SECURITY PROTOCOL FOR ACTIVE NETWORKS

Lawrence Cheng, Alex Galis
Electrical Engineering Department, University College London, Torrington Place, London, UK. WC1E 7JE
e-mail: {l.cheng, a.galis}@ee.ucl.ac.uk

Abstract – Active packets carrying management and control code have a dynamic nature and support dynamic routing. Thus, active packets must be protected in an end-to-end and hop-to-hop fashion. In this paper, we present a novel approach, known as Security Protocol for Active Networks (SPAN), which enables an active packet to be securely transmitted *during* (instead of after) Security Association (SA) and management negotiations along a new execution path.

1. INTRODUCTION

It was identified in [2][6][10][24][25][26][32] that there is a need for an *end-to-end* and *hop-to-hop* security approach for active networks due to the *dynamic nature* and *dynamic routing* capability of active packets; details can be found in [2][32]. This paper suggests that an end-to-end and hop-to-hop active network security management protocol must be *efficient* i.e. to reduce as much as possible the performance overhead generated by hop-to-hop SA establishment. Furthermore, as active packets may traverse through heterogeneous administrative domains, the protocol must be *flexible* i.e. allows Security Association (SA) negotiations between active nodes of different administrative domains. The protocol must *not* rely on centralised servers, and should reduce the number of message exchanged and computational processes for key establishment, in order to enhance the *scalability* of the approach. Furthermore, the protocol should be secure i.e. should support anti-replay and man-in-the-middle attacks; and the protocol should be able to identify legitimate requests from DoS attacks as efficiently as possible [32].

We have discussed existing approaches in [2][32]. A summary will be provided in this paper. Asymmetric cryptography requires encrypting, creating, and verifying signatures of *every* modifications on *every* active packets on *every* executing node, which is not scalable [2][32]. Shared key *pre-distribution* does not support shared key negotiation, and it is not practical to be deployed in a large scale network because each pair of hop must be equipped with different shared keys in order to achieve authentication (essentially the same problem experienced in multicast IPSec [13]) [32]. The Keying Server (KSV) approach in [6] is not scalable; Secure Active Network Environment (SANE) recommends a set of workarounds for hop-to-hop key establishment [10][11][24] but the workarounds doe not scale. In Secure Active Node Transfer System (SANTS) [1], *hop-to-hop key establishment* was not addressed. Signed Key Transport (SKT) [5] has limited flexibility. Traditional security management approaches (such as IKE, Kerberos, Oakley, ISAKMP… etc.) must be refined to create less overhead when deployed in a *hop-to-hop* fashion [32]. The Simple Key Exchange for Active Networks (SKEAN) [32] approach was the first approach that attempts to address practical security management in active networks. However, the initial design of SKEAN did not take into account of DoS attacks. The Just Fast Keying (JFK) [19] protocols claim to be DoS-resistant. However, as we will discuss in later section, our proposal, known as Security Protocol for Active Networks (SPAN), is capable of detecting DoS attacks at a much earlier stage than IKEv2, IKEv1 in aggressive mode+IPSec, SKEAN, and JFK.

2. THE SPAN PROTOCOL

Because active networks reside in the core network [23][32], public IP addresses are assumed. Individual node security (i.e. firewalling, packet filtering… etc.) and SA maintenance on nodes are outside the scope of this paper, thus secure storage of keying materials and active node integrity are assumed. Active packets are currently implemented as UDP packets [4][22]. According to the active node architecture [22][26], each active node has only one NodeOS[1], and the NodeOS hosts the security facility that provides security services to all locally hosted EEs/AAs i.e. *a transparent security approach*. Thus, SPAN focuses on supporting secured negotiations and secured active packet transmission between *NodeOSs* of active nodes. As active nodes reside in the core network on the Internet [23][32], we assume that active nodes have access to PKI. Since PKI is a common infrastructure for non-repudiation protection e.g. embedded in all web browsers, we assume that each NodeOS has its own PKI key pair and certificate. But for scalability, we assume that the number of EE involved in an active network may be of large number [32]; thus each EE *may* or *may not* have its own public key pair. We do not address administrative issues in this paper i.e. how Certificate Authorities (CAs) verify actual ownership of valid PKI certificates: we assume the integrity of legitimate PKI public key pair owners. Thus, if a peer uses legitimate keys for signing data, he/she would be traceable i.e. non-repudiation protection enforced through PKI. Because we assume node and key storage integrity and the integrity of PKI public key pair owners, we assume that any requests with

[1] A NodeOS is a software platform installed on high speed routers (i.e. passive nodes) in the core network of today's Internet. A NodeOS provides essential functionalities such as security support and de-multiplexing to support the operation of active packets [3][21][23].

valid signature are *legitimate* requests; else they are attack messages. We assume attackers are capable of intercepting *all* messages on the Internet, and are able to create/modify *all* types of messages.

The SPAN protocol involves an exchange of three messages only to complete both SA and Execution Environment[2] (EE) (optional) negotiation *and* secured active packet transmission. The SPAN exchange involves two peers: an Initiator (I) and a Responder (R). The Initiator is the NodeOS which wishes to start a hop-to-hop SA and EE (optional) establishment process for secured active management packet transmission; the Responder would be the NodeOS which is about to receive a request for hop-to-hop SA and EE (optional) establishment process.

```
HDR_INIT, SAi, [EEi], [CERTi], D-Hi,
        NONCEi, AUTHi →
```

Fig. 1. SPAN_INIT

The first message (SPAN_INIT) is sent from the Initiator to the Responder, and is shown in Fig. 1. HDR_INIT is the SPAN_INIT message header. SAi is a set of security association parameters offered by the Initiator to the Responder. These parameters are for example the supported or preferred encryption algorithms, supported or preferred key size… etc. D-Hi and NONCEi are the Diffie-Hellman (D-H) public value and a random 128-bit, never reused nonce [8] generated by the Initiator. The nonce is needed for anti-replay attacks (see later section). Both D-Hi and NONCEi are required for symmetric secret establishment between the Initiator and the Responder [8][14]. Items in square brackets are optional. [CERTi] is the PKI certificate of the Initiator. [EEi] is included only when the Initiator needs certain specific parameters or information regarding an existing remote EE *prior to active packet creation*. For example, if a principle is about to create an active packet for (re)configuring a video-service guarantied EE that is residing on a remote active node, the principle might need certain information regarding that remote EE i.e. the type of QoS controllers being used e.g. tc, DiffServ… etc, so that it can construct the active (re)configuration packet in the way that is supported by the remote EE. This arrangement in SPAN therefore enhances the level of flexibility of SPAN in the sense that the principle can now make authenticated and integrity protected queries for remote management information and receive *protected* responses prior to active packet creation (see later). AUTHi is a digital signature that is created by using the Initiator's private key that covers all items contained in SPAN_INIT except CERTi and itself. This signature is essential as it is used for authentication, integrity, and non-repudiation protection on the exchanged materials that are generated by the Initiator. Furthermore, the inclusion of this signature in SPAN_INIT enables more efficient detection of DoS attacks in SPAN (see later section). Fig. 2 shows the list of items that are digitally signed by the Initiator.

```
HDR_INIT, SAi, [EEi], D-Hi, NONCEi
```

Fig. 2. Items signed in AUTHi

Upon receiving the first message i.e. SPAN_INIT at the Responder, the Responder verifies the (digitally signed) materials in SPAN_INIT by using [CERTi] and AUTHi. If the digitally signed items cannot be verified, the Responder stops proceeding further because SPAN_INIT might have been subjected to man-in-the-middle attacks, or was created for DoS attacks (see later section). If the signature is verified (hence the contents of SPAN_INIT), the Responder will look into the message. If an EEi is included, the Responder will evaluate the corresponding (locally residing) EE against the list of requested parameters in EEi. There are two possible outcomes: 1) the Responder is *unable* to response to the requests as specified in EEi. This could happen for example when the targeted EE no longer exists on the Responder, or the EE has already been re-configured such that its current operational status is not as expected by the Initiator… etc. In this case, the active packet is simply forwarded to the Responder's neighbouring active node other than the originating node i.e. the Initiator, where the SPAN protocol exchange may potentially continue; 2) the Responder is able to response to the requests as specified in EEi. It creates a list of its responses, and stores them in an EEr payload. In case 2 or in the case where no EEi is included in SPAN_INIT, the Responder generates its own D-H public value (D-Hr) and a random 128-bit nonce (NONCEr). By using these values in conjunction with the Initiator's values i.e. D-Hi and NONCEi, the Responder is capable of creating a SKEYSEED using the D-H algorithm. SKEYSEED is a shared secret from which a subsequent set of keys can be generated for specific purposes (e.g. authenticity protection, integrity checks… etc.) [8].

Once the Responder has computed the shared secret and the subsequent shared key set, it responses to the Initiator with the 2nd message (SPAN_AUTH) which is shown in Fig. 3. Note that the items quoted in curly brackets {...} are protected accordingly as embedded payloads in the same Encrypted payload, by using the shared keys derived from the shared secret SKEYSEED i.e. SK_e for encryption, SK_a for integrity protection, note that one key for one direction [8]. The Encrypted payload is appended with integrity protection data – in this case a keyed hash value – that covers the *entire* message of SPAN_AUTH (including message header).

```
← HDR_AUTH, SAr, [CERTr], D-Hr, NONCEr,
        AUTHr, {[EEr], IDr}
```

Fig. 3. SPAN_AUTH

HDR_AUTH is the message header. SAr keeps the Responder's choices on SAi. [CERTr] is the PKI certificate of the Responder. D-Hr and NONCEr are needed by the Initiator to create the shared secret SKEYSEED, and therefore must be listed in cleartext i.e. not encrypted. AUTHr is a digital

[2] EEs are software modules that can be viewed as resource abstractions to support services – in the form of active packet execution - on an active node [3][21][22][23].

signature created by using the Responder's private key over a list of items (Fig. 4).

```
HDR_AUTH, SAr, D-Hr, NONCEr, NONCEi
```

Fig. 4. Items signed in AUTHr

The idea of digitally signing the items listed in Fig. 4 is to enable the Initiator to verify the non-repudiation, integrity and authenticity of the parameters from the Responder. Note that the Initiator's nonce (`NONCEi`) is also digitally signed in `AUTHr`. This arrangement is necessary to prevent replay attacks (see later section). Also note that the Responder does not simply sign any anonymous nonce values. The Responder must first verify `NONCEi` (that was included in `SPAN_INIT`) by verifying the value against the digital signature (`AUTHi`) made by the Initiator *prior to* digitally signing `NONCEi` (see later section). `{[EEr], IDr}` is the protected response to EE information request and the identity of the Responder that will be used for future identification respectively. They are protected by using the appropriate shared key i.e. `SK_e` and `SK_a`. `IDr` does not necessary to be the identity of the Responder as listed in `CERTr`. It could be any form of identifier (e.g. IP addresses, host names… etc.) that the Responder considers to be appropriate to be used in future for identifying itself. These information are protected so that the protected response payload can be used by the Responder as a proof-of-knowledge of the shared key set i.e. a *precaution* step (see later section).

Upon receiving `SPAN_AUTH` from the Responder, the Initiator must first verify `AUTHr` using `CERTr`. If the verification process is successful, the Initiator would be able to generate the shared secret i.e. `SKEYSEED` and the subsequent shared keys by using `D-Hi`, `D-Hr`, `NONCEi`, and `NONCEr`. The Initiator can then use the corresponding keys derived from the shared secret to decrypt the encrypted items in `SPAN_AUTH` i.e. `{[EEr], IDr}`, and verifies the integrity of the entire message by using the corresponding shared key i.e. `SK_e` and `SK_a` respectively. If the authenticity and integrity of the `SPAN_AUTH` message is verified, and if the Initiator is the originating node, the Initiator sends to the Responder the third, and the last message: `SPAN_AP`, which is shown in Fig. 5.

```
HDR_AP, {IDi, NONCEr, active_packet,
            code_sig} →
```

Fig. 5. SPAN_AP

`HDR_AP` is the header of a `SPAN_AP` message. Note that all items in curly brackets are protected as embedded payload in an `Encrypted` payload. The payload is appended with a keyed hash value that covers the *entire* message for integrity protection. `IDi` is the identity of the Initiator. `NONCEr` is protected so that the Initiator can acknowledge to the Responder that it has received the correct nonce value and for anti-replay attacks (see later). This value does not need to be digitally signed by the Initiator because it is protected by the authenticated and integrity verified shared keys i.e. `SK_e` and `SK_a` (see later section). `active_packet` contains the *entire* active packet i.e. both the static code and dynamic data. The dynamic data would be the execution results of the static code on the originating node. `code_sig` is the static code signature that is created for end-to-end protection by using the *principle's* private key. Remember that the principle is the actual creator of the code. In cases where the principle does not have its own public key pairs (see assumption), the private key of the NodeOS on which the principle is residing on may be used for signing instead. This arrangement is more scalable at the expense of a less ideal non-repudiation protection (see later section for details).

Once the Responder receives `SPAN_AP`, the protected items in the message are subjected to verification by using the established shared key set. The static code in the active packet is verified against the digital signature i.e. `code_sig`. If the verifications are successful, the embedded code in the active packet is executed. Under this arrangement, the hop-to-hop authenticity, integrity, and confidentiality of the dynamic data of the active packet are protected by the shared key set. The static code is digitally signed by the principle, so the source authenticity and integrity of the static code is verified i.e. end-to-end protection enforced. The confidentiality of the entire active packet including both static code and dynamic data is protected by the shared key set.

Once the first Responder has executed the active packet, results of code execution i.e. new dynamic data will be added back to the packet, and the packet will be forwarded to its next hop i.e. the second Responder. The SPAN protocol repeats along the execution path but no new signature on the static code is created; this is because the code is static and should be verified by verifying the *principle's* authenticity. Once a SA has been established between a pair of active nodes, the SA should be retained. The established SA is identified at each peer by the respective Security Parameter Index (SPI) (that was assigned by each peer during the protocol exchange). By including an Initialisation Vector (IV) [16] in the SPAN `Encrypted` payload header, we can reuse the shared key set generated after a SPAN exchange to protect subsequent active packets travelling along the *same* execution path, as long as the SA has not expired [16].

3. DESIGN CONSIDERATIONS

3.1 Proof-of-Knowledge of Shared Key and Key Binding

In `SPAN_AUTH`, `IDr` and `[EEr]` are protected by using the established shared key set. This is a safety precaution step so that the Initiator knows the Responder has computed correctly the same shared key set, prior to using the shared key set to protect *subsequent* communications. Additionally, this arrangement enables a binding to be established between the peer's identity with the established shared key set which eliminates identity mis-binding attacks [18].

3.2 Replay, Man-in-the-Middle, DoS Attacks

A typical form of replay attack is that the attacker copies a legitimate message, and re-sends the message to one of the peers or other peers. To provide anti-replay protection, all messages exchanged are cryptographically protected. Particularly, randomly generated, never reused, authenticated and integrity protected 128-bit nonces are used [7][8]. Each peer in SPAN must also either digitally sign or use symmetric cryptography to protect the authenticity and integrity of each other's nonce: otherwise an attacker can copy a legitimate message from the Responder in one session, and impersonate the Responder on a key exchange with another party [18].

A common form of man-in-the-middle attack on the D-H algorithm would be the attacker pretends it is the Initiator or the Responder. Suppose the attacker pretends it is the Responder. Upon receiving `SPAN_INIT` from the Initiator, it creates the a forged `SPAN_AUTH` message. Because the attacker does not have the Responder's valid private key, he can only create a forged signature i.e. `AUTHr*` using another private key (for instance, his own private key). When this forged `SPAN_AUTH` is received by the Initiator, the Initiator would be able to determine immediately the message is forged because the signature (i.e. `AUTHr*`) cannot be verified against the certificate of the legitimate Responder (i.e. the actual owner of `CERTr`).

Key exchange protocols are subjected to DoS attacks [17]. For example, a DoS attacker can either create (large number of) legitimate key establishment instantiation requests in an attempt to overload the Responder [27]; or they can flood the Responder with initialisation requests with forged IP addresses [8]; or they can randomly modify the encrypted payload of a legitimate request message, causing a cache miss at the Responder [19]. For example, an IKEv2 Responder - upon receiving an initialisation message i.e. message 1 from an attacker - would be wasting computational resources to create (new) D-H values (for message 2), computing shared key sets (upon receiving valid/invalid message 3 from the Initiator), and will try to decrypt the encrypted message 3 from the Initiator, prior to verifying the authenticity of the Initiator[3]. The same problem is experienced in JFK [19]: a JFK Responder - upon receiving the first message from the Initiator (in this case an attacker) - would be wasting resources on computing (new) D-H exponentials, creating digital signature over the D-H exponentials, creating a keyed hash over keying materials, computing the shared key set (upon receiving message 3), and try to decrypt the encrypted contents in message 3, prior to verifying the Initiator's digital signature[4]. Our previous proposal i.e. SKEAN experiences a similar level of DoS attacks as IKE and JFK, that the SKEAN Responder generates signature prior to verifying the Initiator.

The first countermeasure of SPAN against DoS attacks is that a SPAN Responder may carry out *essential* operations only until it can verify whether the request is a legitimate request or part of a DoS attack. Note that SPAN is capable of verifying the Initiator's authenticity when the Responder receives the *first* message in the protocol i.e. `SPAN_INIT`. This is because SPAN requires the Initiator to digitally sign `SPAN_INIT` using its valid PKI private key. If the signature on `SPAN_INIT` cannot be verified, the Responder will not proceed further. As discussed in the assumptions, we assume all authenticated request messages (i.e. with valid digital signatures) are legitimate requests. Thus, SPAN can quickly identify legitimate requests from DoS attacks (see later on evaluation). To address another form of DoS attacks that involves random modification of encrypted contents of duplicated message 3, the SPAN Responder caches the corresponding SPI values of message 3. Thus, a duplicated (and/or malformed) message 3 can be quickly detected and dropped by the Responder simply by matching SPI values in the message header and those values in its cache.

One may argue that the SPAN Responder is wasting resources on checking digital signatures upon receiving a `SPAN_INIT` message; as such, the SPAN approach *seems to be* less ideal than the one deployed in a variant of IKEv2 that uses COOKIE for limiting DoS attacks originated from spoofed IP addresses. This variant of IKEv2 involves an exchange of six messages (rather than four), and the IKEv2 Responder verifies the Initiator in the *fifth* message in its exchange. In brief, an IKEv2 Responder can be configured to reject initialisation requests, and responses to the Initiator with an unprotected message that contains a COOKIE. The IKEv2 Initiator must then *resend* the initialisation message with the valid COOKIE to prove that it is using a the same IP address as the one used in the (rejected) first initialisation message. The IKEv2 RFC claims this arrangement can make DoS attacks less effective [8], and would enable the protocol to start with weak authentication (of IP addresses) and possibly later performing stronger authentication [27]. However, we argue that the use of COOKIE in IKEv2 does not solve the problem: this is because attackers can intercept and modify all messages from the Responder (assumed). Thus, an attacker can generate the first initialisation message with a forged IP address, and upon intercepting the COOKIE from the Responder, it can resend the same forged initialisation message with the valid COOKIE but still using the same forged IP address. The IKEv2 Responder would be tricked to believe the IP address used by the Initiator is valid; and would then carry out all the computational expensive processes i.e. generating D-H values or computing share key set… etc. Therefore, we argue that, essentially, the very first thing that a SPAN Responder should do - in order to distinguish legitimate requests from DoS attacks - is to verify the authenticity of the SPAN Initiator prior to carrying out any further processing. This is because - as explained in earlier section - all request messages that do not have a valid

[3] In IKEv2, the Initiator's only signature i.e. AUTH on keying materials is kept in an *encrypted* payload in message 3. Thus, the Responder must compute the shared key set, prior to being able to verify the Initiator's signature.

[4] JFK suggests a mechanism to address DoS attacks by requiring the Responder to periodically generating D-H exponential tuples (every 30s), and use a FIFO approach for assigning D-H exponentials to Initiator's requests. But a JFK Responder would still be wasting resources for all other computational expensive processes e.g. computing signature, computing shared key set… etc.

authenticator i.e. digital signature should be considered as DoS attacks. Although one may argue this would not eliminate DoS attacks completely i.e. still requires some resources to detect DoS attacks, but we argue that: 1) this is the only arrangement that would enable the Responder to distinguish legitimate request messages from DoS attacks at the *very first stage* of the protocol exchange *prior to any other computational expensive processes* such as D-H exponential computations or shared key set computations… etc.; 2) the rule-of-thumb in key exchange design to *reduce* the impact of DoS attacks or to *limit* DoS attacks [18][19][20][27]: in the evaluation section, we will show that our approach enables much *rapid detection of DoS attacks* than existing approaches i.e. less impact on the Responder; 3) the cost of signature verification (at the Responder) can be reduced [29] (or even neglected [28]) by using carefully selected parameters for asymmetric algorithms: for example use a relatively small public exponent `e` but larger values for secret prime numbers `p` and `q` [28][29] to achieve quicker RSA signature verification. It was discussed in [28][29][30][31] that with careful selection of parameters, we can improve RSA performance but does not lower the security of the protocol[5].

3.3 Comparing with Related Work

Existing protocols either do not support SA and EE negotiation which limits their flexibility to cope with heterogeneity between active nodes of different administrative domains (e.g. pre-distributed shared key, SKT, SANE), or follows a centralised approach which is not scalable (e.g. KSV), or does not provision for key management (e.g. SANTS), or their performance was not optimised for hop-to-hop deployment (e.g. IKE, Kerberos, Oakley, ISAKMP… etc.); or wastes more resource prior to detecting DoS attacks (e.g. IKEv1, IKEv2, JFK, SKEAN… etc.) (see later for evaluation).

4. EVALUATION RESULTS

We have developed a prototype of SPAN and the relevant components of IKEv2[6] in Java respectively. We used two laptops (each with an Intel Pentium M processor 1.70 GHz, 796 MHz cache with 1 GB RAM), one as the Initiator and the other one as the Responder. In each trial, a dummy active packet (of 1024 bytes with a static code of 512 byte) is transmitted securely between the two peers during the SPAN protocol exchange as specified earlier in this paper. All encryption uses DESede in CBC mode and digital signatures are created using DSA. To avoid long and variable delay on D-H parameter generation, we used the 1024-bit prime modules and base generators that are used by SKIP [9][7]. We used a set of private values (e.g. 240, 241, 242) as the Exchange Type of our SPAN messages. SPAN specific payload types (e.g. `EEi`, `EEr`… etc.) are specified by using private values in-between 128-255.

As an initial assessment, we compared the SPAN performance with 1) IKEv2+IPSec without Perfect Forward Secrecy (PFS), and 2) IKEv2+IPSec with PFS support (new D-H values). PFS is defined in [12]. PFS is optional [15] because it enables *strong* security in certain situations [13], but incurs a high performance overhead because new D-H values are generated [15]. We measured the time it takes to complete one SA establishment (excluding packet execution time which is application-specific). Our experiment results show that SPAN generates a 15% to 40% less in performance overhead when comparing to IKEv2+IPSec without/with PFS support (3103.76 milliseconds for SPAN, 3652.78 milliseconds for IKEv2+IPsec without PFS, 5177.315 milliseconds for IKEv2+IPSec with PFS).

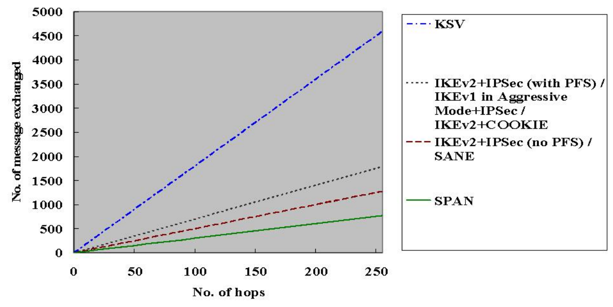

Fig. 6. No. of message exchanged Vs. no. of traversed hops

As an initial attempt for scalability evaluation, we compare SPAN with IKEv2+IPSec (with/without PFS), SANE, IKEv1 in aggressive mode+IPSec, IKEv2+COOKIE, and KSV by determining the number of messages that are required to be exchanged between peers in order to complete the protocols respectively along an execution path of 256 active nodes i.e. the maximum Time-to-Live (TTL) value (to simulate large scale deployment). As shown in Fig. 6, SPAN scales better than existing approaches especially under large scale of deployment.

5. CONCLUSION

In this paper, we presented SPAN which is a *secure*, *scalable*, *efficient*, and *flexible* hop-to-hop SA and EE establishment protocol for active networks, that enables secure management information exchange and active packet transmission *during* SA negotiation, instead of after. Our initial experimental results show promising results that SPAN achieves 15% to 40% less in performance overhead when comparing to some of the existing approaches; and SPAN is designed to detect

[5] Attacks on a message *encrypted* by using low-exponent RSA (public key) is possible, which enables the recovery of the plaintext [30]. But this does *not* affect the use of low-exponent RSA in SPAN because we use RSA private key for *signatures*, rather than encryption. Details on parameters selection can be found in [29][31],

[6] The reason for developing (relevant parts of) IKEv2 is that at the time when our implementation starts (early 2005), no open source of IKEv2 was available.

[7] These algorithms were chosen for implementing our proof-of-concept prototypes for *evaluation* purposes only.

DoS attacks much more efficiently than some existing approaches.

6. REFERENCES

[1] S. Murphy, E. Lewis, R. Puga, R. Watson, “Strong Security for Active Networks”, IEEE OpenArch 2001.

[2] L. Cheng, A. Galis, “Strong Authentication for Active Networks”, IEEE-Softcom 2003, http://www.ee.ucl.ac.uk/~lcheng/Papers/SOFTCOM_2003.pdf

[3] T. Suzuki, et al., “Dynamic Deployment & Configuration of Differentiated Services Using Active Networks”, IWAN 2003, http://www.ee.ucl.ac.uk/~lcheng/Papers/IWAN_2003.pdf

[4] J. Moore, M. Hicks, S. Nettles, “Practical Programmable Packets”, IEEE-INFOCOM 2001.

[5] S. Murphy, A. Hayatnagarkar, S. Krishnaswamy, “Prophylactic, Treatment and Containment Techniques for Ensuring Active Network Security”, IEEE DARPA, 2003.

[6] S. Krishnaswamy, et al., “A Prototype Framework for Providing Hop-by-Hop Security in an Experimentally Deployed Active Network”, IEEE-DANCE 2002.

[7] Krywaniuk, et al., “Using ISAKMP Message IDs for Replay Protection”, Internet Draft: draft-krywaniuk-ipsec-antireplay-00.txt, July 2001, http://www3.ietf.org/proceedings/01aug/I-D/draft-ietf-ipsec-antireplay-00.txt

[8] C. Kaufman, “Internet Key Exchange (IKE v2) Protocol”, RFC4306, Dec 2005, http://www.rfc-archive.org/getrfc.php?rfc=4306

[9] Java Cryptographic Extension (JCE), http://java.sun.com

[10] M. Hicks, A. Keromytis, J. Smith, “A Secure PLAN”, IEEE Transactions on Systems, MAN, and Cybernetics, VOL.33, NO.3, AUGUST 2003.

[11] D. S. Alexander, W. Arbaugh, A. Keromytis, J. Smith, “A Secure Active Network Environment Architecture: Realization in SwitchWare”, IEEE Network, June 1998.

[12] W. Diffie, P. Oorschot, M. Wiener, “Authentication and Authenticated Key Exchanges”, Designs Codes and Cryptography, 2, 107-125, 1992.

[13] N. Doraswamy, D. Harkins, “IPSec: The New Security Standard for the Internet, Intranets, and Virtual Private Networks”, 2nd edition, ISBN: 0-13-046189-X, Prentice-Hall PTR, 2003, pp.220-232.

[14] E. Rescorla, “D-H Key Agreement Method”, RFC2631, June 1999.

[15] B. Springer, L. Kilmartin, “Performance Evaluation of the IKE Protocol under Dynamic VoIP Network Conditions”, in proceedings of the Irish Signals and Systems Conference, Limerick, Ireland, 2003.

[16] “Initialization Vector”, http://en.wikipedia.org/wiki/Initialization_vector

[17] Mailing list, ”UDP DoS attack in Win2K via IKE”, http://marc.theaimsgroup.com/?l=bugtraq&m=100774842520403&w=2

[18] H. Krawczyk, “SIGMA: the SIGn-and-Mac Approach to Authenticated Diffie-Hellman and its Use in the IKE Protocols”, in Advances in Cryptography – CRYPTO 2003 Proceedings, LNCS 2729, Springer, 2003, http://www.ee.technion.ac.il/~hugo/sigma.html

[19] W. Aiello, S. Bellovin, M. Blaze, R. Canetti, J. Ioannidis, A. Keromytis, O. Reingold, “Efficient, DoS-Resistant Secure Key Exchange for Internet Protocols”, ACM Computers and Communications Security Conference (CCS), 2002.

[20] P. Karn, W, Simpson, “The Photuris Session Key Management Protocol”, draft-ietf-ipsec-photuris-03.txt, September 1995.

[21] S. Denazis, et al., “D7-FAIN Active Node Architecture and Design”, Deliverable 7, May 2003, http://www.ist-fain.org/deliverables/del7/d7.pdf

[22] K. Calvert, et al., “Architectural Framework for Active Networks”, draft version 1.0, July 27, 1999, http://protocols.netlab.uky.edu/~calvert/arch-latest.ps

[23] S. Denazis, S. Karnouskos, T. Suzuki, S. Yoshizawa, “Component-based Execution Environments of Network Elements and a Protocol for their Configuration”, IEEE-Transactions on Systems, Man and Cybernetics, 2003.

[24] W. Arbaugh, A. Keromytis, D. Farber, J. Smith, “Automated Recovery in a Secure Bootstrap Process”, Network and Distributed System Security Symposium, Internet Society, March 1998, pp.155-167.

[25] S. Alexander, “Security in Active Networks”, Secure Internet Programming: Issues in Distributed and Mobile Object Systems, 1999.

[26] S. Murphy, “Security Architecture for Active Nets”, Nov 2001, http://protocols.netlab.uky.edu/~calvert/sec-latest.ps

[27] P. Eronen, “Denial of service in public key protocols”, in Proceedings of the Helsinki University of Technology Seminar on Network Security, December 2000, http://www.niksula.hut.fi/~peronen/publications/netsec_2000.pdf

[28] K. Matsuura, H. Imai, “Modification of Internet Key Exchange Resistant against Denial-of-Service”, in Proc. of Internet Workshop 2000 (IWS2000), pp.167-174, Feb. 2000.

[29] Y. Zheng, “Digital Signcryption or How to Achieve Cost(Signature & Encryption) << Cost(Signature) + Cost(Encryption)”, in Advances in Cryptology – Crypto 1997, pp.165-179, Springer-Verlag, LNCS 1294, Berlin, August 1997.

[30] D., Coppersmith, M., Franklin, J. Patarin, M. Reiter, ”Low-exponent RSA with related messages”, in Advances in Cryptology – EUROCRYPT 1996, vol. 1070 of LNCS, Springer-Verlag pp. 1-9.

[31] A. Odlyzko, “The future of integer factorisation”, CryptoBytes 1 (1995) 5-12.

[32] L. Cheng, A. Galis, “Simple Key Exchange for Active Networks”, IEEE-ICON 2005, http://www.ee.ucl.ac.uk/~lcheng/Papers/ICON_2005.pdf