

# Experimental Analysis of the Arcanum Key Exchange Protocol

Ajmal S. Mian

School of Computer Science & Software Eng.  
The University of Western Australia  
35 Stirling Hwy., Crawley, WA 6009, Australia

Raja Iqbal

Computer Science Department, MCS  
National University of Science & Technology  
Tamizuddin Rd., Rawalpindi, Pakistan

## Abstract

*A VPN establishes a secure network using the insecure media of the Internet. However, before a VPN can be established keys must be exchanged between the communicating peers. In this paper, we present the implementation details of the Arcanum key exchange protocol along with its experimental analysis. We simulated a number of active and passive attacks in order to test the robustness and efficiency of the protocol. Our results show that Arcanum protocol is robust against passive eavesdropping, reflection attack, replay attack, redirection attack, man-in-the-middle attack and DoS attacks.*

**Keywords:** VPN, Internet Key exchange, Diffie-Hellman key generation, security, authentication.

## 1. Introduction

The Internet traffic is generally unencrypted and can be copied, deleted, modified, replaced, replayed, reflected or redirected by anyone who has access to the Internet media. In addition to the privacy and integrity of the data, the authenticity of the node sending the data is also questionable. These vulnerabilities of the Internet call for a more secure network. Such networks are known as Virtual Private Networks (VPN). IPSec [7], L2TP and PPTP are some well known protocols designed to establish a VPN. VPNs establish a secure network using the insecure infrastructure of Internet. Security is achieved by establishing encrypted tunnels between authenticated peers. VPNs offer secrecy and integrity of the data as well as authenticity of the node sending the data. In some cases it may also offer non-repudiability. However, before a VPN can function, keys must be exchanged between the communicating peers with the help of a key exchange protocol. Examples of key exchange protocols include Arcanum [11], IKE [5], IKEv2 [6], SIGMA [8], Photuris [13], JFK [1], Oakley [12] and SKEME [9].

Mian and Masood [11] proposed a novel key exchange

protocol (Arcanum) and carried out its theoretical analysis and comparison with IKE, IKEv2 and JFK. In this paper, we present the implementation details and experimental analysis of the Arcanum key exchange protocol. We simulated a number of active and passive attacks in order to test the robustness and efficiency of the Arcanum protocol. These attacks include passive eavesdropping, reflection attacks, replay attacks, redirection attacks, man-in-the-middle attacks and DoS attacks. Our results show that the Arcanum protocol is robust against all attacks. However, a man-in-the-middle attack could only cause a partial DoS and compromise of the initiator's identity only in Digital Signature mode.

## 2. Implementation of Arcanum Protocol

Implementation of Arcanum required a number of encryption algorithms, hash functions, HMAC functions, digital signature algorithms and Diffie-Hellman (DH) key generation to be implemented (see Section 2.10). Digital signature (RSA) algorithm and DH key generation were implemented from scratch. However, off-the-shelf library functions were used for the remaining algorithms. We implemented Arcanum at the application layer for our simulations (source code available at [10]). We chose Java for implementing Arcanum because it can handle multitasking and has a large number of security related libraries. Multithreading was required to cater for multiple Security Association (SA) negotiations in progress at a single node.

The design of Arcanum protocol comprises of two main threads. The initiator thread and the responder thread. These threads have further child threads. Any number of these child threads can be spawned so that at a given time a peer can initiate multiple SA requests and be a responder for multiple SA requests that arrive from other peers. The following subsections give a brief description of the different threads of the software. All messages of the protocol use the ISAKMP [3] header and payloads. For details of the message contents of the protocol the reader is referred to [11].

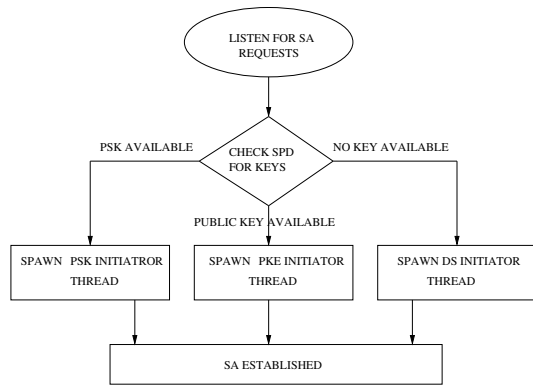


Figure 1. The main initiator thread spawns a child thread depending upon the local security policy.

### 2.1 The Main Initiator Thread

This thread waits for SA requests arriving from other security protocols like IPSec. Once an SA request arrives, this thread looks into the Security Policy Database (SPD) in order to decide which type of SA is to be initiated. It then spawns the corresponding child thread. There are three different methods for establishing an SA depending upon the type of authentication used. These methods are Digital Signature (DS), Pre-shared Key (PSK) and Public Key Encryption (PKE). The choice of authentication depends upon the local policy. Fig. 1 shows the flow diagram of the main initiator thread.

### 2.2 The Main Responder Thread

This thread continuously listens for SA requests arriving from initiators and responses arriving from other responders to own requests. When a message is received, this thread looks at the Exchange Type field in the ISAKMP header to find out if it is an SA request or a response to an SA request locally generated. In the former case it opens a child responder thread and in the latter case it passes the message to the concerned initiator thread. To avoid DoS attacks in PKE and DS modes, upon receiving message 1 this thread opens a small thread which dies immediately after replying without making any state. When a message 3 arrives another responder thread is generated which verifies message 3 before proceeding further.

### 2.3 The DS SA Initiator Child Thread

Fig. 2 shows the flow diagram of the DS SA initiator child thread. This thread is spawned generally when there is no PSK with the responder and the public key is also not available. This thread sends message 1 with SA proposals, a nonce and the initiator’s DH public value. A retransmission timer is set which retransmits message 1 in case a

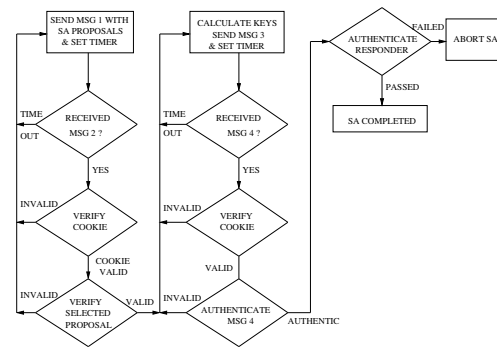


Figure 2. The PSK and DS SA initiator child thread. The thread tries its best to deliver its messages to the responder.

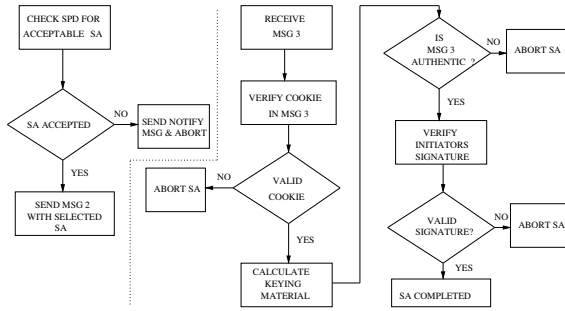
reply is not received. The maximum limit for retransmissions was set to 10 however, it is a local policy matter and can be changed according to the reliability of the network. Message 1 is also retransmitted if the received response is invalid. On the arrival of a valid message 2 all the keying material is calculated and message 3 is sent after encryption and authentication. This message contains the initiator’s signature and certificate. The responder’s DH public value is not sent back, instead only the  $KE_{rid}$  sent back to save message space.

The thread again makes 10 attempts to get a valid and authenticated response back from the responder. If an authenticated message arrives, it is decrypted and the responder’s signature is verified using his public key. If the signature is valid the SA is established successfully.

### 2.4 The DS SA Responder Child Threads

Fig. 3 shows the flow diagram of this thread. There are two different threads that respond to the DS SA request (see Fig. 3). The first one replies to message one and dies without making any state. It chooses an acceptable proposal and gets a DH public value from the DH key stack and its corresponding  $KE_{rid}$ . It then calculates a cookie using a local secret  $S$  as a key and performing an HMAC-MD5 over the fields of message 1 and 2. This thread dies without making any state after transmitting message 2. If message 2 is lost the initiator will resend message 1 and another such thread will be spawned which may choose a different key pair from the DH stack and calculate an new cookie.

The second thread responds to message 3 and completes the SA. It first checks if the  $KE_{rid}$  is valid and retrieves the corresponding DH public value for recalculating the cookie. If  $KE_{rid}$  is valid and the calculated cookie matches with the returned cookie, the responder calculates all the keying material and authenticates the message. If the message fails authentication the responder checks drops it and waits for another message 3. If it passes authentication the responder



**Figure 3. The DS SA responder child thread. This thread does not save any state before the initiator's DS is verified in message 3.**

decrypts the encrypted part of the message and verifies the initiator's signature. If the signature is not valid the SA is aborted without any notification. If the signature is valid the responder replies with message 4 containing its own signature and completes the SA.

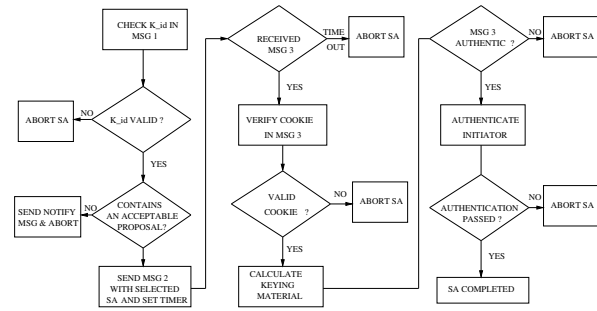
## 2.5 The PSK SA Initiator Child Thread

Fig. 2 shows the flow diagram of the PSK SA initiator child thread. The main initiator thread spawns this thread only if there is a PSK already established with the responder. This child thread checks the SPD for algorithm suits that are allowed with the responder. It then generates a random cookie, a nonce and calculates the key pointer  $K_{id}$  to the PSK [11]. It takes a DH public value from the DH stack [11] and sends message 1 to the responder with its SA proposals. At the same time it starts a retransmission timer and retransmits message 1 if a response does not arrive in the specified time or an invalid response is received.

Upon receiving a reply to message 1, this thread verifies its cookie and checks whether the responder has selected one of the proposed algorithm suits. Next, the responder is authenticated. If any one of these verification steps fail, the initiator retransmits the message 1. Otherwise, it calculates all the keying material and makes message 3 (encrypted and authenticated with the keying material). The thread transmits message 3 and starts a retransmission timer to ensure its delivery by retransmitting message 3 in case of no reply or if the response message fails authentication. On receipt of a valid reply (message 4), an SA is established.

## 2.6 The PSK SA Responder Child Thread

Fig. 4 shows the flow diagram of the PSK SA responder child thread. This thread is spawned by the main responder thread when a message 1 of a PSK SA arrives. The child thread checks the key pointer first and gets the corresponding PSK. It then selects an acceptable algorithm suit by consulting the SPD and replies with message 2. Possession of a valid  $K_{id}$  authenticates the initiator to some extent



**Figure 4. The PSK SA responder child thread saves state on receipt of a valid  $K_{id}$  in message 1.**

(full authentication is done in message 3). Therefore, this thread saves state at this stage. The responder also calculates the DH shared secret and keying material at this stage and authenticates itself to the initiator in message 2.

On receiving message 3, this child thread verifies its cookie first and then authenticates the message. If cookie verification or authentication of the message fails, the thread waits for another message 3. Otherwise, it decrypts the message using the selected algorithm and established key. Next, it authenticates the initiator. If the authentication fails, the SA is aborted. Otherwise, an SA is successfully established.

## 2.7 The PKE SA Initiator Child Thread

This child thread is similar to the one in Fig. 2 except that it transmits three messages. This thread sends SA proposals in message 1 after consulting the SPD and sets a retransmission timer to ensure the delivery of the message. Upon receiving message 2, this child thread verifies its cookie and ensures that one of the proposed SAs has been selected. A delayed or invalid message 2 triggers retransmission of message 1. A valid message 2 is responded to by a message 3. A retransmission timer is again set to ensure the delivery of message 3. After receiving message 4 all the keying material is calculated. Message 5 is transmitted which includes the initiator's authentication. Upon receiving message 6 the responder is authenticated and an SA is setup.

## 2.8 The PKE SA Responder Child Threads

There are two such threads, the first one responds to message 1 only by sending message 2 and dies without making any state. Selection of an SA proposal according to the SPD is the responsibility of this thread. The second thread is spawned on receipt of message 3. This thread verifies its cookie in message 3 and aborts if the cookie is not valid. If the cookie is valid, it decrypts the encrypted part of message 3 and replies with message 4 and calculates all the keying material. When message 5 arrives it verifies its cookie and

authenticates the message. If any of these checks fail, the responder child thread drops message 5 and waits for another message 5. If message 5 passes authentication, it is decrypted and the initiator is authenticated. If authentication of the initiator fails, it is considered as a man-in-the-middle attack and the SA is aborted. Otherwise, the SA is successfully established and message 6 is transmitted which includes the responder's authentication.

## 2.9 Background Threads

A stack of DH public and private values is maintained in the background by a dedicated low priority thread. This thread calculates DH public and private values to fill up the stack. Another thread is used in the background to clean any unnecessary state that is generated as a result of SAs that are aborted before completion, SAs that are no longer in use due to some reason e.g. the other peer is dead. In our implementation we used a static class that contained all the variables needed to be shared between the various threads. These variables could only be accessed in a synchronized manner. This ensured that there was no simultaneous access made to a memory location from different threads. It also ensured that no thread made an invalid access to a memory location.

## 2.10 Algorithm Suits

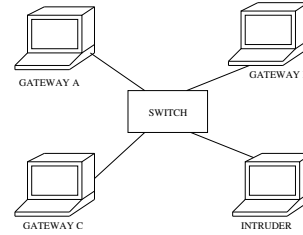
We used two algorithm suits in our experiments. Suit one comprised of Rijndael [14] encryption using 256 bit keys and HMAC-SHA1 [4] with 160 bit keyed message digest. Suit two used TwoFish [15] encryption with 192 bit key size and HMAC-RIPEMD160 [2] with 160 bit keyed message digest. Both suits used a DH group with 1024 bit prime and generator 2. Similarly RSA algorithm with 1024 bit keys was used for DS and public key encryption. New DH groups, encryption algorithms and DS algorithms could easily be added because of our object oriented design. The RSA key size was not limited to 1024 bits and could be increased without any modification in the software.

## 2.11 Logging of Events

During our experiments we logged all events. The log contained time stamped entries of all the events that took place while the protocol was running e.g. SA initiation, SA abortion, reason for SA abortion, error messages, invalid messages, masquerade attempts etc. The log file was used for analysis of the protocol. In real scenarios logging of events is useful for intrusion detection.

## 3. Simulation Results and Analysis

Our experimental setup comprised of a peer-to-peer network of three nodes simulating gateways and a fourth ma-



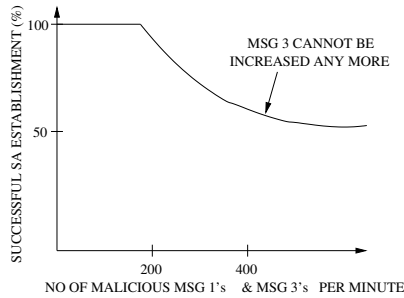
**Figure 5. Simulation setup comprised of three legitimate nodes and an intruder which has full control over the traffic.**

licious node simulating an intruder (Fig. 5). The nodes were connected through a switch controlled by the intruder. The intruder had full control of the switch and could sniff, delete, reflect, redirect and replay packets. In the passive eavesdropping attack, we applied a packet sniffer to extract information from SA negotiation messages. The attack was not successful due to the use of encryption in the messages. We did not perform cryptanalysis of the messages since checking the strength of encryption algorithm was outside the scope of our experiments.

The second attack simulated a highly unreliable network. Messages were deleted at random by the intruder during SA negotiations. We noted that setting the number of retransmissions to  $2/p$  (where  $p$  is the probability of message delivery) ensured the successful establishment of an SA. In the next attack, the intruder reflected the messages of the SA initiator in an attempt to establish its SA with its own self. The attack could only proceed up to message 2 since a reflected message 3 could not pass authentication. This is because a different set of keys are used in the two directions of the SA [11].

In the redirection attack, the intruder redirected messages from gateway A intended for gateway B to gateway C. This attack failed at message 4 when the responder could authenticate itself as gateway B. Similarly a man-in-the-middle attack by the intruder also failed at message 4 since the intruder could not authenticate itself as any of the legitimate identities. However, this attack revealed the identity of the initiator in DS mode.

The robustness of Arcanum to DoS was tested by launching two types of attacks. In the first type of DoS attack, the intruder flooded the network with a large number of message 1's. This attack could only be launched in the DS and PKE modes since the PSK mode requires a valid (and different)  $K_{id}$  for each message 1. This attack was not successful since a responder does not generate any state on receiving message 1 (see Fig. 3) and there is no heavy computation involved (DH public and private value generation) in making message 2. In the second type of DoS attack, in addition to sending a large number of message 1's, the intruder also replied to message 2's with message 3's. On receipt



**Figure 6. Percentage of legitimate SAs established in the presence of malicious messages.**

of a message 3, a responder must calculate the DH shared secret and authenticate the initiator. Both these operations involve heavy computations. Only after performing these computations does the responder come to know that the initiator is illegitimate and aborts the SA. This attack provided a partial DoS. The severity of the attack could not be increased since the rate at which message 3's could be transmitted was limited by the speed at which the responders reply with message 2's. A valid message 2 is necessary to get a valid cookie for generating message 3. We observed that the gap provided between message 1 and message 3 was sufficient for the legitimate SA negotiations resulting in successful SAs. Fig. 6 shows the percentage of legitimate SAs established against the number of illegitimate message 1 and message 3 pairs. SA establishment is not affected when there are less than 200 malicious messages present. The rate drops to 50% between 200 and 400 malicious messages and then stabilizes. This is because no more message 3's can be produced by the intruder due to limited message 2's from the initiators.

The final attack launched by the intruder was a replay attack. The intruder collected a large number of message 3's (of DS and PKE mode only) and retransmitted them almost simultaneously. We observed that none of the message 3's could pass the cookie verification step. This is because the responder's cookie is a function of two continuously changing values, the key identifier to the DH public value in the stack  $KE_{rid}$  and the local secret  $S$  of the responder [11].

## 4. Conclusion

In this paper we presented the implementation details of Arcanum key exchange protocol along with its experimental analysis. We also presented a setup for simulating different types of attacks on a key exchange protocol (Arcanum in our case). Our setup can be used to practically test any key exchange protocol. We also presented qualitative and quantitative results of our experiments. Our results show that Ar-

canum protocol is robust to a number of attacks including eavesdropping, reflection attack, replay attack, redirection attack, man-in-the-middle attack and DoS attacks.

## References

- [1] W. Aiello, S. Bellovin, M. Blaze, R. Canetti, J. Loannidis, A. Keromytis, and O. Reingold. Efficient, dos-resistant, secure key exchange for internet protocols. In *9th ACM Conference on Computer and Communication Security*, Nov 2002.
- [2] H. Dobbertin, A. Bosselaers, and B. Preneel. RIPEMD-160: A strengthened version of RIPEMD. In *Fast Software Encryption*, pages 71–82, 1996.
- [3] D. M. et al. Internet security association and key management protocol (isakmp). *IETF RFC 2408*, Nov, 1998, available at <http://www.ietf.org/rfc/rfc2408.txt>.
- [4] FIPS 180-1. Secure Hash Standard, April 1995. U.S. Department of Commerce/National Bureau of Standards, National Technical Information Service.
- [5] D. Harkins and D. Carrel. The internet key exchange protocol (ike). *IETF RFC 2409*, Nov, 1998, available at <http://www.ietf.org/rfc/rfc2409.txt>.
- [6] D. Harkins, C. Kaufman, S. Kent, T. Kivinen, and R. Perlman. Proposal for the ikev2 protocol. *Internet Draft (draft-ietf-ipsec-ikev2-02.txt)*, April, 2002 (expired Oct, 2002).
- [7] S. Kent and R. Atkinson. Security architecture for the internet protocol (ipsec). *IETF RFC 2401*, Nov, 1998, available at <http://www.ietf.org/rfc/rfc2401.txt>.
- [8] H. Krawczyk. The ike-sigma protocol. *Internet Draft (draft-krawczyk-ipsec-ike-sigma-00.txt)*, Nov, 2001 (expired May, 2002).
- [9] H. Krawczyk. Skeme: a versatile secure key exchange mechanism for internet. In *IEEE Symposium on Network and Distributed System Security*, pages 114–117, May 1996.
- [10] A. S. Mian. Source code of Arcanum Protocol, 2003. available at <http://www.cs.uwa.edu.au/~ajmal/key.html>.
- [11] A. S. Mian and A. Masood. Arcanum: A secure and efficient key exchange protocol for the internet. In *International Conference on Computers and Communications*, April 2004. to appear.
- [12] H. Orman. The oakley key determination protocol. *IETF RFC 2412*, Nov, 1998, available at <http://www.ietf.org/rfc/rfc2412.txt>.
- [13] P. Qualum, W. Simpson, and DayDreamer. Photuris: Session key management protocol. *IETF RFC 2522*, March, 1999, available at <http://www.ietf.org/rfc/rfc2522.txt>.
- [14] V. Rijmen and J. Daemen. The Block Cipher Rijndael. *Lecture Notes in Computer Science*, pages 288–296, 2000.
- [15] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, and C. Hall. On the twofish key schedule. In *Selected Areas in Cryptography*, pages 27–42, 1998.