The

cbcp

Capability-Based Command Protocol Specification

Version 1.0 Draft

QuasiOS

February 2022

Contents

List of Figures

List of Tables

1. Introduction

The Capability-Based Command Protocol (CBCP) is an application layer host-to-host communication protocol. It is intended for well-defined networks where the identities and possible interactions of connected hosts are known. In CBCP, an interaction implies a client issuing a command to a server. The server only accepts the command if the client has a capability that grants access to the command in question. Capabilities in CBCP are based on extended password capabilities described in [1]. CBCP ensures confidentiality, integrity and authenticity using concepts from both public and symmetric key cryptography. CBCP addresses availability by limiting the amount of work being done for malformed or malicious packets.

1.1. About this Document

The rest of Section 1 motivates the creation of the CBCP and explains the scope of the protocol. Section 2 covers the broader concepts that underpin the design of CBCP. Section 3 specifies the functional aspects of the protocol such as packet formats and preconditions. Possible future work is outlined in Section 4.

1.2. Motivation

Cyber security has become increasingly important to the industrial sector with the advance of Industry 4.0, which introduces ever more connectivity and automation. This shift in paradigm has put the vulnerability of industrial networks under scrutiny. Policies to address cyber security in the past have often involved physical partioning of business- and industrial networks. However, such policies are mostly incompatible with the goals of Industry 4.0.

To address these circumstatnces, CBCP has been designed to facilitate Industry 4.0 by taking advantage of the fact that industrial networks can often be well-defined, i.e., the identities and possible interactions of connected hosts are known. It does so by requiring a formal specification of the network. This network specification serves as a basis for generating the appropriate public-private key pairs to ensure authenticity in all communication in addition to capabilities used to limit which host-to-host iteractions are possible.

Flexibility has played a role in the design of the CBCP as it can interoperate with multiple underlying network transport protocols. Figure 1 shows where CBCP fits in the Open Systems Interconnection (OSI) Model.

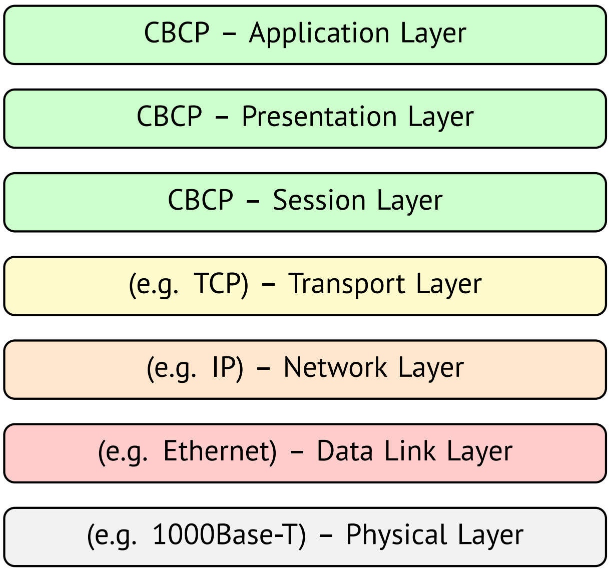

Figure 1: CBCP in relation to the OSI Model. Note that CBCP is neither dependent on TCP, IP, nor Ethernet; any underlying network implementation fulfilling the requirements of CBCP would suffice. In the OSI model, the CBCP takes on responsibilities relevant to the session layer, presentation layer, and application layer.

1.3. Scope

As noted above, the CBCP has been designed to provide increased cyber security in closed, mostly static, networks – e.g. a factory floor with robots and other machines needing to coordinate in a well-defined way. CBCP additionally aids in documenting and reasoning about the network by making the identity and capabilities of every participating host explicit.

2. Conceptual Model

This section covers the broader concepts that underpin the design of CBCP.

2.1. CBCP Network

The CBCP works in a network of connected machines. Each machine is a host offering zero or more operations grouped into interfaces. Hosts can command other hosts to perform specific operations by sending a command. The commands are only accepted on the serving host if the client host includes a valid capability (see Section 3.6 about command validation).

2.2. Hosts

In a CBCP network, hosts communicate in a point-to-point manner. Every host can function both as a client and a server simultaneously. When a given CBCP network configuration has a pair of interacting hosts, it is the responsibility of the client to initiate the connection.

This happens through a four-way handshake protocol described in Section 3.5.

2.3. Interfaces

Interfaces define groups of related operations a server can perform when issued a command by a client. They are comparable to the interfaces of objects in object oriented programming.

Access to specific operations are managed through capabilities. If the same set of permitted commands is shared among multiple clients, they can share the same capability.

Associated with an interface is a revokation table of capability entries. Every valid capability to a particular interface must map to one of the capability entries in this table.

A server is able to revoke some or all access rights of a capability via its associated entry in the revokation table.

2.4. Commands

In CBCP, commands are requests sent (see Figure 2) by a client for a server to perform an operation offered in one of the server's served interfaces. A command can be accompanied by an application-defined payload which can provide contextual data for the operation.

When the server has completed an operation it has been commanded to perform, it responds to the client with confirmation of completion and optionally an application-defined payload providing information about the result of carrying out the operation.

2.5. Capabilities

Conceptually, a capability corresponds with a set of permitted commands for a particular interface along with a secret key used for validation.

The most permissive capability would be one where all commands of the interface were permitted.

Capabilities are what clients use to prove that they are allowed to issue a particular command at a server.

3. Functional Specification

This section specifies the functional aspects of the CBCP, including conventions, sub-protocols, as well as concrete network packet formats, implementation requirements and recommendations.

3.1. Implementation Requirements and Recommendations

Implementations of the CBCP **must** provide the following aspects:

- Full duplex command issuing and handling. This implies that a host application must be able to act simultaneously as a client and as a server.
- The ability to handle handshake requests at all times, see Section 3.5.

Implementations of the CBCP **should** provide the following aspects:

- The ability to revoke capabilities at runtime.
- The ability to look up hosts, interfaces, and commands by name.
- The ability as a command-issuing host to specify which capability to use when multiple capabilities give access to the command in question.

3.2. System Requirements and Recommendations

The underlying transport layer **must** provide the following aspects:

- Reliable data transfer.
- Enough isolated communication channels to satisfy the requirement that each pairwise connection between hosts has two isolated communication channels, one command channel, and one control channel.

The computing platform **must** provide the following aspects:

- RSA encryption and decryption.
- AES encryption and decryption.

The computing platform **should** provide the following aspects:

- Safe storage of encryption keys and secrets. The degree of safety is intentionally left unspecified; in general, the more critical the application, the stronger the emphasis on this aspect should be.

3.3. Data Requirements

This section specifies the data needed by implementations of the CBCP.

This data is expected to be generated in advance of the network becoming operational. Future CBCP specifications may expect dynamic changes of this data (see Section 4).

The following tables describe the data from the perspective of a particular host denoted, *Self*.

Data About *Self*

Host Name	A name that uniquely identifies the host, *Self*. Must not exceed 256 characters in length.
Address(es)	One or more transport-layer-specific addresses. For TCP, this would likely be an IP-address and port number.
Public Key	The public RSA key of *self*.
Private Key	The private RSA key of *self*.
Own Interfaces	Interfaces Hosted by *Self*. Table 2 specifies what data is needed for each such interface.
Remote Interfaces	What interfaces *Self* relies on at different hosts.
Capabilities	Which capabilities *Self* owns. See Table 4 for a specification of what data a capability corresponds to.

Table 1: Data About *Self*

Data About Each Interface Hosted by *Self*

Interface Name	A name that uniquely identifies the interface. Must not exceed 256 characters in length.
Capability Master Secret	A 128-bit secret number used for validating capabilities for this interface.
Command Name List	An ordered list of names to commands this interface can receive. The position of each command name should correspond to the command ID that is transmitted in command packets (see Figure 2).

Table 2: Data About Each Interface Hosted by *Self*

Data About Each Remote Interface Used by *Self*

Interface Name	A name that uniquely identifies the interface. Must not exceed 256 characters in length.
Used Remote Commands	The names, and command IDs of the remote commands used in this remote interface. The amount of commands must not exceed 64.

Table 3: Data About Each Remote Interface Used by *Self*

Data Corresponding to a Capability

Permitted Commands	The set of permitted commands represented as multiple bit vectors which need to be bitwise AND-ed together. Each bit vector is called a reduction sub-field. The bit position (in little-endian bit ordering) in each bit vector corresponds to the command ID in the interface. A 1-bit corresponds to the command being permitted, a 0-bit corresponds to the command *not* being permitted. Each reduction sub-field is bitwise AND-ed together to produce the final set of permitted commands.
Secret	A secret derived from the permitted commands reduction sub-fields. This prevents unnoticable tampering with the reduction sub-fields. See Section 3.6 for how this is used in the verification of commands.

Table 4: Data Corresponding to a Capability

3.4. Commands

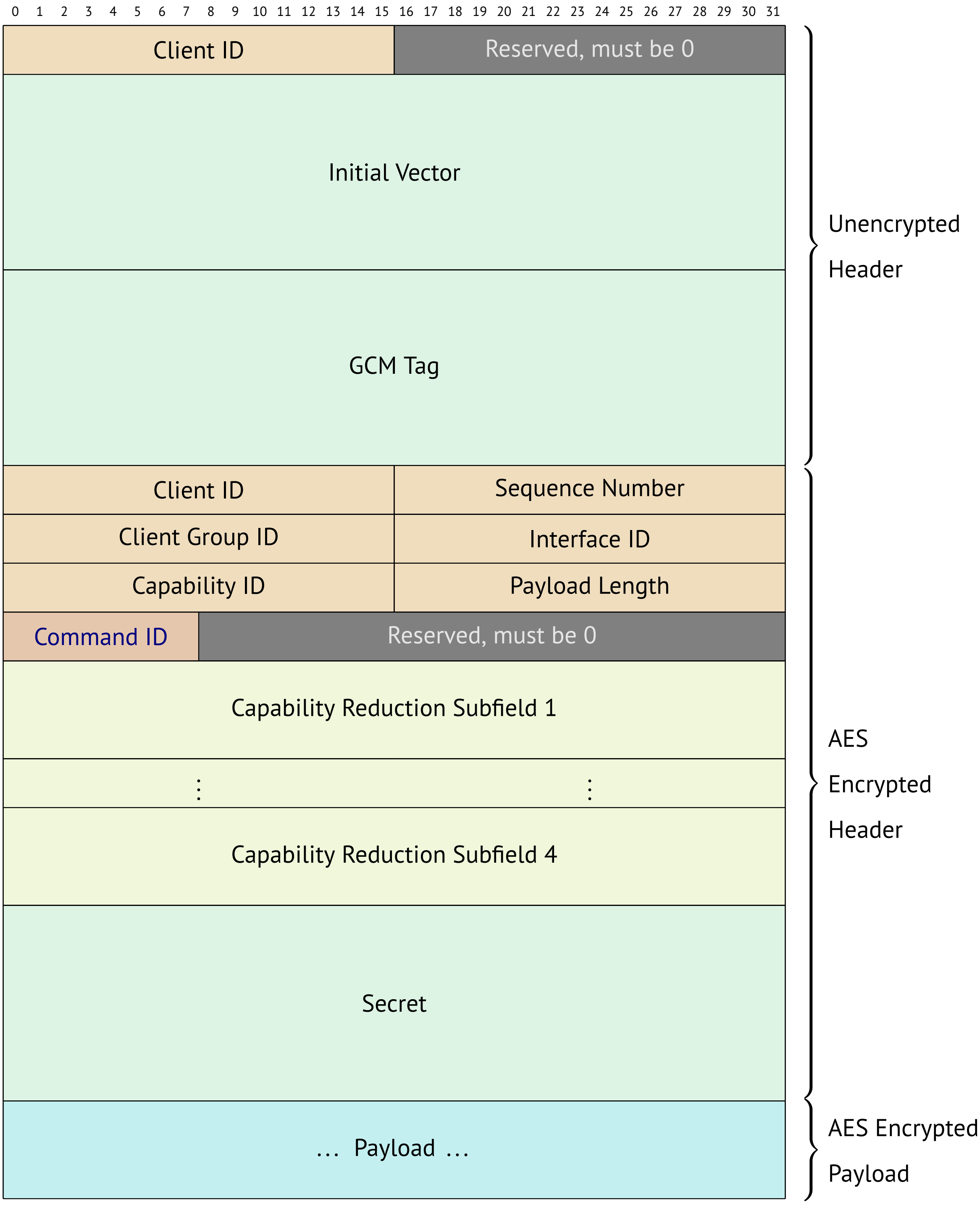

Figure 2: Command Packet Format. The numbers 0-31 at the top of this diagram denotes the bit offset of the column below. The full width of the diagram corresponds to 32 bits.

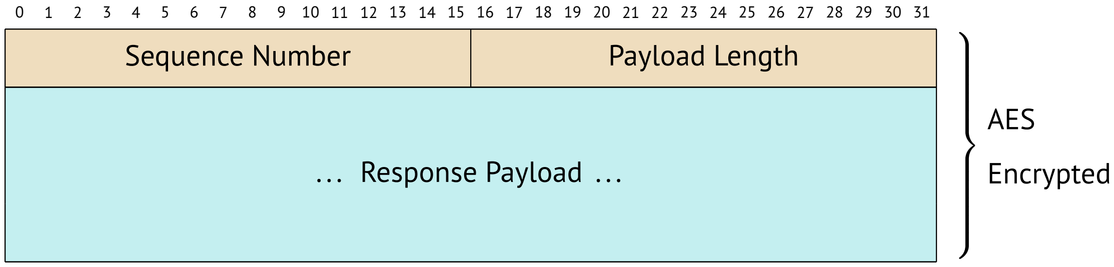

Figure 3: Command Response Packet Format. The numbers 0-31 at the top of this diagram denotes the bit offset of the column below. The full width of the diagram corresponds to 32 bits.

3.5. Connection and Handshake

For any pair of hosts needing to interact, a point-to-point connection needs to exist between them. It is always the responsebility of the client to establish the connection. Establishing a CBCP-connection happens via a four-way handshake illustrated in Figure 4.

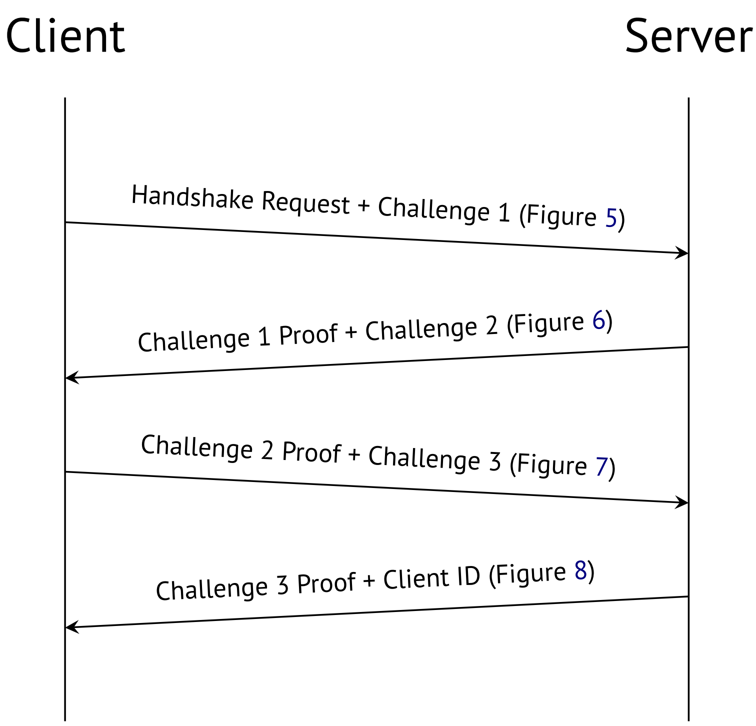

Figure 4: Four-Way Handshake Sequence

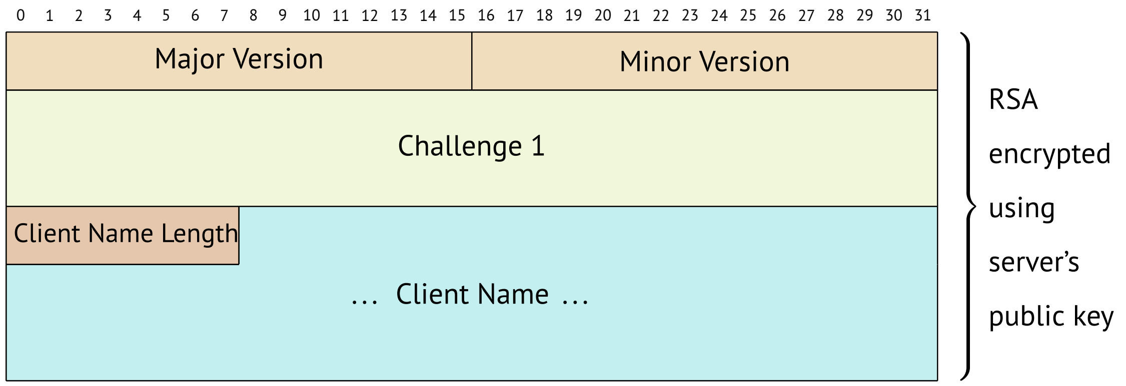

Figure 5: The initial handshake request sent by the client. It establishes the version of CBCP to be used for further communication. 'Challenge 1' represents a decryption challenge that the server must proof it can decypt by replying with the decrypted value. Finally, the name of the client is provided for the server to look up the data associated with the client. The numbers 0-31 at the top of this diagram denotes the bit offset of the column below. The full width of the diagram corresponds to 32 bits.

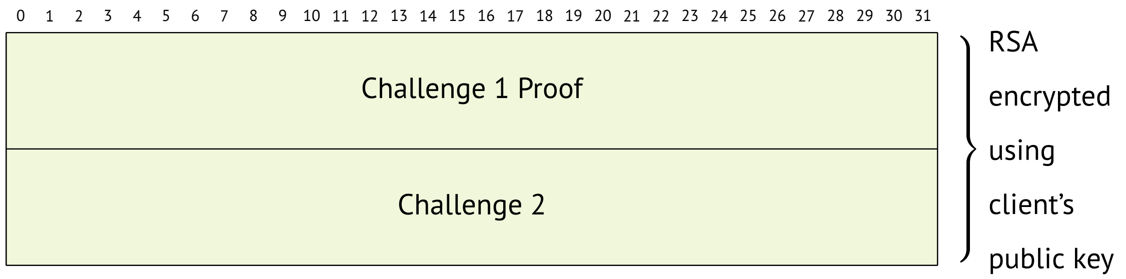

Figure 6: The server's response to the initial handshake request by the client. The numbers 0-31 at the top of this diagram denotes the bit offset of the column below. The full width of the diagram corresponds to 32 bits.

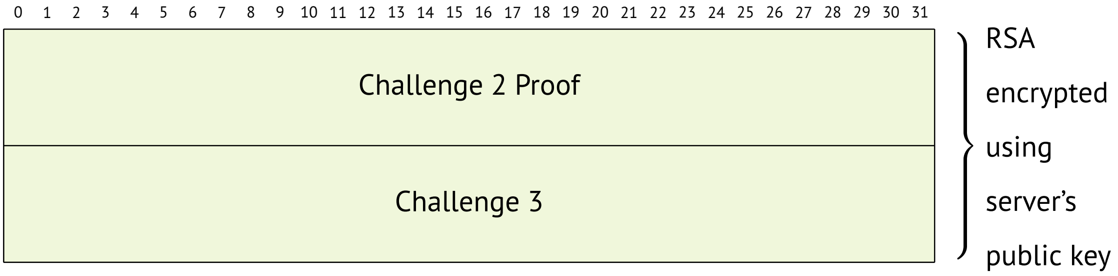

Figure 7: The client proofs the new challenge by the server and provides a third challenge. The numbers 0-31 at the top of this diagram denotes the bit offset of the column below. The full width of the diagram corresponds to 32 bits.

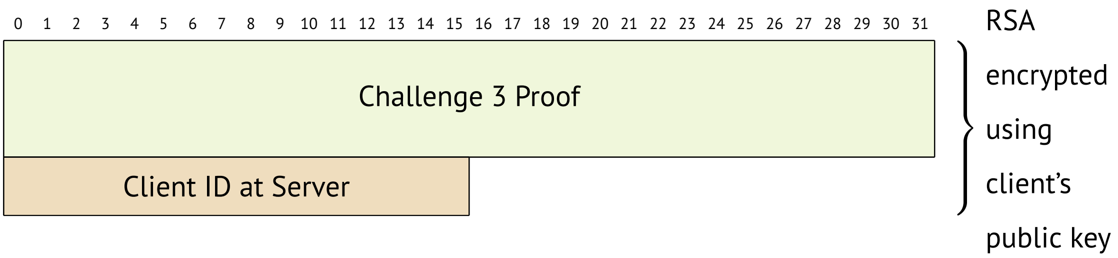

Figure 8: Server responds with the ID the client is supposed to identify itself with when issuing commands. The numbers 0-31 at the top of this diagram denotes the bit offset of the column below. The full width of the diagram corresponds to 32 bits.

CBCP Connection A connection in CBCP is always point-to-point and has two isolated communication channels: One for commands (and responses to those commands), and one for control messages like the handshake packets. These channels are referred to as the command channel and the control channel.

If a pair of hosts are mutually both client and server, the first host to reach the other with a handshake request packet becomes the initiator of the handshake.

Once a connection is setup, there is no distinction between client and server.

Handshake Outcome The outcome of the handshake is a shared secret key used to encrypt all commands and response packets. This key is a 128-bit AES key.

The connection state diagram in Figure 9 illustrates under which circumstances a handshake needs to be made.

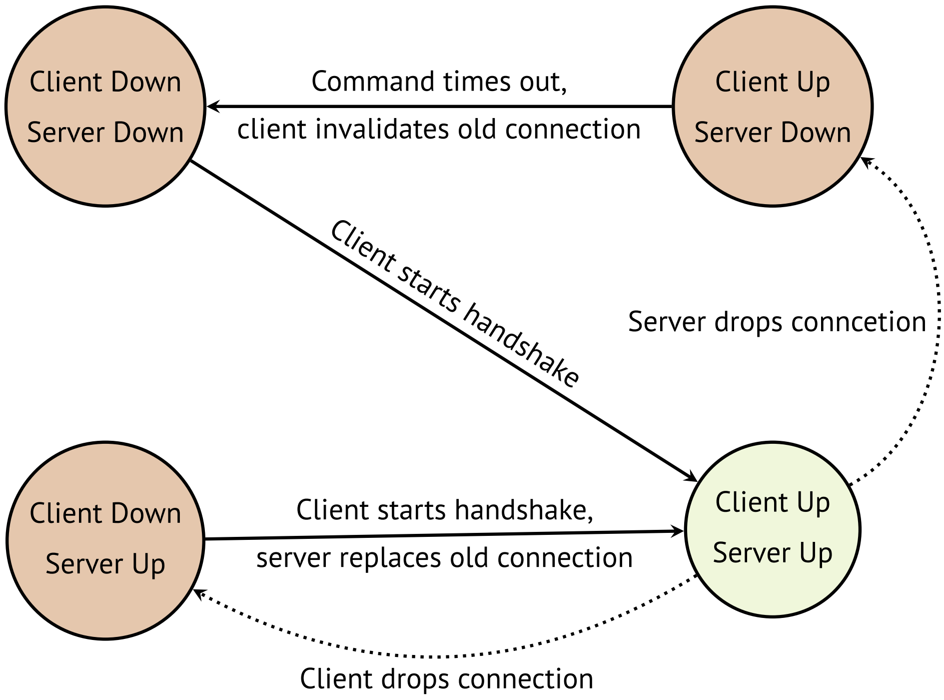

Figure 9: CBCP Connection State Diagram

When the network initially goes live, there are no connections between any pair of hosts; hence, for every client-server pair, the clients will initiate handshakes.

3.6. Command Validation

This section lists a C (ISO/IEC 9899:1999) reference implementation of the command validation procedure as well as the procedure for computing the capability secrets.

```
#include <stdint.h>      /* explicitly sized integer types */
#include <stdbool.h>     /* bool */
#include <string.h>      /* memcpy, memcmp */
#include <assert.h>      /* assert */
#include <openssl/aes.h> /* AES_set_encrypt_key, AES_ecb_encrypt */

#define REDUCTION_SUBFIELD_COUNT 4
#define KEY_BIT_COUNT 128

typedef struct {
    /* sizeof(Secret) == AES encryption block size */
    uint8_t bytes[16];
} Secret;

typedef struct {
    uint64_t subfields[REDUCTION_SUBFIELD_COUNT];
} Reduction_Field;
```

```c

typedef struct {
    uint16_t id;
    Secret secret;
    Reduction_Field reduction_field;
} Capability;

Secret compute_secret(Secret interface_master_secret, Capability capability) {
    Secret result = {0};
    Secret id_as_secret = {0};

    memcpy(&id_as_secret, &capability.id, sizeof(capability.id));

    AES_KEY key;
    AES_set_encrypt_key(
        (const uint8_t *)&interface_master_secret,
        KEY_BIT_COUNT,
        &key);

    AES_ecb_encrypt(
        (const uint8_t *)&id_as_secret,
        (uint8_t *)&result,
        &key,
        AES_ENCRYPT);

    for (int i = 0; i < REDUCTION_SUBFIELD_COUNT; ++i) {
        uint64_t reduction_subfield = capability.reduction_field.subfields[i];

        /* Early-out of loop if subfield is all one bits */
        if (~reduction_subfield == 0) {
            break;
        }

        /* Copy reduction subfield into a big enough buffer */
        Secret reduction_subfield_as_secret = {0};

        memcpy(&reduction_subfield_as_secret,
               &reduction_subfield,
               sizeof(reduction_subfield));

        /* Set partial result as new key */
        AES_set_encrypt_key(
            (const uint8_t *)&result,
            KEY_BIT_COUNT,
            &key);

        /* Encrypt lcrypto */
        AES_ecb_encrypt(
            (const uint8_t *)&reduction_subfield_as_secret,
            (uint8_t *)&result,
            &key,
            AES_ENCRYPT);
    }

    return result;
```

```
}

bool validate_command(
    uint8_t command_id,
    Capability capability,
    Secret interface_master_secret,
    uint64_t interface_command_field
) {
    assert(command_id < 64);

    uint64_t command_field = (1 << command_id);

    uint64_t final_field = command_field;

    /* Bitwise AND */
    final_field &= interface_command_field;

    for (int i = 0; i < REDUCTION_SUBFIELD_COUNT; ++i) {
        final_field &= capability.reduction_field.subfields[i];
    }

    if (final_field == 0) {
        /* Command not permitted */
        return false;
    }

    /*
    ** Validate reduction field by recomputing the secret and
    ** checking that it comes out the same as `capability.secret`.
    */

    Secret computed_secret = compute_secret(interface_master_secret, capability);

    if (memcmp(&computed_secret, &capability.secret, sizeof(Secret)) != 0) {
        /* The computed secret did not match the secret in the capability */
        return false;
    }

    /* Command permitted */
    return true;
}
```

4. Future Work

Type safe interface specification: Instead of only providing names of commands, data types for the command- and response payloads could likewise be specified in an appropriate interface definition language.

Dynamic network updates: Updates to the network configuration could happen incrementally. Then differences from the previous configuration could be sent to the affected hosts as updates that could be applied at runtime.

The abilitiy for hosts to broadcast commands to groups of hosts. Different communication methods should in general be explored; in particular, event-driven publisher/subscriber communication should be considered for enabling more decoupled network configurations.

A distinction between long and short running commands with the purpose of responding to the client at the onset of long running commands so it does not need a long time out.

5. Acronyms

CBCP

Capability-Based Command Protocol. 3–7, 11, 13, 18

IP

Internet Protocol. 4, 7

OSI

Open Systems Interconnection. 3, 4

TCP

Transmission Control Protocol. 4, 7

6. Glossary

authenticity

A security attribute that implies that the identity of hosts in the network can be verified and thus trusted. 3

availability

A security attribute that implies that services are accessible to authenticated users when needed. 3

capability

A set of access rights associated with a particular protected entity. In CBCP, the entity being protected is an interface to a collection of operations. 18, 19

capability entry

An entry of a revokation table that a particular capability maps to. It contains information sufficient for revoking some or all access rights of the capability that maps to it. 5, 20

client

A host that has license to one or more capabilities offered by one or more servers. 3, 5, 6, 11, 13, 14, 20

command

See Section 2.4.
18

command channel

An isolated communication channel dedicated to commands and responses to those commands. 6, 13

command ID

An 8-bit number that uniquely identifies a particular command in an interface. Used in the command packet format (see Figure 2). 8–10

computing platform

An environment in which software is executed. This includes the operating system, available software libraries, and capabilies of the hardware. 6

confidentiality

A security attribute that implies that information shared between trusted parties remains private to those trusted parties. 3

control channel

An isolated communication channel dedicated to meta control messages like handshake packets. 6, 13

full duplex

An attribute of a two-party communication channel implying that both participants can both send and receive simultaneously. 6

host

A computer connected to the network. Read about the roles of a host in Section 2.2. 17–20

industry 4.0

A common abbreviation for The Fourth Industrial Revolution – speculated to be the next big advancement of the modern industrialized world. Industry 4.0 implies increased automation through interconnected production machinery and internet connectivity for remote monitoring and management. 3

integrity

A security attribute that implies that sent messages arrive at the recipient unchanged. 3

interface

See Section 2.3.
18, 20

license

A cryptographic entity that proofs ownership of a capability. 18

must

Used in this specification to indicate that something is a requirement. 6

packet

A unit of data transmitted over the network. 3

point-to-point

An attribute of a network connection implying that only two hosts will be able to communicate via the connection. 5, 11, 13

revokation table

A table of capability entries belonging to an interface at a particular server. 5, 18

server

A host that offers one or more interfaces to one or more clients. 3, 5, 6, 13, 14, 18, 20

should

Used in this specification to indicate that something is a recommendation. 6

7. References

[1] Lanfranco Lopriore. “Access right management by extended password capabilities.” In: *International Journal of Information Security* 17.5 (2018), pp. 603–612.

A. CBCP Configuration File Format

```
(*
GRAMMAR FOR CAPABILITY-BASED COMMAND PROTOCOL CONFIGURATION FILES
EBNF variant defined in ISO/IEC 14977:1996(E)
Version 1.0
*)

CbcpConfiguration   = VersionSection,
                      HostsSection,
                      [GroupsSection],
                      InterfacesSection,
                      ImplementsSection,
                      CapabilitiesSection ;

VersionTitle        = "!", Cc, Bb, Cc, Pp ;
HostsTitle          = "!", Hh, Oo, Ss, Tt, Ss ;
GroupsTitle         = "!", Gg, Rr, Oo, Uu, Pp, Ss ;
InterfacesTitle     = "!", Ii, Nn, Tt, Ee, Rr, Ff, Aa, Cc, Ee, Ss ;
ImplementsTitle     = "!", Ii, Mm, Pp, Ll, Ee, Mm, Ee, Nn, Tt, Ss ;
CapabilitiesTitle   = "!", Cc, Aa, Pp, Aa, Bb, Ii, Ll, Ii, Tt, Ii, Ee, Ss ;

VersionSection      = {Space}, VersionSectionHeader, SpaceLine, Version, End ;
HostsSection        = HostsTitle, End, [HostDefList], End ;
GroupsSection       = GroupsTitle, End, [GroupDefList], End;
InterfacesSection   = InterfacesTitle, End, [InterfaceDefList], End ;
ImplementsSection   = ImplementsTitle, End, [ImplementsDefList], End ;
CapabilitiesSection = CapabilitiesTitle, End, [CapabilityDefList], {Space} ;

Version             = VersionMajor, ".", VersionMinor ;
VersionMajor        = "1" ;
VersionMinor        = "0" ;

HostDefList         = HostDef, {End, HostDef} ;
HostDef             = HostName, Sep, HostAddressList ;
```

```
HostAddressList      = HostAddress, {Sep, HostAddress} ;
HostAddress          = NetImplName, SubSep,
                         ? Network Implementation Specific
                           String Not Containing "\n" ?;

GroupDefList         = GroupDef, {End, GroupDef} ;
GroupDef             = GroupName, Sep, HostName, {SubSep, HostName} ;

InterfaceDefList     = InterfaceDef, {End, InterfaceDef} ;
InterfaceDef         = InterfaceName, Sep, CommandNameList ;

ImplementsDefList    = ImplementsDef, {End, ImplementsDef} ;
ImplementsDef        = ServerName, Sep, InterfaceName, {SubSep, InterfaceName} ;

CapabilityDefList    = CapabilityDef, {End, CapabilityDef} ;
CapabilityDef        = ClientName, Sep,
                       ServerName, Sep,
                       InterfaceName, Sep,
                       CommandNameList ;

CommandNameList      = CommandName, {SubSep, CommandName} ;

HostName             = Name ;
GroupName            = "@", Name ;
InterfaceName        = Name ;
NetImplName          = Name ;
CommandName          = Name ;
ClientName           = HostName | GroupName ;
ServerName           = HostName | GroupName ;

Name                 = NameBoundsChar, [{NameMiddleChar}, NameBoundsChar] ;
NameBoundsChar       = Letter | Digit | "-" | "_" | "+" | "." | "/" ;
NameMiddleChar       = NameBoundsChar | " " ;

SubSep               = {SpaceLine}, ",", {SpaceLine} ;
```

```
Sep             = {SpaceLine}, ";", {SpaceLine} ;
End             = {Space}, "\n", {Space} ;

Space           = SpaceLine | SpaceLineEnd ;
SpaceLineEnd    = "\n" | "\r" | "\v" | "\f" ;
SpaceLine       =  " " | "\t" ;

Digit           = "0" | "1" | "2" | "3" | "4"
                | "5" | "6" | "7" | "8" | "9" ;

Letter          = Aa | Bb | Cc | Dd | Ee | Ff | Gg | Hh | Ii | Jj
                | Kk | Ll | Mm | Nn | Oo | Pp | Qq | Rr | Ss | Tt
                | Uu | Vv | Ww | Xx | Yy | Zz ;

Aa              = "a" | "A" ;
Bb              = "b" | "B" ;
Cc              = "c" | "C" ;
Dd              = "d" | "D" ;
Ee              = "e" | "E" ;
Ff              = "f" | "F" ;
Gg              = "g" | "G" ;
Hh              = "h" | "H" ;
Ii              = "i" | "I" ;
Jj              = "j" | "J" ;
Kk              = "k" | "K" ;
Ll              = "l" | "L" ;
Mm              = "m" | "M" ;
Nn              = "n" | "N" ;
Oo              = "o" | "O" ;
Pp              = "p" | "P" ;
Qq              = "q" | "Q" ;
Rr              = "r" | "R" ;
Ss              = "s" | "S" ;
Tt              = "t" | "T" ;
Uu              = "u" | "U" ;
```

```
Vv          = "v" | "V" ;
Ww          = "w" | "W" ;
Xx          = "x" | "X" ;
Yy          = "y" | "Y" ;
Zz          = "z" | "Z" ;
```