodes must be known so that keys can be shared to protect the inter-node communication. The ABONE document [10] describes the use of hop-hop integrity and authentication protections.

Note that if active nodes are not directly connected, there is a possibility that an active node that is not the configured neighbor might see a packet. If there is a domain in that intermediate node that has a filter that matches the packet, the packet will be processed in that node. This has the chance of violating the hop-hop integrity protections. The effect is the same as when an outsider intercepts and modifies a packet. The intended recipient is likely to drop such a modified packet. Active nodes and domains should be aware of the effect of changes they make on packets not addressed to them.

Hop-hop protections are adequate for integrity and authentication protections on inter-node communications. If authorization is based only on the identity of the neighbor from which the packet was received, then this would be sufficient. If any other origination identity is the basis for authorization, then hop-hop protection is not sufficient, because any node in the network could produce a packet with a bogus origination identity. Using only hop-hop protections would require trusting all the active nodes in a network, which is much too broad a trust model. We require a stronger end-end authentication, provided by cryptographic protections.

The choice of end-end cryptographic techniques is a challenge in active networks. Cryptographic techniques can be dividied into two types: asymmetric (public/private key pairs) and symmetric. The trust relationships are considerably different between the two types.

Asymmetric techniques (i.e., digital signatures) require that only the originator be trusted. Only an entity holding the private key can verifiably claim the originator identity associated with the key. Asymmetric techniques also can provide non-repudiation. However, if the packet is changed in a node, as can happen in active networks, the digital signature will fail the next verification. Sharing the private key with the receiving node, in order that it might recompute the digital signature over the changed contents, would eliminate the benefits of the digital signature of having only one trusted entity.

Symmetric techniques (i.e., HMAC-MD5 or DES-MAC) could be used if a key associated with the origination identity could be installed at each node of the packet's path through the network. However, this broadens the trust model in that all nodes who know the key can produce a packet with that origination identity and must be trusted. Symmetric techniques can only provide non-repudiation in that the source of the packet must be one of the set of nodes that shares the key. The trust model depends strongly on the method of key distribution.

A symmetric key could be distributed by an active application that could install a symmetric key in various nodes that subsequent active packets would encounter. Some of the techniques for distributing a symmetric key are prohibitively expensive in terms of processing and latency. Also, the set of nodes who are distributed the key all must be trusted - they can all produce a packet that looks like it came from the originator. Taken together, these features motivate a distribution of the key to a specific path through the network that must be followed by the subsequent active packets. Any deviation from this path will broaden the trust model and will incur latency as the key is distributed to the off-path node. Distributing keys at the application level also requires that the application be security aware.

A symmetric key could also be distributed at the NodeOS level by signing the key at the originating point and encrypting the signed key hop by hop for the neighboring node. The trust model in this case includes all the nodes in the network who might have received this packet at some point. All those nodes know the symmetric key and can produce a packet that looks like it came from the originator. This key distribution technique allow the nodes to protect applications that are not security aware.

To support the narrowest trust model and provide non-repudiation, we divide the packet payload into two areas: an area containing the code and static data and an area that may vary. The digital signature covers the static area only, leaving the variable area unprotected. Symmetric techniques can cover either the static or variable areas or both.

3.3 Principals and Credentials

A *principal* in a security architecture is an entity that can make a request that is subject to authorization. Put another way, a principal is an authorizable entity in a system. Security in the active network relies not only on the identity of the principals but also on the attributes to which they are authorized. A secure identification of the principals and the principals' authorized attributes, embodied in *credentials*, is a necessary part of the security architecture.

Experience in designing secure systems has shown that attempting to describe a security policy in terms of each individual principal's authority to access each individual object is not scalable. Furthermore, such a policy description would be difficult to understand for those instituting the policy. Various ways have been proposed for aggregating principals and objects into sets with common attributes, where the attributes are used in making security decisions rather than the individual identities. Roles, labels, domains, groups, etc. are examples of attributes that have been used in various policy languages in order to make policy specification scalable and meaningful.

Several existing credential schemes would serve to represent security relevant attributes. Both X.509 certificates with v3 extensions and Attribute Certificates and also Keynote and SPKI certificates, for example, were created expressly for this purpose. They all have the advantage of requiring no previous relationship between the principal and the authorizer in order for the authorization decision to take place.

As a packet traverses a wide area active network, it will encounter many different security domains. The aggregate security attributes relevant to each domain will differ. Because the packet will be subject to authorization everywhere it executes, all the different security attributes must be represented. It would infeasible for one credential to serve to represent all the needed attributes. A wide area active network will require the use of many different credentials to represent the principal's attributes at different points in the network. The active packet structure must be capable of carrying a list of credentials.

The amount of packet space consumed by the credentials could be large, given the size of the credentials and the length of the list of credentials. Therefore, we expect that credentials will frequently be represented in the packet indirectly, by reference to a distributed secure credential infrastructure with globally understood and accessible locators (PKI, Kerberos, DNSSEC, a secured LDAP, etc.)

3.4 Packet Structure

The packet structure must support lists of credentials, a separation of static and variable packet payloads, and both asymmetric and symmetric cryptographic authentication. The packet structure is as follows:

PacketArea	*Definition*
ANEP header	Standard Header
Static Payload	EE Static Header and Code and Data
Varying Payload	EE Variable Header and Data
Security Fields	a list of tuples, each containing:
Credential	X509, Keyid, etc.
Coverage	list of Packet Areas
Authenticator	digital signature or MAC, covers Credential and Coverage areas
In-line policy	
Original ANEP Options	i.e., Source Identifier Destination Identifier, Integrity Checksum, or N/N Authentication
Hop Integrity	covers everything)

This structure is depicted in the figure *Packet Format.*

4 Security Components

We place responsibility for authentication of packets in the NodeOS, for three reasons. First, these functions are of common use to all EE's. Second, the NodeOS has resources of its own to protect and requires the functionality itself. Third, cut-through channels, in particular, would be difficult to protect if the authentication support were in an EE. A cut-through channel's usage is charged against the original domain. As the cut-through channel is specifically designed to avoid EE processing, neither the EE nor the domain's active code would have any opportunity to authenticate the packets consuming its resources.

4.1 Cryptography Subsystem

The cryptographic subsystem must provide an engine that computes integrity and authentication functions. The cryptographic subsystem must also provide for key management, to include generation, retrieval, exchange, agreement, etc. of keys as well as management of the key database.

4.2 Credential Subsystem

The credential subsystem includes both global storage in a distributed infrastructure and local storage. Global storage must be a distributed secure system capable of creation, storage, retrieval and/or dissemination, and revocation of credentials. Local storage provides storage, receipt and/or retrieval, validation and revocation of local copies of credentials. The credential subsystem also provides the verification of credentials, which may include retrieval of other credentials.

4.3 Policy Subsystem

The policy subsystem is a policy database manager and a policy engine. The policy manager must store and protect policy statements for retrieval in the enforcement process. It must be capable of adding, changing and revoking policy statements. The policy engine evaluates policy statements during the enforcement process.

4.4 Enforcement

The enforcement may not be separable into a component, but is certainly a vital part of the function of the security.

For strong assurance, any enforcement mechanism must meet the three classic attributes of a reference monitor:

non-bypassable: the enforcement mechanism must capture every request to access a protected resource or service,

tamper proof: the enforcement mechanism must be safe from modification that would prevent it from performing its function, and

analyzable: the enforcement mechanism must be analyzable. This is usually taken to mean that it must be small.

Not usually stated, but necessary to ensure that the enforcement mechanism is non-bypassable, is another attribute:

non-spoofable: the enforcement mechanism must have some immutable (or at least, unaccessible to user code) authentication/authorization state, called a security context, associated with every execution.

The security context, then, is a crucial part of the enforcement. There must be a component that is responsible for binding a security context to the execution of code so that the security context is available to the authorization enforcement but not to the executing code.

Enforcement includes the following steps:

1. intercept request for access to an object

2. extract security context

3. retrieve policy for the context, object, and request (possibly including parameters of the request and features of the environment like time)

4. evaluate policy

5. compose evaluation of separate policy components, if necessary

Various enforcement techniques have been designed and used over the years – wrappers, namespace thinning, callgates, pointer/capabilities, virtual memory via hardware, virtual machines – all achieve much the same goal, they just rearrange code placement and accessibility a little. They all have relative advantages and disadvantages. This architecture does not mandate what technique should be used.

5 Packet's Security Events

The following enumerates the events in the life of a packet and domain that have security relevance.

1. The Hop-Hop Key Identifier is pulled out of the packet and used the to retrieve appropriate key, algorithms, etc.

   The hop-hop integrity is checked. Failure means to drop the packet.

2. The packet is matched to a domain filter.

3. The list of credentials (or credential references) is extracted, retrieved and the credentials are verified (possibly a recursive process to verify a chain of credentials). If any credentials fails the verification process, the packet processing continues, using those credentials that succeed.

4. If the domain is set for authentication, the packet authenticators are verified. If any authenticators fail, the packet processing continues, using the credentials whose authenticators succeed.

   The NodeOS checks the authorization of the EE to be receiving such packets.

   The NodeOS checks the authorization of the domain to be receiving such packets.

   The NodeOS checks the packet's authenticated identity according to the authorization policy of the domain.

5. The packet, minus the ANEP header and hop-hop integrity fields is passed to the appropriate domain.

6. The code is extracted or retrieved (for a stream of related packets, this may only happen on the first packet). The code could be inline in the packet, in the code cache already, or stored off-node. The retrieval process, which is probably EE specific, will have its own security protections for authenticity, integrity, etc. During this process,

security attributes would be associated with the code, like author, software house, UL seal, etc. An instance of the retrieved code ("Active Application") is prepared for execution.

7. Any authenticated credentials are bound to the domain's execution in a security context.

8. The domain begins execution of or with the packet.

9. Attempted access from a domain to other AA's or shared state areas, or to resouurces or services provided by the EE are mediated by the EE (based on EE policy, NodeOS policy and the credentials associated with the domain).

   Attempted access from the AA domain to other AA's may also be mediated by the other AA's (based on the AA policy and the credentials associated with the domain).

   Attempted access from the EE domain to the NodeOS (carrying identification of the AA that is causing the access, if any) is mediated by the NodeOS (based on NodeOS policy, the credentials associated with the EE and the provided credentials).

10. Packets sent by a domain are cryptographically protected by the NodeOS.

11. Packets transmitted by the NodeOS have hop-hop integrity applied.

At some point in the processing of a packet, the domain may decide to create a subdomain. When a new domain is created, the create call must specify whether subsequent packets should be authenticated and if so what packets are authorized to be processed in this domain. This means that the parent domain may not have required that packets be authenticated. If the sub-domain is to be authenticated, then credentials in the packet being handed to the new sub-domain that had not previously been verified as authentic must be verified.

Figure *Packet Processing* depicts these security events.

6 Editor Contact Information

Comments can be sent to the mailing list ActiveNets_Security@ittc.ukans.edu.

The editor can be reached at:

Sandra Murphy
NAI Labs at Network Associates, Inc.
3060 Washington Road
Glenwood, MD 21738
email: sandy@tislabs.com
email: Sandra_Murphy@nai.com
phone: 443-259-2303
fax: 301-854-4731

7 Acknowledgements

Many people have enjoined the editor in discussions that contributed to this document, including the SANP project team at NAI, Bob Braden and the ISI team at ISI, Steve Schwab and the Exokernel project team at NAI, Jay Lepreau and the project team at Utah, Dan Sterne and the ANIDR project team at NAI, and Roy Campbell and the Seraphim project team at UIUC.

References

[1] Martin Abadi, Joan Feigenbaum, and Joe Kilian. On hiding information from an oracle. *Journal of Computer and Systems Science*, 39(1):21–50, 1989.

[2] D. Alexander et al, "Active Network Encapsulation Protocol, Draft," July 1997. available online at http://www.cis.upenn.edu/switchware/ANEP.

[3] AN Node OS Working Group, "NodeOS interface specification," January 2000, available online at http://www.cs.princeton.edu/nsg/papers/nodeos.ps.

[4] AN Composable Services Working Group, Ellen Zegura, editor, "Composable Services for Active Networks," May 1998, available online at http://www.cc.gatech.edu/projects/canes/papers/cs-draft0-3.ps.gz.

[5] Lee Badger. Self-protecting mobile agents, 2001. see the web site at http://www.pgp.com/research/nailabs/secure-execution/self-protecting.asp.

[6] Lee Badger et al. "Domain and Type Enforcement UNIX Prototype," in *USENIX Security Symposium*, 1995.

[7] K.L. Calvert, "Architectural framework for active networks, version 1.0," University of Kentucky, July 1999, available online at http://www.cc.gatech.edu/projects/canes/papers /arch-1-0.ps.gz.

[8] Christian S. Collberg and Clark Thomborson. Watermarking, tamper-proofing, and obfuscation - tools for software protection. Technical Report 2000-03, University of Arizona, 2000.

[9] "CORBAservices: Common Object Services Specification. Chapter 15: Security Service Revision 1.2."

[10] Bob Lindell, draft-nodeos-security-00.txt, "Active Networks Protocol Specification for Hop-By-Hop Message Authentication and Integrity", available online at http://www.isi.edu/abone/DOCUMENTS/OSsec.txt

[11] Zhaoyu Liu, Roy H. Campbell, and M. Dennis Mickunas, "Securing the node of an active network," in *Active Middleware Services*, Salim Hariri, Craig Lee, and Cauligi Raghavendra, Eds. Kluwer Academic Publishers, Boston, MA, September 2000.

[12] Larry Peterson, Yitzchak Gottlieb, Mike Hibler, Patrick Tullmann, Jay Lepreau, Stephen Schwab, Hrishikesh Dandekar, Andrew Purtell, and John Hartman, "An OS interface for active routers," *IEEE Journal on Selected Areas of Communications*, to appear 2001.

[13] Tomas Sander and Christian F. Tschudin. Protecting Mobile Agents Against Malicious Hosts. In Giovanni Vigna, editor, *Mobile Agent Security*, pages 44–60. Springer-Verlag: Heidelberg, Germany, 1998.

[14] R. Shirey, Internet Security Glossary, Technical Report RFC 2828, GTE/BBN Technologies, May 2000.

A Appendix A: Packet Format

The packet structure explained above is embodied in the following packet format.

A.1 Base ANEP Definition

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Version     |     Flags     |             Type ID           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      ANEP Header Length       |      ANEP Packet Length       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
~                            Options                            ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               .                               |
~                            Payload                            ~
|                               .                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

For us, the Payload will be 0 length. ANEP Header Length * 4 is the same as ANEP Packet Length. (32bit words to bytes conversion.)

The first 64 bits are the "ANEP header."

Option template...

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|FLG|       Option Type         |         Option Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               .                               |
~                  Option Payload (Option Value)                ~
|                               .                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

A.2 Previously Defined Options

These option types are taken from the ANEP specification.

```
Source Identifier              1
Destination Identifier         2
Integrity Checksum             3
N/N Authentication             4
```

Source Identifier, Destination Identifer, Integrity Checksum
In the ANEP specification, these are all defined as:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Scheme Identifier                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               .                               |
~                        Option Payload                         ~
|                               .                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

For the Source Identifier option, there are three Scheme Identifiers defined:

```
IPv4 address (32 bits) 1
IPv6 address (128 bits) 2
802.3 address (48 bits) 3
```

Non-Negotiated Authentication

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Scheme Identifier                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               .                               |
~                         Option Payload                        ~
|                               .                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

```
 The following values have been reserved for the Scheme Identifier:

            SPKI  Self-signed Certificate     1
            X.509 Self-signed Certificate     2
```

A.3 New Option Definitions

Option Type Value	Name
5	Hop Integrity
40	Varying NodeOS data (reserved)
48	Static Payload
36	Varying Payload
100	List of Credential-Authenticator Tuples
62	In-line Policy

A.4 Hop Integrity

This holds a keyed hash which covers nearly all of the packet as transmitted - the keyed hash itself is the only item not covered. This should be the absolutely last option prior to the payload field. The identifier is an index into a table that exists fairly persistently compared to the lifetime of individual packets or even a long-lived stream. This assumes that each node knows all of its neighbors and establishes security associations with each.

The following is taken from the ABONE Hop-By-Hop specification [10].

Key Identifier: An unsigned 64-bit number that MUST be unique for a given sender. Locally unique Key Identifiers can be generated using some combination of the address (IP or MAC) of the sending interface and the key number. The combination of the Key Identifier and the sending system's network address uniquely identifies the security association.

Sequence Number: An unsigned 64-bit monotonically increasing, unique sequence number.

Sequence Number values may be any monotonically increasing sequence that provides the INTEGRITY option with a tag that is unique for the associated key's lifetime.

Keyed Message Digest: The digest MUST be a multiple of 4 octets long. For HMAC-MD5, it will be 16 bytes long.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Key Identifier                         |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
```

```
|                        Keyed Message Digest                   |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

A.5 Credential-Authenticator Tuples

This option consists of an list of credential-authenticator (Cred-Auth) tuples, one for every principal associated with the packet.

```
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|  ANEP type = Cred-Auth        |     Cred-Auth Length          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                       Cred-Auth Tuple                         ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                       Cred-Auth Tuple                         ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
z.....z.....z.....z.....z.....z.....z.....z.....z.....z.....z...z
```

Each Cred-Auth Tuple ends in a 32-bit boundary. If necessary, dummy bytes are padded at the end. A Cred-Auth Tuple has the following format:

```
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
| Credential Type               |     Tuple-Length              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Identifier Length             |                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               |
|                                                               |
~                         Identifier                            ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Credential Length             |                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               |
|                                                               |
~                         Credential (Opaque)                   ~
|                                                               |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

```
| Auth Coverage Length         |                                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               +
|                                                               |
~      List of option types covered (16 bits each)              ~
|                                                               |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|            Auth Type         |   Auth Data Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                          Auth Data                            ~
|                                                               |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

If the credential-type is not recognized, then we can directly skip "Tuple Length" bytes for the next credential.

You're out of credential-auth tuples when you're out of Cred-Auth Length bytes.

We divide the packet format for this option type into three sections:

A.5.1 Credential

The credential has the components Credential Type, Identifier Length, Identifier, Credential Length, Credential. We presently have three credential types identified:

Credential Type = X509

This credential is intended to carry a reference to an X.509 credential. The credential is identified by issuer distinguished name and serial number. Locator hints are given as to where this credential might reside and the retrieval method.

Credential Type: A type of "1" indicates X.509 credentials.

Tuple Length: The number of bytes in the following fields

Identifier Length: The Identifier Length covers the Serial number and the (X509) Distinguished Name fields.

DN-Length: The length of the issuer distinguished name.

Distinguished Name: The certificate issuer's distinguished name

Serial Number: The issuer's serial number for the certificate, a variable length field. It begins after DN-Length bytes.

Retrieval Method: Protocol to be used to retrieve the credential

Data Length: Length of retrieval location name

Retrieval Lookup Data - location from which credential can be retrieved

The list of locator hints is optional and variable in length (0 meaning that the packet is relying on the node to find the credential bits).

The only values currently defined for Retrieval Method, Data Length, and Retrieval Lookup Data are DNS, FQDN-length, FQDN (where FQDN means fully qualified domain names).

You know that you are out of locator hints when you are out of Credential Length bytes.

The format would look like this:

```
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
| Credential Type = X509        |     [Tuple Length]            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Identifier Length             |      DN-Length                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                      Distinguished Name                       ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                         Serial Number                         ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Credential Length             |  Retrieval Method             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Data Length         |                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               |
|                                                               |
~                 Retrieval Lookup Data                         ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
z................................z.............................z
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Retrieval Method             |        Data Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                 Retrieval Lookup Data                         ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Credential Type = Just KeyID

This type is used to refer to the identifier of a pre-distributed key.

```
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
| Credential Type = JustKeyID   |      [Tuple Length]           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Identifier Length             |                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               |
~                          Identifier(KeyID)                    ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Credential Length = 0         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

A Credential type of “2” indicates JustKeyID.

The Identifier Length gives us the length of the predistributed KeyID. Since this is simply a referencing mechanism for a particular key, the Credential data itself is empty (Credential Length = 0)

Credential Type = KEY_TRANSPORT

This type is used when transporting a session key inside the packet.

Credential Type: A credential type of “3” indicates Key Transport.

Symmetric Key ID: The symmetric key Id should be a globally unique identifier, and SHOULD be 128 bits. Identifier Length defines the size of this symmetric Key Id.

Credential Length: The number of bytes in the next three fields

Credential Sub-type: The type of the credential used to authenticate this key during transport. If the Credential Sub-type is not recognized, then we skip the entire Cred-Auth tuple using the tuple length.

Encrypted Key Id Length, Encrypted Key Id: This names the key encrypting key (KEK) i.e. the key used to protect the session key being transported inline.

Encryption Crypto Algo, Encrypted key Length, Encrypted Symmetric key: This defines the encrypted session key itself. The Encrypted Symmetric key field contains the result of encrypting the session key using the KEK.

The only currently defined value for the Encryption Crypto Algo is DES.

Crypto Data: The Crypto Data is used to verify the authenticity of the symmetric key being transported. It is computed as a signature over (keyid, symmetric key, static-area of packet, cred) using the cred where "cred" is the opaque portion in the above packet format inclusive of the credential subtype field.

```
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
| Credential Type=KEY_TRANSPORT |     [Tuple Length]            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Identifier Length             |                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               |
|                                                               |
~                       Symmetric Key ID  [KId]                 ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Credential Length             | Credential SubType            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Sub-credential length         |                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  Sub-credential(Opaque)       ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Encryption Key ID Length      |                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               |
|                                                               |
|                    Key Encryption Key ID                      ~
~                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Encryption Crypto Algo     |  Encrypted Key Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               ~
~                 Encrypted Symmetric Key                       |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Crypto Algo        |            Crypto Length      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                      Crypto Data                         ~    
|                                                               |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

Presently, we have one credential sub-type defined:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

```
| Credential SubType [X509]     |   Identifier Length           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Distinguished Name Length    |                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               +
|                                                               |
~                       Distinguished Name                      ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                          Serial Number                        ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Hints Length              |    Method  [DNS]              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Data Length                   |                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               +
|                                                               |
~                           Lookup Data                         ~
|                                                               |
z...............................z...............................z
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Method             |        Data Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                           Lookup Data                         ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The fields here are the same as in the X.509 Credential type.

Note: (forward reference to Authenticator section) For Credential Type = KEY_TRANSPORT, the Auth Type MUST be some MAC using the symmetric key being transported.

Other restrictions such as Auth Type MUST be a digital signature for Credential-Type = X509 apply as before.

A.6 Coverage

This section identifies the different options within the packet covered by the authenticator-section.

Currently defined options are:

Option Type Value	Name
1	Source Identifier
2	Destination Identifier
3	Integrity Checksum
4	N/N Authentication
40	Varying NodeOS data (reserved)
48	Static Payload
36	Varying Payload
62	In-line Policy
100	List of Credential-Authenticator Tuples

A.6.1 Authenticator

The authenticator portion includes the Auth Type, Auth Data Length, and Auth Data fields.

Auth Type: The type of the authenticator, tied to the type of the credential component of the tuple.

Auth Data Length: The length of the following authentication data.

Auth Data: The cryptographic protection bits themselves.

For X.509 credentials, the authenticator must be a digital signature, where DSA is the only currently defined algorithm. For JustKeyID and KeyTransport types, the authenticator can be HMAC-MD5, HMAC-SHA1 or DES-MAC. The values of the Auth Type are:

Auth Type Value	Name
1	DSA authentication
2	HMAC-MD5 authentication
3	HMAC-MD5 authentication
4	DESMAC authentication

A.6.2 In-line Policy

These fields are used to carry policy data. Each policy data must refer to one of the credential fields to bind the policy to this packet.

Policy Type: The type of policy carried

Policy Length: The length of the data

Policy Data: The policy information itself. It is expected that the policy data will itself carry an authenticator of the policy data originator.

You know you are out of in-line policy data when you are out of option bytes.

The format by which the policy data refer to a credential and carries an authenticator is type specific.

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Policy Type               |    Policy Data Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               .                               |
~                   Policy Data                                 ~
|                               .                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

At present, there are two policy data types defined:

Active Capabilities:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 1                    |           Data Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               .                               |
~                   Active Capability                           ~
|                               .                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Attribute Certificates: The X.509 specification details the format of the reference to one of the credential fields and the authenticator (a signature by the Attribute Certificate authority).

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type = 509                |           Data Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               .                               |
~                   X.509 Attribute Certificate                 ~
|                               .                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

A.7 Static Payload and Varying Payload

These have no generic definition. Separate documents will define the formats for the prototype protocol.

A.8 Order Of The Options

This is significant for signing only.

Only options present in packet are considered in the ordering, there are no null-valued options. Also, an option is assumed to exist only once per packet - duplicate option values are formatting errors.

```
ANEP header (not covered by Credential Authenticators)
Credential Fields
Static Payload (EE header and Code and  Data)
Credential Authenticators
Varying Resource Management (placeholder)
Varying Payload (EE header and Data)
In-line Policy
Old Option (Source Identifier)
Old Option (Destination Identifier)
Old Option (Integrity Checksum)
Old Option (N/N Authentication)
Hop Integrity
```

B Appendix B: Basic Concepts

B.1 Credentials

A principal in a security architecture is an entity that can make a request that is subject to authorization. Put another way, a principal is an authorizable entity in a system. Security in the active network relies not only on the identity of the principals but also on the activities to which they are authorized. A secure identification of the principals and their authorized activities, embodied in *credentials*, is a necessary part of the security architecture.

Experience in designing secure systems has shown that attempting to describe the security policy in terms of each individual principal's authority to access each individual object is not scalable. Furthermore, such a policy description would not be understandable for those instituting the policy. Various ways have been proposed for aggregating principals and objects into sets with common attributes, where the attributes are used in making security decisions rather than the individual identities. For example, the following aggregations have been used in different secure systems:

roles

: In role based access control, each principal is assigned a role and it is the role that is used to decide the authorization to accomplish actions or access.

groups

: A principal may belong to one or more groups, each of which are entitled to some particular activities. Principals can be assigned to groups or may request to join groups and leave groups dynamically.

clearance label

: In the military security model, both principals and objects are labelled with a non-forgeable clearance level. The clearance levels are ordered and the ordering decides the authorization to access objects.

domain

: This is a frequently used term with different semantics. For example, in a domain and type enforcement system, each principal has an associated domain which is used in determining the authorization to access objects. In CORBA Sec [9], the domain is a set of objects that are all subject to the same security policy.

types

: In domain and type enforcement, objects are assigned types. The domain of the requester and the type of the object are used in making the authorization decision.

The credentials combine a description of the identity of the principal and the aggregation attributes associated with the principal. In networks with a single administrative domain and a common security policy, the attributes may be well known and can be represented by appending to the principal identity. For example, domain and type enforcement can work in such a distributed environment by inserting domain and type labels in the packets. In more heterogeneous environments, there may be authorities who will bind the identity and attributes and produce a certificate asserting the validity of that binding. Usually, a certificate includes a key that represents the principal in any cryptographic exchange and a cryptographic binding of the key to the identity and aggregation attributes.

The credentials that will be used to authorize the requests of a packet or its active code must be bound in and to the packet. In the active network world, the credentials (identity and attributes) can be bound to the packet simply by inclusion in the packet. This would be an appropriate choice in a homogeneous world with a common security policy, little risk of attack, and node OS's that can be trusted to correctly include the appropriate credentials. In a more heterogeneous environment, the credentials would be best cryptographically bound to the packet. To eliminate cut-and-paste attacks, the key in the credential certificate should be used to bind the credentials and packets.

We believe that X.509v3 and SPKI certificates could be used as credentials in heterogeneous active networks, with a key used as the primary identification of a principal.

B.2 Least Privilege

The principle of least privilege dictates that only the minimally powerful authority possessed by a principal should be used to authorize the principal's request for access. This means that a mistake made by someone with authority will cause the least possible damage. If a principal has credentials granting it authority for many different accesses, then it should be possible for the principal to indicate which of those authorities are to be used in deciding a request. This could by implemented by having the principal deliver the appropriate authority along with the access request or by having the principal request a restriction or expansion of its effective authority before the access request.

B.3 Delegation

Sequential composition of services [4] in an active node brings with it the issue of delegation of authority. When one principal invokes the service of another in an active network, it can be ambiguous which principal's authority should be used to decide subsequent accesses. In some systems, the callee's authority is always used in deciding subsequent accesses. In other systems, the callee adopts the caller's authority. In still others, the intersection of both principals' authorities is used in deciding the access. And finally, some systems allow flexibility, so that either the caller may voluntarily and dynamically restrict the authority to the intersection of both authorities (as in the TIS DTE work [6]) or the caller may voluntarily and dynamically expand the authority from the intersection to just its own authority (as in the Java 1.2 model).

Each method of delegation of authority has good and bad points. If a principal A is calling a new network service associated with a principal B, for example when the authorization to be decided is resource usage with limits in force, then:

none If no authority is delegated, then B cannot misuse A's authority. On the other hand, if B is a less powerful principal than A, B may not be able to get access to the resources of A needed to perform the service. This may necessitate that a service be associated with a principal possessing far reaching authorities in order to access the needed resources of all callers, e.g., a printer process needing access to all files. Finally, any resources consumed in performing the service for A (e.g., CPU time) will come from the allocated resources of B, which may not always be fair.

adopt On the positive side, adoption of the caller's authority follows the least privilege rule when a powerful B is performing service for a less powerful A. However, adopting the caller's authority would allow a malicious B to employ that authority to ill effect. For example, the resources of A could be extravagantly consumed by whatever B decides to do during performance of this service. This is just the opposite of the previous case of no delegation of authority. Principal A must trust B to make judicious use of A's resources.

intersect When the intersection of A's and B's authorities are used, then both A and B are following least privilege. Unfortunately, B may have some private resources it requires to perform the service. Either A must be given access to these private resources or the intersection of authorities would prevent B from accessing those private resources while performing service for A.

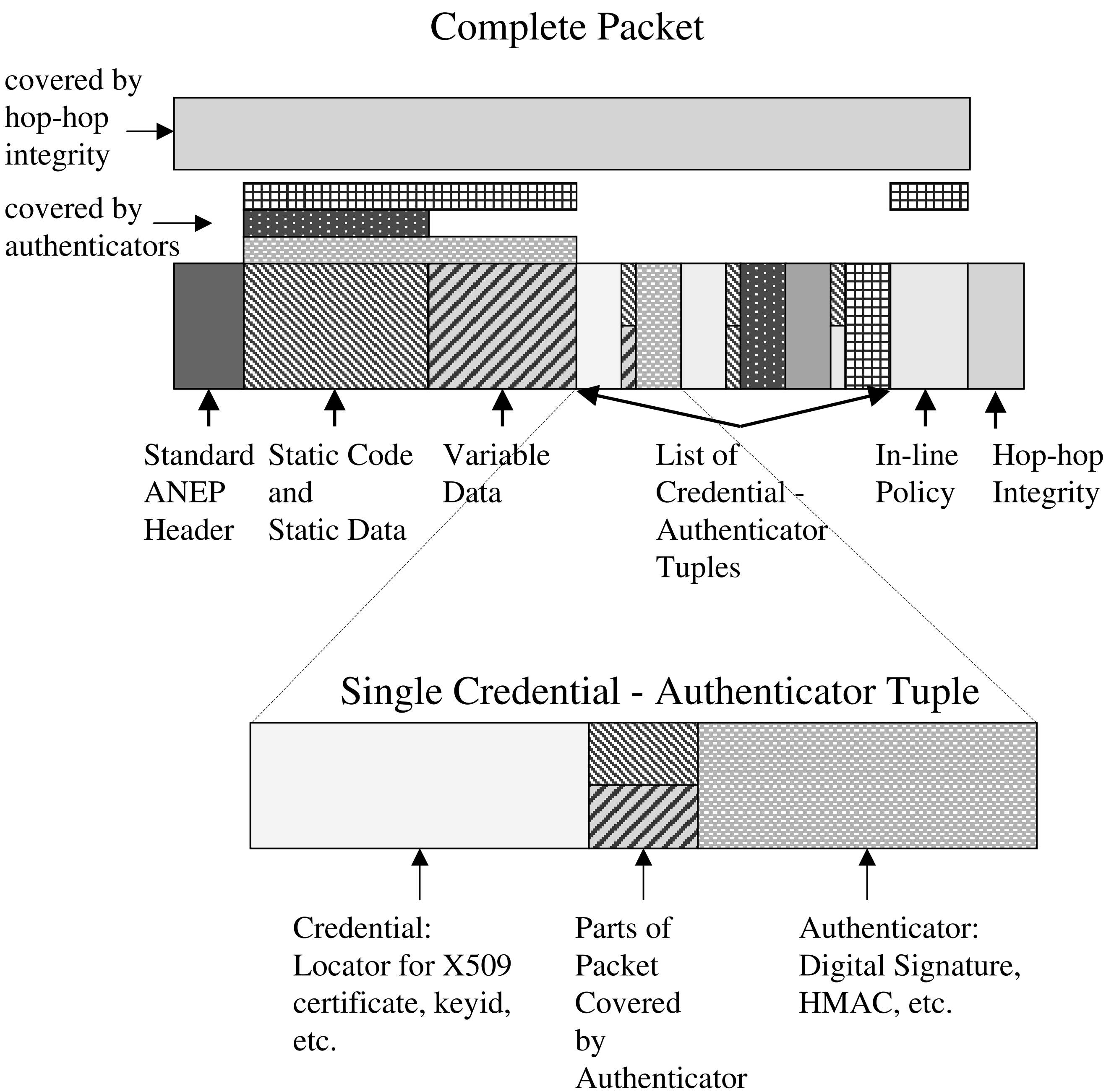
voluntary intersection If B is allowed to voluntarily expand the joint authority from the intersection of A's and B's authorities to just its own authority, then B has to be careful what it does on behalf of A when the authority is expanded. B could serve as a covert channel for A. B would be liable to resource consumption during the time that its authority alone was governing the access decision. If B is allowed to voluntarily restrict the joint authority from just its authority to the intersection, then it cannot provide A with a covert channel.

A flexible system that allowed a choice of delegation models would be ideal. The initial active networks should choose to adopt the caller identity, so that the authorities of the originating principal of a packet would govern access decisions and resource consumption throughout the packet processing.

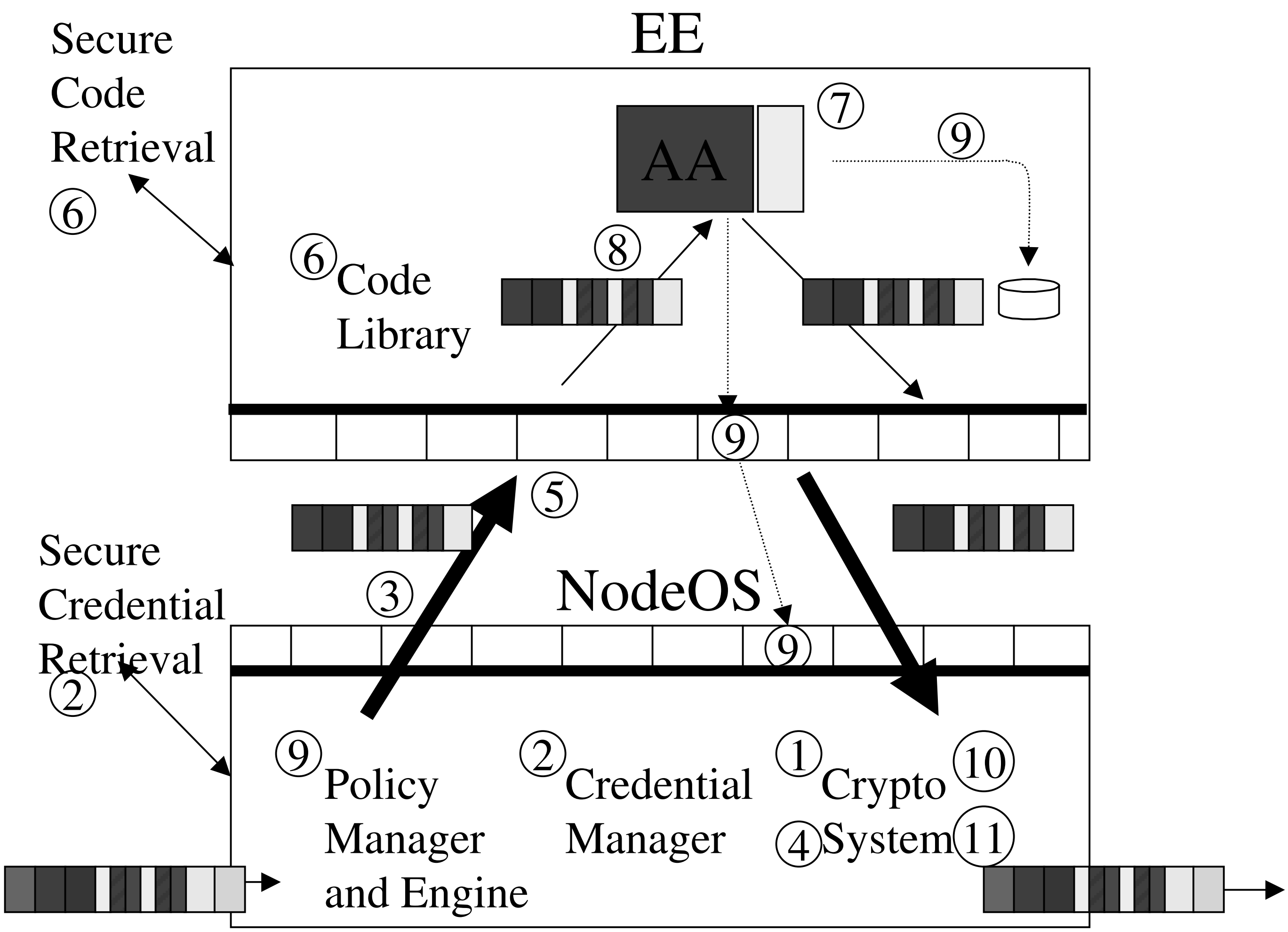
B.4 Revocation

Another difficult problem in active networks will be revocation. The security community is familiar with the problem of revocation of privileges. Active networks will have not only the problem of revocation of privileges but of revocation of active code. When a reason is discovered to terminate an active code that has been distributed throughout the system, it must be possible to revoke the active code wherever it is found. Although the need to prevent denial of service attacks means that each active network architecture includes some provision for limiting resource consumption, it is still necessary to provide for revocation so that an active code can be terminated immediately even if it is still richly endowed with resources. There may be need for active nodes to be required to respond to an authorized termination request, so that applications as well as networks can be protected from deviant active code.

Packet Format



Packet Processing



1. Check hop-by-hop
2. Retrieve credentials
3. Assign to domain
4. Check authentication
5. Deliver to EE:
   packet minus header
   and integrity
6. Retrieve AA code
7. Bind identity to AA
8. Execute AA on packet
9. Mediate accesses
10. Add authentication +
    credentials
11. Add hop-by-hop