Analyzing the TCP Authentication Option (TCP-AO)

Rahel A. Fainchtein, Shrey Arora

December 18, 2018

Abstract

We evaluated the TCP Authentication Option (TCP-AO)'s security properties using the Cryptographic Protocol Shapes Analyzer (CPSA) and attempted an implementation of the protocol using GMAC. As of the current draft we reached the cursory conclusion that the protocol is secure as specified, but faced numerous challenges in process, many of which prevented more thorough analysis. In future work we hope to compare CPSA's evaluation to analysis of TCP-AO with other tools to get a better understanding of the protocol's security properties and to shine light on today's cryptographic provers' ability to model implementation standards.

Index terms— Protocol Analysis, CPSA, GMAC, Key Derrivation Function (KDF), implementation

leofcontents

1 Introduction

Arguably its most well-known vulnerability, TCP's lack of authentication, leaves it vulnerable to attack. In fact, this far-reaching vulnerability is commonly exploited to deploy reset attacks, in which the attacker sends a reset segment to the connection initiator A, causing A to not only close the connection, but to delete her stored connection state. Thus, the TCP connection establishment (3-way handshake) or maintenance between A and her intended recipient is forced to fail. Given TCP's widespread use for reliable in-order data transmission over IP, it also means other services including higher layer protocols such as the Border Gateway Protocol (BGP) remain vulnerable to DoS due to TCP reset attacks. Original solution given was to use MD5 HMACs, as specified in RFC 2385[1] and updated in RFC 6691[3]. Due largely to MD5's known vulnerabilities[6], this solution was updated with the more generic TCP-AO. However, in spite concerns regarding TCP-MD5's security, no full scale implementations of TCP-AO have been deployed. In fact, as of this writing, TCP-AO has been implemented using the SHA1 and SHA256-128 HMACs, but has only been implemented as a proof of concept. To date the protocol has only been implemented by Cisco systems[8] and still has yet to undergo cross platform testing. Further the protocol has yet to be systematically analyzed via cryptographic prover. In this paper we attempt to analyze TCP-AO using the Cryptographic Protocol Shapes Analyzer (CPSA) and to implement the protocol using GMAC. We then share our insights into our status in this process and our intended goals in future work.

2 TCP-AO Construction

As alluded to in its name TCP-AO is an extension of TCP. Therefore, all AO relevant fields are contained in this option field. These include the kind number (29), length, keyID, RNextKeyID and the message authentication code digest (MAC)[6]. Combining these with the components specified in RFC793, yields augmented segments as shown in figures 1 and 2, which contain the following fields.

- The IP PseudoHeader[9]:
  - Source address (32 bits IPv4, 128 bits IPv6)
  - Destination Address (32 bits IPv4, 128 bits IPv6)
  - Zero field (8 bits IPv4,16 bits IPv6)
  - Proto (8 bits IPv4 only)
  - TCP Length (16 bits IPv4 only)
  - Next Header (16 bits IPv6 only)
- The main TCP segment header[9]:
  - Source Port (16 bits)
  - Destination Port (16 bits)
  - Sequence number (32 bits)
  - Acknowledgement number (32 bits)
  - Data offset (4 bits)
  - Reserved field (6 bits-generally set to 0)
  - Flags (6 bits total)[9]
    - Urgent (URG)
    - Acknowledgement (ACK)
    - Push (PSH)
    - Reset (RST)
    - Syn (SYN)
    - Fin (FIN)
  - Window (16 bits)
  - Checksum (16 bits)
  - Urgent Pointer (16 bits)
  - Options
    - TCP-AO[6]:
      - Kind =29 (8 bits)
      - Length (8 bits)
      - KeyID (8 bits)

· RNextKeyID (8 bits)
· MAC Digest (96-128 bits)

As shown in the figures, segment headers can contain other TCP options both before and after AO. ⟨CLINCHER/TRANSITION SENTENCE NEEDED⟩

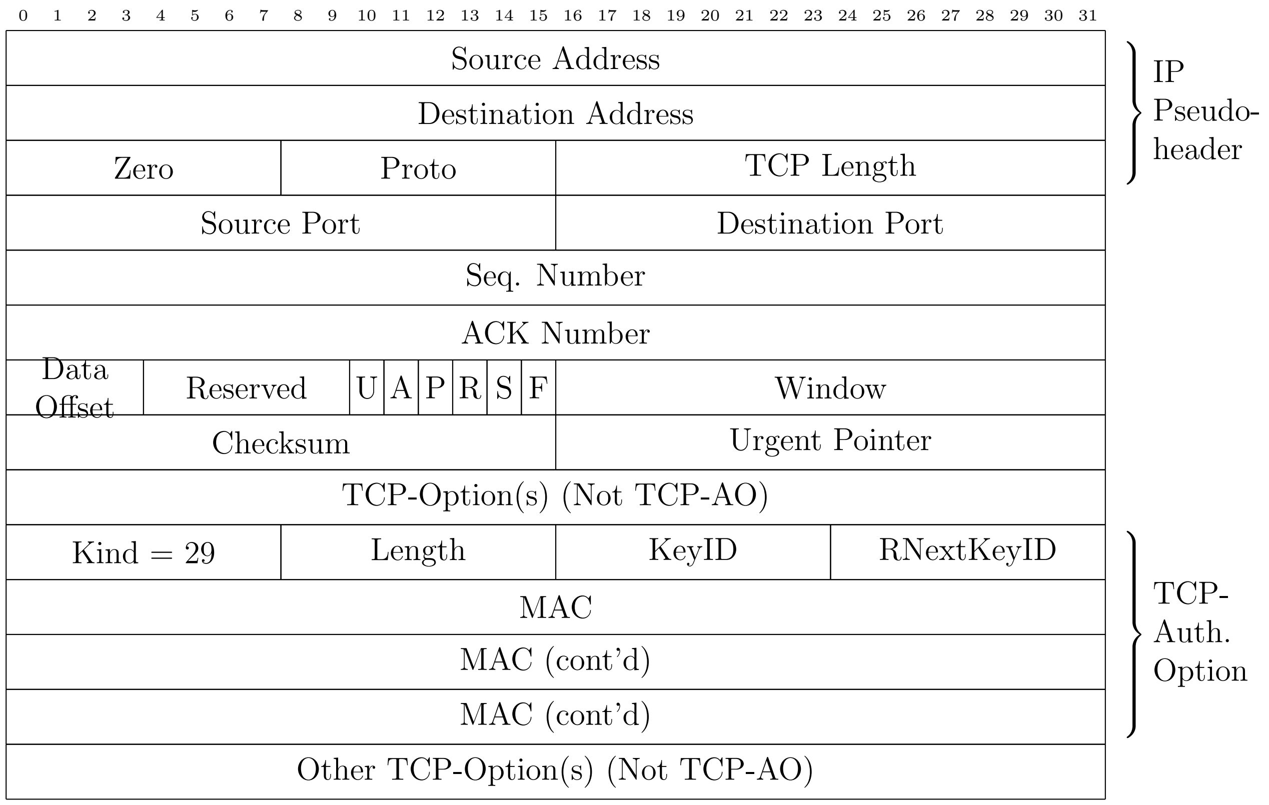

Figure 1: Header of a segment using TCP-AO over IPv4

3 Analyzing TCP-AO's Security Properties

To analyze the TCP-AO, the protocol's prescribed exchanges were modeled in CPSA and the tool's outputs were analyzed to discern which, if any, attacks were possible.

3.1 Background: The Cryptographic Protocol Shapes Analyzer (CPSA)

The Cryptographic Protocol Shapes Analyzer (CPSA) was originally developed in 2007 as a "pure Dolev-Yao model."[11] Since then it has been expanded to support a more generalized and descriptive version of the model (see **??** for a description of the Dolev-Yao model). Specifically, it was expanded to model protocols using Diffie-Hellman exchanges, Symmetric Encryption and Authentication via message authentication codes (MACs) and Digital signatures, and to boast advanced functionality via the addition of additional modules. As such the goal of CPSA's adversary Z was also generalized from that of the Dolev-Yao model. Instead of simply decrypting a message M sent between two honest participants X and Y over a network shared by all legitimate users (of which Z is a member), Z now seeks to successfully break the protocol T's cryptographic security goals. In the cse of TCP-AO, that is to somehow do one of the following:

1. Forge a message that is accepted by one of the honest parties as a legitimate message within the protocol
2. Replay a message that has already been accepted such that it will once again be accepted by a recipient X in place of the legitimate message X expected to receive (usually from the second honest participant Y) at point i in the protocol, where $0 \leq i \leq t$

However, when using CPSA's main module, the following conditions remain unchanged:

1. Each party has a finite set or "alphabet" of operations they can perform. These must be declared before the protocol begins iteration as the assumption of statelessness regarding messages sent over the network still hold that is
   (a) Messages sent by an honest party must be immediate computations on the last message received. They cannot use information or state based on information earlier messages
   (b) The adversary Z can see and use anything that has been sent over the network, and can also initiate iterations of the protocol with honest parties. Z can then use information received from the alternate protocol iterations to achieve their goal of:

- Decrypting a message M, in the case of an encryption scheme.
- Successfully passing off a replay or forgery as a legitimate message, in the case of a signature or MAC scheme.

CPSA does this by depicting each two-party exchange described two unilateral points of view. For each transmission within a point of view, the prover determines whether Z could somehow construct the expected message using values that have already been transmitted in this iteration of the protocol, or that were transmitted in an alternate adversary-induced iteration. Hence, the critical term, or the term tested by CPSA at each step of evaluation, is the component checked to determine whether Z could somehow forge a message.[11]

3.2 Analysis Methodology: Our Assumptions

To truly test the protocol's robustness, we wanted to first test the security of proper implementations of TCP-AO. Therefore, in addition to CPSA's baseline assumptions[7], that is

- We are faced with a strong adversary who controls the network and who is effectively synonymous with it.
- Assuming their keys are not leaked, all cryptographic functions performed as part of the protocol are secure against the adversarial network

We assumed the following when building the TCP-AO models:

1. All honest parties implementing TCP-AO compute their initial sequence numbers (ISNs) psuedorandomly with a cryptographically strong pseudorandom generator (PRG).
2. All TCP-AO master keys are composed of k bits selected uniformally at random, where $28 \leq k \leq 128$. [6][5]
3. Neither master keys, nor the traffic keys derrived from them are ever leaked to the adversary by an honest party.

Using these assumptions and the segment configurations as defined by the IETF[6][9][3], and depicted in figures 1 and 2, we modeled the following exchange variants possible within TCP-AO:

- 3-Way Handshake
- A simple exchange including a 3-way handshake, two data packets and a FIN/FIN-ACK
- A full exchange with the one party rolling over to a new master key.
- A legitimate connection reset after one side crashed and/or rebooted(?)

All exchanges were initially modeled and evaluated without considering the TCP options entries not associated with the AO.

Security Goal - Adversary cannot forge a message. That is the adversary cannot:

1. Create and MAC a message that has not been seen in the current conversation, but whose digest, sequence number and ACK number are correct and/or found to be acceptable.
2. Replay a legitimate message that will be accepted by an honest party.

As denoted in the CPSA usage manual, each step of an exchange is denoted from a given party's perspective. Therefore, all segments modeled in the exchange were shown as an honest party sending or receiving a message, and the message was composed of all its included fields. Said fields were to be included as variables in the exchange. Using these requirements and the above description of the TCP-AO segment header, we used the following mapping of tcp header fields to variables in CPSA:

- Source (IP) address$\rightarrow$ addr_$\langle$sender$\rangle$
  where $\langle$sender$\rangle$was either a - the initiator (client), or the responder b (server)
- Destination (IP) address$\rightarrow$ addr_$\langle$receiver$\rangle$
  Where $\langle$sender$\rangle\neq$ $\langle$receiver$\rangle$
- TCP Length$\rightarrow$ `len⟨seg_num⟩`
  Where $\langle$seg_num$\rangle$is counted across all segments in the exchange regardless of source
- Source Port $\rightarrow$ `src_⟨sender⟩`
- Destination Port$\rightarrow$ `dst_⟨receiver⟩`

- Flags (URG,ACK,PSH,RST,SYN,FIN)→ `flgs_⟨sender⟩⟨sender_seg_num⟩`
  Where `⟨sender_seg_num⟩` is the segment index counted across segments from `⟨sender⟩` only. This was a more challenging field to denote and required some additional abstraction. Additionally, we partially denoted the flags' messages as strings at the beginning of each message. These choices will be discussed in greater detail in the analysis.

- Sequence number → `seq_⟨sender⟩⟨sender_seg_num⟩`

- Acknowledgement number→ `seq_⟨receiver⟩⟨receiver_next_seg_num⟩`
  Where `⟨receiver_next_seg_num⟩` is the index of the next segment expected from the receiver (i.e. the `⟨sender_seg_num⟩` of the next segment from the receiver).

- Window→ `wdw_⟨sender⟩`
  Given CPSA's assertion that all honest parties effectively be stateless, we denoted a and b's receive window size as constants. While inconsistent with the TCP specifications, this was important as an initial means of determining whether the window field - denoting the receive window size could be altered in transit by a strong adversary.

- Checksum→ `chksum_⟨message_identifier⟩`
  Where message identifier was some text denoting to which message the checksum corresponded.

- KeyID→ `k_id_⟨sender⟩`

- **RNextKeyID→ `k_id_⟨receiver⟩2` Since this value is only to be specified when the sender is ready to receive segments using a new MKT, this field was only included when its value was non-trivial.

- MAC→ `(Enc (hash ⟨header_fields⟩) ⟨Traffic_Key⟩)` Where `⟨header_fields⟩` denotes all input fields over which the MAC was computed and `⟨Traffic_Key⟩` is a depiction of the KDF function as described in RFC 5925.
  This depiction took the form:

  - `(hash ⟨shared_key⟩addr_⟨sender⟩addr_⟨receiver⟩`
    `src_⟨sender⟩dst_⟨receiver⟩isn_⟨sender⟩isn_⟨receiver⟩)`
    for Send_other_traffic_keys and Receive_other_traffic_keys[6], and

– `(hash ⟨shared_key⟩addr_⟨sender⟩ addr_⟨receiver⟩src_⟨sender⟩dst_⟨receiver⟩isn_⟨sender⟩)` for Send_SYN_traffic_keys and Receive_SYN_traffic_keys[6]

When put together each segment was formed as shown in the following example:

```
(recv
(cat "EST" addr_b addr_a len3 flgs_b1 wdw_b src_b dst_a seq_b1 seq_a2
chksum_est0 k_id_b payld0
(enc (hash "EST" addr_b addr_a len3 flgs_b1 wdw_b src_b dst_a seq_b1
seq_a2 k_id_b payld0)
(hash kba addr_b addr_a src_b dst_a isn_b isn_a))))
```

4 TCP-AO Implemention

As RFC 5925 establishes, we implement two key derrivation functions (KDFs) and two functions to compute TCP segments' Message Authentication Codes (MACs). However, due to $SHA - 1$'s deprecation by the National Institute of Standards and Technology (NIST) in 2011[2], and the demonstration of an attack against it in 2017[10], we recommend support for SHA-1 be replaced with the Galois MAC (GMAC), a special case of the AES CMAC algorithm using Galois counter mode.

In GMAC's implementation, the plain text input is taken as zero hence the resultant output is the authentication tag only. [5] Thus, the two MAC algorithms that we implement are the standard AES-128 and GMAC-128.

Given that TCP-MD5 has already been implemented and may still be in use, we support backwards compatibility with TCP-MD5. However, the lack of published implementation of TCP-AO SHA-1, makes it safe to deprecate completely. Hence, we recommend it be replaced.

The MAC is computed over the following fields as specified by RFC 5925.

1. "The Sequence Number Extension (SNE)"
2. "The IP pseudo-header: IP source and destination addresses, protocol number, and segment length, all in network byte order"[6]
3. The TCP Header by default including Options. [TCP checksum and Mac zeroed out]. The TCP AO option is always included in the MAC

computation. The interface provides the user with an option to specify if the MAC should be computed over the other options or not.

4. "The TCP data, i.e., the payload of the TCP segment"[6].****

5 Results

5.1 CPSA Protocol Analysis

Given its primary implementation as a Dolev-Yao model[11], our ability to analyze the respective connection states of the honest participants running TCP-AO was severely restricted. As we will explain further in our conclusion, the Dolev-Yao model and by extension CPSA required too much abstraction to allow for meaningful analysis of scenarios such as the adversary sending an unauthenticated segment. However, in spite this, CPSA did provide cursory indications that the TCP-AO exchanges met our security goals.

5.2 Simple Exchange -No updates to the Master Key

While analysis with CPSA indicated that the two honest participants a generic exchange would agree on most values received, some of the header fields did not resolve as expected. That is, when looking at the initiator's perspective, the sequence numbers of the sixth seventh and eighth transmissions looked like they could effectively be altered by an adversary. When looking at this protocol as a variant of TCP, this mismatch did not make sense. As indicated in RFC 793[9], TCP requires segment sequence numbers within a given connection increment by one. This is done to ensure all segments sent are received, and that segments within the receive window that arrive out of order can be properly reordered by their recipient. As such it is improbable that an adversary would be able to modify the sent sequence number or insert a different one at any given point. The only feasible way by which an adversary could do this would be by means of replaying a segment from a prior exchange in which the source IP, destination IP, ISN's, source ports and destination ports matched. If we assume ISN's are chosen via a pseudorandom generator, the overwhelming likelihood is that this would not be possible.

5.3 Full Exchange - One party rolling over to a new Master Key

Interestingly, the introduction of a single master key rollover resolved the sequence number mismapping seen in the previous exchange. This is likely due to the introduction of the RNextKey field in the message prior to the key rollover and to the key rollover itself. Not only did the RNextKey field alter the pre-rollover message's structure, it indicated that the initiator A was ready to receive transmissions with MACs computed using the new key. Assuming a fresh key was used, the newly derrived traffic key could no longer match those used in prior exchanges. This, in turn prevented the adversary from a chance of being able to alter the messages' sequence numbers via replay attack. Bearing this difference and TCP's core requirements in mind, the simple exchange model's failure to resolve the sequence numbers in the final three segments pointed towards insufficient specification of sequence numbers' relation to one another. As further attempts to debug our input showed, the additional specification needed to assert this relationship was not allowed within CPSA[7].

6 Conclusions

6.1 Modeling Limitations with CPSA

As we alluded in our results, the simple exchange's mismatched sequence numbers were the first indication that CPSA was likely incompatible with the analysis we wanted to perform on TCP-AO. Further reading of the Dolev-Yao model [4], upon which CPSA is based, confirmed this.

TCP-AO's stated security goals are not as straightforward as described in either the Dolev-Yao model, or the expanded CPSA model. As mentioned above and in 7 the Dolev-Yao model requires honest parties to be stateless. That is, the set of message primitives, (also referred to as the alphabet) sent and received by honest parties not only need to be defined at the start of the protocol iteration, the actors X and Y are further limited in how they can compose the elements of said alphabet to build messages. [4]. Given this restriction, the goal of being able to maintain some form of authenticated connection in the face of a legitimate reset due to one machine rebooting is undefined in these models, as are depictions of the TCP rules dictated by the sliding window protocol. That is, whether or not a given message will be accepted based on the receiver's state, the format of the next message sent by a honest participant etc. To account for these nuances we would need a model that could do the following: Remove the statelessness requirement from honest participants. That is allow honest participant's states to be represented as a Finite State Machine (FSM) or a logic tree and defining receipt actions as functions of the incoming message and their state rather than enforcing that the steps within the protocol be stateless and statically defined. As such we plan on looking into the following alternate tools for analysis:

- Tamarin Prover: Pivotal in discovering a key vulnerability in the TLS 1.3's delayed authentication feature, the Tamarin prover inherently describes the transitions within a model's overall state that result from step within a protocol. Further work will focus on getting familiar with the Tamarin's grammar as well as its model to ascertain its ability to accurately model the intricacies of TCP-AO, and hopefully to give more accurate models of the possible exchanges as specified in RFC 5925.

- CPSA advanced modules: While its basic functionality module uses a more restrictive Dolev-Yao model, its goal module appears to have a more flexible model structure. As Guttman et al illustrate in their 2014 paper, Security Goals and Evolving Standards, this functionality boasts the ability to describe protocols in terms of security goals and a more minimalist and generalized execution structure. Specifically, they note a mechanism by which they can use the CPSA's goal syntax to describe the interaction of components such as API endpoints within a more complex protocol such as TLS.

References

[1] A. Heffernan. *Protection of BGP Sessions via the TCP MD5 Signature Option*. RFC 2385. Obsoleted by RFC 5925. RFC Editor, Aug. 1998. URL: `https://www.rfc-editor.org/rfc/rfc2385.txt`.

[2] Elaine Barker and Allen Roginsky. *NIST Special Publication 800-131A*. Nov. 6, 2015. URL: `https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-131Ar1.pdf`.

[3] D. Borman. *TCP Options and Maximum Segment Size (MSS)*. RFC 6691. RFC Editor, July 2012. URL: `https://www.rfc-editor.org/rfc/rfc6691.txt`.

[4] *On the Security of Public Key Protocols*. IEEE Transactions on Information Theory. Vol. IT-29. 2. IEEE, pp. 198–208.

[5] G. Lebovitz and E. Rescorla. *Cryptographic Algorithms for the TCP Authentication Option (TCP-AO)*. RFC 5926. RFC Editor, June 2010, pp. 1–13. URL: `https://www.rfc-editor.org/rfc/rfc5926.txt`.

[6] A. Mankin J. Touch and R. Bonica. *The TCP Authentication Option*. RFC 5925. RFC Editor, June 2010. URL: `https://www.rfc-editor.org/rfc/rfc5925.txt`.

[7] John D. Ramsdell Moses D.Liskov Joshua D. Guttman and Paul D. Rowe. *cpsa: Symbolic cryptographic protocol analyzer*. June 24, 2016.

[8] A.Sujeet Nayak and B. Weis. *SHA-2 Algorithm for the TCP Authentication Option (TCP-AO)*. Working Draft. Version 02. Jan. 26, 2015. (Visited on 11/18/2018).

[9] J. Postel. *Transmission Control Protocol*. RFC 793. RFC Editor, Sept. 1981. URL: `https://www.rfc-editor.org/rfc/rfc793.txt`.

[10] *Research Results on SHA-1 Collisions*. National Institute of Standards and Technology. Feb. 24, 2017. URL: `https://csrc.nist.gov/News/2017/Research-Results-on-SHA-1-Collisions` (visited on 11/18/2018).

[11] Javier Thayer Shaddin F. Doghmi Joshua D. Guttman. "Searching for shapes in cryptographic protocols". In: *Tools and Algorithms for Construction and Analysis of Systems (TACAS)* (2007).

7 Appendix A: Additional Background - The Dolev-Yao Model

Designed to provide a formal testing framework for protocols using public-key cryptosystems, the Dolev-Yao model evaluates two types of two-party exchange protocols - Cascade Protocols and Name-Stamp Protocols. In both cases, they assume a "perfect public key system." Or rather

- All one-way functions used are unbreakable;
- public keys are all stored in a secure public directory Dir. That is:
  - tampering with Dir is not possible
  - everyone has access to all entries $E_x \in Dir$

In addition to this assumption, the Dolev-Yao model assumes:

1. In the modeled two-party protocols there is no trusted third party to assist in encryption and decryption of messages.
2. Protocols are uniform. This means that any pair of users communicating with a protocol T will use the same message format for each message M_i transmitted, where $0 < i \leq t, t := numberoftransmissionsinT$.
3. Any legitimate user X can be a receiver for another legitimate user Y.
4. The adversary Z is an active one. Specifically, Z:
   - can obtain any message transmitted through the network.
   - is a legitimate network user. Therefore Z can initiate a conversation with any other legitimate user on the network.
   - by assumption (3) Z will have the opportunity to be a receiver to some initializer X in another iteration of T.

Using these assumptions, they define the two-party cascade and namestamp protocols as follows:

A two-party cascade protocol is defined as a protocol in which X and Y are able to modify a message with the following finite sets of operations or messages:

- X's transmission at any iteration i will be an element of $\{Enc_x, Enc_y, Dec_x, \lambda\}$ $1 \leq i \leq t$
- Y's transmission at any iteration i will be an element of $\{Enc_x, Enc_y, Dec_y, \lambda\}$ $1 \leq i \leq t', t' = t, t-1$

Where Enc_a denotes encryption under some user A's public key, Dec_a denotes decryption under A's secret key, λ denotes the identity, or empty string and t and t' respectively denote the number of transmissions by X and Y in protocol T. [4]

A two-party namestamp protocol is defined as a protocol in which at a given transmission j, X and Y can modify a message M_j using the operations from a cascade protocol and the following name-based operations:

1. append an identity i_a for some participant A, where $i_a M_j = M_j A$;
2. name-match d_a , where $d_a M_j A = M_j$;
3. delete d, where $dM_j = head(M_j)$;

More formally, these operations are denoted:

- $\tilde{x}_j \in \{Enc_x, Enc_y, Dec_x, i_x, d_x, d, \lambda\}$ where $\tilde{x}_j$ denotes X's action at transmission j.
- $\tilde{y}_j \in \{Enc_x, Enc_y, Dec_y, i_y, d_y, d, \lambda\}$ where $\tilde{y}_j$ denotes Y's action at transmission j.

For both two-party models a given, protocol is defined to be insecure under the Dolev-Yao model if there exists a word $\gamma \in \{\Sigma_1(Z) \cup Tr_i\}$ such that the completely reduced phrase

$$\overline{\gamma Tr_i} = \lambda$$

Where

- Tr_i is the set of all messages sent between parties up to and including transmission i.
- $\Sigma_1(Z)$ is the set of actions available to Z as a legitimate network user, but not as a participant in the current iteration of T. (Such as $Enc_a \forall$ users A in E, in the case of a cascading protocol.)

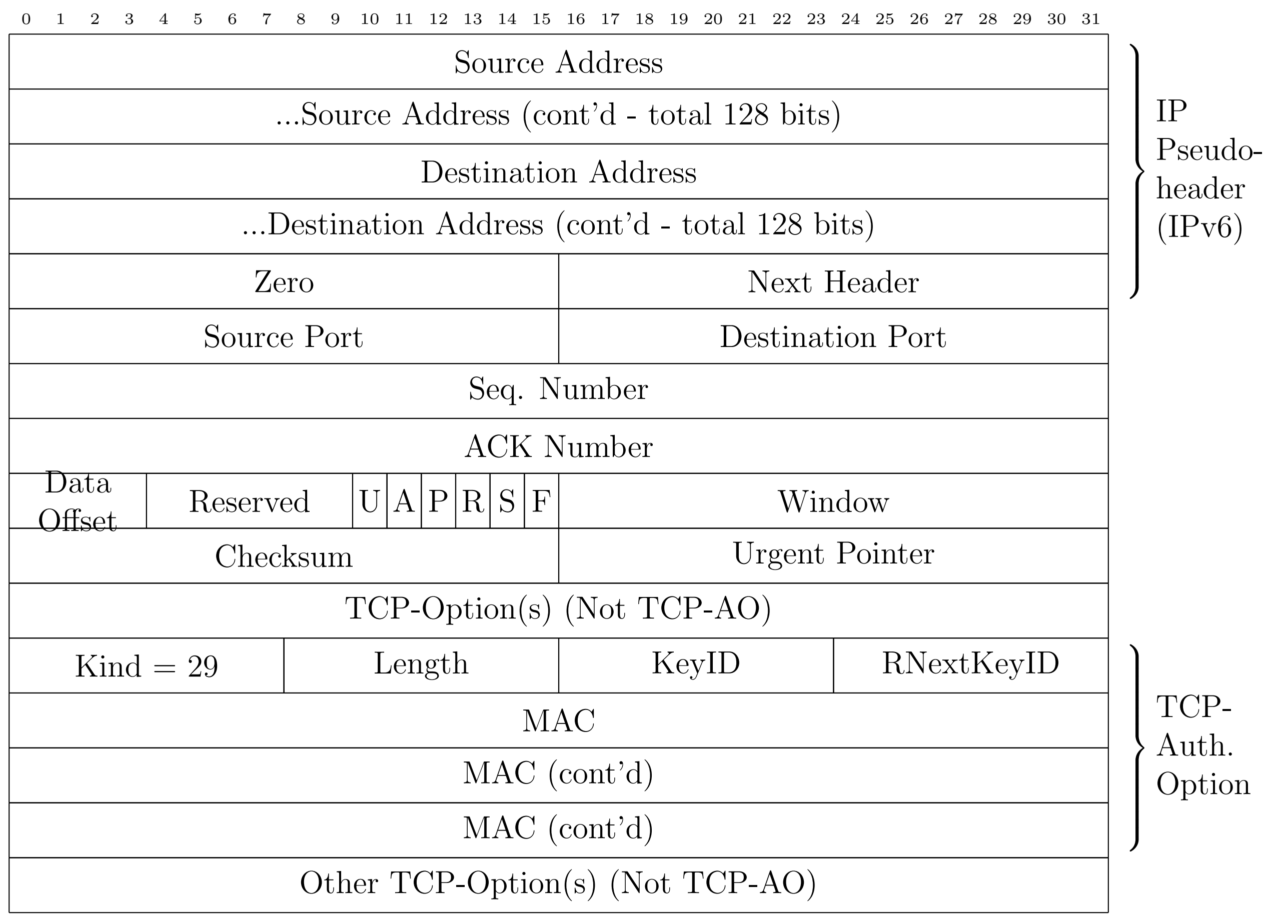

Figure 2: Header of a TCP segment running TCP-AO over IPv6. Only fields in the IP pseudoheader differ here.