DHCP++: Applying an efficient implementation method for fail-stop cryptographic protocols

William A. Arbaugh Angelos D. Keromytis Jonathan M. Smith*
University of Pennsylvania

August 20, 1998

Abstract

The DHCP protocol is used by hosts to dynamically allocate an IP address and configure client hosts. The protocol greatly eases the administration of an IP subnetwork and is thus widely used.

The basic approach of the DHCP protocol is for a client to broadcast a request for an address, and for one or more servers to respond with addresses. This creates significant opportunities for security risks due to active attackers.

We have designed a new, efficient implementation method for the fail-stop cryptographic protocols originated by Gong and Syverson. The implementation method uses cryptographic hashes of the state of the sender and receiver and the exchanged messages to detect if any deviation from expected behavior has taken place. If it has, an attack is assumed and the protocol ceases execution. We present a proof outline of protocol security using our method.

We have applied our method to DHCP, resulting in DHCP++. DHCP++ uses our fail-stop implementation technique to prevent any attacks that could otherwise violate DHCP's security assumptions. The resulting protocol operates entirely within the context of DHCP. The paper analyzes the threats eliminated by this enhancement, and measurements against DHCP show that the incremental performance costs are minimal.

1 Fail-stop protocols

Cryptographic protocols are used in many advanced applications such as electronic banking, networked software distribution, and wireless personal communications systems. Due to the complexity of conditions they may encounter, careful reasoning and formal means such as proofs are used to validate the design of a cryptographic protocol. Such validation is easier if the set of threat conditions is reduced. Techniques resulting in the construction of protocols which *by design* reduce the complexity of threat conditions are thus extremely attractive.

One such technique is a *fail-stop* cryptographic protocol, introduced by Gong and Syverson [FS]:

> A protocol is *fail-stop* if any attack interfering with a message sent in one step will cause all causally-after messages in the next step or later not to be sent.

As Gong and Syverson show, fail-stop protocols possess a very useful security property, namely:

> active attacks cannot cause the release of secrets within the run of a fail-stop protocol

The fail-stop property lets a protocol designer restrict security concerns to passive (eavesdropping) attacks, a significant reduction in the class of threats to the protocol's security. The cost is that the protocol must terminate when active attacks occur, rather than attempt to continue. We believe that reliable termination is greatly preferred to unknown and insecure behavior in the face of active attacks on security, and when embedded in a larger system, protocol termination can be handled by higher-level detection and resolution mechanisms.

1.1 Specifying fail-stop behavior

Syverson and Gong state the following specifications for a fail-stop protocol:

 Authors' email addresses are {waa,angelos,jms}@dsl.cis.upenn.edu. This research was supported by DARPA under contract #N66001-96-C-852.

1. The content of each message has a header containing the identity of its sender, the identity of its intended recipient, the protocol identifier and its version number, a message sequence number, and a freshness identifier.

2. Each message is encrypted under the key shared between its sender and intended recipient.

3. An honest process follows the protocol and ignores all unexpected messages.

4. A process halts any protocol run in which an expected message does not arrive within a specified timeout period.

The above specifications assume that the two communicating parties share a secret encryption key used with a symmetric key cryptosystem (such as DES [FIPS46]). It should be noted that this is not the only method for generating fail-stop protocols, but rather the simplest one.

The freshness identifier can be a nonce issued by the intended recipient or a time stamp (if the clocks are assumed to be securely and reliably synchronized [LG92]).

1.2 Outline of this paper

Section 2 presents our method for chaining the messages of a protocol run. This makes the messages sequenced and non-reusable outside the context of this protocol run, thereby making message tampering and replay attacks infeasible [PS]. Section 3 briefly discusses the DHCP protocol. Section 4 presents our extensions to DHCP. Section 5 discusses our implementation of DHCP++ and, section 6 concludes the paper and summarizes its contributions.

2 Message chaining

Fail-stop cryptographic protocols, as specified by Gong and Syverson, require repeating considerable information in each message of the protocol. This has undesirable consequences. First, the size of the messages increases. Second, the amount of encryption/decryption required also increases [PET]. Third, and possible of most concern, there is more plaintext for an attacker to mount known plaintext attacks [LG90] [PPC].

However, rather than include all the information of their specification in each protocol message, we can rely on one-way hash functions, such as MD5 [MD5] or SHA [SHA] to preserve the sequencing of the protocol messages.

The initiator of a modified protocol run starts by sending the first message including all the aforementioned information (some of which might already be included in the original version of the protocol). Additionally, a hash of the entire message (including the new fields) is sent. This value is also kept as local state.

The responder verifies these values and then proceeds with the normal flow of the protocol. From there on, each message exchanged contains the one-way hash result of the new message and the state of the sender of that particular message. The receiver of each message calculates the hash of the message and the local state and compares it with the hash value received. If they're not the same, then some active attack is assumed to be in progress.

Naturally, the hash values cannot be sent along with the message unprotected; an attacker could tamper with them. There are three alternatives to securing them:

1. Sign the hash values with one's private key, if the peer is known to have one's public key (or can securely obtain or verify it, by using some certificate directory or some certification scheme [X509] [SPKI]), when using a public key cryptosystem such as RSA [RSA]).

2. Encrypt the hash value with a shared secret key using a symmetric cryptosystem.

3. Use keyed hashes (where the key has to be used in each hash function application), again using a shared secret key. Care must be taken to use a strong way of keyed-hashing [BCK]. The advantages of this method can be faster computation (since typically hash functions outperform encryption functions) and avoidance of restrictions in export or use of cryptography.

2.1 The hash function

Here we assume that the hash function used has four desirable properties:

- *Collision resistance*: an attacker cannot create another message with the same hash value.

- *Entropy preservation*: the effect that some value has in the hash result does not disappear, at least not before a reasonably large number of applications of the hash function.

- *Non commutativity*: reordering the messages will make the results invalid.

- Given subset of the input to the hash function and the result, an attacker can not find what the missing input bits are.

Depending on the particular protocol, additional properties might be required from the hash function [RJA].

2.2 Correctness

Keeping in mind that the verification at each step of a protocol is of the form:

- H(*Key, State, Message*) $\stackrel{?}{=}$ Message.*Hash*

There are three methods of active attacks a malicious entity can attempt:

1. Find the key (by cryptanalysis or exhaustive search). We have already made the assumption that this is not possible.

2. Inject a new message such that the verification succeeds. This means that the attacker is capable of creating collisions on the hash function even when a secret quantity (the *Key*) is involved. Again, this is a relatively weak assumption we have already made.

3. Affect the *State* kept by either of the legitimate protocol parties, so that a captured message can be replayed. Manipulating the *State* to some desired value - even if it were possible - would not allow the attacker to introduce a new message of his own, since the *Key* used is unknown. Also, the chance of the *State* being the same as the one at a previous step of the protocol is negligible (approximately $(1/2)^{n/2}$, where **n** is the length in bits of the output of the hash function).

   However, manipulating the *State* to some particular value means that the previous round of the protocol has been tampered with successfully [1]. This in turn (due to the reasons given above) means that the round before that has been successfully attacked.

   Following this argument, the attacker would have to affect the first message in the protocol. More specifically, the message would have to be replaced by another message from a previous or parallel run of the protocol which used the same *Key*. This is not possible:

   - The first message cannot be tampered with, since it involves usage of a secret key (in a symmetric or asymmetric algorithm), and the hash function is collision free.

   - The first message contains enough information to distinguish it from any other first-message of the same protocol.

While hardly a proof of correctness, we outline the direction such a proof will take.

[1] We assume the attacker does not have access to the internal storage of either of the legitimate protocol parties.

2.3 Other attacks

There is one final point of concern: although it is impossible for an attacker to inject a message in the middle of a protocol run under our scheme, it is still possible to use a captured message as the first message in a protocol. Of course, the protocol must allow this attack by message insertion in the first place, and such an attack is precluded if it is impossible for an attacker to remove the hash value from the message (an unencrypted keyed hash field, for example, can be discarded, but not if it's encrypted in CBC mode [CBC] with all other message fields).

If the protocol designer is also worried about insider attacks, the protocol should be designed such that any cryptographically valuable message components (such as a digital signature) are inseparable from the hash value. For example, in the case of a digital signature, one would include the hash value in the signature computation.

A similar method is used in SSLv3 [SSL], but neither any connection to failstop properties nor a proof of security were given.

3 Dynamic Host Configuration Protocol

The DHCP protocol [DHCP] provides clients the ability to configure their networking and host specific parameters dynamically during the boot process. The typical parameters are the IP addresses of the client, gateways, and DNS server. DHCP, however, supports up to 255 configuration parameters, or options. Currently, approximately one hundred options are defined for DHCP [DHCPOPT]. One of these options is an authentication option which is described in Section 5.1.

The initial message exchange between the client and the server is shown in Figure 1.

The client begins the process by sending a DHCPDISCOVER message as a broadcast message on its local area network. The broadcast message may or may not be forwarded beyond the LAN depending on the existence of relay agents at the gateways. Any or all DHCP servers respond with a DHCPOFFER message. The client selects one of the DHCPOFFER messages and responds to that server with a DHCPREQUEST message, and the server acknowledges it with a DHCPACK.

In addition to providing networking and host specific parameters, DHCP can provide the name and server location of a bootstrap program to support diskless clients. After the client receives the IP address of the boot server and the name of the bootstrap program, the client uses TFTP [TFTP] to contact the server and transfer the file.

It should be obvious that any malicious host on the same local network as the requester can claim to be a DHCP server and provide false information to the client. Additionally, since the transfer of the bootstrap is not protected, an attacker can cause the booting machine to load a tampered kernel, completely compromising its security. Finally, since there is no access control, anyone can connect a laptop to such a network an acquire an valid IP address. This problem is particularly prominent in university campus networks, where physical security to the network infrastructure is hard to enforce.

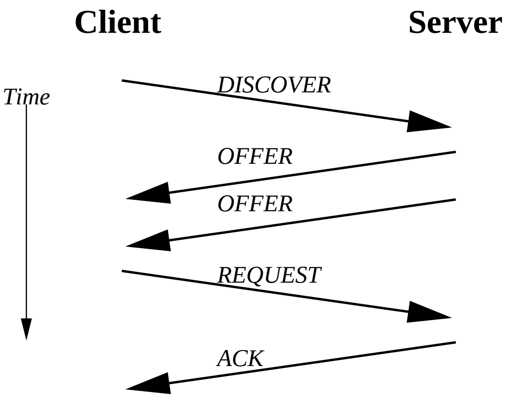

Figure 1: DHCP Message Exchange

4 DHCP++

We first describe our certificate format, which is a subset of SPKI [SPKI]. We then describe our extensions to DHCP.

4.1 Certificates

Certificates are typically used to provide a binding of a public key to an identity [X509] or to grant authorization [SPKI]. In our implementation we used a subset of SPKI. The reasons for this decision were:

- The certificates fit in one DHCP option payload. This preserved backward compatibility.[2]
- SDSI/SPKI provides for the notion of a capability [CAP], such that the certificate carries the authorizations of the holder, eliminating the need for an identity infrastructure and Access Control Lists. Maintaining ACLs in a distributed environment is a complex and difficult task, especially at the low level at which DHCP operates. In our extension of DHCP, we use the capabilities SERVER and CLIENT, with the obvious meanings.
- The code necessary for handling SDSI/SPKI certificates is small enough to (potentially) fit in a smartcard.

```
((cert (issuer (hash-of-key (hash sha1
                clientkey)))
    (subject (hash-of-key (hash sha1
                  clientkey)))
     (tag (client (cnonce cbytes)
            (msg-hash
             (hash sha1 hbytes))))
     (not-before 09/01/97-0000)
     (not-after 09/01/97-0000))
 (signature (hash sha1 hashbytes)
  (public-key dsa-sha1 clientkey)
  (sigbytes)))
```

Figure 2: Client Authentication Certificate

In DHCP++ we use only two types of certificates. The first is an authorization certificate. This certificate, signed by a trusted third party or certificate authority, grants to the key holder (the machine that holds the private key) the capability to generate the second type of certificate - an authentication certificate.

The authentication certificate, examples of which are given in Figures 2 and 3, demonstrates that the client or server actually holds the private key corresponding to the public key identified in the authorization certificate. A nonce field is used along with a corresponding nonce in the server authentication certificate to provide freshness. The *msg-hash* field contains the hash value described in section 2. Also, using the *msg-hash* in the authentication certificate eliminates a signature and verification operation since the entire message no longer needs to be signed. The additional server fields are used to pass optional Diffie-Hellman [DH] parameters to the client so that these parameters need not be global values. While clients are free to set the validity period of the authentication certificate to whatever they desire, we expect that clients will keep the period short. Short certificate expirations are also the chosen method of certificate revocation (as opposed to Certificate Revocation Lists).

4.2 Security Extensions to DHCP

A DHCP client and server wish to communicate and establish a shared secret after mutual authentication. There has been no prior contact between the client and the server

[2] Since the certificate format is still under development, we actually used a subset of all the features, and a mapping between the Advanced Transport Format (example in Figure 2) and a pure binary format (which we call BTF, for Binary Transport Format), to reduce the certificate size. We call this subset *SDSI/SPKI Lite*.

```
((cert (issuer (hash-of-key (hash sha1
                serverkey)))
    (subject (hash-of-key (hash sha1
                  serverkey)))
     (tag (server (dh-g gbytes)
             (dh-p pbytes)
             (dh-Y ybytes)
             (msg-hash
              (hash sha1 hbytes))
             (cnonce cbytes)
             (snonce sbytes)))
     (not-before 09/01/97-0900)
     (not-after 09/01/97-0900))
 (signature
   (hash sha1 hashbytes)
   (public-key dsa-sha1 serverkey)
   (sigbytes)))
```

Figure 3: Server Authentication Certificate

other than to agree on a trusted third party, or a public key infrastructure, to sign their authorization certificates, C_{AR}. The server and the client also need to have a copy of the trusted third party's public key, P_{CA}.

The protocol presented is done as part of the usual DHCP protocol exchange, augmented to include the authentication option in all the messages.

- In the first message (DHCP DISCOVER), the client sends to the server the client's authorization and authentication certificates, C_{AR} and C_{AN}.
- The server receives the message and verifies the client's signature on the authentication certificate and that the hash contained in the authentication certificate matches that of the message M, as per section 2 . The signature of the CA on the authorization certificate (or chain of certificates) is also verified at this stage. If all are valid and the timestamp on the authentication certificate is within bounds, then the server sends to the client a message (DHCP OFFER) containing its authorization and authentication certificates. The server's authentication certificate may include the optional DH parameters, g and p, and Y, where $Y = g^y \mod p$. If the DH parameters are not identified in the certificate, then default values for g and p are used.[3] The server's nonce, *snonce*, is also included in the message.
- The client receives this message and verifies the signatures on the authentication and authorization certificates, and that the hash in the server's authentication certificate matches the message hash combined with the state hash. If all are valid and the timestamp value of the authentication certificate is within bounds then the client sends a signed message (DHCP REQUEST) to the server containing its DH parameter X where $X = g^x \mod p$, and the combination of its state hash with the hash of the message. The server receives the message and verifies the signature and the hash received. If both are valid, then the server can generate the shared secret, k, using DH, $k = X^y \mod p$. The client similarly generates the shared secret, $k = Y^x \mod p$.

[3] Currently, we are using the same default values as those used in SKIP [SKIP].

The shared secret, k, can now be used to authenticate messages between the server and the client until such time as both agree to change k. Figure 4 depicts the entire exchange between the client and the server with the DHCP messages identified.

The use of *snonce* also permits the server to reuse Y over a limited period. This reduces the computational overhead on the server during high activity periods. The potential for a TCPSYN like denial of service attack [TCPSYN] is mitigated in the same manner by the authentication certificate. The authorization certificate also prevents clients from masquerading as a server because of the client/server capability tag. This is a benefit not possible with basic X.509 certificates.

The protocol, as described above, is fail-safe. This means that an attacker can replay the first message and cause the server to reply. The attacker cannot generate the third message however, since he cannot generate a valid signature and cannot replay the third message from the previous protocol run without being detected. If the server keeps the nonce sent by the client, *cnonce*, for as long as the client's authentication certificate is valid (which should be quite short, in the order of a few minutes at most), then the protocol becomes fail-stop.

4.3 Subsequent Message Authentication

After the establishment of the shared secret through the protocol described above, subsequent DHCP messages are authenticated through the use of a SHA1 HMAC [HMAC] similar to the one used in the IPSEC Authentication Header [AH], augmented with a one up counter to prevent replays. The counter is initially set to zero when the shared secret, k, is derived. Subsequent TFTP messages make use of the IPSEC Authentication Header.

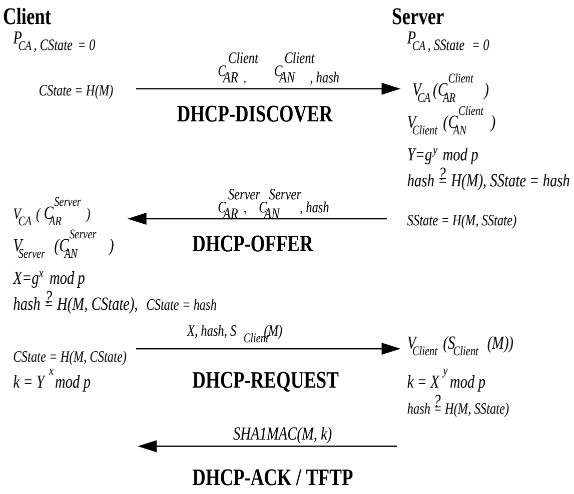

Figure 4: Authentication Message Exchange

5 Implementation

Moving from a high level design to an implementation requires a great deal of work. In this section we take the protocol and certificates described in the previous section and describe their implementation. We also provide the message formats and type information. We conclude the section by providing measured performance of the system.

5.1 DHCP Authentication Option

DHCP is extensible through the use of the variable length options field at the end of each DHCP message. The format and use of this field is currently defined by an Internet RFC [DHCPOPT]. An option for authentication is also defined by a draft RFC [DHCPAUTH].

The DHCP authentication option was designed to support a wide variety of authentication schemes by using single byte protocol and length fields. Unfortunately, a single byte value for the size in octets of authentication information severely limits the ability to include public key certificates, with reasonable key sizes, in the data field of the option. Fortunately however, we do so by using the BTF format briefly described earlier, and using multiple authentication option fields. While this latter approach *technically* violates the DHCP authentication option protocol, it does not cause any interoperability problems. An alternate approach would have required increasing the the option size field from one to two bytes. While interoperability issues could be mitigated, the approach still presented a significant change to the DHCP protocol.

Since we must use multiple authentication option fields in a DHCP message, we must add a field to identify the information contained in the option. The resulting authentication option format is shown in Figure 5. The client and server use this option format to exchange the information required by our protocol.

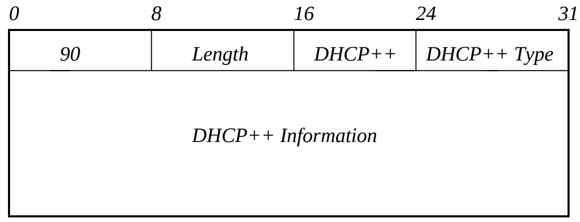

Figure 5: Authentication Option Format

In addition to using BTF formatted SPKI certificates, we support the use of a new DHCP option to permit the continuation of the previous option field. Through the use of this option, any information that exceeds the 256 bytes available in a DHCP option can be extended into the next field. This permits the use of X.509v3 (or other) certificates if desired.

5.2 Performance

We did some performance measurements using both standard DHCP (with no security) and DHCP++. We used Pentium-2 PCs with 64MB of RAM, running Linux. For the cryptographic operations we used CryptoLib 2.0beta. The 1024 bit DSA [DSA] signatures were computed in 20ms and verified in 30ms. The DH public exponent generation took 17ms, and the shared secret key was computed in 52ms. The end result is that DHCP++ takes less than a second more than (insecure) DHCP.

6 Conclusion

We have described the design and implementation of a secure version of the DHCP protocol called DHCP++, which has wide practical applications. We developed a novel and efficient implementation technique for the fail-stop cryptographic protocols originated by Gong and Syverson, which used message-chaining to increase performance by reducing the size of messages relative to Gong and Syverson's specification without increasing the number of cryptographic operations.

The basis of the technique is representing the state of a cryptographic protocol as a secure hash value. We first augmented DHCP with our security extensions, and then hardened the protocol by application of our fail-stop technique. As discussed in Section 5, the performance impact is minimal, while transforming all active attacks on the enhanced DHCP protocol into protocol terminations. The result, DHCP++, allows system designers to focus their attention on passive (eavesdropping) attacks and attacks by malicious insiders.

An alternative approach to securing DHCP would involve using the standard IPSec [IPSEC] mechanisms, using some temporary address while running the key management protocol [ISAKMP]. The main drawbacks to such a solution from our point of view is the complexity of ISAKMP (for which there is no proof of correctness), the code size (our solution has been fully implemented in the PC BIOS) and the synchronization problems involved with using a temporary IP address in the presence of multiple machines booting at the same time (as may be the case after a power failure).

7 Acknowledgments

The authors would like to thank Dave Farber, Li Gong and Paul Syverson whose ideas and previous work influenced this paper.

References

[FS] "Fail-Stop Protocols: An Approach to Designing Secure Protocols", *Gong, Li and Syverson, Paul, Proceedings of IFIP DCCA-5, September 1995*

[PET] "Prudent Engineering Practice for Cryptographic Protocols", *Abadi, Martin and Needham, Roger, IEEE Computer Society Symposium on Research in Security and Privacy, 1994*

[FIPS46] "NBS FIPS PUB 46 - Data Encryption Standard", *National Bureau of Standards, U.S. Department of Commerce, January 1977*

[MD5] "The MD5 Message Digest Algorithm", *R.L. Rivest, RFC 1321, April 1992*

[SHA] "NIST FIPS PUB 180 - Secure Hash Standard", *National Institute of Standards and Technology, U.S. Department of Commerce, May 1993*

[BCK] "Keying Hash Functions for Message Authentication", *Mihir Bellare, Ran Canetti and Hugo Krawczyk, Advances of Cryptology, Crypto '96 Proceedings*

[RJA] "The Classification of Hash Functions", *Ross Anderson, Proceedings of IMA Conference on Cryptography and Coding, 1993*

[LG90] "Verifiable-Text Attacks in Cryptographic Protocols", *Li Gong, Proceedings of the IEEE INFOCOM '90, June 1990*

[LG92] "A Security Risk of Depending on Synchronized Clocks", *Li Gong, ACM Operating Systems Review, v26n1, January 1992*

[PPC] "Protecting Poorly Chosen Secrets from Guessing Attacks", *Li Gong, T. Mark A. Lomas, Roger M. Needham and Jerome H. Saltzer, IEEE Journal on Selected Areas in Communications, v11n5, June 1993*

[RSA] "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", *R.L. Rivest, A. Shamir and L.M. Adleman, Communications of the ACM, v21n2, February 1978*

[PS] "A Taxonomy of Replay Attacks", *Paul Syverson, Proceedings of the Computer Security Foundations Workshop VII (CSFW7), June 1994*

[IPSEC] "Security Architecture for the Internet", *R. Atkinson,*

[CBC] "Applied Cryptography: Protocols, Algorithms and Source Code in C, 2nd edition", *Bruce Schneier, John Wiley & Sons Inc., NY 1996*

[SSL] "The SSL Protocol", *K.E.B. Hickman, RFC, Netscape Communications Corp., October 1994*

[DHCP] "Dynamic Host Configuration Protocol", *Droms, R., RFC 2131, March 1997*

[DHCPOPT] "DHCP Options and BOOTP Vendor Extensions", *Alexander, S. and Droms, R., RFC 2132, March 1997*

[ASSIGNED] "Assigned Numbers", *Reynolds, J. and Postel, J., RFC 1700, October 1994*

[TFTP] "The TFTP Protocol (Revision 2)", *Sollins, K. R., RFC 1350, July 1992*

[X509] "X.509: The Directory Authentication Framework", *Consultation Committee, ITT, ITU, Geneva 1989*

[SPKI] "Simple Public Key Infrastructure", *Carl M. Ellison, Bill Frantz, Ron Rivest and Brian M. Thomas, Work in Progress, April 1997*

[CAP] "Capability Based Computer Systems", *Levy, H. M., Digital Press, 1984*

[DH] "New Directions in Cryptography", *Diffie, W. and Hellman, M. E., IEEE Transactions on Information Theory, Volume 22 p. 644–654, November 1976*

[DSA] "Digital Signature Standard", *National Institute of Standards, FIPS-186, May 1994*

[AH] "IP Security Authentication Header", *Stephen Kent and Randall Atkinson, Work in Progress, October 1997*

[HMAC] "HMAC: Keyed–Hashing for Message Authentication", *Krawczyk H., Bellare M. and Canetti R., RFC 2104, February 1997*

[SKIP] "Assigned Numbers for SKIP Protocols", *Ashar Aziz, Tom Markson and Hemma Prafullchandra, Work in Progress*

[TCPSYN] "Attack Class: Address Spoofing", *Heberlein, L. T. and Bishop, M. Proceedings of the 19th National Information Systems Security Conference, p. 371–377, October 1996*

[DHCPAUTH] "Authentication for DHCP Messages", *Droms, R., Work in Progress, August 1997*

[ISAKMP] "Internet Security Association and Key Management Protocol (ISAKMP)", *D. Maughan, M. Schertler, M. Schneider and J. Turner, Work in Progress, January 1998*