ACME Protocol

# Conventions

* Encryption is denoted Encrypt(P, K) where P is the plaintext and K is the key. Encryption is AES128-CBC. Padding style is PKCS #7. A random IV is chosen with each invocation of Encrypt(P, K) and prepended to the resulting ciphertext. Decryption is denoted Decrypt(C, K).
* All binary data transmitted is little-endian.
* All strings transmitted are null-terminated and UTF-8 encoded.
* All timestamps are 64-bit integers indicating the number of milliseconds elapsed since midnight on 1/1/1970 UTC.
* A | B denotes the byte concatenation of A and B (*not* alternation).
* Each message begins with a one-byte constant value indicating the message type.
* On the server, if at any time a received message is found to have improper formatting, padding, etc., processing of that message immediately terminates and the server responds with message AUTHX\_FAILURE and terminates the TCP conversation. For brevity, these error checks are not spelled out fully in the protocol description but they can be inferred.

# Server Security Model

* The server is configured with a 128-bit secret key, KS. This key is never shared.
* The server is equipped with a database of authorized users. For each user, U, the database contains a plaintext password PU, as well as a list of groups that the user is a member of. Each group is identified by a string.
* There is a well-known user named “admin” belonging to a group named “admin”.
* There is a well-known user named “guest” belonging to a group named “guests”. The password on this account is strong as well.
* There are two defined commands:
  + whoami: This command returns a JSON string containing the username and all groups the user belongs to.
  + getflag: This command returns the CTF flag. Only members of the “admin” group are authorized to execute this command. In this case of refServer.py, this command is implemented but does not return a valid CTF flag.

# Informal Overview of Protocol

To log on, the client first sends the username. The server responds with a challenge nonce and an encrypted cookie. The client authenticates by answering the challenge using knowledge of the password. The client passes the answer together with the cookie back to the server. If the client’s response is correct, the server issues a session ticket containing a timestamp and the user’s identity in encrypted form. The client presents this ticket to establish its identity whenever issuing a command. To log off, the client simply discards the ticket.

# Protocol Messages

## LOGON\_REQUEST (Message Type 0x01)

Message format: Client → Server: 0x01 | U

Explanation: The client sends this message to the server to initiate authentication with username U.

Processing: The server executes the following algorithm upon receipt:

Set Nonce = 8-byte random nonce

Set Timestamp = current timestamp

Set ChallengeCookie = Base64Encode(Encrypt(Nonce | U | Timestamp, KS))

Respond with message LOGON\_CHALLENGE(Nonce, ChallengeCookie)

## LOGON\_CHALLENGE (Message Type 0x02)

Message Format: Server → Client: 0x02 | Nonce | ChallengeCookie

Explanation: Server response to LOGON\_REQUEST.

Processing: The client executes the following algorithm upon receipt:

Set R = SHA256(Nonce | P), where P is the password for authentication

Respond with message LOGON\_RESPONSE(R, ChallengeCookie)

## LOGON\_RESPONSE (Message Type 0x03)

Message Format: Client → Server: 0x03 | R | ChallengeCookie

Explanation: Client response to LOGON\_CHALLENGE.

Processing: The server executes the following algorithm upon receipt:

Set D = Decrypt(Base64Decode(ChallengeCookie), KS)

Scan D sequentially as follows:

Set Nonce = 8 bytes

Set U = null-terminated string

Set Timestamp = 8 bytes

If Timestamp is too old (> 5 min):

Respond with message AUTHX\_FAILURE

End

Retrieve information about user U from the database

If there is a no record in the database of a user named U:

Respond with message AUTHX\_FAILURE

End

Set CorrectResponse = SHA256(Nonce | PU)

If R != CorrectResponse

Respond with message AUTHX\_FAILURE

End

Set TicketTimestamp = current timestamp

Set Identity = JSON string: { user: U, groups: [ G1, G2, ... ] }

where G1, G2, ... are the names of the groups that U belongs to

Set Ticket = Base64Encode(Encrypt(Identity | TicketTimestamp, KS))

Respond with message LOGON\_SUCCESS(Ticket)

## LOGON\_SUCCESS (Message Type 0x04)

Message Format: Server → Client: 0x04 | Ticket

Explanation: Server provides a ticket that the client can use to prove its identity.

## AUTHX\_FAILURE (Message Type 0x05)

Message Format: Server → Client: 0x05

Explanation: Server informs the client that the client failed to authenticate, or the user is not authorized to issue the requested command, or that the client’s message could not be understood.

## COMMAND (Message Type 0x06)

Message Format: Client → Server: 0x06 | Ticket | Command

Explanation: Client requests execution of the command specified by the string Command. Ticket must be a valid, current ticket received via a LOGON\_SUCCESS message.

Processing: The server executes the following algorithm upon receipt:

Set D = Decrypt(Base64Decode(Ticket), KS)

Scan D sequentially as follows:

Set IdentityFromTicket = JSON string (UTF-8, null-terminated)

Set Timestamp = 8 bytes

If Timestamp is too old (> 1 hour):

Respond with message AUTHX\_FAILURE

End

Set U to the string IdentityFromTicket.user

Iterate over IdentityFromTicket.groups, collecting the results into an array of strings, G

Set Identity = object expressing U and G

If Command = “whoami”:

Set Result = JSON string: { user: Identity.U, groups: [ G1, G2, ... ] }

where G1, G2, ... are the elements of Identity.G

Else If Command = “getflag”:

If G contains the string “admin”:

Set Result = CTF flag

Else:

Respond with message AUTHX\_FAILURE

End

Else:

Respond with message AUTHX\_FAILURE

End

Respond with message COMMAND\_RESULT(Result)

## COMMAND\_RESULT (Message Type 0x07)

Message Format: Server → Client: 0x07 | Result

Explanation: Server responds with the text output of a command submitted by the client.