Objectives

A) Understand the basics of authentication methods supported by PostgreSQL
B) Understand how authentication protocols work over the wire to provide user authentication
C) Learn how to setup PostgreSQL to authenticate users using all the supported methods

We have a total of eleven topics to cover:

1. RADIUS (30)
2. PAM (30)
3. IDENT (10)
4. Peer (5)
5. Trust (10)
6. Password (5)
7. MD5 (5)
8. SCRAM (10)
9. Certificate (20)
10. Kerberos (30)
11. LDAP (20)

Total Estimated Time Required including questions if any = 175 minutes

Presenter

My name is Abbas, I have a Masters in Computer Engineering. I have spent most of my career in product development. I work as a Senior Architect at **EnterpriseDB**. My work highlights are as follows:

- Schema Cloning with support for parallelism using Background Workers
- Distributed Transactions (XA) Compliance for PostgreSQL using PgBouncer
- Oracle Compatible Packages for IBM DB2 : UTL_ENCODE, UTL_TCP, UTL_SMTP, UTL_MAIL
- HDFS_FDW, Mongo_FDW, MySQL FDW
- Postgres-XC

Email : abbas.butt@enterprisedb.com
Linkedin : https://pk.linkedin.com/in/abbasbutt
Blog : https://abbas-technical.blogspot.com

Access, Authentication, Authorization and Accounting

Suppose we have a services department in our company that provides the following paid services for personal use over the company wide intranet:

- Printing
- Scanning

In order for the co-workers to use the services they have to connect to the print server and submit their documents for printing in the queue.

In order for co-workers to use the scanner, they have to scan their documents on the scanner, the scanner will save the scanned document in the shared folder on the FTP server. The co-worker can than copy the scanned copy of the document from the shared folder.

Also suppose the following
- Executive Department of the company can use both the services
- Support department can use Printing Services only
- Research department can use Scanning services only.
- The rest of the departments of the company cannot use any of the services.

In order to implement the above scenario with in the company we will use the following strategy

People who do not work for the company cannot **access** the company's intranet hence they cannot use the services. If the company has wired network physical **access** to the company's switches is restricted. If the company has wireless access point, **access** can be restricted using passwords etc.

All the company employees can **access** the company's intranet. To verify which department a particular employee belongs to, each employee chooses a user-name and password that is shared with the services department. The services department creates users on its **authentication** server. Only the accounts of employees working in Executive, Support and Research department are created on the **authentication** server.

When the employee wants to print or scan he connects to the **authentication** server of the services department, and provides user-name and password. This **identifies** the employee and his department.

Once authenticated the authentication server knows which department the user belongs to and hence can decide which services he is **authorized** to use according to the rules defined above.

When the employee actually uses any of the services he is **authorized** to use these actions are recorded so that the employee can be **billed** accordingly. Each service that the employee uses has to be **accounted** for.

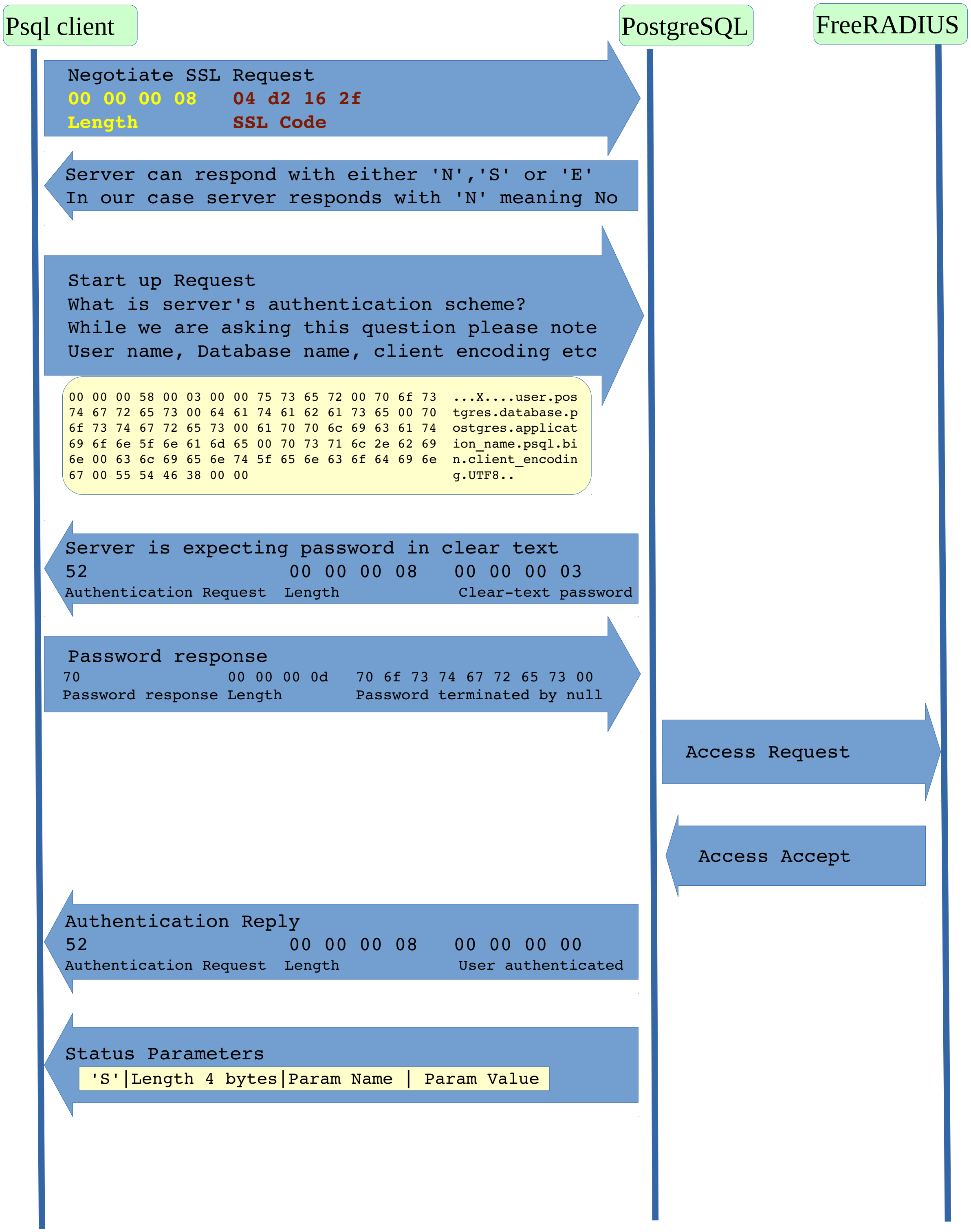
The main purpose of authentication is identification and the main purpose of authorization is to put a control on usage of resources. Accounting on the other hand makes sure that usage of a resource by an authorized user is recorded properly.

Collectively these three functions Authentication, Authorization & Accounting are called AAA. AAA is specified through various RFCs. Generic AAA architecture is specified in RFC 2903.

RADIUS is a protocol which is used to provide AAA on TCP/IP networks. RADIUS is an acronym for Remote Access Dial In User Service. RADIUS was part of an AAA solution delivered by Livingston Enterprises to Merit Network in 1991.
The RADIUS protocol was standardized using RFCs in 1997. RFC2865 covers the RADIUS protocol, and RFC2866 covers RADIUS accounting.

FreeRADIUS is an open source implementation of the RADIUS protocol and its extensions.

An overview of RADIUS protocol when used as authentication server for PostgreSQL



Negotiate SSL Request
00 00 00 08 Length
04 d2 16 2f SSL Code

Server can respond with either 'N','S' or 'E'
In our case server responds with 'N' meaning No

Start up Request
What is server's authentication scheme?
While we are asking this question please note
User name, Database name, client encoding etc

```
00 00 00 58 00 03 00 00 75 73 65 72 00 70 6f 73  ...X....user.pos
74 67 72 65 73 00 64 61 74 61 62 61 73 65 00 70  tgres.database.p
6f 73 74 67 72 65 73 00 61 70 70 6c 69 63 61 74  ostgres.applicat
69 6f 6e 5f 6e 61 6d 65 00 70 73 71 6c 2e 62 69  ion_name.psql.bi
6e 00 63 6c 69 65 6e 74 5f 65 6e 63 6f 64 69 6e  n.client_encodin
67 00 55 54 46 38 00 00                          g.UTF8..
```

Server is expecting password in clear text

```
52                       00 00 00 08    00 00 00 03
Authentication Request   Length         Clear-text password
```

Password response

```
70                          00 00 00 0d     70 6f 73 74 67 72 65 73 00
Password response Length                    Password terminated by null
```

Access Request

Access Accept

Authentication Reply

```
52                       00 00 00 08    00 00 00 00
Authentication Request   Length         User authenticated
```

Status Parameters

'S'|Length 4 bytes|Param Name | Param Value

Access Request Packet

```
01 16 00 42 16 e9 5f 9a 91 ab ba 93 68 d6 04 d8   ...B.._.....h...
51 bb ce 4b 06 06 00 00 00 08 01 0a 70 6f 73 74   Q..K........post
67 72 65 73 20 0c 70 6f 73 74 67 72 65 73 71 6c   gres .postgresql
02 12 db 7c ad 01 60 89 5a b8 00 62 b1 f8 5e 24   ...|..`.Z..b..^$
01 a4                                             ..

[ 1] Code: Access-Request (1)
[ 1] Packet identifier: 0x16 (22)
     RADIUS uses UDP by default. In case a packet is retransmitted
     this field remains the same. This allows the server to respond
     to requests by matching identifiers.
[ 2] Length: 66 - the length of complete packet
[16] Authenticator: 16e95f9a91abba9368d604d851bbce4b
     A random number not to be repeated again.
Attribute Value Pairs
  AVP: t=Service-Type(6)     : l= 6 : Authenticate Only(8)
  AVP: t=User-Name(1)        : l=10 : postgres
  AVP: t=NAS-Identifier(32): l=12 : postgresql
  AVP: t=User-Password(2)    : l=18 : Encrypted
       Generated by XOR-ing the password with the
       md5 hash of the shared secret & authenticator
       To verify the password all the server has to do is
       compute the md5 hash of the shared secret & authenticator
       and XOR with this byte stream. This will reveal the password
       because if a XOR b = c, then c XOR b = a
```

Access Accept Packet

```
02 16 00 2d 41 f5 6e c0 ba f0 c2 e2 99 73 5b 5b   ...^A.n......s[[
8f 4d 91 0a 12 19 48 65 6c 6c 6f 2c 20 70 6f 73   .M...JHello, pos
74 67 72 65 73 20 57 65 6c 63 6f 6d 65            tgres Welcome

[ 1] Code: Access-Accept (2)
[ 1] Packet identifier: 0x16 (22)
[ 2] Length: 45
[16] Authenticator: 41f56ec0baf0c2e299735b5b8f4d910a
     MD5(Code+ID+Length+RequestAuth+Attributes+Secret)
     where + denotes concatenation.
     Client can verify that the server has the secret
     by computing the same MD5 hash
Attribute Value Pairs
  AVP: t=Reply-Message(18) : l=25 : Hello, postgres Welcome
```

Configuration Steps

1. Install FreeRADIUS.

```
yum install freeradius
yum install freeradius-utils
```

2. Check Installation

```
radiusd -v
radiusd: FreeRADIUS Version 3.0.13, for host x86_64-redhat-linux-gnu,
built on Aug 23 2017 at 15:18:22
FreeRADIUS Version 3.0.13
Copyright (C) 1999-2017 The FreeRADIUS server project and contributors
```

3. Configure Shared Secret

WARNING : *Please use a shared secret which contains no capital letters*.

In the file `/etc/raddb/clients.conf` mention the shared secret in the sections

```
client localhost
          {
               ...
               secret         = macbookpro
               ...
          }
client localhost_ipv6
          {
               ipv6addr       = ::1
               secret         = macbookpro
          }
```

4. Configure Users

FreeRADIUS supports many different user stores: Text Files, SQL Databases & Directories.

For Example:

Users file
Linux System Users
LDAP Server
PostgreSQL server
etc

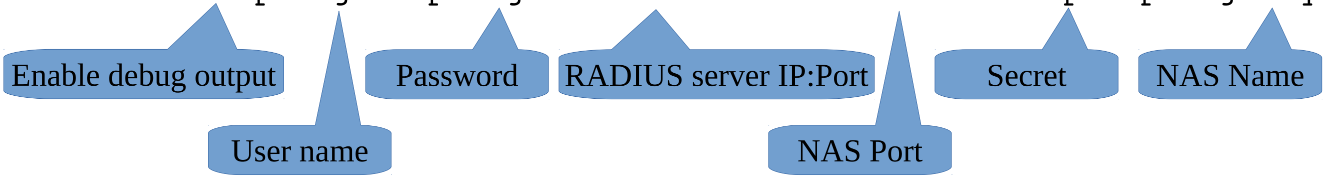
In our example we will use Users file

Edit the file `/etc/raddb/users` and add the following lines in it

```
postgres  Cleartext-Password := "postgres"
          Reply-Message = "Hello, %{User-Name} Welcome"
```

5. Check Configuration

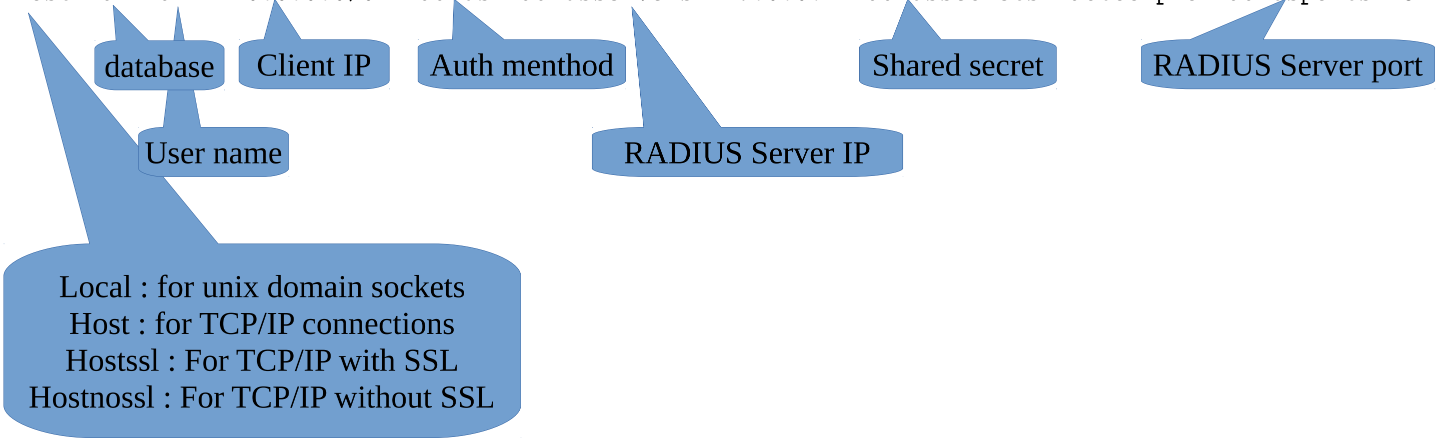
```
radtest -x postgres postgres 127.0.0.1:1812 0 macbookpro postgresql
```



```
Sent Access-Request Id 9 from 0.0.0.0:41103 to 127.0.0.1:1812 length 84
     User-Name = "postgres"
     User-Password = "postgres"
     NAS-IP-Address = 127.0.0.1
     NAS-Port = 0
     Message-Authenticator = 0x00
     Framed-Protocol = PPP
     Cleartext-Password = "postgres"
Received Access-Accept Id 9 from 127.0.0.1:1812 to 0.0.0.0:0 length 45
     Reply-Message = "Hello, postgres Welcome"
```

6. Configure pg_hba.conf

```
local all all              radius radiusservers=127.0.0.1 radiussecrets=macbookpro radiusports=1812
host  all all 127.0.0.1/32 radius radiusservers=127.0.0.1 radiussecrets=macbookpro radiusports=1812
host  all all    0.0.0.0/0  radius radiusservers=127.0.0.1 radiussecrets=macbookpro radiusports=1812
```



Local : for unix domain sockets
Host : for TCP/IP connections
Hostssl : For TCP/IP with SSL
Hostnossl : For TCP/IP without SSL

7. Reload configuration

```
pg_reload_conf();
```

8. Test authentication

```
./psql -p 6655 postgres -U postgres  -h 127.0.0.1
Password for user postgres:
psql.bin (10.0.2)
Type "help" for help.

postgres=> \q
```

9. Password Storing Methods in Users File:

FreeRADIUS supports the following methods of storing passwords in the Users file

#	Hash Type	AVP name
1	Unix-style crypted password	Crypt-Password
2	MD5 hashed password	MD5-Password
3	MD5 hashed password with a salt	SMD5-Password
4	SHA1 hashed password	SHA-Password
5	SHA1 hashed password with a salt	SSHA-Password
6	Windows NT hashed password	NT-Password
7	Windows Lan Manager (LM) password	LM-Password

Lets try MD5 hashed password for example:

9.1. Create a perl script with the following contents:

```
#! /usr/bin/perl -w
use strict;
use Digest::MD5;
use MIME::Base64;
unless($ARGV[0])
{
     print "Argument is missing\n";
     exit;
}
my $passwd = Digest::MD5->new;
$passwd->add($ARGV[0]);
print encode_base64($passwd->digest,'')."\n";
```

9.2. Save the script by the name `md5hash.pl`

9.3. `chmod +x md5hash.pl`

9.4. `./md5hash.pl postgres`

```
6KSGU4UeKMadBQZQj7J/xQ==
```

9.5. Edit the file `/etc/raddb/users`

```
postgres  MD5-Password := "6KSGU4UeKMadBQZQj7J/xQ=="
          Reply-Message = "Hello, %{User-Name} Welcome"
```

9.6. Restart the FreeRADIUS server

9.7. Test authentication

```
./psql -p 6655 postgres -U postgres  -h 127.0.0.1
Password for user postgres:
```

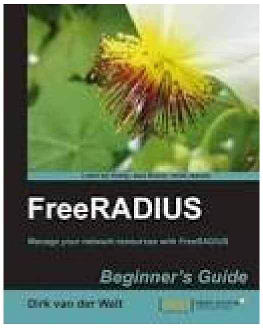
```
psql.bin (10.0.2)
Type "help" for help.

postgres=> \q
```

9.8 Check the relevant content in the server log file

```
(1)   Auth-Type PAP {
(1) pap: Login attempt with password
(1) pap: Comparing with "known-good" MD5-Password
(1) pap: User authenticated successfully
(1)     [pap] = ok
(1)   } # Auth-Type PAP = ok
```

For more information please consult this book:

FreeRADIUS Beginner's Guide
by Dirk Van Der Walt

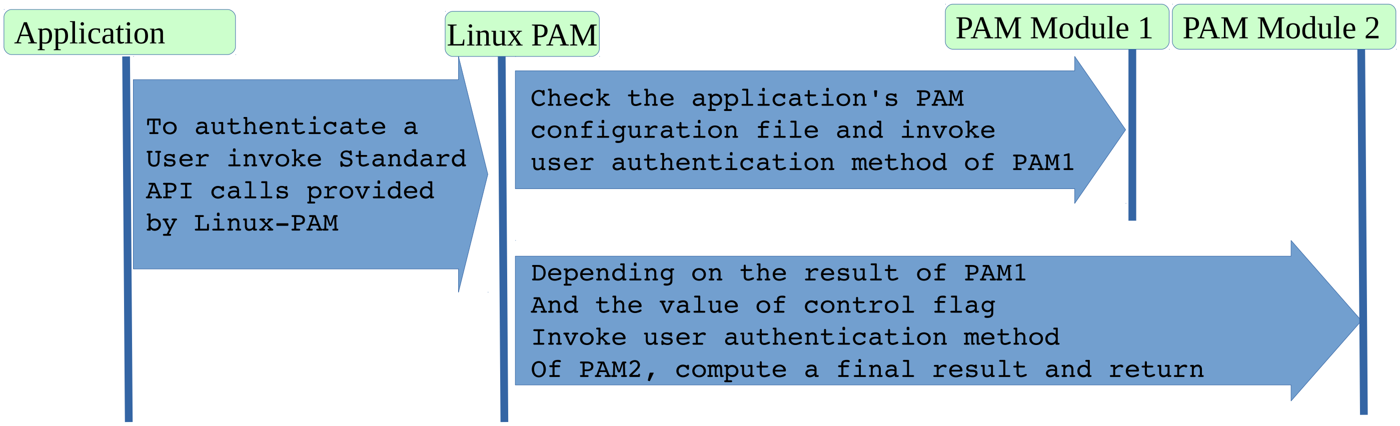


What is PAM

Any software system that needs to authenticate users has to choose what authentication methods the system is going to support. Suppose that it was decided that the system will support authentication using the password file and the software got released. At any latter time the format of the password file can be changed for example to include passwords in MD5 format. Also any new authentication mechanism can get introduced after the software release and organizations might want to adopt the new authentication system. In both the cases the software system will have to be modified, recompiled and redistributed.

Instead software systems needing authentication should use a standard library. Each library providing support for any standard authentication scheme should expose a standard set of interface functions that the software system can invoke. In order to configure which authentication method or methods would the software system try all the user should do is edit a configuration file.

This system is know as Pluggable Authentication Modules PAM. In PAM each library providing support for an authentication method is called a **module**. PAM was developed in 1995 by Sun Microsystems and was standardized in 1997 by Open Group. PAM is supported by all major operating systems for example Linux-PAM. In Linux-PAM the program that uses PAM will make calls to the Linux-PAM library which will in turn invoke functions provided by the PAM module.


Application
Linux PAM
PAM Module 1
PAM Module 2
To authenticate a User invoke Standard API calls provided by Linux-PAM
Check the application's PAM configuration file and invoke user authentication method of PAM1
Depending on the result of PAM1 And the value of control flag Invoke user authentication method Of PAM2, compute a final result and return

A major advantage of this architecture is that on a single system different programs can use different authentication schemes. Each program's configuration file will specify a different set of PAM modules to use.

The configuration file for some software systems can list more than one PAM modules to try, and each is tried in the order listed. This list of modules to try for authentication is called a stack. If the user fails to authenticate using the first PAM module which provides support for say /etc/passwd file, then PAM will try the next module listed, which can attempt authentication using LDAP for example.

In case where the program specifies more than one PAM modules to try in the configuration file, the modules are invoked one by one in the order listed in the stack. Each module can either return success or failure. There are many possibilities that the program can opt for before declaring success or failure to the user. For example the program can declare success to the user only when all the modules return success or when at least one of the modules declares success. The results of all the modules have to be combined into a single result. This accumulation is controlled by a flag provided for each module in the configuration file.

If a program's PAM configuration file is missing it uses a configuration file named “other”. This file should normally deny all access.

PAM modules are generally stored in /lib64/security directory and all PAM module names start with pam_. All PAM modules are shared objects i.e. so files. Modules can be put any where provided their absolute path is specified in the PAM configuration file.

PAM modules can provide support for
Authentication using "**auth**" modules
Authorization using "**account**" modules
Session Management using "**session**" modules &
Password Management using "**password**" modules. Password modules implement policies for acceptable passwords.

Control Flag Options

Sufficient

This control-flag means that if the module passes, that is enough and the remaining modules in the "auth" context will be ignored. However, if the module fails, that doesn't mean an overall result of failure. If a subsequent sufficient passes then the overall result will be success.

Required

This control-flag means that this modules must succeed before access is granted by PAM. If any required module fails, the remaining required modules will be tried before declaring overall failure.

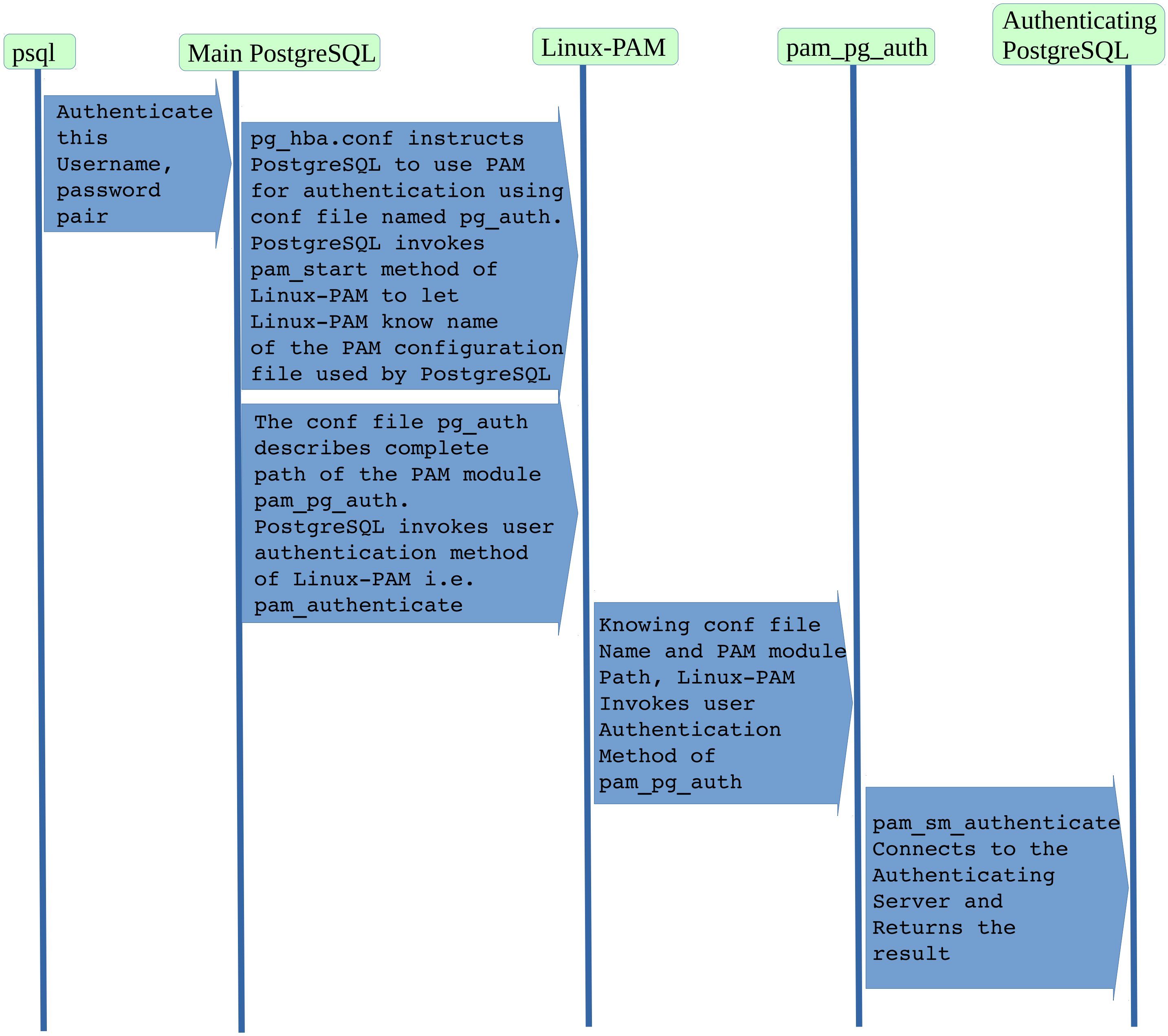
Requisite

This control-flag is the same as required flag, however when the module fails no further modules are tried.

Optional

This control-flag means that the success or failure of that module has no effect. It is used for session modules only.

A sample PAM module pam_pg_auth



psql

Main PostgreSQL

Linux-PAM

pam_pg_auth

Authenticating PostgreSQL

Authenticate this Username, password pair

pg_hba.conf instructs PostgreSQL to use PAM for authentication using conf file named pg_auth. PostgreSQL invokes pam_start method of Linux-PAM to let Linux-PAM know name of the PAM configuration file used by PostgreSQL

The conf file pg_auth describes complete path of the PAM module pam_pg_auth. PostgreSQL invokes user authentication method of Linux-PAM i.e. pam_authenticate

Knowing conf file Name and PAM module Path, Linux-PAM Invokes user Authentication Method of pam_pg_auth

pam_sm_authenticate Connects to the Authenticating Server and Returns the result

Simple PAM Module pam_pg_auth

```
/*
 * pam_pg_auth
 *
 * Authenticate a PG user by contacting another PG server
 * using the auth method specified in the argument
 */

#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include "libpq-fe.h"

/*
 * This is recommended in the module developer's guide
 */

#define PAM_SM_AUTH
#define PAM_SM_ACCOUNT
#define PAM_SM_SESSION
#define PAM_SM_PASSWORD

#include <security/pam_modules.h>
#include <security/_pam_macros.h>


#define DEFAULT_USER    "nobody"
#define DEFAULT_LEN    1024
#define DEFAULT_LOG_FILE  "/tmp/pam/pam_pg_auth.txt"
#define DEFAULT_CONF_FILE  "/tmp/pg_auth.conf"

typedef enum
{
  TRUST = 0,
  SCRAM_SHA_256,
  MD5,
  PASSWORD,
  GSSAPI,
  IDENT,
  PEER,
  LDAP,
  RADIUS,
  CERTIFICATE
}pg_auth_type;


typedef struct
{
  char    con_str[DEFAULT_LEN + 1];
}pg_auth_conf;
```

```
int pam_parse_args(int argc, const char **argv);
int pam_parse_conf(void);
int connect_auth_server(void);

char log_file_name[DEFAULT_LEN + 1];
char auth_name[DEFAULT_LEN + 1];
int params_parsed = 0;
int conf_parsed = 0;
pg_auth_type auth_type = 0;
pg_auth_conf auth_conf;

/* connection_string=host=localhost port=8888 dbname=postgres connect_timeout=10 */
int pam_parse_conf()
{
  FILE *fp = NULL;
  char line[DEFAULT_LEN + 1];
  char key[DEFAULT_LEN + 1];
  char *sep;
  int key_len;
  int c = 0;
  char *cr;

  if (conf_parsed)
    return 0;

  conf_parsed = 1;

  memset(auth_conf.con_str, 0, DEFAULT_LEN + 1);

  fp = fopen(DEFAULT_CONF_FILE, "r");
  if (fp == NULL)
    return 0;

  while (1)
  {
    memset(line, 0, DEFAULT_LEN + 1);
    memset(key, 0, DEFAULT_LEN + 1);

    cr = fgets(line, DEFAULT_LEN, fp);
    if (cr == NULL)
      break;

    sep = strchr(line , '=');
    if (sep != NULL)
    {
      key_len = sep - line;
      sep = sep + 1;      /* point sep to value */
      memcpy(key, line, key_len);
    }
    c++;
    if (strcmp(key, "connection_string") == 0)
      strcpy(auth_conf.con_str, sep);
  }
  return c;
}

int pam_parse_args(int argc, const char **argv)
{
  int i;

  if (params_parsed)
    return 0;

  params_parsed = 1;

  strcpy(log_file_name, DEFAULT_LOG_FILE);
  auth_type = TRUST;

  for (i = 0; i < argc; i++)
  {
    if (!strncasecmp(argv[i],"log_file=", 9))
    {
      memset(log_file_name, 0, DEFAULT_LEN + 1);
      strcpy(log_file_name, (*argv) + 9);
    }
    if (!strncasecmp(argv[i],"auth_type=", 10))
    {
      memset(auth_name, 0, DEFAULT_LEN + 1);
      strcpy(auth_name, (*argv) + 10);

      if (strcmp(auth_name , "trust") == 0)
        auth_type = TRUST;
      else if (strcmp(auth_name , "scram-sha-256") == 0)
        auth_type = SCRAM_SHA_256;
      else if (strcmp(auth_name , "md5") == 0)
        auth_type = MD5;
      else if (strcmp(auth_name , "password") == 0)
        auth_type = PASSWORD;
      else if (strcmp(auth_name , "gssapi") == 0)
        auth_type = GSSAPI;
      else if (strcmp(auth_name , "ident") == 0)
        auth_type = IDENT;
      else if (strcmp(auth_name , "peer") == 0)
        auth_type = PEER;
      else if (strcmp(auth_name , "ldap") == 0)
        auth_type = LDAP;
      else if (strcmp(auth_name , "radius") == 0)
        auth_type = RADIUS;
      else if (strcmp(auth_name , "certificate") == 0)
        auth_type = CERTIFICATE;
    }
  }

  return argc;
}
```

```
int connect_auth_server()
{
  PGconn *conn;
  FILE *fp;

  fp = fopen(log_file_name, "a+");

  switch (auth_type)
  {
  case TRUST:
    break;
  case SCRAM_SHA_256:
    break;
  case MD5:
    break;
  case PASSWORD:
    break;
  case GSSAPI:
    break;
  case IDENT:
    break;
  case PEER:
    break;
  case LDAP:
    break;
  case RADIUS:
    break;
  case CERTIFICATE:
    break;
  }

  conn = PQconnectdb(auth_conf.con_str);
  if (PQstatus(conn) != CONNECTION_OK)
  {
    if (fp != NULL)
    {
      fprintf(fp, "\n[%s][%d] Connection with auth server failed, reason [%d], [%s]",
                __FUNCTION__, __LINE__, PQstatus(conn), PQerrorMessage(conn));
      fflush(fp);
    }

    return 0;
  }
  PQfinish(conn);

  return 1;
}
```

```
PAM_EXTERN int pam_sm_authenticate(pam_handle_t *pamh,int flags,int argcc,const char **argv)
{
  FILE *fp;
  int retval;
  const char *user=NULL;
  pam_parse_conf();
  pam_parse_args(argc, argv);
  fp = fopen(log_file_name, "a+");
  if (fp != NULL)
  {
    fprintf(fp, "\n[%s][%d] Passed parameters flags[%02X] argc[%d]", __FUNCTION__, __LINE__,
flags, argc);
    fflush(fp);
  }
  /*
   * authentication requires we know who the user wants to be
   */
  retval = pam_get_user(pamh, &user, NULL);
  if (retval != PAM_SUCCESS)
  {
    if (fp != NULL)
    {
      fprintf(fp, "\n[%s][%d] pam_get_user falied with error[%s]", __FUNCTION__, __LINE__,
pam_strerror(pamh,retval));
      fflush(fp);
    }
    return PAM_CRED_INSUFFICIENT;
  }
  if (user == NULL || *user == '\0')
  {
    if (fp != NULL)
    {
      fprintf(fp, "\n[%s][%d] empty username", __FUNCTION__, __LINE__);
      fflush(fp);
    }
    pam_set_item(pamh, PAM_USER, (const void *) DEFAULT_USER);
    return PAM_CRED_INSUFFICIENT;
  }
  else
  {
    pam_set_item(pamh, PAM_USER, (const void *) user);
    if (fp != NULL)
    {
      fprintf(fp, "\n[%s][%d] username[%s]", __FUNCTION__, __LINE__, user);
      fflush(fp);
    }
    retval = connect_auth_server();
    if (retval != 1)
      return PAM_AUTH_ERR;

    return PAM_SUCCESS;
  }
  user = NULL;
  return PAM_SUCCESS;
}
```

```
PAM_EXTERN int pam_sm_setcred(pam_handle_t *pamh,int flags,int argc ,const char **argv)
{
  FILE *fp;

  pam_parse_conf();
  pam_parse_args(argc, argv);

  fp = fopen(log_file_name, "a+");
  if (fp != NULL)
  {
    fprintf(fp, "\n[%s][%d] Passed parameters flags[%02X] argc[%d]", __FUNCTION__, __LINE__,
flags, argc);
    fflush(fp);
  }

  return PAM_SUCCESS;
}

/* --- account management functions --- */
PAM_EXTERN int pam_sm_acct_mgmt(pam_handle_t *pamh,int flags,int argc ,const char **argv)
{
  FILE *fp;

  pam_parse_conf();
  pam_parse_args(argc, argv);

  fp = fopen(log_file_name, "a+");
  if (fp != NULL)
  {
    fprintf(fp, "\n[%s][%d] Passed parameters flags[%02X] argc[%d]", __FUNCTION__, __LINE__,
flags, argc);
    fflush(fp);
  }

  return PAM_SUCCESS;
}
```

```
/* --- password management --- */
PAM_EXTERN int pam_sm_chauthtok(pam_handle_t *pamh,int flags,int argc ,const char **argv)
{
  FILE *fp;

  pam_parse_conf();
  pam_parse_args(argc, argv);

  fp = fopen(log_file_name, "a+");
  if (fp != NULL)
  {
    fprintf(fp, "\n[%s][%d] Passed parameters flags[%02X] argc[%d]", __FUNCTION__, __LINE__,
flags, argc);
    fflush(fp);
  }

  return PAM_SUCCESS;
}

/* --- session management --- */
PAM_EXTERN int pam_sm_open_session(pam_handle_t *pamh,int flags,int argc ,const char **argv)
{
  int i;
  int retval;
  const char *user=NULL;
  FILE *fp;

  pam_parse_conf();
  pam_parse_args(argc, argv);

  fp = fopen(log_file_name, "a+");
  if (fp != NULL)
  {
    fprintf(fp, "\n[%s][%d] Passed parameters flags[%02X] argc[%d]", __FUNCTION__, __LINE__,
flags, argc);
    fflush(fp);
  }

  retval = pam_get_user(pamh, &user, NULL);
  if (retval != PAM_SUCCESS)
  {
    if (fp != NULL)
    {
      fprintf(fp, "\n[%s][%d] pam_get_user falied with error[%s]", __FUNCTION__, __LINE__,
pam_strerror(pamh,retval));
      fflush(fp);
    }

    return PAM_USER_UNKNOWN;
  }

  if (user == NULL || *user == '\0')
  {
    if (fp != NULL)
    {
      fprintf(fp, "\n[%s][%d] empty username", __FUNCTION__, __LINE__);
      fflush(fp);
    }
    return PAM_USER_UNKNOWN;
  }

  if (fp != NULL)
  {
    fprintf(fp, "\n[%s][%d] username[%s]", __FUNCTION__, __LINE__, user);
    fflush(fp);
  }

  for (i = 0; i < argc; i++)
  {
    if (fp != NULL)
    {
      fprintf(fp, "\n[%s][%d] Param #[%d] Param Value[%s]", __FUNCTION__, __LINE__, i,
argv[i]);
      fflush(fp);
    }
  }
  return PAM_SUCCESS;
}

PAM_EXTERN int pam_sm_close_session(pam_handle_t *pamh,int flags,int argc ,const char **argv)
{
  FILE *fp;

  pam_parse_conf();
  pam_parse_args(argc, argv);

  fp = fopen(log_file_name, "a+");
  if (fp != NULL)
  {
    fprintf(fp, "\n[%s][%d] Passed parameters flags[%02X] argc[%d]", __FUNCTION__, __LINE__,
flags, argc);
    fflush(fp);
  }

  return PAM_SUCCESS;
}
```

Deploying the pam_pg_auth module

1. Build the .so file pam_pg_auth.so
2. In /etc/pam.d create a file called **pg_auth** containing

```
auth     sufficient /home/abbas/pam_pg_auth/pam_pg_auth.so log_file=/tmp/pam_auth.txt auth_type=trust
account  sufficient /home/abbas/pam_pg_auth/pam_pg_auth.so log_file=/tmp/pam_auth.txt auth_type=trust
password sufficient /home/abbas/pam_pg_auth/pam_pg_auth.so log_file=/tmp/pam_auth.txt auth_type=trust
session  sufficient /home/abbas/pam_pg_auth/pam_pg_auth.so log_file=/tmp/pam_auth.txt auth_type=trust
```

3. Create the configuration file for the module

```
vim /tmp/pg_auth.conf

connection_string=host=localhost port=8888 dbname=postgres connect_timeout=10
```

4. Build PostgreSQL with PAM support

```
git clone git://git.postgresql.org/git/postgresql.git
git checkout REL_10_STABLE
sudo yum install readline*
sudo yum install zlib*
./configure --prefix=/usr/local/pg10_pam --with-pam --enable-debug CFLAGS="-O0 -g"
make && make install

./configure --prefix=/usr/local/pg10_auth --enable-debug CFLAGS="-O0 -g"
make && make install

cd /usr/local/pg10_pam/bin/
./initdb -D ../data

cd /usr/local/pg10_auth/bin
./initdb -D ../data
```

5. Modify the pg_hba.conf file of the main PostgreSQL server (`/usr/local/pg10_pam`) as follows

```
local all  all                     pam   pamservice=pg_auth
host  all  all  127.0.0.1/32       pam   pamservice=pg_auth
host  all  all  ::1/128            pam   pamservice=pg_auth
```

6. Modify the pg_hba.conf file of the authenticating PostgreSQL server (`/usr/local/pg10_auth`) as follows

```
local all  all                     trust
host  all  all  127.0.0.1/32       trust
host  all  all  ::1/128            trust
```

7. Start both the servers

Main PostgreSQL server

```
./postgres -D ../data -p 9999 -d 2
```

Authenticating PostgreSQL server

```
./postgres -D ../data -p 8888 -d 2
```

8. Create the user in the Main Server

```
./createuser -d -l -P -r -s -h 127.0.0.1 -p 9999 harry
Password : test
```

9. Test PAM Authentication

```
 ./psql -h 127.0.0.1 -p 9999 -U harry postgres
psql (10.3)
Type "help" for help.

postgres=#
```

10. Check the PAM module's log file

```
cat /tmp/pam/pam_auth.txt

[pam_sm_authenticate][196] Passed parameters flags[00] argc[2]
[pam_sm_authenticate][229] username[harry]
[pam_sm_acct_mgmt][276] Passed parameters flags[00] argc[2]
```

Note that PostgreSQL does not use “session” or “password” functions of the PAM module.

11. Check the Server log files

Main Server

```
2018-04-11 10:13:48.496 PKT [43754] LOG:  connection received:
host=127.0.0.1 port=55458
2018-04-11 10:13:48.509 PKT [43754] LOG:  connection authorized:
user=harry database=postgres
```

Authentication Server

```
2018-04-11 10:13:48.503 PKT [43755] LOG:  connection received: host=::1
port=42294
2018-04-11 10:13:48.504 PKT [43755] LOG:  connection authorized:
user=abbasbutt database=postgres
```

All sorts of combinations are possible with PAM, here user harry gets authenticated if authentication server can be connected with default username.

Note : Work is under way to support other authentication methods in pam_pg_auth.

Overview of IDENT protocol

Identification protocol is defined by RFC 1413. It provides an option to determine the identity of the user initiating a particular TCP connection. Given a TCP source and destination port number pair, the IDENT server returns a character string which identifies the owner of that connection on the IDENT server's system. PostgreSQL checks whether this user is an allowed database user.

IDENT Server is supposed to be run on the client machine i.e. the machine where psql is running. The IP address of the IDENT server is the same from where the psql connects with the PostgreSQL server. The TCP port is standard 113.

PostgreSQL sends Query to the IDENT server

```
    39422,7777
```

where 39422 is the source TCP port used by psql while connecting with the PostgreSQL server
and 7777 is the destination TCP port used by psql while connecting with the PostgreSQL server
i.e the port on which PostgreSQL server is listening.

PostgreSQL asks the IDENT server:
What user initiated the connection that goes out of IDENT server's port 39422 and connects to port 7777 on my machine?

The Server responds with

```
    39422 , 7777 : USERID : Linux :abbasbutt
```

where 39422 is the port being used by psql client running on the IDENT server,
7777 is the port on IDENT's client i.e. PostgreSQL server.
Response Type is USERID meaning that the response is the name of operating system username
Linux is the name of the operating system, abbasbutt is the username.
Response could also be of the form

```
    ERROR : NO-USER
```

PostgreSQL compares the username provided by IDENT server with the username provided by psql. If both are equal then PostgreSQL checks whether the username provided is a valid database user or not.

Installing and Configuring the IDENT server

Note that the server has to be installed on the machine where psql is running.

```
sudo yum install authd
sudo yum install xinetd

sudo vim /etc/xinetd.d/auth

service auth
{
        disable = no
        socket_type = stream
        wait = no
        user = ident
        cps = 4096 10
        instances = UNLIMITED
        server = /usr/sbin/in.authd
        server_args = -t60 --xerror --os
}

sudo service xinetd restart
```

Configuring & Testing PostgreSQL server

Modify the PostgreSQL server's pg_hba.conf as follows

```
     local  all  all                   trust
     host   all  all   127.0.0.1/32    ident
     host   all  all   ::1/128         ident
```

Run the server and test the configuration as follows:

```
     ./psql -h 127.0.0.1 -p 7777 -U abbasbutt postgres
     psql (10.3)
     Type "help" for help.

     postgres=#
```

Test the case when the username provided by IDENT server and psql are different

```
whoami
abbasbutt

./createuser -d -l -P -r -s -h 127.0.0.1 -p 7777 tom
Enter password for new role:
Enter it again:

./psql -h 127.0.0.1 -p 7777 -U tom postgres
psql: FATAL:  Ident authentication failed for user "tom"
```

Peer Authentication

Peer Authentication is supported for unix domain sockets only. It is not applicable to TCP/IP connections to the server. This method works by obtaining the client's operating system user name from the kernel and using it as the allowed database user name.

To configure the server to use Peer Authentication pg_hba.conf is modified as follows:

```
local   all     all                   peer
host    all     all     127.0.0.1/32  md5
host    all     all     ::1/128       md5
```

To configure the server to use Peer Authentication pg_hba.conf is modified as follows:

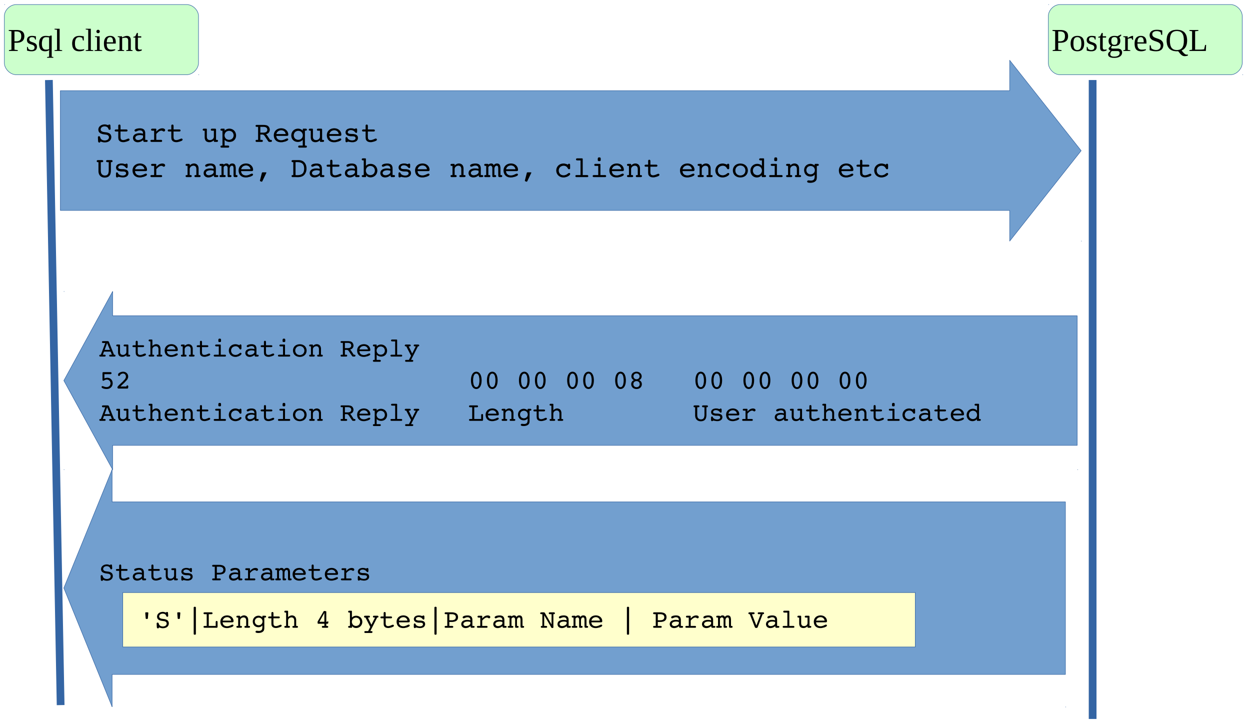
```
[abbasbutt@ublnetbanking bin]$ whoami
abbasbutt
[abbasbutt@ublnetbanking bin]$ ./psql -p 7777 -U abbasbutt postgres
psql (10.3)
Type "help" for help.

postgres=# \q
[abbasbutt@ublnetbanking bin]$ ./psql -p 7777 -U xyz postgres
psql: FATAL:  Peer authentication failed for user "xyz"
```

Trust Authentication

In trust authentication the server does not ask client for any password. Only the username is checked. The entries in pg_hba_conf are as follows

```
local    all      all                        trust
host     all      all      127.0.0.1/32      trust
host     all      all      ::1/128           trust

./psql -h 127.0.0.1 -p 7777 -U abbasbutt postgres
psql (10.3)
Type "help" for help.

postgres=#
```

The protocol is as follows:



```
Start up Request
User name, Database name, client encoding etc

Authentication Reply
52                    00 00 00 08    00 00 00 00
Authentication Reply  Length         User authenticated

Status Parameters
'S'|Length 4 bytes|Param Name | Param Value
```

Start up Packet

```
0000    00 00 00 55 00 03 00 00 75 73 65 72 00 61 62 62   ...U....user.abb
0010    61 73 62 75 74 74 00 64 61 74 61 62 61 73 65 00   asbutt.database.
0020    70 6f 73 74 67 72 65 73 00 61 70 70 6c 69 63 61   postgres.applica
0030    74 69 6f 6e 5f 6e 61 6d 65 00 70 73 71 6c 00 63   tion_name.psql.c
0040    6c 69 65 6e 74 5f 65 6e 63 6f 64 69 6e 67 00 55   lient_encoding.U
0050    54 46 38 00 00                                    TF8..
```

```
[ 4] Length (85)
[ 4] Protocol Version (3.0)
Followed by null terminated strings of Param name and Param value pairs.
```

Authentication Reply & Status Parameters

```
0000    52 00 00 00 08 00 00 00 00 53 00 00 00 1a 61 70   R........S....ap
0010    70 6c 69 63 61 74 69 6f 6e 5f 6e 61 6d 65 00 70   plication_name.p
0020    73 71 6c 00 53 00 00 00 19 63 6c 69 65 6e 74 5f   sql.S....client_
0030    65 6e 63 6f 64 69 6e 67 00 55 54 46 38 00 53 00   encoding.UTF8.S.
0040    00 00 17 44 61 74 65 53 74 79 6c 65 00 49 53 4f   ...DateStyle.ISO
0050    2c 20 4d 44 59 00 53 00 00 00 19 69 6e 74 65 67   , MDY.S....integ
0060    65 72 5f 64 61 74 65 74 69 6d 65 73 00 6f 6e 00   er_datetimes.on.
0070    53 00 00 00 1b 49 6e 74 65 72 76 61 6c 53 74 79   S....IntervalSty
0080    6c 65 00 70 6f 73 74 67 72 65 73 00 53 00 00 00   le.postgres.S...
0090    14 69 73 5f 73 75 70 65 72 75 73 65 72 00 6f 6e   .is_superuser.on
00a0    00 53 00 00 00 19 73 65 72 76 65 72 5f 65 6e 63   .S....server_enc
00b0    6f 64 69 6e 67 00 55 54 46 38 00 53 00 00 00 18   oding.UTF8.S....
00c0    73 65 72 76 65 72 5f 76 65 72 73 69 6f 6e 00 31   server_version.1
00d0    30 2e 33 00 53 00 00 00 24 73 65 73 73 69 6f 6e   0.3.S...$session
00e0    5f 61 75 74 68 6f 72 69 7a 61 74 69 6f 6e 00 61   _authorization.a
00f0    62 62 61 73 62 75 74 74 00 53 00 00 00 23 73 74   bbasbutt.S...#st
0100    61 6e 64 61 72 64 5f 63 6f 6e 66 6f 72 6d 69 6e   andard_conformin
0110    67 5f 73 74 72 69 6e 67 73 00 6f 6e 00 53 00 00   g_strings.on.S..
0120    00 1a 54 69 6d 65 5a 6f 6e 65 00 41 73 69 61 2f   ..TimeZone.Asia/
0130    4b 61 72 61 63 68 69 00 4b 00 00 00 0c 00 00 c7   Karachi.K.......
0140    0c 4b 79 a0 47 5a 00 00 00 05 49                  .Ky.GZ....I
```

```
[ 1] Authentication Reply (0x52)
[ 4] Length (8)
[ 4] User Authenticated
```

```
Followed by Status Parameters in the format
```

```
'S'|Length 4 bytes|Param Name | Param Value
```

Password Authentication

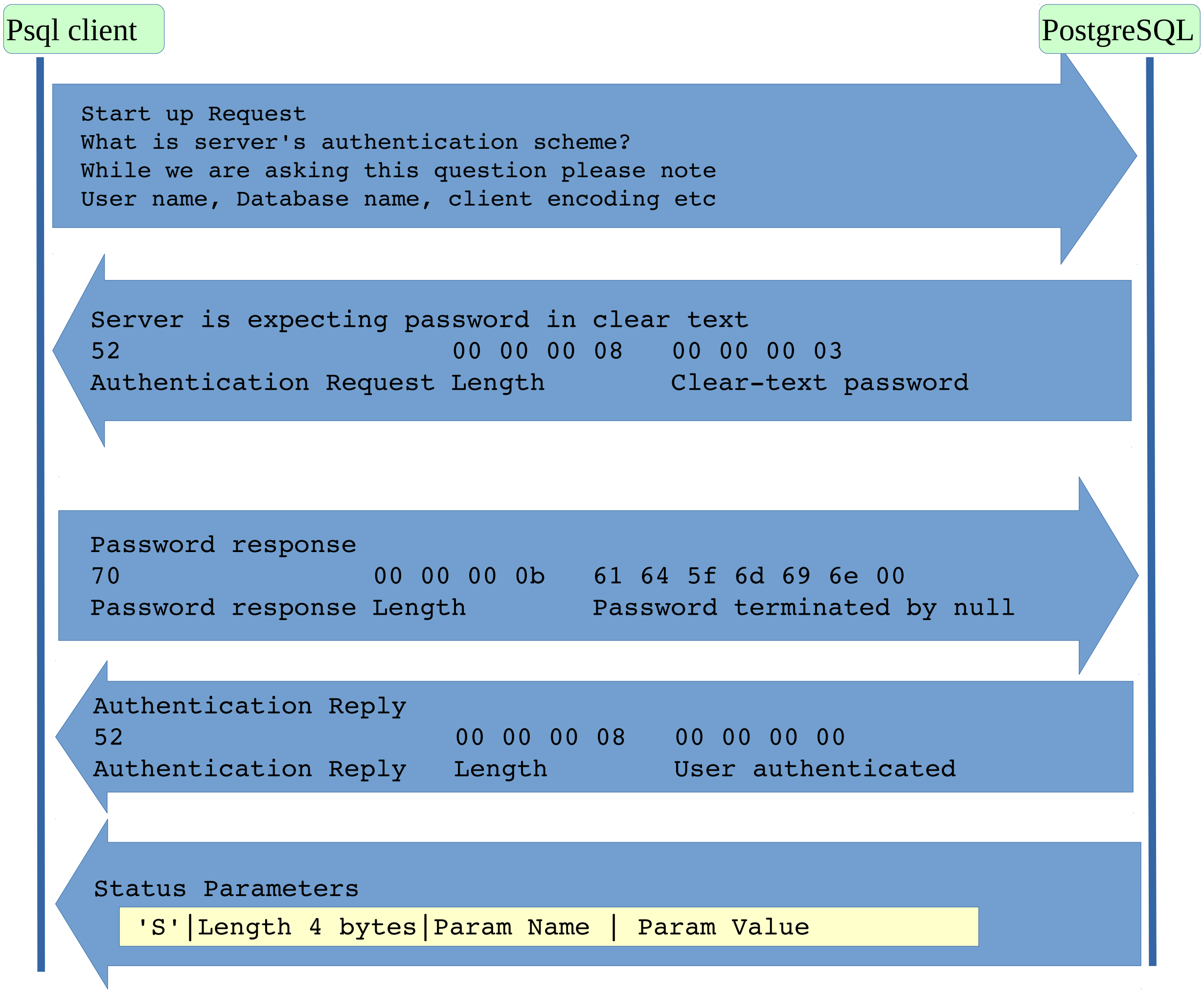
In password authentication the server asks for password in clear text. The entries in pg_hba_conf are as follows

```
local    all      all                        trust
host     all      all      127.0.0.1/32     password
host     all      all      ::1/128          password
```

Using trust authentication create a user first

```
     ./createuser -d -l -P -r -s -p 7777 admin
     Enter password for new role: ad_min
     Enter it again: ad_min

     ./psql -h 127.0.0.1 -p 7777 -U admin postgres
     Password for user admin: ad_min
     psql (10.3)
     Type "help" for help .
     postgres=#
```

The protocol is as follows:



```
Start up Request
What is server's authentication scheme?
While we are asking this question please note
User name, Database name, client encoding etc
```

```
Server is expecting password in clear text
52                       00 00 00 08    00 00 00 03
Authentication Request Length           Clear-text password
```

```
Password response
70                  00 00 00 0b    61 64 5f 6d 69 6e 00
Password response Length           Password terminated by null
```

```
Authentication Reply
52                       00 00 00 08    00 00 00 00
Authentication Reply    Length          User authenticated
```

```
Status Parameters
   'S'|Length 4 bytes|Param Name | Param Value
```

MD5 Password Authentication

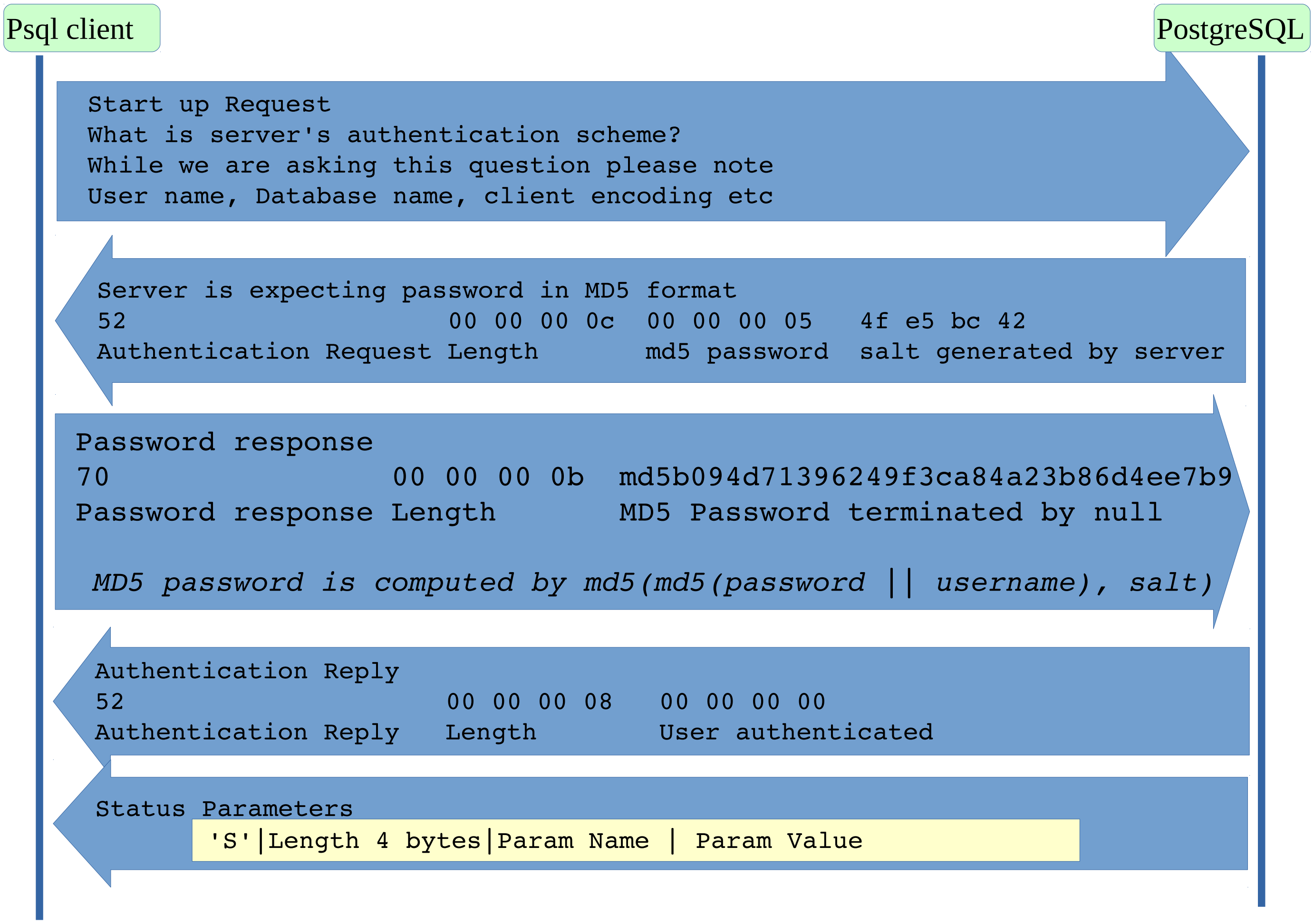
In md5 password authentication the server asks for password in md5 format. The entries in pg_hba_conf are as follows

```
local   all     all                     trust
host    all     all     127.0.0.1/32    md5
host    all     all     ::1/128         md5
```

Using trust authentication create a user first

```
     ./createuser -d -l -P -r -s -p 7777 admin
     Enter password for new role: ad_min
     Enter it again: ad_min

     ./psql -h 127.0.0.1 -p 7777 -U admin postgres
     Password for user admin: ad_min
     psql (10.3)
     Type "help" for help.
     postgres=#
```

The protocol is as follows:



Psql client

PostgreSQL

```
Start up Request
What is server's authentication scheme?
While we are asking this question please note
User name, Database name, client encoding etc
```

```
Server is expecting password in MD5 format
52                      00 00 00 0c   00 00 00 05   4f e5 bc 42
Authentication Request Length         md5 password  salt generated by server
```

```
Password response
70                   00 00 00 0b  md5b094d71396249f3ca84a23b86d4ee7b9
Password response Length          MD5 Password terminated by null

 MD5 password is computed by md5(md5(password || username), salt)
```

```
Authentication Reply
52                      00 00 00 08   00 00 00 00
Authentication Reply    Length        User authenticated
```

```
Status Parameters
         'S'|Length 4 bytes|Param Name | Param Value
```

What is SASL & SCRAM-SHA-256

Simple Authentication and Security Layer (SASL) is specified in RFC 4422.

"*The Simple Authentication and Security Layer (SASL) is a framework for providing authentication and data security services in connection-oriented protocols via replaceable mechanisms*."

In SASL the client and server negotiate a common SASL mechanism that they will use for authentication. The server provides a list of supported authentication mechanisms to the client. The client can decide which authentication mechanism it is going to use. The authentication then takes place using the mechanism both client and server agree to use. The client and server then keep exchanging authentication data encapsulated in SASL messages until the authentication successfully completes, fails, or is aborted.

SCRAM-SHA-256 is one of the authentication mechanisms supported by SASL. **Salted Challenge Response Authentication Mechanism** is specified by RFC 5802 & 7677. **Secure Hashing Algorithm** 256 always generates a 32 byte hash.

SCRAM Attributes

Each SCRAM attribute has a one letter name. The attributes used by PostgreSQL are described as follows:

n : username

r : random nonce

c : channel binding data

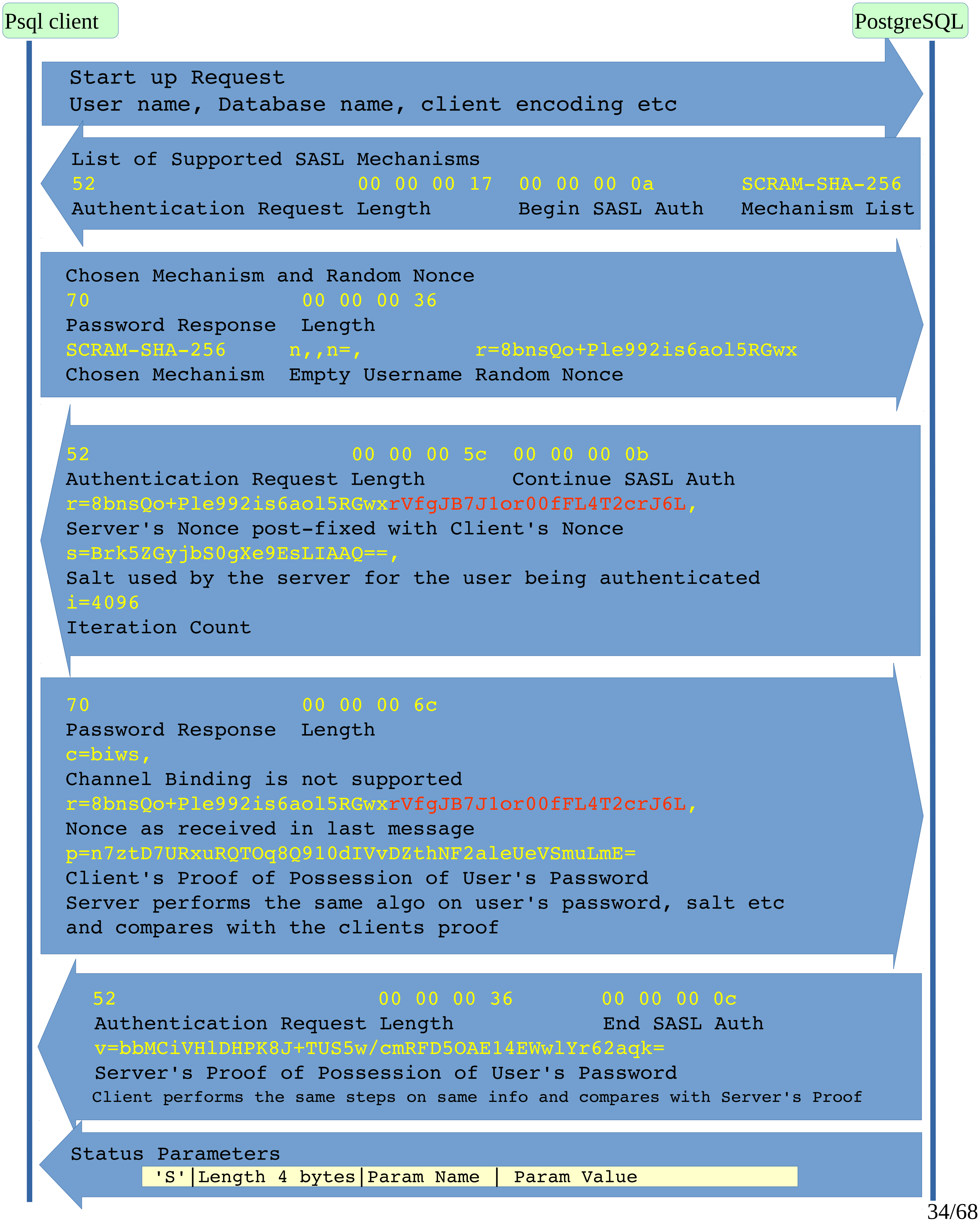
s : salt used by the server for the user being authenticated

i : iteration count

p : base-64 encoded Client's Proof

v : base-64 encoded Server's Proof

SCRAM Authentication



```
Start up Request
User name, Database name, client encoding etc
```

```
List of Supported SASL Mechanisms
52                     00 00 00 17  00 00 00 0a      SCRAM-SHA-256
Authentication Request Length       Begin SASL Auth  Mechanism List
```

```
Chosen Mechanism and Random Nonce
70                 00 00 00 36
Password Response  Length
SCRAM-SHA-256     n,,n=,         r=8bnsQo+Ple992is6aol5RGwx
Chosen Mechanism  Empty Username Random Nonce
```

```
52                     00 00 00 5c  00 00 00 0b
Authentication Request Length       Continue SASL Auth
r=8bnsQo+Ple992is6aol5RGwxrVfgJB7J1or00fFL4T2crJ6L,
Server's Nonce post-fixed with Client's Nonce
s=Brk5ZGyjbS0gXe9EsLIAAQ==,
Salt used by the server for the user being authenticated
i=4096
Iteration Count
```

```
70                 00 00 00 6c
Password Response  Length
c=biws,
Channel Binding is not supported
r=8bnsQo+Ple992is6aol5RGwxrVfgJB7J1or00fFL4T2crJ6L,
Nonce as received in last message
p=n7ztD7URxuRQTOq8Q910dIVvDZthNF2aleUeVSmuLmE=
Client's Proof of Possession of User's Password
Server performs the same algo on user's password, salt etc
and compares with the clients proof
```

```
52                     00 00 00 36        00 00 00 0c
Authentication Request Length             End SASL Auth
v=bbMCiVHlDHPK8J+TUS5w/cmRFD5OAE14EWwlYr62aqk=
Server's Proof of Possession of User's Password
Client performs the same steps on same info and compares with Server's Proof
```

```
Status Parameters
      'S'|Length 4 bytes|Param Name | Param Value
```

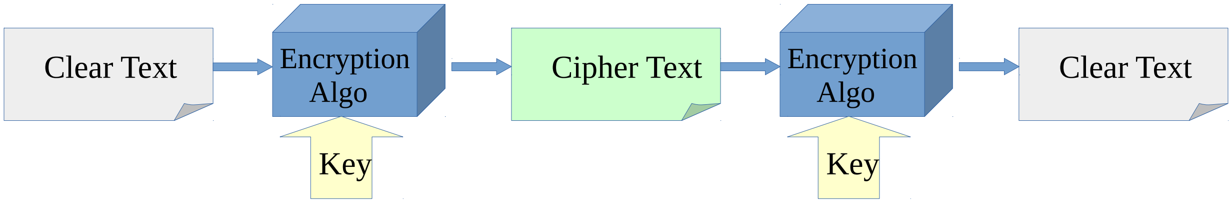
Introduction to Cryptography

Cryptographic algorithms can be classified into two main categories:

Symmetric Key Encryption & Public Key Encryption

Symmetric Key Encryption

Symmetric key algorithms encrypt and decrypt data using a single key.

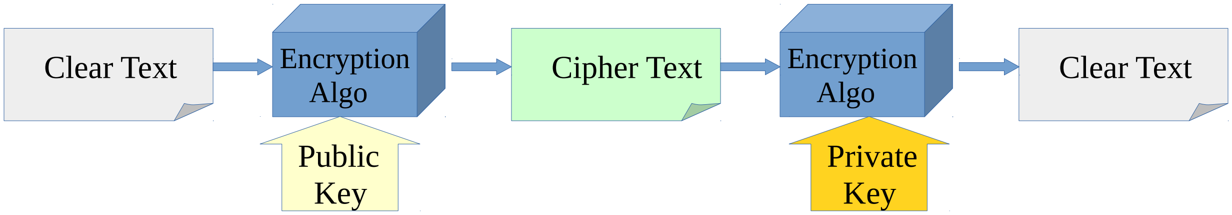


The key in symmetric key algorithms must be kept secret. Exchanging key between the sender and the receiver can be difficult. The same communication channel cannot be used and sending keys in clear is not a very good idea. Security is related to the key length, the longer the better.

Popular symmetric key algorithms are Triple DES, AES. Triple DES uses 112 bit key, AES supports key lengths of 128 bit or more.

Public Key Encryption

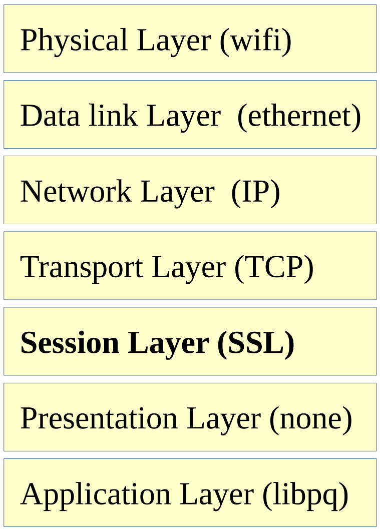
Public Key Encryption uses two keys: one that must remain secret is the **private key** and the one that has to be freely distributed is the **public key**. The public and the private key pair are related to each other in such a manner that a message encrypted by the public key can be decrypted only by its private key pair. Hence there is no issue of key distribution.



Public keys are distributed with a bunch of supporting information called a **certificate**. Certificates are validated by trusted third parties called certification authority. A **certification authority** (CA) certifies that the owner of the public key is the one who is the named subject of the certificate.

Overview of SSL

The secure sockets layer sits in between the application and the transport layer in the OSI model.



Physical Layer (wifi)

Data link Layer (ethernet)

Network Layer (IP)

Transport Layer (TCP)

Session Layer (SSL)

Presentation Layer (none)

Application Layer (libpq)

Setting up SSL in PostgreSQL

Mostly steps are same as mentioned here

https://www.depesz.com/2015/05/11/how-to-setup-ssl-connections-and-authentication/

Check OpenSSL version

```
openssl version
```

```
OpenSSL 1.0.2k-fips  26 Jan 2017
```

Building PostgreSQL with SSL support

```
git clone git://git.postgresql.org/git/postgresql.git
git checkout REL_10_STABLE
sudo yum install readline*
sudo yum install zlib*
./configure --prefix=/usr/local/sslpg10 --with-openssl --enable-debug
CFLAGS="-O0 -g"
make && make install
```

Setup OpenSSL

Make a directory named ca in the home directory

Make a copy of **/etc/pki/tls/openssl.cnf** in the ca directory

Change the following parameters

```
     dir       = /home/abbasbutt/ca
     countryName_default      = PK
     stateOrProvinceName_default    = Punjab
     localityName_default     = Wah
     0.organizationName_default    = EDB
```

Install the openssl-perl package

```
     sudo yum install openssl-perl
```

Copy the **/etc/pki/tls/misc/CA.pl** in the ca directory

Create new CA

```
./CA.pl -newca

     In response to
          Enter PEM pass phrase:
     Enter the pass phrase logitech

     In response to
          Common Name (eg, your name or your server's hostname) []:
     Enter pg/ca

     In response to
          Email Address []:
     Enter your email address

     In response to
          Enter pass phrase for /home/abbasbutt/ca/private/cakey.pem:
     Enter the pass phrase logitech
     Accept defaults for the rest
```

Create public-private key pair for PostgreSQL Server

```
./CA.pl -newreq

     In response to
          Enter PEM pass phrase:
     Enter the pass phrase logitech

     In response to
          Common Name (eg, your name or your server's hostname) []:
     Enter pg/server

     In response to
          Email Address []:
     Enter your email address

     Accept defaults for the rest

./CA.pl -sign

     In response to
          Enter pass phrase for /home/abbasbutt/ca/private/cakey.pem:
     Enter the pass phrase logitech

Rename the public private key pair and set the permissions
     mv newcert.pem pg-server.crt
     mv newkey.pem pg-server.key
     chmod 0600 pg-server.key
```

Make changes in postgresql.conf and pg_hba.conf

```
In postgresql.conf
     ssl = on
     ssl_cert_file = '/home/abbasbutt/ca/pg-server.crt'
     ssl_key_file = '/home/abbasbutt/ca/pg-server.key'
     ssl_ca_file = '/home/abbasbutt/ca/cacert.pem'

In pg_hba.conf
     local      all all                  trust
     hostssl    all all     127.0.0.1/32 trust
     hostssl    all all     ::1/128      trust
     hostnossl  all all     0.0.0.0/0    reject
```

Test the setup

Start the server

```
     ./postgres -D ../data/ -p 6789

     Enter PEM pass phrase:logitech
```

Connect using psql without authentication but with SSL

```
 ./psql -h 127.0.0.1 -p 6789 -U abbasbutt postgres

psql (10.3)

SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits:
256, compression: off)

Type "help" for help.

postgres=#
```

Authenticating using certificates

Create public-private key pair for psql user

```
./CA.pl -newreq
     Generating a 2048 bit RSA private key
     ................................+++
     .................+++
     writing new private key to 'newkey.pem'
     Enter PEM pass phrase:pageup
     Verifying - Enter PEM pass phrase:pageup
     -----
     You are about to be asked to enter information that will be incorporated
     into your certificate request.
     What you are about to enter is what is called a Distinguished Name or a DN.
     There are quite a few fields but you can leave some blank
     For some fields there will be a default value,
     If you enter '.', the field will be left blank.
     -----
     Country Name (2 letter code) [PK]:
     State or Province Name (full name) [Punjab]:
     Locality Name (eg, city) [Wah]:
     Organization Name (eg, company) [EDB]:
     Organizational Unit Name (eg, section) []:
     Common Name (eg, your name or your server's hostname) []:pg/user/simba
     Email Address []:

     Please enter the following 'extra' attributes
     to be sent with your certificate request
     A challenge password []:
     An optional company name []:
     Request is in newreq.pem, private key is in newkey.pem

./CA.pl -sign
     Using configuration from ./openssl.cnf
     Enter pass phrase for /home/abbasbutt/ca/private/cakey.pem:logitech
     Check that the request matches the signature
     Signature ok
     Certificate Details:
           Serial Number:
               f3:94:69:41:67:a1:3c:d3
           Validity
               Not Before: Apr 15 13:16:46 2018 GMT
               Not After : Apr 15 13:16:46 2019 GMT
           Subject:
               countryName               = PK
               stateOrProvinceName       = Punjab
               localityName              = Wah
               organizationName          = EDB
```

```
            commonName                = pg/user/simba
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                F4:44:00:D9:1B:5D:87:CC:B9:E6:27:72:34:D6:3F:77:D8:E1:F2:A9
            X509v3 Authority Key Identifier:
                keyid:5D:62:C7:A2:8A:16:7A:98:A0:81:10:2A:84:DB:2E:39:7E:AC:BD:72

Certificate is to be certified until Apr 15 13:16:46 2019 GMT (365 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
Signed certificate is in newcert.pem
```

Remove the password from the generated key

```
openssl rsa -in newkey.pem -out user-simba.key
     Enter pass phrase for newkey.pem:pageup
     writing RSA key

mv newcert.pem user-simba.crt
rm newreq.pem
```

Copy the public private key pair where psql looks for them

```
mkdir ~/.postgresql

cp user-simba.crt ~/.postgresql/postgresql.crt
cp user-simba.key ~/.postgresql/postgresql.key

chmod 0600 ~/.postgresql/postgresql.key
```

Modify the pg_hba.conf

```
     local      all all                  trust
     hostssl    all all     127.0.0.1/32 cert map=abc
     hostssl    all all     ::1/128      trust
     hostnossl  all all     0.0.0.0/0    reject
```

Modify the pg_ident.conf

Each entry in the pg_ident.conf file takes the form

MAPNAME SYSTEM-USERNAME PG-USERNAME

where

MAPNAME
is the name of the entry to refer to it in pg_hba.conf

SYSTEM-USERNAME
Detected user name of the client

PG-USERNAME
The PostgreSQL user to which SYSTEM-USERNAME should get mapped to

Each entry in this file tells the server that a user SYSTEM-USERNAME may connect as PG-USERNAME.

We need this because for us `psql` tries to connect as user `simba` whereas the `cname` in the certificate carries the name `pg/user/simba`

We add the following line in the pg_ident.conf

```
     abc             pg/user/simba           simba
```

Creatye the user using unix domain trust

```
     ./createuser -d -l -P -r -s -p 6789 simba
     Enter password for new role: simba
     Enter it again: simba
```

Test the authentication setup

```
export  PGSSLCOMPRESSION=0
./psql -h 127.0.0.1 -p 6789 -U simba postgres
psql (10.3)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits:
256, compression: off)
Type "help" for help.

postgres=#
```

SSL Client Side Parameters

libpq allows the following parameters to be set by clients while trying to connect to the PostgreSQL server

sslmode

disable	only try a non-SSL connection
allow	first try a non-SSL connection; if that fails, try an SSL connection
prefer (default)	first try an SSL connection; if that fails, try a non-SSL connection
require	only try an SSL connection
verify-ca	only try an SSL connection, and verify that the server certificate is issued by a trusted certificate authority (CA)
verify-full	only try an SSL connection, verify that the server certificate is issued by a trusted CA and that the requested server host name matches that in the certificate

sslcompression

1 means data sent over SSL connections will be compressed. 0 means compression will be disabled.

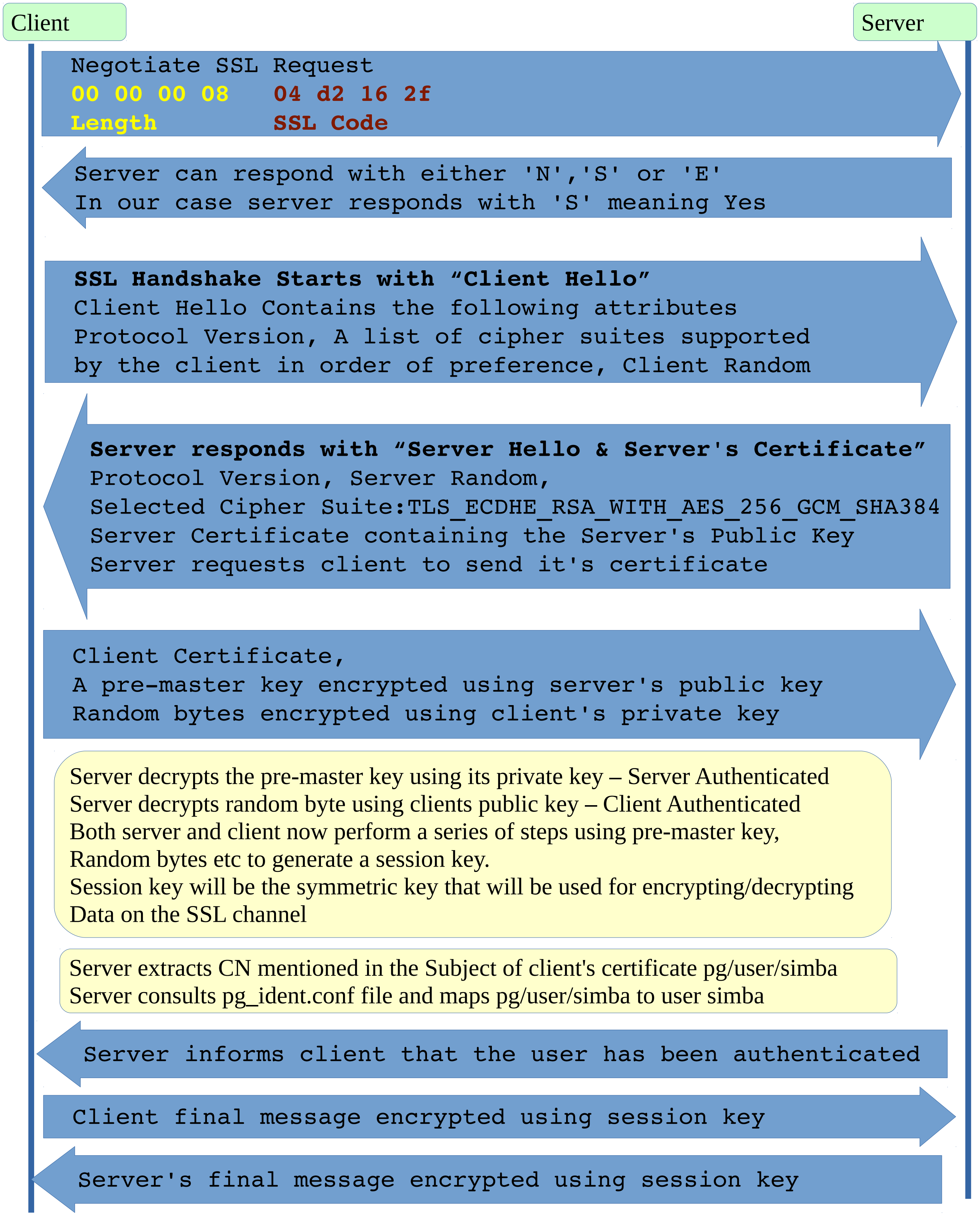
sslcert

This parameter specifies the file name of the client SSL certificate, replacing the default ~/.postgresql/postgresql.crt.

sslkey

This parameter specifies the location for the secret key used for the client certificate, replacing the default ~/.postgresql/postgresql.key.

The SSL Protocol



Client

Server

Negotiate SSL Request
00 00 00 08   04 d2 16 2f
Length   SSL Code

Server can respond with either 'N','S' or 'E'
In our case server responds with 'S' meaning Yes

SSL Handshake Starts with "Client Hello"
Client Hello Contains the following attributes
Protocol Version, A list of cipher suites supported
by the client in order of preference, Client Random

Server responds with "Server Hello & Server's Certificate"
Protocol Version, Server Random,
Selected Cipher Suite:TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
Server Certificate containing the Server's Public Key
Server requests client to send it's certificate

Client Certificate,
A pre-master key encrypted using server's public key
Random bytes encrypted using client's private key

Server decrypts the pre-master key using its private key – Server Authenticated
Server decrypts random byte using clients public key – Client Authenticated
Both server and client now perform a series of steps using pre-master key,
Random bytes etc to generate a session key.
Session key will be the symmetric key that will be used for encrypting/decrypting
Data on the SSL channel

Server extracts CN mentioned in the Subject of client's certificate pg/user/simba
Server consults pg_ident.conf file and maps pg/user/simba to user simba

Server informs client that the user has been authenticated

Client final message encrypted using session key

Server's final message encrypted using session key

What is Kerberos

Kerberos is a Centralized Network Authentication System with the following features:

- Kerberos not only ensures that the person using the desktop is the who he claims to be, but also ensures that the server he is communicating with is who it claims to be.
- Kerberos makes sure that the end users log in once to access all the services and network resources. This is called single sign on.
- Kerberos uses a Kerberos password – the one passwords that the user has to remember to use the entire network resources and services.
- Kerberos ensures that the passwords and other sensitive data is never sent over the network in clear text.

Kerberos Key Distribution Center (KDC)

Kerberos operates through a centralized Key Distribution Center (KDC). Each KDC consists of three logical components:

- Kerberos Database
- Authentication Server
- Ticket Granting Server

Kerberos Realm

A Kerberos realm consists of a set of nodes that use the same Kerberos database.

Kerberos Principal

A Kerberos principal is a service or a user known to the Kerberos database.

A Kerberos 4 principal can take the following forms:
user[.instance]@REALM
service.hostname@REALM

A Kerberos 5 principal can the following forms:
username[/instance]@REALM
service/fully-qualified-domain-name@REALM

Kerberos Database

It contains all the principals of a Kerberos Realm along with their associated secrets.

Kerberos Ticket

It is an encrypted data structure issued by the KDC to confirm the identity of the end participants and to establish a session key. It contains the following information:

- The user's principal
- The service's principal
- Ticket Validity
- Ticket Expiry
- A list of IP addresses the ticket can be used from
- A shared secret encryption key – the session key

Ticket Granting Ticket

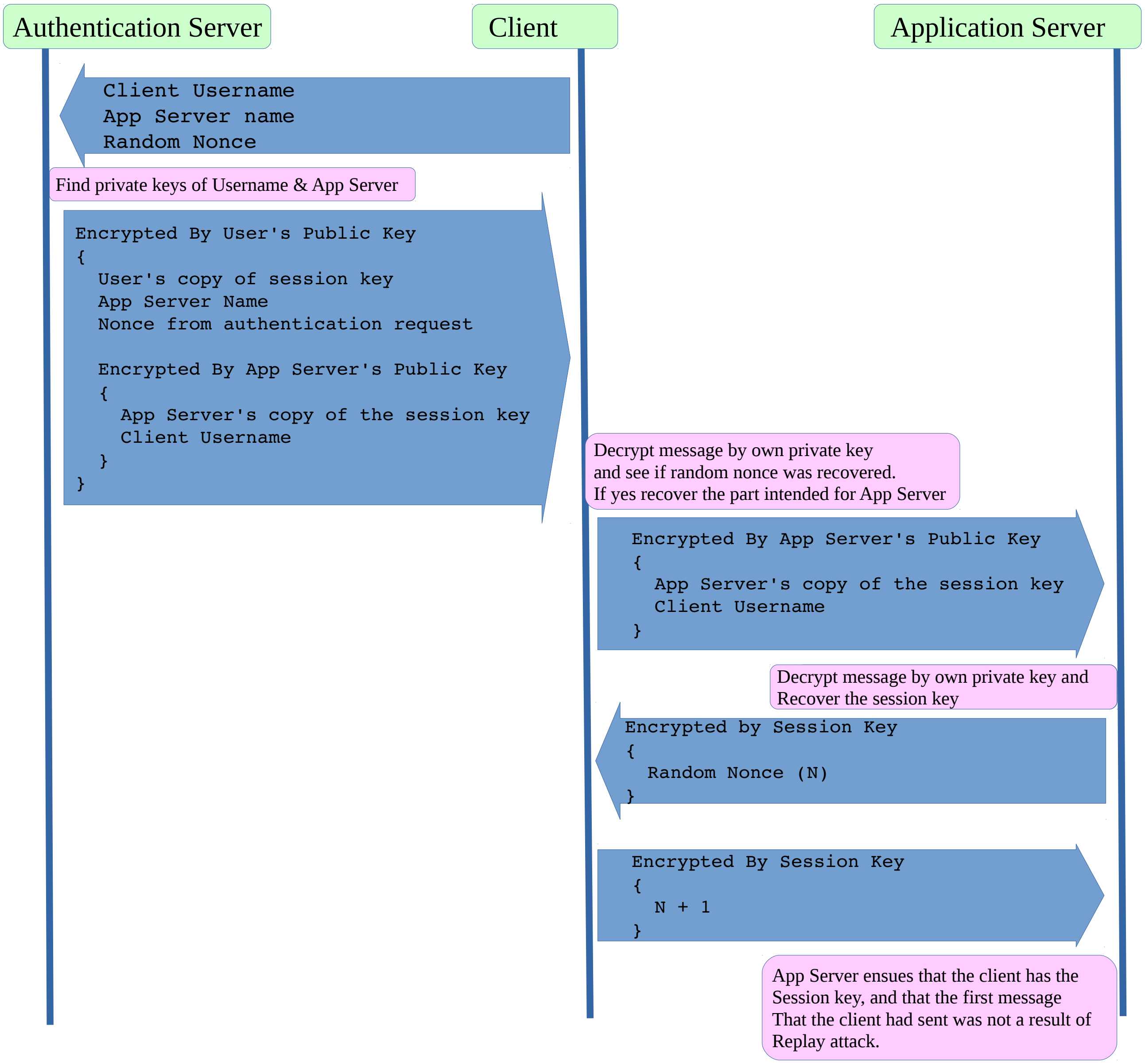
The authentication server issues an encrypted Ticket Granting Ticket (TGT) to the clients who want to login to the Kerberos realm. This ticket can only be decrypted with the user's password. The user types in his password and the login process tries to decrypt the TGT. The correct password will correctly decrypt the TGT, incorrect password will decrypt the TGT into garbage. Once decrypted the user will have access to the session key.

Ticket Granting Server

Ticket Granting Server (TGS) issues individual service tickets to the clients as they request them. The clients sends service's principal name and a TGT to the TGS. TGS verifies that the TGT is valid by checking that it has been encrypted using the Authentication server's TGT key and then issues the service ticket.

The Needham-Schroeder Protocol

Rodger Needham and Michael Schroeder published a paper in 1978 describing a framework for providing secure network authentication system. Kerberos authentication is based on this paper.



```
Client Username
App Server name
Random Nonce
```

Find private keys of Username & App Server

```
Encrypted By User's Public Key
{
  User's copy of session key
  App Server Name
  Nonce from authentication request

  Encrypted By App Server's Public Key
  {
    App Server's copy of the session key
    Client Username
  }
}
```

Decrypt message by own private key
and see if random nonce was recovered.
If yes recover the part intended for App Server

```
Encrypted By App Server's Public Key
{
  App Server's copy of the session key
  Client Username
}
```

Decrypt message by own private key and
Recover the session key

```
Encrypted by Session Key
{
  Random Nonce (N)
}
```

```
Encrypted By Session Key
{
  N + 1
}
```

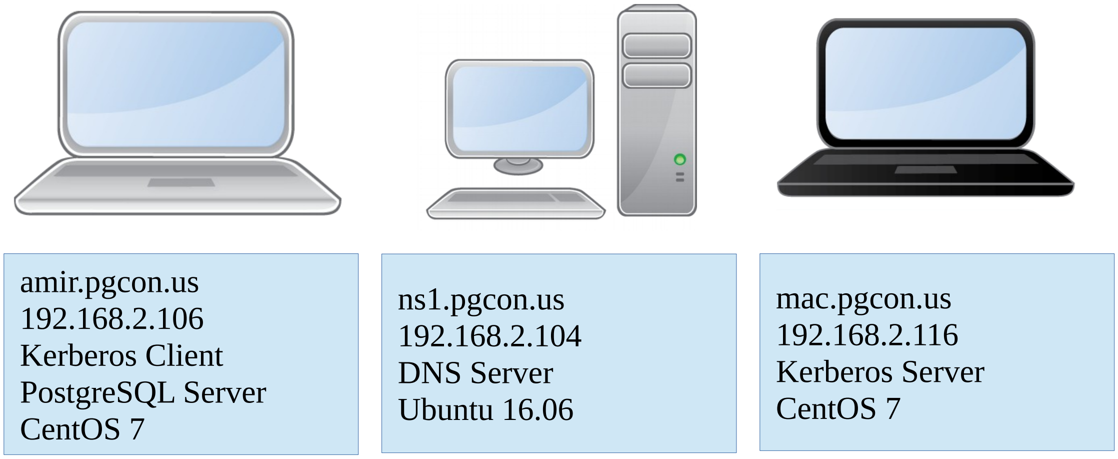
App Server ensues that the client has the
Session key, and that the first message
That the client had sent was not a result of
Replay attack.

The General Security Services API (GSSAPI)

PostgreSQL uses GSSAPI as a means to provide Kerberos 5 support. GSSAPI provides an abstraction layer over a particular platform, security mechanism, type of protection or transport protocol. In addition to Kerberos, GSSAPI provides support for other security mechanisms too. GSSAPI shields complexities of libkrb5. GSSAPI v2 is specified in RFC 2743, RFC 2744 & RFC 7546.

Kerberos Setup

The setup consists of network of three computers as follows:



amir.pgcon.us
192.168.2.106
Kerberos Client
PostgreSQL Server
CentOS 7

ns1.pgcon.us
192.168.2.104
DNS Server
Ubuntu 16.06

mac.pgcon.us
192.168.2.116
Kerberos Server
CentOS 7

Setting up the DNS Server

1. `sudo apt-get install bind9 bind9utils`
2. `sudo vim /etc/bind/named.conf.options`

```
acl "trusted" {
        192.168.2.106;
        192.168.2.116;
        192.168.2.104;
        192.168.2.1;
};
options {
        directory "/var/cache/bind";
        recursion yes;
        allow-recursion { trusted; };
        listen-on { 192.168.2.104; };
        allow-transfer { none; };
     forwarders {
          8.8.8.8;
     };
};
```

3. `sudo vim /etc/bind/named.conf.local`

```
zone "pgcon.us" {
    type master;
    file "/etc/bind/zones/db.pgcon.us";
};

zone "2.168.192.in-addr.arpa" {
    type master;
    file "/etc/bind/zones/db.2.168.192";
};
```

In the folder /etc/bind/zones create the following files

4. sudo vim db.2.168.192

```
$TTL    604800
@       IN      SOA     pgcon.us. admin.pgcon.us. (
                              3         ; Serial
                         604800         ; Refresh
                          86400         ; Retry
                        2419200         ; Expire
                         604800 )       ; Negative Cache TTL
; name servers
      IN      NS      ns1.pgcon.us.

; PTR Records
104  IN      PTR     ns1.pgcon.us.
106  IN      PTR     amir.pgcon.us.
116  IN      PTR     mac.pgcon.us.
```

5. sudo vim db.pgcon.us

```
$TTL    604800
@       IN      SOA     ns1.pgcon.us. admin.pgcon.us. (
                  3       ; Serial
             604800     ; Refresh
              86400     ; Retry
            2419200     ; Expire
             604800 )   ; Negative Cache TTL
;
; name servers - NS records
     IN      NS      ns1.pgcon.us.

; name servers - A records
ns1.pgcon.us.         IN      A      192.168.2.104

amir.pgcon.us.        IN      A      192.168.2.106
mac.pgcon.us.         IN      A      192.168.2.116
```

6. Check configuration should not throw any error

```
abbas@abbas-Studio-1537:/etc/bind/zones$ sudo named-checkconf
abbas@abbas-Studio-1537:/etc/bind/zones$
```

7. sudo service bind9 restart

8. sudo service bind9 status

```
[sudo] password for abbas:
● bind9.service - BIND Domain Name Server
   Loaded: loaded (/lib/systemd/system/bind9.service; enabled; vendor preset: enabled)
  Drop-In: /run/systemd/generator/bind9.service.d
           └─50-insserv.conf-$named.conf
   Active: active (running) since 12:15:26 13-04-2018 جمعه PKT; 2h 53min ago
     Docs: man:named(8)
  Process: 24524 ExecStop=/usr/sbin/rndc stop (code=exited, status=0/SUCCESS)
 Main PID: 24532 (named)
   CGroup: /system.slice/bind9.service
           └─24532 /usr/sbin/named -f -4 -u bind
```

9. sudo named-checkzone pgcon.us db.pgcon.us

```
zone pgcon.us/IN: loaded serial 3
OK
```

10. sudo named-checkzone 2.168.192.in-addr.arpa /etc/bind/zones/db.2.168.192

```
zone 2.168.192.in-addr.arpa/IN: loaded serial 3
OK
```

11. Test DNS Server from both Kerberos Client & Kerberos Server

```
nslookup ns1.pgcon.us
     Server:         192.168.2.104
     Address:   192.168.2.104#53

     Name:ns1.pgcon.us
     Address: 192.168.2.104

nslookup mac.pgcon.us
     Server:         192.168.2.104
     Address:   192.168.2.104#53

     Name:mac.pgcon.us
     Address: 192.168.2.116

nslookup amir.pgcon.us
     Server:         192.168.2.104
     Address:   192.168.2.104#53

     Name:amir.pgcon.us
     Address: 192.168.2.106
```

Setting up the Kerberos Server

1. `sudo yum install krb5-libs krb5-server krb5-workstation`

2. `sudo vim /etc/krb5.conf`

```
[logging]
    default = FILE:/var/log/krb5libs.log
    kdc = FILE:/var/log/krb5kdc.log
    admin_server = FILE:/var/log/kadmind.log

[libdefaults]
    default_realm = PGCON.US
    dns_lookup_realm = false
    dns_lookup_kdc = false
    ticket_lifetime = 24h
    renew_lifetime = 7d
    forwardable = yes
    default_tgs_enctypes = aes128-cts des3-hmac-sha1 des-cbc-crc des-cbc-md5
    default_tkt_enctypes = aes128-cts des3-hmac-sha1 des-cbc-crc des-cbc-md5
    permitted_enctypes = aes128-cts des3-hmac-sha1 des-cbc-crc des-cbc-md5

[realms]
    PGCON.US = {
        kdc = mac.pgcon.us:88
        admin_server = mac.pgcon.us:749
        default_domain = pgcon.us
    }

[domain_realm]
    .pgcon.us = PGCON.US
    pgcon.us = PGCON.US
```

3. `sudo kdb5_util create -s`

```
Loading random data

Initializing database '/var/kerberos/krb5kdc/principal' for realm 'PGCON.US',

master key name 'K/M@PGCON.US'

You will be prompted for the database Master Password.

It is important that you NOT FORGET this password.

Enter KDC database master key:

Re-enter KDC database master key to verify:
```

```
4. sudo kadmin.local -q "addprinc abbas/admin"

    Authenticating as principal abbas/admin@PGCON.US with password.

    WARNING: no policy specified for abbas/admin@PGCON.US; defaulting to no policy

    Enter password for principal "abbas/admin@PGCON.US":

    Re-enter password for principal "abbas/admin@PGCON.US":

    Principal "abbas/admin@PGCON.US" created.

5. sudo krb5kdc

6. sudo kadmin.local -q "addprinc postgres/amir.pgcon.us@PGCON.US"

    Authenticating as principal abbas/admin@PGCON.US with password.

    WARNING: no policy specified for postgres/amir.pgcon.us@PGCON.US; defaulting to no policy

    Enter password for principal "postgres/amir.pgcon.us@PGCON.US":

    Re-enter password for principal "postgres/amir.pgcon.us@PGCON.US":

    Principal "postgres/amir.pgcon.us@PGCON.US" created.


7. sudo kadmin.local -q "xst -k pgcon.us.keytab postgres/amir.pgcon.us@PGCON.US"

    Authenticating as principal abbas/admin@PGCON.US with password.

    Entry for principal postgres/amir.pgcon.us@PGCON.US with kvno 2, encryption type aes256-cts-hmac-sha1-96 added to keytab WRFILE:pgcon.us.keytab.

    Entry for principal postgres/amir.pgcon.us@PGCON.US with kvno 2, encryption type aes128-cts-hmac-sha1-96 added to keytab WRFILE:pgcon.us.keytab.
```

Setting up the Kerberos Client

1. sudo vim /etc/krb5.conf

```
[logging]
    default = FILE:/var/log/krb5libs.log
    kdc = FILE:/var/log/krb5kdc.log
    admin_server = FILE:/var/log/kadmind.log

[libdefaults]
    default_realm = PGCON.US
    dns_lookup_realm = false
    dns_lookup_kdc = false
    ticket_lifetime = 24h
    renew_lifetime = 7d
    forwardable = yes
    default_tgs_enctypes = aes128-cts des3-hmac-sha1 des-cbc-crc des-cbc-md5
    default_tkt_enctypes = aes128-cts des3-hmac-sha1 des-cbc-crc des-cbc-md5
    permitted_enctypes = aes128-cts des3-hmac-sha1 des-cbc-crc des-cbc-md5

[realms]
    PGCON.US = {
        kdc = mac.pgcon.us:88
        admin_server = mac.pgcon.us:749
        default_domain = pgcon.us
    }

[domain_realm]
    .pgcon.us = PGCON.US
    pgcon.us = PGCON.US
```

2. Copy pgcon.us.keytab from the Kerberos Server machine to the client machine and sudo chown abbas:abbas pgcon.us.keytab

3. klist

```
klist: No credentials cache found (filename: /tmp/krb5cc_1000)
```

4. kinit -k -t pgcon.us.keytab postgres/amir.pgcon.us@PGCON.US

5. klist

```
Ticket cache: FILE:/tmp/krb5cc_1000

Default principal: postgres/amir.pgcon.us@PGCON.US


Valid starting       Expires              Service principal

04/13/2018 04:30:18  04/14/2018 04:30:18  krbtgt/PGCON.US@PGCON.US
```

Setting up the PostgreSQL Server for Kerberos

1. Build and install PostgreSQL Server

```
git clone git://git.postgresql.org/git/postgresql.git

git checkout REL_10_STABLE

./configure --prefix=/usr/local/pg10 —with-gssapi

            --enable-debug CFLAGS="-O0 -g"

make && make install
```

2. Modify the pg_hba.conf

```
local all all           trust

host  all all 0.0.0.0/0 gss include_realm=1 krb_realm=PGCON.US

host  all all ::1/128   trust
```

3. Modify postgresql.conf

```
krb_server_keyfile = '/home/abbas/pgcon.us.keytab'
```

4. Start the server

```
./postgres -D ../data -p 5678
```

5. Create user using trust authentication

```
./psql -U abbas -p 5678 postgres -c 'CREATE ROLE
"postgres/amir.pgcon.us@PGCON.US" SUPERUSER LOGIN'
```

6. `./psql -U postgres/amir.pgcon.us@PGCON.US -h amir.pgcon.us -p 5678 postgres`

```
psql (10.3)

Type "help" for help.


postgres=#
```

Common LDAP Terms

In the good old days there used to be a telephone directory containing a complete list of names and telephone numbers of a certain region, company or a service provider. Using this directory it was possible to find the telephone number of a friend.

With the advent of computers there is no end of information that needs organizing. Even DOS had a directory. In computers directories provide an efficient way of managing information so that its easy to find the required information. Each directory has a list of entries. Each **entry** has a list of attribute value pairs. A **container** is a special type of entry which helps organize other entries by a parent/child relationship. A commonly used container **object class** is OU, Organizational Unit. Person entries in a directory can go to container People, while product entries can be contained in container Products.

Containers can have other containers as children, but child entries can have only a single container as a parent allowing only a pyramid (hierarchical) organizational structure.

Each entry in a directory has a unique name know as **distinguished name DN**.

Each entry also has a name local to its immediate container known as the **relative distinguished name (RDN)**.

Each directory has a root. The name of the root of the directory is directory's **base DN**. The base DN typically is same as the server's domain name.

Schema provides the set of rules that define what type of entries can be in a directory. Schema acts as a packaging unit.

Object classes provide a grouping for sets of attributes. Object classes are defined with in schemas.

Commonly used object classes are as follows:

c	countryName
cn	commonName
dc	domainComponent
co	friendlyCountryName
gn	givenName
homePhone	homeTelephoneNumber
l	localityName
mobile	mobileTelephoneNumber
o	organizationName
ou	organisationalUnitName
postalCode	postalCode
sn	surname
st	stateOrProvinceName
street	streetAddress
uid	userid

What is LDAP

LDAP stands for Lightweight directory access protocol. LDAP version 3 is defined by a set of nine RFCs: 2251-2256, 2829, 2830 & 3377. LDAP defines a set of server operations used to manipulate information stored by the directory. The operations are add, modify, delete, search, compare, bind etc. LDAP uses TCP/IP port 389 for communication between the LDAP server and the LDAP client.

The bind operation is used to authenticate clients using the username password pair provided.

LDAP server is provided by many popular vendors, we are however going to use 389-DS.

LDAP Authentication in PostgreSQL

PostgreSQL supports LDAP authentication in two modes: simple bind mode & search + bind mode.

Simple Bind Mode:

In simple bind mode distinguished name is constructed as *prefix username suffix.* PostreSQL binds with the directory server using this DN and client provided password to do the authentication.

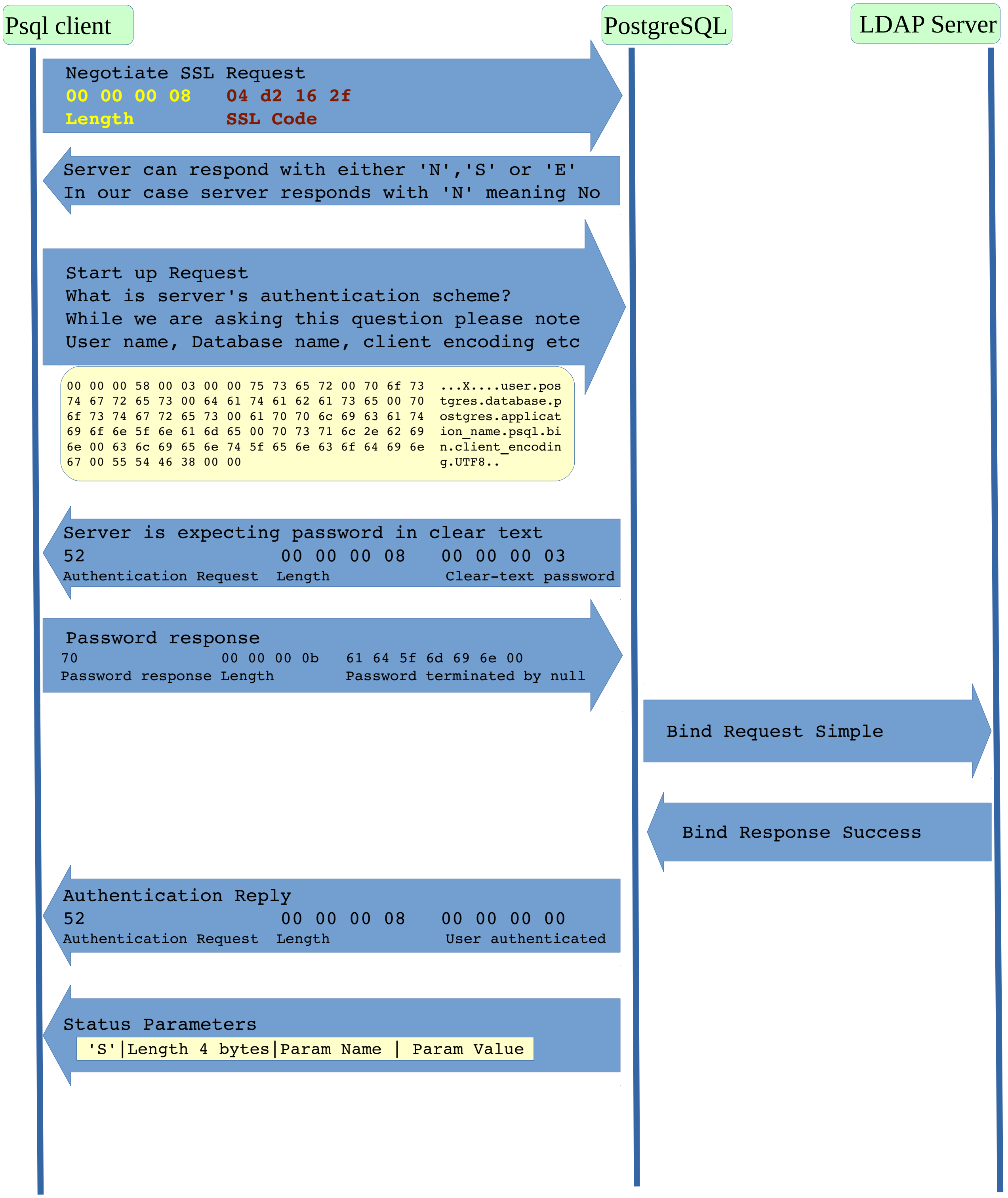
Search + Bind Mode:

This is a multi step process:

- Bind with the directory server using *ldapbinddn* and *ldapbindpasswd.*
- Search for the user provided by the client in the sub-tree starting at *ldapbasedn*, trying to do an exact match of the attribute specified in *ldapsearchattribute*.
- If the user provided by client is found, rebind to the directory server using the client provided username and password to authenticate.

We are using Simple Bind Mode in our example.

Overview of LDAP protocol



```
Negotiate SSL Request
00 00 00 08    04 d2 16 2f
Length         SSL Code
```

```
Server can respond with either 'N','S' or 'E'
In our case server responds with 'N' meaning No
```

```
Start up Request
What is server's authentication scheme?
While we are asking this question please note
User name, Database name, client encoding etc
```

```
00 00 00 58 00 03 00 00 75 73 65 72 00 70 6f 73  ...X....user.pos
74 67 72 65 73 00 64 61 74 61 62 61 73 65 00 70  tgres.database.p
6f 73 74 67 72 65 73 00 61 70 70 6c 69 63 61 74  ostgres.applicat
69 6f 6e 5f 6e 61 6d 65 00 70 73 71 6c 2e 62 69  ion_name.psql.bi
6e 00 63 6c 69 65 6e 74 5f 65 6e 63 6f 64 69 6e  n.client_encodin
67 00 55 54 46 38 00 00                          g.UTF8..
```

```
Server is expecting password in clear text
52                     00 00 00 08    00 00 00 03
Authentication Request  Length          Clear-text password
```

```
Password response
70                      00 00 00 0b    61 64 5f 6d 69 6e 00
Password response Length               Password terminated by null
```

```
Bind Request Simple
```

```
Bind Response Success
```

```
Authentication Reply
52                     00 00 00 08    00 00 00 00
Authentication Request  Length          User authenticated
```

```
Status Parameters
 'S'|Length 4 bytes|Param Name | Param Value
```

LDAP Bind Request

```
30 52 02 01 01 60 4d 02 01 03 04 40 75 69 64 3d   0R...`M....@uid=
61 64 6d 69 6e 2c 6f 75 3d 41 64 6d 69 6e 69 73   admin,ou=Adminis
74 72 61 74 6f 72 73 2c 6f 75 3d 54 6f 70 6f 6c   trators,ou=Topol
6f 67 79 4d 61 6e 61 67 65 6d 65 6e 74 2c 6f 3d   ogyManagement,o=
4e 65 74 73 63 61 70 65 52 6f 6f 74 80 06 61 64   NetscapeRoot..ad
5f 6d 69 6e                                       _min
```

```
[ 1] LDAP Tag Sequence (0x30)
[ 1] Sequence Length (82)
[ 1] LDAP Tag Integer (2)
[ 1] Integer Length (1)
[ 1] Message ID (1)
[ 1] Bind Request (0x60)
[ 1] Length (77 bytes)
[ 1] LDAP Tag Integer (2)
[ 1] Integer Length (1)
[ 1] LDAP Version (3)
[ 1] LDAP distinguished name (0x04)
[ 1] length (64)
[64] Value (uid=admin,ou=Administrators,ou=TopologyManagement,o=NetscapeRoot)
```

```
[ 1] LDAP Auth Simple (0x80)
[ 1] length (6)
[ 6] Value (ad_min)
```

LDAP Bind Response

```
30 0c 02 01 01 61 07 0a 01 00 04 00 04 00         0....a......…
```

```
[ 1] LDAP Tag Sequence (0x30)
[ 1] Sequence Length (12)
[ 1] LDAP Tag Integer (2)
[ 1] Integer Length (1)
[ 1] Message ID (1)
[ 1] Bind Response (0x61)
[ 1] Length (7)
[ 1] LDAP Tag Enum (0x0a)
[ 1] Enum Length (1)
[ 1] Bind Result 0x00:Success 0x31:Invalid Credentials
```

LDAP Configuration Steps

Installing 389-DS

```
sudo yum install epel-release
sudo yum install 389-ds-base openldap-clients idm-console-framework 389-adminutil 389-admin
```

Configuring 389-DS

```
sudo setup-ds-admin.pl
[sudo] password for abbas:

==============================================================================
This program will set up the 389 Directory and Administration Servers.

It is recommended that you have "root" privilege to set up the software.
Tips for using this program:
  - Press "Enter" to choose the default and go to the next screen
  - Type "Control-B" then "Enter" to go back to the previous screen
  - Type "Control-C" to cancel the setup program

Would you like to continue with set up? [yes]:

==============================================================================
Your system has been scanned for potential problems, missing patches,
etc.  The following output is a report of the items found that need to
be addressed before running this software in a production
environment.

389 Directory Server system tuning analysis version 14-JULY-2016.

NOTICE : System is x86_64-unknown-linux3.10.0-693.el7.x86_64 (2 processors).

NOTICE : The net.ipv4.tcp_keepalive_time is set to 7200000 milliseconds
(120 minutes).  This may cause temporary server congestion from lost
client connections.

WARNING: There are only 1024 file descriptors (soft limit) available, which
limit the number of simultaneous connections.

WARNING  : The warning messages above should be reviewed before proceeding.

Would you like to continue? [no]: yes

==============================================================================
Choose a setup type:

   1. Express
       Allows you to quickly set up the servers using the most
       common options and pre-defined defaults. Useful for quick
       evaluation of the products.
   2. Typical
       Allows you to specify common defaults and options.

   3. Custom
       Allows you to specify more advanced options. This is
       recommended for experienced server administrators only.

To accept the default shown in brackets, press the Enter key.

Choose a setup type [2]: 2

==============================================================================
Enter the fully qualified domain name of the computer
on which you're setting up server software. Using the form
<hostname>.<domainname>
Example: eros.example.com.

To accept the default shown in brackets, press the Enter key.

Warning: This step may take a few minutes if your DNS servers
can not be reached or if DNS is not configured correctly.  If
you would rather not wait, hit Ctrl-C and run this program again
with the following command line option to specify the hostname:

    General.FullMachineName=your.hostname.domain.name

Computer name [localhost.localdomain]:

==============================================================================
The servers must run as a specific user in a specific group.
It is strongly recommended that this user should have no privileges
on the computer (i.e. a non-root user).  The setup procedure
will give this user/group some permissions in specific paths/files
to perform server-specific operations.

If you have not yet created a user and group for the servers,
create this user and group using your native operating
system utilities.

System User [dirsrv]: ldapadmin
System Group [dirsrv]: ldapadmin

==============================================================================
Server information is stored in the configuration directory server.
This information is used by the console and administration server to
configure and manage your servers.  If you have already set up a
configuration directory server, you should register any servers you
set up or create with the configuration server.  To do so, the
following information about the configuration server is required: the
fully qualified host name of the form
<hostname>.<domainname>(e.g. hostname.example.com), the port number
(default 389), the suffix, the DN and password of a user having
permission to write the configuration information, usually the
configuration directory administrator, and if you are using security
(TLS/SSL).  If you are using TLS/SSL, specify the TLS/SSL (LDAPS) port
```

```
number (default 636) instead of the regular LDAP port number, and
provide the CA certificate (in PEM/ASCII format).

If you do not yet have a configuration directory server, enter 'No' to
be prompted to set up one.

Do you want to register this software with an existing
configuration directory server? [no]:

==============================================================================
Please enter the administrator ID for the configuration directory
server.  This is the ID typically used to log in to the console.  You
will also be prompted for the password.

Configuration directory server
administrator ID [admin]: admin
Password: ad_min
Password (confirm): ad_min

==============================================================================
The information stored in the configuration directory server can be
separated into different Administration Domains.  If you are managing
multiple software releases at the same time, or managing information
about multiple domains, you may use the Administration Domain to keep
them separate.

If you are not using administrative domains, press Enter to select the
default.  Otherwise, enter some descriptive, unique name for the
administration domain, such as the name of the organization
responsible for managing the domain.

Administration Domain [localdomain]:

==============================================================================
The standard directory server network port number is 389.  However, if
you are not logged as the superuser, or port 389 is in use, the
default value will be a random unused port number greater than 1024.
If you want to use port 389, make sure that you are logged in as the
superuser, that port 389 is not in use.

Directory server network port [389]:

==============================================================================
Each instance of a directory server requires a unique identifier.
This identifier is used to name the various
instance specific files and directories in the file system,
as well as for other uses as a server instance identifier.

Directory server identifier [localhost]:

==============================================================================
The suffix is the root of your directory tree.  The suffix must be a valid DN.
It is recommended that you use the dc=domaincomponent suffix convention.
For example, if your domain is example.com,
you should use dc=example,dc=com for your suffix.
```

```
Setup will create this initial suffix for you,
but you may have more than one suffix.
Use the directory server utilities to create additional suffixes.

Suffix [dc=localdomain]:

==============================================================================
Certain directory server operations require an administrative user.
This user is referred to as the Directory Manager and typically has a
bind Distinguished Name (DN) of cn=Directory Manager.
You will also be prompted for the password for this user.  The password must
be at least 8 characters long, and contain no spaces.
Press Control-B or type the word "back", then Enter to back up and start over.

Directory Manager DN [cn=Directory Manager]:
Password:
Password (confirm):

==============================================================================
The Administration Server is separate from any of your web or application
servers since it listens to a different port and access to it is
restricted.

Pick a port number between 1024 and 65535 to run your Administration
Server on. You should NOT use a port number which you plan to
run a web or application server on, rather, select a number which you
will remember and which will not be used for anything else.

Administration port [9830]:

==============================================================================
The interactive phase is complete.  The script will now set up your
servers.  Enter No or go Back if you want to change something.

Are you ready to set up your servers? [yes]:
Creating directory server . . .
Your new DS instance 'localhost' was successfully created.
Creating the configuration directory server . . .
Beginning Admin Server creation . . .
Creating Admin Server files and directories . . .
Updating adm.conf . . .
Updating admpw . . .
Registering admin server with the configuration directory server . . .
Updating adm.conf with information from configuration directory server . . .
Updating the configuration for the httpd engine . . .
Starting admin server . . .
The admin server was successfully started.
Admin server was successfully created, configured, and started.
Exiting . . .
Log file is '/tmp/setupdt8sC5.log'
```

Testing 389-DS

Check the **/etc/dirsrv/admin-serv/adm.conf** file for the user created by the configuration script.

```
ldapwhoami -vvv -h 192.168.115.219 -D
"uid=admin,ou=Administrators,ou=TopologyManagement,o=NetscapeRoot" -x -w
ad_min
ldap_initialize( ldap://192.168.115.219 )
dn: uid=admin,ou=administrators,ou=topologymanagement,o=netscaperoot
Result: Success (0)
```

Building PostgreSQL with LDAP support

```
git clone git://git.postgresql.org/git/postgresql.git
git checkout REL_10_STABLE
sudo yum install readline*
sudo yum install zlib*
sudo yum install openldap-devel*
./configure --prefix=/usr/local/pg10 --with-ldap --enable-debug CFLAGS="-O0
-g"
make && make install
```

Configure pg_hba.conf

```
local all all              trust
host  all all 127.0.0.1/32 ldap ldapserver=192.168.115.216 ldapprefix="uid="
ldapsuffix=",ou=Administrators,ou=TopologyManagement,o=NetscapeRoot"
host  all all ::1/128      ldap ldapserver=192.168.115.216 ldapprefix="uid="
ldapsuffix=",ou=Administrators,ou=TopologyManagement,o=NetscapeRoot"
```

Test LDAP support

```
cd /usr/local/pg10/bin
./initdb -D ../data
./postgres -D ../data -p 6543

Create the user to test
./createuser -d -l -P -r -s -h 127.0.0.1 -p 6543 admin
Give password test,it will not be used any way

./psql -h 127.0.0.1 -p 6543 -U admin postgres
Password for user admin: ad_min
psql (10.3)
Type "help" for help.

postgres=#
```