COM SYS Communication & Distributed Systems

RWTH AACHEN UNIVERSITY

Service Negotiation and Selective Disclosure of Endpoint Information for on-path Middleboxes

Master Thesis

Anupam Ashish

RWTH Aachen University, Germany

Chair of Communication and Distributed Systems

Advisors:

Dipl.-Inform.	René Hummen
Prof. Dr.-Ing.	Klaus Wehrle
Prof. Dr.-Ing.	Ulrike Meyer

Registration date: 2012-01-10
Submission date: 2012-07-24

I hereby affirm that I composed this work independently and used no other than the specified sources and tools and that I marked all quotes as such.

Hiermit versichere ich, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Aachen, den 24. Juli 2012

Abstract

The Internet infrastructure is no longer end-to-end and is greatly dependent on the functionalities of on-path devices. These network devices called middleboxes have enriched the network functionality in many ways (e.g. NATS, Quality of Service). The availability of high-level information about the application, user or the host greatly improves the functionality of these middleboxes. Hence, approaches like SEAMS and NUTSS have provided mechanisms for the end-point to signal these high level end-point information to the on-path middleboxes [19, 13].

These end-points depend heavily on the security protocols to protect their sensitive information. Securing the communication end-to-end makes the information on the signaling channel opaque to the network devices. Hence, the middleboxes have to fallback on coarse identifiers from IP address and DNS namespaces or rely on expensive deep packet inspection mechanisms to extract end-point information from the signaling channel. These have mechanisms have disadvantages such as unreliability (e.g. blocking of the connection request from a legitimate application). Moreover, the reliance on cryptographic identifiers for these security protocols exposes the end-points to identify privacy problems. These cryptographic identities when transmitted insecurely can be tracked and all the transactions where these identities have been used can be linked to the identity of the certificate holder. Blinding a popular solution to deal with the identity privacy problem of public key cryptography renders the upper layer identifiers, on which the middlebox policies are based, useless.

We propose a solution where the middlebox can negotiate for the end-point information required. Our system allows the end-points and the middleboxes to mutually establish trust with these network entities and decide on the information request for end-point information based on the mutual trust level. It also provides mechanisms to the end-points to secure the information transmitted in the signaling channel either by encryption or selective disclosure. Finally, we also discuss the integration of blinding with our system which helps in mitigating the identity privacy problems of public key cryptography.

Acknowledgments

I would like to thank all the people who have directly or indirectly supported me during this thesis.

First of all I would like to thank Prof. Klaus Wehrle and Prof. Ulrike Meyer who provided me with an opportunity to work on this thesis.

I would like to mention special thanks to my advisor René Hummen for supervising my thesis. Thank you René for constant motivation during the long working hours of my thesis and valuable and informative discussions at the start of my thesis.

Finally, I would like to thank my family for their constant support and the love they have bestowed upon me.

Contents

1

Introduction

The Internet, since its inception, has witnessed two major developments. Firstly, the middleboxes have been playing a greater role in enriching network functionality (e.g. NATs, Firewalls). Secondly, the concern for security and privacy due to the presence of untrustworthy network entities, has led to the development of numerous end-to-end security protocols (e.g. SSL/TLS, SSH etc.). These security protocols omnipresent in Internet, are based on Public Key Infrastructure (PKI) which entails the usage of cryptographic identities like digital certificates (e.g. X.509 certificate) and public keys. The digital certificates enable the entities involved to cryptographically verify the other side and encryption helps in protecting the sensitive information being shared on the communication channel. Furthermore digital signatures help in verifying the integrity of data. As a result, popular security protocols like TLS, IPSec have proven to be sufficient in protecting the end-to-end communication. But with these security protocols in play, there is a gap in the information available in the communication channel (encrypted end-to-end) and the information required by the middleboxes. This leaves the middleboxes in a tight position, where securing the payload prevents the middleboxes from gauging the information that is necessary for their functionality. Furthermore, the perpetual use of cryptographic identities have concerns of their own, particularly when these identities are shared insecurely. Their dependence on X.500 global database design exposes more than necessary information about the issuer [11] and the holder of the certificate. For example, "Distinguished Name" (an information field in X.509 certificate) is a hierarchical and unique name (e.g. informatik.rwth-aachen.de). An immediate repercussion of it is that the identity pointed by the Distinguished Name (DN) can be associated to all the transactions where the same public key and certificate have been used [25]. In order to circumvent these privacy concerns of Public Key Cryptography, techniques like RSA Blinding for TLS [6], SPSL [36] and BLIND for HIP have been developed [40].

Middleboxes have been extending network functionalities in numerous ways [5]. In some networks, they ensure Quality of Service (QoS) while in others they act as proxy servers or transcode multimedia streams on the fly etc. However, an effective

functioning of these middleboxes requires more information from the end-points. These middleboxes hence, depend on the coarse information available from the IP addresses and packet headers (like application layer protocols, HTTP). This information might not be sufficient for all the cases so, they have to look inside the packets to extract specific information (e.g. application information in case of a QoS box). Availability of information about the user, the application or the host in the signaling path can make the lives of middleboxes easier and their functionality faster.

Securing all information end-to-end is not always desirable, as we saw above. SEAMS discusses how signaling of end-point information to the middleboxes extends network functionality [19]. It showed how devices on signaling path can request for specific information from the end-points and hence made selective disclosure of information possible. This allowed the middleboxes to have fine granular policies based on specific application, user or host information, which in turn enhanced the functionality of middleboxes. For example, signaling application information can be useful for a QoS box to prioritize voice traffic over data traffic without resorting to deep packet inspection. Although SEAMS allows negotiation between end-points and middleboxes [19], the information transmitted is still visible to all network devices. Hence, we need a stricter implementation of information disclosure in the signaling channel.

The above approaches (Blinding and SEAMS) are mutually exclusive to each other. Revealing cryptographic identities or the digital certificates in plain text disregards the privacy concerns of end-points (e.g. Distinguished Names in X.509 certificates). Conversely, blinding of cryptographic identities makes it impossible for the middleboxes to extract specific information about user or host which causes the middleboxes to fallback to traditional mechanisms like looking into packet headers or stronger measures like deep packet inspection. Also, blinding of identities leaves the negotiation mechanisms of SEAMS completely useless. Summarizing the contrasting positions of each of these approaches, we are still in need of a solution where the privacy concerns of end-points and the concerns of middleboxes are more in accordance with each other.

1.1 Contribution

In this thesis, our focus is to address the above discussed issues with an extension to Host Identity protocol (HIP). From a detailed analysis of the conflict of interest between the middleboxes and the end-points, we derive our problem statement.

We extend Host Identity Protocol (HIP) to allow for signaling of information requests, information responses and protection of end-point information within the HIP Base Exchange. Firstly, we enable the middleboxes to have a policy based on cryptographically verifiable identifiers and informations, and allow the middleboxes to request for specific information. The end-points can then choose to respond to the middleboxes they trust with a tighter control over the visibility of the information hence, allowing them to make policy decisions based on requested information.

Secondly, we provide a mechanism for the end-points to establish a trust relationship with network devices and vice-versa. The level of trust may depend on the policy of

the end-point as well as the position of the middlebox in the signaling path. This establishment of trust is based on Public Key Cryptography wherein the involved entities have to prove their identity and the authenticity.

Thirdly, we combine the principle of "minimal disclosure" [1] of information with limiting the visibility of information to authorized entities. We also provide tools that work within the signaling layer to secure end-point information. Finally, we discuss the architecture of our solution and the implementation details. We then present the evaluation of our implementation in various scenarios.

2

Background

Our work analyzes the excess or absence of information available on the signaling path and establishes mechanisms that allow a secure and trustful end-to-middle sharing of sensitive information. We extend the Host Identity Protocol (HIP) and exploit the HIP Base Exchange for negotiation of relevant information and signaling of sensitive information. This chapter focuses on providing a brief overview of the concepts, terminologies, and protocols that are extensively used in our approach. In the first section we provide the definition of middleboxes and discuss the role that they play in current network infrastructure. In the second section, we provide basics and key concepts of Public Key Cryptography along with an emphasis on symmetric and asymmetric key cryptography. In the third section we give a brief overview of some of the important security protocols like TLS and HIP. In the final section, we introduce Host Identity namespace and present an overview of HIP Base Exchange which is important for the scope of this work.

2.1 Middlebox

The Internet as we see today stands on three important components: computing platforms (end-points), packet transport (i.e internetworking) infrastructure and services (application) [29]. This architecture of Internet combined with tightly associated global namespaces of Internet Protocol (IP) and Domain Name Service (DNS) have made the Internet scalable to the point where it can support billions of devices and millions of services. This growth has been successful largely because of the large scale deployment of millions of network devices whose roles range from being a firewall to Application Gateway. These network devices which serve purposes much more than simple packet routing, form a new category of infrastructure elements called *middleboxes*. The term *"middlebox"* originally coined by Lixia Zhang is defined as follows,

A middlebox is defined as any intermediary device performing functions other than the normal, standard functions of an IP router on the datagram path between a source host and destination host. [5]

2.1.1 Middlebox Functions

The presence of middleboxes throughout the Internet has enriched the network functionality with a variety of functions that they could offer to new services, based over Internet and towards higher security. However, in order to perform these functionalities require information from the signaling path. Traditionally, the middleboxes had to depend on coarse identifiers like IP address, Domain Names to extract the information they. Certain middleboxes also resort to the costly deep packet inspection in order to extract information not freely available (e.g. Application Information). With the absence of context information in signaling channel, many approaches (e.g. NUTSS, Pedigree, SEAMS) have come up which already provide the high level information (e.g. User Identifier or Host Identifier) to the middleboxes. We discuss these approaches in more detail in the next chapter. Here, we would like to discuss few of the common middlebox functionalities [5].

2.1.1.1 Network Address Translation NAT

Network Address Translation is one of the most common of all middlebox functionalities often found in home routing solutions. Network Address Translation is the process of transforming the IP information in IP packet headers while the packet is being transited across a routine device. The most basic form of a NAT device does an one-to-one IP address translation i.e one IP address is mapped to exactly to some other IP address. This needs changing only the IP header checksum and the checksums that include IP address while the rest of the packet can be left unmodified. However, today Network Address Translation is used along with the Port Translation also known called as NAPTs. These NAPTs devices make use of the port information available in the IP packet header and multiplex the traffic originating from a private network to a public network with a globally routable IP address. Correspondingly, NAPTs have to update the checksums corresponding to IP address and port numbers in the packet. NATs and NAT derivates like NAPTs have been instrumental in handling the cause of IP address exhaustion, interconnecting two networks with incompatible addressing schemes and reducing the cost of holding multiple public IP addresses.

2.1.1.2 SOCKS Gateway

SOCKS is a stateful method for authenticated firewall traversal where the client must authenticate with the SOCKS server in the firewall before it can traverse the firewall.The IP address and port information used outside the firewall is determined by the SOCKS server. However, applications at the client side must support the usage of SOCKS and address-sensitive application must take SOCKS into account. Furthermore, the authentication mechanism requires the client to send host/user

identifiers to the SOCKs gateway (middlebox) which might be of privacy concern particularly if these identifiers are sent in plain-text.

2.1.1.3 Firewall

The primary role of a firewall is to allow or block incoming or outgoing traffic based on a set of static or dynamic rules derived from a network policy. Firewalls are usually located at the edge of the network acting as a bridge between the trusted internal network and the untrusted external network or the Internet. However, this does not limit the firewalls from being installed on user machines. These firewalls also called *personal firewalls*, control and restrict the access of application and services on the system to the external network and vice versa. The personal firewalls provide an additional layer of security particularly useful when connected to an untrusted network. Firewalls can be broadly classified into the following two types,

- **IP Firewalls** : These are the simplest and oldest firewalls that control and restrict packets based on IP and Transport headers, i.e., based on IP address, subnets and port numbers. Typically the packets are not modified while being processed by the firewall.
- **Application Firewalls** : Application-level firewall act as a protocol end-point and relay (e.g. SMTP client/server). They usually perform extensive protocol validity checks.

The functionalities of the firewall can be further enriched if the context information about the user, the host, or the application is signaled directly to them. In that case, they would not need to resort to deeper inspection of packet or rely on costly and complicated learning mechanisms to learn information about the traffic flow. They can also have fine-granular policies based on context information.

2.1.1.4 Quality of Service

In certain resource constrained networks, bandwidth management might be important. This requires the network to have various 'Quality of Service' levels. The network traffic can be systematically categorized into different levels (for example, based on user privileges) and marking them accordingly. The classification of traffic allows the QoS middlebox to assign higher priority to a flow marked with high category. In times of higher congestion, the QoS middlebox can drop the packets from a lower priority traffic flow. The classification is typically based on the privileges assigned to the sender (user), the subscription to a higher service level (user) or packet header information like application layer protocol. The QoS boxes can benefit greatly from the availability of user identifier and user privileges or the availability of the application information in the signaling channel.

2.1.1.5 Tunnel End-Points

A tunnel is a virtual communication between two end-points, an end-point and a network, or two networks. Tunnels create a new "virtual" network and network interfaces based on Internet infrastructure. Generally, tunnels do not correspond to

actual end-points, instead the tunnel end-points are gateways inside the network. Also tunnels usually multiplex traffic from multiple sources and destination pairs inside the network. A tunnel needs an encapsulator and a decapsulator. The encapsulator takes a datagram and encapsulates it and forwards it to a pre-defined tunnel end-point. Upon reaching the tunnel end-point, the encapsulation header is removed and the original datagram is then processed further. The tunnel endpoints follow the end-to-end principle in the outer internet. Although tunnels play an important role in network security they also some side-effects to the MTU size (they reduce it) and to ICMP replies (lack necessary diagnostic information). The securing of payload channel by the tunnel protects the sensitive end-point information from being leaked to untrusted on-path devices.

2.1.1.6 Proxies

HTTP1.1 defines a Web proxy as follows [12]:

> *"An intermediary program which acts as both a server and a client for the purpose of making requests on behalf of other clients. Requests are serviced internally or by passing them on, with possible translation, to other servers. A proxy MUST implement both the client and server requirements of this specification. A "transparent proxy" is a proxy that does not modify the request or response beyond what is required for proxy authentication and identification. A "non-transparent proxy" is a proxy that modifies the request or response in order to provide some added service to the user agent, such as group annotation services, media type transformation, protocol reduction, or anonymity filtering."*

If the firewall does not allow outgoing packets (e.g. a university network employing a proxy server to connect to the Internet), the web proxy should be associated with a firewall. The client side should also explicitly use proxy in such a system. It is important because the IP packet flow is terminated at the proxy servers and a new one is recreated. As defined earlier, such devices are also counted as middleboxes. Web Proxies require authentication from users, hence their functionalities would benefit from the signaling of user identifiers on the signaling channel.

2.2 Cryptography

Cryptography provides a service in which information can be shared between participants in a way that prevents non-trusted parties to access or read it. Cryptographic functions generally involve a secret and an algorithm and are based on the fact that it is computationally infeasible to reverse compute a plain text from cipher text without knowing the secret [23]. Apart from preventing unauthorized access to the information, cryptographic algorithms also ensure that the information has been received untampered. The security properties on the information and the participants which must be ensured by any cryptographic infrastructure are as follows,

- **Confidentiality** Confidentiality limits the access of any data to only the authorized or trusted entities. Encrypting the information with the secret known only to the trusted entities can keep information confidential.
- **Authentication** Authentication is process of establishing the identity of an entity to the one it claims to be. This process is not only applicable to the entity who requests for a service but also the one providing the service. The usage of public key pairs with certificates signed by a Certificate Authority are popular ways to authenticate an entity.
- **Integrity** Message integrity provides protection against any malicious or accidental changes to a message. It is typically achieved using cryptographic checksums like Message Authentication Codes (MAC).
- **Non-repudiation** Repudiation is an act of denying that you sent a message. Non-repudiation guarantees that an entity can not deny a statement it had made earlier. Digital Signatures are the most common method to achieve it.
- **Privacy** Privacy refers to the right of an entity to divulge only the amount of personal information it has explicitly agreed to. Often, an endeavor to keep an information private requires the above discussed methods along with fine granular policies.

We will again look into some of these properties in the third and the fourth chapters.

2.2.1 Symmetric Key Cryptography

Symmetric Key Cryptography is one of oldest and the best-know cryptographic techniques. Symmetric key cryptography requires a single private key to be used for both encryption and decryption of data. This means that both the parties need to have the private key prior to any sharing of encrypted information. Symmetric key cryptography is popular because given the secret it is relatively inexpensive to compute the plain text from a cipher text, but it computationally infeasible to produce the plain text from a cipher text without the knowledge of the secret key. Moreover, unlike asymmetric cryptography it is relatively inexpensive to compute a strong key and the algorithms for symmetric key cryptography are computationally very efficient. This is because the keys for asymmetric key cryptography require generation of large prime numbers, while keys for symmetric keys cryptography are pseudo-random strings of fixed length (e.g. 128 bit length for AES-128). Encrypting data using symmetric key cryptography does not ensure that the data has not been tampered with. Hence, Message Authenticated Codes (MACs) are appended to the message to check the integrity of data.

MACs are vulnerable to “length-extension attacks”, where data may be appended to the message without knowing the key yet obtaining a valid MAC [2]. In order to circumvent such attacks, HMACs are used which are resilient against “extension attacks”.

The major drawback of symmetric key ciphers is the sharing of secret securely in an insecure communication channel. Hence, to prevent risking a compromise of secret

key, asymmetric key exchanging algorithms like Diffie-Hellman Key Exchange are used to generate session specific secret key based on a pre-shared secret. Sometimes, asymmetric cryptography is also used to distribute the secret e.g. TLS [8].

Symmetric key encryption can be used with *block-ciphers* or *stream-ciphers*. Block-ciphers typically work on a fixed-length of data by dividing the data into blocks and then operating on each of these blocks. They are widely used for encrypting large chunks of data that are fixed i.e unvarying transformation. Stream ciphers on the other hand, work by operating on each of the plain-text digit with a pseudo randomly generated cipher stream. These are usually computationally less intensive when compared to block ciphers. Hence, they have large scale implementation with low hardware complexity.

Symmetric key cryptography being mainly composed of simple operations like substitutions and permutations, have found efficient implementations in both software and hardware. Popular symmetric key algorithms include AES [32, 3], RC4 [24], 3DES [31] etc.

2.2.2 Public Key Cryptography

Asymmetric Key Cryptography first proposed by Witfield Diffie and Martin Hellman in 1977, is based on a pair of associated keys to secure a message. The principal behind asymmetric key cryptography is a *trap door* or *one-way* function. Such functions are easy to compute one-way but difficult to reverse. For example, computing the product of two numbers is easy but factorizing the product is not. However, given the product and one of the factors its easy to find the second factor. Thus, computation is easy on one direction and the reverse can be made easy by the knowledge of a secret. *Asymmetric Key Cryptography* is also commonly referred to as *Public Key Cryptography*.

An asymmetric key pair comprises of a public key and a private key. The public key is distributed freely to anyone interested in the sending a secure message and the private key is kept secret. Public key is used to encrypt data or a message and the encrypted cipher text can be decrypted only using the algorithm used for encryption and the matching private key. Unlike symmetric key cryptography, the distribution of public keys does not require the communication channel to be secure. However, the public key still needs to verified to be the one from whom it claims to be. The verification can accomplished through a mutually Trusted Third Party (TPP) or other certification mechanisms. Public Key Cryptography satisfies some key security properties mentioned in Section 2.2.

The first property we would like to talk about is *confidentiality*. In order to send a message securely, one has to encrypt the message using the public key of the entity on the other side. Only the entity who possesses the corresponding private key can successfully decrypt it. Furthermore, the message can be made more secure, by first encrypting the message using the public of the other side, followed by encrypting the generated cipher text using own private key. The other side would have to decrypt the cipher text using the public key of the sender, followed by its own private key. Therefore, Public Key Cryptography provides a strong mechanism to keep the data confidential and it can be read only the person who was meant to.

Secondly, Public Key Cryptography make repudiation nearly impossible (*non-repudiation*). Public key algorithms like RSA, DSA and ECDSA provide digital signatures over a plain-text using the sender's private key. The receiver can decrypt a cipher text and the verify the signature over the plain-text using the sender's public key.

The advantages of public cryptography, albeit significant, are not suitable in every scenario because of their performance. Asymmetric key cryptography is based on big number operations and that makes the operations in the process relatively expensive, hence not so efficient when compared to symmetric key cryptography. Encrypting or signing large chunks of data or a stream of data with public key cryptography is not feasible due to the complexity overhead. Next, we would discuss few of the most popular public key cryptography algorithms.

2.2.2.1 RSA

RSA named after Rivest, Shamir and Adleman, is a popular public key cryptographic algorithms that does encryption, decryption and signing. RSA key length can ben chosen to be large for increased security and short for increased efficiency. As mentioned previously, it is not efficient to encrypt large chunks of data using RSA. However, it is a common trend to encrypt a secret key using RSA and use the symmetric key obtained after decryption for further communication.

Algorithm : The algorithms generatesits two large prime numbers p and q commonly larger than 512-bits. n is obtained after multiplying these two primes such that p and q are the factors of n. From p and q, $\varphi(n) = (p-1)(q-1)$ is obtained. To generate the public key, another number e, public exponent, is chosen such that it is relative prime to $\varphi(n)$. The public key is $< e, n >$. To generate the private key, d is computed as the multiplicative inverse of $e \ mod \ \varphi(n)$. The private key is $< d, n >$.

A message $m(< n)$ can be encrypted using the public key by computing the cipher text $c = m^e \ mod \ n$. The cipher text when received by the receiver can be decrypted by computing $m = c^d \ mod \ n$. Conversely, a message $m(< n)$ can be signed with the private key, $s = m^d \ mod \ n$ and the signature can be verified using the public key as $m = s^e \ mod \ n$. Choosing a small public exponent allows for faster encryption and signature verification, but the private key has to large enough so that it can not be broken by brute force.

2.2.2.2 Diffie Hellman Key Exchange

Diffie Hellman is the oldest public key crypto system still in use and came into existence even before RSA [9]. Although it can neither do encryption nor signing it holds an important place in public key cryptography. It facilitates two participants to agree upon a shared key in spite of the fact that they can exchange messages only in public. Both the participants exchange messages in public and after the exchange they both have are in a possession of secret key (secret from the rest of the world). Any further communication can be encrypted with this secret using any of the know symmetric key algorithms.

Diffie Hellman Exchange works by generating two numbers p and g, where p is a large prime and g is called the *generator*. Both the parties must have already agreed on these numbers and both p and g can be publicly known. Each of the parties correspondingly choose a large number S_A, S_B at random and then compute $T_A = g^{S_A} \ mod \ p$, $T_B = g^{S_B} \ mod \ p$. The results T_A and T_B are exchanged and consequently can be used to generate the *secret* $T_A^{S_B} \ mod \ p = T_B^{S_A} \ mod \ p = g^{S_A S_B} \ mod \ p$. The principle behind this algorithm is that, from g^{S_A} and g^{S_B} it is not feasible to compute $g^{S_A S_B}$ without the knowledge of S_A or S_B.

The weakness of Diffie Hellman Key Exchange lies in the fact that there is no authentication involved before setting up a shared key. Hence, one party may establish a shared secret with a malicious party. This makes DH Key Exchange vulnerable to *man in the middle* attack [23, 10]. To overcome this weakness, often a variation of Diffie Hellman Key exchange, called *authenticated Diffie Hellman Key Exchange*, is used by combining asymmetric key cryptographic algorithms. One possible example of this is signing the Diffie Hellman values with the corresponding private key so that the other party can verify the authenticity of the sender and the integrity of the message.

2.2.2.3 How secure are RSA and Diffie Hellman?

The security of the RSA algorithm is based on the problem of factoring. Similarly, the Diffie Hellman problem is based on the discrete logarithm problem. Hence, a brute force attack on these crypto systems would essentially require a computation of exponential complexity. These problems have been proven to be computationally hard. However, sub-exponential (less than exponential complexity) algorithms that have transformed the complexity to super polynomials (more than fixed degree polynomial) have been developed [23]. Since the complexity for these newer algorithms is sub-exponential, the security has to be increased by increasing the key size (generally larger than 1024 bits), but this also results in making the cryptographic operations more expensive. Large keys also take large transmissions. Next, we look into a new crypto system called Elliptic curve cryptography which allows us to have smaller key sizes without compromising the security.

2.2.2.4 Elliptic Curve Cryptography ECC

The weakness of popular public key crypto systems like RSA and Diffie-Hellman Key Exchange has led to the popularity of an alternative set of cryptographic algorithms, i.e., Elliptic Curve Cryptography. The strength of Elliptic Curve Cryptography comes from the absence of any known sub-exponential algorithms to break it [23]. Hence, it is expected to be more secure with even smaller key sizes. A 256-bit elliptic curve is cryptographically as secure as 3072-bit RSA key-pair. Elliptic curve operations have been computationally more expensive than the algorithms based on modular arithmetic, hence seeking for more efficient algorithms.

An elliptic curve is a set of points on the coordinate plane satisfying an equation of the form $y^2 + axy + by = x^3 + cx^2 + dx + e$. An elliptic curve over real numbers is also called Weierstrass's equation. This equation can be further simplified without

any loss of generality to $y^2 = x^3 + ax + b$. Elliptic curves can be solved with at least two types of arithmetic and both of them are based on two different definitions of arithmetic. In order to solve any generic cubic function we draw a line cutting the graph of the function at two points and the third point of intersection with the graph gives the third root. The multiplication in the context of elliptic curves is slightly different. The product of two points in this multiplication can be obtained from the graph by drawing a line through the two points, finding the third intersection point of the curve and the line and the reflecting the third point across the x-axis. This multiplication is associative and by defining the inverse of a point to its reflection across x-axis, the point at infinity the identity **I**, we get a group. From above, the two types of arithmetic are [23]

- $\mathbf{Z_p}$: modular arithmetic with a large prime p as the modulus
- $\mathbf{GF(2^n)}$ arithmetic, which can be done with shifts and $\oplus$s. The operations for this *group* can be thought of as modular arithmetic of polynomials with coefficients *mod* 2

Instead of taking the coordinate plane as the real plane, we can also take the plane to be coordinate plane of finite fields. Then this elliptic curve group is also finite. By choosing an element from this group, we can use the resulting cyclic sub-group. In other words, we can transform the popular cryptographic algorithms like Diffie Hellman which are also based on the cyclic group of $\mathbf{Z_p}^*$, to their elliptic curve variations. Also the discrete logarithm problem over elliptic curve groups is believed to be harder than over $\mathbf{Z_p}^*$. Similarly, for other public key cryptographic algorithms, the modular arithmetic can be substituted with elliptic curve multiplication, giving us more such transformed crypto systems like ECC ElGamal etc.[23]

Elliptic Curve Diffie Hellman ECDH - As described in section 2.2.2.2, for regular Diffie-Hellman, two numbers p and g are generated on which both the parties have to agree on. Similarly, for ECDH p can be replaced with constants from the elliptic curve equation and the type of arithmetic to be used. g can be replaced with a point on the elliptic curve [23].

2.2.2.5 RSA vs ECC

As shown earlier, elliptic curve operations can been done with at least two type of arithmetics ($\mathbf{Z_p}$ and $\mathbf{GF(2^n)}$), which are similar in some ways. Hence, the implementation of Elliptic Curve cryptography is not more complex than modular arithmetic with large integers. ECC private key operations are believed to be faster than RSA typically because of smaller keys [23]. However, public key operations such as signature verification can be faster for RSA even with large keys because a small public exponent can be chosen.

2.2.3 Public Key Infrastructure PKI

Public Key Infrastructure provides the users of an insecure public network the ability to share data securely using an asymmetric key pair that can be acquired from a

trusted authority. The public key infrastructure (PKI) makes use of *Public Key Cryptography* to authenticate a message sender and encrypting a message. It also serves the infrastructure with the facility to obtain *digital certificates* for a public key pair, authenticates the holder of the digital certificate and a directory service which can store and when necessary revoke certificates. We discuss more about public key certificates below. The public key infrastructure consists of the following,

- **Certificate Authority, CA** It is responsible of issuing and verifying digital certificates.
- **Registration Authority, RA** It verifies the Certificate Authority prior to issue of certificate to the requestor.
- **Directory Service** to store and index the certificates issued.
- **Certificate Management System** for generation, distribution, verification and revocation of digital certificates.

2.2.3.1 Public Key Certificates (X.509 Certificates)

A **public key certificate** is a digitally signed statement from a trusted entity, saying that the corresponding public key (and some additional information) has a specific value [7]. As we have seen earlier , public key cryptography needs access to public keys of the entity it wants to communicate to. In a large-scale network we can not assume that the communicating entities already possess the public keys of each other owing to a prior relationship or a trusted repository exists with all the public keys. Hence, a *public key distribution mechanism* is required. Digital certificates help in solving the public key distribution problem [7]. A certificate authority (CA) can act as a trusted third party (TPP) to sign and issue certificates for other parties. The signature on one hand guarantees the integrity of data in the certificate and on the other hand guarantees the validity of the certificate to the entities which trust the certificate authority. The holder of the certificate can then prove its identity to the verifier by proving that it possesses the private key corresponding to the public key found in the certificate (that has been signed by a trusted third party).

The role of public key cryptography is to provide an entity with methods to authenticate an unknown entity with the help of a trusted entity but without any prior knowledge of the other entity's public key. The digital certificates not only help in establishing the identity of an unknown entity but also securely associate additional informations regarding the entity to that identity.

ITU-T X.509 [21] standard defines the information than can be bundled with the certificate and also describes the data format. X.509v3 is the most recent version of the standard and supports the notion of extensions [21]. Extensions can be defined and included in the certificate. Some of the common pre-defined extensions are *AltenativeNames*, *KeyUsage* etc. Extensions can also be marked *critical* to make that extension mandatory and the party verifying the certificate must verify the extensions marked as critical. An X.509 certificate has the following data,

- **Version**: It contains the information about the version of X.509 standard that is being used in the certificate.

- **Serial Number**: It contains information about the serial number of the certificate when the CA issued it. This serial number is put in the Certificate Revocation List when a certificate is revoked.

- **Signature Algorithm Identifier**: The algorithm used by the CA to sign the certificate.

- **Issuer Name**: It contains the X.500 name of the issuer of the certificate i.e the CA and signifies that the party using the certificate trusts this issuer.

- **Validity Period**: Validity of the certificate.

- **Subject Name**: It contains the name of the entity to which the certificate belongs. The name of the holder of the certificate is also sometimes called Distinguished Name (DN) and follows the X.500 standard.

- **Subject Public Key Information**: It contains the public key of the certificate holder, along the algorithm that was used to generate the public key pair.

2.3 Security Protocols

The Internet is an untrusted domain with a variety of trusted and untrusted network devices and end-hosts. Network entities need to trust each other before they share any information (authentication) and they should be able to verify that the information received has not been tampered (integrity protection) in the communication path. Owing to these reasons, security protocols have been developed which aim at providing the security properties to the information being shared (see Section 2.2). For the scope of our discussion, we would be presenting two popular security protocols here, HIP and TLS.

2.3.1 Host Identity Protocol, HIP

Host Identity Protocol, defined in RFC 5201 [30] is a two-party cryptographic protocol to establish a communication context between hosts. The first party is called the 'Initiator' and the second party is called the 'Responder'. The HIP handshake is called the Base Exchange (BEX). It is a four-way message exchange, where the first packet triggers the Base Exchange. The second and the third packet are used to exchange Diffie-Hellman Keys, to be used to secure the payload channel. The third and the fourth packets allow the parties to mutually authenticate each other. HIP also incorporates a puzzle mechanism to protect the Responder against Denial of Service (DoS) attacks. After the completion of the HIP BEX, HIP establishes a secure payload channel (IPSec tunnel over Encrypted Security Payload [30]) for the data packets which follow after the fourth message. We discuss HIP is more detail in Section 2.4.

2.3.2 Transport Layer Security (SSL/TLS)

SSL/TLS is a protocol that allows two end-points to authenticate themselves and agree upon a session key to cryptographically secure the remainder of the session [23]. TLS is designed to run on top of TCP. This allows it to run as a user-level process, hence preventing the need for any change in the lower layers or the operating system. Also, running it on top of TCP gets rid of any need to handle time out and retransmission of lost data.

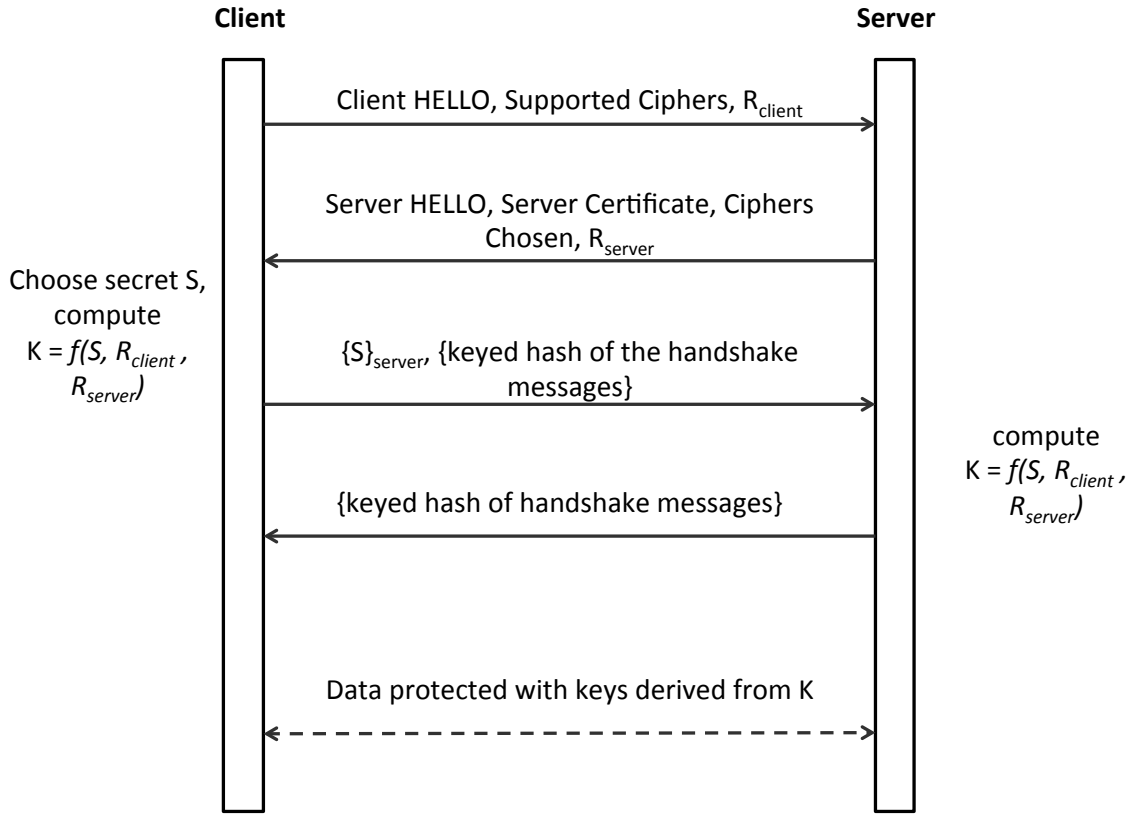

Figure 2.1 Simplified view of TLS Hanshake

In the basic protocol the client initiates a connection with the server. The server follows it up by sending its certificate. The client then verifies the certificates and generates a random key '**S**' from which a session key can be generated. The client encrypts S with the public key of the server and sends it. The server receives the message, decrypts it to get S and then generates the session key. The payload transfer of the session between the client and the server is encrypted and integrity-protected with those session key. This is however a simplified form of the protocol. In Figure 2.1, we have a detailed view of the TLS handshake [8],

- **Message 1** : The client initiates the connection and sends a HELLO message to the server along with information about the cryptographic algorithms it supports and a random number R_{client}. R_{client} will be used with S in *message 3*.
- **Message 2** : The server upon receiving the HELLO from client, send its certificate and a random number R_{server} in the second message. The server

replies with a cipher which it supports from the list of supported cryptographic algorithms sent by the client. Similar to *message 1*, R_{server} will be combined with those received to generate a symmetric key.

- **Message 3** : The client verifies the certificate received, generates a random number S (also called **pre-master secret**), encrypts it with the server's public key and sends it. Along with the encrypted secret, the client also sends a hash K of the **pre-master** secret and the handshake messages. The hash is for the integrity protection of messages in the handshake as well as to prove that it has received the public key of the server successfully. The keys to be used for encrypting the rest of the communication is generated after hashing K, R_{client} and R_{server}.

- **Message 4** : The server proves its knowledge of session keys and of all the earlier handshake messages, by computing a keyed hash of the all the earlier handshake messages and encrypting it with the encryption key. The keyed hash is also integrity-protected with the corresponding integrity protection key. Since the session keys are derived from S, it can be assumed that the server holds the private key as S was received encrypted.

We can observe from the above set of messages that the server is authenticated by the client but not vice-versa. However, TLS does provide a mechanism for the server to explicitly authenticate the client if the client has a certificate (*CertficateRequest*), but it is not common [8]. More commonly, the server is authenticated via its certificate and the client is authenticated by the server at application level (on the payload channel) by asking for its identity and password cryptographically protected (encrypted and integrity protected) with the session keys.

2.4 Host Identity Protocol (HIP)

The problem of naming of hosts and data have been discussed for long in the Internet Engineering Task Force (IETF). Traditionally, IP addresses have been used both as "identifiers" (end-point or host identifiers) as well as "locators" (routing labels) [30]. There are two main problems with the dual use of IP addresses. Firstly, these identifiers are not cryptographically bound to the identity of the end-point (anyone can fake it). Secondly, packet routing follows IP addresses as locators, they fail in roaming scenarios.

The first significant step towards addressing this problem came from the conception of Host Identity Protocol (HIP) by Robert Moskowitz in 1999 in RFC 4423 [29]. HIP is an end-to-end authentication and key establishment protocol.

HIP uses public cryptographic keys, a public-private key pair, as identifiers (e.g. host identifier) [30]. In Section 2.4.2, we discuss how these new identifiers can mitigate the problem with the dual use of IP addresses. A detailed outline of Host Identity Protocol can be found in RFC 5201 [30]. Numerous extensions to HIP have been developed and can be found at separate drafts and RFCs: Mobility and multihoming [33], IPSec ESP integration with HIP [22], NAT Traversal [26] and middlebox authentication [16].

2.4.1 HIP in the Internet Architecture

The only routable network layer protocol in the Internet has been the Internet Protocol (IP). Transport layer protocols such as TCP and UDP run on top of IP layer. Furthermore, a number of application layer protocols like HTTP and SMTP have been developed and work on top of transport layer protocols. This organization of protocols, stacked on top of each other gives a *hour-glass* model of Internet with IP as the narrow waist of the model, as shown in Figure 2.2.

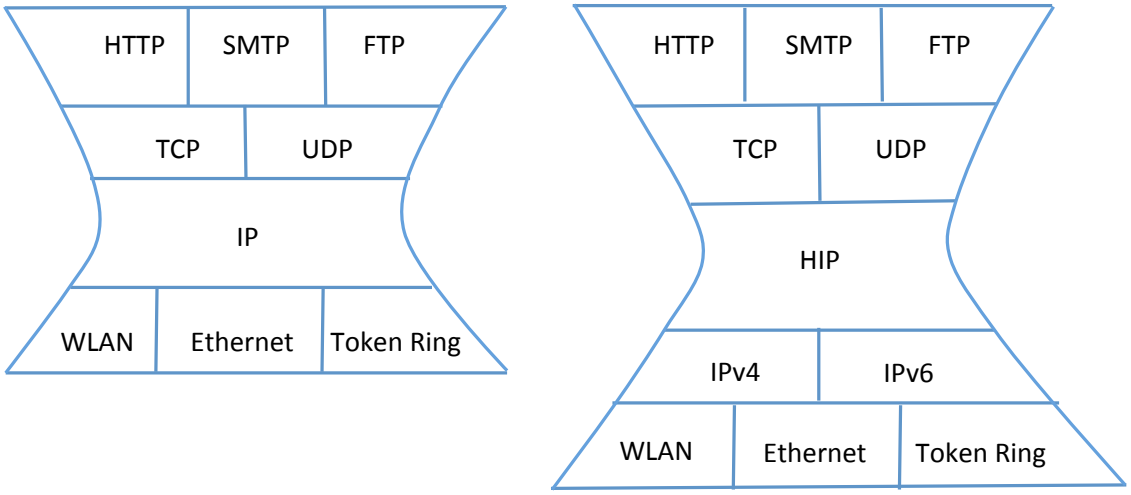

Figure 2.2 Hour Glass Model of Internet

There is a strong coupling between network and transport layer in the Internet architecture. For example, TCP has a pseudo-header that includes the IP address in checksum calculations. Thus, independent modifications to any of the protocol is very difficult without affecting the other layers and introduction of a new transport layer or network layer protocol would ask for extensive changes to the existing infrastructure. Introduction of IPv6 added to the existing woes. its incompatibility with the existing network layer and transport layer protocols and the legacy IPv4 protocol have been detrimental to its adoption. Moreover, the co-existence of both of these protocols have created a dual-stack IP layer where both of them have to be separately routed.

Host Identity Protocol can get rid of the dual-stack waist of the Internet architecture by becoming the new waist and sitting on top of IP layer. So, HIP layer would run on top of IPv4 and IPv6 and the transport layer protocols like TCP and UDP would run on top of the HIP layer, as shown in Figure 2.2.

2.4.1.1 New HIP Layer

According to RFC 5201, HIP proposes an alternative architecture to the dual use of IP address as “locators” and “identifiers” [30]. In HIP public cryptographic keys are used as identifiers (host identifiers). Host Identifiers have helped in getting rid of the need of IP addresses and along with it the limitation of IP address namespaces. Higher layer protocols can now be tied to these cryptographically verifiable host

identifiers rather than IP addresses. Hence, dynamic modifications to IP addresses can be directly authenticated between hosts (mobility) and if requested, a strong authentication between TCP/IP stack level can also be achieved.. As a result end-host mobility and multi-homing leverage the support provided by HIP namespace. Applications and implementation relying on upper layer protocols do not need any modification to attain compatibility with HIP layer.

The new **HIP-layer** is inserted above the network layer and below the transport layer, as shown in Figure 2.2, which can translate non-routable HIP identifiers to routable IP-addresses. Also, the HIP-layer may store a mapping of HIs with IP address representations and update its entries as soon as host changes its location and announce it new address.

2.4.2 Host Identity Namespace

In HIP, hosts possess a self-generated public key-pair that can be used for host identity. The public key is used as Host Identifier (**HI**). A typical public key generated using RSA can be of 512, 1024 or 2048 bits. As a part of RFC 5201, most implementations of HIP also support DSA algorithm. However, using public keys directly as host identifiers is inconvenient owing to their large and variable sizes. Even the shortest of public keys would easily exceed a typical Maximum Transmission Unit (MTU) of 534 bits [37]. Hence, we need smaller representations of Host Identifiers. For the sake of compatibility, with existing application using Berkley Socket Interface, two more forms of host identifiers were proposed as depicted in Figure 2.3.

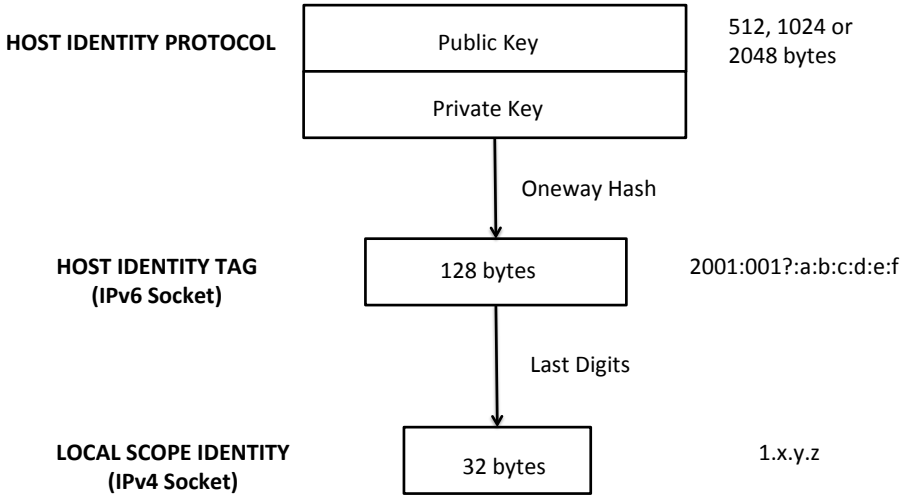

Figure 2.3 Host Identity Namespace

2.4.2.1 Host Identity Representations

A Host Identifier Tag (**HIT**) is a 128-bit hashed encoding of Host Identifier. HITs are mostly used in the HIP Base Exchange (signaling handshake) to address the Host Identity. Keeping the HITs to 128-bit serves three important purposes for host identifiers,

1. It is of same length as the IPv6 addresses and hence can be used in address sized fields in APIs and protocols.

2. It is **self-certifying**, i.e., it is not computationally feasible to generate a public key that matches the given HIT.

3. The probability of HIT-collision between two host identifier keys is very low, which makes it infeasible for an attacker to compute a collision with the HIT in use.

HITs have a prefix of **2001:0010::/28**. This is to differentiate them from currently allocated IPv6 addresses. Also, a fixed length identifier protects the upper-layer protocols from any dependency on cryptographic algorithms used to generate public-private key pairs.

Another form of host identifier derived from the **HI** is the Local-Scope Identifier (**LSI**). It is 32-bit identifier formed by taking the last few bytes of HIT. The probability of collision of LSIs is significantly higher owing to their shorter lengths. Hence, LSIs are considered to be safe to use only in local sense and can not be considered to be globally unique. Since LSIs have the same lengths as IPv4 address it can also be used in legacy IPv4 applications in socket interface. In order to differentiate them from publicly allocated IPv4 addresses, LSIs have **1.** as their prefix.

2.4.2.2 Resolving HIP Identifiers

The cryptographic identifiers introduced by HIP are limited to identifying the corresponding hosts, but do not provide any means to locate them. Hence, a HIP-enabled host needs to lookup for the registered IP address corresponding to the HIP identifier, whenever it establishes connection for the first time. RFC 5205 provides an extension to DNS resource records to support HIP Host Identifiers [34]. Another HIP extension defines a new lookup architecture for HIT to IP address mapping using a Rendezvous server. More solutions based on Distributed Hash Tables (DHTs) and Internet Indirection Infrastructure (I3) [39] were provided by Nikander et. al [14].

2.4.3 HIP Control Channel

HIP separates the signaling (*control*) channel messages from the data (*payload*) channel messages. HIP Control Channel contains all the messages that are necessary for the authentication of peers and setup/maintenance of HI to IP mapping and HIP states between the two peers. It is also responsible for negotiation of the characteristics of the payload channel. The state achieved after the establishment of the connection between two HI pairs is known as *host association*. The control messages on the signaling channel are not encrypted. But they are signed and integrity protected using host signatures and message authentication codes (MACs). The signatures serve an additional function for allowing the middleboxes to track and inspect ongoing HIP associations. Along with all aforementioned, the control channel messages also signal a Diffie Hellman Key Exchange which is used to derive a

keying material. This keying material is utilized later in protecting the HIP Payload Channel. Thereby the control and the maintenance of payload channel are delegated to IPSec.

HIP Payload Channel is protected by IPSec using the Encapsulated Security Payload (ESP) transport mode. The HIP architecture was designed to be friendly with middleboxes such as firewalls. Diffie-Hellman Key Exchange followed by IPSec allow middleboxes like firewall to examine an ongoing HIP association. The use of IPSec is not always preferable as its not feasible in busy web servers or light weight devices.

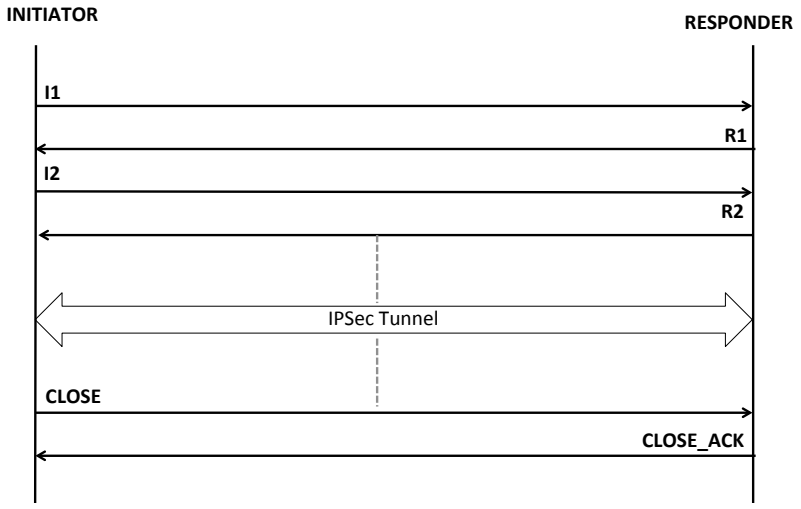

Figure 2.4 The Lifetime of a Connection set up over HIP

2.4.3.1 HIP Message formats

HIP Control messages include seven different messages `I1`, `R1`, `I2`, `R2`, `UPDATE`, `NOTIFY`, `CLOSE` and `CLOSE_ACK`. `I1`, `R1`, `I2`, `R2` are part of the initial handshake. `I1`, `I2` are sent by the Initiator (I) and `R1`, `R2` are sent by the Responder(R). `UPDATE` messages are sent during mobility updates or changes in host associations. `NOTIFY` messages allow to communicate errors and additional information between peers. `CLOSE` messages are sent when the HIP association is terminated. All HIP message exhibit a common header as shown in Figure 2.5. Besides the HITs of the two peers, the HIP header contains the checksum, control flags, encapsulated protocol and some HIP parameters (variable length). This leaves the HIP header without any HIP Parameter to be of size 40 bytes. The lifetime of a connection setup over HIP is shown in Figure 2.4.

2.4.3.2 HIP Parameter formats

HIP Parameters follow a Type-Length-Value (TLV) format as depicted in Figure 2.5. The least significant bit of the *type field* indicates whether a parameter is critical or not. The receiver must be capable of handling a parameter flagged as *critical*. If not then the message should not be processed and the sender must be notified with the `NOTIFY` message. A parameter not marked as critical, may be ignored by the receiver even though it do not handle it. Critical parameters are considered important for security. The provision of critical and uncritical parameters make the

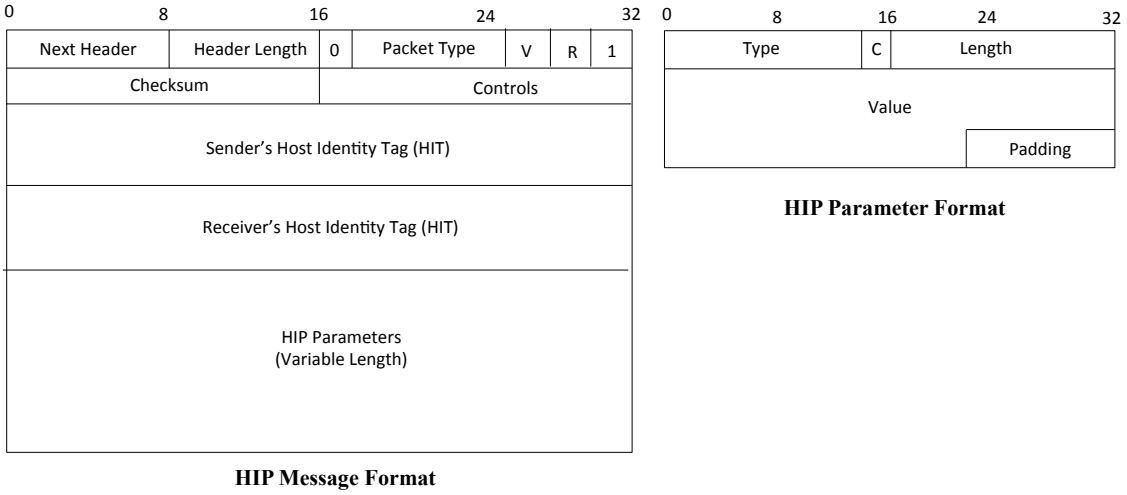

Figure 2.5 HIP Message and Parameter Formats

HIP Control Channel extensible while maintaining compatibility and interoperability with standard HIP implementations.

HIP parameter type values can range from 0 to 65535. Parameter types from 61440 to 62463 are reserved for signature and MACs. All parameters before 61440 are in the signed portion of the packet and are covered by signature and MACs. Conversely, any parameter after the type 62463 is in the unsigned part and is not covered under signature. An important condition for the layout of the parameters in the HIP packet that all the parameters must be in ascending order of their type value. If the parameters are not in order, the packet is considered to be malformed and dropped. An overview of HIP parameter type ranges can be seen in Figure 2.6.

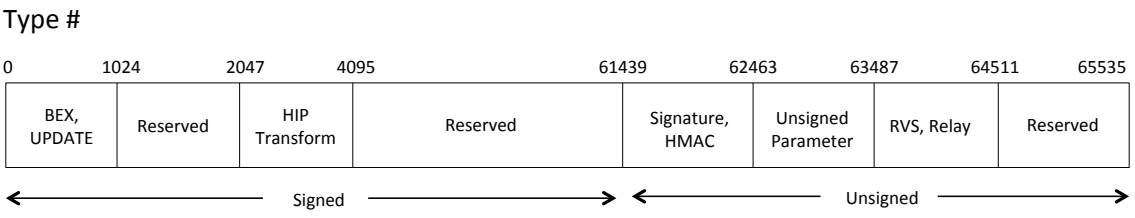

Figure 2.6 The complete HIP Parameter type range

2.4.4 HIP Base Exchange

The initial handshake process of Host Identity Protocol is referred to as `Base Exchange (BEX)`. A brief outline of HIP BEX can be seen in Figure 2.7. In HIP BEX, a host initiates a connection to another host and in this connection setup process both the host authenticate themselves, establish a shared secret session key and setup an secure payload channel using IPSec.

2.4.4.1 Triggering the Base Exchange: I1

A HIP BEX is triggered whenever an application initiates a connection. The initiator resolves the HIT corresponding to the IP address and finds out if there is an existing connection. A new BEX is triggered if there is no existing host association and the

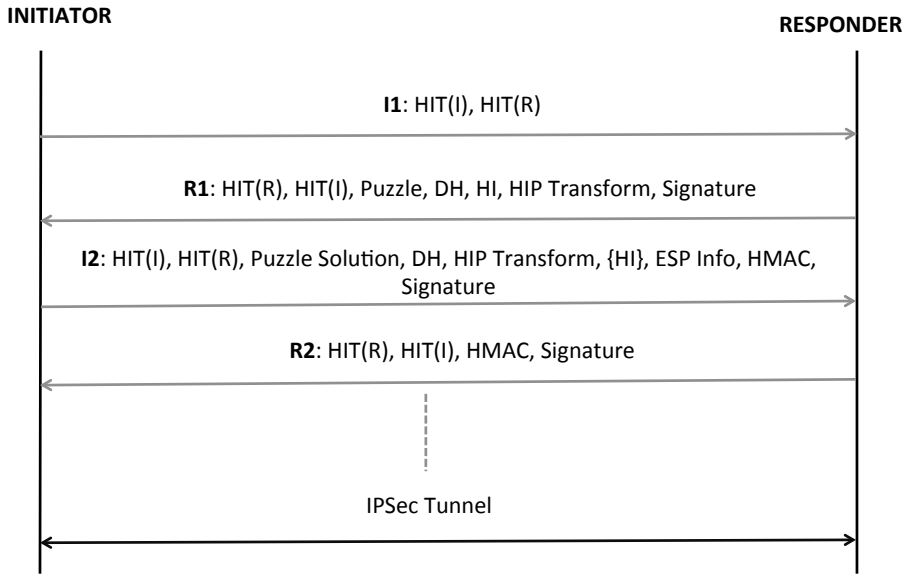

Figure 2.7 Host Identity Protocol Base Exchange

first packet `I1` is sent. `I1` is the minimal packet with only the source and destination HITs, hence does not contain any additional HIP parameters. The idea behind this is to keep the packet as small as possible and the processing of `I1` packet as low as possible. This minimizes the risk of DoS attacks by `I1` flooding.

2.4.4.2 Initiating the Information Exchange: R1

On receiving the `I1` packet, the Responder responds with a pre-created `R1` which is the first packet to begin with any state information exchange. The pre-created `R1` consists of the full Host Identifier (HI) of the Responder, the supported cryptographic algorithms for encryption and integrity protection, the public Diffie-Hellman value (initiating a DH key exchange) and a puzzle (computationally expensive to process). The purpose of sending a puzzle is for the Initiator to solve it before the Responder can allocate a state to the host association. The puzzle also discourages malicious hosts and protects the Responder from a DoS attack. Unlike the `I1` packet, the `R1` packet is signed. However, there is no HMAC as Diffie-Hellman Key Exchange is not complete yet.

2.4.4.3 Continuing with the BEX: I2

On receiving the `R1`, the Initiator verifies the Host Identifier (HI) to HIT mapping, the host signature on the signed part of the packet. After the verification of the `R1` packet, the Initiator continues to compute a solution to the puzzle and chooses its preferred cryptographic scheme from the list proposed by the Responder. The chosen cryptographic transform will be used later to secure the payload channel. The Initiator then chooses the strongest DH-key from the DH-keys sent by the Responder and chooses its own public DH-key value belonging to the same DH-suite, as the one chosen from the Responder. The DH-shared secret is derived from the DH-public key of responder and its own DH-private key. The initiator builds the `I2` and appends the `I2` packet with the solution to the puzzle, its choice of transform, its DH public

value and its complete HI (maybe encrypted). Finally, I2 is signed and integrity protected with the keying material obtained from the shared secret.

2.4.4.4 Finishing the BEX: R2

On receiving the I2 packet, the Responder verifies the HI to HIT mapping and the solution to the puzzle received in I2. Only if the previous is successfully verified, the Responder goes in to verify the host signature on the I2 packet and the HMAC. The Responder then generates the shared secret from the DH public value of the Initiator and its own DH-private value. This shared secret is then used to compute the keying material and an IPSec *Security Association* is derived form the keying material. The Responder has to send an ESP Info parameter which is needed by the Initiator to setup the outgoing *Security Association (SA)*. The Responder appends a HMAC for integrity protection and signs the R2 packet as before.

Upon receiving R2, the Initiator verifies the signature, the HMAC and that the Responder is in possession of the correct session keys. With the ESP Info parameter received with the R2 message it setups the outgoing IPSec Security Association. This concludes the HIP BEX.

2.4.5 HIP Update Exchange

HIP has provisions to update the existing host associations for maintenance of IPSec security associations, for end-host mobility and for multi-homing. UPDATE packet as mentioned in Section 2.4.4.1, is dedicated to exchange of relevant information for updating the existing host associations. The receiver, however, must differentiate between various UPDATE messages by maintained state of host association, parameters in the UPDATE packet and their contents. In most scenarios, the recipient (peer) and on-path middleboxes participating in the connection are the receivers of the update packets. The sender is required sign the packet and integrity-protect it with HMAC for the peer to make secure changes to state. Not doing it would make the peer susceptible to malicious hosts which can inject new HIT-to-IP mapping or launch DDoS attacks.

2.4.6 Middlebox Authentication

Middleboxes need to verify the Host Identifiers of communication peers before they trust the end-point and the end-point context in the HIP Control Channel messages. HIP BEX allows for the authentication of hosts and integrity checking of HIP control messages, but it does not provide mechanisms for the on-path middleboxes to check the freshness of the messages received. This passive verification of identity and integrity is not sufficient as it makes the middleboxes vulnerable to recording and replaying of old HIP control messages by malicious hosts i.e *replay attack*.

The malicious hosts can overhear an old HIP Base Exchange and record the control messages as they are not secured. At a later time, the malicious hosts can replay the old Base Exchange with the help of an adversary. Although the on-path middleboxes can verify and examine the packet, they might not be able to predict if BEX has been replayed. Hence, the adversaries might be granted authenticated privileges

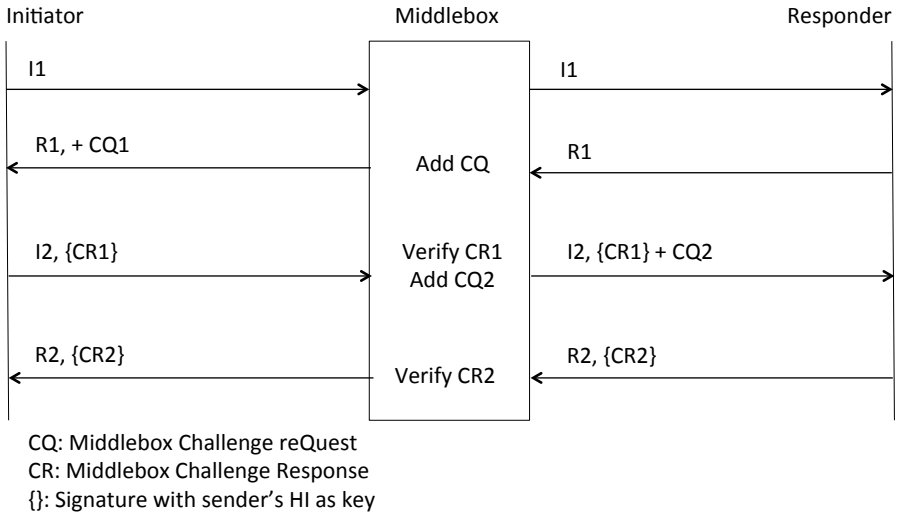

Figure 2.8 Middlebox Authentication of HIP Base Exchange

which are meant only for certain hosts. For example, a firewall which performs access control based on Host Identities.

Heer et. al. provide a solution based on challenge/response method to the replay attack in [16]. It is similar to the older `ECHO_REQUEST/ECHO_RESPONSE` mechanism already employed in HIP-enabled entities [17, 15]. Middleboxes may add `CHALLENGE_REQUEST` parameters to the `R1` and `I2` packets, for the purpose of verification of receiver's identity. `CHALLENGE_REQUEST` parameter contains opaque data which must be echoed back by its receiver (`I2` and `R2` packets respectively). The receiver echoes the opaque data back in a `CHALLENGE_REPONSE` parameter. This parameter must be covered under the packet signature using the host identifier key, proving to the middlebox that the receiver is indeed in possession of the secret corresponding to public host identifier key.

Middlebox must not add the `CHALLENGE_REQUEST` parameters to the the `I1` packets as this would expose the Responder to DoS attacks. So, it should allow the `I1` packet to traverse unauthenticated. However, middlebox can choose to verify only the Inititator or the Responder or both. An outline of middlebox authentication integrated with HIP Base Exchange can be seen in Figure 2.8.

3

Related Work

In this chapter, we start with a brief overview of the existing approaches for signaling of end-point information to the middleboxes. In the second section, we look into the privacy concerns with cryptographic identities and the mechanisms to mitigate such problems e.g. Blinding for HIP [41] and SPSL for SSL/TLS [36]. Lastly, we discuss the principle of "minimal disclosure" and "selective disclosure" of end-point information along with the existing approaches for "minimal credential disclosure" in any negotiation.

3.1 Existing approaches for Signaling

In the Introduction, we have identified the conflict of interest between the middleboxes and the end-points. On one hand information in the signaling channels aids the functionality of middleboxes, on the other hand the end-points do not want to give away the sensitive information to untrusted entities. Before we discuss the identity privacy concerns for end-hosts, we would like to discuss the previous approaches who have identified that the information in the signaling path is good for middleboxes.

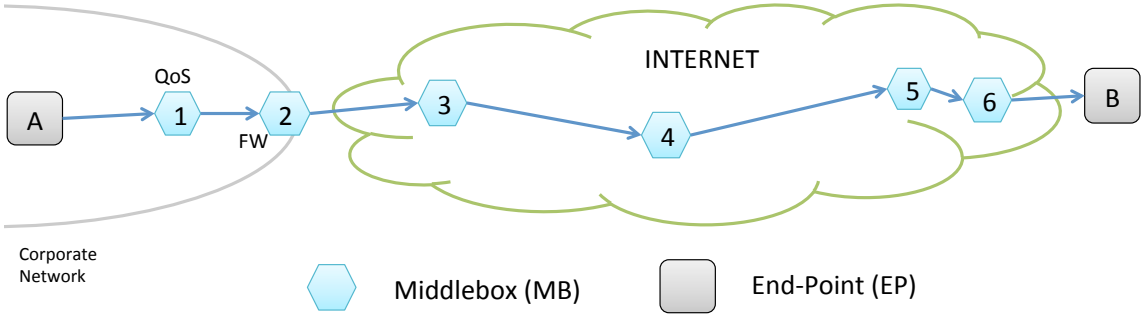

Figure 3.1 Network Scenario

Consider a typical network scenario in the Figure 3.1. The host 'A' wants to connect to another host 'B'. Albeit, there are many on-path middleboxes, we would like to focus on the first two. Host 'A' lies inside the corporate network. The middlebox '1' is a Quality of Service box and the middlebox '2' is a Corporate firewall. Suppose that the QoS box is configured to permit certain users to have additional network privileges like more bandwidth and the Firewall is configured to allow traffic from only certain applications. It is impossible for the QoS box to determine the user without a dedicated user-authentication mechanism in place before the connection is setup. The Firewall would have to employ costly deep packet inspection techniques to determine the application, which is still not reliable and may result in blocking traffic that should have been accepted. Hence, the addition of the user and application information in the signaling path eases the job of middlebox, makes them more reliable. Furthermore, the information about the application, the user or the host must be verifiable for the middleboxes to completely trust them. There have been previous works, e.g. NUTSS, Pedigree, SEAMS which have proven how information on the signaling path can be advantageous.

3.1.1 NUTSS

NUTSS was introduced Guha et. al. in [13]. NUTSS is a network architecture that uses overlay signaling before setting up the data channel. The overlay signaling could provide the middleboxes with signals which would help them discover the intent of the connection. The overlay channel works on *SIP* and the data channel works on the popular *TCP/IP*. NUTSS stands for: NATs which extend IP address space, URI for stable end-to-end addressing, Tunnels that work through NATs, SIP for end-to-end signaling using URIs and STUNT to establish direct IP connectivity through NATs. NUTSS presents an architecture to tackle the End-Middle-End (EME) naming and addressing problem [13].

NUTSS Connection Establishment Simpilfied: An initiating host wishes to establish a connection with a responding host. The initiating host transmits a message containing source and destination name and application name. All these names are based on URIs and hence can be used as a base for routing. With these URI addresses, the signal traverses through *off-path* policy aware boxes (*overlay signaling*), which authenticate the identities (URI names) and make a decision to allow or block the connection. If the connection is allowed, *ephemeral* addresses, ports and firewall-traversal tokens are sent to both the end-points. The information received front the off-path signaling is used to perform on-path signaling and traverse the data channel through the firewall with the help of tokens received.

The usage of stable, URI based unique identifiers allows NUTSS to separate identification from network location [13], which allows it to support network mobility and multi-homing.

NUTSS has strong requirements for a name-based signaling approach. The popular way to bind names with addresses, i.e., DNS suffer from inherent weaknesses. DNS is not aware of the entity making a DNS query and typically unaware of the purpose of the query [13]. Moreover, DNS reveals information about the physical location of an entity making it unsuitable for NUTSS [13]. NUTSS uses SIP for name-based

routing since it can carry the identity of both the end-points which can also be authenticated. The inherent properties of SIP like mobility, stable nomenclature of end-points and users, discoverability make it ideal for NUTSS to be used as an overlay signaling.

3.1.1.1 NUTSS Architecture

In NUTSS, an end-point can be a host, a user, an application or a service. A typical NUTSS end-point is a 3-tuple (*user, domain, service*). The *user* is a user-friendly but not a globally unique name (e.g. bob). The *domain* is a globally unique, hierarchical DNS name (e.g. rwth-aachen.de) and the *service* is a globally-unique, user-friendly identifier for the service provided by the end-point (e.g. ftpd for FTP-server). The names are long-term stable and location-independent.

Whenever an end-point wants to establish a connection with another end-point, it opens a *NUTSS Socket* based on the names (end-point identifiers) and not IP addresses. This triggers a signaling exchange which authenticates the identifiers and ends up establishing a 5-tuple(source and destination IP address, source and destination ports and IP protocol) data flow channel through on-path middleboxes. Along with the establishment of data channel, it also provides the end-points with necessary authorization tokens to traverse the middleboxes doing access control.

A NUTSS architecture has two important components: *P-boxes* (policy-box) and *M-boxes* (middlebox). These can be deployed in the network as well as the end hosts. *P-boxes* form an overlay on which name-routed signaling takes place between end-to-end. Data flows do not traverse the *P-boxes* but do traverse the IP path where *M-boxes* can be deployed. Also, the signaling with *P-boxes* may result in steering the data flow through *M-boxes* (for example an anonymizer). In short, the P-boxes take decisions based on the richly-named identifiers whether to allow or block them, steer them through specific *M-boxes* or indicate an encrypted data flow. The function of *M-boxes* is to enforce these policy decisions.

Signaling based on rich names (not IP address) through the *P-boxes* are hence called named-routing and the data path routing with IP addresses through the *M-boxes* is called *address-routing*. The architecture is depended on a bi-directional coupling of *name-routed* and *address-routed* signaling. Hence, an overlay signaling provides the end-points with necessary authorization tokens to traverse the data path through *M-boxes* and in case an unapproved flow is received by a *M-box* it redirects it to appropriate *P-box* from which an authorization token can be received.

The authentication of identifiers on the name-routed path that is used by end-point for overlay signaling with the help of P-boxes can be run over standard authentication mechanisms, for example public key cryptography or challenge response method. Either way, NUTSS does not incorporate mechanisms for establishing trust. Trust can be established with the help of reputation based mechanisms like “webs-of-trust” or trusted hardware etc.

The authorization token is point of serious concern to prevent against replay attacks. An eavesdropper can quietly listen to an ongoing overlay signaling and save the authorization tokens for replay attacks or unauthorized access at a later point of time. In order to deal with this problem, A P-box never sends the authorization

tokens in plain-text. It produces three encrypted copies of the authorization token: two of them are sent to the end-points and one is sent to the M-box. The end-point signs the messages on the data path with the authorization token and includes the encrypted copy of M-box in the signature. The M-box verifies the end-points signature before allowing the data flow.

3.1.2 PEDIGREE

Pedigree was introduced by A. Ramachandran et. al. in [38]. The authors argue that the traffic classification based on the coarse identifiers like IP headers is not reliable. Hence, Pedigree allows the traffic classification based on *provenance* and *privileges*. Furthermore, it also allows the network operators to have policy rules based on the *container* (a persistent entity) from which the traffic originated and the inputs to the process that governed the generation of traffic "tainted set". The motivation behind Pedigree is to allow the network operators (middleboxes) to make their policy decision based on more reliable and fine granular information about the traffic.

There are two major components of Pedigree: a *tagger* and an *arbiter* [38]. The tagger is a trusted module residing within the host and tags the outgoing traffic with an identification ("container ID") and the information about the processes that initiated the traffic ("tainted sets"). The function of the arbiter is to take policy decisions based on these tags. The arbiter is another truster module but resides on a network element. These network elements (middleboxes) can upgrade a traffic (e.g. higher QoS) or downgrade a traffic (e.g. drop) based on these tags.

Tags are dependent on the host container, i.e., process, process groups, virtual machine container, which initiated the traffic. It expresses the specific properties of the container, for example, special privileges of the container or the effect that other processes had on the container. The "container ID" maps the coarse grained information about the process and the "tainted set" identifies all the other processes and resources that have interacted with the container. The "container ID" is deterministic and unique but may or may not be cryptographically verifiable. Pedigree provides *provenance* of the traffic by annotating them with tags, essentially indicating the process that generated the traffic and the effect of the interaction of other processes with it. These tags can also be used for audit trail on the traffic by network elements (middleboxes) based on previous interactions [38].

When an application initiates a connection, the tagger running on the background adds a tag to traffic generated with *context information*, for example, about the other processes that affected this process (e.g. parent process, a separate module). The network element (arbiter) exhibiting middlebox functionalities verifies the context based on the provenance of the tag and provisions its services based on a policy satisfied by the tag.

Securing Transmission of Provenance Information: When a new connection is initiated, the first packet contains the full "tag" and a signature covering the contents of the tag. Additionally, the first element of a re-initializable hash chain is also added [38]. All subsequent packets contain the hash of the full tag and the next element of

the hash-chain. Therefore, every packet is bound to its predecessor and effectively to the first packet.

The payload of the packet is however not bound to the tag of the packet. Hence, a malicious adversary can easily eavesdrop on an active traffic flow and replay the same connection with the same tag and the hash-chain elements but with different payload. The network elements cannot identify any foul play since the signature on the initial tag does not cover any freshness information. Hence, Pedigree is limited to a network infrastructure where all entities are equally trusted to not play foul by modifying tags or reusing tags.

The "container ID" and other such informations in the tag are not cryptographically verifiable [38]. The authors indicate that these identities should not be considered while authenticating a traffic flow. However, they also argue that replay attacks become impossible if all the entities employ a trusted tagger, but this requirement on the ubiquitous deployment of trusted taggers limits its applicability only to tightly controlled networks (e.g. enterprise network). The author suggests encrypting the tags with the public key of the arbiter for additionally security, but this requires that the same private key is installed on all the other arbiters. This again is possible only in a tightly controlled scenarios.

3.1.3 SEAMS: Signaling Layer for End-host Assisted Middlebox Services

SEAMS provides a signaling layer that enables the dynamic negotiation of the required end-point information and allows the information provided by the end-point to be verifiable and accountable.

3.1.3.1 SEAMS protocol overview

SEAMS is designed as an extension of Host Identity Protocol [19]. This extends the potential benefits of Host Identity Protocol like verifiable identities and verifiability of signaled information in the signed message exchange. SEAMS comes to operation as soon as a new application triggers a HIP handshake between the two end-hosts. For example, in the Figure 3.1, as soon as connection attempt is made by the application running on host 'A' to host 'B', it is intercepted by the SEAMS signaling layer. The signaling is handled in a three way message exchange.

The connection request coming from the host 'A' is intercepted by a middlebox. The middlebox inspects the first message origination from the connection request, looks up against its policy if the connection is permitted. There is no information in the signaling channel until and unless the middlebox specifically requests for it. After verification with the policy, the middlebox might require additional context information from the end-point. Here, the middlebox would be looking at the context required from the Responder 'B' since the signaling flow for the first message is from the Initiator 'A' to Responder 'B'. The middlebox would then add the *request items* from the end-point in a *middlebox context request* [19]. For the sake of freshness and identification of this particular request, the middlebox adds a unique identifier, *request ID*, to the request.

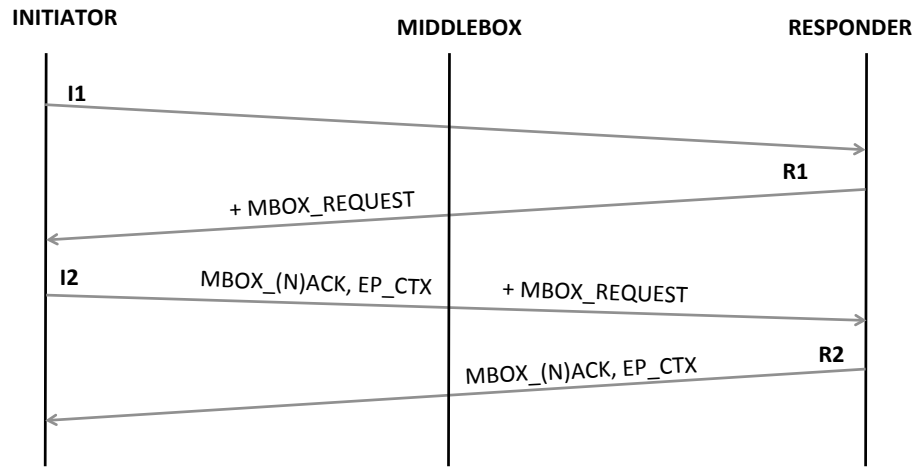

Figure 3.2 SEAMS Handshake

As soon as the *middlebox context request* is received by the Responder, it gathers its context information. The end-point might also choose to verify with its policy if *context information* should be signaled or not. If the policy permits the Responder acknowledges the request. Additionally it also adds the context information suggested by the *request values* [19]. If the policy does not permit, a negative acknowledgement is added instead notifying the to *requestor* middlebox with the reason for the decline. Either way, the Responder replays back the *request ID* with the acknowledgement. The middlebox on receiving the response, verifies the acknowledgement (with the echoed *request ID*) and checks the information replied against its policy. If policy check succeeds the middlebox services are provisioned for the Responder. A signature on the response packets protects the integrity of the message and makes all the context information in the response verifiable. After provisioning of the services by the middleboxes, it looks up the policy for the Initiator side and requests for the context information required. The three way signaling handshake is symmetric for the context information negotiation between the middleboxes and the Responder [19].

Verifiability of End-Point Information : The primary requirement of SEAMS is the verifiability of the identity of the end-points and the verifiability of context information signaled by the end-points at the middleboxes. SEAMS depends on public key cryptography for this purpose [19]. Each entity be it a host or a user possesses a cryptographic identity as proposed by HIP [30]. The hosts are assigned a public key pair by the administrator and the users may generate their own public key pair. SEAMS permits the middleboxes to request for host and user identities whenever it wishes to authenticate the end-points. The end-points can then add the requested identities to the signaling layer, and use the identities to sign the message. Middleboxes can verify the signature and therefore authenticate the end-point (host or user) identity. Additionally, the signature may also cover the context information. Hence, the signature verification can associate the signaled context to the identity with which the signature was computed. Apart from integrity protection, the signaling of cryptographic identities also provides non-repudiation of the communication. Unlike host or user context information, the applications do not have any identifier. However, an application that was executed by a user or an application service running on a host can be bound with the user or host identity respectively. An appli-

cation specifically runs with the privileges of the user and is continuously monitored by the operating system (memory allocation and the files accessed by them). This makes it impossible for the application to hide a secret from the operating system on which it is running and the user who executed the application. So, a middlebox can extend its trust from the verified user or host identity to the application.

All the three approaches discussed here have enriched the middlebox functionality with additional context information from the end-points. NUTSS provides named identifiers to user, host or application so that they used in overlay for policy decisions. But these identities are not cryptographically verifiable. Also, NUTSS does not allow for distinguishing the host, the user or the application context, instead all the context information is signaled to the overlay channel. Pedigree, on the other hand neither provides cryptographically verifiable identifiers nor does it provide the verifiability of context information. Hence, it is vulnerable to replay attacks and forgery. SEAMS distinguished between user, host or application context and allows the middlebox to negotiate for the information. On top of that, it ensures the verifiability of end-point identity and the verifiability of signaled information. Although, the solution presented by SEAMS, seems to be complete. It is still insufficient with respect to the privacy concerns of the end-points. The privacy issues are introduced by the extensive use of public key cryptography. Also, signaling information in plain text does not limit the visibility of the context information to only those who need them. In the next section, we discuss the privacy concerns of public key cryptography and present few approaches to mitigate them.

3.2 Privacy Concerns

In Section 2.2, we discussed how public key cryptography can be used to secure any communication channel and protect the information being shared over that channel. Most of the security protocols including TLS, IPSec and SSH have an underlying dependency on public key cryptography, as we saw in Section 2.3. The widespread use of X.509 public key certificates and sharing them in plain-text exposes identity privacy problems of public key cryptography. These vulnerabilities stem up from the assumption that a digital certificate without the availability of the corresponding private key, can not be used for impersonating the identity of the holder of the certificate. In this Section we discuss how the usage of X.509v3 certificates create an identity privacy problem and present few approaches to deal with identity privacy problem (e.g. BLIND).

3.2.1 Privacy Concerns with Cryptographic Identities

The digital “certificates” quite popular within the PKI are in true sense “identity certificates”. Ellison defines a certificate to be digitally signed, structured message which delegates an attribute of some form to a public key [11]. He also defines an identity certificate to be a certificate that binds the name and other information concerning an entity to its public key [11]. Two common examples of identity certificates are X.509 certificates and PGP.

X.509 certificates as described in Section 2.2.3 contain an information field called "Distinguished Name" which is a unique name following the X.500 global database design. For the sake of storage and management these names are organized hierarchically, forming the X.500 namespace. So, a trusted identity certificate would successfully bind a public key to the identity pointed by the Distinguished Name (DN). For this binding to hold, there should be a secure mapping from the namespace of the issuer to that of the verifier. Apart from the Distinguished Name (DN), an X.509 certificate consists of *attributes* also called *extensions*. These extensions may contain additional information pertaining to the identity of the holder or some other information related to the certificate itself. Similar to the X.509 certificate, PGP allows a binding between a public key and a user id. The user id is a combination of a common name and an email address. The email address also acts as unique name from a globally deployed name space.

The hierarchical nomenclature followed in both of the above cases reveal information about the issuer as well as the holder of the certificate which were neither necessary nor requested in a transaction. The ubiquitous presence of extensions in the X.509 certificates makes this additional information available which may not have been requested [36]. Furthermore, the secure and verifiable binding of the DN to the public key links together all the transactions where the public key has been used. This unintentional revelation of information is an *identity privacy problem* and is manifested by fact that public key is designed to be widely available.

Next, we discuss an approach that protects the cryptographic identifiers by blinding them.

3.2.1.1 An Extension of HIP Base Exchange to support Identity Privacy using BLIND

In the previous two subsections, we showed how the weaknesses of Public Key Infrastructure exposed the identity privacy problems of the end-points. A common approach to deal with such vulnerabilities is *blinding* of cryptographic identifiers [40]. Blinding the public key information (cryptographic identifier) with a random nonce each time a cryptographic identity is signaled protects the public key information from being revealed unintentionally. The identity is sent only after encrypting it to the concerned peer [40]. We would be using Host Identity Protocol extensively to support our design as discussed in Chapter 5. Hence, for the sake of this work, we present a *Blinded HIP Base Exchange* which borrows heavily from BLIND [40].

In Figure 3.2, we show how BLIND can be used to extend the HIP Base Exchange to support identity privacy [41]. The initiator generates the blinded HITs for both parties using a random nonce N. The blinded HIT of the responder, B-HIT-R can be generated by taking a SHA1 hash over the concatenation of HIT-R and N, B-HIT-R = SHA1(N ‖ HIT-R) [41]. Similarly, B-HIT-I can be computed using the nonce N and the initiator's HIT, HIT-I [41].

Instead of the original I1, the I1 packet now contains B-HIT-I, B-HIT-R and N. Upon receiving the I1 packet, the responder computes the SHA1 hash from the concatenation of N and HIT-R and verifies it with the B-HIT-R received in the I1 packet.

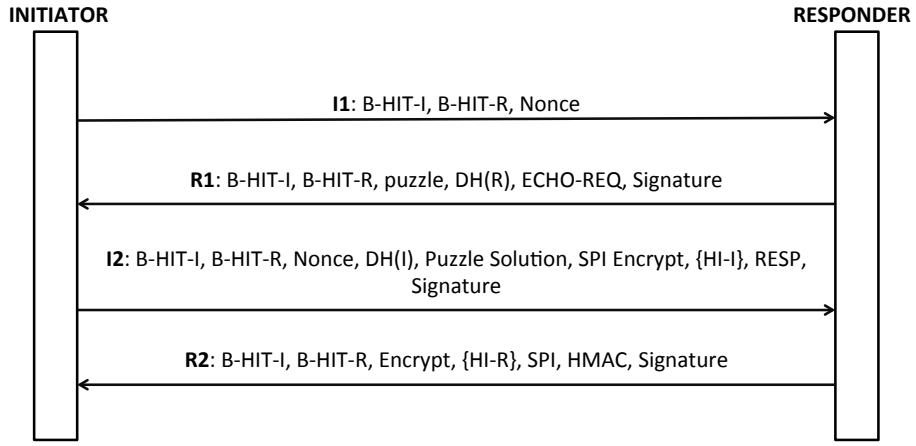

Figure 3.3 HIP Extension to support Blinding

If the responder has multiple HITs, it needs to perform this action multiple times. If there is no match for B-HIT-R, the packet is discarded. If a hash value matches to B-HIT-R, then the responder sends the pre-created R1 of the corresponding HI to the initiator. The responder may also choose to maintain a mapping from the pseudonym to the association HI, although it is not recommended to prevent DoS attacks [41].

On receiving R1, the initiator may verify the signature on the packet is already possesses the valid HI of the responder. If not then the initiator is not in a position to verify the signature of R1 until it receives the H1 from the responder. Either way, the initiator generates a symmetric key, KDH, using the Diffie-Hellman algorithm. It also adopts the key to compute the keying material from the HIT-R and B-HIT-I:

$$\begin{aligned} KEY_1 &= SHA1(KDH \parallel HIT-R \parallel B-HIT-I \parallel 1) \\ &..... \\ &..... \\ KEY_n &= SHA1(KDH \parallel HIT-R \parallel B-HIT-I \parallel n) \\ KEY_{MATERIAL} &= KEY_1 \oplus ... \oplus KEY_n \end{aligned}$$

The HI of the initiator is encrypted using the symmetric key generated from the keying material. The encrypted information is sent via I2 to the responder. The initiators also sends the pseudonym received in R1 and the nonce sent in the I1. On receiving the I2 packet the responds generates the keying material similarly as done by the initiator and extracts the symmetric key from the keying material. It decrypts the HI and HIT of the initiator using the symmetric key and verifies the correctness of B-HIT-I. As the responder had already received the HI of the initiator, it can proceed with the signature verification on I2.

Finally, the responder encrypts its own HI using the symmetric key generated from I2 and sends the encrypted HI to the initiator over R2. The initiator decrypts the encrypted HI and verifies the correctness of B-HIT-R and the validity of R1 packet. With this, the whole BEX completes successfully.

3.2.2 Privacy Concerns for Cryptographically Bound Information

In Section 3.1.3, we showed how SEAMS established verifiability of end-point information, by cryptographically binding it to the end-point identity. This is a privacy concern because the association of information and identity is revealed to more entities than required. In SEAMS, the verifiability could be checked by any on-path network entity which is not desirable. In the next subsection, we discuss an approach that helps the end-point to make information cryptographically verifiable but without revealing its identity, i.e., SPSL [36].

3.2.2.1 Secure and Private Socket Layer Protocol (SPSL)

The SPSL extension to SSL propose an extension of the certificate structure so as to allow partial disclosures of information. The certificates are still integrity verifiable, but the user is given a choice of revealing only certain extensions. However, the holder of the certificate its modify the contents of the extension at any point.

Basic Anonymous Identification Protocol: Consider a scenario, where a user 'U' has to prove his identity to a server 'S'. In order to prove his identity, the user proves to the server that it is in possesses the secret key corresponding to one of the public key the server trusts, but without revealing which one. So, for authentication 'S' publishes a list of 'l' certificates (public key) which it trusts and sends a message 'm' to the user. The server requests for a signature of 'l' messages, i.e, one for each public key such that $\oplus$ of the 'l' messages is equal to the original message 'm'. If the user does not possess a private key corresponding to any of public keys in the list of certificate certificates, it is impossible for the user to generate the signature of 'l' messages whose $\oplus$ makes up for the message 'm'. This way the user is able to prove to the server, that it knows a secret (private) key corresponding to at least one of the public keys without revealing its identity. An overview of the protocol can be seen in Figure 3.4 [36].

Common Input: l RSA public keys: $(N_1, e_1), \cdots, (N_l, e_l)$ of length n.
User's Private Input: (i, d_i) such that $d_i \cdot e_i \equiv 1 \pmod{\varphi(N_i)}$.

Instructions for Manager and User:

M.1 Pick a random message $m \in \{0,1\}^n$. Send m to U.

U.1 For each $1 \le j \le l$ and $j \ne i$

U.1.1 Pick a random signature $s_j \in \{0,1\}^n$.

U.1.2 Compute $m_j \equiv s_j^{e_j} \pmod{N_j}$

U.2 Compute $m_i = m \oplus m_1 \oplus \cdots m_{i-1} \oplus m_{i+1} \oplus \cdots m_l$. Compute $s_i \equiv m_i^{d_i} \pmod{N_i}$.
Send $(m_1, \cdots, m_l, s_1, \cdots, s_l)$ to M.

M.2 Verify that $m_1 \oplus m_2 \oplus \cdots \oplus m_{l-1} \oplus m_l = m$. For $i = 1, \cdots, l$ if $s_i^{e_i} \ne m_i$ then ABORT

Figure 3.4 The basic anonymous identification protocol

SPSL Protocol Overview: In SSL/TLS Handshake the two parties perform negotiation for a cipher suite by making the client send a *ClientHello* and the server send a *ServerHello*. Additionally, if Server can request the client to authenticate

with the server by sending a *CertificateRequest* message. The client then responds to this request with its certificate. The server validates and verifies the certificate and request the client to prove that it holds the corresponding secret key. This is important since the certificates are publicly available and just holding the certificate does not prove the identity. The client proves to the server that it possesses the secret key, by signing the hash of the transcripts of the entire conversation so far. Since, the transcript of the entire conversation is random and not fixed, the signature does not reveal any information about the secret key.

SPSL proposes slight modification to the way the client proves its knowledge of the private key corresponding to the certificate sent [36]. In this case, the server publishes a list of 'l' certificates and the client has to prove the knowledge of at least one private key corresponding to the one of the 'l' public keys. This list of certificates is also called the *mask* as it masks the actual certificate of the client. The user then proves his knowledge of the secret key using the *Anonymous Identification Protocol* discussed earlier. This process can be followed either way, i.e, the client masking the server certificate and the server masking the client certificate.

We can observe from the SPSL protocol that the client can ensure verifiability of the message (hash of the transcript of the entire conversation during SSL handshake) without actually revealing its identity. And in this case server has successfully masked the identity of the client from other network entities. Thus, the information can be cryptographically verifiable without revealing the client's identity and only designated entities can perform the verification.

After looking at the various mechanisms to deal with privacy concerns of cryptographically verifiable identity and the cryptographic verifiable information, we would like to highlight that the solution to hide, mask or blind the identifiers is not enough. Information provided to the signaling channel willingly without consideration is another source of privacy concerns [27]. In the next section we discuss the approaches which allow the end-points to reveal the least of the information possible without breaking the verifiability.

3.3 Privacy Preserving mechanism for Signaling Protocols

Signaling protocols entail within the provision for supplying end-point information to the signaling channel. Some approaches only support signaling all of the information (NUTSS) [13], while others have provisions for negotiating for the information required (SEAMS) [19]. In spite of the existence of mentioned approaches, they are still not able to prove verifiability selectively, i.e., an entity can verify only the information that it has negotiated for. The current solutions provide for verifiability on the entire message and not chunks of it. Hence, the end-point has to put all the information together in order for it to be covered with the signature. This is another source of privacy concern because the end-point bundles all the information together which can be received and verified by every entity on the signaling channel.

A privacy sensitive negotiation should obey both "minimal disclosure" and "selective disclosure". Bauer et. al. define the principle of "*minimal disclosure*", i.e., an

entity requesting for private information should be provided the minimum amount of information that is necessary to authorize the operation [1]. Similarly, "*selective disclosure*" of information means that the information can be verified even when parts of it are hidden or removed [27]. Next, we provide existing approaches to allow for selectively disclosing the minimum amount of information [35, 18].

3.3.1 Minimum Credential Disclosure

This approach supports *minimal credential disclosure* technique which guarantees that only the attributes necessary for the completion of the transaction are disclosed. A message or document is split into smaller sets and the user (end-point) makes micro-claims over each of these sets (verifiability). In a digital transaction (e.g. signaling) the user reveals only the information that has been negotiated for and some additional information that is necessary for the verifiability of the information.

3.3.1.1 Overview

The approach for *minimum credential disclosure* is based on the *Merkle hash trees*. It is similar to "redactable signatures" which can be verified successfully even when parts of the message that has been signed are hidden (redacted). Any document or message can be broken down into micro-claims (claim for verifiability of a particular portion). A *credential* comprises of two parts: *public* and *private*. The public part is the certificate, which contains information about the issuer, certificate chain for the issuer, the validity of the certificate, the type of the certificate, the user's public key and a signature using the public key on the root node of the *Merkle Hash Tree*. The private part of the credential is the private key and the Merkle hash tree.

The leaves of the Merkle hash tree are the hash of the micro-claims by the user. A user can make assertions on a bunch of micro-claims by revealing to the verifier, the values of the micro-claims, their respective position in the Merkle Hash Tree, some additional hash tree intermediate nodes necessary to rebuild the value of the root of the tree and the certificate. The verifier can reconstruct the root value of the hash tree with the help of the information as provided and verify the signature on the root with the one in the certificate. The position of the leafs of the hash tree (micro-claims which are asserted) and the intermediate nodes from the hash tree necessary for the reconstruction of the root value of the hash tree is also called the *authentication path*.

Consider the following example as shown in Figure 3.5. The user needs to produce verifiable information about the micro-claims (subsets of information), C_1 and C_7. In order to prove the verifiability of C1, the authentication path should contain $\{H(C_7)$, $H(C_8)$, N_3, N_4, N_5, N_6 and $R\}$. Similarly the authentication path for C_7 is $\{H(C_1)$, $H(C_2)$, N_1, N_2, N_5, N_6 and $R\}$. The verifier reconstructs the root value of the hash tree as follows,

- computes the hash of $H(C_1) \mid\mid H(C_2)$ to get N_1
- computes the hash of $N_1 \mid\mid N_2$ to get N_5

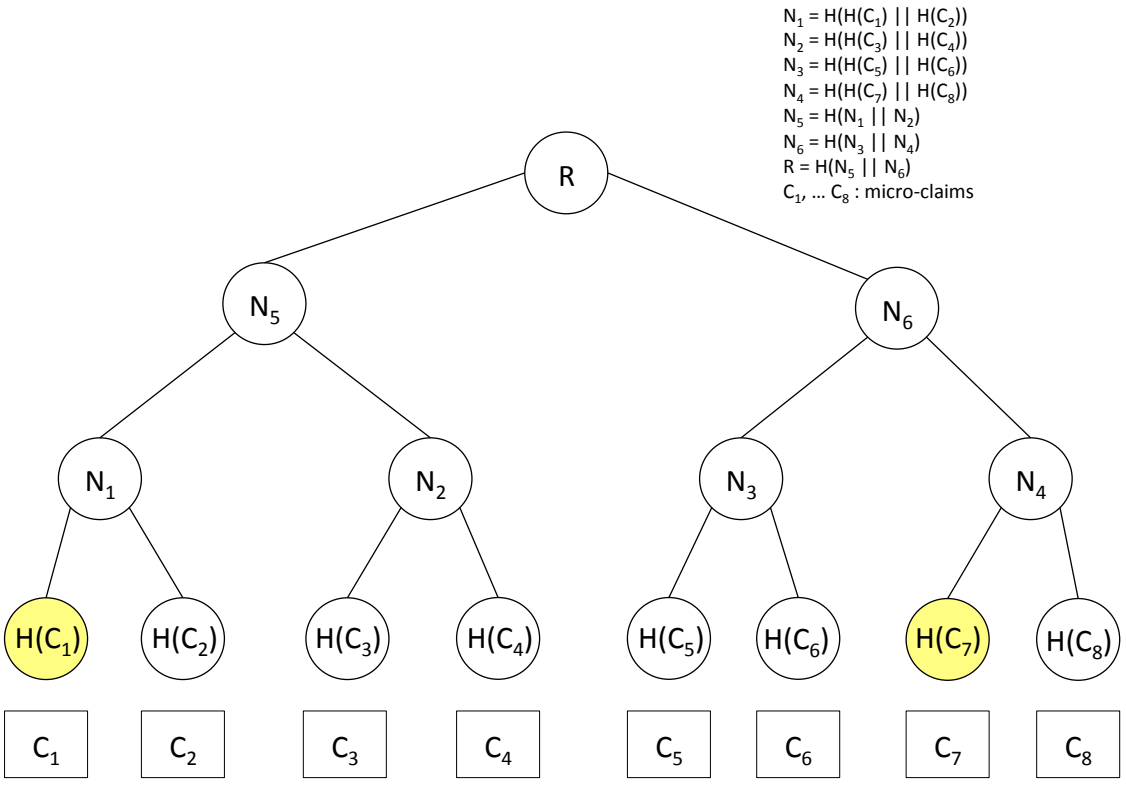

Figure 3.5 Merkle Hash Tree

- computes the root value R as the hash of $N_5 \mid\mid N_6$

Similarly, root value of the hash tree can also be obtained by following the authentication path for C_7. This approach obeys both the principles of "minimal disclosure" and "selective disclosure". It emphasizes the revelation of the minimum amount of information (minimal disclosure) and provides verifiability even when not all of the information is visible (selective disclosure).

In this chapter, we have highlighted the privacy concerns for end-points and the existing approaches which have dealt with the privacy problems. In the next chapter, we formulate a problem statement and try to incorporate the learnings from the existing approaches into designing an architecture that can mitigate the issues discussed above.

4

Problem Analysis

The concern for the privacy of end-point information and the ubiquitous presence of middleboxes have created a conflict of interest between the two entities. On one hand, the extensive usage of public key cryptography and their widespread use in security protocols have led to an identity-privacy problem. On the other hand, there is a huge gap in the information required by the middleboxes and the information available to them. Previous approaches, NUTSS and SEAMS, have been instrumental in bridging this gap. However, the existing solutions to deal with privacy concerns of public key cryptography (e.g. BLIND and SPSL) do not play well with the middlebox functionalities. This chapter focuses on analyzing the conflict between the interests of end-points and the middleboxes in context of the implications of privacy on the relationship between the two. Lastly, defining the guidelines for a privacy-sensitive stable system.

4.1 The Implications of Privacy on Revelation of Information

In the words of Stewart Baker, chief counsel for the American National Security Agency (NSA) [27],

> *"The biggest threats to our privacy in a digital world come not from what we keep secret but from what we reveal willingly. We lose privacy in a digital world because it becomes cheap and easy to collate and transmit data ..... Encryption can't protect you from the misuse of data you surrendered willingly."*

A privacy sensitive system works on the basis of identity relations. These identity relations define the manner in which information can be shared with each other. Private entities which have vested interests in protecting their sensitive information

and would like to have *uni-directional* identity relationships [4]. A *uni-directional* identity relation between a private entity 'A' with another entity 'B' is specific to this relation and the identifier used in this case can't be used anywhere else [4]. Another relation between the private entity 'A' and some other entity 'C' is entirely unique and new set of identifiers have to be created to be used in this case. On the contrary, public entities (beacons of a WiFi router) would like themselves to be discovered and wait for other entities to make use of their service. The identifier to be used by such public entities should be trusted by all and the identifier used by the public entity in a relationship with a private entity can be the same or even related to the identifier used in another identity relation. Hence, the public entities should possess *omni-directional* identifiers [4].

The communication channels in modern day network infrastructures are not limited to point-to-point communications (between private entities), but in fact are dependent on and exposed to a variety of middlebox services (public entities). The role of middleboxes (public entity) requires them to possess "omni-directional" identifiers [4]. On the contrary, the end-points (private entity) need to have "uni-directional" identifiers [4]. This is essential for the discovery of a trusted service while preventing unnecessary revelation of private attributes [27].

4.2 Network Scenario

Before we start with a detailed discussion on the challenges faced by existing approaches, we would start with a typical network scenario. In a modern day network, there are variety of on-path and off-path devices directly and indirectly affecting the signaling between two parties. A successful architecture should be able to work in a network infrastructure with varying *trust levels* (with public entites) and yet be *privacy preserving* (protecting the sensitive end-point information).

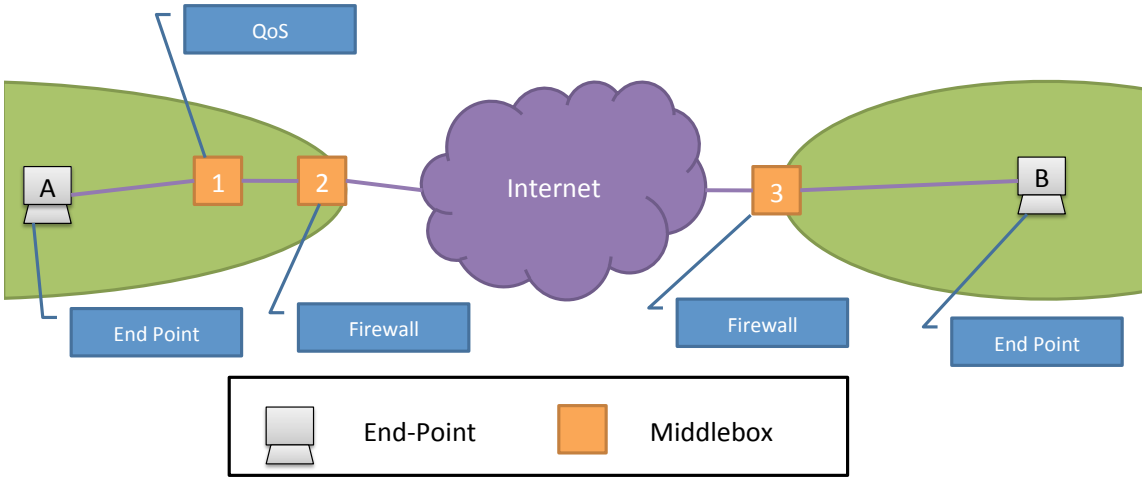

Figure 4.1 A typical network model with a variety of on-path devices

In Figure 4.1, there is a typical network scenario in which an end-point 'A' within a corporate network tries to access a known web-server 'B' somewhere outside of its network. The end-point 'A' should be able to easily trust the QoS box '$MBOX_1$', the

edge Firewall '$MBOX_2$' since they are within the same corporate network. However, it will be very difficult for the end-point to trust some random device in the Internet. But as the end-point knows specifically which other party it wants to connect to, it can trust the end-point and it's firewall at the other end of the communication channel.

Moreover, the QoS box '$MBOX_1$' may have been configured to grant special network privileges to a group of users. Similarly, the edge Firewall '$MBOX_2$' will only allow connections originating from a trusted application (e.g. Mail Client). Without an off-path signaling mechanism, it would be impossible for '$MBOX_1$' to determine the user identity associated with the end-point 'A'. Also, the '$MBOX_2$' would have to go for costly deep packet inspection schemes in order to determine the application which initiated the connection.

From the above example, we can outline two diverging lines of interest. On one hand, the end-points are very sensitive about their information and would like to trust the network entities before providing them with private information. On the other hand, the middleboxes benefit immensely from the information availability on the signaling path. Previous approaches, like NUTSS and SEAMS have provided mechanisms to make end-point information available to the middlebox [19, 13]. SEAMS also allowed to verify the identity of the end-points (establish trust) and to establish verifiability of end-point information before it can trust the information received [19].

4.2.1 The Middlebox Perspective

In spite of the effectiveness of the previously mentioned approaches (e.g. SEAMS) [19], there are privacy concerns due to the usage of cryptographically verifiable identity and cryptographically verifiable information in these schemes. In section 3.2 we discussed the privacy aspects of public key cryptography and showed how the widespread usage of X.509 certificates reveal unnecessary information about the identity of the holder of the certificate. Moreover, a secure and verifiable binding of the identity of the holder of the certificate to the public key contained in the certificate links all the transactions of that entity wherever that public key has been used. The linking of public key identifier to the transaction where it has been used allows an eavesdropper to track the activity of the holder of the certificate and collection information about the frequency of usage. These privacy concerns can be implicated to most of the popular security protocols like SSL/TLS, HIP etc.

Not only has the public key cryptography been widely deployed, but it is also strongly trusted. It has been proven to be secure enough for protecting any data and provides other desirable cryptographic properties like integrity protection, non-repudiation. Hence, public key cryptography is critical to any system that needs *"verifiability"* of information. In Section 3.2, we explored few of the existing solutions, BLIND and SPSL, and how they mitigated privacy concerns of cryptographically verifiable identity and cryptographically verifiable information respectively. But these approaches come at a cost for the middlebox functionality.

Blinding of end-point identifiers [40] does not play well with the middleboxes. Signaling layer protocol like SEAMS and NUTSS [19, 13], have allowed the middleboxes to leave policies based on coarse identifiers like IP addresses, packet headers and rely

on cryptographically verifiable identifiers for policy decisions. Blinding of these identifiers (for example, host identities in HIP BEX [41]) makes them unavailable to the middleboxes. So, the middleboxes can neither trust these blinded identities nor can take policy decisions based on the blinded identifiers. As a result, the middleboxes have to fallback to old methods of relying on the coarse information available in the network layer packets. Blinding in short, breaks the trust establishment mechanism and the policy model of the existing approaches (e.g. SEAMS and NUTSS).

Privacy concerns for cryptographically verifiable information were dealt with by allowing the end-points to establish verifiability without actually revealing its identity (e.g. in SPSL) [36]. Although it fulfills a key middlebox requirement (verifiability of end-point information), it does not fulfill the other requirement, i.e., verifiability of end-point identity. Without the knowledge of identity, the middlebox policies which are base cryptographic identifiers do not work.

4.3 The End-Point Perspective

The trust level that an end-point has with a network entity is very subjective. For example, in Figure 4.1, the end-point 'A' has strong trust levels with the devices in the corporate network and it can trust all the devices equally. On the contrary, the end-point 'A' can not trust anything in the Internet. But 'A' might be able to trust the other end of the network because of a previous trust establishment. So, we have varying trust levels within the same signaling path. We can represent these trust levels as shown in the Figure 4.2.

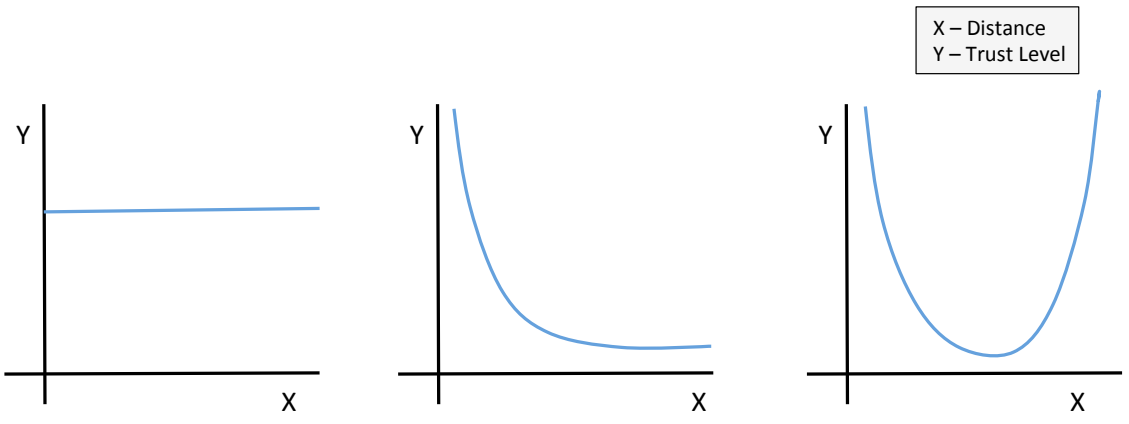

Figure 4.2 Trust levels in the Signaling Path

The first graph shows the trust levels within the corporate network. Every device has strong but equal trust levels. The graph in the middle, shows the trust declining as we move along the signaling channel (see Figure 3.1) from the corporate network to the Internet. The third graph gives a complete picture of the trust levels in the entire signaling path, starting from a highly trusted corporate network, to an untrusted Internet domain and finally on a trusted end of the communication path. These graphs are mere samples of the extensive privacy requirements that the end-points have. We have discussed few of the approaches to allow the end-points to go for a minimum disclosure of private information and yet preserve verifiability of the

information in Section 3.3. There are additional privacy constraints that need to be fulfilled in the varying trust scenario.

4.3.1 Minimal Disclosure of End-Point Information

With SEAMS, the middleboxes can negotiate for the information they require for the provisioning of their services [19]. The end-points on the other hand can choose to provide the information requested based on their local policy. We achieve the minimal disclosure of end-point information here, albeit partially. For example, consider the network scenario from the Figure 3.1, '$MBOX_1$' requires user information and the '$MBOX_2$' requests for application information. With SEAMS negotiation, the end-point 'A' would end up adding both user and application information in the same packet while both of them are verifiable anywhere in the signaling path (no selective disclosure). In this context, we should note that the '$MBOX_2$' does not need to know the user information and the '$MBOX_1$' does not need to know the application information. Since both the application and user information are signaled within the same packet, either of the middleboxes know more than what they negotiated for (no minimal disclosure). Also, even though the signaling of information is integrity protected and can't be forged, other on-path devices still have access to the information (since the signaling channel is not secured).

The "minimal disclosure" of information should permit an entity to receive only the information it negotiated for. The system should protect all the signaled information in a way that an entity that did not negotiate for a specific information should have no way to access it. Also, the system should allow the end-points to make "selective disclosure proofs".

4.3.2 Unlinkability of Signaled Information

In order to link an information to a specific end-point (user/host), a middlebox must be able to authenticate/verify that context information. The information signaled to a specific entity should only be verifiable at that entity. Neither should the information sent to one entity be correlated to another entity nor should an entity be able to link information meant for a third party to the third party's identity.

Signaling layer architectures like SEAMS [19], allow linkability of information to the identity of the end-points whenever it is used with integrated handshake. The integrated handshake allows all the on-path middelboxes to participate in the signaling at the same time. The limitation of such architecture lies in the fact that all the information has to be signaled in the same packet and the complete packet is signed and integrity protected with the cryptographic identifier of the end-point. Hence, the architecture can not choose to limit the linkability of information (to the end-point identity) at only specific middleboxes. Instead, the signaled information is so tightly linked with the identity of the end-point that anybody on the signaling path can obtain the corresponding public portion of the end-point identifier can verify and link the information to the identity provided. However, when a cascading handshake is used [19], the end-point provides the required information to all

the middleboxes one by one. Thus at each iteration, the information on the signaling channel can be linked to the end-point only by the middlebox with whom the handshake is performed.

4.4 Problem Statement

In the previous section, we discussed the challenges affecting the existing middlebox friendly solutions (e.g. NUTSS and SEAMS [19, 13]) and the additional constraints on the signaling channel (with varying trust levels) to protect the sensitive end-point information. In accordance with the implications of privacy, a *stable system* that can handle the information requirement and trustability of a discoverable middlebox service, and yet be privacy-preserving, must possess the following three properties [27]:

- **Verifiable** The information signaled by an end-point (private entity) should be verifiable at the middleboxes (public entity) i.e some way of checking it to be true.
- **Minimal** Middleboxes should be provided the least of the information they need to know. This is also referred to as the principle of *"minimal disclosure"*.
- **Unlinkable** In a multi-party signaling channel, information signaled to one of them (middleboxes or hosts) should not be linkable to another piece of information signaled to some other party.

We should note that any system satisfying all the three properties will automatically satisfy the principle of selective disclosure. In the next chapter, we discuss the design of our solution and incorporate some of the mechanisms shown in Section 3.2 and 3.3 to present an amicable solution that respects these three main characteristics of a privacy preserving stable system.

5

Design

In this chapter, we lay down the design goals for our approach and discuss our approach using various trust vs distance models. The architecture of our system, the motivation to use Host Identity Protocol (HIP) and finally the integration of our architecture with HIP.

5.1 Design Goals

Our design goals are a set of necessary characteristics that a system needs to possess to deal with the problems discussed in Chapter 4. Also, these guidelines are supposed to maintain compatibility with the existing network infrastructure and interoperability with existing protocols.

1. **Verifiability of End-Point Information**
   All informations provided by the end-points should be cryprographically verifiable only at the middleboxes who requested for the information. The information requested by one middlebox should not be verifiable at other middleboxes who have requested for some other information. So, a middlebox which was trusted to receive the information can prove non-repudiation but not any other middlebox.

2. **Minimal and Selective Disclosure of End-Point Information**
   The middleboxes should request for the minimal information required and the end-points should be able to respond with only the information that they deem necessary or conversely completely decline the request for information. The end-point should also be able to limit the disclosure of private information to a selective group of middleboxes. Furthermore, it is important that the end-point can make a selective proof of only a portion of information but not necessarily all of it.

3. **Unlinkability of End-point Information**
   Any network entity on it's own or with the help of another entity should not be able to link (associate identity with information) together different pieces of end-point information sent in the same packet or even different packets or sent at different times. The end-point information can be linked to the identity of the sender only at the middlebox to which it was trusted with. It should not linked with any of the previous transactions of the end-point or other transactions with different middleboxes.

4. **Trust between End-points and Middleboxes**
   The end-points should be able to trust the middleboxes before providing them with any private information. The design should allow the end-points to model their trust level with the middleboxes based on the network topology (e.g. corporate network) or authentication (using public key certificates).

5. **Performance**
   The architecture should provide both, mechanisms for less secure but lightweight symmetric encryption or more secure but computationally intensive asymmetric encryption. The choice for the methodology of protecting end-point information should depends on the trust levels of the end-point with various network entities.

5.2 Modeling Trust in End-to-Middle Signaling

We have already talked about the weaknesses of existing protocols and solutions and also identified the guidelines around which we have to build our architecture. We would now like to discuss the various factors governing the trust levels between various devices.

In Section 4.3, we emphasized that on any signaling channel there are a multitude of devices and each of them may have a different trust level with the end-points. There may be a variety of factors deciding the trust level. For example, a host within the local area network might be strongly trusted. Few of the factors that determine the trust level are the position of the on-path device on the signaling path, the identity of an on-path device and the functionality provided by the on-path device.

- **Requested Context**
  The context information requested by a middlebox governs the sensitivity of information. For example, a NAT does simple IP address translation and does not require any additional information except the publicly available IP address information. However, an Application-level Gateway might require specific information about the application being run at the end-host or the user who is running the application at the end-host. Depending on the type of information requested and the service provided by the middlebox the end-point can decide if the information is sensitive or not. An insensitive information can be readily signaled while an end-point might be selective about the entity with whom it will share sensitive information.

- **Position of the middlebox in signaling path**
  Position in the sense of a network infrastructure may mean different things. For example, in one of the cases a host might consider the devices which are located at a distance of few hops to be more trusted than a far away device located deep within the internet. The definition may also vary with the reference to the logical division of IP network. A device within it's own subnet are more trusted than the ones located on another subnet. So, there might be three possible position of any device; located on the same subnet, located on a known but different subnet, rest of the IP network. There may be more ways to look at this, but the basic idea governing the determination of position of a device, is the logical location of the device in the signaling channel.

- **Identity of a network entity**
  The identity of a network entity might be based on something as loose as IP address or as strong as cryptographically verifiable identifiers. A device with a known identity can always be trusted more than a middlebox which has been discovered for the first time. For example, the identity of the Mail Server may already be known or directly configured to the application accessing it. Either way, the knowledge of the identity of the respective firewall makes it be more trusted. Similarly, a device with which there has been identity sharing in a previous transaction can be continued to be trusted.

The above discussed trust levels are very important in a signaling layer architecture. These govern the policy engines running in the middleboxes and the end-points and which in turn determine the decisions being taken in a signaling layer handshake and service negotiation.

5.2.1 Use Case

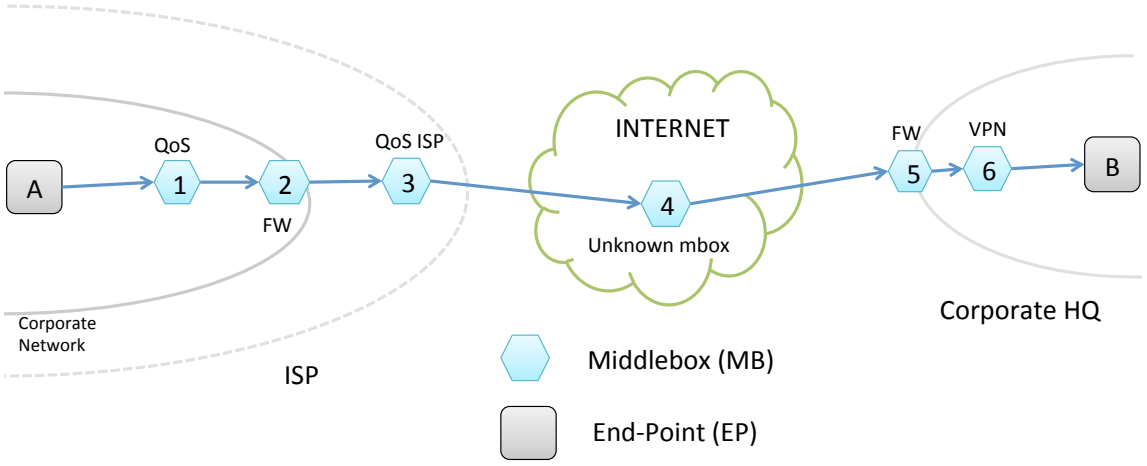

Figure 5.1 Use case scenario of a typical end-to-end communication setup

In Figure 5.1 we see a very common network scenario. Host 'A' in Corporate Network is trying to establish a secure communication channel with another host 'B' located in the Corporate Headquarters. As we can see from the Figure 5.1, the connection setup takes a route which has a variety of on-path devices. We can break this signaling path

into four major parts. The first part lies within the corporate network and has two middleboxes i.e a QoS box 1 and a corporate firewall 2 respectively. The corporate network in return may have been provided by the ISP itself (e.g. a small corporate office). The second portion of the channel contains another middlebox, possibly a QoS box 3 employed by the ISP. Beyond the ISP network lies the entire Internet (third part) and may have some unknown middleboxes 4 on the communication channel. At the other end of the communication channel lies the network of the Corporate Headquarters (fourth part). At the edge of this network lies the Corporate Firewall 5 and behind the firewall is a VPN 6. Correspondingly, the middleboxes have been numbered.

A network setup would require host 'A' to provide information about the application, user or host to the various on-path middleboxes. The QoS box 1 might be configured to provide better throughput for certain applications and prefer a privileged group of users. The corporate firewall 2 might have some policies based on higher level protocols and to protect the internal network from Denial of Service Attacks. The ISP QoS box 3 might have been entrusted to ensure a consistent throughput in the ISP network. Meanwhile, the Internet is home to numerous known and unknown on-path devices (middleboxes) 4. Similar to the previous firewall, the corporate firewall's 5 job is to restrict the incoming traffic based on certain policies and protect the network from attacks like DoS attacks. Behind the firewall lies the VPN which must authenticate the user and setup a secure IPSec tunnel to securely encapsulate the traffic between host 'A' and host 'B'.

Apart from the presence of multiple on-path middleboxes, we can also observe that the end-point 'A' has different trust levels with the different on-path devices based on their location in their signaling path. The devices lying on the internal corporate network have stronger trust levels compared to the devices on the ISP network or the Internet. On the contrary, the middleboxes on the other end of the communication path 5,6 have a higher trust level than the middleboxes on the Internet as their identities and functionalities are partially known. However, the trust levels of the middleboxes 5,6 from the corporate headquarters are still lesser than those on the internal network 1,2 because the middleboxes from the internal corporate network belong to the same underlying IP network.

5.2.2 Trust vs Distance Model

The *trust* in our context means that the end-point information will only be used for the purposes of service provisioning.

Consider the same network scenario as shown in Figure 5.1, where 'A' is trying to set up a secure communication with 'B'. Figure 5.2 is a trust vs distance model of this network scenario. The Y-axis represents the the trust levels. The X-axis represents the distance of the various entities lying on the signaling path. At the origin lies the end-point 'A' itself, the next point on the X-axis is the QoS box 1 from the internal corporate network followed by the firewall 2 of the corporate network. If we take all the devices which lie on the signaling path and put them on the X-axis as described above, we will get the "trust vs distance" model.

So analyzing the network scenario from Figure 5.1, the devices that lie on the signaling path but within the same corporate network as end-point 'A' are strongly

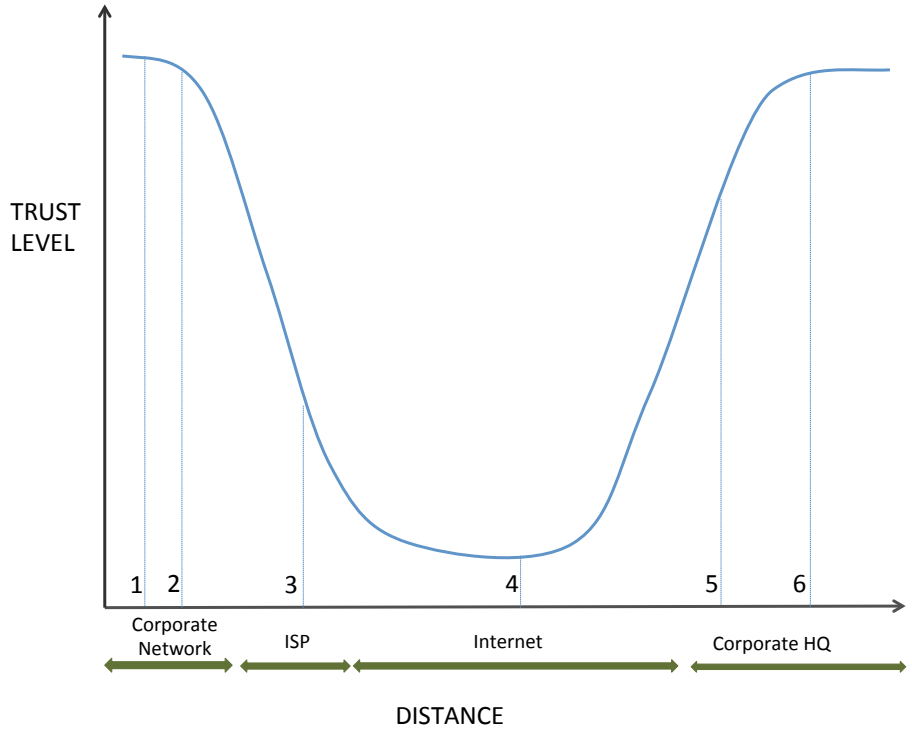

Figure 5.2 Trust vs Distance model of a typical network scenario

trusted. Hence, the highest level of trust with the devices. As we move further down the signaling path, the trust level decreases. Outside of the ISP network, devices in the vastly insecure domain of Internet have the lowest levels of trust. Moving further down the signaling path, we reach the network of the Corporate Headquarter to which the end-point 'B' belongs. Since, this is an already know network it is partially trusted. Hence, it has a higher trust level than the Internet but lower than the network to which end-point 'A' belongs.

5.3 Our approach

Our architecture consists of three major components, each of them function based on the trust level within the signaling channel. The first component provides the basic service negotiation between the middleboxes and the end-points based on the policy engines operating on both of the entities. The second component plays the important role of establishing trust with various entities on the signaling channel. The third component is critical for scoping of end-point information. We provide the details of various mechanisms for scoping, how it is based on the design goals and how they manipulate the trade-offs between performance and security.

5.3.1 Policy based Signaling: Service Negotiation

In a naive approach, an end-host signals all its information to the communication channel. This is later picked up by a variety of network entities and is utilized in provisioning middlebox services. Also, an information placed on the communication

channel is available throughout the channel and can be picked up by all the network entities lying on this path. However, this is not the most ideal way of signaling the end-point information. A better solution for this would be to place only the amount of information in the channel that is sufficient for the provisioning of on-path middleboxes. Hence, we propose a negotiation between the end-points and middleboxes for the services offered by middleboxes. The negotiation process for the middlebox includes declaring the type of service it provides and the information it requires for the provisioning of the services. The service offer for this scenario is also referred to as *"unsigned service offer"*.

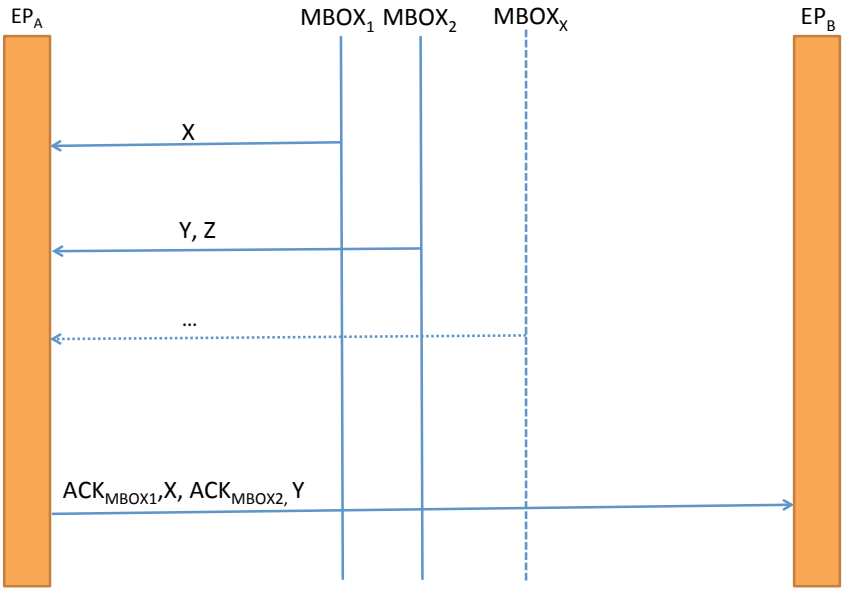

Figure 5.3 Service Negotiation and Policy based Signaling

As shown in Figure 5.3, the end-point processes the service offers received, analyzes the service offers with the policy engine. Based on the above, it may find that the policy engine completely or partially agrees with the information being requested. Or conversely, may deem the information requested by the middlebox completely unnecessary for the service being provided. In the Figure 5.3, the EP_A does not consider the information 'Z' requested by middlebox ($MBOX_4$) necessary. So, in spite of $MBOX_4$'s request for information (Y,Z), the end-point EP_A only approves for (Y). If the policy engine is affirmative of the information being requested by the middebox, the service offer is acknowledged. Conversely, if the policy engine deems the information being requested unnecessary a negative acknowledgement is sent.

The end-point replies with the positive or negative acknowledgements to the middleboxes. Based on the identifier sent earlier, the middebox can find the acknowledgements corresponding to their service offer. If a positive acknowledgement is received, then the middleboxes match the information received with their own policy engines. If all of the information that was earlier requested has been received then the service is provisioned for the end-point. However, if a partial set of the information requested is received then the middlebox may choose to provision a reduced functionality for the midldlebox services. In case of negative acknowledgement, it may choose to completely block the signaling or refuse to provide any service or in some cases choose to provide a reduced service functionality.

The overall process of negotiation allows the end-points to achieve, ***minimal disclosure***. Instead of signaling all the end-point information on the signaling channel, only a negotiated (reduced) amount of information is signaled. However, the availability of signaled information in the whole communication path is still undesirable.

However, the middleboxes can request for the specific (host or user) end-point identifiers and verify the identity after receiving it. This approach is good only for controlled scenarios (e.g. an internal network), because the end-points on one hand can not verify the identity of the middleboxes but on the other hand may need to provide their own identifiers.

The service negotiation as presented here is still not sufficient as the end-points have no way to verify the identity of the middlebox. So, a malicious middlebox along with it's partners can impersonate multiple identities and add multiple requests. That way, it will be able to fetch all the end-point information (also because the information is not encrypted).

5.3.2 Authentication and Authorization of Middleboxes

The policy based signaling works great for the controlled scenarios where trust levels are already known or strong enough, such as in a local area network. In a more sophisticated network scenario the trust levels are not known. Hence, the end-points need to determine the trust levels associated with the middelboxes on their own. Public key cryptography can be used to verify the identity of the middlebox by authenticating the public key identifier of the middlebox with the help of a trusted third party.

Consider the use case scenario from the Figure 5.1, where the end-point receives a service offer from an on-path middlebox and the trust level for this middlebox (e.g $MBOX_4$) is very low or can not be determined. As shown in Figure 5.4, the end-point replies with a negative acknowledgement with the reason specifying that the middlebox needs to authenticate itself ($NACK_{MBOX_4}(NEED\_AUTH)$). On receiving the negative acknowledgement, the middlebox responds with it's digital certificate signed by a trusted third party (TPP). Meanwhile, the end-point may choose not to avail the middlebox services till the time it authenticates. However, it may continue with the signaling handshake but without the untrusted middlebox.

Only after a middlebox has been authenticated the end-point authorizes the information request from the middlebox. A clear advantage of authenticating the identity of middlebox is that now the end-point policies can be based on the identity of the middlebox, i.e., *who requests*. Hence, the information can be scoped to only the requests from authenticated middleboxes. Also, since the end-point policies will be based on identity of the middlebox, the information requests have to atomic i.e. all information required has to be put in one request and only one request per middlebox. Moreover, imilar to the previous approach, the end-point information is still signaled in plain-text and it is available on the whole communication path.

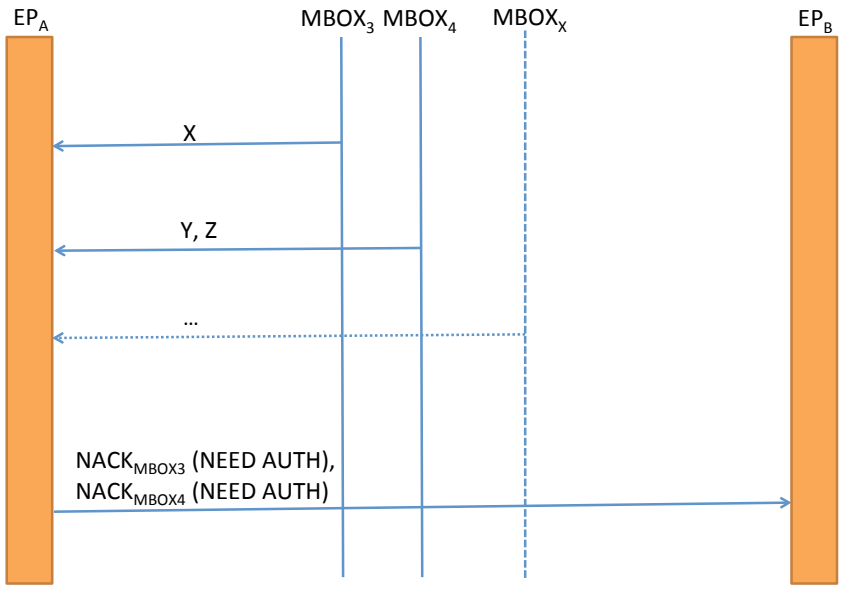

Figure 5.4 Authentication and Authorization of Middeboxes

5.3.3 Scoping of Information

"Scope" in the world of programming languages means the lifetime accessibility of a variable, object etc. ***"Scoping of information"*** would mean the accessibility of an end-point information on the signaling path. Policy based signaling can also be considered a part of the scoping of information, but it is inherently weak as we observed earlier. Hence, we present here more advanced mechanisms to incorporate the scoping of information within the handshake itself.

The two methods we present here are "encryption" and "selective signature". In the Section 2.2.1 and Section 2.2.2, we have talked about the various encryption/decryption algorithms and the advantages and disadvantages of both of them. We incorporated both the symmetric key and asymmetric key options into our approach. The service negotiation presented in Section 5.3.3 is extended to include more information in the service offer like the methods supported for the scoping of information and the cryptographic algorithms supported. So, a middlebox supporting scoping of information may choose to add that information to the service offer. An end-point policy based on the trust level it has with the middlebox, may agree to follow up with the suggested method of scoping. Or reject it altogether if its policy engines deny it or the trust level is not sufficient enough.

5.3.3.1 Encryption

One of the reliable ways to secure any piece of information is encryption. Encrypting the information negotiated with the middlebox before sending it, protects the information by restricting access to only the trusted entity. On receiving a signed service offer from the middlebox, the end-point decides whether it wants to support encryption as a mechanism to scope information. There are two options henceforth, use symmetric key encryption to encrypt the information or use asymmetric encryption.

If the end-point decides to use *symmetric key encryption*, the symmetric key can be derived from one of popular key exchange algorithms like Diffie-Hellman Key Exchange and it's Elliptic Curve variants. Symmetric encryption has obvious performance improvements. Although the performance improvements are not so visible for end-points, but the effect is clearly visible for middleboxes (less powerful processor). Another important advantage of using symmetric encryption is the saving of packet space as we show later. Since, most of the security protocols like TLS, HIP already have a symmetric key-exchange method integrated, a simple extension to the already existing protocols would suffice.

In case *asymmetric key encryption* is chosen, a symmetric key is generated by the end-point. The end-information that has been negotiated is encrypted using the symmetric key before sending. The symmetric key is encrypted using the public key of the middlebox and then sent within the same packet. The pre-requisite of using the asymmetric key encryption is that the end-point must have the middlebox certificate. This certificate could have already been obtained while authenticating the middlebox or it may have been installed at the time of configuration (for example, configuring a VPN server). Anyway, if the end-point does not have the middlebox certificate and wants to use the asymmetric encryption it has to respond with a negative acknowledgment asking the middlebox to authenticate with the end-point. Consider the

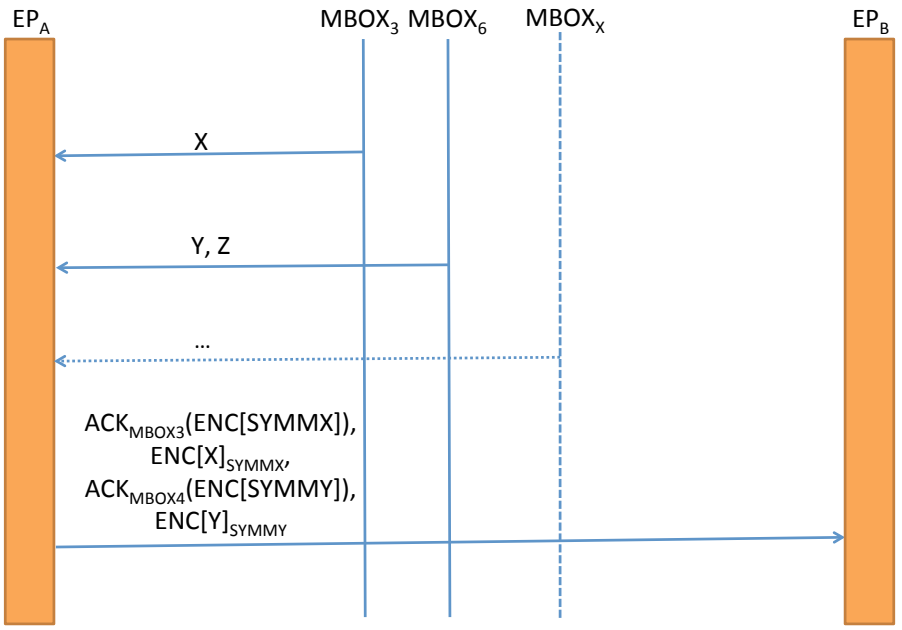

Figure 5.5 Scoping of Information using Encryption

same use case scenario as discussed in Figure 5.1. The end-point EP_A does not have sufficient trust level with the middlebox ($MBOX_3$). Although the EP_A partially trusts the $MBOX_6$, $MBOX_6$ is a VPN box and would require sensitive information about the user identity from EP_A. In both cases, EP_A must have already authenticated the middleboxes so that it can send the information after securing it. So, as shown in Figure 5.5, EP_A sends the information encrypted using a symmetric key ($ENC[X]_{SYMMX}$) and sends the symmetric key encrypted in the acknowledgment corresponding to the service offer from $MBOX_3$ ($ACK(ENC[SYMMX])$).

Although computationally intensive, encryption makes it possible for the end-points to completely hide the information from any untrusted entity. This is the most clear solution for minimal disclosure and unlinkability. Without accessing the information (after decrypting it), no device can actually link the end-point's identity with a specific information received now or previously.

5.3.3.2 Selective Signature

Encryption of any form adds complexity to the architecture. If a network or a device has a satisfactory trust level, the end-point might choose not to encrypt the information. But as we saw earlier if the information is not encrypted, it is still available on the channel to be accessed by undesired entities.

The precedence of our approach for this case comes from a daily life scenario. Our banks still rely on sending the PIN (secret) of an ATM card through post. The underlying trust model here is that the post arrives untampered and as soon the holder of the card receives the PIN, he remembers the PIN and destroys that piece of paper . At a later point of time he may choose to change the PIN also. Similar to the analogy here, the end-points can send the information in plain-text only till the point that has acceptable trust levels, with the hope that the recipient destroys/discards the information as soon as it processes it. This is shown in Figure 5.6. But in a digital world, there are more cryptographic checks in place, like signature and integrity checking. Any removal of a portion of the packet, will break the signature verification and integrity checks.

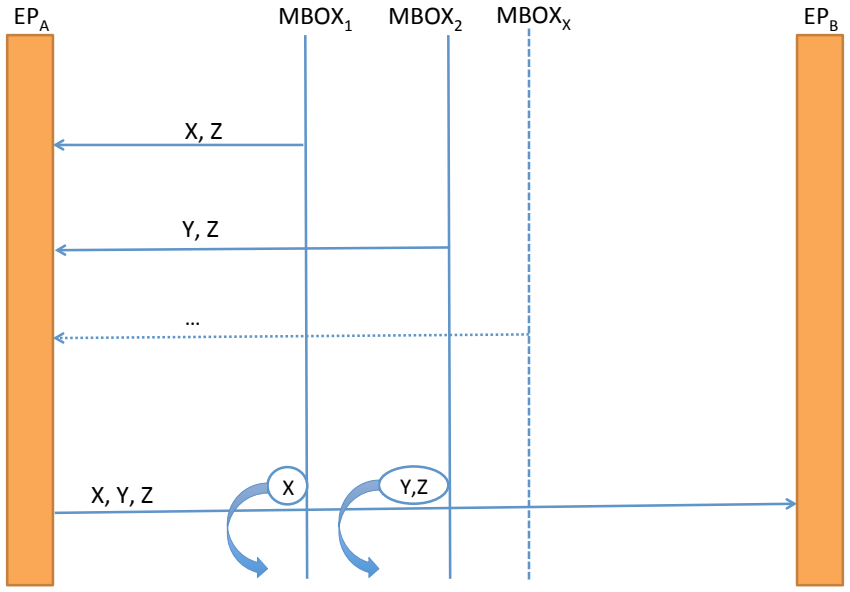

Figure 5.6 Scoping of Information using Selective Signature

We deal with this using a mechanism called "selective signature". One of the many ways to do this is to divide the packet into multiple chunks, build a hash tree from the chunks of the packet, and sign the root of the hash-tree. Both the signature and MACs are done on the root of the hash-tree. Whenever, a data chunk is removed it can be replaced with the hash of the message. This way the hash tree is not

broken and the signature verification and integrity checks can be done successfully. Translating the idea to our use case, the hash tree can be build in a way that each of the pieces of information form one data chunk. This way it's easier to remove an end-point information without breaking the signature and integrity checks.

The selective signature scheme serves two key purposes for us. Firstly, the ability to remove messages after it has been processed makes the information available in the signaling channel only till the point trusted by the end-point. Secondly, since the end-point information is removed as soon as it is processed, none of the devices on the signaling-path lying further receive the information. The strength of hash algorithms make it impossible for the malicious devices from computing the pre-image of the hash (actual end-point information).

We have to realize that the "selective signature" scheme is only applicable to the controlled scenarios for example inside a corporate network. Figure 5.6 is illustrative of the use case shown in Figure 5.1. The middleboxes ($MBOX_1$ and $MBOX_2$) are within the same corporate network as EP_A. Hence, according our trust vs distance model from Figure 5.2, these middleboxes have the highest level of trust from EP_A. The end-point can send information unencrypted to these middleboxes. EP_A however does not trust the middlebox beyond the corporate network to which it belongs. So, as shown in Figure 5.6, the information X is discarded from the packet as soon as it's processed by the middlebox $MBOX_1$. However, Z is not discarded from the packet since it is being used further down the signaling path. Finally, Y and Z are processed by $MBOX_2$ and then discarded. The information about which of the information pieces not to discard can be added to the corresponding acknowledgements.

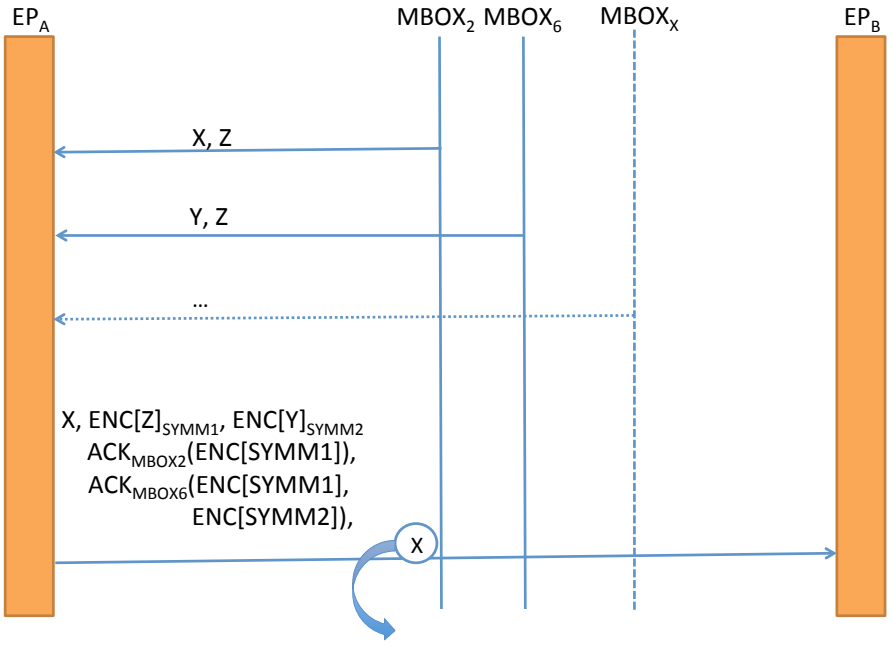

Figure 5.7 Scoping of Information using Selective Signature and Encryption

5.3.3.3 Selective Signature and Encryption

The real world scenarios, can not be handled exclusively with one of the cases as presented above. So, in order to handle a complicated scenario as the one depicted in Figure 5.1 with a variety of middleboxes together, we would require an approach that can combine all of the principles discussed above. For the sake of simplicity, consider the two middleboxes $MBOX_2$ and $MBOX_6$ from Figure 5.1, as shown above. $MBOX_2$ lies within the same corporate network (trusted network) as the end-point EP_A, hence can receive the end-point information unencrypted. $MBOX_6$ being a VPN requires sensitive end-information like user and host identifiers. Since the complete signaling path is not secure, it's necessary for EP_A to secure those identifiers before sending them to $MBOX_6$.

In Figure 5.7, middlebox $MBOX_2$ requires the information X and Z from EP_A and the middlebox $MBOX_6$ requires information Y and Z. After checking with it's policy, the end-point can send X, Z unencrypted to $MBOX_2$ using the Selective Signature scheme and an encrypted version of Y and Z using the Encryption scheme to $MBOX_6$. Conversely, it can also choose to send both X, Z without encryption using the Selective Signature and an encrypted Y using the Encryption scheme. The first alternative is not efficient in terms of packet space which is already limited (Z is sent twice). The second alternative has security risks as we discussed in the problem analysis. A better alternative to the problem is to allow for sending of X unencrypted, encrypt Z using a symmetric key SYMM1 for both of the middleboxes $MBOX_2$ and $MBOX_6$ and encrypt Y using another symmetric key SYMM2 for $MBOX_6$. SYMM1 is sent encrypted with the acknowledgement to $MBOX_2$, while both SYMM1 and SYMM2 are sent encrypted to $MBOX_6$. This way $MBOX_2$ can only decrypt Z using SYMM1. On receiving the packet, $MBOX_2$ decrypts Z first, processes both X and Z and finally discards X from the packet. $MBOX_6$ decrypts Z and Y from SYMM1 and SYMM2 respectively and processes the information received.

5.4 The Extension of HIP Base Exchange

HIP Base Exchange although signaling friendly and middlebox aware does not provide a detailed service negotiation. We have already seen some previous efforts on enabling service negotiation within the HIP Control Channel. However, our design goals provide more functionality to the service negotiation than SEAMS (see Section 3.1.3.1). Similar to our discussion in Section 5.3 we have extended our service negotiation into three schemes: *negotiation, authentication and selectively disclosure*. Our extension is compliant with the standard HIP BEX and does not in anyway alter the existing BEX mechanism. As an optimization to our extension, we perform the context lookup for the host at the bootstrap of the HIP Daemon. This can be done because the host context does not change over a period of time and remains unaffected over multiple connections, different applications and users.

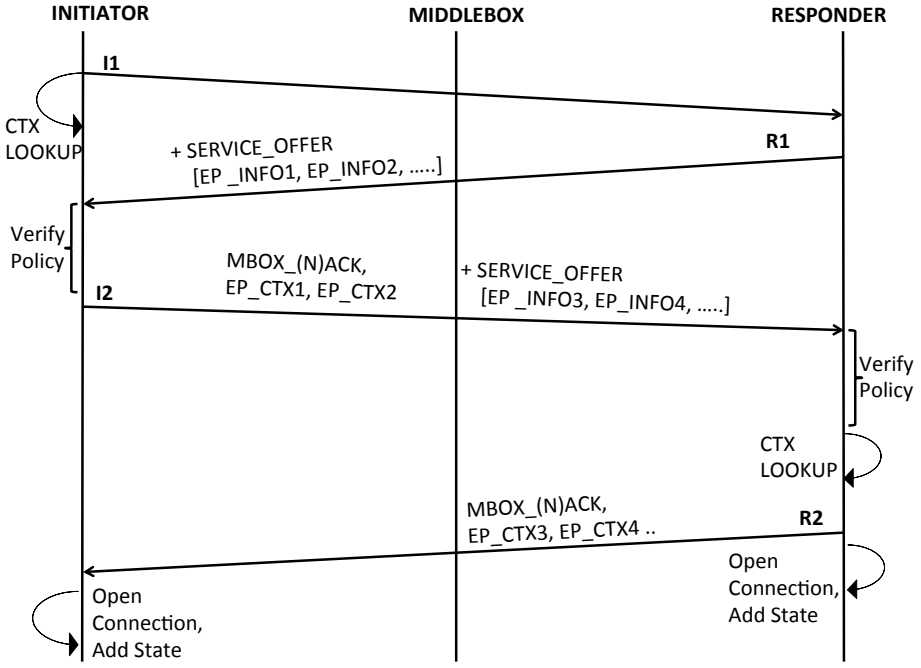

Figure 5.8 Unsigned Service Negotiation

5.4.1 Negotiation: Unsigned

It is the most basic form of service negotiation. The base exchange is started whenever an application tries to establish a connection with a peer. This connection setup is intercepted by the HIP Firewall and instructs the HIP Daemon on the Initiator side to trigger a new BEX if there is no existing HIP association between those two peers. Otherwise, a HIP Update is triggered. HIP BEX for an unsigned service negotiation is shown in Figure 5.8.

5.4.1.1 I1, Trigger BEX

`I1` packet has been designed to contain as minimal information as necessary for the BEX. The design has been to protect the Responder from Denial of Service attacks as we also discussed in Section 2.4.4. We do not want to fiddle with the existing mechanisms and in turn introduce more vulnerabilities. `I1` is completely unaffected. The processing of `I1` at the middleboxes is also completely unchanged and unaffected in our extension. The Initiator knows which user and application have triggered the BEX. So, the Initiator can already do the context lookup for application and user information.

5.4.1.2 I1 Processing and Sending of R1

The handling of `I1` does not need to be changed since the `I1` packet is unaffected. `R1` packet has been designed to be pre-computed. The pre-computation of `R1` is essential to protect the Responder from Denial of Service attacks. Also, at this point of time, the Responder does not maintain any connection state which further provides protection against DoS attacks. Our design preserves these functionalities.

We do not initiate any context lookup, state maintenance or processing at the Responder. The Responder does not have any knowledge of the application service that the Initiator has request for or information about which ports will be involved in the connection. Without this information the Responder can not perform context lookup. Hence, the Responder does not have application or user contexts available at this point and should not signal any of this information to the signaling channel yet. The corresponding information relating to the connection setup will be provided by the Initiator subsequently.

5.4.1.3 The processing of R1 by middleboxes

`R1` contains the HITs of both the Initiator and the Responder. This rudimentary information can be used by the middlebox to perform a policy check. Based on the minimal information available, the policy engine determines the information it requires from the Initiator for the provisioning of it's middlebox. This decision may or may not be based on the HITs. For example, a Firewall that has been configured to only permit connections to authorized users would always request for user identifier. The middleboxes may request for specific information parameters or may request for user or host identities and the corresponding certificate chains for authentication.

All the information items that the middleboxes require from the Initiator are compiled in the form of an unsigned *Service Offer* parameter and appended at the end of `R1` message. The service offer can not be covered in the HIP signature, hence is appended to the unsigned portion of the `R1` message. This process is repeated by every on-path middleboxes.

5.4.1.4 R1 processing and Sending of I2

The preliminary handling of `R1` remains unchanged. `R1` packet is validated and the puzzle is solved as usual. The creation of `I2` starts as usual with the addition of transforms and the solution of the puzzle. Next, the Initiator start processing the service offers received. There might be multiple service offer parameters. It can be assumed that the Initiator has already completed its context lookup and the context is fresh. The Initiator then processes each of the service offer through its policy engine. The policy engine makes a quantitative decision on which of the requested information items should be considered private or unnecessary for the provisioning of the middlebox services. The Initiator summarizes all of the policy decisions and adds one by one each of the approved information items. If the user identity is requested by a middlebox and it is deemed appropriate by the Initiator's policy engine, the corresponding user identity is added. Along with the user identity, the user signature on the HIP message is also added. However, if the certificate chain corresponding to the user identity is also requested, it is not possible for the Initiator to provide the complete certificate within the already crammed `I2` packet. The certificate chain is then provided in a separate certificate exchange.

Corresponding to each of the service offers, the Initiator adds a positive acknowledgement if the policy engine has deemed the information requested in the service offer

to be appropriate. Otherwise a negative acknowledgement is added with the reason for the decline of service offer mentioned. For the scenario, where the certificate chain is requested, the Initiator chooses to respond with a negative acknowledgement indicating that the middlebox expect for a separate certificate exchange with the Initiator. The acknowledgement parameters contain the hash of the service offer which helps the middleboxes to identify the acknowledgement corresponding to the service offer.

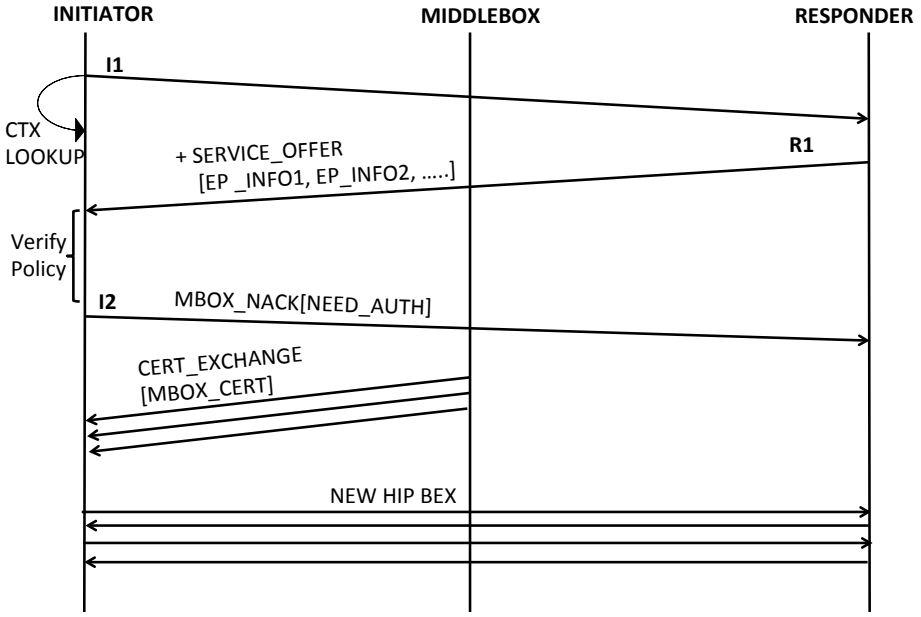

Figure 5.9 Certificate Exchange

In certain scenarios, the Initiator's policy engine might not trust the middlebox enough to provide any of the information requested. The Initiator then responds with a negative acknowledgement asking the middlebox to authenticate itself via a certificate exchange. Even if the BEX does not complete owing to the missing certificate, it is fair to assume that the Initiator can trigger another BEX after the completion of certificate exchange as shown in Figure 5.9.

5.4.1.5 The processing of I2 by the middleboxes

Upon receiving the `I2` packet the middlebox verifies the Initiator's signature. This way the middlebox is able to bind the Initiator's identity to the information that have been provided in the `I2` packet. Only after the signature verification, it looks for the acknowledgement corresponding to its service offer. It then verifies the information available with the policy engine and if the policy check is successful the middlebox service is provisioned for the Initiator. If a user identity was requested in the service offer, the middlebox expects a user signature on the `I2` packet using the public key in the user identifier. This way the middlebox can bind the user identity to the information provided. Failure in the verification of user signature implies that the user identity can not be trusted. Conversely, if a negative acknowledgement is received, the reason provided is analyzed and the response is determined again by the middlebox policy engine. In some cases, the on-going BEX is completely blocked

or in other cases the BEX may be allowed to continue without the middlebox service. If the middlebox chooses to block the connection, it has to notify both the peers so that the BEX can be stopped. For the certificate exchange, the middlebox withdraws from the on-going BEX and only after completion of certificate exchange and the verification of the Initiator's identity the middlebox services are provisioned. On the other hand, if the Initiator asks the middlebox to authenticate itself, the middlebox would start a certificate exchange with the Initiator. In the meantime, the middlebox may choose to block the connection or allow the BEX to continue.

After the processing of service acknowledgement, the middlebox again does a similar policy check for the Responder. The information items required by the middlebox are added in the form of service offers at the end of the `I2` packet. All of the on-path middleboxes have to perform the same process one by one along the signaling path as outlined here.

5.4.1.6 I2 Processing and Sending of R2

The preliminary verification of the `I2` packet is performed as usual. The host signature and the HMAC are verified and the solution to the puzzle checked. After the usual validation and verification process, the Responder determines if the user or the application context lookup is necessary for the processing of the service offers. Only if necessary, it initiates application and user context lookup. The host context has already been looked at the time of bootstrap of the HIP Daemon. The Responder the starts processing the service offers in the `I2` packet similar to the way the service offers are processed by the Initiator on the `R1` packet.

After the successful processing of the service offers and the addition of the service acknowledgements, the `R2` packet is sent. The Responder then opens the connection on his side and adds the corresponding connection state to the HIP association database. The Responder does not keep any state before this point.

5.4.1.7 The processing of R2 by middleboxes

The `R2` packet is processed by the middleboxes similar to the processing of `I2` packet. On a successful validation of identities produced and verification of information provided in the `R2`, the middlebox services are provisioned for the Responder. If a user certificate was requested, then similar to the processing `I2` the middlebox waits for the completion of certificate exchange before provisioning of any middlebox services for the Responder.

5.4.1.8 R2 processing

The Initiator inspects the `R2` packet and does the necessary verification and validation as in standard HIP implementation. After the successful inspection of `R2`, the Initiator opens up the connection on his side.

The Base Exchange is concluded and the middleboxes services are provisioned at the conclusion of this step. The extension to HIP BEX presented here form the

foundation to addition of more advanced functionalities within the BEX. We have mentioned in the Section 5.3.3.3, that the basic service negotiation is only suitable for a highly trusted network scenario. We provide more advanced schemes next.

5.4.2 Selectively Disclosure: Selectively Signed Service Negotiation

The network scenario presented in Figure 5.1, provide a much vivid picture of a typical network scenario where the signaling channel extends across changing trust levels. Selectively Signed Service Negotiation allows the middleboxes to discard information items as they are processed. Although most of the service negotiation is similar to the one we have presented above, the generation of signature and HMAC are different in order to incorporate the removal of parameters. The BEX is the same as above till the point `R1` is received. After processing the service offers in `R1`, the Initiator builds a hash tree of the `I2` packet as shown in Figure 5.10. Selective Signature and HMAC are taken over the root of the hash tree. In order for the end-points to trust the middlebox with the information removal process, the design compulsorily requires the middlebox to sign their service offers with their public keys. Earlier service offers were neither signed nor integrity protected. The signature ensures the integrity of the service offer and allows the Initiator to authenticate the middlebox before it can trust it. The middlebox signature on service offer, adds an additional layer of trust between the middlebox and the Initiator. If the middlebox certificate is required for authentication, the Initiator can request for certificate exchange just as we discussed earlier. If the signature verification fails, then the Initiator can not trust the middlebox and will have to reply with a negative acknowledgement.

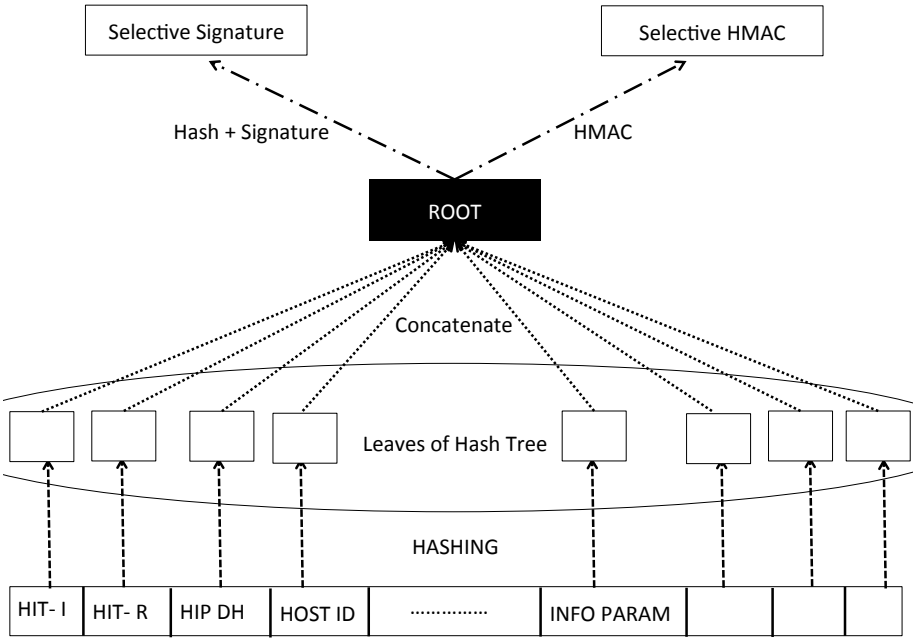

Figure 5.10 Hash Tree from a HIP Packet

Upon the receiving the `I2` packet, the middleboxes recreates the hash tree form the `I2` packet, verifies the signature on the root of the hash tree. The presence of hash

leaf parameter at the end of the I2 packet would suggest that some of the parameters have been removed. The middlebox then inserts the hash leaf at the appropriate position while building the hash tree. After the verification, validation of the I2 packet and the policy checking, the middlebox looks for the parameters that can be safely removed. This information is provided by the end-point in the service acknowledgement itself. The parameters that have been marked to be removed are discarded and the I2 packet is appended with the hash leaf corresponding to the removed parameters.

The verification of signature and HMAC on the I2 packet is similarly done at the Responder side. Handling of service offers in I2 packet is done in the same way as unsigned service offer. The hash tree from R2 is generated and the root of the hash tree is selectively signed. The processing of R2 by the middleboxes is similar.

5.4.3 Authentication: Signed Service Negotiation

Signed Service Negotiation protects the sensitive end-point information using encryption. This provides the end-points with strong mechanisms to undergo service negotiation in a network scenario with very low trust levels. It makes it compulsory for the middleboxes to authenticate with the end-points and only then securely transmit the negotiated information. The pre-requisite for a signed service negotiation is that the end-point already possesses the middlebox certificate. If the end-point does not have the middlebox certificate, it should request the middlebox certificate with a negative acknowledgement for the certificate.

The service offer parameter can be used by the middlebox to provide the end-point with a hint about the certificate, The end-point can use the hint to locate the middlebox certificate. This certificate helps the end-point to authenticate the middlebox preemptively without requesting for a new certificate exchange. Once the middlebox has been authenticated, the end-points can either choose to use the public key in the certificate for encryption or a DH based mechanism as we show below. But before discussing the BEX with signed service negotiation, we want to present *grouping of information items*.

5.4.3.1 Grouping of Information Items

Signed Service Negotiation is the strongest form of protection provided to secure the sensitive end-point information. A privacy compliant architecture, would not allow the information requested by one middlebox be visible to any other middlebox. However, in common scenarios it is possible that two middlebox request for some common information items. Without breaking any privacy requirements, the end-point can group the common information items together and encrypt the group such that only the middleboxes which requested for the items can decrypt it. This kind of restriction on who can decrypt is necessary to limit the availability of information to only the trusted entities.

To achieve this we generate a symmetric key, encrypt the whole group of information items together with the symmetric key cipher. The symmetric key is then encrypted for each of the middlebox with its public key. So, only the middleboxes who are

able to decrypt the symmetric key with their private key can actually decrypt the information. This indirection in necessary to secure the information in a privacy preserving way. This mechanism is illustrated by an example as shown in Figure 5.11. In this example, `MBOX_1` requests for information items `A, B, C` and `MBOX_2` requests for information items `B, C`. Information items `B, C` can be grouped together and encrypted together as shown in the figure. As a result of grouping, the information items request by the `MBOX_1` are split into multiple groups `(B,C) and A`. `MBOX_1` must decrypt both `SYMM1` and `SYMM2` in order to read all three of the information items `A, B, C`.

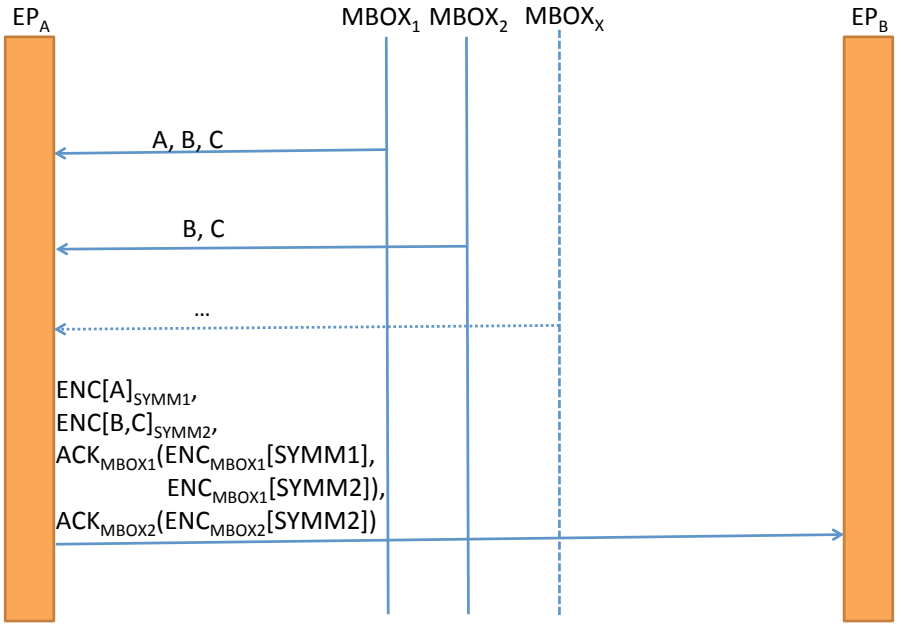

Figure 5.11 Grouping of Information Items

5.4.3.2 Signed Service Negotiation using Asymmetric Key Encryption

On receiving the `R1` packet, the Initiator looks into the service offers. If a signed service offer has been received, the Initiator locates the middlebox certificate with the help of the hint available in the offer. Initiator collects the information items requested by the middleboxes and runs them across its policy engine. After the policy check, the Initiator can group service offers as described in the previous subsection. Initiator also maintains a mapping of the groups to the middleboxes to which they belong. For each of these groups, a symmetric key is generated. Each of these group of information items are encrypted with their corresponding symmetric key. The encrypted information is carried over the `HIP_ENCRYPTED` parameter. A `KEY_HINT` is copied to the reserved field of the `HIP_ENCRYPED` parameter. The symmetric key is then encrypted for all of the middleboxes to which the group belongs to. This is done for all the symmetric keys. The encrypted symmetric keys along with `KEY_HINTs` are then added to the corresponding service acknowledgements. All of these parameters are put together in the `I2` packet and sent.

The middleboxes on receiving the `I2` packet, look for the service acknowledgements corresponding to their service offers based on the hash of the service offer. From

the acknowledgement, they can extract the encrypted symmetric keys and decrypt them using their private keys respectively. The symmetric keys are bundled with `KEY_HINTs` to help the middleboxes to find the corresponding `HIP_ENCRYPTED` parameter. As we saw in Figure 5.11, the information items requested by a middlebox can be split into multiple groups. After decrypting the related encrypted groups and collecting the information items from the groups, the middlebox verifies the information received with its policy. If the policy check is successful, the middlebox service is provisioned for the Initiator. Our extension is symmetric for both peers, hence the above process is repeated for the Responder over `I2` and `R2` packets.

5.4.3.3 Signed Service Negotiation using Symmetric Key Encryption

Signed Service Negotiation can be used similarly with symmetric key encryption. Instead of using the public key of the middlebox to encrypt the symmetric keys, a shared secret key may be used to encrypt the symmetric keys. HIP BEX already includes a Diffie Hellman Key Exchange to establish a shared secret key between the two peers. `R1` message from the Responder and the `I2` message from the Initiator contain the respective Diffie Hellman public keys. The Diffie Hellman keys are large primes and it is not feasible for the middlebox to also include their own DH public keys into the HIP Control Channel. Hence, the middleboxes have to resort to `static Diffie Hellman Keys`. The usage of static DH keys makes the shared key between the end-point and the middlebox vulnerable if the static keys are compromised. However, they do not affect the *forward secrecy* because HIP depends on new DH keys for each protocol run and the static DH keys can be updated periodically.

The static DH keys of the middleboxes can be obtained by the end-point from a centralized directory service which updates them periodically. The end-point can then have to depend on the directory service for the association between middlebox identity and the static DH key. The directory service provides the middlebox with the DH private key and the end-points with the public counterpart. However, the end-points still need to trust the middlebox before committing to a signed service negotiation.

On receiving `R1`, the Initiator can use the hint in the service offer to locate the middlebox certificate and authenticate the middlebox. Thereafter, it can continue with policy checking and splitting of the information items into groups. Symmetric keys are generated for each of the groups and used to encrypt the information groups as described previously. Using the static DH public key of the middlebox and its own DH private key, the Initiator can compute a shared key. This shared key can be used to encrypt the symmetric keys of the groups related to the service offer. The encrypted symmetric keys are wrapped in the service acknowledgement and sent to the middlebox in `I2` packet.

The middlebox on receiving the `I2` packet, finds the service acknowledgment corresponding to its own service offer, extracts the encrypted keys from the acknowledgments. Using the DH public keys of the Initiator present in the `I2` packet and the DH private key of the middlebox, the middlebox computes the shared key. This shared key is then used to decrypt the symmetric keys found in the service acknowledgement. Rest of the handling is same as the asymmetric key approach discussed before.

The service negotiation as we have described here is a *three way* message exchange. `R1` carries the service offers from the middleboxes to the Initiator. The Initiator provides the requested information in the `I2` packet. The middlebox processes the `I2` packet for the policy verification with the information received. After a successful verification, the middlebox appends the service offer for the Responder. The Responder replies with the information required in the `R2` packet. Owing to the three way message exchange for the service negotiation, the `UPDATE Exchange` is also a three way exchange.

5.5 The UPDATE Exchange

The HIP Update Exchange is also a three-way exchange as discussed previously. Whenever an application starts a connection setup, the HIP Daemon checks for an existing host association from an earlier BEX. If a previous host association is found, an UPDATE Exchange is triggered, otherwise a new BEX is triggered. The motivation behind an UPDATE Exchange is to aggregate the new connection into the already existing payload channel. Figure 5.12 gives an overview of the HIP Update Exchange.

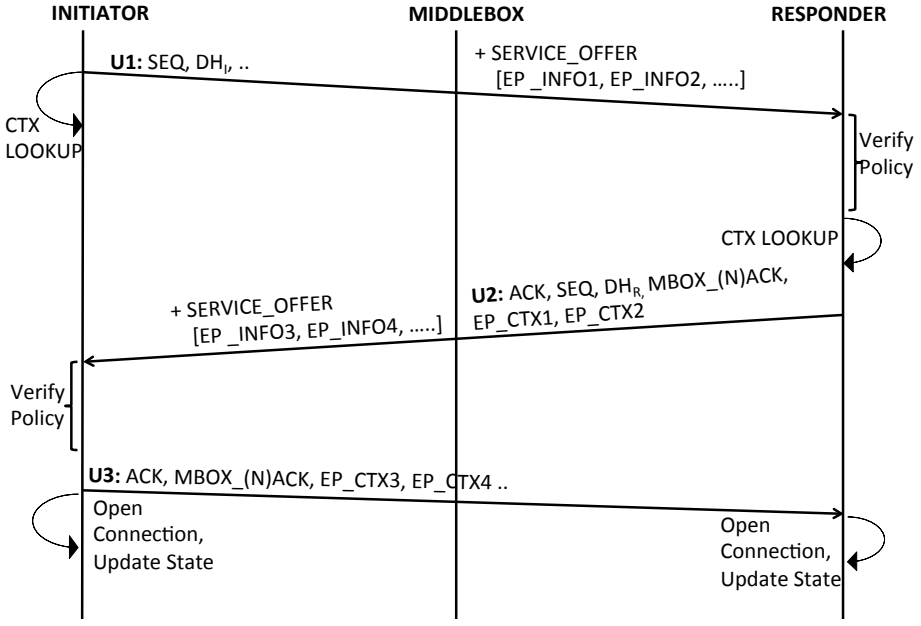

Figure 5.12 HIP Update Exchange

5.5.1 The U1 Packet

The `U1` packet triggers the Update Exchange with a sequence number, `SEQ`. This `SEQ` is used by the Responder to identify an outdated update and also indicates to the Responder to acknowledge the `U1` packet. In order to allow for the middlebox to have a signed service negotiation, the Initiator adds its DH public key. The Initiator also does a context lookup after sending of `U1` packet. The middlebox on receiving the `U1` packet, check with the policy engine and add the service offer. The middlebox

can already compute the shared key between the Initiator and itself using the DH public key in the U1 packet and its own DH private key. This will be used later in the Update Exchange and destroyed thereafter.

5.5.2 U1 Processing and the U2 Packet

The handling of U1 packet is similar to the R1 packet at the middleboxes and the end-hosts, except for the context lookup by Responder after receiving U1. HIP Update between two peers with no prior host association must be dropped. The Responder verifies and validates the U1 packet, adds responses to the information requested in U2. The U2 packet carries the Responder's sequence number and an acknowledgement to the one received in U1. Just like U1, the Responder also adds its DH public key to the U2 packet.

The middlebox processes the U2 packet like the I2 packet. It uses the Responder's DH public key and its own DH private key to generate the shared key between the Responder and itself. The shared key may be used in case of signed service negotiation for decrypting the information received. It verifies its policy with the information received in U2 packet and provisions its services on success. In case of selectively signed service offer, the middlebox can remove the marked info parameters. Consequently, the middlebox also does a policy check for the Initiator and adds the corresponding service offer.

5.5.3 U2 Processing and the U3 Packet

The processing of U2 at the Initiator is largely the same as the processing of the I2 packet at the Responder. The Initiator processes the service offer and adds responses to the required information in the U3 packet. U3 packet also contains the acknowledgement to the sequence number received. If everything goes on properly, the Initiator opens the new connection.

The middlebox processes the U3 packet similar to the R2 packet. It may need to use the shared key between the Initiator and itself, computed when processing U1. On a successful policy verification of the information received, the middlebox provisions its services. It may need to remove info parameters from the U3 packet if selectively signed service offer is used.

5.5.4 U3 Processing

The Responder verifies and validates the U3 packet and thereafter opens the new connection.

5.6 Integration with BLIND

In Section, we provided an overview of BLIND and how the blinding of identifiers protects the end-point identity from being revealed to untrusted entities. Furthermore, Zhang et. al. showed how BLIND can be integrated with HIP BEX. Although

these mechanisms help in mitigating the weaknesses of Public Key Cryptography, they break the existing functionalities of middlebox (cf. Section 4.2.1). In this section, we discuss how can we integrate the blind extension of HIP with our extension.

The most basic form of blinding schemes can be implemented without any alteration to the unsigned service negotiation. The underlying assumption in these cases is that the signaling channel has enough trust level for the end-points to completely trust middleboxes and service offers and for the middleboxes to consider the information provided to be true.

Conversely, in a typical network scenario, it is not safe to assume that the various network entities can be trusted without authentication. The authentication process however, requires the usage of public key cryptography and exchange of certificate chains. Exchanging certificates in plain text is a source of *identity privacy* problem. The blinded HIP BEX would work fine with our service negotiation if the network entities have already established mutual trust. The blinded BEX allows the sending of identifiers only after encrypting them and not in plain text. Encrypting the identities with the public key of the other party would securely deliver the the end-point identity to the other party. The primary requirement to squeeze our service negotiation into a blinded BEX is to securely transmit certificates and then establish trust.

We have two scenarios, one where the end-point needs the middlebox to authenticate and the other where the middlebox wants the end-point to authenticate. A middlebox normally needs to cater to a large number of end-points or large volumes of traffic. It is not feasible for the middlebox to perform an encrypted certificate exchange without severely degrading it's performance. So, we propose an encrypted certificate exchange for the end-point to authenticate with a middlebox and an unencrypted certificate exchange for the middlebox to authenticate with an end-point.

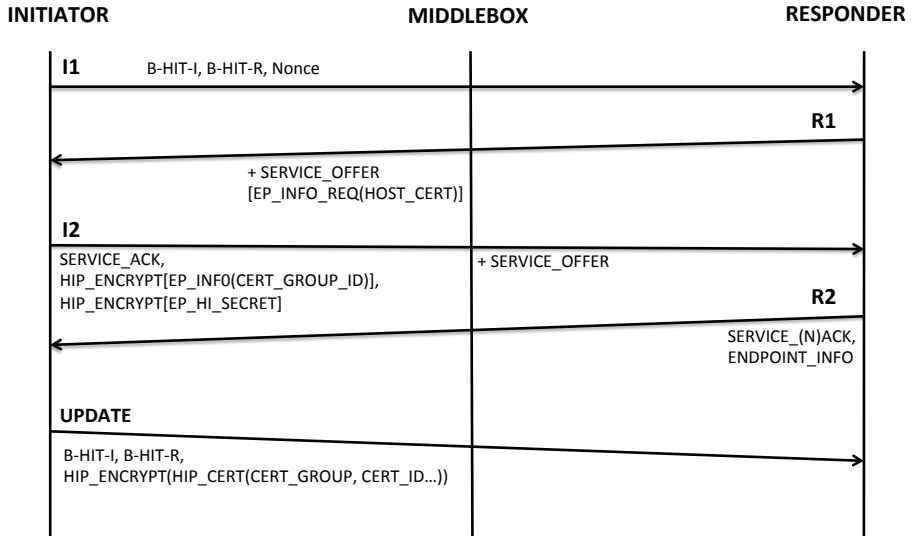

Figure 5.13 Blinded HIP BEX where End-Point has to authenticate with the Middlebox

5.6.1 Certificate Exchange where End-point has to authenticate with the Middlebox

When a middlebox receives the R1 message, it does not have any end-point identity available to base it's policy on. Although it can add a service offer based on some general policies like application information for a application level gateway. In this case, however, middlebox might choose to request the Initiator to authenticate itself. The authentication process requires the Initiator to supply its certificate chain to the middlebox, but sending the certificate in plain is not a great idea. So, the middlebox adds a service offer with an information request for the certificate (e.g. host identity certificate). Figure 5.13 depicts the handshake in blinded HIP Base Exchange when the end-point has to authenticate with the middlebox.

The Initiator on receiving the service offer responds back in the I2 packet with a signed service acknowledgement and a Certificate Group ID wrapped in HIP_ENCRYPTED. With the help of the symmetric key from the service offer, the middlebox can decrypt the Certificate Group ID. It will identify several parts of a certificate chain. On receiving the I2 packet, the middlebox can either choose to block the BEX till the Initiator verifies with it or allow the BEX to run without provisioning its service for the Initiator.

If the middlebox allows the BEX to continue, the Initiator completes its BEX and the middlebox stores a state corresponding to this connection. After the completion of the BEX, the Initiator triggers a certificate exchange with the middlebox. The certificate exchange in this case is carried over HIP_UPDATE packets. The certificate chain is broken into multiple parts. Each of these parts is wrapped around in a HIP_CERT parameter which also carries the Certificate Group ID which we mentioned earlier. These HIP_CERT parameters are encrypted and wrapped in HIP_ENCRYPTED parameter. The HIP_ENCRYPTED parameter is added to the HIP_UPDATE and sent to the middlebox. Depending on the size of the certificate chain there might be multiple HIP_ENCRYPTED parameters and multiple HIP_UPDATE messages. After the certificate chain has been verified, the middlebox sends a certificate acknowledgement that the Initiator's certificate has been received and verified. The Initiator can then trigger an *Update Exchange* so that middlebox can now send a list of information required on a new service offer. Upon receiving the information items requested, the middleboxes can provision its services for the Initiator. The same process can be repeated for the Responder as the BEX is symmetric with respect to the message exchanges.

If the middlebox wants the end-point to authenticate before continuing through the BEX, it can notify the end-point about its intentions using HIP_NOTIFY. After the end-point completes it's certificate exchange as described above, it can trigger a new BEX.

5.6.2 Certificate Exchange where the Middlebox has to authenticate with the End-Point

The middleboxes usually are in the middle of many ongoing service negotiations and simultaneously handle large volumes of traffic. Hence, it is not wise for them to send encrypted certificates. Since. encryption is a computationally intensive operation,

encrypting the middlebox certificate before sending can also make them vulnerable to Denial of Service attacks.Figure 5.14 depicts the handshake in blinded HIP Base Exchange when the middlebox has to authenticate with the end-point.

On receiving the R1 packet, the Initiator can respond with a negative acknowledge met asking the middlebox to authenticate itself. The middlebox would then go for a certificate exchange. After the Initiator has verified the middlebox certificate, it can trigger a fresh BEX.

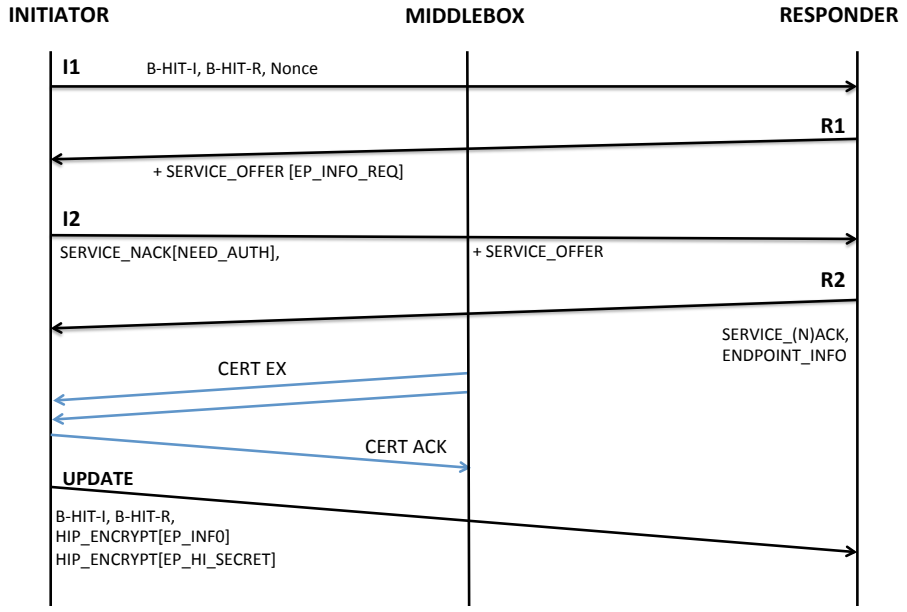

Figure 5.14 Blinded HIP BEX where Middlebox has to authenticate with the End-Point

5.7 HIP Control Channel Integration

In previous section, we discussed why HIP is suitable for the implementation of our approach. The reusability of existing HIP mechanisms, messages and parameters is critical to our implementation. Thereby, our design is extensively based on the HIP Control Channel which provides compatibility with the standard HIP implementations. However, we would still be needing new message and parameter types and extending the HIP Control Channel messages.

5.7.1 New Parameter Types

The integration of our architecture to HIP requires the introduction of new HIP parameters. These parameters will be used for adding middlebox service offers, responding to service offers, adding end-point context information to the signaling channel, selective signatures and much more. All of the new parameters obey the HIP guidelines for TLV format of all parameters (cf. Section 2.4.3.2).

5.7.1.1 Parameters for Service Offer

Our design requires the middleboxes to explicitly request for the information they need in order to provision their services. Primarily, the service offer parameter contains information about the service, a unique identifier for the service offer and a list of information items required from the end-point. Although this is sufficient for the basic service negotiation as described in Section 5.3.1, we need additional information like service signature to verify the origin of service offer or cryptographic identifiers to authenticate a middlebox.

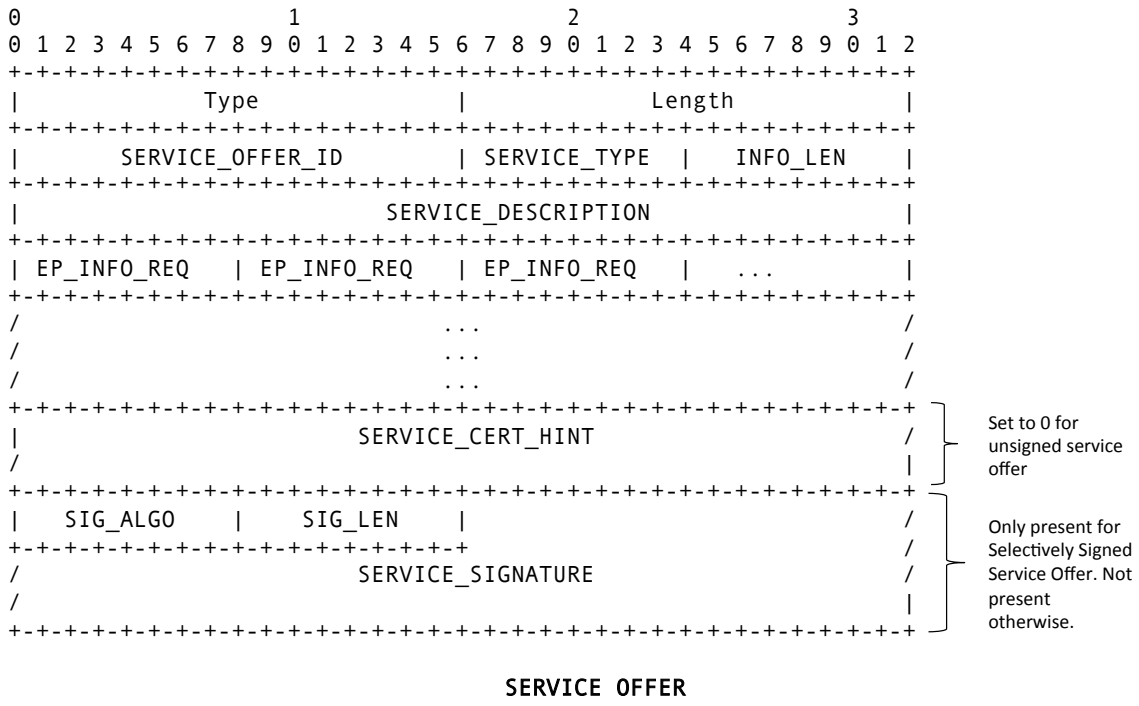

Figure 5.15 Parameter for Service Offer

Figure 5.15 depicts the service offer parameter used in our implementation. The parameter type is set to be 62506 that makes it an uncritical parameter and hence compatible with the existing HIP implementations. The service offer parameter is appended by the middlebox to the end of HIP packet (unsigned part) as inserting it in the signed part would cause the HIP signature to fail. `SERVICE_OFFER_ID` is a unique number identifying a specific service offer. It has to be unique with respect to one middlebox, i.e, one service offer per middlebox and also unique across the signaling channel. It helps in preventing replay attacks, where a malicious entity may record an on-going service negotiation and replay and old service offer to obtain information about the end-point. `SERVICE_OFFER_TYPE` indicates the type of service offer, i.e, an signed, unsigned or selectively signed service offer (cf. Section 5.4.1, 5.4.2, 5.4.3). `SERVICE_DESCRIPTION` provides information about the service provided by the middlebox. For example, it may contain information about the encryption algorithms supported by the middlebox (e.g VPN) or the codec types supported by a transcoder.

The service offer parameter also contains information about the required information items (`EP_INFO_REQ`). The number of included information items is stored in

INFO_LEN. It is safe to assume that an end-point may already possess the certificate of a middlebox owing to a previous certificate exchange or the certificate may have been received at the time of configuration (e.g VPN configuration). So, in either case the end-point does not need to go through the entire certificate exchange again. SERVICE_CERT_HINT is a hint to help the end-points in finding the corresponding middlebox certificate. The middlebox certificate obtained with the help of the hint is used to authenticate the middlebox before proceeding with the processing of service offer. The hint field is set to 0 if the service offer is unsigned. For the signed and unsigned service offer types, the service offer parameter ends after the SERVICE_CERT_HINT.

The selectively signed service offer contains an additional signature of the middlebox on the service offer. While the middlebox certificate is used to authenticate the middlebox, the signature verifies the origin of the packet (integrity protection). In Section 5.3.3.2, we showed that the selective signature scheme is useful in controlled and trusted scenarios as it has a lower overhead compared to the signed service offer. Hence, it's important that the end-points completely trust the on-path middleboxes and the origin of the service offer before transmitting sensitive information in plain text.

5.7.1.2 Parameters for Service Acknowledgement

In the previous subsection, we saw how we could use the same parameter for different types of service offer. However, the same functionality couldn't be achieved with the same parameter for the acknowledgements corresponding to the different service offers. We have defined three different types of service acknowledgement parameters corresponding to the service offer. The parameter type is 5122, keeping it in the signed portion of the HIP Packet. Although the parameter types are same for all the three service acknowledgement parameters, their types can be determined quickly by looking at the length of the packet and the packet structure. Service Acknowledgements have to be signed for integrity protection and verifiability of the origin of the parameter. The three fields, SERVICE_OFFER_ID, SERVICE_OPTION and SERVICE_OFFER_HASH, are present in all the three types of service acknowledgements. The service offer identifier received in the service offer from the middlebox is echoed back in SERVICE_OFFER_ID. This checks the freshness of the service offer and protects the middlebox against replay attacks. SERVICE_OPTION is 0 when no information was found in the SERVICE_DESCRIPTION of the service offer. Otherwise, it contains information about the response of the end-point with regards to the service description. For example, the end-point may respond back with its choice from the list of encryption algorithms supported by a VPN box. SERVICE_OFFER_HASH is the cryptographic hash of the corresponding service offer parameter. Service offers are sent in the unsigned portion of the HIP message. The middleboxes have no way to ensure that the end-points received an untampered service offer. Sending the hash of the service offer received earlier helps the middlebox to verify if the service offer was received correctly by the end-point. Figure 5.16 shows the service acknowledgement corresponding to an unsigned service offer.

The selectively signed acknowledgement is however, slightly different from the unsigned counterpart as shown in Figure 5.17. In Section 5.3.3.2 we had shown that in

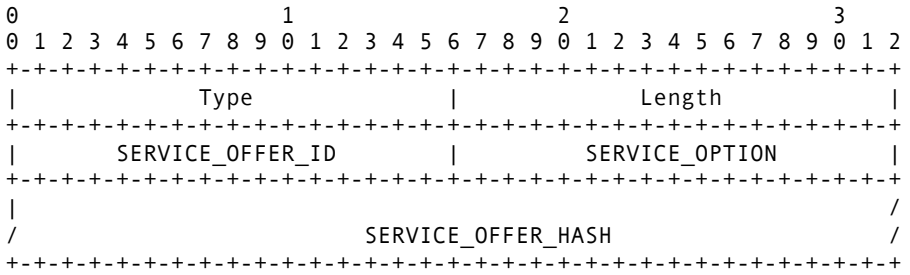

Figure 5.16 Parameter for Unsigned Service Ack

selective signature scheme, some of information items can be removed by the middlebox from the HIP message after processing. Hence, the selectively signed service acknowledgement contains a list of the information items that can be safely removed by the middlebox from the HIP message. A list is important here because it is not always safe to remove all the information items requested in a service offer. This can be better understood with the scenario in Figure 5.6, where 'Z' do not be safely removed at $MBOX_1$ as it is needed by $MBOX_2$ further down the signaling path.

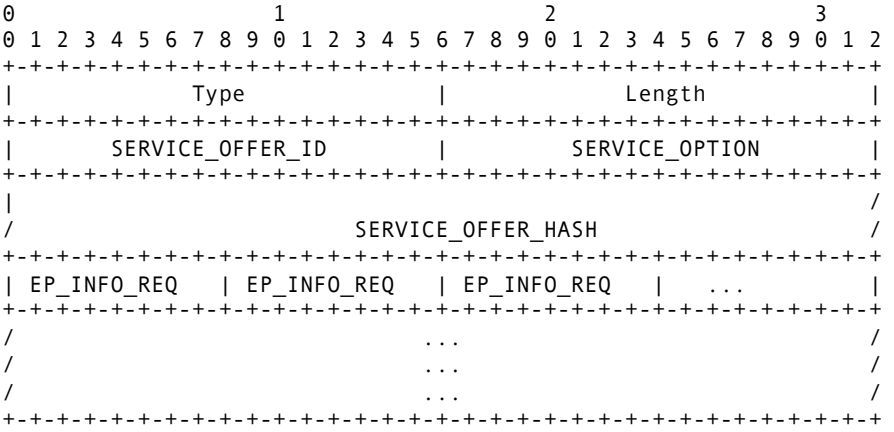

Figure 5.17 Parameter for Selectively Signed Service Ack

In Section 5.3.3.1 we had discussed, how symmetric and asymmetric encryption can be used to protect sensitive end-point information. Both of them rely on encrypting the end-point information using a symmetric key and encrypting the symmetric key with the public key of the middlebox or a key generated from DH shared secret. The encrypted information and the encrypted symmetric key are sent to the middlebox. The encrypted symmetric key is sent to the middlebox with the service acknowledgements (signed service acknowledgements). The layout of a signed service acknowledgement can be seen in Figure 5.18. The length of the key, algorithm used for symmetric key encryption (e.g AES 128-bit), key hint (`KEY_HINT`) and the symmetric key are put together in one field. This is encrypted and added to the corresponding service acknowledgement. `KEY_HINT` is used by the middlebox to locate the information item encrypted with this particular symmetric key. WIthout the `KEY_HINT` the middlebox would have to go on decrypting all the encrypted end-point information items and find the correct one. Also, for a symmetric key encryption unlike asymmetric key encryption it is not possible to determine if the decryption process has been successful or not. Without the hint, the middlebox would have to

decrypt everything and look inside the decrypted information to decide if this is the right one. Key hint helps in avoiding all the mess. The key hint plays an important role in linking the information item to the end-point identity as we see later in the chapter.

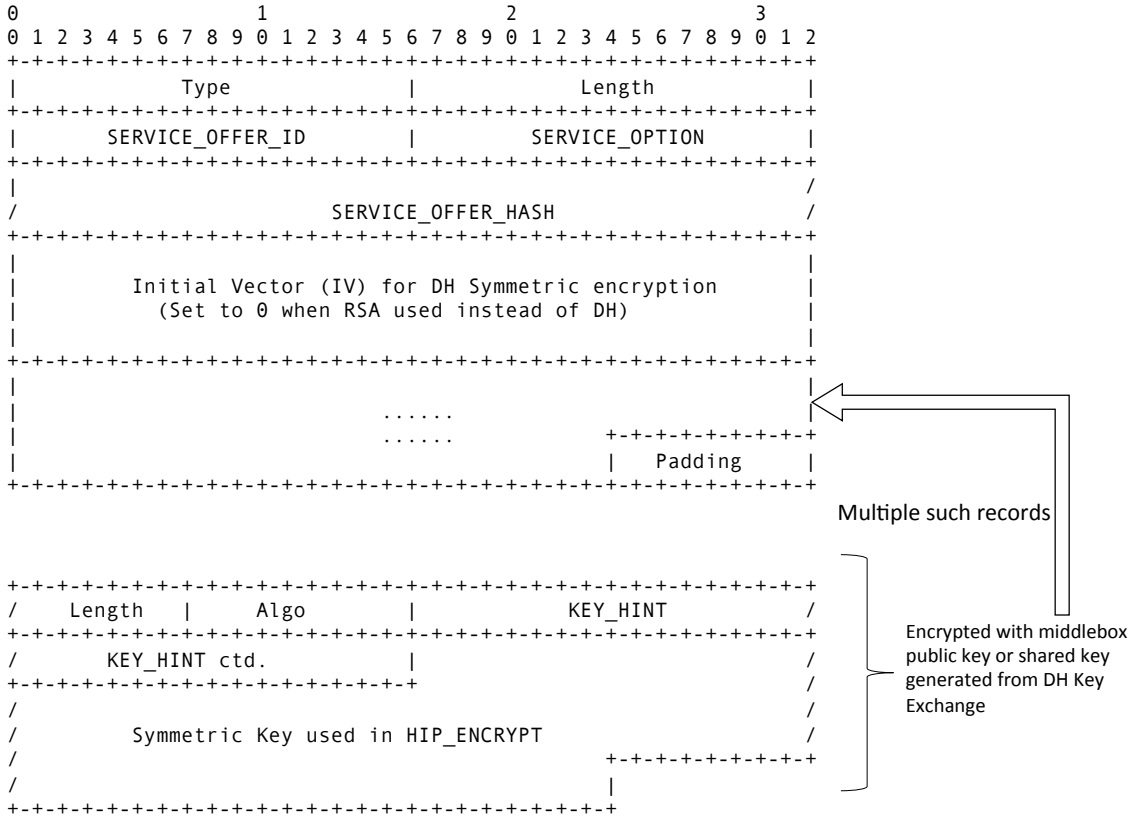

Figure 5.18 Parameter for Signed Service Ack

As we can see in the Figure 5.18, the end-point appends an *Initial Vector (IV)* after the service offer hash. `IV` is set to 0 when the middlebox public key is used to encrypt the symmetric key. If a shared key obtained from the DH Key Exchange is used to encrypt the symmetric key, then `IV` contains the Initial Vector used. The encrypted symmetric key (encoded in the form shown in Figure 5.11) is then appended to the service offer after `IV`. In Section 5.3.3.3 we looked in to a possible scenario, Figure 5.7, where the information items have to be encrypted separately and not together. In such cases, more than one symmetric keys have to be transmitted. Correspondingly, in the acknowledgement on or more of these encrypted symmetric keys are stacked together and appended to the service acknowledgement after the `IV`.

In some cases, the end-point might refuse to provide the information requested by the middlebox depending on it's policy or the trust level with the middlebox. So, the end-point responds back to the middlebox with a negative acknowledgement as shown in Figure 5.19. The end-point can specify a reason for the refusal in `NACK_REASON`. For example, the middlebox needs to authenticate with the end-point.

The middleboxes and the end-points negotiate for the sharing of end-point information. Hence, we have some pre-defined parameters for sharing of user, application or host context.

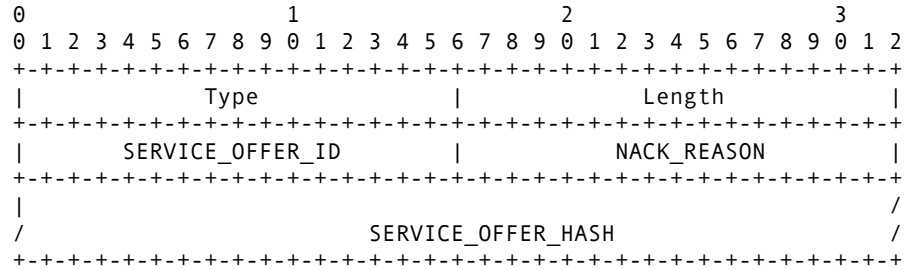

Figure 5.19 Parameter for Service Nack

5.7.1.3 Parameters for User Context

The middleboxes can request for a variety of user context information like the user identity (public). This user identifier can be used by the middlebox to verify and authenticate the user identity. The user identity paramter is primarily based on the host identity parameter as described in RFC 5201 [30] and is necessary to distinguish between host and user identities. Figure 5.20 shows the parameter we have defined for user identity. The parameter type for user ID is 5112. Apart from the public

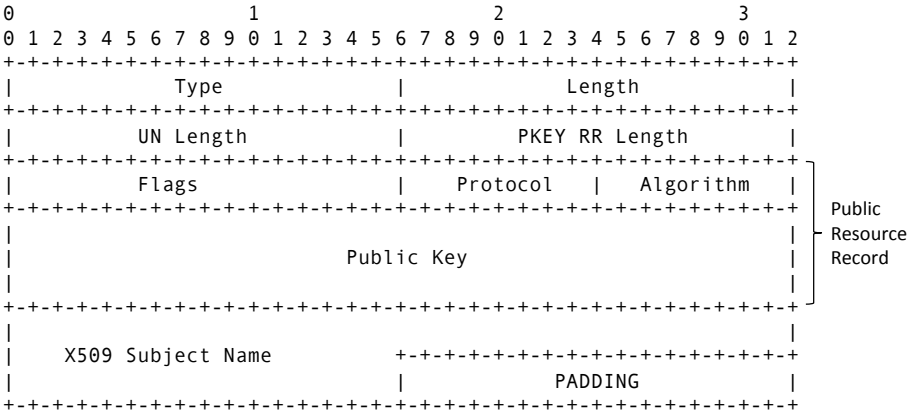

Figure 5.20 Parameter for User Identifier

key resource record as shown in the Figure 5.20, the parameter also contains the DER encoded subject name (based on X.500 global database design) of the public keys (RSA and DSA) and key format defined in RFC 5201-bis for ECDSA keys [28]. Moreover, the user identifier has to be accompanied with the user signature for verification and authentication purposes.

A complete authentication of user identity would require the middleboxes to be in possession of the end-point user certificates. The certificate exchange we have in our signaling handshake is carried over `HIP_ENCRYPTED` parameter (RFC 5201, 5.2.15) and not in plain text [30]. The complete certificate chain may require the middlebox to split it into multiple packets and send each of the parts encrypted over `HIP_ENCRYPTED` parameter. The parts of the certificate belonging to the same chain is indicated using a certificate group. This certificate group is transmitted in the parameter as shown in Figure 5.21

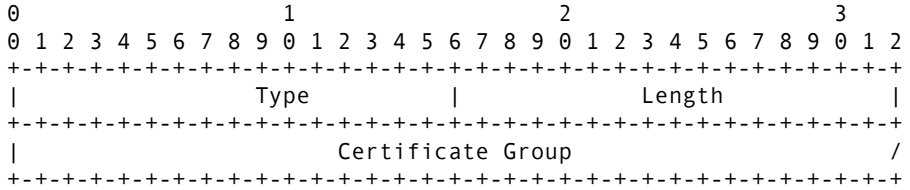

Figure 5.21 Parameter for Information of User Certificate

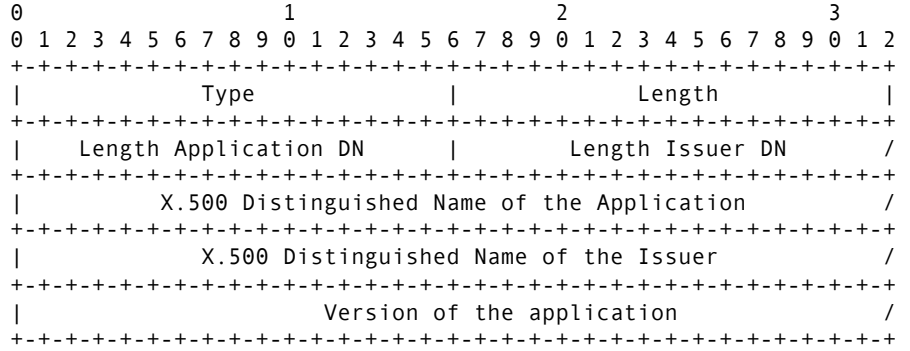

Figure 5.22 Parameter for Application Context Information

5.7.1.4 Parameters for Application Context

In the previous subsection we have discussed about the parameters for sharing user context information. Application context information is equally important in the signaling channel. We would be discussing here two of the most important application context parameters. This parameter is based on the attribute certificate and the context information of the application. In Figure 5.15, we have the layout of *Application Info* parameter. The parameter type is 5116, hence it is in the signed part of the HIP message. Since, the parameter is in the signed part of the message, host signature binds the application context to the host identity. Similarly, a user signature on this message binds the user identity to this application context. This parameter is uncritical and can be ignored by standard HIP implementations. As we can see from Figure 5.22, it contains the DER encoded form of the application name and the issuer of the application attribute certificate. It also contains information about the version of the application which may be important for an application specific firewall.

Apart from application specific information, a middlebox might also require some connection specific information. For example, number of connections opened by an application for resource management purposes like limiting it to a pre-determined value. Figure 5.23 gives the layout for *Application Connection Info* parameter. The parameter type is 5118, which again keeps it in the signed portion of the HIP message. It contains a list of port pairs corresponding to all the connections established by the application between the same hosts. We have mentioned in Section 2.4.3, that HIP aggregates traffic flow over multiple connections between the same hosts. This port pair information is used to communicate port-pair information to the signaling channel. The peer host can also use the port information to generate the respective application context. Since the parameter is in the signed portion of the packet, the

middleboxes and the peer host can bind the host identity and user identity to the information provided.

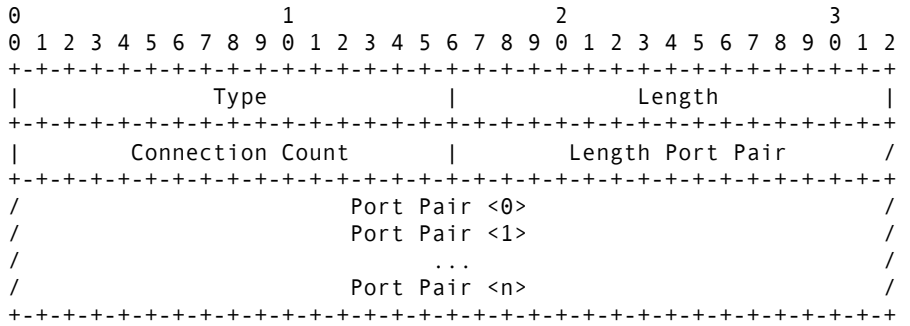

Figure 5.23 Parameter for Port-Pair Information for the Application

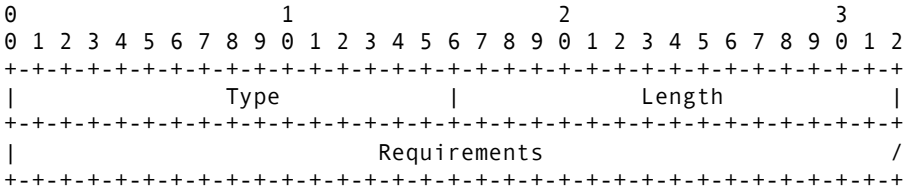

Figure 5.24 Parameter for Application specific Requirements

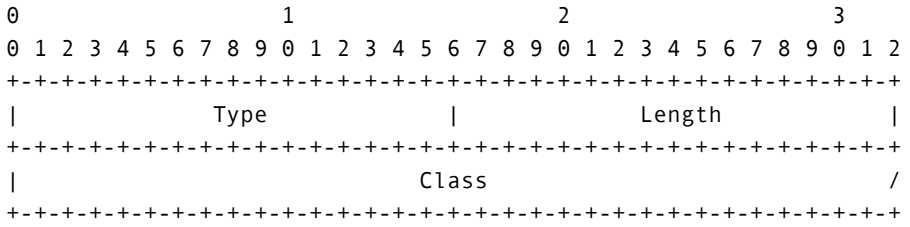

Figure 5.25 Parameter for Application QoS class

We have defined more parameters for application context. The parameters shown in Figure 5.24 and Figure 5.25, contains application specific requirements like bandwidth requirements or more information about the application like QoS class (e.g VoIP application).

5.7.1.5 Parameters for Host Context

We have defined parameters for signaling of host context which will help the middlebox to have more specific policies like operating system supported or the minimum kernel version supported (for security reasons). The parameters shown in Figure 5.26 are self explanatory.

5.7.1.6 Other Parameters

Apart from the parameters for signaling context information we also need some other supplementary parameters for the various schemes of signaling. In Section

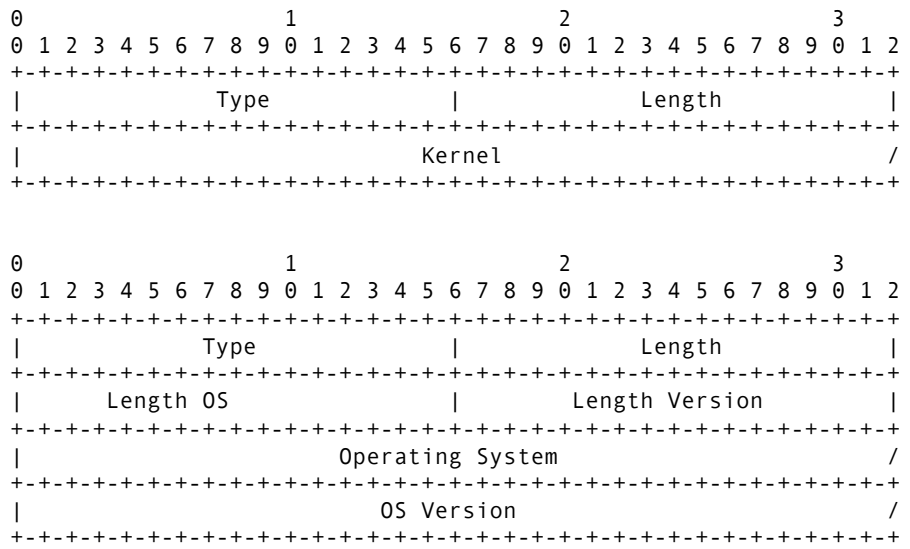

Figure 5.26 Other parameters for Host Context Information

5.3.3.2, we had discussed how certain pre-determined end-point information items can be removed from the signaling channel by the middleboxes. Correspondingly, we also had this information transmitted by the end-points to the middleboxes in the selectively signed acknowledgements. Whenever an information parameter is removed, the hash of the parameter (leaf of the hash tree) is appended at the end of the HIP message. This hash is used again to build the hash tree of the HIP message and computes the selective signature and selective HMAC for authentication of identity and verification of integrity. The layout of the *Hash Leaf* parameter is shown in Figure 5.27. The parameter type is 62508, making it another uncritical parameter. This also means that the parameter is in the unsigned portion of the HIP message. This does not affect the integrity of the message since all the information than need to be integrity checked are already in the signed portion. Also, due to pre-image resistance of the hash functions it is nearly impossible to regenerate the removed info parameter. The parameter contains the position of the hash leaf which can be used by the verifier to regenerate the hash tree.

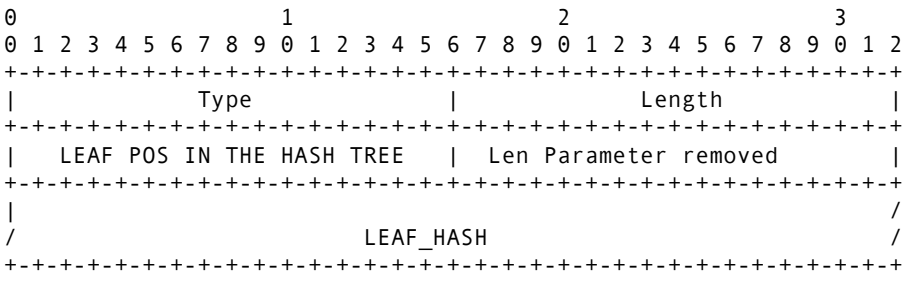

Figure 5.27 Leaf of the Hash Tree

The parameters for Selective Signature and Selective HMAC have the same parameter structure as the HIP Signature and HMAC respectively. However, we have defined new parameter types for them. The parameter type for Selective Signature is 62498 and Selective HMAC is 62502.

6

Implementation

In the previous chapter, we had discussed about our approach and a detailed discussion into our extension of HIP Base Exchange. Our implementation and the integration with HIP is based on the open-source implementation *HIP for Linux*, HIPL. In Section 6.1 we present a brief overview of HIPL. In Section 6.2 we show the interaction between various components of HIP during a Base Exchange with service negotiation. In Section 6.3 we discuss the implementation for service negotiation on the middlebox side. In Section 6.4, we discuss the implementation and various optimization in the end-host side.

6.1 HIP for Linux (HIPL)

HIP for Linux (HIPL) is an open-source implementation of Host Identity Protocol. The first stable version of HIP, 1.0.0, was released in 2006 [20] and since then it has been actively maintained with latest HIP specifications by the Aalto University and RWTH Aachen University. HIPL consists of three separate user space applications, namely, the *HIP Firewall* (`hipfw`), the *HIP Daemon* (`hipd`) and the *HIP Configuration tool* (`hipconf`).

The `hipd` handles the HIP specific messages with HIP enabled peers and existing HIP functionality for connection establishment, maintenance and teardown. It maintains and updates the state of existing HIP associations. The `hipd` also maintains a local socket for inter-process communication with the `hipfw` and the `hipconf`. The `hipd` uses messages of protocol type 139 to communicate with other HIP enabled peers. The protocol type 139 is defined by IANA to denote HIP Control messages and 17 for udp encapsulated HIP Control messages.

The `hipfw` performs filtering of HIP-based traffic in user space similar to the `iptables`-like filtering in kernel space. It intercepts the packets from network stack with the help of `libipq` and transfers them to userspace. `libipq` is a userspace API to

the kernel space `iptables` implementation. The `hipfw` has been extended to support extensions for HIP Proxy implementations, userspace IPSec implementations and LSIs. Furthermore, the `hipfw` uses inter-process communication to trigger HIP control messages and to retrieve and update resulting state information.

The `hipconf` provides a command-line utility to mange and configure the parameters governing the behavior of `hipd` at runtime. It also allows for looking up of state informations maintained by the `hipd` for active HIP associations. However, it is not significant for the scope of this work.

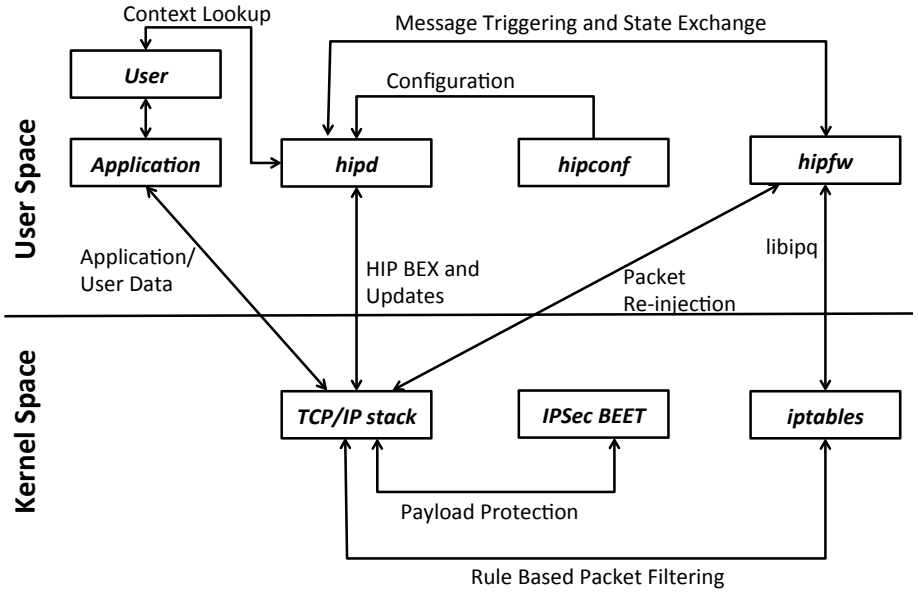

Figure 6.1 An Overview of HIPL implementation and interaction between various components.

Figure 6.1 shows the interaction between different components of HIPL. It also depicts the interaction of these components to external components like an interaction between a user-level application and `hipconf`.

The initial implementation of HIPL was in the kernel space, but the acceptance of IPSec BEET mode into the Linux Kernel 2.6.28 allowed for the migration of HIPL to user space. HIPL code is written in C and dependent on standard C libraries along with the some external libraries like `libcrypto` of OpenSSL.

6.2 Interaction between different layers of HIPL

We have given a brief overview of the functionalities of the different layers of `HIPL` in the previous section. Our extension comes to play as soon as an application requests for a connection establishment between the peer-hosts. Consider the situation depicted in Figure 6.2, where an application, `APP1` at the *Initiator* wants to connect to the application, `APP2` at the *Responder*. In Section 5.5.3, we have already discussed why we need a *three way exchange* for service negotiation in this scenario. We represent the three messages with the three colored lines as shown in the figure. The box on the left hand side represents the `HIPL` layers on the Initiator, the box in the

middle shows the HIPL layers working in the middlebox and the box on the right represents the HIPL layers functioning in the Responder.

When APP1 initiates a connection request, it is captured by iptables based on a set of rules (TCP or UDP) and libipq transfers these packets from the network stack to the user space (hipfw). The hipfw receives the connection request information and checks in the signaling database if there exists an active connection between the peer-hosts specified in the connection request, then the connection request is dropped. Otherwise, the hipfw checks with the policy engine if the connection request is allowed by the policy rule. If policy decision allows the connection, the hipfw sends all this information to the hipd via user message.

The hipd at the initiator checks if there exists an active host association between the hosts specified in the request. If not then a new signaling handshake is triggered, otherwise an update is triggered. Either way, after the triggering of the handshake, the hipd looks up the connection context. The first message represented by the blue line triggers the handshake.

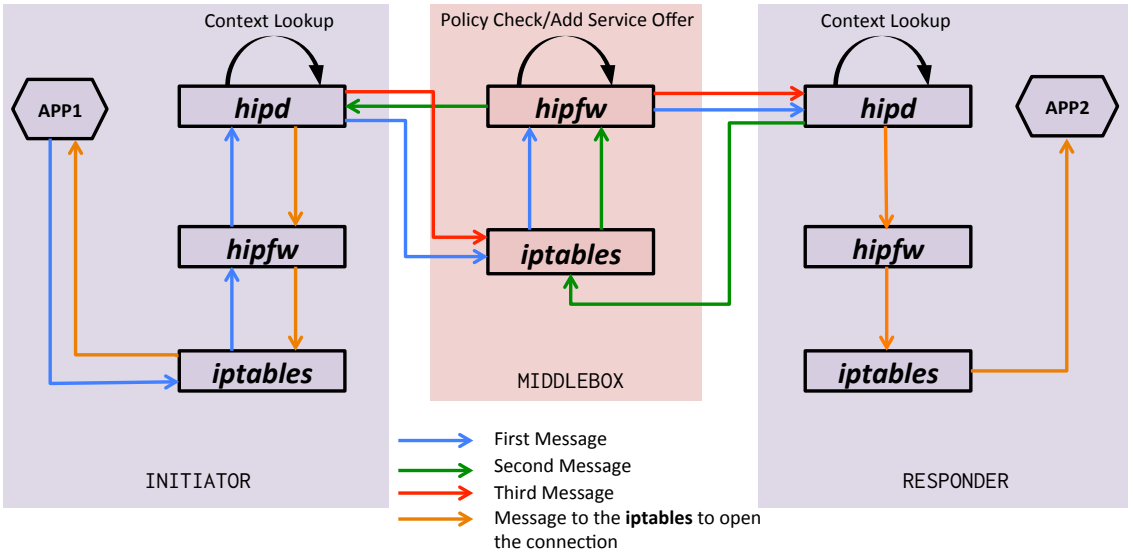

Figure 6.2 Interaction between various layers of HIPL during a signaling handshake

The first message from the hipd of the Initiator is received by the iptables in the middlebox and subsequently captured and transferred to the user space for message processing by the middlebox hipfw. The middlebox checks if the connection is allowed by the policy engine and subsequently, adds the information it requires from the Responder as a service offer to the first message.

The hipd of the *Responer* receives the HIP message, it verifies the service offer and performs lookup of context information. The context information that were requested in the service offer are added to second message (green color line). The second message is sent to the *Initiator* thereafter.

The second message is intercepted by the iptables in the hipfw and transferred to the user space in the middlebox for processing. The middlebox verifies the information received in the second message with the policy engine and correspondingly adds the information required from the *Initiator* to the second message.

The hipd at the *Initiator* receives the second message, verifies the service offer and adds the context information as required in the service offer. Since the Initiator had already performed a context lookup before, it has all the information available to add in the response. The response to the second message along with the informations requested is added to the third message (red colored line) and sent back to the *Initiator*. As soon as the third message is sent by the *Initiator*, the hipd sends a user message to the hipfw to set the right iptables rules in order to allow the connection through.

The third message is again intercepted by the iptables of the middlebox and transferred to the user space for the processing by the middlebox hipfw. The hipfw verifies the informations received in the third message similar to the second message. It then passes the third message to the signaling channel in order to be received by the *Responder*.

The third message is received by the hipd of the *Responder* and the handshake is completed. The hipd sends a user message to the hipfw to set up the iptables rules and open the connection for APP2.

This was a bird's eye view of the interaction between HIPL layers during a service negotiation. However, as we saw above the context lookup and service negotiation need additional functionality to be integrated with the HIPL layers. We discuss the various functionalities that were integrated to handle the signed, unsigned and selectively signed service negotiation in Section 6.4.1 and Section 6.3.3. Also, the context lookup required integration of netstat functionalities to the hipd layer which we discuss in detail in Section 6.4.1. We also describe the working of policy decision engine in Section 6.3.1. Our implementation is based on the work of Ziegeldorf et. al. which helps us to reuse the functionality that had already been implemented [42].

6.3 The Middlebox Architecture

In a privacy sensitive network, the middleboxes have to rely on negotiating for information. Service negotiation in our design starts with the middlebox requesting for end-point information in a service offer and the end-point acknowledging the offer by providing the information requested. The functionality of middleboxes is implemented as an extension to hipfw. The libipq captures packets from the network stack and brings them to the user space, hipfw for processing. So, we implemented the security layer as an extension of hipfw.

We implement our functionalities in a firewall which uses the rules in the policy engine to take decisions on incoming traffic. And in case more information is required for decision making, the middlebox requests for those in the outgoing traffic. The security layer integrated with the hipfw in our implementation also provides the middlebox with necessary tools to deal with the signed and selectively signed schemes. The modularization tools available for the hipd are not available to the hipfw, hence our implementation is not modular but is tightly integrated with the hipfw. Because of the lack of modularization, the modules in hipfw can't be dynamically loaded and unloaded. However, we have taken care of the separability of our extension wherein the hipfw can be run with or without our signaling extension.

6.3.1 Policy Engine

The `iptables` rules do not provide the middlebox firewall with sufficient tools to take decisions based on anything other than packet header information. Since, HIP does not have a mechanism to take *policy decisions* based on the end-point identifiers (host, user or application), we implemented a separate *policy engine* that could work on upper layer identifiers.

The `hipfw` if booted with the signaling extension, loads a *policy file* containing a set of rules containing specific details about which connections to drop or allow based on cryptographic identifiers. Additionally the set of rules also contain more detailed policies governing the connection, for example the number of connections permitted for the application.

The *policy file* is written in an easy to read format specified by `libconfig`. The middlebox firewall in our implementation merely acts a forwarder to the signaling traffic hence the rules for incoming and outgoing connections are not very important. Figure 6.2, shows a sample policy file with multiple rules based on host, user or application identifiers.

```
rules_fwd = (
  {
    host =
    {
      hit = "2001:001c:df46:b968:a716:12e9:2038:24eb";
    };
    application =
    {
      name  = "/C=DE/ST=NRW/L=Aachen/O=RWTH/OU=COMSYS/CN=CLIENT/emailAddress=client@email.com";
      connections = 10;
    };
    user =
    {
      name = "/C=DE/ST=NRW/O=RWTH Aachen University/CN=Pisa1/emailAddress=pisa1@gmail.com";
    };
    target = "ALLOW";
  },

  {
    user =
    {
      name = "/C=DE/ST=NRW/O=RWTH Aachen University/CN=Pisa2/emailAddress=pisa2@gmail.com";
    };
    target = “DROP“;
  }

);
```

Figure 6.3 A sample Policy File

The policy depicted on the Figure 6.3 will drop all connections from the user (`name = "/C=DE/ST=NRW/O=RWTH Aachen University/CN=Pisa2/emailAddress=pisa2@gmail.com"`). The user identifier here follows a X.500 global namespace design. However, a connection from host (`hit="2001:001c:df46:b968:a716:12e9:2038:24eb"`) is allowed if the application that triggered the connection is `"/C=DE/ST=NRW/L=Aachen/O=RWTH/ OU=COMSYS/CN=CLIENT/emailAddress=client@email.com"` and the user running the application is `name = "/C=DE/ST=NRW/O=RWTH Aachen University/CN=Pisa1/emailAddress=pisa1@gmail.com"`. However, the rule has an additional restriction that the number of permitted connections for these set of identifiers is 10. If the number of active connections is more than the limit, the subsequent connection requests will be dropped. Similarly, the cryptographic identifiers and additional higher layer rules on the connection can be combined together for more descriptive policies. It is implicit from the policy rule that whenever an identifier is

used in the policy rule, the middlebox requests for the corresponding certificates if it does not already have it.

Sometimes, there is not enough information available within the HIP Control Channel messages for an accurate decision on the policy. For example, for the first policy from the sample presented in Figure 6.3, a new connection might not contain information about the application which triggered the BEX. The middlebox can then ask the concerned end-point for the application information (identifier) and the number of active connections of this application. This information request can be added the corresponding HIP Control Channel messages (`R1` or `I2`).

On receiving a response, the middlebox extracts the end-point information from the packet. If the information available is sufficient to satisfy the policy rules, the connection is accepted. The advanced mechanisms like signed and selectively signed service negotiation schemes require additional functionality at the middleboxes for handling cryptographic processing (encryption/decryption). We would discuss those mechanisms in the next few subsections.

6.3.2 ECDH for Signed Service Negotiation

The signed service negotiation allows the end-point to respond to the information encrypted. The `hipfw` by its own does not support decrypting any parameter. So, we had to extend the functionality to support various encryption and decryption algorithms. Our implementation supports RSA for the asymmetric encryption schemes and AES (128-bit and 256-bit, 3DES etc.) for the symmetric encryption schemes. The middleboxes are computationally less powerful devices, hence any cryptographic processing takes a significant amount of time for them. Hence we emphasize the use of symmetric key algorithms over asymmetric key ones. In spite of the computationally intensive asymmetric key operations, we still provide them for the sake of completeness. Symmetric key operations are simple row and column shifts and $\oplus$ operations, hence ideal for the usage with the middleboxes.

In Section 5.5.3, we presented how signed service negotiation with symmetric key encryption required setting up of a shared key through a key exchange algorithm like Diffie Hellman. This shared key is used to decrypt the symmetric key which was used by the sender to encrypt the end-point information. However, the generation of shared key using Diffie Hellman requires multiplicative and modular arithmetic operations on large prime numbers. Such operation are computationally intensive and hence slow on middleboxes. Typically, shared key computation for a Diffie Hellman key based on `1536-bit modp group` suite takes 280 ms in the middleboxes. Although the higher key sizes provide better performance, it becomes more and more difficult to accommodate them in the HIP Packet. Moreover, the shared key generation has to be done twice for each Base Exchange once for the Initiator and once for the Responder. The numbers look worse when there are multiple on-path middleboxes. This computationally intensive task severely degrades the performance of middlebox. In order to circumvent this problem, we use Elliptic Curve Diffie-Hellman rather than the regular Diffie-Hellman.

HIPL supports the usage of `384-bit modp group`, `OAKLEY group 1`, `1536-bit` and `3072-bit group`. RFC-5201-bis-08 however, has added `NIST P-256, NIST P-384`

and NIST P-521 as supported Diffie Hellman suites for key exchange. These new Diffie-Hellman suites are based on Elliptic Curve Cryptography, but they are still not supported in the existing HIPL implementation. We have extended the HIPL implementation to support NIST P-256 for Elliptic Curve Diffie Hellman (ECDH) which takes around 23 ms for shared key generation.

6.3.3 Removing Info-Parameters in Selectively Signed Service Negotiation

In Section 5.5.2, we described the method for Selective Signing where information parameters can be removed by the middlebox after processing. The acknowledgement to a selectively signed service offer, contains information about the parameters that can be safely removed by the middlebox. In order to selectively sign a HIP message, the sender has to build a hash tree from the HIP message as shown in Figure 5.24 and sign the root of the hash tree. In order to verify the selective signature, the verifier needs to re-build the hash tree and verify the signature on the root of the hash tree. So, when a middlebox removes an information parameter, it copies the leaf of the hash tree corresponding to this parameter at the end of the HIP message along with its position in the hash tree. Next time an entity needs to verify the HIP message, it rebuilds the hash tree and inserts the leaf at the position specified and then computes the signature on the root of the hash tree.

The processing required to determine the info parameters that can be removed safely is done at the end-points since the end-points are usually computationally more powerful. Also, in order for the middlebox to determine the parameters that can be removed safely requires it look into other service offers into the signaling channel. This inspection of HIP message for service offers can be in direct violation of privacy requirements. Given the information parameters to be removed, the process requires few additional pointer arithmetic operations and hash computation, both of which are highly optimized for middlebox hardware.

6.3.4 Connection tracking

The previous implementation had a connection tracking database which maintained the information about active connections as well as the end-point context information. We simplified the connection tracking database to only maintain information about the state of the active connections. In accordance to our privacy specific architecture, the middleboxes use the context information for policy verification but do not store it. This also helped in reducing the memory consumption.

6.4 The End-Point Archtecture

The End-Point architecture has been implemented in two layers: the policy and security layer implemented as extension to the `hipfw` and the signaling layer implemented as extension to the `hipd`. The functionality of the policy and security layer to capture outgoing connection request from the network and divert them to the

signaling layer for processing and receive the incoming connection request from the signaling layer and verify it against the policy engine. It also maintains a connection tracking database for active connections. The signaling layer on receiving the outgoing connection request from the policy and security layer triggers a handshake with the other side to establish a secure channel for the connection request. The `hipfw` already has a connection tracking database and has support for capturing packets from the network stack using `iptables`. The `hipd` already provides support for signaling between HIP-enabled peers based on HIP Base Exchange. We discuss the extensions build onto the HIP components, `hipd` and `hipfw` in the next few subsections.

6.4.1 HIP Daemon

The extension to HIP Daemon consists of two major parts. Firstly, we integrate the new parameters and messages discussed in the Section into the HIP Control Channel. Secondly, we extend the HIP Base Exchange to incorporate different modes for service negotiation. Additionally, we have optimized the context lookup and overall packet processing for improvement in signaling. The `hipd` has been extensively modularized which offers us with great advantage to load and unload various functionalities supporting our architecture dynamically. This also helps in keeping the functionalities of our extension separate from the HIPL implementation and allows introduction of new association states in the HIP Control Channel. We also extend the supported Diffie Hellman suites to include Elliptic Curve Diffie Hellman.

6.4.1.1 General Packet Handling

The HIP daemon depends on packet handlers for processing of incoming HIP message to generating a response to the HIP message. HIPL modularization framework allows for registering of various packet handlers which are assigned different priorities. These handlers are triggered depending on the HIP association states and HIP Packet (`I1`, `R1` etc.) and the assigned priorities determine the order of execution. The modularization framework allows extension like ours to register additional packet handlers which can be used to handle and process HIP Control messages. It also supports the registering of new parameters and packet types. Further, modularization allows our extension to be loaded and unloaded dynamically.

We add the required packet handlers for processing of HIP messages, context lookups, handling of service negotiation in the Extended HIP Base Exchange and HIP Update Exchange. From the HIP Base Exchange perspective packet handlers are added as follows

1. Handling of service offer received in the `R1` message, addition of context information (application information, host and user identifiers) and user signature on outgoing`I2` message

2. Verification and Validation of `I2` message (only for selectively signed service negotiation)

3. Handling of service offer received in `I2` message, context lookup if necessary, addition of context information and user signature on outgoing `R2` packet

4. Verification and Validation of `R2` message (only for selectively signed service negotiation)

5. Handling and creation of Update packets and the handling of service negotiation within the UPDATE exchange.

Here, handling of service offer includes verification of service offers received with the policy engine, general packet processing and additional functionalities like grouping of information parameters received for signed service negotiation.The first four could be implemented within the modularization framework of HIPL while the handling of updates could be implemented modularly along with the HIP Update extension.

6.4.1.2 Connection Context Lookup

The connection context has to be looked up each time a new connection request comes through. The connection tracking database determines if there already exists an active connection corresponding to the host association, then an Update Exchange is triggered otherwise a new Base Exchange is triggered. The context information requires looking into the user and the application identity and the corresponding network layer information like IP addresses and port information. In order to do a context lookup, the previous implementations used `netstat` utility to retrieve information about the application, the user who ran the application and the ports being used by the application. This would require forking a shell to execute `netstat`.

We have optimized the context lookup significantly by removing the forking of shell for `netstat` and adding a new module which directly implements the `netstat` functionality within the `hipd`. This saved us the processing and the delay of forking a new shell. The gains on the Initiator's side for `netstat` lookup is not significant (order of 2 ms), but the gain on the Responder's side is significant (order of 10 ms). The `netstat` utility looks in the `/proc/net/tcp` and `/proc/net/tcp6` for connection information. So, instead of going through the unix style `netstat` lookup, we extract the information directly from these files. The Initiator has to lookup the connection information corresponding to the outgoing local port and the destination port on the Responder, while the Responder has to only look for the incoming connection to a local port (listening port). The parsing of the `/proc/net/tcp` and `/proc/net/tcp6` files for the information related to the ports provided, takes more time when two ports (source and destination port) have to be matched and less time when only one port (listening port) has to be matched.

Furthermore, the host context information remains static for the lifetime of the HIP layer (`hipd` and `hipfw`). Hence, it would not affect the freshness of host context information, if host the context lookup is performed at the time bootstrapping of `hipd`.

The rest of the functionality for context lookup remains same as in the previous implementation. The connection information retrieved from `netstat` lookup provided information about the user who ran the application and the location of the

application binary. The hash of the application binary is verified with the hash that was shipped with the attribute certificate of the application. The user information is used to locate the user certificate, which provides further information about the use identifier and other such data.

6.4.1.3 Signed and Selectively Signed Service Negotiation

We have already discussed in detail how the signed service negotiation results in the sending the end-point information encrypted and the various symmetric and asymmetric key schemes to secure it. For either schemes, the information requested across multiple service offers need to grouped and encrypted separately for the reasons we discussed in Section 5.5.3.1.

Grouping of Information
After the policy decision on which of the information requests are eligible for a response, the end-point needs to organize the information requested into groups before encrypting them. The grouping of information can be further understood by the Figure 6.4. After the service offers have been analyzed, the `hipd` already has a list of the information items that have been requested and the middleboxes who have requested them. This information is organized further into a table which is indexed by the information items that were requested, where each member of the first column is linked to the member in the second column of the same row. The second column contains a list of the middleboxes who requested for that particular information (in the first column of the same row). In the second run over the table, we look into the entries in the second column. For all of the entries of which second column is the same, the rows are merged as shown in the figure.

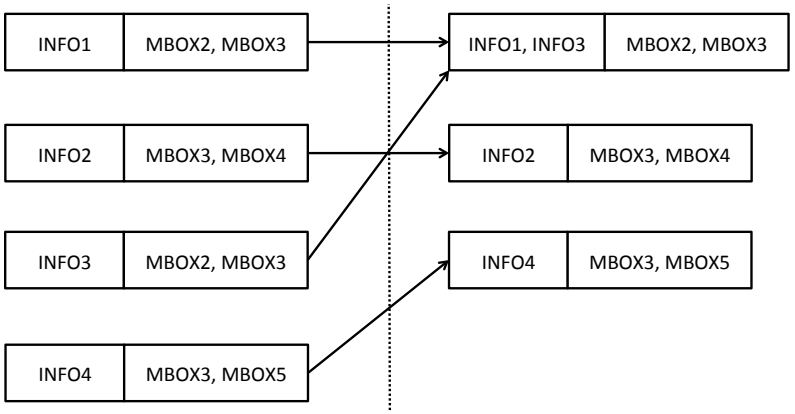

Figure 6.4 Grouping of Information Items before Encryption

We achieved two things with this kind of grouping as described. Firstly, the information items pertaining to one group contain only the middleboxes who have requested for them. So, encrypting this group would ensure that only the middleboxes who have requested for the information can decrypt them. Secondly, any two groups of information are mutually exclusive to each other and our solution is *optimal* for the splitting of information into the groups. The argument for optimality rests from looking at this problem in the form of a *bipartite graph*. The first round of grouping distributes the information items requested and the middleboxes who requested

them into a *bipartite graph*. The second round merges the vertices representing information items in a way that produces a *maximal and perfect matching bipartite graph*.

Removal of Information Items
"Grouping of Information" is important for the encryption scheme in the signed service negotiation scheme. In selectively signed service negotiation, a middlebox should remove the information parameters after processing. Since these information parameters may be required further down the signaling path, it is important that the middlebox should only remove the parameters which are safe to remove. The processing of determining which info items can be removed safely by which of the middlebox is best done at the end-point. Figure 6.5 depicts this process in a tabular form. The left hand side of the figure is constructed similar to the previous approach (grouping of information). However, if we ensure that the order in which the middleboxes are added to the second column is the same order in which the service offers were added to the HIP message our solution is easy. A middlebox which is at the furthest point in the signaling path adds the service offer first, followed by the second last middlebox in the signaling path. So, the middlebox lying first in the second column is selected for removing the information item safely. Our solution waits till the last point where the information item is required before removing it completely from the signaling channel. The table generated by this process is shown in the right hand side. The first column of the table can be further merged producing a list of information items than can be safely removed by that particular middlebox.

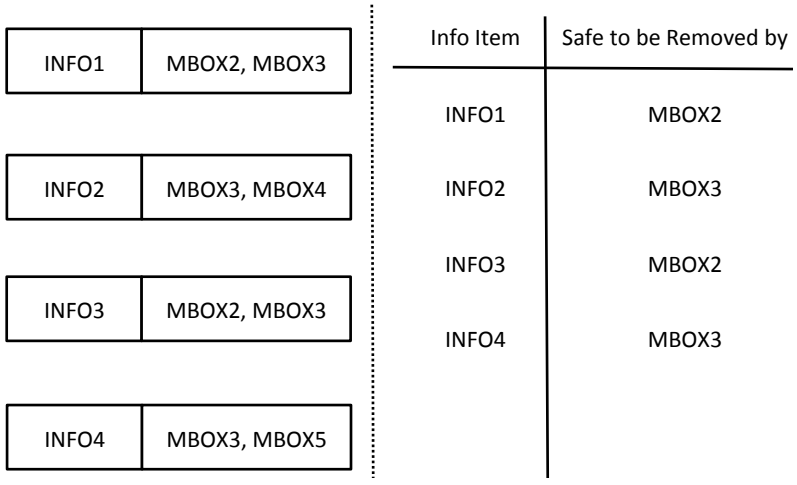

Figure 6.5 The Information Items which can be removed safely

6.4.1.4 Connection State

The `hipd` maintains the connection state in a *host association database*, which contained one entry per association. The implementation of Ziegeldorf et. al. extended the *host association database* to the granularity of connection [42]. The modularization framework of *hipd* further allowed us to register additional states per host association. We further extended the `hipd state` to maintain more information required for our Base Exchange,

- **Middlebox Certificates:** Middlebox certificates are especially important in selectively signed and signed service negotiation. A typical certificate chain maybe large enough to be received in multiple messages or it may be installed locally at the time of configuration. These certificates need to be cached so that the service signature can be verified in the selectively signed service offer and the public key from the certificate can be used for encrypting the end-point information in the signed service negotiation.
- **Service NACK:** In a typical Base Exchange, the end-point can request for the middlebox certificate before trusting it. To keep track of the middleboxes to which a certificate request has been sent, the end-point saves the *Service Offer ID* in the `hipd state`. The middlebox in it's response replays the *Service Offer ID* along with the certificate group of the certificate chain. This protects the end-point from replay attacks where an old negotiation could be replayed.

The connection contexts that are stored in the `hipd state` can cleared after the association is complete. The end-point has to always lookup for connection context whenever a hip association establishment is triggered, hence storing the connection context (except for host context) persistently is not a great idea.

6.4.2 HIP Firewall

The policy decision and security layer is implemented as an extension to the `hipfw`. The `hipfw` could capture packets from the network stack using `iptables`. These packets were transferred to the user space for further processing by the `hipfw`. The `hipfw` checked if there exists a connection between the peer hosts in its signaling database and sends this information along with the port-pairs (source port on the initiator and the destination port on the Responder) to the `hipd`. Upon completion of the Base Exchange, the `hipd` again sends a user message to the `hipfw` so that it can add the new connection state to its signaling database. Our work on the `hipfw` was simplification of the signaling database and the policy decision engine.

6.4.2.1 Signaling Database

The implementation from Ziegeldorf et. al. of the signaling database stored the connection context along with the connection association information in the signaling database [42]. Connection context lookup before passing the connection request to `hipd` is not necessary since, not always does the signaling channel needs to provided with the context information. Moreover, the saving of connection context in the `hipfw` and sending it to the `hipd` as a user space message is not very optimal. We simplified the signaling database to only store the information about the hip association and the port pairs (for each association) and moved the context lookup to the `hipd`.

6.4.2.2 Policy Engine

The policy engine in the `hipfw` here at the end-points works similar to the policy engine in the `hipfw` at the middleboxes except for the end-points there will rules

for incoming and outgoing connections. We have simplified the layout of the policy file to include more descriptive rules like the number of connections permitted for a particular hip association.

6.5 State of the Implementation

We have integrated all the above described functionalities into the `HIPL` architecture. Our work is built on top of the work from Ziegeldorf et. al.[42]. Along with the implementation of functionalities for service negotiation, we have optimized various aspects of the architecture like lookup of context information. The service negotiation mechanism incorporated by us is specific to our implementation and other similar implementation may have to make their system compatible with ours. However, the modularization framework of `hipd` still permits other implementors to look in our design and implement the design in their own specific way without breaking compatibility with our implementation.

The integration of blinding of `HIP Base Exchange` has been left at conceptual level. There is already some support in the `hipd` layer for encrypted Host Identifiers (symmetric key approach). With the modularization framework of the `hipd`, blinding can also integrated without breaking compatibility with existing implementations.

The implementation of our design into `HIP` makes it easier to evaluable the performance of our proposed system.

7

Evaluation and Discussion

In this chapter, we analyze the performance of our implementation and provide a discussion on the results. In the first section, we discuss the design of our test setup and the measurement framework. In the subsequent sections, we present the results of the performance measurements and analyze the measurements. Finally, we analyze the congruence of our implementation to the design goals of Section 5.1.

7.1 Performance Evaluation

The users should be able to use our system without any significant delay or obstruction. Hence, it is important that our framework does not introduce any undesirable delays or unacceptable load on the user's system. We evaluate our implementation in regards to the overhead our extension generates when compared to the HIP Base Exchange. We quantify our results based on the measurement framework available in the HIPL implementation. We present our test-bed setup in Section 7.1.1. In Section 7.1.4 to Section 7.1.10 we present the evaluation results of the mechanisms we discussed in design analysis (see Section 5.5). Furthermore, we discuss the overhead of our extension with respect to the standard HIP BEX and SEAMS [19].

7.1.1 Test setup

Our test setup consists of two end-hosts, two high-end firewalls and two consumer-grade firewalls connected to the same switch. The two end-hosts communicate with each other over a logical path of 2-hops. We had two alternate logical paths as shown in Figure 7.1. The first logical path between the two end-hosts passed through the two consumer-grade firewalls. The second logical path had the two high-end firewalls in between.

The end-hosts were standard desktop PCs equipped with Intel Core i7-870 CPUs (quad cores running at 2.93 GHz) and 4GB DDR3 RAM. They were running Ubuntu

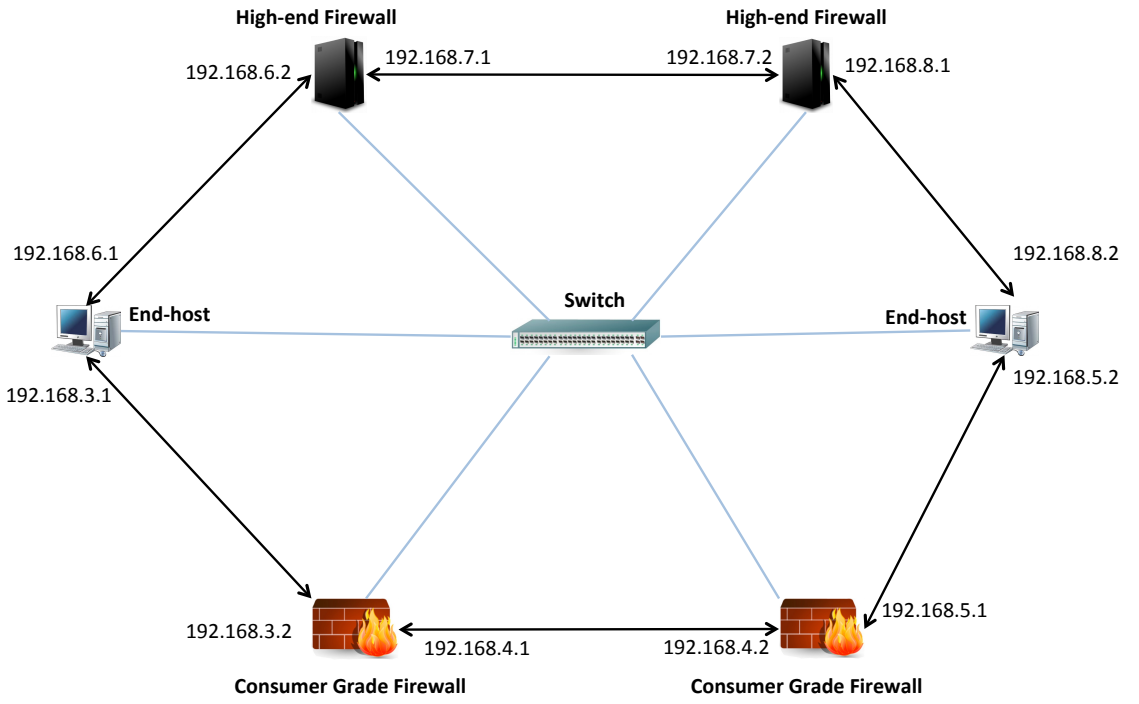

Figure 7.1 The test setup

10.10 with the kernel version 2.6.35. We used machines with the same configurations to model high-end firewalls. Consumer grade firewalls were modeled using PC Engine ALIX boards, equipped with AMD Geode processors running at 500 MHz and 256 MB DDR SDRAM. The ALIX boards were running a custom configuration of a vanilla Linux kernel version 2.36.1 and Ubuntu 9.04.

The end-hosts and the high-end firewalls had 1Gbit/s network adapter installed on them. They were connected to the same switch which also operated at the same speed. However, the consumer-grade firewall running on ALIX boards had 100Mbit/s network adapter installed. All the machines were connected using only one network interface, hence realizing the full-duplex capabilities of the network adapter. The machines were set up to be adjacent to each other on the logical path, by configuring them to be on the same subnets. Figure 7.1, shows the network setup, the resulting configurations of IP address and the two alternate logical path through which traffic could be routed. The two corresponding network scenarios were: the entire traffic between the end-hosts is routed logically through the high-end firewalls and the other where the traffic was routed through the consumer-grade firewalls. Depending on our use cases, we configured the high-end firewall devices and the consumer-grade firewall devices to act as passive packet forwarders or as active SEAMS aware middleboxes.

All of the devices show in the Figure 7.1, had a public-private key pair and certificates (RSA and ECDSA) for the user and the host installed on them. However, for the sake of simplicity, we discuss the measurements taken with RSA 1024-bit keys for the user and the host. Apart from user and host identifiers, the end-hosts also had the static Elliptic Curve Diffie-Hellman (static ECDH) public key of the middleboxes based on NIST-P-256 curve installed on them (see Section 5.5.3.3). Furthermore, we would be doing our evaluation with ECDH based on NIST-P-256 curve rather

than regular Diffie-Hellman (based on 1536-bit modp group). We use ECDH here, because the computation of shared secret is faster with smaller key sizes.

7.1.2 Network Properties

We measured the delay of our network setup as depicted in the Figure 7.1 using the `ping` utility. The average round-trip time between the two End-Hosts was 0.807 ms for the logical path through the high end firewalls and 0.559 ms for the logical path through the consumer grade firewalls. The round-trip times have been aggregated over 100 measurements. However, these RTTs were not interesting for our use case scenarios because `ping` is situation and network dependent

7.1.3 Measurement Framework

We had setup the evaluation framework by setting up the various measurement points in our implementation. The measurement points were used to determine the computational time consumed by the end-host for a certain operation in the packet processing. The time measurements were based on the utility *gettimeoftheday* which has a resolution of 1 μs on the end-hosts with Intel Core i7-870 CPUs and 11 μs on the ALIX boards. This resolution is sufficient for measuring the atomic components we define on the `hipd` and `hipfw`. The measurements framework takes the measurements and keeps it in the memory till an idle time is reached and writes the values to the disk only in the idle time. This ensures that the measurement framework does not obstruct the processing at the `hipd` and `hipfw` and has minimal effect on the measurements themselves. However, the measurement framework has overhead of its own.

7.1.4 Test Procedure

We conducted our evaluation in the following steps. Firstly, the test bed was setup and the logical path verified using the `traceroute` utility. Thereafter. the latency on the path was determined using `ping`. After the network setup was complete, we started the `hipd` and `hipfw` on the both the end-hosts and the `hipfw` on the middleboxes (firewalls). Once the signaling framework was setup, a simple client application was made to establish a TCP connection (using server's HIT) from one end-host to a simple server application running on the other end-host. The TCP connection establishment would be intercepted by the local `hipfw` (see Section 6.2) and thereby trigger a HIP Base Exchange.

7.2 Base reference: Overhead of HIP

We start our discussion with a simple Base Exchange between the two end-hosts with the signaling channel routed through the logical path. In this case, we consider the path through the consumer-grade firewalls where these on-path devices act as

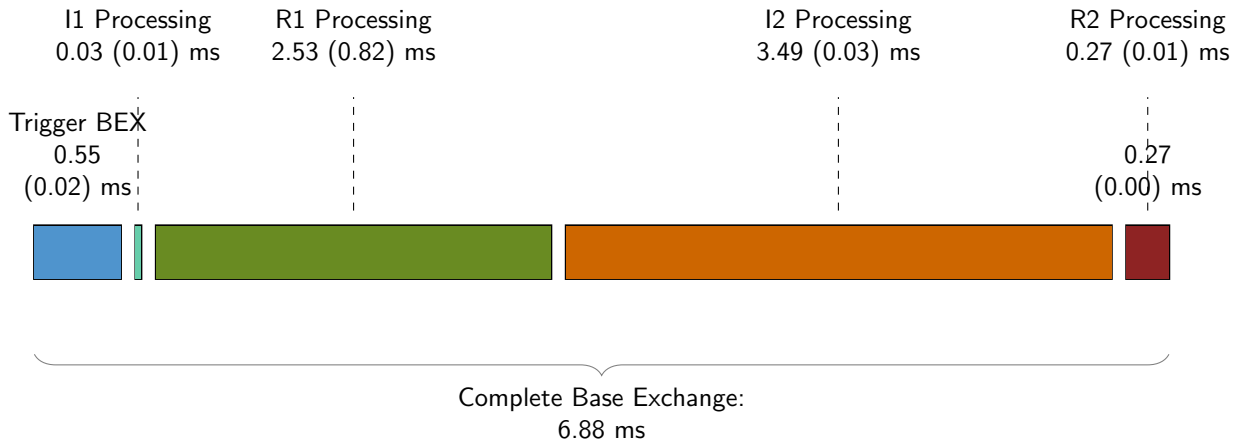

Figure 7.2 HIP BEX with no middelboxes

passive packet forwarders. So, essentially the BEX is between two end-hosts and no middleboxes in between. Figure 7.2 represents the breakdown of the time taken to process the `I1`, `R1`, `I2`, `R2` packets and the time taken to complete the BEX. For example, processing of `I2` (3.50 ms) consists of HMAC and host signature verification on `R1`, creation of `R2` packet and computing HMAC and host signature on `R2`. The complete BEX is very fast with 6.88 ms. Packet level overview does not help us distinguish between the various extensions and their overheads. Since, our implementation is based on SEAMS, we wanted to distinguish between the overheads of the mechanisms we have introduced and the overhead of SEAMS.

7.3 SEAMS BEX with no middleboxes

In Figure 7.3 we see a basic BEX with SEAMS and no middleboxes. In Figure 7.3, we split the BEX into time taken by the packet processing as in standard HIP implementation (6.14 ms), the residual overhead introduced (20.43 ms), network latency (2 ms) and general packet processing (1.76 ms). The residual overhead introduced in the system by an optimization found in SEAMS. Since this work is integrated with SEAMS, we discuss the BEX scenarios with respect to the time taken by standard end-to-end BEX, overhead introduced by SEAMS and the overhead introduced by our extension. The effects of middleboxes on the BEX are similarly split into the overhead introduced by SEAMS, the overhead introduced by our extension and the time taken by them during a standard BEX exchange.

7.3.1 Residual Overhead

The *residual overhead* in introduced by one of the optimizations in SEAMS. The Initiator performs a context lookup soon after it triggers the BEX. The context lookup performs a `netstat`-like query to fetch the application information like the ports opened and the *UID* of the user who ran the application. With the application information available, it it also fetches the user certificates installed on the device. Since, the R1 packet is pre-created (so no creation of response required) and the setup for us is very fast with low network load, the Initiator receives the response

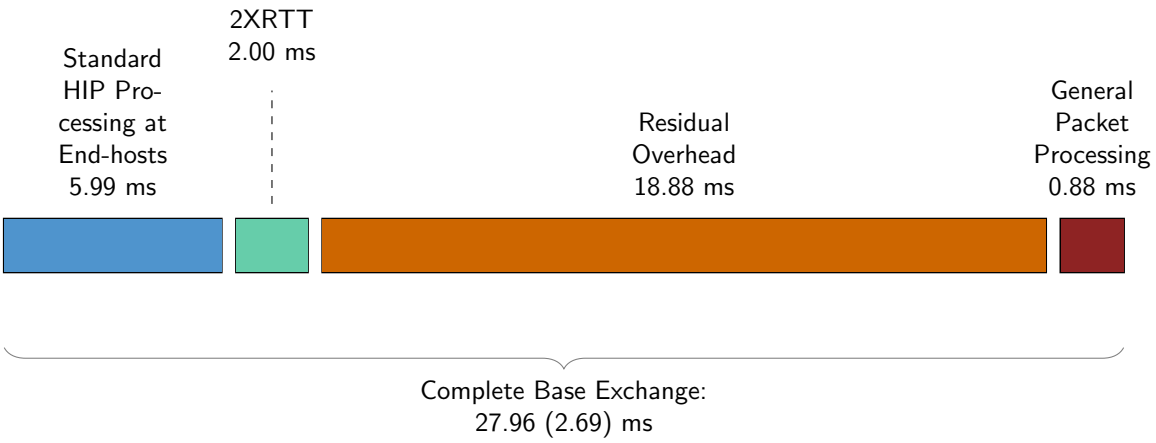

Figure 7.3 Breakdown of the Complete Base Exchange in terms of overhead

Process	Time (ms)	Percentage (%)
Standard HIP Processing at End-hosts	**6.88 (0.00)**	**100.00**
Trigger BEX	0.55 (0.01)	8.02
Processing of I1 at Responder	0.03 (0.00)	0.51
Verification of Host Signature in R1	0.10 (0.04)	1.45
Create ECDH Shared Key (Initiator)	1.20 (0.45)	17.49
Computing and Adding HMAC to I2	0.01 (0.01)	0.20
Computing and Adding Host Signature to I2	1.03 (0.39)	14.93
Verification of HMAC in I2	0.02 (0.00)	0.25
Verification of Host Signature in I2	0.12 (0.00)	1.77
Create ECDH Shared Key (Responder)	1.49 (0.02)	21.69
Computing and Adding HMAC to R2	0.02 (0.00)	0.25
Computing and Adding Host Signature to R2	1.32 (0.03)	19.12
Verification of HMAC in R2	0.02 (0.00)	0.22
Verification of Host Signature in R2	0.09 (0.00)	1.27
Processing	0.88	12.83

Table 7.1 Overhead of the standard HIP-implementation

(`R1`) from the Responder even before it is finished looking up for the context. This creates an over-head which is introduced by our own implementation. Furthermore, it can be computed by subtracting the time taken by the Responder to process the `I1` and send the `R1` (0.03 ms) and the delay introduced by the network (2 ms) from the time taken by the Initiator to perform the context lookup (19.90 ms).

The Table 7.1 gives a break down of a standard HIP BEX Exchange. The breakdown shows the significant measurement points only, the computation time for other measurement points is not significant enough to have any observable change.

Triggering BEX (Initiator): Triggering BEX is fast (0.55 ms) because the HIP Packet size is minimal with only the HITs of the Initiator and the Responder.

Processing of I1 and Sending the pre-created R1 (Responder): The pre-creation of R1 makes it fast for the Responder to process the received `I1` and send the `R1` packet (0.03 ms).

Processing of R1 and Creation of I2 (Initiator): Signature Verification with RSA can be fast when a sufficiently small public exponent is chosen (see Sec-

tion 2.2.2.1). Hence, the verification of host signature in the `R1` packet is fast with 0.1 ms. We observe that the biggest gain of using ECDH in place of regular DH is that the time for shared key generation has decreased from an earlier 11.83 ms (with regular DH) to 1.20 ms (with Elliptic Curve Diffie-Hellman). Computation and the addition of HMAC and Host Signature to the `I2` packet takes 0.01 ms and 1.03 ms respectively.

Processing of I2 and Creation of R2 (Responder): On receiving the `I2` packet, the Responder verifies the HMAC and the Host Signature which takes 0.02 ms and 0.12 ms respectively. The computation of the ECDH shared key takes another 1.49 ms. The `R2` packet is created and HMAC and Host Signature added to it. This takes 0.02 ms and 1.32 ms respectively.

Processing of R2 (Initiator): Verification of HMAC and Host Signature of `R2` take 0.02 ms and 0.09 ms respectively.

The inaccuracy in our measurement is around 0.88 ms. Summarizing the overhead of standard HIP-implementation and the residual overhead with respect to the complete BEX we get the following (similar to the Figure 7.3),

Process	Time (ms)	Percentage (%)
Complete Base Exchange	**27.76 (2.69)**	**100.00**
Overhead of Standard HIP implementation	5.99 (0.00)	21.59
RTT	2.00 (0.00)	7.20
Residual Overhead (SEAMS)	18.88 (0.00)	68.13
Processing	0.88	3.07

Table 7.2 Overheads in Complete Base Exchange

In the Chapter 6, we have talked about the three components of our architecture: *negotiation*, *authentication and authorization* and *selective disclosure*. In the next sections we provide an analysis of the performance of these mechanisms.

7.4 Performance of HIP Base Exchange with Negotiation

In this section, we discuss the time measurements of the Base Exchange with the basic service negotiation (see Section 5.2.4). For this scenario, the network setup was the same as before with the logical network path through the two consumer grade firewall (see Figure 7.1). However, in this case, one of the firewalls was used as a middlebox rather than both of them being used as a passive packet forwarder. We had pre-installed the corresponding peer certificates (user certificates) in each of the end-hosts. The firewall policies were configured to accept the connection request only after authenticating the user identity and processing the application information from the end-hosts, i.e., the firewall would negotiate for the user and application context.

The measurement was taken by making the Initiator establish a new connection with the peer. It starts from the time when the `hipd` at the Initiator initiates the BEX,

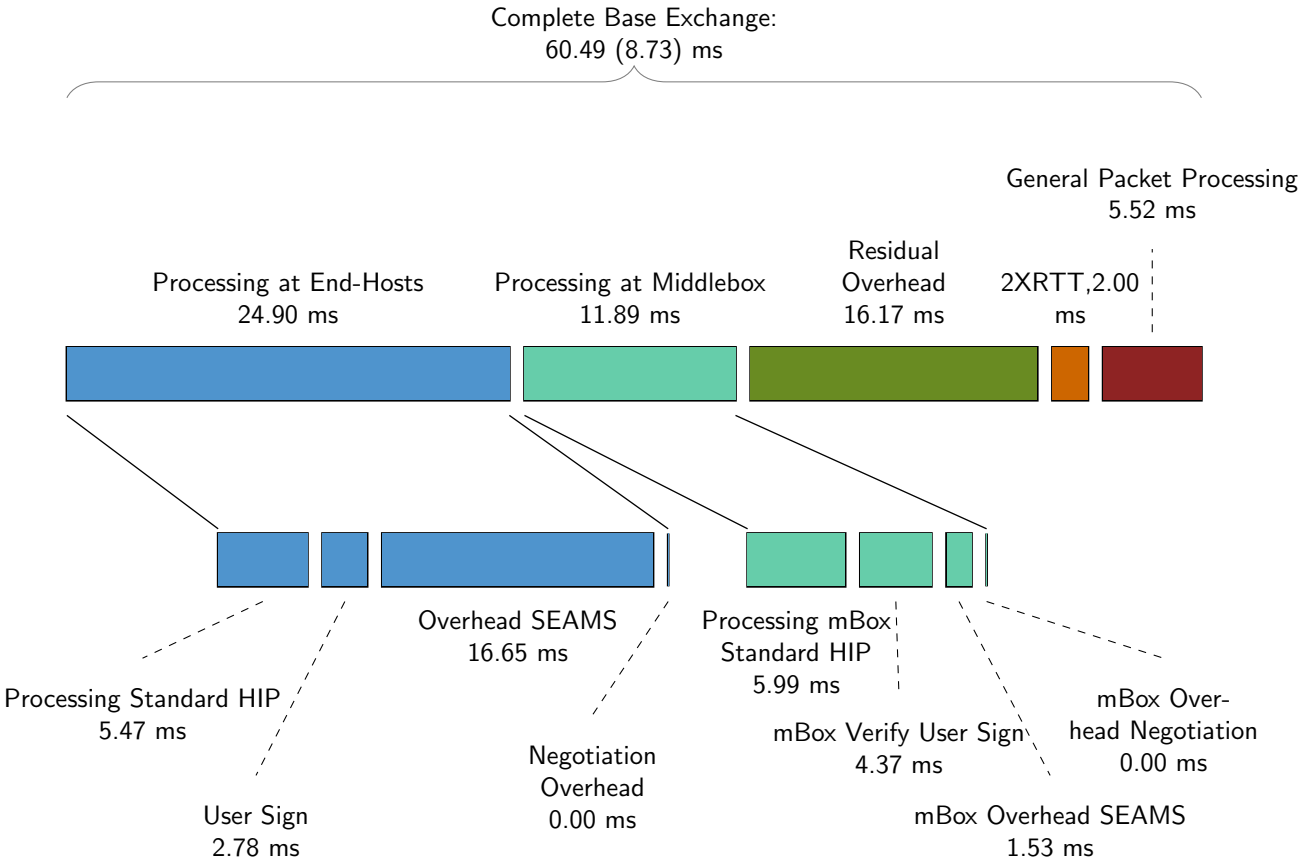

Figure 7.4 HIP Base Exchange with Service Negotiation for user identity and application information from the End-Hosts. {mBox} = Middlebox

i.e., triggers BEX to the time when the security layer at both hosts permits the traffic flow for the connection. Since, the middlebox requested for the user identity, the end-point provided the user identity and ensured the verifiability of the identity by producing a signature on the packet with it secret key. Figure 7.4 provides a high-level overview of the time taken by processing at the end-hosts (24.90 ms), the middleboxes (11.89 ms), the residual overhead (16.17 ms) and general packet processing (5.52 ms). The complete BEX takes 60.49 ms.

BEX with negotiation	**Time (ms)**	**Percentage (%)**
Timing analysis of the BEX with Negotiation	**60.49 (8.73)**	**100.00**
Processing of Standard HIP functions at the End-Hosts	5.47 (0.00)	9.05
Computation and Addition of User Signature	2.78 (0.00)	4.59
Overhead at the End-Host due to SEAMS	16.65 (0.00)	27.53
Overhead of our Extension at the End-Hosts	0.00 (0.00)	0.00
Processing of Standard HIP functions at the Middlebox	5.99 (0.00)	9.90
Verification of User Signature	4.37 (0.00)	7.23
Overhead at the Middlebox due to SEAMS	1.53 (0.00)	2.53
Overhead of our Extension at the Middlebox	0.00 (0.00)	0.00
Residual Overhead	16.17 (0.00)	26.74
Network Latency (2 X RTT)	2.00 (0.00)	3.31
Processing	5.52	9.12

Table 7.3 Timing analysis of the HIP BEX with Negotiation

Process	Time (ms)	Percentage (%)
Standard HIP Processing	**5.49 (0.00)**	**100.00**
Trigger BEX	0.53 (0.10)	9.57
Processing of I1 at Responder	0.03 (0.00)	0.62
Verification of Host Signature in R1	0.09 (0.04)	1.66
Create ECDH Shared Key (Initiator)	1.16 (0.47)	21.06
Computing and Adding HMAC to I2	0.02 (0.01)	0.29
Computation and Adding Host Signature to I2	0.98 (0.41)	17.79
Verification of HMAC in I2	0.02 (0.00)	0.33
Verification of Host Signature in I2	0.11 (0.03)	2.05
Create ECDH Shared Key (Responder)	1.34 (0.35)	24.36
Computing and Adding HMAC to R2	0.02 (0.01)	0.39
Computing and Adding Host Signature to R2	1.10 (0.36)	20.01
Verification of HMAC in R2	0.02 (0.00)	0.35
Verification of Host Signature in R2	0.08 (0.02)	1.53

Table 7.4 Time taken by the standard HIP processing at the End-Hosts for BEX with negotiation

Process	Time (ms)	Percentage (%)
SEAMS	**16.65 (0.00)**	**100.00**
Handling of Service Offer received in R1	0.01 (0.00)	0.06
Acknowledging the Service Offer received in R1	0.00 (0.00)	0.02
Handling of Service Offer received in I2	16.63 (3.29)	99.88
Acknowledging the Service Offer received in I2	0.00 (0.00)	0.03

Table 7.5 Overhead due to SEAMS

7.4.1 Processing at the End-Hosts and Overheads

The processing at the End-hosts includes the standard HIP processing (), user signature for verification of user identity (5.47 ms), the overhead introduced by SEAMS (2.78 ms) and the overhead introduced by our extension(16.65 ms). The basic service negotiation here is same as for SEAMS, hence the BEX does not need handlers provided by our extension. So, the overhead for our extension here is null. The distribution of processing at the End-Hosts as mentioned here can also be seen in the Figure 7.4.

7.4.1.1 Processing Overhead from SEAMS

SEAMS Service negotiation allows the middleboxes to request (*service offer*) for the end-host information necessary for service provisioning. The end-host processes the service offer and adds the information required into the outgoing HIP Packet. Table 7.5 shows the atomic components in which the handling of service offer can be distributed. Upon receiving the service offer in `R1`, the Initiator verifies the information request with its policy and adds all the information parameter which the policy engine deems safe to the outgoing `I2` packet. We have considered this process as one atomic function. Similarly, on receiving the service offers from the middlebox in `I2` packet, the Responder performs policy check and accordingly adds the information parameters to the outgoing `R2` packet. In Table 7.1, the time taken for the Responder to handle the service offer is 16.63 ms while the same operation in the Initiator takes a negligible amount of time, 0.01 ms. This disparity in the

performance measurements at the Initiator and the Responder is due to the on-demand lookup for context information at the Responder. In Section 6.4.1.2, we have discussed that the host context can be considered static during the lifetime of the `hipd`, hence can be looked up at the time of bootstrapping of `hipd`. But the application information and the user context are dependent on the specific connection. Hence, whenever a user or application context is requested the `hipd` must look for the context information. Due to the optimization in our architecture, the Initiator would have already lookup up for the context information by the time `R1` is received (in a real world scenario with more network load or load on the responder). Owing to this, the Initiator is able to respond with the information parameters very fast (0.01 ms). On the contrary, the responder does a context lookup which involves calling a *netstat* like functionality (integrated with our implementation) and looking for the user certificates (disk read cycle). These are time taking operations, hence the Responder takes much more time to handle the service offer and respond with the information parameters. These atomic operations are already a part of SEAMS, hence are considered as overload due to SEAMS in our discussion. Time taken for looking up of application information (the UID of the user running the process, port information) and verifying the application hash with the one received with the attribute certificate takes 16.11 ms. With the help of the UID fetched earlier, the `hipd` loads the user certificate from disk and verifies the certificate, which takes 0.50 ms. Hence, a major performance hit to the handling of service offer at the Responder is from context lookup.

Apart from the handling of service offer, both the Responder and the Initiator respond with an acknowledgement (see Section 5.5) to the service offer which includes a hash of the service offer for the middlebox to verify that the initiator received the service offer untampered. Hashing is another cryptographic operation but it has been highly optimized already, hence we observe a very little contribution of hashing to SEAMS overload.

The reason that we consider the *residual overhead* separately is because, in the other schemes of authentication and selective disclosure the `R1` is usually received after the context lookup has completed. This optimizes the idle time of the Initiator while it waits for the response from the peer. This may also be observed in network with high latency or loads. In such scenarios, the residual overhead might be nil.

7.4.1.2 User Signature

Verifiability of end-point information is an important goal for our architecture. For the middleboxes to verify the user identity (public key), the end-host must sign the identity with the secret key corresponding to that identity. A successful verification of signature ensures that the end-host is indeed in possession of the secret key corresponding to revealed identity.

In our case of service negotiation, the middelbox requests for the user identity, hence it is compulsory for the end-hosts to sign the HIP Packet (covering the user identity). There are two user signature operations in the BEX, once when the Initiator signs the user identifier in `I2` packet and the other when the Responder signs the user identifier in the `R2` packet.

Process	Time (ms)	Percentage (%)
User Signature	**2.78 (0.00)**	**100.00**
User Signature on I2	1.47 (0.65)	52.95
User Signature on R2	1.31 (0.58)	47.05
Processing	0.00	0.00

Table 7.6 User Signature on HIP Packet to establish verifiability of user identity.

Process	Time (ms)	Percentage (%)
Standard HIP Processing at Middlebox	**6.00 (0.00)**	**100.00**
Host Signature Verification in R1	2.09 (0.04)	34.77
Host Signature Verification in I2	2.10 (0.08)	35.00
Host Signature Verification in R2	1.81 (0.06)	30.22

Table 7.7 Processing of Standard HIP functionality at the middlebox

7.4.2 Processing at the Middlebox and Overheads

Similar to the performance of processing at the end-hosts, we split the time taken for processing at the middlebox into 4 groups: standard HIP processing by the middleboxes, the verification of user signatures (ensure verifiability), overhead from SEAMS and overhead of our extension. However, for this case of BEX negotiation, the overhead of our extension is nil. Table 7.7, depicts the atomic components of the standard HIP processing at the middlebox. These are signature verification of the control channel packets. The middlebox spends 5.99 ms for the standard HIP processing, which includes the time measurements for signature verification of `R1`, `I2` and `R2`.

7.4.2.1 SEAMS Overhead

The negotiation of end-point information requires the middlebox to send a service offer to the end-points. Upon receiving the end-point information, the middlebox verifies the information received and if successful HIP BEX is allowed to proceed or blocked depending on the satisfaction of policy.

Overhead Processing R1
On receiving the `R1` packet, the middlebox checks the connection with its policy (0.24 ms). If there are additional information required in order to satisfy the policy, the middlebox requests for them. The policy file is already loaded into the memory at the time of bootstrapping of firewall at the end-point. Hence, the policy checking is fast. After checking with the policy, the end-point adds the information to be requested in the `R1` packet. The middlebox also stores a hash of the service offer which it will later use to verify the acknowledgement in `I2` from the Initiator. Addition of information request and computing the hash of the service offer takes 0.23 ms together.

Overhead Processing I2
On receiving the `I2` packet, the middlebox looks for the acknowledgement in the packet and verifies it. Since the hash was already calculated earlier when processing `R1`, the verification is a simple compare of the two hashes which takes only 0.02 ms. After the verifiability of the end-point information is ensured, the middlebox parses the information received against its policy file. Verifiability of user identity is ensured

Process	Time (ms)	Percentage (%)
SEAMS Overhead	**1.75 (0.00)**	**100.00**
Check Policy for Initiator (R1)	0.24 (0.03)	13.82
Adding Service Offer to R1	0.23 (0.03)	13.34
Verification of Service Acknowledgment (I2)	0.07 (0.02)	4.00
Verification of Information received with I2	0.38 (0.08)	21.71
Check Policy for Responder (I2)	0.17 (0.06)	9.64
Adding Service Offer to I2	0.22 (0.05)	12.76
Verification of Service Acknowledgment (R2)	0.06 (0.01)	3.69
Verification of Information received with R2	0.37 (0.04)	21.68

Table 7.8 Processing Overhead at the Middleboxes due to SEAMS

Process	Time (ms)	Percentage (%)
User Signature	**4.38 (0.00)**	**100.00**
Verification of User Signature in I2	2.21 (0.08)	50.45
Verification of User Signature in R2	2.17 (0.04)	49.55
Processing	0.00	0.00

Table 7.9 Overhead at the middleboxes due to User Signature Verification

with the verification of the user signature. However, we consider this as separate from SEAMS. Since, the policy file is already loaded in memory, the policy verification here takes a mere 0.17 ms. After the successful verification, the middlebox checks for the policy for Responder and adds the information it requires in another service offer to the `I2` packet. The three-way message exchange is symmetric and follows the same sequence of message flows and processing of messages as mentioned here. The details for the rest of the processing overhead for the SEAMS can be found in the Table 7.8.

The total processing overhead at the middlebox owing to SEAMS is 1.75 ms.

7.4.2.2 Verification of User Signature

After the verification of service acknowledgement in the `I2` packet, the middlebox verifies the user signature. This is important in order to ensure the verifiability of end-point information received in the `I2` packet. Verification of user signature in the `I2` packet takes 2.21 ms. Similarly, verification of user signature in the `R2` packet takes 2.17 ms (see Table 7.9).

Since, the basic service negotiation as in this case does not require any handlers from our extension, there is no processing overheads for our extension.

7.5 Performance of HIP BEX with Authentication of Middleboxes

In the Figure 5.1, we had shown that the end-host might need to authenticate on-path devices depending on the trust level before revealing information to them. We discuss the different mechanisms for authenticating the middleboxes and the end-hosts to reveal their sensitive information securely in Section 5.4.3. We provide the

performance analysis based on the Diffie-Hellman approach to protect the end-point information and the RSA-based approach. The processing time for standard HIP operations at the end-points and the middleboxes remains the same with 5.95 ms and 6.00 ms. Similarly, the SEAMS overhead remains very similar with 17.99 ms for the end-points and 6.00 ms for the middlebox. Hence, we do not be discussing the computation times of the components of SEAMS in this section. Figure 7.5 providers a breakdown of the timing analysis of the HIP BEX with Authentication.

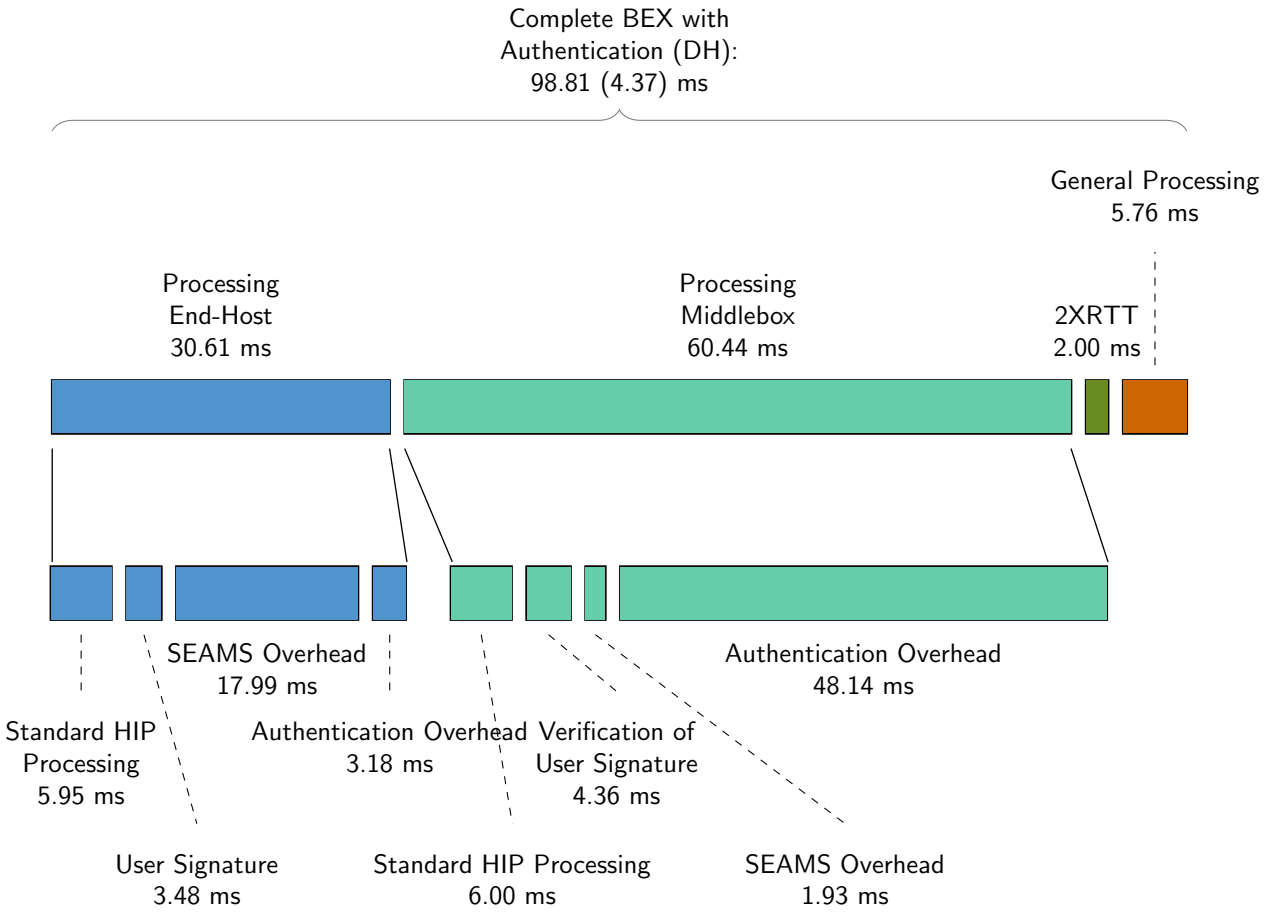

Figure 7.5 Timing Analysis of BEX with Authentication (Encrypting of sensitive information using shared key from DH) {mBox} = Middlebox

7.5.1 Performance Analysis for Diffie-Hellman based approach

In this case, the service offer from the middlebox contains a hint to the middlebox certificate (see Section 5.4). The end-host might possess this hint and the corresponding middlebox certificate from an out-of-band communication or through a mutually trusted third party (e.g. Directory Service). Upon receiving such a service offer, the end-host attempts to locate the middlebox certificate and if found, verifies the certificate of the middlebox. The certificate verification is sufficient to prove the identity of the the middlebox.

7.5.1.1 Processing overhead of our extension at the end-hosts

Table 7.11, provides the processing overhead at the end-host from the point where the end-host looks up for the certificate and authenticates the middlebox to the point when the BEX is complete.

On receiving the service offers in `R1` packet, the Initiator parses the service offers, groups the information requested in the service offer as described in Section 5.5.2. Also, while parsing the service offer it locates the corresponding middlebox certificate with the help of the hint provided in the service offer. It also verifies the certificates and when it is successfully verified, it fetches the middlebox DH-public key. This complete process is very fast and only takes 0.21 ms.

Thereafter, the Initiator generates a new random symmetric key which takes 0.01 ms. This is used to encrypt the sensitive end-point information. With the help of the symmetric key generated, the end-point information is encrypted and added into the `I2` packet which takes 0.02 ms (symmetric key encryption is fast). Furthermore, a Diffie-Hellman shared key is generated from the Initiator's DH-private key and the middlebox's DH-public key (installed on the device). Both of these are ECDH keys based on NIST-P-256. The shared key is then used to encrypt the symmetric key (used to encrypt end-point information). Generating the DH-shared key, computing the hash of the service offer, encrypting the symmetric key using the shared key and adding these two wrapped in a service acknowledgement to the `I2` packet takes 1.47 ms. Since, the three-way handshake is symmetric the same steps are reiterated for the Responder. Due to fast symmetric key operations, the total overhead of DH-based authentication is a minimal 3.18 ms.

7.5.1.2 Processing overhead of our extension at the middleboxes

The `R1` contains the DH-public key of the Responder. So, on receiving the `R1` packet, the middlebox generates the DH-shared key (shared with the Responder) using its own DH-private key and the Responder's DH-public key. Generation of DH-shared key requires modular arithmetic over large primes, hence this process is very expensive and takes a lot of time for the weak processor in the middlebox. By using Elliptic-Curve Diffie Hellman the computation was reduced to an acceptable

Process	Time (ms)	Percentage (%)
Timing analysis of BEX with Authentication	**98.81 (4.37)**	**100.00**
Standard HIP Processing at End-Hosts	5.95 (0.00)	6.03
User Signature at End-Hosts	3.48 (0.00)	3.52
SEAMS Overhead at End-Hosts	17.99 (0.00)	18.21
Overhead of our Extension at the End-Points	3.18 (0.00)	3.22
Standard HIP Processing at the Middleboxes	6.00 (0.00)	6.08
Verification of User Signature at the Middleboxes	4.36 (0.00)	4.42
SEAMS Overhead at the middleboxes	1.93 (0.00)	1.96
Overhead of our Extension at the Middleboxes	48.14 (0.00)	48.72
Network Delay (2XRTT)	2.00 (0.00)	2.02
Processing	5.76	5.83

Table 7.10 Processing Overheads at End-Hosts and Middleboxes when authentication is used.

Process	Time (ms)	Percentage (%)
Timing analysis of our Extension at End-Host	**3.18 (0.00)**	**100.00**
Grouping of Information & Middlebox certificate lookup (I2)	0.21 (0.01)	6.52
Generate Symmeteric Key for Encrypting Info Parameters (I2)	0.01 (0.00)	0.31
Encrypt end-point information (I2)	0.02 (0.00)	0.51
Encrypt Symmetric Key and add to the Ack (I2)	1.47 (0.02)	46.11
Grouping of Information & Middlebox certificate lookup (R2)	0.00 (0.00)	0.00
Generate Symmeteric Key for Encrypting Info Parameters (R2)	0.01 (0.00)	0.33
Encrypt end-point information (R2)	0.02 (0.00)	0.50
Encrypt Symmetric Key and add to the Ack (R2)	1.45 (0.14)	45.71

Table 7.11 Overhead of our Extension for BEX with authentication at the End-point

Process	Time (ms)	%
Timing Analysis of our Extension at the Middlebox	**48.14 (0.00)**	**100.00**
Generation of Shared DH Key for the Responder (R1)	23.90 (0.18)	49.65
Generation of Shared DH Key for the Initiator (I2)	23.90 (0.22)	49.65
Decrypt Symmetric Key in the Service ACK (I2)	0.10 (0.02)	0.22
Use the Symmetric Key to decrypt the End-Point Info (I2)	0.16 (0.03)	0.33
Decrypt Symmetric Key in the Service ACK (R2)	0.07 (0.01)	0.15
Use the Symmetric Key to decrypt the End-Point Info (R2)	0.00 (0.00)	0.00

Table 7.12 Overhead of our extension for BEX with authentication at the Middleboxes

23.9 ms. Since, the computation time of the DH-shared key is quite large, there is no residual overhead as seen in the Figure 7.5. The shared key generated here is saved for the lifetime of the BEX and deleted thereafter.

Similar to the above, on receiving the `I2` packet, the middlebox generates the DH-shared key for the Initiator. This takes 23.9 ms again. Using the shared key, the middlebox decrypts the symmetric key wrapped in the service acknowledgement which takes 0.02 ms. The symmetric key is further used to decrypt the end-point information which arrive wrapped in `HIP_ENCRYPTED` parameter, which takes another 0.16 ms. The time for decryption is more than before because the size of the data encrypted is larger and the difference is more pronounced because the middlebox has a weak processor. The decrypted end-point information is used for verification with policy engine, but it is covered in the SEAMS overhead.

The shared key established with the Responder is used to decrypt the symmetric key received in the acknowledgement with the `R2` packet. The rest of the operations are similar to before and can also be observed in the Table 7.12. The complete Base Exchange takes 98.81 ms.

7.6 Performance Analysis for RSA based approach

In the Figure 7.6, we show the contribution of individual components (end-hosts, middleboxes etc.) in a HIP BEX with RSA-based approach for authentication. As in the previous action, the middlebox sends a service offer to the end-host with a service certificate hint. However, in this case the symmetric key used to encrypt the end-point information is sent encrypted with the public key of the middlebox. The complete BEX takes 139.05 ms. This is largely due to the processing overhead of

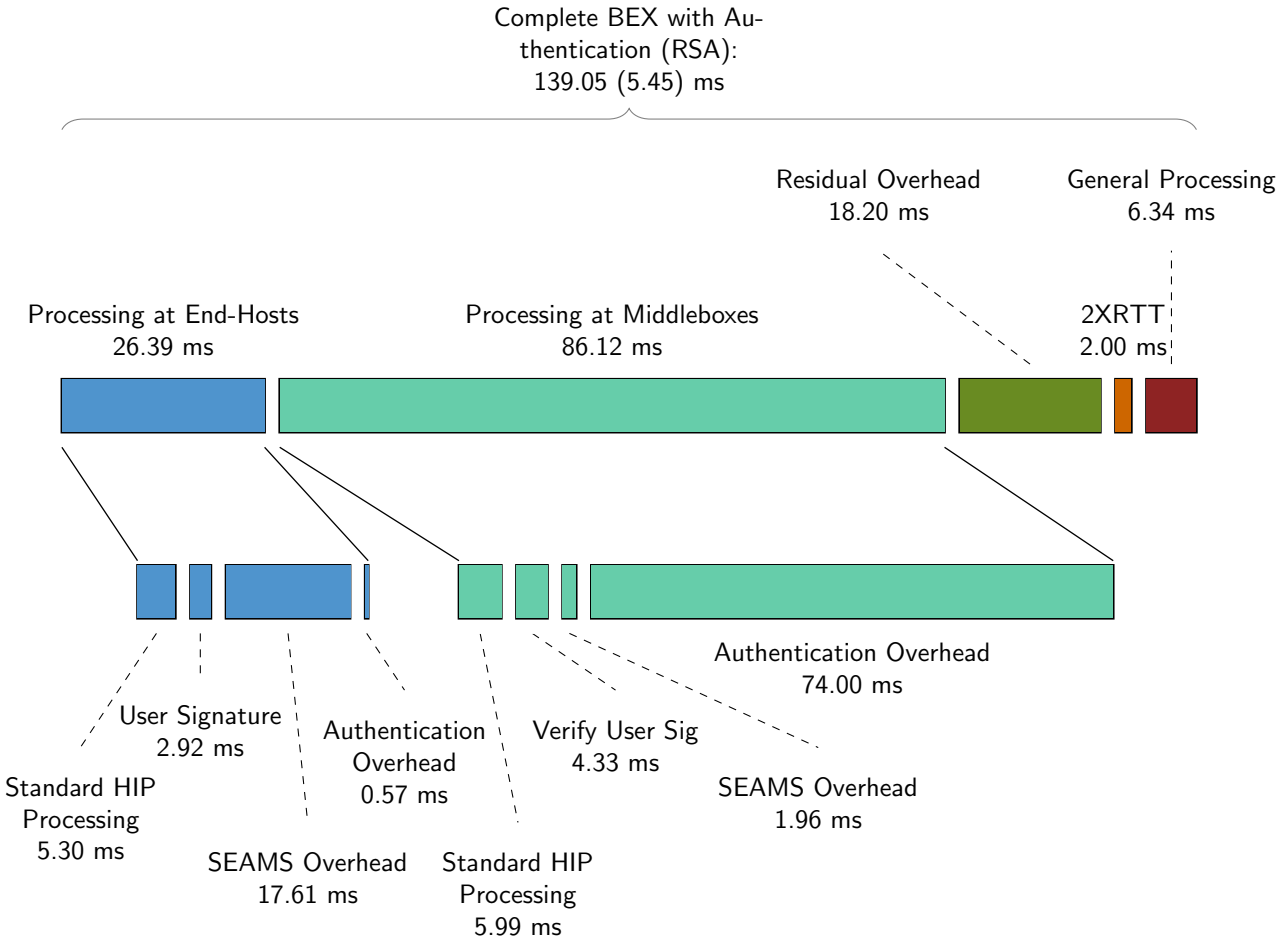

Figure 7.6 Timing Analysis of BEX with Authentication (Encrypting of sensitive information using shared key from RSA)

decryption using the private key of the middlebox. As before, the computation time of Standard HIP Processing, the overhead of SEAMS and User Signature, remain the same in this case also. From the Figure 7.5, we can see that the SEAMS overhead at the end-host and the middlebox is 17.61 ms and 1.96 ms respectively. Similarly, the Standard HIP Processing remains approximately the same for the end-hosts and the middleboxes with 5.30 ms and 5.99 ms respectively. The measurement has been made with RSA 1024-bit keys for the middlebox.

Process	**Time (ms)**	**%**
Timing Analysis of the overhead at the End-Hosts	**0.57 (0.00)**	**100.00**
Grouping of Information & Middlebox certificate lookup (I2)	0.15 (0.06)	26.98
Generate Symmeteric Key for Encrypting Info Parameters (I2)	0.01 (0.00)	1.40
Encrypt end-point information (I2)	0.01 (0.00)	2.21
Encrypt Symmetric Key and add to the Ack (I2)	0.10 (0.04)	17.03
Grouping of Information & Middlebox certificate lookup (R2)	0.15 (0.06)	26.98
Generate Symmeteric Key for Encrypting Info Parameters (R2)	0.01 (0.00)	1.80
Encrypt end-point information (R2)	0.02 (0.00)	2.70
Encrypt Symmetric Key and add to the Ack (R2)	0.12 (0.02)	20.90

Table 7.13 Overhead at the End-Hosts for a BEX with authentication using RSA-based approach

7.6.1 Processing overhead of our extension at the end-hosts

The Initiator receives the service offer with the service certificate hint in the R1 packet. The Initiator looks up for the middlebox certificate with the help of the certificate hint in the service. If found, it verifies the certificate. It also splits the information requested into groups before encrypting them together. We have considered this in as one atomic function and it takes 0.15 ms as shown in the Table 7.13. After the middlebox certificate has been verified, a random symmetric key is generated which takes 0.01 ms. This symmetric key is used to encrypt the sensitive end-point information and the encrypted information is added to the I2 packet, which takes 0.01 ms (symmetric key encryption is fast). The symmetric used here is encrypted using the public key (RSA-1024) of the middlebox and added along with the service acknowledgement to the outgoing I2 packet. This takes another 0.10 ms. Since, the three way handshake is symmetric, the Responder will make similar contributions to the overhead. The total overhead of our extension at the end-hosts is a 0.57 ms for this case.

7.6.2 Processing overhead of our extension at the middlebox

On receiving the I2 packet, the middlebox looks for the service acknowledgement corresponding to the service offer. It then decrypts the symmetric key information from the acknowledgement in 40.85, using its private key (RSA 1024-bit). RSA has been designed to have a strong private key (a large number), hence the private key operations (modular arithmetic) are computationally intensive. However, the public exponent can be appropriately chosen to be small enough (hence faster). The decryption of Symmetric Key information is a private key operation, hence computationally expensive. The middlebox with low processing power takes a large amount of time (40.85 ms) as shown in the Table 7.14. The symmetric key is then used by the middlebox to decrypt the encrypted end-point information parameters. It takes another 0.16 ms. The processing at the Responder is similar to the one for Initiator.

Process	Time (ms)	%
Timing Analysis of the overhead at the middlebox	**74.00 (0.00)**	**100.00**
Decrypt Symmetric Key in the Service ACK using its private key (I2)	40.85 (0.43)	55.32
Use the Symmetric Key to decrypt the End-Point Info (I2)	0.16 (0.03)	0.22
Decrypt Symmetric Key in the Service ACK using its private key (R2)	32.83 (3.07)	44.45
Use the Symmetric Key to decrypt the End-Point Info (R2)	0.16 (0.00)	0.00
Processing	0.00	0.00

Table 7.14 Overhead at the Middlebox for a BEX with authentication using RSA-base approach

In the Figure 7.6, we can see there is residual overhead of 18.20 ms in this case. *Residual Overhead* can be observed whenever there the response packet R1 from the peer is received before the Initiator completes the context lookup. Since, in this case the processing overhead of R1 at the middlebox is very low (3.89 ms), there is a residual overhead.

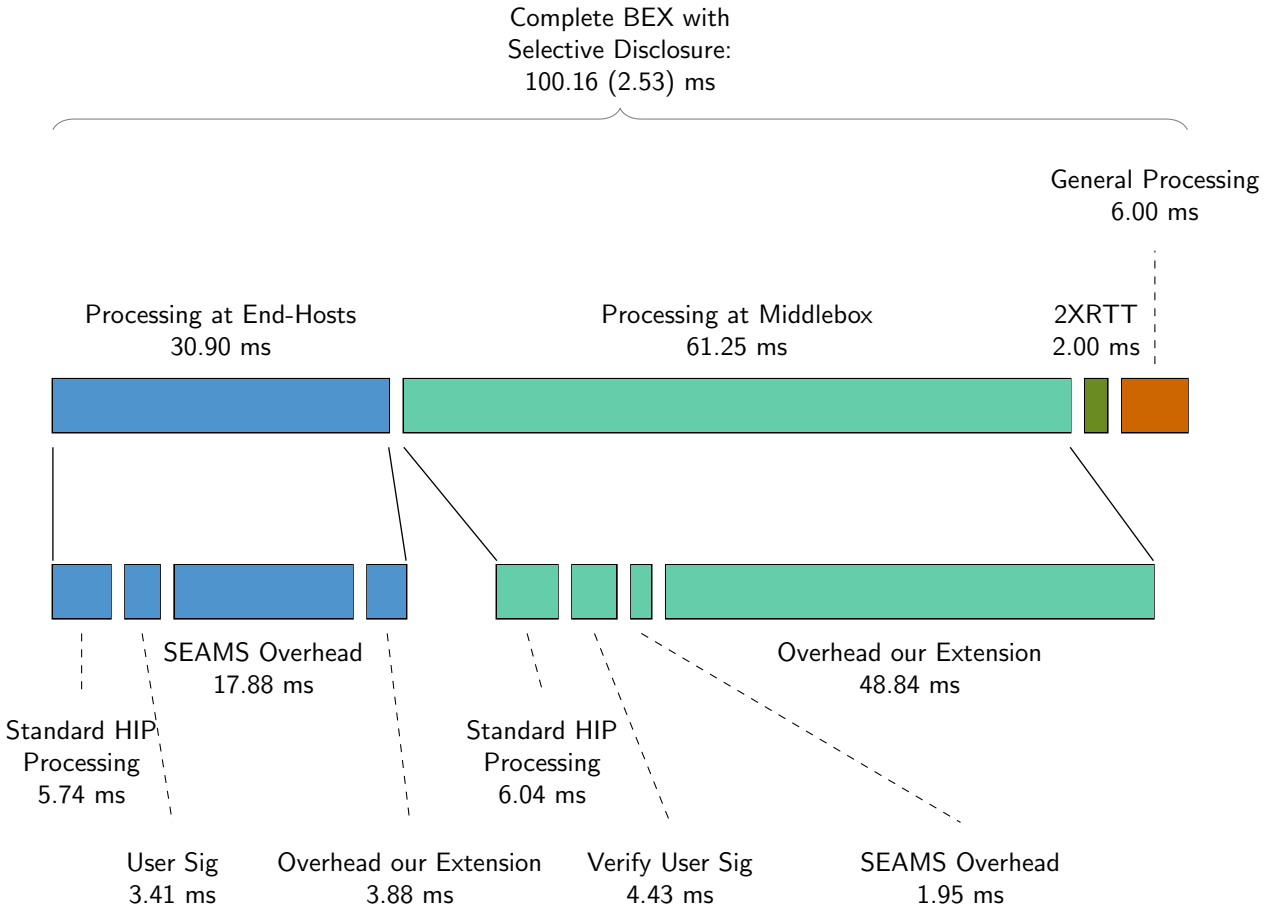

Figure 7.7 Timing Analysis of BEX with Selective Disclosure

7.7 Performance of HIP BEX with Selective Disclosure

The BEX with Selective Disclosure needs the end-host to trust the middlebox with sensitive information such that the middlebox will remove the information as soon as it has processed it. From the Figure 7.7, we can observe that the overhead of SEAMS, user signature and the Standard HIP Processing follows the same pattern as before. Hence, the SEAMS overhead at the end-points and the middleboxes remains nearly the same at 17.88 ms and 1.95 ms respectively. The standard HIP Processing takes a similar amount of time with 5.74 ms at the end-hosts and 6.04 ms at the middleboxes. We have taken the measurements, with the middlebox public-private key pair based on ECDSA (NIST-P-256). The motivation behind this was that, the time for signature generation using RSA private key operations is computationally more expensive when compared to ECDSA. Also, the ECDSA key is stronger with a smaller key size, which can be significant for the space constraint HIP Packet.

7.7.1 Processing Overhead of our extension at the end-hosts

On receiving the service offer from the middlebox in the `R1` packet, the Initiator performs two main operations. Firstly, the information requested in the service

offer is parsed and the information remove list is built. This list is later sent to assist the middleboxes in removing the parameters that can be removed safely. This process takes 0.21 ms as shown in the Table 7.15. Secondly, the Initiator verifies the signature of the middlebox on the service offer, so that it can trust the middlebox with safe removal of information parameters from the signaling channel (I2 packet). Verification of Service Signature takes 1.81 ms. The Responder performs the same operations on the service offers received with the I2 packet. The overhead at the End-Host remains low at 3.88 ms.

Process	Time (ms)	%
Timing analysis of Selective Disclosure at End-Host	**3.88 (0.00)**	**100.00**
Build the information parameter remove list (R1)	0.21 (0.00)	5.34
Verifies the Middlebox Signature R1	1.81 (0.02)	46.64
Build the information parameter remove list (I2)	0.24 (0.00)	6.28
Build the information parameter remove list (I2)	1.62 (0.43)	41.75

Table 7.15 Overhead of our Extension at the End-Host

7.7.2 Overhead of our Extension at the Middleboxes

The middlebox adds the service offer for the Initiator to the R1 packet. It also adds its signature on the service offer allowing the end-hosts to trust it before it reveals any sensitive information that should not be in the signaling channel. The signature generation takes 24.29 ms at the middlebox as shown in the Table 7.16.

On receiving the I2 packet, the middlebox looks for the service acknowledgement corresponding to the service offer, retrieves the safe remove list from the service offer. The information parameters that were deemed safe to remove by the End-Host are removed from the signaling channel. The removal of parameters and addition of the hash of the parameter from the hash graph to the end-point takes 0.14. The same process is also performed at the Responder with the service offer in the I2 packet and removal of information parameters from the R2 packet.

7.7.3 Performance of BEX with no request for User Identity

The user signature in the HIP Packet is important for the end-host to prove the verifiability of user information to the middlebox. If the service offer from a middlebox does not contain the request for user identity, then the end-host does not need to compute the user signature. Hence, whenever user information is not requested

Process	Time (ms)	%
Timing analysis of Selective Disclosure at Middlebox	**48.84 (0.00)**	**100.00**
Signing the Serice Offer in R1	24.29 (0.26)	49.74
Removal of End-Point Info Parameters from I2	0.14 (0.01)	0.29
Signing the Service Offer in I2	24.30 (0.30)	49.75
Removal of End-Point Info Parameters from R2	0.11 (0.02)	0.23
General Processing	0.00	0.00

Table 7.16 Overhead of our Extension at the Middlebox

in the service offer, the time for computation of signature at the end-hosts and the verification of signature at the middleboxes is saved. This can also be observed in the BEX.

In the Table 7.17, we present the timing analysis for another BEX with Authentication. The complete BEX with a service offer for application information but no user information takes 90.83 ms.

Since, the middlebox did not request for the user identity, the end-host does not need to produce the user signature. From the Table 7.10, the computation of user signature by the end-host takes 3.48 ms and the verification of signature by the middlebox takes 4.36 ms. Hence, there should be a reduction of approximately 7.48 ms (3.48 ms + 4.36 ms) in the BEX which is consistent with the results in the Table 7.17.

Process	Time (ms)	%
Timing Analysis of BEX with Authentication	**90.83 (3.15)**	**100.00**
Standard HIP Processing at the End-Host	5.92 (0.00)	6.52
User Signature	0.00 (0.00)	0.00
SEAMS Overhead at the End-Host	17.58 (0.00)	19.36
Overhead of our Extension at the End-Host	3.11 (0.00)	3.42
Standard HIP Processing at the Middlebox	6.38 (0.00)	7.03
Verify User Signature at the Middlebox	0.00 (0.00)	0.00
SEAMS Overhead at the Middlebox	1.78 (0.00)	1.96
Overhead of our Extension at the Middlebox	48.15 (0.00)	53.01
Network Delay (2 X RTT)	2.00 (0.00)	2.20
General Processing	5.91	6.51

Table 7.17 BEX with Authentication. (Application Info requested in the offer)

Moreover, if the service offer does not contain any request for application or user information, the time required for context lookup is saved at the Responder. The Initiator always performs a context lookup after the triggering of BEX. The Responder on the other hand performs the context lookup on-demand (application or user information). In such a case there should be additional saving of the time required for the context lookup at the Responder. From our measurements, the time taken for a context lookup is around 17.30 ms. Hence, there should be an observable reduction in the BEX time of 24.78 ms (17.30 ms + 7.48 ms) which is also consistent with our result (72.71 ms) in Table 7.18.

Although we have presented only one of the few cases that we have discussed in this chapter, we observed similar patterns in other cases as well.

7.7.4 Performance Overhead of the UPDATE Exchange

The Update Exchange is similar to the three way handshaking as we have in the HIP BEX. Hence, we have an additional Diffie-Hellman parameter in the `U1` and the `U2`. These Diffie-Hellman parameters can be used by the middlebox in a BEX or Update Exchange with authentication (DH-based approach). The only additional overhead apart from the ones we discussed for the HIP BEX is Diffie-Hellman parameter. However, the usage of ECDH (NIST-P-256) instead of the regular Diffie-Hellman has helped keep this overhead to only 0.14 ms.

Process	Time (ms)	%
Timing Analysis of BEX with Authentication	**72.71 (0.69)**	**100.00**
Standard HIP Processing at the End-Host	5.96 (0.00)	8.20
User Signature	0.00 (0.00)	0.00
SEAMS Overhead at the End-Host	0.06 (0.00)	0.08
Overhead of our Extension at the End-Host	3.17 (0.00)	4.36
Standard HIP Processing at the Middlebox	5.96 (0.00)	8.20
Verify User Signature at the Middlebox	0.00 (0.00)	0.00
SEAMS Overhead at the Middlebox	1.75 (0.00)	2.42
Overhead of our Extension at the Middlebox	48.03 (0.00)	66.05
Network Delay (2 X RTT)	2.00 (0.00)	2.75
General Processing	5.77	7.94

Table 7.18 BEX with Authentication. (Only host information requested in the offer)

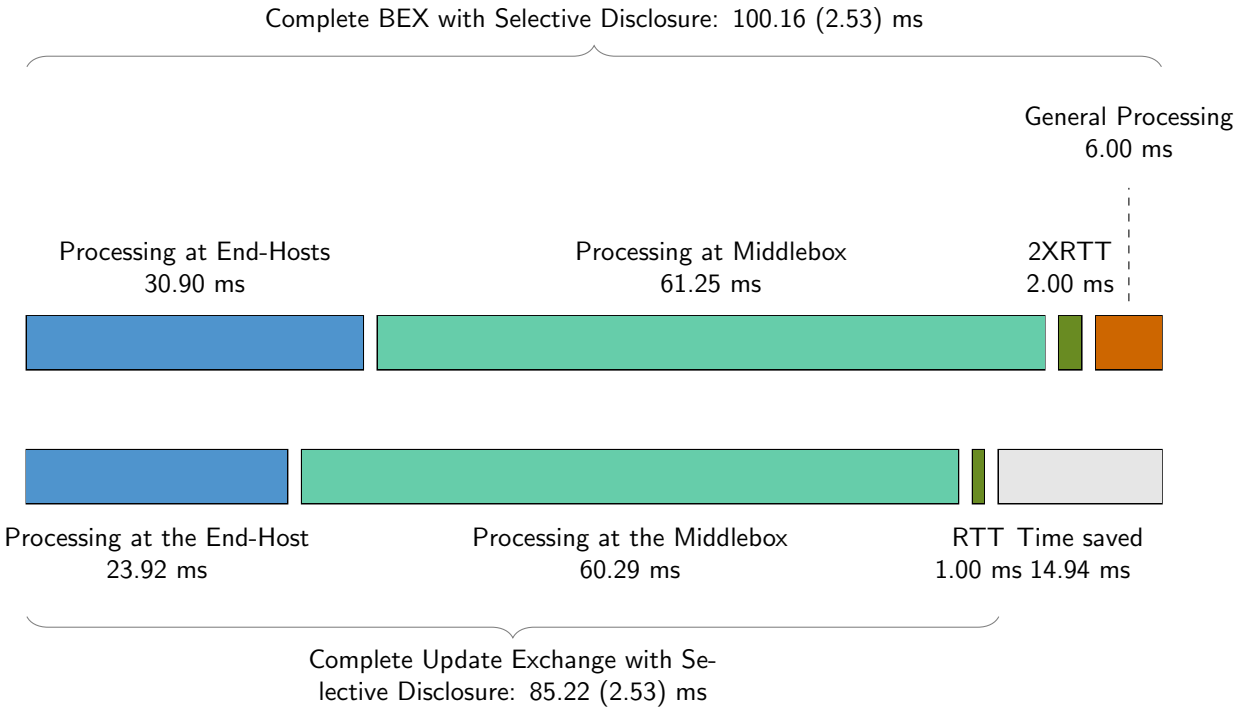

Figure 7.8 Comparison of BEX with Update Exchange for Selective Disclosure

The Update Exchange is supposed to be light-weight when compared to the normal BEX. In Figure 7.8, we see a comparison between the BEX and the Update Exchange for the selective disclosure case. The time saved by the Update Exchange here is 14.94 ms. We also observe that the processing at the end-points is faster by approximately 6 ms. This can be due to the fact there is no host signature verification for Update Exchange. Also, unlike the BEX, the Update Exchange does not need to compute the DH-shared key (1.2 ms approximately on both peers). Processing overhead at the middlebox, however, remains the same.

Figure 7.9 compares the BEX and the Update Exchange for the authentication with RSA based approach. The reduction in processing time at the end-hosts remains the same as before. However, we see that there is a difference between processing overhead at the middleboxes. On a deeper analysis we found out that the time for decryption of symmetric key received in `I2` takes 42 ms, while the time for decryption of symmetric key in `R2`, `U2` and `U2` remain the same at 33 ms. This however is not

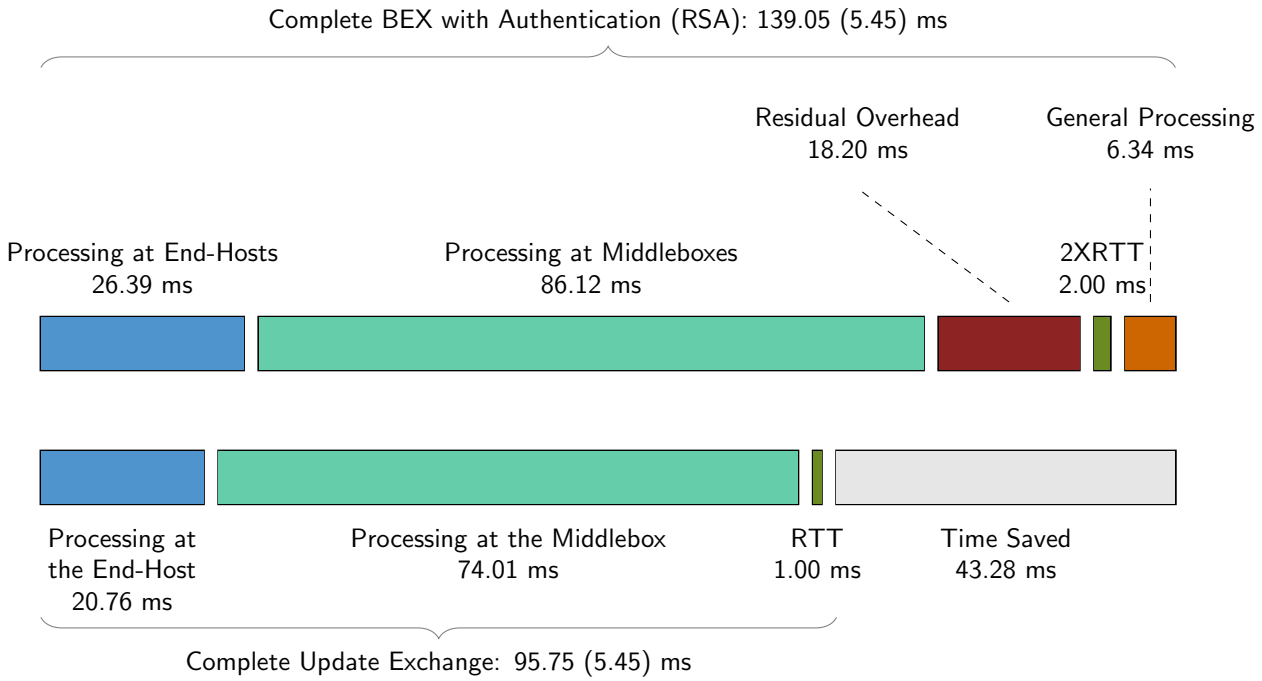

Figure 7.9 Comparison of BEX with Update Exchange for Authentication (RSA)

explainable and could have been due to an increase in the load on the middlebox processor from some unassociated factor. There is a residual overhead which also increases the time for completion of BEX by 17.92 ms. The total saving in time for the Update exchange is 43.28 ms.

Figure 7.10 provides the comparison between the BEX and the Update Exchange for authentication with DH approach. There is a slight increase in the overhead at the end-points because of the ECDH Shared Key generation. The time saved in the Update Exchange is 15 ms.

7.8 Discussion of Design Goals

We have designed our system in accordance to the design goals we have discussed in Chapter 5. In the following sections we quantify, how our implementation has achieved the design goals. Design goals cover the aspects of verifiability of end-point information, establishment of trust between network entities and the implications of privacy like minimal disclosure of information and unlinkability of information etc.

7.8.1 Verifiability of End-Point Information

An important consideration was verifiability of end-point information. The system should allow the trusted entities on the signaling path to verify the information provided by the end-point. Verifiability of information as well as identity is important for our system.

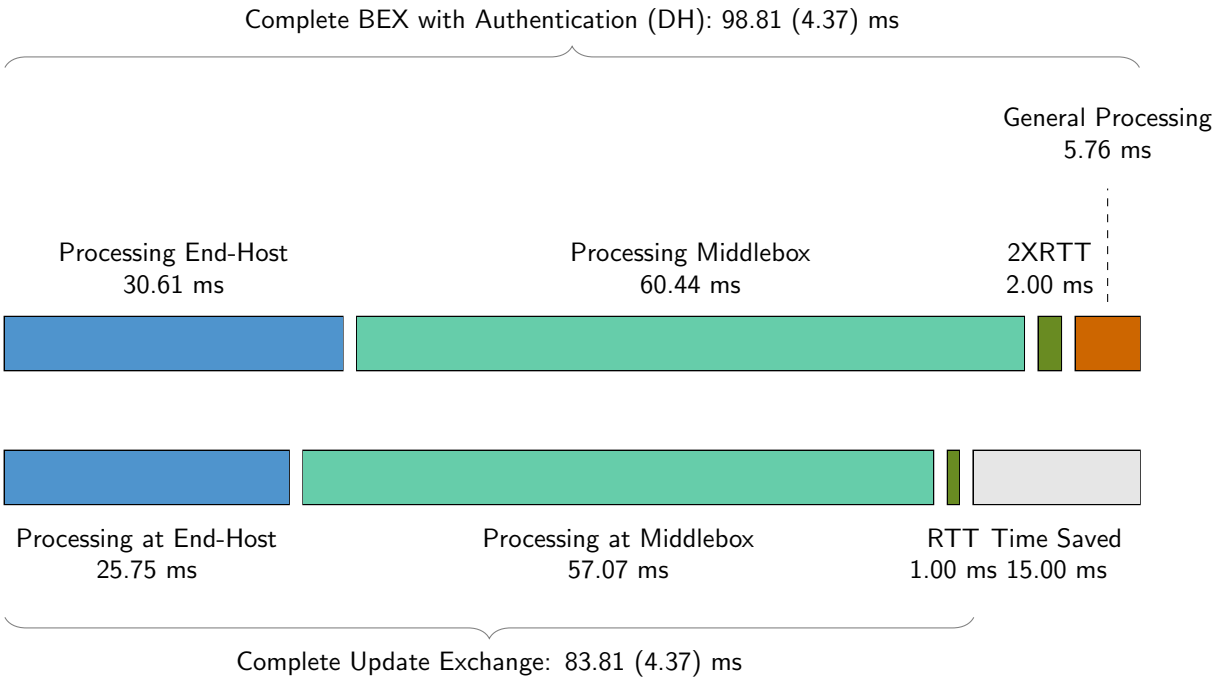

Figure 7.10 Comparison of BEX with Update Exchange for Authentication (DH)

7.8.1.1 Verifiability of Host Identities

Our design and implementation does not alter the way Host Identities are handled in HIP because we did not wish to introduce any new security problems. However, the Host Identities in HIP are based on public-key cryptography. This makes them verifiable to any on-path entity. The host public key signaled in the Host Identity is necessary for the verification of HIP signatures in I2 and R2 packet. The signature verification confirms the integrity of HIP Packet and ensures that the packet has originated from the host which the packet claims.

The HIPL implementation supports RSA and DSA for key sizes upto 2048-bits only, because of the packet size limitations. However, the ECDSA keys of atleast 384-bit curve sizes are also supported by the implementation from Ziegeldorf et. al. Since, our work in based on the work from Ziegeldorf et. al. our implementation also supports ECDSA keys.

7.8.1.2 Verifiability of User Identities

The user identities have been implemented just like the existing implementation of Host Identities. This entails that these identities possess the same security characteristics as the ones held by the host identities. The user identity is bundled within the signed portion of the HIP Packet. Along with the host signature, the end-point also signs the packet with the secret key corresponding to user identity (public key). On receiving the HIP Packet, the on-path entity can extract the public key from the user identity and verify the signature. The verification of signature by the on-path entity ensures that the originator of the HIP Packet is indeed in possession of the secret corresponding to the public key, hence ensures the *verifiability of user identity*.

7.8.1.3 Verifiability of Application Identities

The application identities can not be verified cryptographically in the network, since the applications are not independent entities capable of protecting a secret key. They are always tied to a user executing the application or a host on which the application is being run. Hence, we assign each of the application, attribute certificates which contain the hash of the application binary and a user or host signature on the hash. Any modification to the application by a malicious host can be easily detected by the security layer by computing the hash of the binary and verifying the user or host signature on the hash.

7.8.2 Minimal and Selective Disclosure

We have talked about the role of policy engines operating on both the middleboxes and the end-points (see Section 5.4.2). The policy engine on the middlebox negotiate for the minimum amount of information that they need in order for the end-point to access their service. Conversely, the policy engine on the end-point determines if the information requested is necessary and sufficient for the middlebox services. BEX with negotiation is not sufficient to restrict the visibility of end-point information to only the trusted entities. The information would still be freely available on the signaling channel for others to access and verify.

BEX with authentication, splits the information requested by on-path devices into multiple groups of information items, encrypts these groups separately and only then make the encrypted information available on the signaling channel. Encrypting these groups in a manner in which only the pre-determined set of on-path devices can decrypt them, limits the visibility of end-point information. Hence, the other untrusted or unauthenticated on-path devices can not verify the user identity. It is also good for privacy of the end-point information. So, in order to verify the user identity, a middlebox has to decrypt the end-point information, fetch the user identity from the decrypted information, extract the public key from the user identity and verify the user signature.

BEX with selective disclosure provides a mechanism for the trusted middleboxes to remove certain information parameters from the HIP message. It prevents the information from being available on the signaling path beyond that middlebox. Hence, the information is only available till the point the end-point trusts the signaling channel. So, combining these approaches, the end-point can control the amount of information and to whom the information can be made available, i.e., minimal and selective disclosure.

7.8.3 Unlinkability of End-Point Information

A verifiable signature on the HIP message, links the identity of the sender to all of the information available in the message. BEX for selective disclosure permits the middleboxes to remove pre-determined information items from the HIP message. Hence, those end-point information items which have been removed from the HIP

message can not be linked to the end-point's identity as they are not available in the signaling channel anymore.

Furthermore, for BEX with authentication, the end-point authenticates the middlebox and encrypt the sensitive information before sending them. Only the trusted middleboxes who had negotiated for the information are able to decrypt it. Therefore, it is not possible for any other on-path entity to link the end-point's identity to that of the information available.

7.8.4 Trust between End-point and Middleboxes

Our work has been designed on the basis of trust models between the end-points and the middleboxes. In Section 5.2, we have talked about the various methods in which trust can be achieved. Accordingly, we have modeled our solution based on these methods.

Authentication with the help of public key cryptography and certificates is a popular approach to authenticate and authorize a network entity, hence establish trust with them. BEX with Authentication, locates the middlebox certificate which may have been obtained in an out-of-band communication and authenticates the middlebox. The end-point does not proceed with the revelation of sensitive information until and unless the middlebox has been trusted. Once the middlebox has been trusted, the information is sent encrypted such that only the entities trusted with that information can decrypt it.

BEX with selective disclosure enables an end-point to establish trust based on distance (see Section 5.4). The entities which are nearer in the communication path and in the same trusted network can be trusted without explicit authentication. With selective disclosure the end-point trusts the middleboxes with the removal of end-point information parameter. Hence, the sensitive information is removed from the signaling path before the control channel messages are available in the untrusted domain. The mechanism for removal of information parameters exposes the BEX to a risk where important information that are needed for the middlebox service provisioning can be removed by an on-path adversary. In order to prevent such scenarios the middleboxes sign the service offer. The end-point agrees to proceed with selective disclosure of information only after the verification of service signature. Also, the end-point provides a list of information parameters to the middlebox that can be safely removed by it. If a middlebox finds that an information parameter that it has been trusted to remove has already been been removed it has detected foul play in the network. It can then notify the end-point about the presence of a malicious adversary and block the handshake.

7.8.5 Performance

The overhead introduced by our system can be significant when secure methods of revealing end-point information like BEX with authentication or BEX with selective disclosure are used. The reason behind the increase in overhead is the dependence of these mechanisms on cryptographic algorithms. For example, the BEX with Authentication requires the end-points to encrypt sensitive information before sending

and the middleboxes to decrypt the information before processing. With the RSA-based approach for BEX with authentication, the symmetric key used to encrypt the information is sent encrypted using the public key (RSA 1024-bit) of the middlebox. The decryption of the symmetric key at the middleboxes is heavy owing to private key operation.

Similar to other cryptographic protocols, our system also comes with a *performance vs security tradeoff*. Hence, we have built mechanisms and optimizations within the BEX so that the communicating peers can negotiate for an amicable solution. If security is the most important requirement, the BEX can be pointed to run cryptographically stronger schemes like BEX with authentication (RSA-based) approach (139.05 ms). Moreover, communicating peers concerned about the security would readily choose to sacrifice performance in favor of security. If performance is critical to the communicating peers (e.g. Local area network), they can also rely on basic service negotiation with minimal protection of end-point information (60.49 ms). Moreover, for trusted network scenarios like a corporate network, the communicating peers might agree on using selective disclosure mechanisms (100.16 ms) which can provide minimal disclosure with relatively lower overhead compared to BEX with authentication. Hence, it is important for the end-points to exploit the BEX negotiation to agree upon the mechanism which best suits their interests.

Moreover, we have also incorporated some performance enhancing mechanisms to our system. For example, the usage of certificate hints allows the middleboxes to authenticate themselves with the end-point without sending their certificate or signing the service offer(Certificate exchanges are slow). Also, signing of service offers takes 33 ms for a RSA (1024-bit key) signature and 24 ms for a ECDSA (NIST-P-256 key) signature. Hence, our system allows the middleboxes and the end-points to negotiate for tiny details like which signature algorithm to follow. We also observed that using Elliptic Curve Diffie-Hellman (NIST-P-256), the processing overhead of Diffie-Hellman operations (shared key generation) reduced from 11.2 ms to 1.4 ms. Since there are two DH operations in the BEX (in `I2` and `R2`), we were able to minimize the overhead by approximately 20 ms.

Hence, our system is flexible with respect to negotiation for a BEX mechanism which can balance the performance and security requirements of the communicating peers. Moreover, for all of our measurements the middlebox negotiated for information with both the Initiator and the Responder. This may not be the usual case and the middlebox might request for information to only one of the peer. This may also bring down the overhead.

7.8.6 Limitations

The extensive usage of cryptography in our design on one hand allows the security properties of our framework and trust establishment between the network entities. But on the other hand they introduce overheads. For example, the computation of RSA signature on middlebox takes around 33 ms. With an increase in the number of connections handled by the middlebox and the number of the middleboxes on the signaling path, these overheads will rise sharply. But by choosing appropriate mechanisms (basic negotiation, selective disclosure etc.) the communicating entities can bring down the overhead to an acceptable value.

8

Conclusion

The motivation for our work originated from the conflict of interest between the middleboxes and the end-points, such that to provide a solution which can harmonize the information requirements of the middleboxes with the privacy concerns of the end-points. On one hand information from the end-points is critical to the functionality of middleboxes, but on the other hand the end-hosts are sensitive about their private information.

Our design models the solution to the problem in three steps. Firstly, it allows for negotiation of information required by the middleboxes. The middleboxes can request for the minimal amount of information that is necessary for the provisioning of their services and the end-points can verify with their policy that the information requested is necessary and sufficient for the middlebox provisioning. This way the end-point can classify the information requested into sensitive and insensitive. The end-hosts would reveal sensitive information only to the trusted entities. Hence, the amount of information available on the signaling channel can be restricted to only what is necessary for the provisioning of middlebox services.

Secondly, our design extended the basic negotiation for information required to allow the end-points and middleboxes to establish trust with each other. We have identified two ways for establishing trust between the end-points and the middleboxes. The first approach is based on public key cryptography. In this case, the service negotiation is extended to allow the communicating entities to explicitly request for authentication (with the help of certificates). The second approach is more dependent on the network topology. For example, an end-host can trust a network entity if it is within the same trusted network. Since, mutual authentication with cryptographic identities has high overhead, it is often beneficial for the communicating peers to fall back on lighter methods for establishment of trust like the one based on network topology.

Thirdly, the two trust models were further exploited in two different ways to secure the end-point information in the signaling channel. If a middlebox lies in an untrusted network, the end-point authenticates the middlebox before it reveals any

sensitive information. After verification, it ensures that information remains protected while passing through the untrusted network by encrypting it. Only the trusted middleboxes can decrypt the information (e.g. with their own private key). However, when the network entities lies within the same trusted network, it is not necessary for the end-point to encrypt the information. It can trust the middlebox to hide or remove the sensitive information (selective disclosure) from the signaling channel after it has been processed by the middlebox.

The purpose of these different schemes is to allow the end-points and the middleboxes to *negotiate for a performance-security trade-off* and choose a method that suits the best for their requirement.

The implementation of these mechanisms required us to depend extensively on the public cryptography and cryptographic algorithms. The cryptographic algorithms added overhead to our system. Hence, our evaluation focused extensively on the overheads introduced by them in the various scenarios. The basic service negotiation was fast with 60.49 ms and most of the overhead was caused by the context lookup. However, a complete Base Exchange with authentication of middleboxes takes 139.05 ms for RSA-based approach.

However, the high overhead of certain approaches should not discourage the adopters of our system. We have extensively designed the negotiation to cover the different scenarios of performance and security appropriately. For example, within a trusted network the end-hosts can choose to rely on basic service negotiation with no encryption (60 ms). But for scenarios where the end-host communicates with an entity in an untrusted network, it can also choose to go for selective disclosure (100 ms). End-point private information can thus be removed by the edge middlebox before the information leaves the trusted domain. The strength of our system lies in the flexibility with which different approaches can be negotiated within BEX depending on the performance or security considerations.

Apart from the above, we have also went into tiny details into optimizing the computationally expensive operations within the BEX. We suggested a faster static DH-key based approach for BEX with Authentication which allowed us to bring down the timing of Base Exchange to 98.81 ms. Similarly, switching to ECDH key exchange from the regular DH key exchange saved us a critical 20 ms.

Furthermore, we have been careful in designing the different approaches so that our system does not break the verifiability of end-point identities (user and host) and the verifiability of end-point information in the signaling channel. Also, selective disclosure (hiding or removing of end-host information) and the securing the end-point information using encryption are sufficient in ensuring the minimal disclosure of end-point information.

Bibliography

[1] Bauer, D., Blough, D. M., and Mohan, A. Redactable signatures on data with dependencies and their application to personal health records. In *Proceedings of the 8th ACM workshop on Privacy in the electronic society* (New York, NY, USA, 2009), WPES '09, ACM, pp. 91–100. http://doi.acm.org/10.1145/1655188.1655201.

[2] Bellare, M., Canetti, R., and Krawczyk, H. Keying hash functions for message authentication. Springer-Verlag, pp. 1–15.

[3] Burr, W. E. Selecting the advanced encryption standard. *IEEE Security and Privacy 1*, 2 (Mar. 2003), 43–52. http://dx.doi.org/10.1109/MSECP.2003.1193210.

[4] Cameron, K. Laws of identity. http://www.identityblog.com/stories/2004/12/09/thelaws.html.

[5] Carpenter, B., and Brim, S. Middleboxes: Taxonomy and Issues. RFC 3234 (Informational), Feb. 2002. http://www.ietf.org/rfc/rfc3234.txt.

[6] Chaum, D. *Blind signatures for untraceable payments*, vol. 82. Plenum Publishing, 1983, pp. 199–203. http://www.hit.bme.hu/~buttyan/courses/BMEVIHIM219/2009/Chaum.BlindSigForPayment.1982.PDF.

[7] Corporation, O. X.509 certificates and certificate revocation lists (crls). http://docs.oracle.com/javase/1.5.0/docs/guide/security/cert3.html.

[8] Dierks, T., and Rescorla, E. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), Aug. 2008. Updated by RFCs 5746, 5878, 6176. http://www.ietf.org/rfc/rfc5246.txt.

[9] Diffie, W., and Hellman, M. New directions in cryptography. *Information Theory, IEEE Transactions on 22*, 6 (nov 1976), 644 – 654.

[10] Diffie, W., Van Oorschot, P. C., and Wiener, M. J. Authentication and authenticated key exchanges. *Des. Codes Cryptography 2*, 2 (June 1992), 107–125. http://dx.doi.org/10.1007/BF00124891.

[11] Ellison, C. M. Establishing identity without certification authorities. In *Proceedings of the 6th conference on USENIX Security Symposium, Focusing on Applications of Cryptography - Volume 6* (Berkeley, CA, USA, 1996), SSYM'96, USENIX Association, pp. 7–7. http://dl.acm.org/citation.cfm?id=1267569.1267576.

[12] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. Updated by RFCs 2817, 5785, 6266, 6585. http://www.ietf.org/rfc/rfc2616.txt.

[13] Guha, S., and Francis, P. An end-middle-end approach to connection establishment. *SIGCOMM Comput. Commun. Rev. 37*, 4 (Aug. 2007), 193–204. http://doi.acm.org/10.1145/1282427.1282403.

[14] Gurtov, A., Korzun, D., Lukyanenko, A., and Nikander, P. Hi3: An efficient and secure networking architecture for mobile hosts. *Comput. Commun. 31*, 10 (June 2008), 2457–2467. http://dx.doi.org/10.1016/j.comcom.2008.03.014.

[15] Heer, T., Hummen, R., Komu, M., Götz, S., and Wehrle, K. End-host authentication and authorization for middleboxes based on a cryptographic namespace. In *Proceedings of the 2009 IEEE international conference on Communications* (Piscataway, NJ, USA, 2009), ICC'09, IEEE Press, pp. 791–796. http://dl.acm.org/citation.cfm?id=1817271.1817419.

[16] Heer, T., Hummen, R., Wehrle, K., and Komu, M. End-Host Authentication for HIP Middleboxes. IETF Draft Version 4, 2011. http://tools.ietf.org/html/draft-heer-hip-middle-auth-04.

[17] Heer, T., Wehrle, K., and Komu, M. End-Host Authentication for HIP Middleboxes. draft-heer-hip-middle-auth-02, Feb. 2009. http://tools.ietf.org/html/draft-heer-hip-middle-auth-02.

[18] Holt, J. E., and Seamons, K. E. Selective disclosure credential sets. *IACR Cryptology ePrint Archive* (2002), 151–151.

[19] Hummen, R., Ziegeldorf, H., Heer, T., Götz, S., and Wehrle, K. Seams: A signaling layer for end-host-assisted middlebox services. In *Proceedings of the 11th IEEE International Conference on Trust, Security and Privacy in Computing and Communications* (Piscataway, NJ, USA, 2012), IEEE TrustCom-12, IEEE Press.

[20] Infrahip. Hip for linux, hipl. http://www.infrahip.net/.

[21] International Telecommunication Union. *Information technology - Open systems interconnection - The Directory: Public-key and attribute certificate frameworks*. Aug. 2011. http://www.itu.int/rec/T-REC-X.509.

[22] Jokela, P., Moskowitz, R., and Nikander, P. Using the Encapsulating Security Payload (ESP) Transport Format with the Host Identity Protocol (HIP). RFC 5202 (Experimental), Apr. 2008. http://www.ietf.org/rfc/rfc5202.txt.

[23] Kaufman, C. W., Perlman, R., and Speciner, M. *Network Security: Private Communication in a Public World*. Prentice-Hall, Englewood Cliffs, New Jersey, March 1995.

[24] Kaukonen, K., and Thayer, R. A Stream Cipher Encryption Algorithm "Arcfour". IETF Draft Version 3, Mar. 1996. http://tools.ietf.org/html/draft-kaukonen-cipher-arcfour-03.

[25] Kohlas, R., and Maurer, U. M. Reasoning about public-key certification: On bindings between entities and public keys. In *Proceedings of the Third International Conference on Financial Cryptography* (London, UK, UK, 1999), FC '99, Springer-Verlag, pp. 86–103. http://dl.acm.org/citation.cfm?id=647503.728473.

[26] Komu, M., Henderson, T., Tschofenig, H., Melen, J., and Keranen, A. Basic Host Identity Protocol (HIP) Extensions for Traversal of Network Address Translators. RFC 5770 (Experimental), Apr. 2010. http://www.ietf.org/rfc/rfc5770.txt.

[27] Laurie, B. Selective disclosure (v0.2), 2007.

[28] Moskowitz, R., Heer, T., Jokela, P., and T.Henderson. Host Identity Protocol Version 2 (HIPv2). IETF Draft Version 8, Mar. 2012. http://tools.ietf.org/html/draft-ietf-hip-rfc5201-bis-08.

[29] Moskowitz, R., and Nikander, P. Host Identity Protocol (HIP) Architecture. RFC 4423 (Informational), May 2006. http://www.ietf.org/rfc/rfc4423.txt.

[30] Moskowitz, R., Nikander, P., Jokela, P., and Henderson, T. Host Identity Protocol. RFC 5201 (Experimental), Apr. 2008. Updated by RFC 6253. http://www.ietf.org/rfc/rfc5201.txt.

[31] National Institute of Standards and Technology. *Federal Information Processing Standard Publications 46: Data Encryption Standard*. 1993. http://www.itl.nist.gov/fipspubs/fip46-2.htm.

[32] National Institute of Standards and Technology. *Federal Information Processing Standard Publications 197: Advanced Encryption Standard*. 2001. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[33] Nikander, P., Henderson, T., Vogt, C., and Arkko, J. End-Host Mobility and Multihoming with the Host Identity Protocol. RFC 5206 (Experimental), Apr. 2008. http://www.ietf.org/rfc/rfc5206.txt.

[34] Nikander, P., and Laganier, J. Host Identity Protocol (HIP) Domain Name System (DNS) Extensions. RFC 5205 (Experimental), Apr. 2008. http://www.ietf.org/rfc/rfc5205.txt.

[35] Paci, F., Bauer, D., Bertino, E., Blough, D. M., and Squicciarini, A. Minimal credential disclosure in trust negotiations. In *Proceedings of the 4th ACM workshop on Digital identity management* (New York, NY, USA, 2008), DIM '08, ACM, pp. 89–96. http://doi.acm.org/10.1145/1456424.1456439.

[36] Persiano, P., and Visconti, I. User privacy issues regarding certificates and the tls protocol: the design and implementation of the spsl protocol. In *Proceedings of the 7th ACM conference on Computer and communications security* (New York, NY, USA, 2000), CCS '00, ACM, pp. 53–62. http://doi.acm.org/10.1145/352600.352609.

[37] Postel, J. Internet Protocol. RFC 791 (Standard), Sept. 1981. Updated by RFC 1349. http://www.ietf.org/rfc/rfc791.txt.

[38] Ramachandran, A., Bhandarkar, K., Tariq, M. B., and Feamster, N. Packets with provenance, 2008.

[39] Stoica, I., Adkins, D., Zhuang, S., Shenker, S., and Surana, S. Internet indirection infrastructure. *IEEE/ACM Trans. Netw. 12*, 2 (Apr. 2004), 205–218. http://dx.doi.org/10.1109/TNET.2004.826279.

[40] Ylitalo, J., and Nikander, P. Blind: A complete identity protection framework for end-points. In *In Proc. of the Twelfth International Workshop on Security Protocols* (2004).

[41] Zhang, D., Huwaei, and Komu, M. An Extension of HIP Base Exchange to Support Identity Privacy. IETF Draft Version 4, Jan. 2012. http://tools.ietf.org/html/draft-zhang-hip-privacy-protection-04.

[42] Ziegeldorf, H. Enabling and capitalizing origin and flow information on the communication path using identity-based signaling, 2011.