The OpenSSH agent

Its use and custom hacks

Mirko Mariotti

Security course
MSc course in Computer Science
Department of Mathematics and Computer Science
University of Perugia

Monday 24th June, 2013

Contents

The SSH protocol

- SSH is the well known protocol used to:

The SSH protocol

- SSH is the well known protocol used to:
  1. Launch a shell (or in general spawn any command) on a remote machine in a secure way.

The SSH protocol

- SSH is the well known protocol used to:
  1. Launch a shell (or in general spawn any command) on a remote machine in a secure way.
  2. Transfer files for and to a remote machine.

The SSH protocol

- SSH is the well known protocol used to:
  1. Launch a shell (or in general spawn any command) on a remote machine in a secure way.
  2. Transfer files for and to a remote machine.
- And its use is straightforward:

The SSH protocol

- SSH is the well known protocol used to:
  1. Launch a shell (or in general spawn any command) on a remote machine in a secure way.
  2. Transfer files for and to a remote machine.
- And its use is straightforward:
  1. **Connect to a remote shell**:
     ssh username@remotehostname

The SSH protocol

- SSH is the well known protocol used to:
  1. Launch a shell (or in general spawn any command) on a remote machine in a secure way.
  2. Transfer files for and to a remote machine.
- And its use is straightforward:
  1. **Connect to a remote shell**:
     ssh username@remotehostname
  2. **Download files**:
     scp username@remotehostname:remotepath localpath

The SSH protocol

- SSH is the well known protocol used to:
  1. Launch a shell (or in general spawn any command) on a remote machine in a secure way.
  2. Transfer files for and to a remote machine.
- And its use is straightforward:
  1. **Connect to a remote shell**:
     ssh username@remotehostname
  2. **Download files**:
     scp username@remotehostname:remotepath localpath
  3. **Upload files**:
     scp localpath username@remotehostname:remotepath

The SSH Client and Server

SSH has a client-server architecture, a daemon called *sshd* run on the remote machine and the *ssh* client is called in the local one.

The SSH Client and Server

SSH has a client-server architecture, a daemon called *sshd* run on the remote machine and the *ssh* client is called in the local one.

When called *ssh* connect to *sshd* and ask for username and password.

The SSH Client and Server

SSH has a client-server architecture, a daemon called *sshd* run on the remote machine and the *ssh* client is called in the local one.

When called *ssh* connect to *sshd* and ask for username and password.

Upon right authentication a secure channel is established beetwen the two part and a remote process fds are connected to the local ssh.

The SSH Client and Server

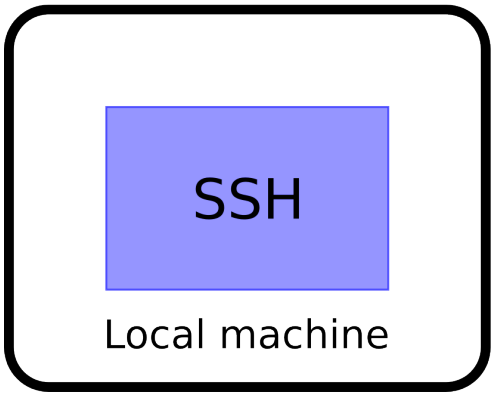
SSH has a client-server architecture, a daemon called *sshd* run on the remote machine and the *ssh* client is called in the local one.

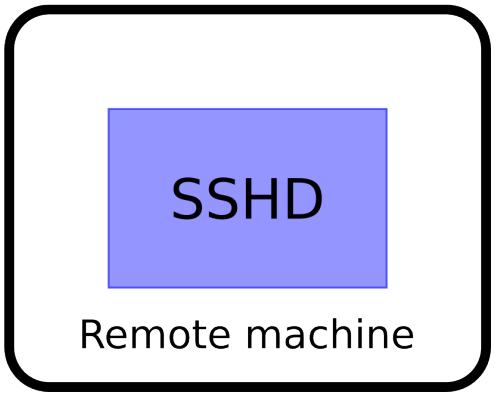
When called *ssh* connect to *sshd* and ask for username and password.

Upon right authentication a secure channel is established beetwen the two part and a remote process fds are connected to the local ssh.

The user can control the remote process from the local machine.

The SSH Client and Server





The SSH Client and Server

User wants to connect with the remote machine and issues the ssh command

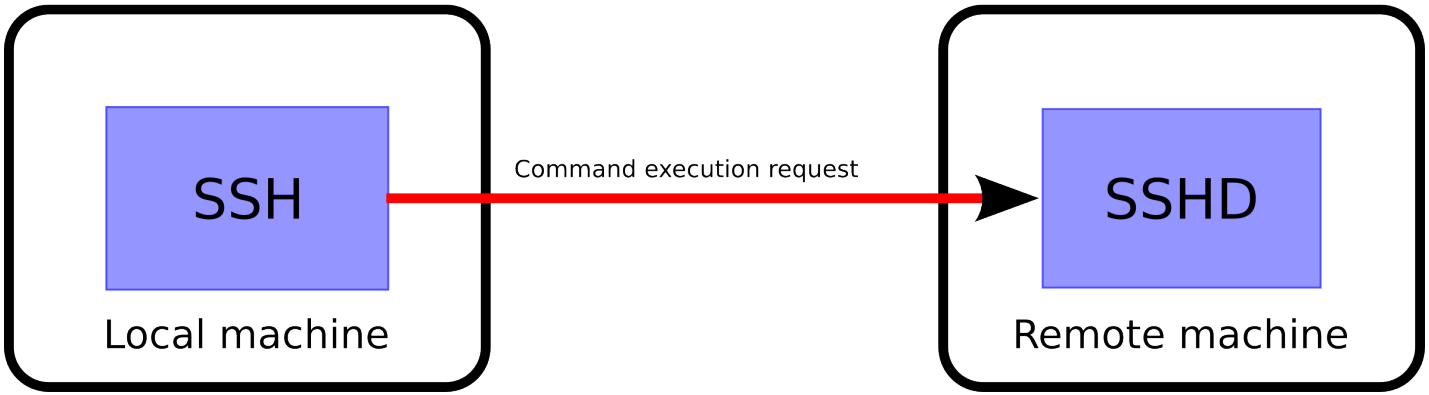
SSH

Local machine

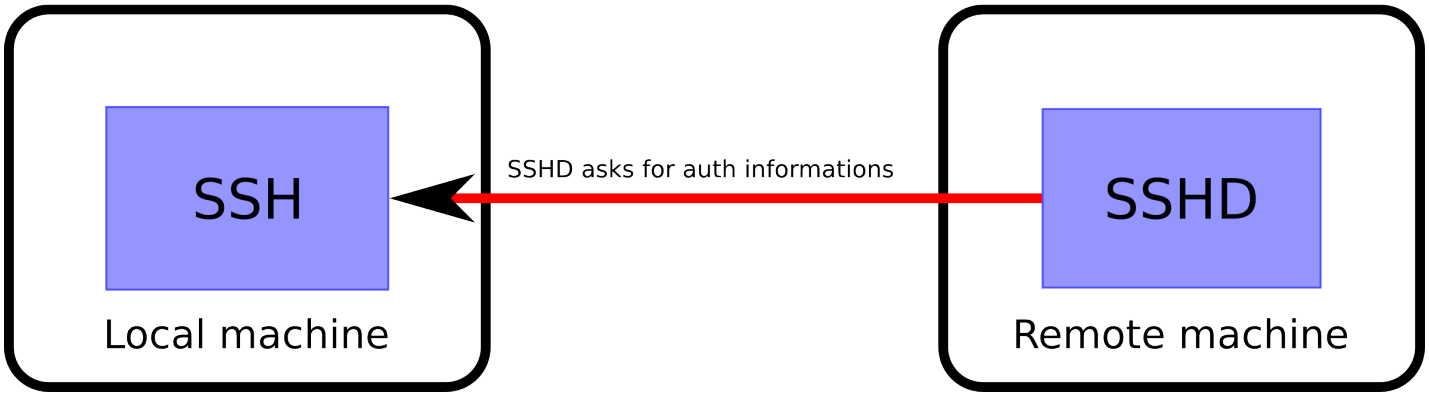
SSHD

Remote machine

The SSH Client and Server

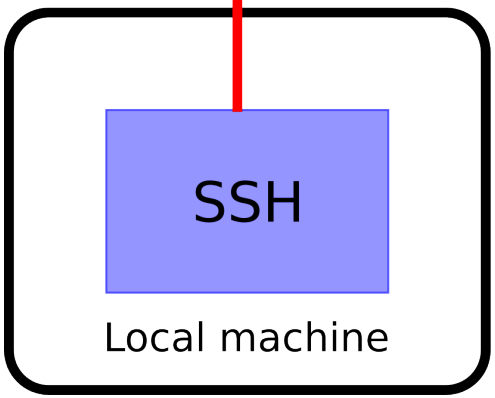


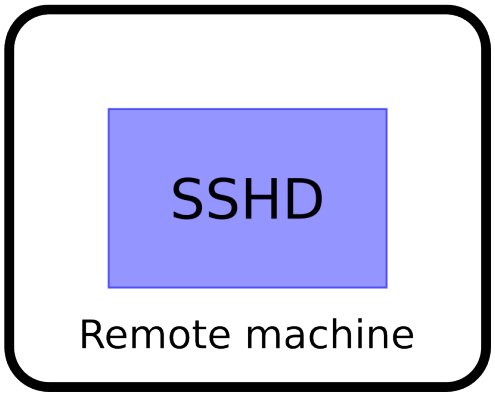
The SSH Client and Server



The SSH Client and Server

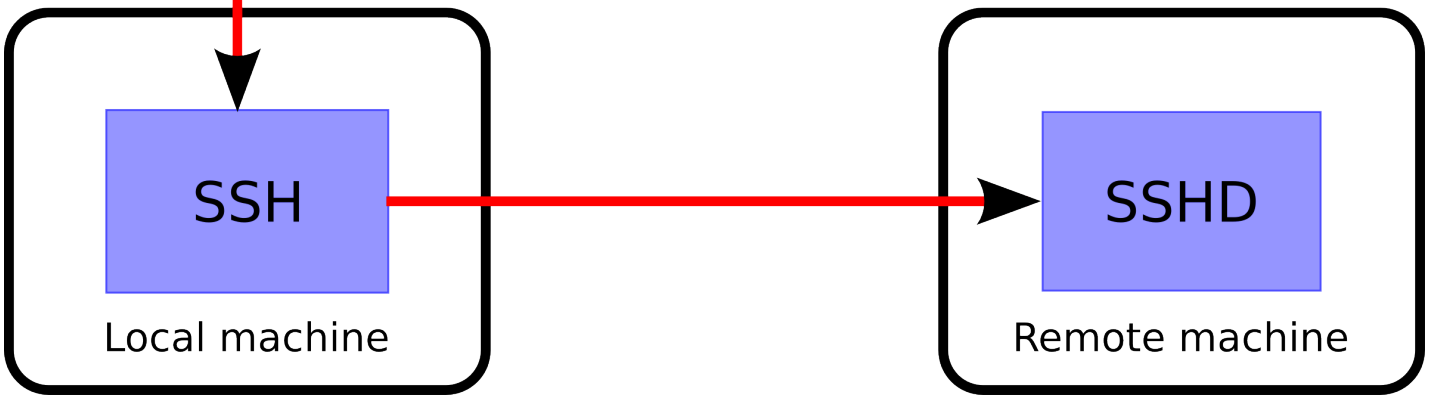
ssh asks for the password
on the remote machine





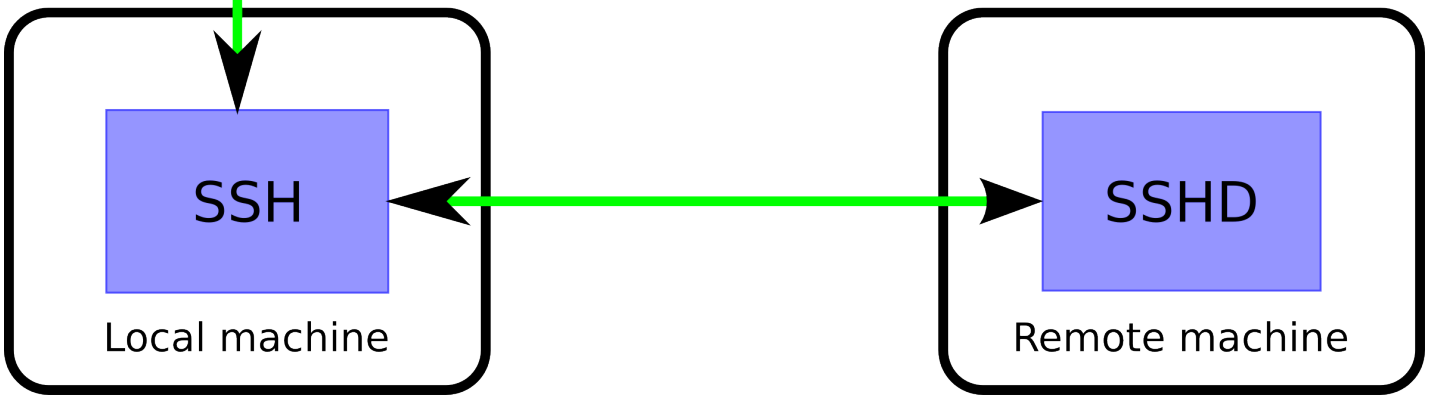
The SSH Client and Server

The user enters the password
that via a secure connection
reach the remote machine



The SSH Client and Server

The remote command fds are connected in a secure way to the local ssh process. The user can use the remote process as is it was local.



Using Keys

A couple of private/public keys may substitute password in this process, they can be created with `ssh-keygen -t dsa`

Using Keys

A couple of private/public keys may substitute password in this process, they can be created with `ssh-keygen -t dsa`

The private key is *id_dsa* and the relative public key is *id_dsa.pub*, the private one is usually protected with a passphrase asked at creation time.

Using Keys

A couple of private/public keys may substitute password in this process, they can be created with *ssh-keygen -t dsa*

The private key is *id_dsa* and the relative public key is *id_dsa.pub*, the private one is usually protected with a passphrase asked at creation time.

Placing *id_dsa.pub* within the file *$HOME/.ssh/authorized_keys* on the remote machine it is possible to implement a passwordless access (provided the passphrase ...).

Using Keys

A couple of private/public keys may substitute password in this process, they can be created with *ssh-keygen -t dsa*

The private key is *id_dsa* and the relative public key is *id_dsa.pub*, the private one is usually protected with a passphrase asked at creation time.

Placing *id_dsa.pub* within the file *$HOME/.ssh/authorized_keys* on the remote machine it is possible to implement a passwordless access (provided the passphrase ...).

... so what's the benefit ?

The OpenSSH agent

OpenSSH Agent

The SSH agent is a daemon that is able to store users private keys and let SSH use them whenever necessary.

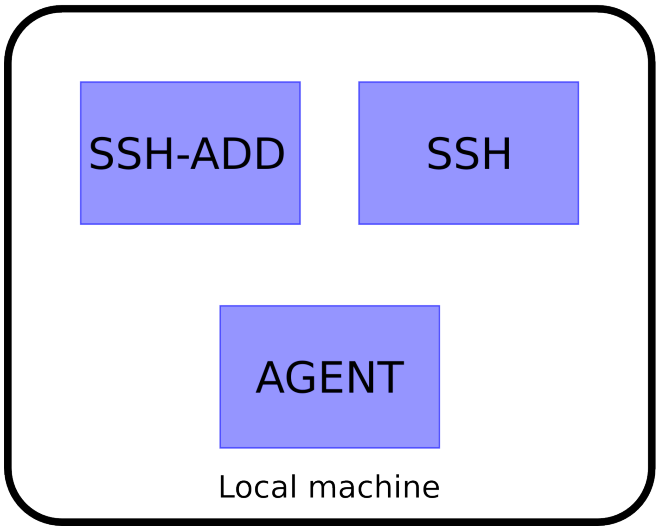
The OpenSSH agent

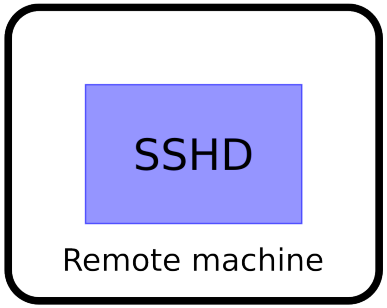
OpenSSH Agent

The SSH agent is a daemon that is able to store users private keys and let SSH use them whenever necessary.

The agent talks with its own client *ssh-add* to let the user make keys operations such as: listing, storing, deleting, etc.

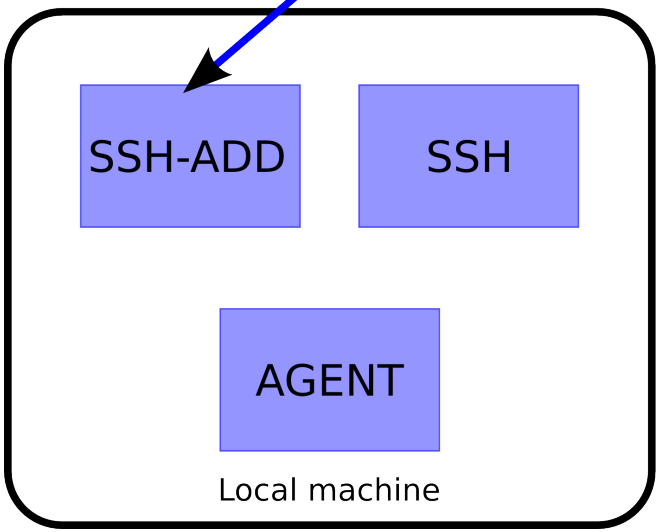
The OpenSSH agent

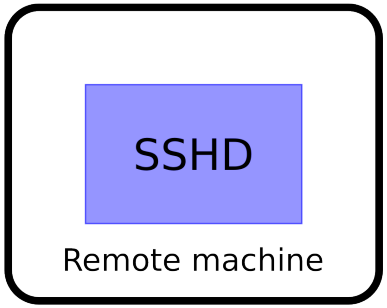




The OpenSSH agent

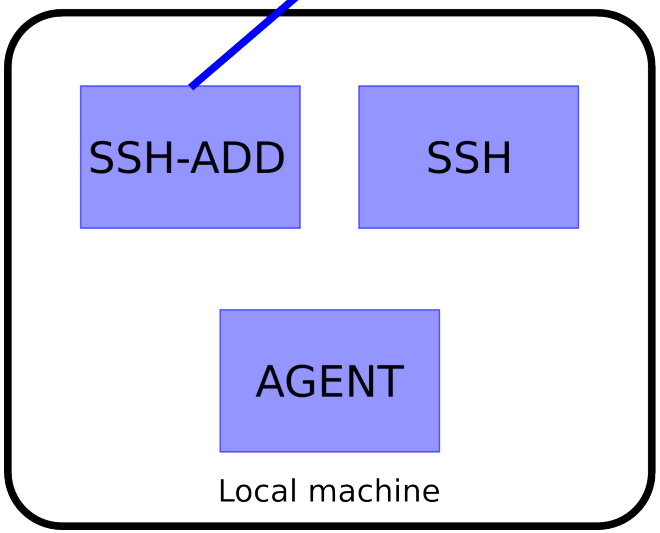
The user wants to connect with the remote machine and issues the ssh-add command to store his key within the agent

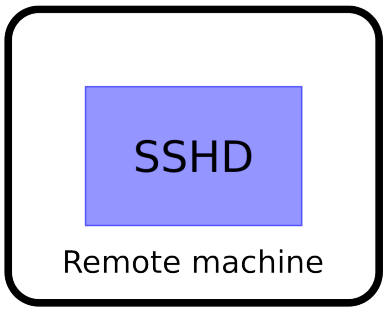




The OpenSSH agent

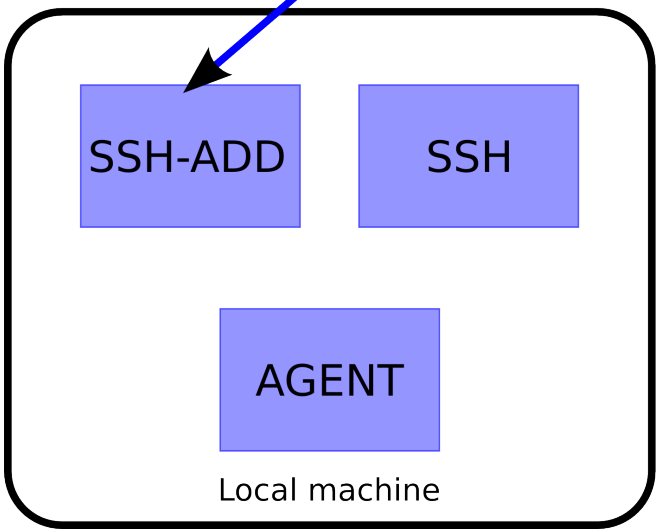
ssh-add asks for the passphrase needed to decrypt the private key

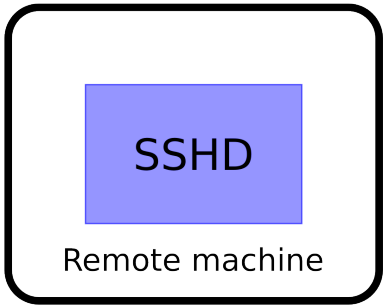




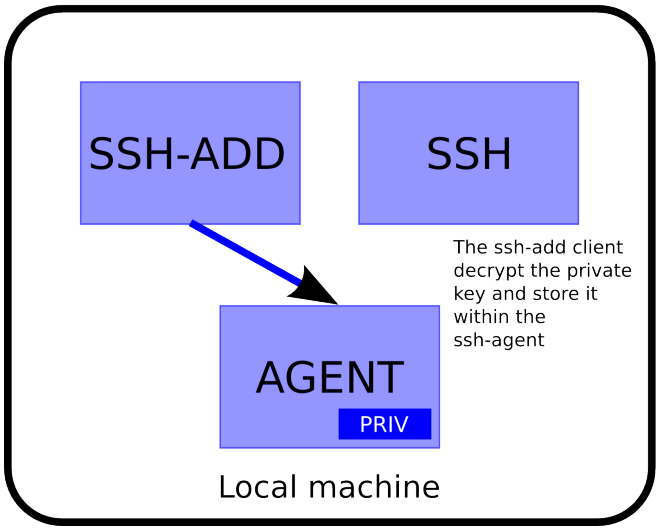
The OpenSSH agent

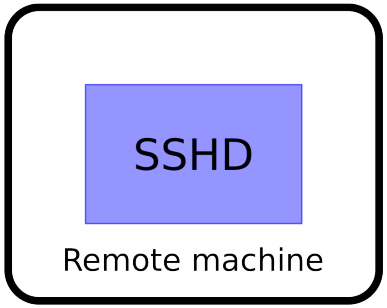
The user enters the passphrase





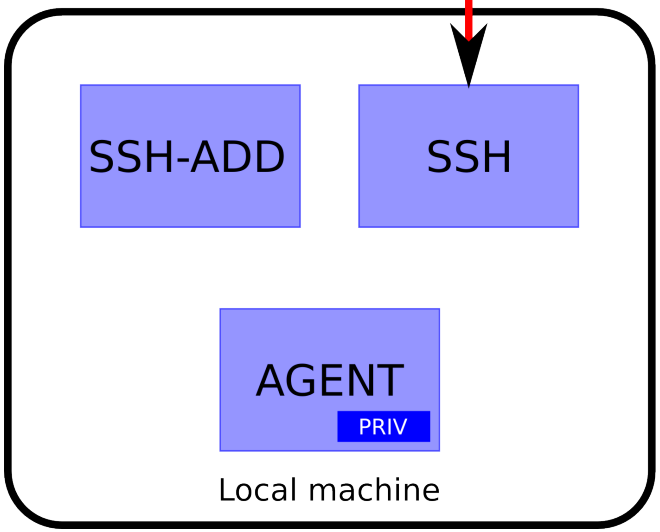
The OpenSSH agent

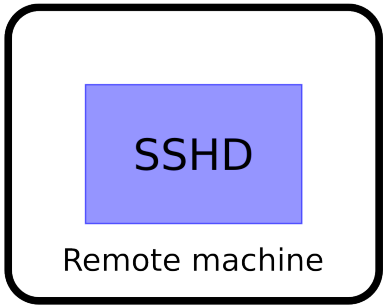




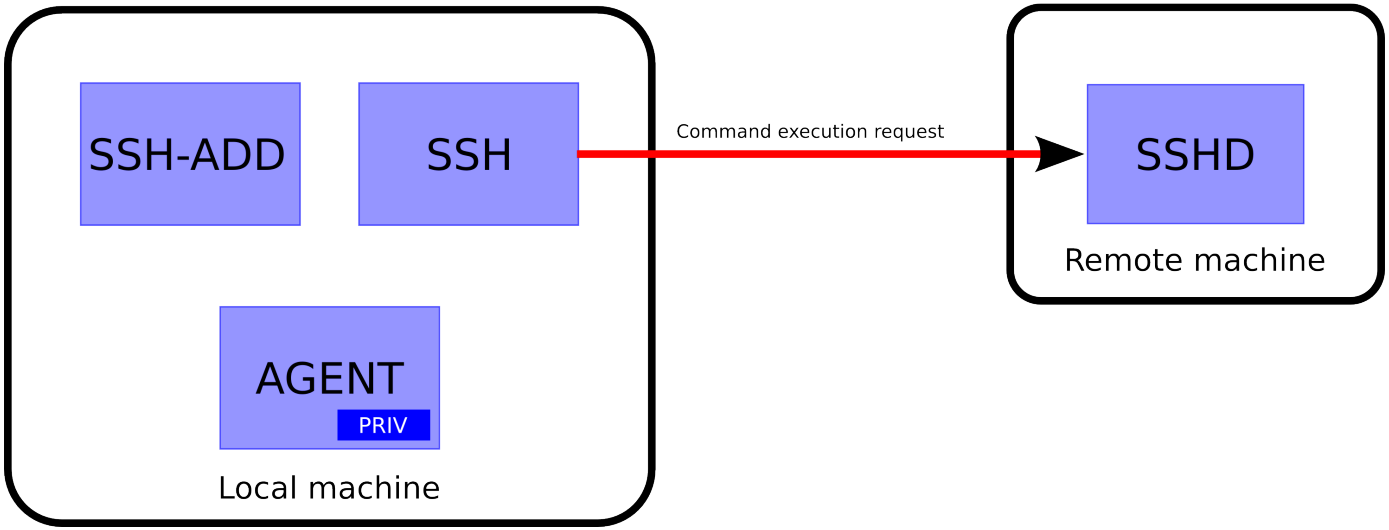
The OpenSSH agent

The user use the ssh command
to connect a remote machine

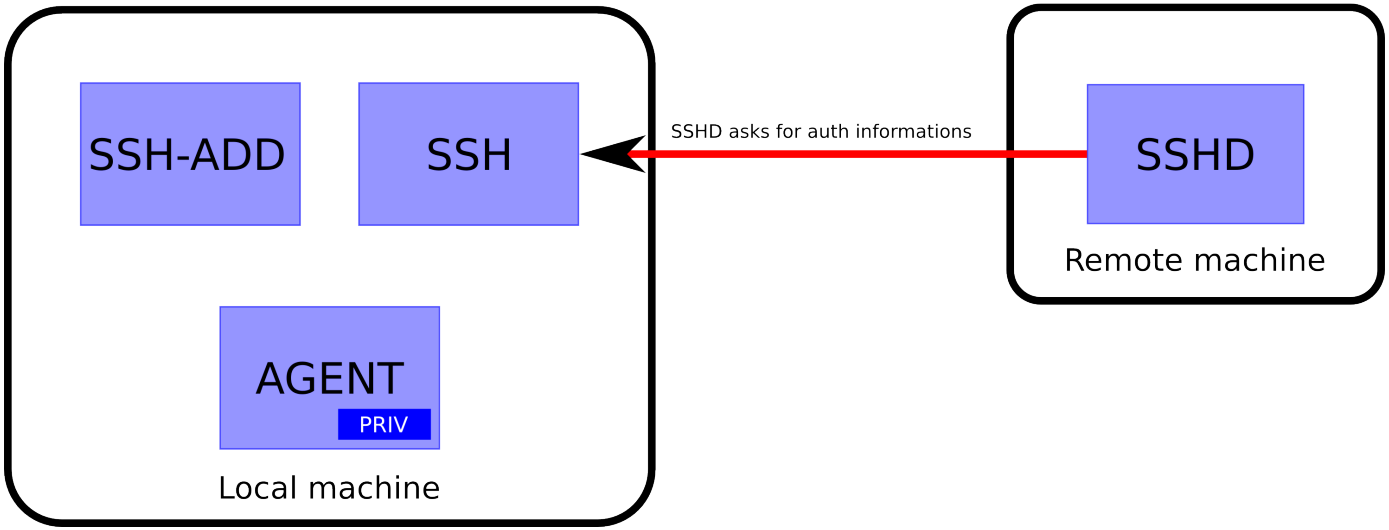




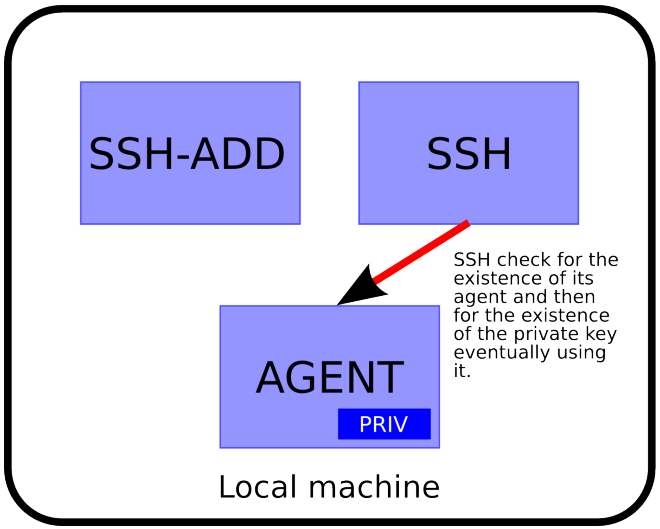
The OpenSSH agent

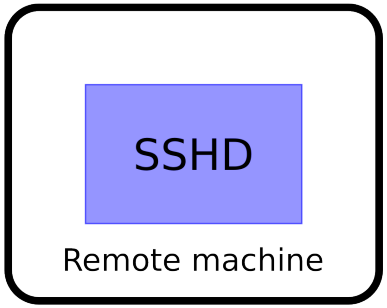


The OpenSSH agent

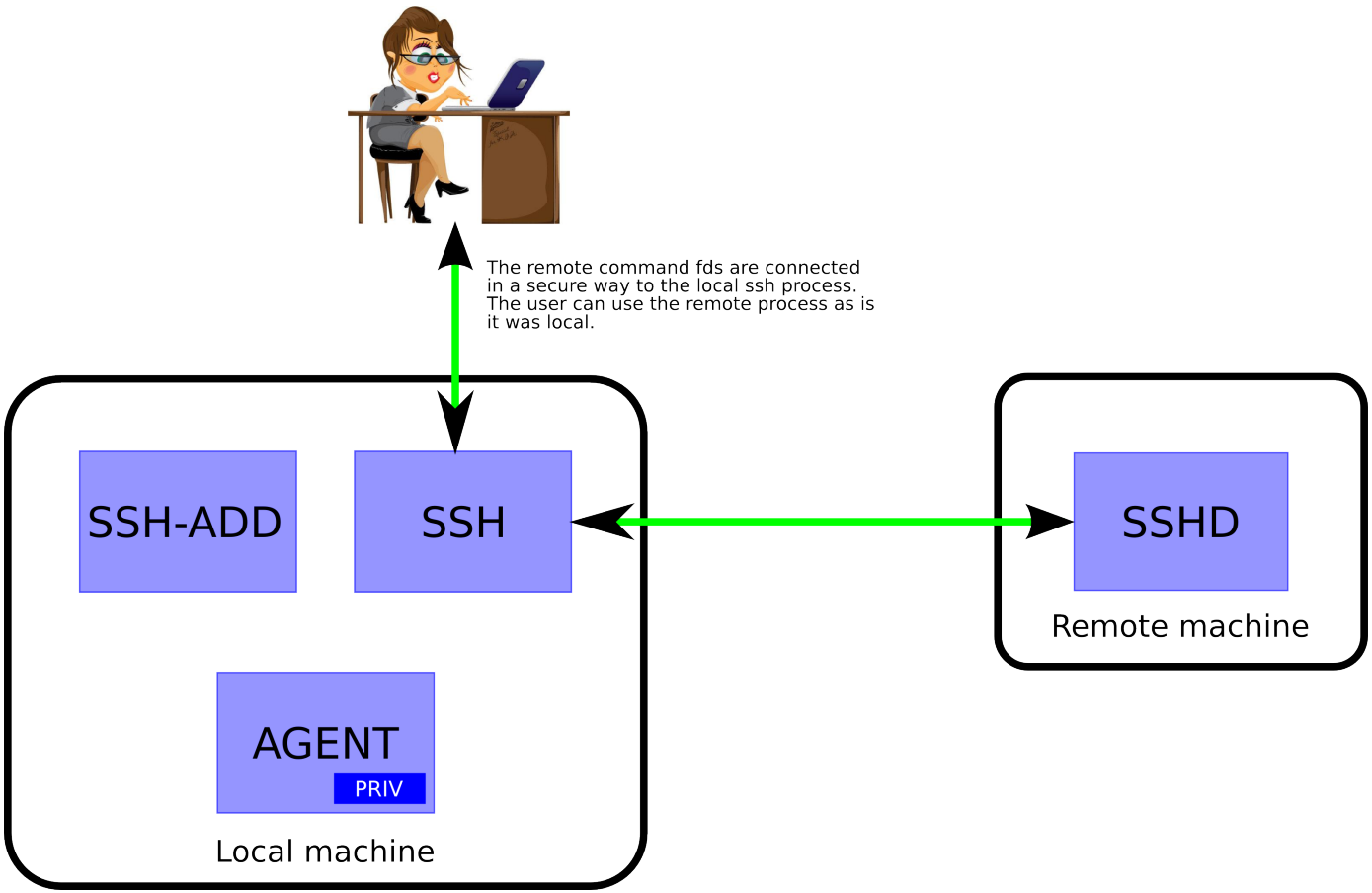


The OpenSSH agent





The OpenSSH agent



Start of the agent

There are basically two ways of starting the agent:

Start of the agent

There are basically two ways of starting the agent:

- Upon starting the agent goes in background and write to stdout its environment parameters in the form of shell commands. so it can be launched from a shell this way:
  eval ‘ssh-agent‘

Start of the agent

There are basically two ways of starting the agent:

- Upon starting the agent goes in background and write to stdout its environment parameters in the form of shell commands. so it can be launched from a shell this way: *eval ‘ssh-agent‘*
- The agent itself may spawn a command and export to it its own environment: *ssh-agent [command]*

Agent options

When stating the agent several options may be specified via command line arguments:

Agent options

When stating the agent several options may be specified via command line arguments:

- What kind of shell is the environment commands target.

Agent options

When stating the agent several options may be specified via command line arguments:

- What kind of shell is the environment commands target.
- The default duration of a stored key: *ssh-agent -t [n]*, so after n seconds the key will expire and removed from the agent.

Agent options

When stating the agent several options may be specified via command line arguments:

- What kind of shell is the environment commands target.
- The default duration of a stored key: *ssh-agent -t [n]*, so after n seconds the key will expire and removed from the agent.
- The agent may be closed with: *ssh-agent -k*

Managing the agent's keys

ssh-add is the agent's client. The two of them communicate with a UNIX socket and ssh-add manage keys stored within the ssh-agent deamon via command line arguments. Several operation are possible:

Managing the agent's keys

ssh-add is the agent's client. The two of them communicate with a UNIX socket and ssh-add manage keys stored within the ssh-agent deamon via command line arguments. Several operation are possible:

- Add an identity (a pair of keys) to the agent:
  ssh-add [filename]

Managing the agent's keys

ssh-add is the agent's client. The two of them communicate with a UNIX socket and ssh-add manage keys stored within the ssh-agent deamon via command line arguments. Several operation are possible:

- Add an identity (a pair of keys) to the agent:
  ssh-add [filename]
- Lock and unlock the agent: *ssh-add -x [-X]*

Managing the agent's keys

ssh-add is the agent's client. The two of them communicate with a UNIX socket and ssh-add manage keys stored within the ssh-agent deamon via command line arguments. Several operation are possible:

- Add an identity (a pair of keys) to the agent: *ssh-add [filename]*
- Lock and unlock the agent: *ssh-add -x [-X]*
- List the fingerprints or the public key parameters of the managed identities: *ssh-add -l [-L]*

Managing the agent's keys

ssh-add is the agent's client. The two of them communicate with a UNIX socket and ssh-add manage keys stored within the ssh-agent deamon via command line arguments. Several operation are possible:

- Add an identity (a pair of keys) to the agent: *ssh-add [filename]*
- Lock and unlock the agent: *ssh-add -x [-X]*
- List the fingerprints or the public key parameters of the managed identities: *ssh-add -l [-L]*
- Remove an identity or all: *ssh-add -d [filename] [-D]*

Sharing keys

ssh provide a way to forward automatically keys from an endpoint to another:

ssh -o ForwardAgent=yes remoteuser@remoteendpoint

Sharing keys

ssh provide a way to forward automatically keys from an endpoint to another:

ssh -o ForwardAgent=yes remoteuser@remoteendpoint

it will implicitly set the agent endpoint on the remote system, and set the authentication environment to the called program.

Sharing keys

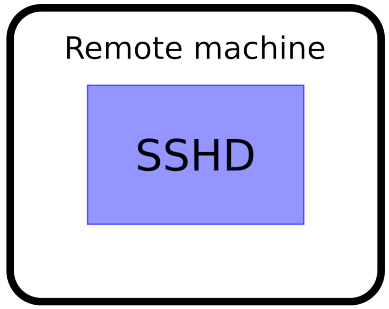
ssh provide a way to forward automatically keys from an endpoint to another:

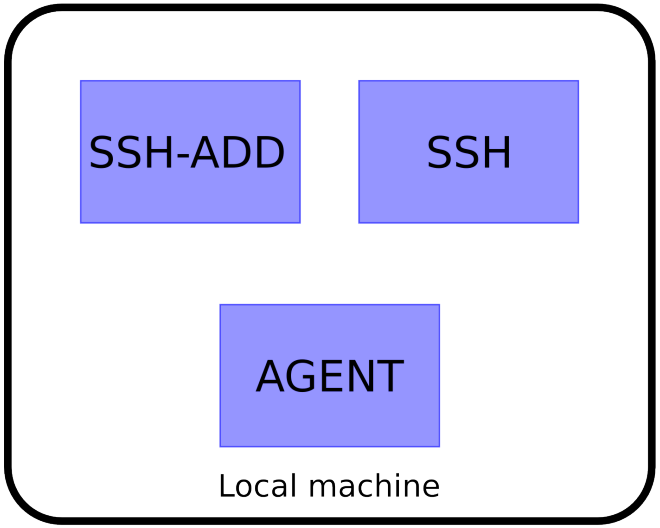
ssh -o ForwardAgent=yes remoteuser@remoteendpoint

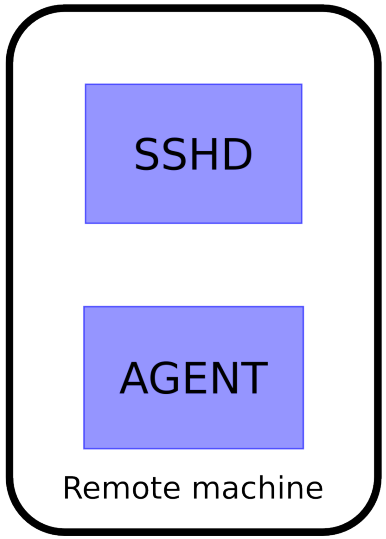
it will implicitly set the agent endpoint on the remote system, and set the authentication environment to the called program.

The remote program may use the authentication information.

Sharing keys

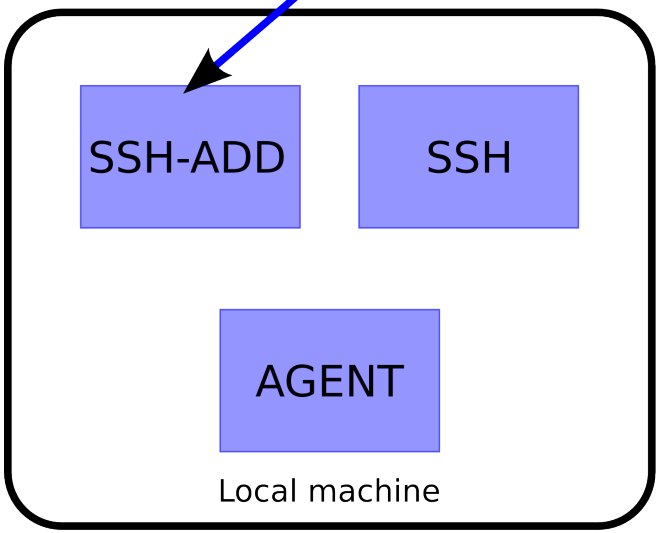


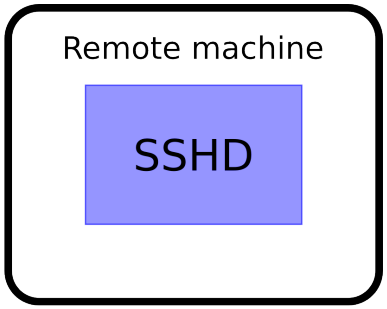


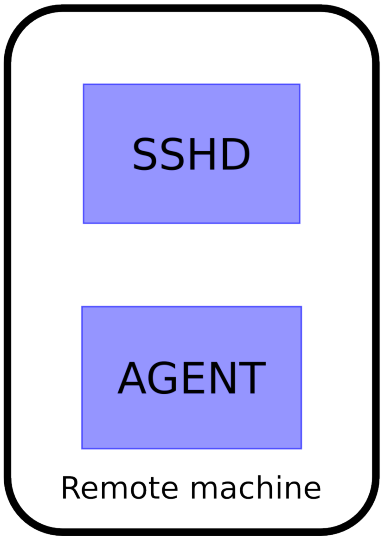


Sharing keys

The user wants to connect with the remote machine and issues the ssh-add command to store his key within the agent







Sharing keys

ssh-add asks for the passphrase needed to decrypt the private key

Remote machine

SSHD

SSH-ADD

SSH

AGENT

Local machine

SSHD

AGENT

Remote machine

Sharing keys

The user enters the passphrase

Remote machine

SSHD

SSH-ADD

SSH

AGENT

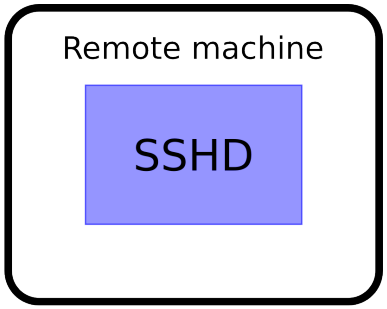
Local machine

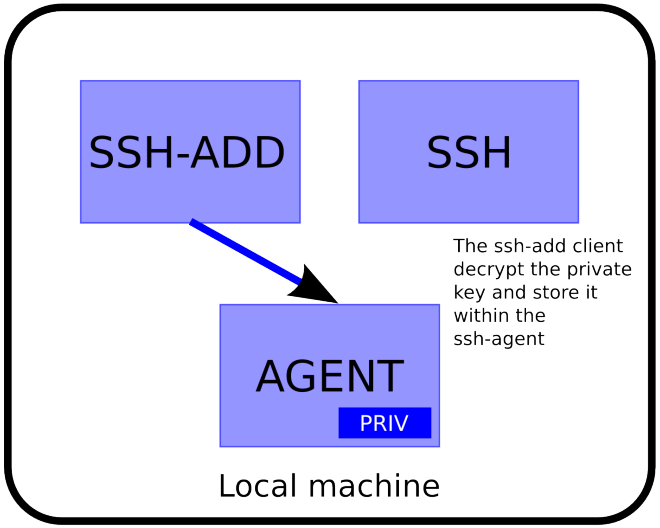
SSHD

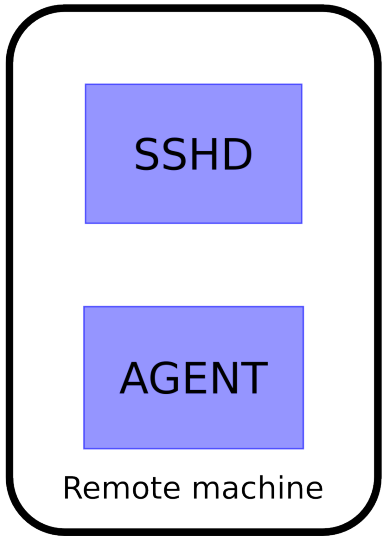
AGENT

Remote machine

Sharing keys







Sharing keys

The user use the ssh command
to connect a remote machine
with the forwarding options

Remote machine

SSHD

SSH-ADD

SSH

AGENT

PRIV

Local machine

SSHD

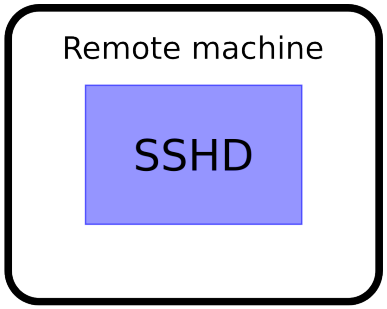
AGENT

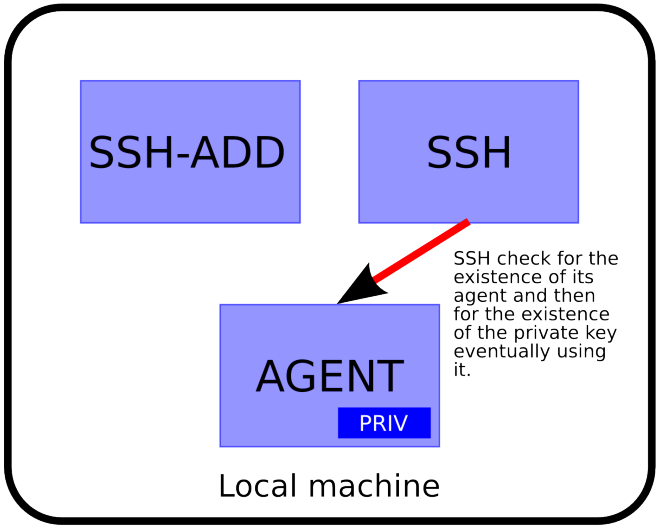
Remote machine

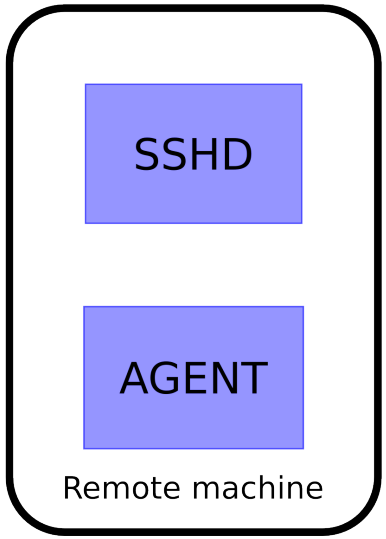
Sharing keys

Sharing keys

Sharing keys

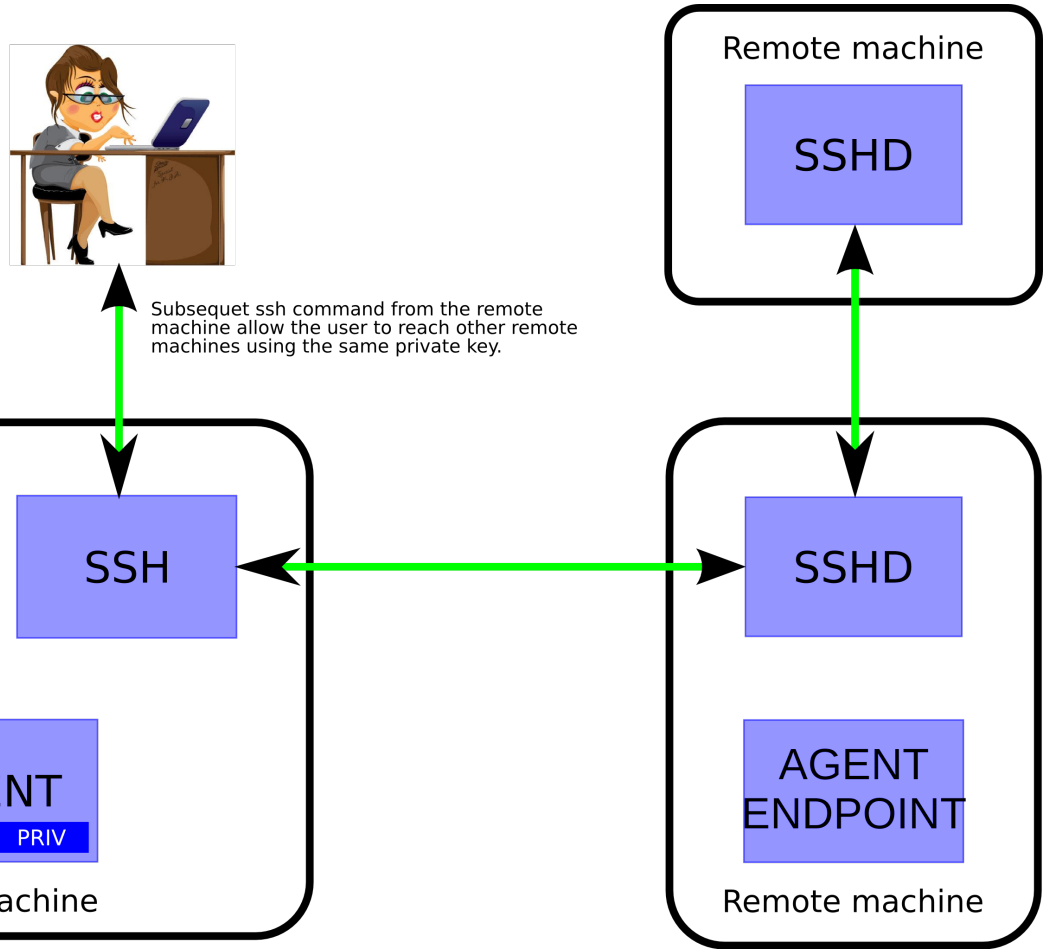






Sharing keys

Sharing keys



Some notes about security

As all ssh operations, an agent is not meant to be used in an insecure system. Indeed gain the access to the authentication socket opened by an agent gives an attacker access to all the systems protected by those keys.

Some notes about security

As all ssh operations, an agent is not meant to be used in an insecure system. Indeed gain the access to the authentication socket opened by an agent gives an attacker access to all the systems protected by those keys.

In the case of a forwarding agent among different system, remote machines can be insecure since the private keys are always stored on the agent of the first (secure) machine.

What is this work about

The agent is a powerfull tool that allows an easy and secure connection among machines via the ssh protocol, with it keys may be moved and used from a system to another ...

What is this work about

The agent is a powerfull tool that allows an easy and secure connection among machines via the ssh protocol, with it keys may be moved and used from a system to another ...
... But its use is specific to the ssh context.

What is this work about

The agent is a powerfull tool that allows an easy and secure connection among machines via the ssh protocol, with it keys may be moved and used from a system to another ...
... But its use is specific to the ssh context.
What if we want it to be a more general tool? If we want for example use a user's private key to create security tickets or to crypt and decrypt data with a symmetric algorithm.

What is this work about

The agent is a powerfull tool that allows an easy and secure connection among machines via the ssh protocol, with it keys may be moved and used from a system to another ...
... But its use is specific to the ssh context.
What if we want it to be a more general tool? If we want for example use a user's private key to create security tickets or to crypt and decrypt data with a symmetric algorithm.
To do so, several modification is to be made to the openssh code.

What is this work about

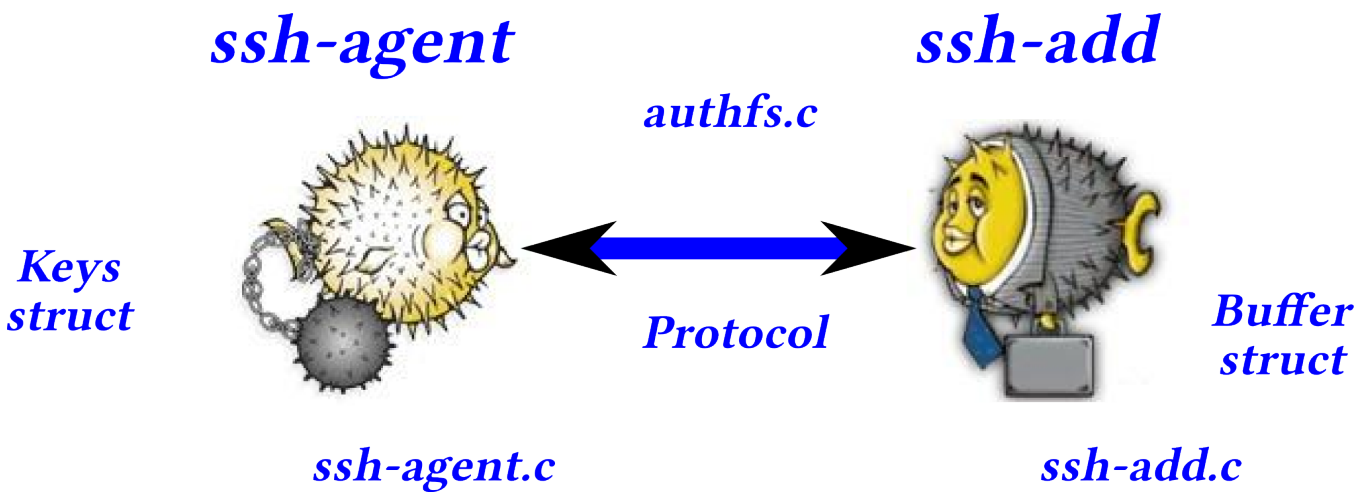
The agent is a powerfull tool that allows an easy and secure connection among machines via the ssh protocol, with it keys may be moved and used from a system to another ...
... But its use is specific to the ssh context.
What if we want it to be a more general tool? If we want for example use a user's private key to create security tickets or to crypt and decrypt data with a symmetric algorithm.
To do so, several modification is to be made to the openssh code.

This work is about ...

... the modifications made by me to reach this goal.

OpenSSH agent internals

Before going any further it is necessary to take a look at the internal behaviour of the ssh-agent and the ssh-add.



Interacting with the agent

When the agent starts opens a UNIX socket that is the only interaction method possible, the socket location is generated by the agent as shell commands:

Interacting with the agent

When the agent starts opens a UNIX socket that is the only interaction method possible, the socket location is generated by the agent as shell commands:

```
$ ssh-agent
SSH_AUTH_SOCK=/tmp/ssh-aOHLwBp25411/agent.25411; export SSH_AUTH_SOCK;
SSH_AGENT_PID=25412; export SSH_AGENT_PID;
echo Agent pid 25412;
```

Interacting with the agent

When the agent starts opens a UNIX socket that is the only interaction method possible, the socket location is generated by the agent as shell commands:

```
$ ssh-agent
SSH_AUTH_SOCK=/tmp/ssh-aOHLwBp25411/agent.25411; export SSH_AUTH_SOCK;
SSH_AGENT_PID=25412; export SSH_AGENT_PID;
echo Agent pid 25412;
```

Two environment variable *SSH_AUTH_SOCK* and *SSH_AGENT_PID* are generated. If a program of the OpenSSH suite find these variables it will try to talk with the agent via the socket.

The buffer struct

All the communication are done via a struct build to manipulate fifo buffers that can grow if needed. It is the buffer struct:

The buffer struct

All the communication are done via a struct build to manipulate fifo buffers that can grow if needed. It is the buffer struct:

The buffer struct

```
typedef struct {
   u_char   *buf;     /* Buffer for data. */
   u_int  alloc;      /* Number of bytes allocated for data. */
   u_int  offset; /* Offset of first byte containing data. */
   u_int  end;     /* Offset of last byte containing data. */
}       Buffer;
```

The buffer struct

All the communication are done via a struct build to manipulate fifo buffers that can grow if needed. It is the buffer struct:

The buffer struct

```
typedef struct {
   u_char   *buf;     /* Buffer for data. */
   u_int  alloc;      /* Number of bytes allocated for data. */
   u_int  offset; /* Offset of first byte containing data. */
   u_int  end;     /* Offset of last byte containing data. */
}       Buffer;
```

Several functions are included to manipulate buffers such as create, free, add new data, etc.

The buffer struct

All the communication are done via a struct build to manipulate fifo buffers that can grow if needed. It is the buffer struct:

The buffer struct

```
typedef struct {
   u_char   *buf;     /* Buffer for data. */
   u_int  alloc;      /* Number of bytes allocated for data. */
   u_int  offset; /* Offset of first byte containing data. */
   u_int  end;     /* Offset of last byte containing data. */
}       Buffer;
```

Several functions are included to manipulate buffers such as create, free, add new data, etc.
The agent and its client exchange buffers within the UNIX socket.

Agent Protocol

A single interaction beetween *ssh-add* and *ssh-agent* is in the form of a request-response query.

Agent Protocol

A single interaction beetween *ssh-add* and *ssh-agent* is in the form of a request-response query.

Each query starts with a byte that we may call *request ID* followed by all the data needed to process the request.
Each response starts with a *response ID* byte followed by the response data. Some generic failure and success response exist.

Agent Protocol

A single interaction beetween *ssh-add* and *ssh-agent* is in the form of a request-response query.

Each query starts with a byte that we may call *request ID* followed by all the data needed to process the request.
Each response starts with a *response ID* byte followed by the response data. Some generic failure and success response exist.

The protocol is described in the *PROTOCOL.agent* file while *authfd.h* has all the *request ID* and *response ID* in the form of C constants.

Agent Protocol

A single interaction beetween *ssh-add* and *ssh-agent* is in the form of a request-response query.

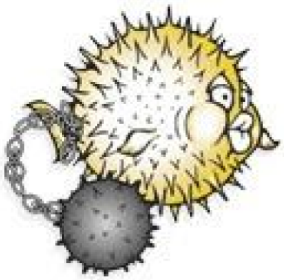
Each query starts with a byte that we may call *request ID* followed by all the data needed to process the request.
Each response starts with a *response ID* byte followed by the response data. Some generic failure and success response exist.

The protocol is described in the *PROTOCOL.agent* file while *authfd.h* has all the *request ID* and *response ID* in the form of C constants.

The message maximum lenght is a constant so it may be necessary to have multiple request-response for a single *ssh-add* invocation. A function that we may call request manager can accomplish this task.

The Request Process

ssh-agent

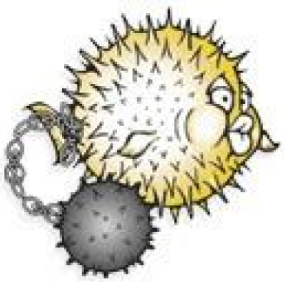


UNIX Socket

ssh-add

The Request Process

ssh-agent



UNIX Socket

ssh-add

Source: ssh-add.c

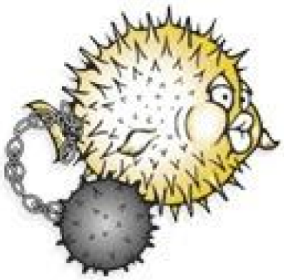
Function: main

parsing of the command line arguments

according to arguments a request manager function is called

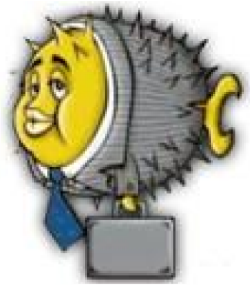
The Request Process

ssh-agent



UNIX Socket

ssh-add



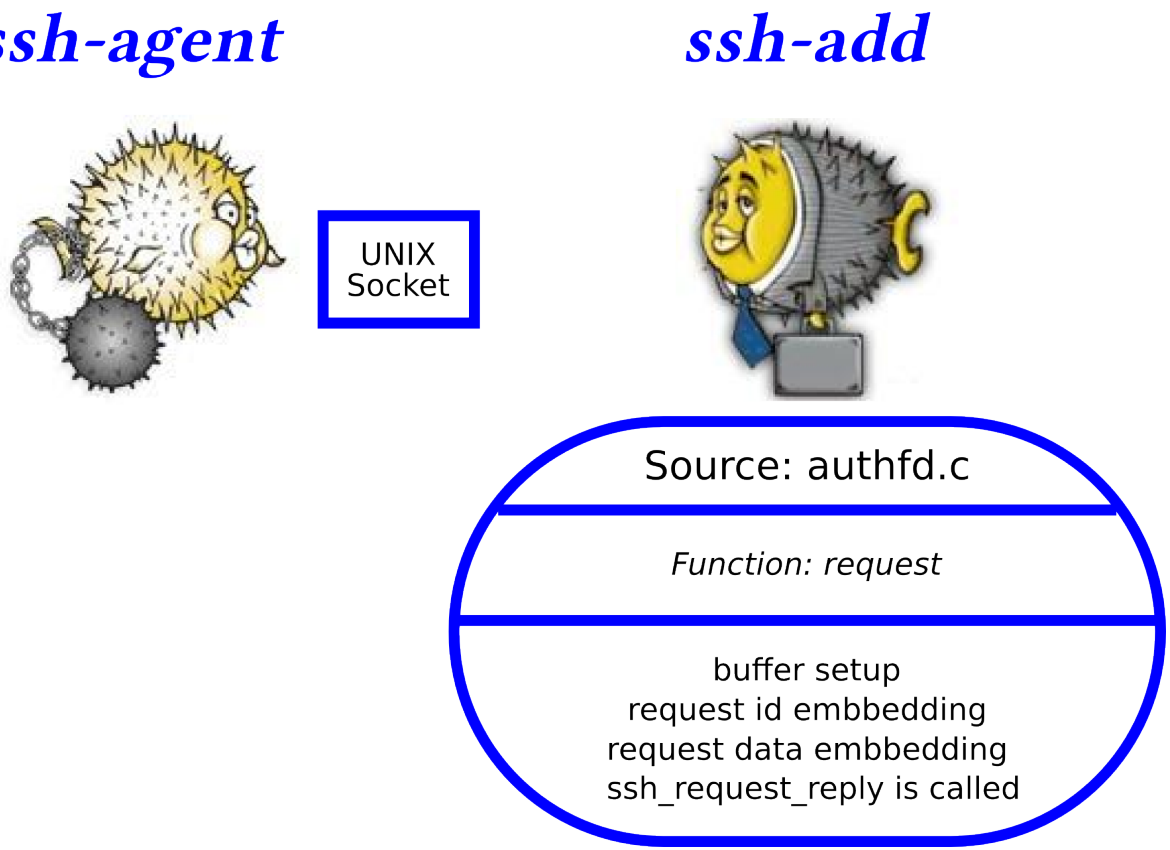
Source: ssh-add.c

Function: request manager

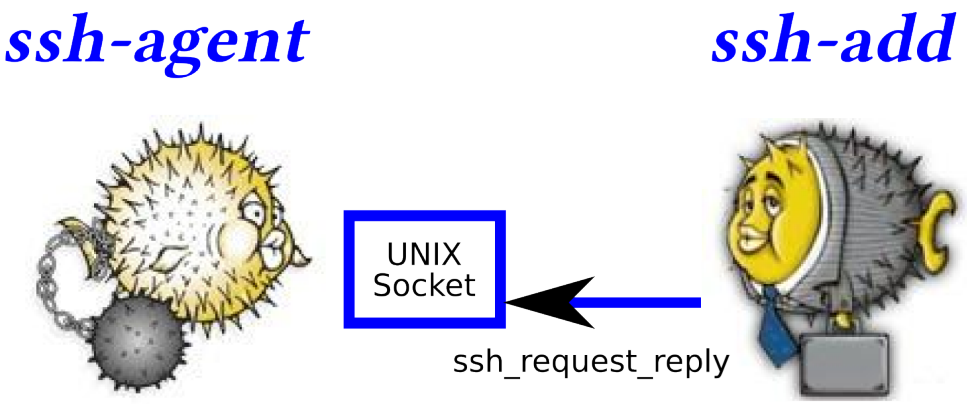
identity processing and file handling

a request function is called

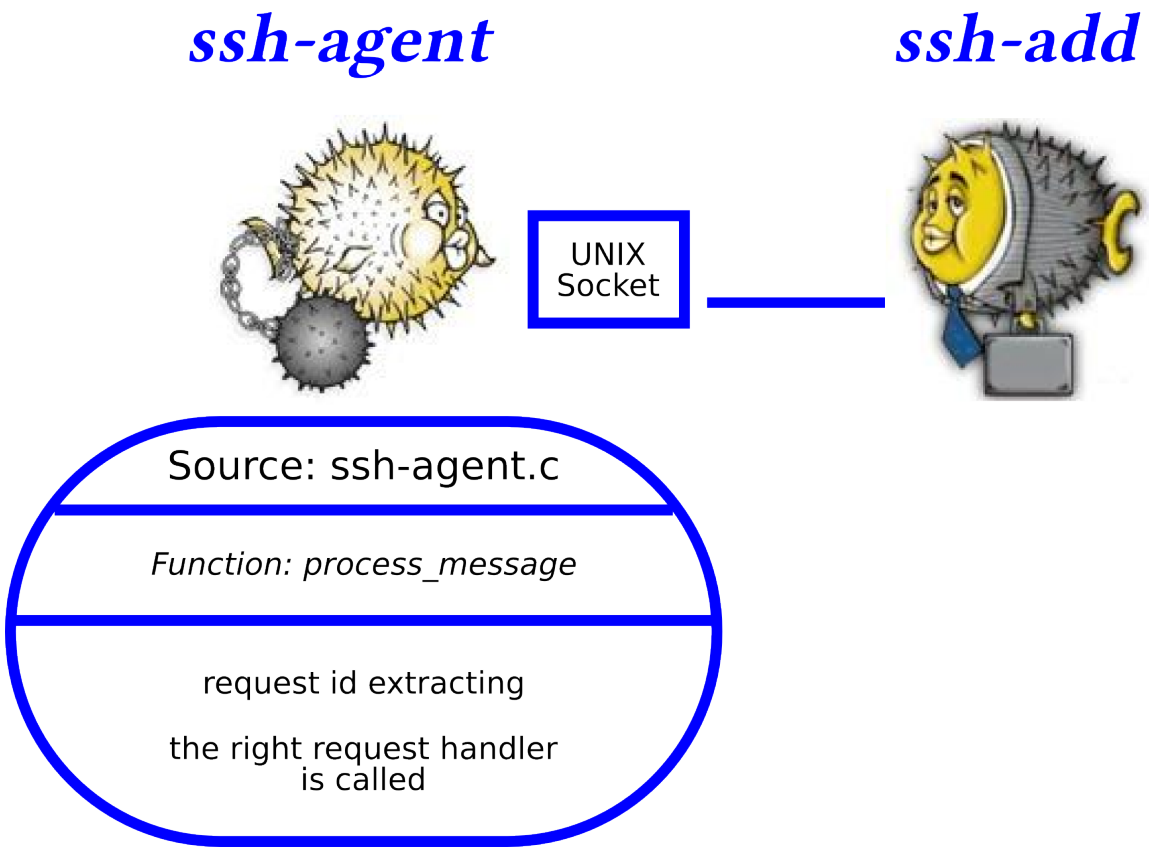
The Request Process



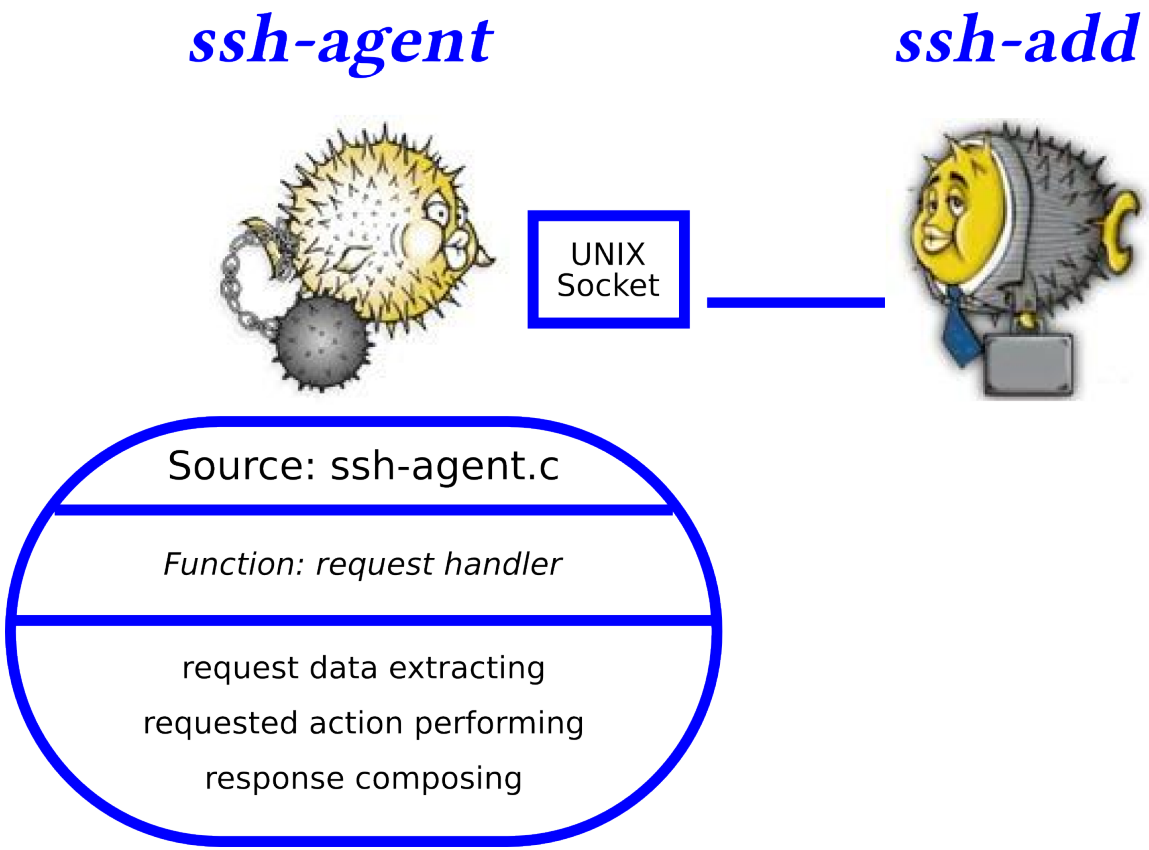
The Request Process



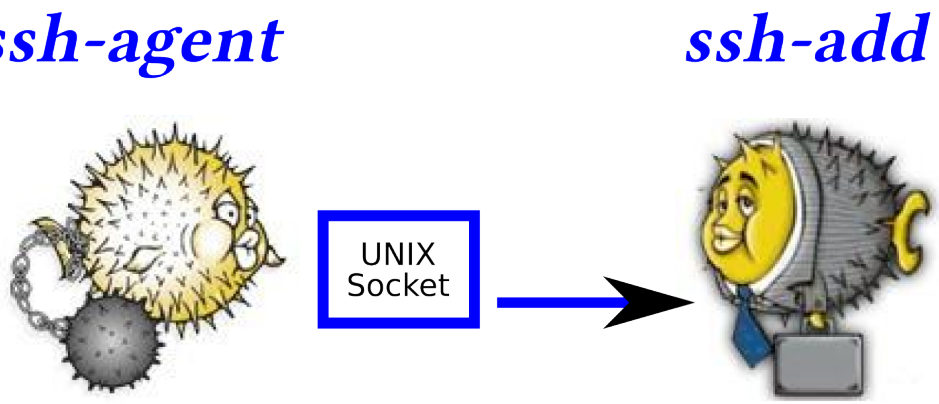
The Request Process



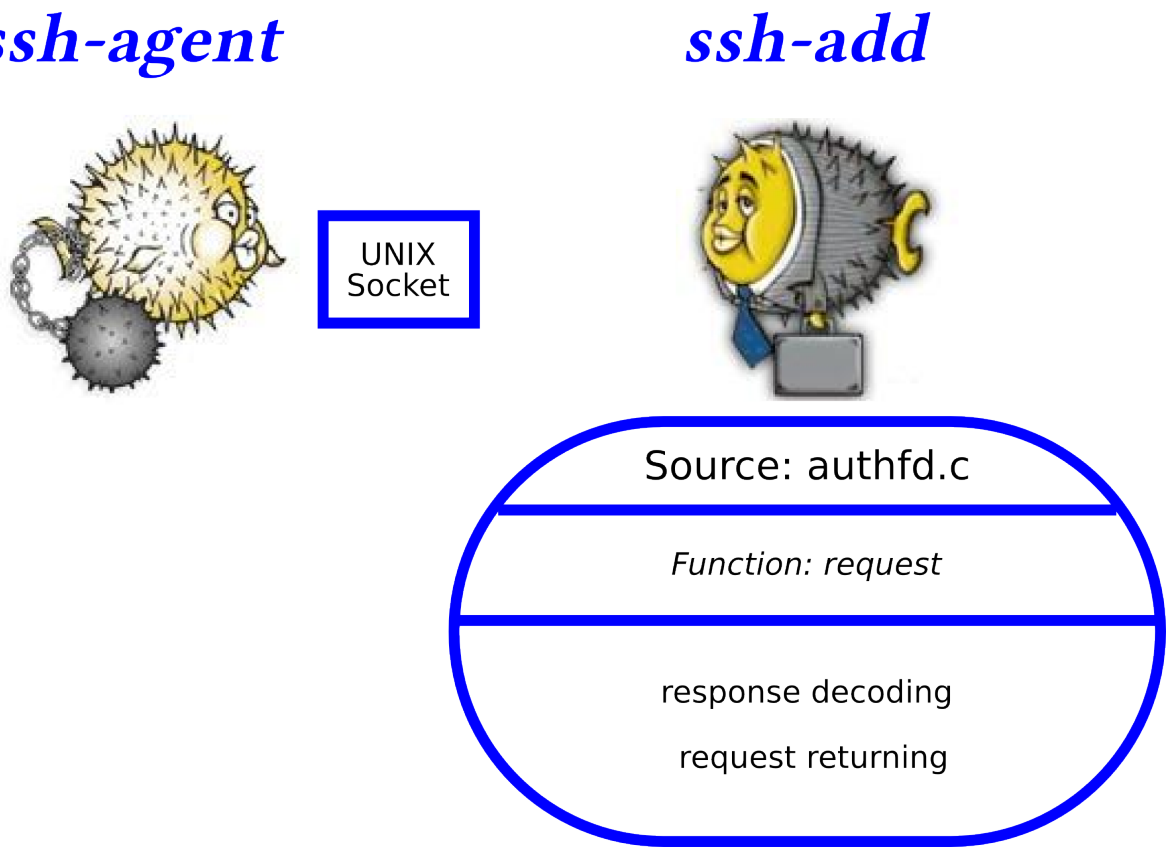
The Request Process



The Request Process

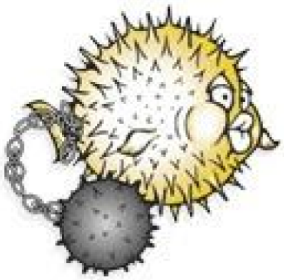


The Request Process



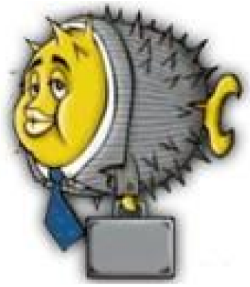
The Request Process

ssh-agent



UNIX Socket

ssh-add



Source: ssh-add.c

Function: request manager

Returned data used to do the asked thing

return to main ad exits

Identities

The agent store its managed keys in a linked list queue of structs called Identity.

Identities

The agent store its managed keys in a linked list queue of structs called Identity.

The identity struct

```
typedef struct identity {
        TAILQ_ENTRY(identity) next;
        Key *key;
        char *comment;
        char *provider;
        u_int death;
        u_int confirm;
} Identity;
```

Identities

The agent store its managed keys in a linked list queue of structs called Identity.

The identity struct

```
typedef struct identity {
        TAILQ_ENTRY(identity) next;
        Key *key;
        char *comment;
        char *provider;
        u_int death;
        u_int confirm;
} Identity;
```

Each Identity stores:

Identities

The agent store its managed keys in a linked list queue of structs called Identity.

The identity struct

```
typedef struct identity {
        TAILQ_ENTRY(identity) next;
        Key *key;
        char *comment;
        char *provider;
        u_int death;
        u_int confirm;
} Identity;
```

Each Identity stores:

- The effective keys.

Identities

The agent store its managed keys in a linked list queue of structs called Identity.

The identity struct

```
typedef struct identity {
        TAILQ_ENTRY(identity) next;
        Key *key;
        char *comment;
        char *provider;
        u_int death;
        u_int confirm;
} Identity;
```

Each Identity stores:

- The effective keys.
- The identity lifetime.

Identities

The agent store its managed keys in a linked list queue of structs called Identity.

The identity struct

```
typedef struct identity {
        TAILQ_ENTRY(identity) next;
        Key *key;
        char *comment;
        char *provider;
        u_int death;
        u_int confirm;
} Identity;
```

Each Identity stores:

- The effective keys.
- The identity lifetime.
- A description.

Identities

The agent store its managed keys in a linked list queue of structs called Identity.

The identity struct

```
typedef struct identity {
        TAILQ_ENTRY(identity) next;
        Key *key;
        char *comment;
        char *provider;
        u_int death;
        u_int confirm;
} Identity;
```

Each Identity stores:

- The effective keys.
- The identity lifetime.
- A description.
- Some other optional parameter.

Keys

The keys are stored in a struct called Key and defined in key.h

Keys

The keys are stored in a struct called Key and defined in key.h

The Key struct

```
struct Key {
        int       type;
        int       flags;
        RSA      *rsa;
        DSA      *dsa;
        int       ecdsa_nid;     /* NID of curve */
#ifdef OPENSSL_HAS_ECC
        EC_KEY   *ecdsa;
#else
        void     *ecdsa;
#endif
        struct KeyCert *cert;
};
```

Keys

The keys are stored in a struct called Key and defined in key.h

The Key struct

```
struct Key {
        int       type;
        int       flags;
        RSA      *rsa;
        DSA      *dsa;
        int       ecdsa_nid;     /* NID of curve */
#ifdef OPENSSL_HAS_ECC
        EC_KEY   *ecdsa;
#else
        void     *ecdsa;
#endif
        struct KeyCert *cert;
};
```

The function to manage the keys are within the *openssl* library.

DSA Keys

The DSA keys are defined in the *openssl* library.

DSA Keys

The DSA keys are defined in the *openssl* library.

The DSA Key struct

```
struct dsa_st
{
    int pad;
    long version;
    int write_params;
    BIGNUM *p;
    BIGNUM *q;       /* == 20 */
    BIGNUM *g;
    BIGNUM *pub_key;  /* y public key */
    BIGNUM *priv_key; /* x private key */

    BIGNUM *kinv;    /* Signing pre-calc */
    BIGNUM *r;       /* Signing pre-calc */

    int flags;

    // ...
};
```

DSA Keys

The DSA keys are defined in the *openssl* library.

The DSA Key struct

```
struct dsa_st
{
    int pad;
    long version;
    int write_params;
    BIGNUM *p;
    BIGNUM *q;      /* == 20 */
    BIGNUM *g;
    BIGNUM *pub_key;  /* y public key */
    BIGNUM *priv_key; /* x private key */

    BIGNUM *kinv;   /* Signing pre-calc */
    BIGNUM *r;      /* Signing pre-calc */

    int flags;

    // ...
};
```

This is the type of key used in this work, in particular *priv_key* is the DSA private key.

Hacks overview

These are the improvement made:

Hacks overview

These are the improvement made:

- Added a ssh-add command option to get the amount of time (in seconds) that the key will stay on agent.

Hacks overview

These are the improvement made:

- Added a ssh-add command option to get the amount of time (in seconds) that the key will stay on agent.
- Added a feature that allows to take a file and make an hash out of it after having merged it with the private key (create a sort of ticket).

Hacks overview

These are the improvement made:

- Added a ssh-add command option to get the amount of time (in seconds) that the key will stay on agent.
- Added a feature that allows to take a file and make an hash out of it after having merged it with the private key (create a sort of ticket).
- Added commands to crypt and decrypt files with a symmetric key derived from those stored in the agent.

Common Traits

For every one of these the modification made are similar, expecially regarding the protocol extension, the add of command line arguments, the creation of new request functions and handlers.

Common Traits - Protocol

Every request need to have a unique request id and response id, within *authfd.h* the defines for this ids ha to be added.

Common Traits - Protocol

Every request need to have a unique request id and response id, within *authfd.h* the defines for this ids ha to be added.

The *process_message* function within *ssh-agent.c* has to be modified to handle the new ids an point to the handlers functions.

Common Traits - Protocol

Every request need to have a unique request id and response id, within *authfd.h* the defines for this ids ha to be added.

The *process_message* function within *ssh-agent.c* has to be modified to handle the new ids an point to the handlers functions.

process_message function in ssh-agent.c

```
        switch (type) {
// ...
        case SSH2_AGENTC_REQUEST_TIMELEFT:
                process_request_timeleft(e);
                break;
        case SSH2_AGENTC_REQUEST_CRYPT:
                process_request_crypt(e);
                break;
        case SSH2_AGENTC_REQUEST_DIGEST:
                process_request_digest(e);
                break;
// ...
```

Common Traits - Client Arguments

Every hack has to be invoked from *ssh-add* as command line arguments. That arguments has to be added to the source code of *ssh-add.c*

Common Traits - Client Arguments

Every hack has to be invoked from *ssh-add* as command line arguments. That arguments has to be added to the source code of *ssh-add.c*

The arguments has to be added to the main *getopt* call usign its conventions.

Common Traits - Client Arguments

Every hack has to be invoked from *ssh-add* as command line arguments. That arguments has to be added to the source code of *ssh-add.c*

The arguments has to be added to the main *getopt* call usign its conventions.

Main getopt operations

```
while ((ch = getopt(argc, argv, "m:z:Z:TklLcdDxXe:s:t:o:")) != -1) {
   switch (ch) {
      ...
      case 'T':
         if (list_times(ac) == -1)
      ...
}}
```

Common Traits - Client Arguments

Every hack has to be invoked from *ssh-add* as command line arguments. That arguments has to be added to the source code of *ssh-add.c*

The arguments has to be added to the main *getopt* call usign its conventions.

Main getopt operations

```
while ((ch = getopt(argc, argv, "m:z:Z:TklLcdDxXe:s:t:o:")) != -1) {
   switch (ch) {
      ...
      case 'T':
         if (list_times(ac) == -1)
      ...
}}
```

After the getopt operation each new argument has to be handled and the relative request manager function has to be called.

Common Traits - Request Manager

Every new functionality has its own manager function within *ssh-add.c*, the manager function prepare the data that will be sent to the real request (or to the real multiple requests if needed) and handle the responses.

Common Traits - Request Manager

Every new functionality has its own manager function within *ssh-add.c*, the manager function prepare the data that will be sent to the real request (or to the real multiple requests if needed) and handle the responses.

A function for every hack has to be included in *ssh-add.c*.

Common Traits - Request Manager

Every new functionality has its own manager function within *ssh-add.c*, the manager function prepare the data that will be sent to the real request (or to the real multiple requests if needed) and handle the responses.

A function for every hack has to be included in *ssh-add.c*.

manager function example

```
static int make_digest(AuthenticationConnection *ac,char * datas) {
   //declarations
   //external file handling
   key = ssh_get_first_identity(ac, &comment, 2); // Key handling
   while (ret != 0) { pos=0;
      while ((ret != 0)&&(pos<MAX_AGENT_BUFFER)) {
         ret=read(fileinp,filedata+pos,MAX_AGENT_BUFFER-pos);
         if (ret == -1 ) { fprintf(stderr,"Failed"); return -1; } else { pos=pos+ret; } }
      if (pos !=0) {
         ecode = ssh_agent_digest(ac,key,&digest, &slen, filedata, pos); // real request
         for(i=0;i<slen;i++) printf("%02x",*(digest+i)); // Hack purpose
         free(digest); } }
   return ret; }
```

Common Traits - Request

Several new request submit functions have to be writed in order to obtain new functionalities, these are declared in the file *authfd.h* and implemented in *authfd.c*.

Common Traits - Request

Several new request submit functions have to be writed in order to obtain new functionalities, these are declared in the file *authfd.h* and implemented in *authfd.c*.

Each function is the atomic block of the agent-client comunication, it runs on the client and is responsible for the making the sigle request and getting back the response.

Common Traits - Request

Several new request submit functions have to be writed in order to obtain new functionalities, these are declared in the file *authfd.h* and implemented in *authfd.c*.

Each function is the atomic block of the agent-client comunication, it runs on the client and is responsible for the making the sigle request and getting back the response.

Its standard sequence of actions are:

Common Traits - Request

Several new request submit functions have to be writed in order to obtain new functionalities, these are declared in the file *authfd.h* and implemented in *authfd.c*.

Each function is the atomic block of the agent-client comunication, it runs on the client and is responsible for the making the sigle request and getting back the response.

Its standard sequence of actions are:

- Initialize a buffer.

Common Traits - Request

Several new request submit functions have to be writed in order to obtain new functionalities, these are declared in the file *authfd.h* and implemented in *authfd.c*.

Each function is the atomic block of the agent-client comunication, it runs on the client and is responsible for the making the sigle request and getting back the response.

Its standard sequence of actions are:

- Initialize a buffer.
- Write the request ID.

Common Traits - Request

Several new request submit functions have to be writed in order to obtain new functionalities, these are declared in the file *authfd.h* and implemented in *authfd.c*.

Each function is the atomic block of the agent-client comunication, it runs on the client and is responsible for the making the sigle request and getting back the response.

Its standard sequence of actions are:

- Initialize a buffer.
- Write the request ID.
- Fill the buffer with the data needed to perform the requested operations.

Common Traits - Request

Several new request submit functions have to be writed in order to obtain new functionalities, these are declared in the file *authfd.h* and implemented in *authfd.c*.

Each function is the atomic block of the agent-client comunication, it runs on the client and is responsible for the making the sigle request and getting back the response.

Its standard sequence of actions are:

- Initialize a buffer.
- Write the request ID.
- Fill the buffer with the data needed to perform the requested operations.
- Send the request via a *ssh_request_reply* call.

Common Traits - Request

Several new request submit functions have to be writed in order to obtain new functionalities, these are declared in the file *authfd.h* and implemented in *authfd.c*.

Each function is the atomic block of the agent-client comunication, it runs on the client and is responsible for the making the sigle request and getting back the response.

Its standard sequence of actions are:

- Initialize a buffer.
- Write the request ID.
- Fill the buffer with the data needed to perform the requested operations.
- Send the request via a *ssh_request_reply* call.
- Check the response exit codes.

Common Traits - Request

Several new request submit functions have to be writed in order to obtain new functionalities, these are declared in the file *authfd.h* and implemented in *authfd.c*.

Each function is the atomic block of the agent-client comunication, it runs on the client and is responsible for the making the sigle request and getting back the response.

Its standard sequence of actions are:

- Initialize a buffer.
- Write the request ID.
- Fill the buffer with the data needed to perform the requested operations.
- Send the request via a *ssh_request_reply* call.
- Check the response exit codes.
- Return with data from the response.

Common Traits - Request Handler

Each request has an handler function in *ssh-agent.c*. The handler receive the request buffer process it and compose the response buffer.

Common Traits - Request Handler

Each request has an handler function in *ssh-agent.c*. The handler receive the request buffer process it and compose the response buffer.

Handlers make the effective operations, the interaction with keys happen here.

Timeleft hack

As seen the key struct within the agent store in the deathtime field how many seconds the key will stay in the agent before beeing removed. The standard openssh implemetation does not have a way to get this value.

Timeleft hack

As seen the key struct within the agent store in the deathtime field how many seconds the key will stay in the agent before beeing removed. The standard openssh implemetation does not have a way to get this value.

The timeleft hack add the -*T* option to ssh-add allowing the user to get this information.

Timeleft hack

As seen the key struct within the agent store in the deathtime field how many seconds the key will stay in the agent before beeing removed. The standard openssh implemetation does not have a way to get this value.

The timeleft hack add the -*T* option to ssh-add allowing the user to get this information.

This value may be put in a desktop widget, or in a command line prompt.

Timeleft hack - Examples

Agent with no key

```
$ eval `ssh-agent` ; ssh-add -l ; ssh-add -T
The agent has no identities.
none
```

Timeleft hack - Examples

Agent with no key

```
$ eval `ssh-agent` ; ssh-add -l ; ssh-add -T
The agent has no identities.
none
```

Agent with an expiring key

```
$ eval `ssh-agent` ; ssh-add -t 7200 ; ssh-add -l ; ssh-add -T
[passphrase ask ...]
1024 ea:98:a2:1d:f7:18:a1:11:22:2e:14:39:1b:66:63:3b /home/mirko/.ssh/id_dsa (DSA)
7198 sec
```

Timeleft hack - Examples

Agent with no key

```
$ eval `ssh-agent` ; ssh-add -l ; ssh-add -T
The agent has no identities.
none
```

Agent with an expiring key

```
$ eval `ssh-agent` ; ssh-add -t 7200 ; ssh-add -l ; ssh-add -T
[passphrase ask ...]
1024 ea:98:a2:1d:f7:18:a1:11:22:2e:14:39:1b:66:63:3b /home/mirko/.ssh/id_dsa (DSA)
7198 sec
```

Agent with a not expiring key

```
$ eval `ssh-agent` ; ssh-add ; ssh-add -l ; ssh-add -T
[passphrase ask ...]
1024 ea:98:a2:1d:f7:18:a1:11:22:2e:14:39:1b:66:63:3b /home/mirko/.ssh/id_dsa (DSA)
forever
```

Timeleft hack - Request and Handler

Since the task of retriving the time left is very simple (only one request) the request manager is trivial. It purpose is just to write the duration. “forever” or “none”

Timeleft hack - Request and Handler

Since the task of retriving the time left is very simple (only one request) the request manager is trivial. It purpose is just to write the duration. “forever” or “none”

The request is only the request ID since there is no other argument. The replay sends back an integer.

Timeleft hack - Request Manager

Timeleft Request Manager

```
static int
list_times(AuthenticationConnection *ac)
{
  int deathtime;
  int currenttime;
  deathtime=ssh_get_identity_timeleft(ac);
  currenttime=time(NULL);

  if (deathtime == 0 )
  {
    fprintf(stdout,"forever\n");
  }
  else
  {
    if ( deathtime > currenttime )
    {
      fprintf(stdout,"%d sec\n",
        deathtime-currenttime);
    }
    else
    {
      fprintf(stdout,"none\n");
    }
  }

  return 0;
}
```

Timeleft hack - Request and Handler

Timeleft Request

```
int
ssh_get_identity_timeleft(
      AuthenticationConnection *auth)
{
   int howmany;
   Buffer msg;
   buffer_init(&msg);
   buffer_put_char(&msg,
      SSH2_AGENTC_REQUEST_TIMELEFT);
   if (ssh_request_reply(auth, &msg, &msg)
      == 0) {
     buffer_free(&&msg);
     return 0;
   }
   type = buffer_get_char(&msg);
   if (agent_failed(type)) { return 0; }
   else if (type !=
      SSH2_AGENT_TIMELEFT_ANSWER) {  return
      0; }
   howmany = buffer_get_int(&msg);
   buffer_free(&msg);
   return howmany;
}
```

Timeleft Handler

```
static void
process_request_timeleft(SocketEntry *e,
      int version)
{
  Idtab *tab = idtab_lookup(version);
  //...
  buffer_init(&msg);
  buffer_put_char(&msg,
    SSH2_AGENT_TIMELEFT_ANSWER);
  id = TAILQ_FIRST( &tab->idlist);
  //...
  buffer_put_int(&msg, id->death);
  //...
}
```

Token hack

With the Token hack is possible to create an hash of a file merged with the agent's private key.

Token hack

With the Token hack is possible to create an hash of a file merged with the agent's private key.

The hack add the *-m [file]* option to ssh-add allowing the user to specify the file (on stdin with -).

Token hack

With the Token hack is possible to create an hash of a file merged with the agent's private key.

The hack add the *-m [file]* option to ssh-add allowing the user to specify the file (on stdin with -).

Internally the SHA1 hash function is used.

Token hack

With the Token hack is possible to create an hash of a file merged with the agent's private key.

The hack add the *-m [file]* option to ssh-add allowing the user to specify the file (on stdin with -).

Internally the SHA1 hash function is used.

This value may be use as token to authenticate other services.

Token hack - Examples

Token from a file

```
$ echo "This is a test" > testfile ; ssh-add -m testfile
d2aa41840bebdf183ee5bc4b4104716dc3342f7b
```

Token hack - Examples

Token from a file

```
$ echo "This is a test" > testfile ; ssh-add -m testfile
d2aa41840bebdf183ee5bc4b4104716dc3342f7b
```

Token from stdin

```
$ echo "This is a test" | ssh-add -m -
d2aa41840bebdf183ee5bc4b4104716dc3342f7b
```

Token hack - Request Manager

The Token hack request manager does the following actions:

Token hack - Request Manager

The Token hack request manager does the following actions:

- Open the file source for the token.

Token hack - Request Manager

The Token hack request manager does the following actions:

- Open the file source for the token.
- Spilt it in several parts and for every part:

Token hack - Request Manager

The Token hack request manager does the following actions:

- Open the file source for the token.
- Spilt it in several parts and for every part:
  - Compose a token request

Token hack - Request Manager

The Token hack request manager does the following actions:

- Open the file source for the token.
- Spilt it in several parts and for every part:
  - Compose a token request
  - Send the request

Token hack - Request Manager

The Token hack request manager does the following actions:

- Open the file source for the token.
- Spilt it in several parts and for every part:
  - Compose a token request
  - Send the request
  - Print the result request

Token hack - Request Manager

The Token hack request manager does the following actions:

- Open the file source for the token.
- Spilt it in several parts and for every part:
  - Compose a token request
  - Send the request
  - Print the result request
- Use the concatenation of the partial results as final token

Token hack - Request Manager

Token Request Manager

```
static int make_digest(AuthenticationConnection *ac,char * datas) {
   // Variables omitted
        if (datas!=NULL) {
                fileinp=open(datas,O_RDONLY);
                if (fileinp == -1) return -1;
                lseek(fileinp, 0, SEEK_SET);
        } else { fileinp=fileno(stdin); }
        key = ssh_get_first_identity(ac, &comment, 2);
        ret=-1;
        while (ret != 0) {
                pos=0;
                while ((ret != 0)&&(pos<MAX_AGENT_BUFFER)) {
                        ret=read(fileinp,filedata+pos,MAX_AGENT_BUFFER-pos);
                        if (ret == -1 ) {
                                fprintf(stderr,"File read failed"); return -1;
                        } else { pos=pos+ret; }
                }
                if (pos !=0) {
                        ecode = ssh_agent_digest(ac,key,&digest, &slen, filedata, pos);
                        if (ecode != 0 ) { fprintf(stderr,"Digest failed"); return -1; }
                        for(i=0;i<slen;i++) printf("%02x",*(digest+i));
                        free(digest);
                }
        }
   // ...
        return ret;
}
```

Token hack - Request and Handler

The request and the Request Handler behave like this:

Request

Handler

Token hack - Request and Handler

The request and the Request Handler behave like this:

Request

Handler

- Buffer initializzation

Request

```
int ssh_agent_digest(AuthenticationConnection *auth, Key *key, u_char **sigp ...) {

    // ...

    buffer_init(&msg);

    // ...

    buffer_free(&msg);
    return ret;
}
```

Token hack - Request and Handler

The request and the Request Handler behave like this:

Request

Handler

- Buffer initializzation
- Request ID write

Request

```
// ...

buffer_init(&msg);
buffer_put_char(&msg, SSH2_AGENTC_REQUEST_DIGEST);

// ...
```

Token hack - Request and Handler

The request and the Request Handler behave like this:

Request

Handler

- Buffer initializzation
- Request ID write
- Request Data write

Request

```
// ...

buffer_put_string(&msg, blob, blen);
buffer_put_string(&msg, data, datalen);
buffer_put_int(&msg, flags);

// ...
```

Token hack - Request and Handler

The request and the Request Handler behave like this:

Request

Handler

- Buffer initializzation
- Request ID write
- Request Data write
- Request Submit

Request

```
// ...

if (ssh_request_reply(auth, &msg, &msg) == 0) { buffer_free(&msg); return -1; }
type = buffer_get_char(&msg);
if (agent_failed(type)) {
   logit("Agent admitted failure to sign using the key.");
} else if (type != SSH2_AGENT_DIGEST_ANSWER) {
   fatal("Bad authentication response: %d", type);
} else {

// ...
```

Token hack - Request and Handler

The request and the Request Handler behave like this:

Request

- Buffer initializzation
- Request ID write
- Request Data write
- Request Submit

Handler

- Get the key specs and data

Request Handler

```
static void process_request_digest(SocketEntry *e) {

   // ...

   blob = buffer_get_string(&e->request, &blen);
   data = buffer_get_string(&e->request, &dlen);

   flags = buffer_get_int(&e->request);

   // ...
```

Token hack - Request and Handler

The request and the Request Handler behave like this:

Request

- Buffer initializzation
- Request ID write
- Request Data write
- Request Submit

Handler

- Get the key specs and data
- Get the the private key

Request Handler

```
// ...

key = key_from_blob(blob, blen);

// ...

ppkk=BN_bn2hex(id->key->dsa->priv_key);

// ...
```

Token hack - Request and Handler

The request and the Request Handler behave like this:

Request

- Buffer initializzation
- Request ID write
- Request Data write
- Request Submit

Handler

- Get the key specs and data
- Get the the private key
- Concatenate the private key with the data and make a SHA digest

Request Handler

```
// ...

intre=(unsigned char *) malloc ((strlen(ppkk)+dlen+1)*sizeof(unsigned char));
memset(intre,0x00,strlen(ppkk)+dlen);
memcpy(intre,ppkk,strlen(ppkk));
memcpy(intre+strlen(ppkk),data,dlen);
SHA((unsigned char *) intre, strlen(ppkk)+dlen, (unsigned char *) &digest);

// ...
```

Token hack - Request and Handler

The request and the Request Handler behave like this:

Request

- Buffer initializzation
- Request ID write
- Request Data write
- Request Submit

Handler

- Get the key specs and data
- Get the the private key
- Concatenate the private key with the data and make a SHA digest
- Send back the result

Request Handler

```
// ...

buffer_init(&msg);
buffer_put_char(&msg, SSH2_AGENT_DIGEST_ANSWER);

// ...

buffer_put_string(&msg, digest, SHA_DIGEST_LENGTH);

// ...
```

Token hack - Request and Handler

The request and the Request Handler behave like this:

Request

- Buffer initializzation
- Request ID write
- Request Data write
- Request Submit
- Return the result to the calling function

Handler

- Get the key specs and data
- Get the the private key
- Concatenate the private key with the data and make a SHA digest
- Send back the result

Request

```
    // ...

    } else {
        ret = 0;
        *sigp = buffer_get_string(&msg, lenp);
    }

    buffer_free(&msg);
    return ret;
}
```

Crypt/Decrypt hack

The Crypt/Decrypt hack adds the possibility to encrypt and dectrypt files using the private key (or an hash from it derived) as symmetric key.

Crypt/Decrypt hack

The Crypt/Decrypt hack adds the possibility to encrypt and dectrypt files using the private key (or an hash from it derived) as symmetric key.

The hack add the *-z [file]* option to encrypt a cleartext file to the ciphertext specified with the *-o [file]* option, and the *-Z [file]* to do the opposite (with the *-o [file]* option as well).

Crypt/Decrypt hack

The Crypt/Decrypt hack adds the possibility to encrypt and dectrypt files using the private key (or an hash from it derived) as symmetric key.

The hack add the *-z [file]* option to encrypt a cleartext file to the ciphertext specified with the *-o [file]* option, and the *-Z [file]* to do the opposite (with the *-o [file]* option as well).

Internally the AES symmetric cipher is used.

Crypt/Decrypt hack - Examples

Crypt and Decrypt

```
$ echo "This is another test" > plaintext
$ hexdump plaintext
0000000 6854 7369 6920 2073 6e61 746f 6568 2072
0000010 6574 7473 000a
0000015
$ ssh-add -z plaintext -o ciphertext
$ hexdump ciphertext
0000000 0000 2000 4e56 27c1 47c1 812a 5205 1f86
0000010 cbd3 f152 b783 6c13 c505 e42v b7ce a809
0000020 58fd 0757
0000024
$ ssh-add -Z ciphertext -o plaintext_new
$ hexdump plaintext_new
0000000 6854 7369 6920 2073 6e61 746f 6568 2072
0000010 6574 7473 000a
0000015
```

Crypt/Decrypt hack - Request

The request is almost identical to the one from the Token hack.

Crypt request

```
int ssh_agent_crypt(AuthenticationConnection *auth, Key *key, u_char **sigp, ...) {
   // ..
   if (en_or_de != 0) en_or_de=1;

   buffer_init(&msg);
   buffer_put_char(&msg, SSH2_AGENTC_REQUEST_CRYPT);
   buffer_put_string(&msg, blob, blen);
   buffer_put_string(&msg, data, datalen);
   buffer_put_int(&msg, en_or_de);
   buffer_put_int(&msg, flags);
   xfree(blob);

   if (ssh_request_reply(auth, &msg, &msg) == 0) {
      buffer_free(&msg); return -1;
   }
   type = buffer_get_char(&msg);
   if (agent_failed(type)) { logit("Agent admitted failure to sign using the key.");
   } else if (type != SSH2_AGENT_CRYPT_ANSWER) { fatal("Bad authentication response: %d", type);
   } else { ret = 0;
      *sigp = buffer_get_string(&msg, lenp);
   }
   buffer_free(&msg);
   return ret;
}
```

Crypt/Decrypt hack - Handler

To handle the AES cryptography i used the *openssl EVP library* that provides high-level interface to cryptographic functions, and created 3 functions within *ssh-agent.c*:

Crypt/Decrypt hack - Handler

To handle the AES cryptography i used the *openssl EVP library* that provides high-level interface to cryptographic functions, and created 3 functions within *ssh-agent.c*:

AES Initialization

```
int aes_init(unsigned char *key_data, int key_data_len, unsigned char *salt, ... ) {
  // ...

  EVP_CIPHER_CTX_init(e_ctx);
  EVP_EncryptInit_ex(e_ctx, EVP_aes_256_cbc(), NULL, key, iv);
  EVP_CIPHER_CTX_init(d_ctx);
  EVP_DecryptInit_ex(d_ctx, EVP_aes_256_cbc(), NULL, key, iv);

  return 0;
}
```

Crypt/Decrypt hack - Handler

To handle the AES cryptography i used the *openssl EVP library* that provides high-level interface to cryptographic functions, and created 3 functions within *ssh-agent.c*:

AES Crypt

```
unsigned char *aes_encrypt(EVP_CIPHER_CTX *e, unsigned char *plaintext, int *len)
{
  int c_len = *len + AES_BLOCK_SIZE, f_len = 0;
  unsigned char *ciphertext = malloc(c_len);

  EVP_EncryptInit_ex(e, NULL, NULL, NULL, NULL);
  EVP_EncryptUpdate(e, ciphertext, &c_len, plaintext, *len);
  EVP_EncryptFinal_ex(e, ciphertext+c_len, &f_len);
  *len = c_len + f_len;
  return ciphertext;
}
```

Crypt/Decrypt hack - Handler

To handle the AES cryptography i used the *openssl EVP library* that provides high-level interface to cryptographic functions, and created 3 functions within *ssh-agent.c*:

AES Decrypt

```
unsigned char *aes_decrypt(EVP_CIPHER_CTX *e, unsigned char *ciphertext, int *len)
{
  int p_len = *len, f_len = 0;
  unsigned char *plaintext = malloc(p_len + AES_BLOCK_SIZE);

  EVP_DecryptInit_ex(e, NULL, NULL, NULL, NULL);
  EVP_DecryptUpdate(e, plaintext, &p_len, ciphertext, *len);
  EVP_DecryptFinal_ex(e, plaintext+p_len, &f_len);

  *len = p_len + f_len;
  return plaintext;
}
```

Crypt/Decrypt hack - Handler

The functions are then used in the request handler:

Crypt/Decrypt hack - Handler

The functions are then used in the request handler:

Crypt/Decrypt handler

```
static void process_request_crypt(SocketEntry *e) {
// ...
   EVP_CIPHER_CTX en, de;
   unsigned int salt[] = {12345, 54321};
// ...
   ppkk=BN_bn2hex(id->key->dsa->priv_key);
// ...
   buffer_init(&msg);
   buffer_put_char(&msg, SSH2_AGENT_CRYPT_ANSWER);
   if (aes_init(ppkk, strlen(ppkk), (unsigned char *)&salt, &en, &de)) {
// ...
   if (en_or_de==0) {
      crypttext = aes_encrypt(&en, (unsigned char *)data, &len);
   } else {
      crypttext = (unsigned char *)aes_decrypt(&de, (unsigned char *)data, &len);
   }
   buffer_put_string(&msg, crypttext, len);
// ...
   EVP_CIPHER_CTX_cleanup(&en);
   EVP_CIPHER_CTX_cleanup(&de);
// ...
}
```

Crypt/Decrypt hack - Request Manager

The request manager (called crypt_thigs) is much complicated than the others in many ways:

Problem	Solution

Crypt/Decrypt hack - Request Manager

The request manager (called crypt_thigs) is much complicated than the others in many ways:

Problem	Solution
It has to handle an output file.	

Crypt/Decrypt hack - Request Manager

The request manager (called crypt_thigs) is much complicated than the others in many ways:

Problem	Solution
It has to handle an output file.	Added a new argument to *ssh-add.c* as well to crypt_thing and let the request manager to open the file for writing

Crypt/Decrypt hack - Request Manager

The request manager (called crypt_thigs) is much complicated than the others in many ways:

Problem	Solution
It has to handle both encryption and decryption.	

Crypt/Decrypt hack - Request Manager

The request manager (called crypt_thigs) is much complicated than the others in many ways:

Problem

It has to handle both encryption and decryption.

Solution

Added a flag to the signature of the crypt_thing function to specify if it is crypt or decrypt.

Crypt/Decrypt hack - Request Manager

The request manager (called crypt_thigs) is much complicated than the others in many ways:

Problem	Solution
The lenght of AES ciphertext may be different (and in general is) from the plaintext lenght.	

Crypt/Decrypt hack - Request Manager

The request manager (called crypt_thigs) is much complicated than the others in many ways:

Problem

The lenght of AES ciphertext may be different (and in general is) from the plaintext lenght.

Solution

A minimal structcure is needed for the output file, keeping the length of each block just before the ciphertext.

Crypt/Decrypt hack - Request Manager

The request manager (called crypt_thigs) is much complicated than the others in many ways:

Problem	Solution
The endianness-neutrality and data type length of the file write operations have to be assured to have portability.	

Crypt/Decrypt hack - Request Manager

The request manager (called crypt_thigs) is much complicated than the others in many ways:

Problem

The endianness-neutrality and data type length of the file write operations have to be assured to have portability.

Solution

The use of C99 standards for data type (as uint32_t) solve the data type length while i used the network ordering function to have endianness-neutrality (as htonl).

Regression tests

The openssh sources has a Makefile target that provides a suite of regression tests for all the protocols and keys operations among different versions and architectures.

Regression tests

The openssh sources has a Makefile target that provides a suite of regression tests for all the protocols and keys operations among different versions and architectures.

I added tests to check the determinism and coherence of the tokens and ciphertexts, these are the systems where the code has been tested succesfully:

Regression tests

The openssh sources has a Makefile target that provides a suite of regression tests for all the protocols and keys operations among different versions and architectures.

I added tests to check the determinism and coherence of the tokens and ciphertexts, these are the systems where the code has been tested succesfully:

Tested Systems

Architecture	Operating System	Kernel
amd64	Gentoo	Linux 3.8.13
i386	OpenBSD	OpenBSD 5.0 GENERIC
amd64	Debian Squeeze	Linux 2.6.32
amd64	Debian Wheezy	Linux 3.2.0
i686	Debian Wheezy	Linux 3.2.0
alpha EV67 Tsunami	Gentoo	Linux 3.7.10

Use cases

Now let's see some examples that may benefit from this improved *ssh-agent*.

Luks - 1

Linux Unified Key Setup or LUKS is a disk-encryption specification, it allows a user to encrypt a device with one ore more passphrase:

Luks - 1

Linux Unified Key Setup or LUKS is a disk-encryption specification, it allows a user to encrypt a device with one ore more passphrase:

Creating a LUKS device

```
cryptsetup luksFormat /dev/sdp [keyfile]
```

Luks - 1

Linux Unified Key Setup or LUKS is a disk-encryption specification, it allows a user to encrypt a device with one ore more passphrase:

Creating a LUKS device

```
cryptsetup luksFormat /dev/sdp [keyfile]
```

Add a key to a LUKS device

```
cryptsetup luksAddKey /dev/sdp [keyfile]
```

Luks - 1

Linux Unified Key Setup or LUKS is a disk-encryption specification, it allows a user to encrypt a device with one ore more passphrase:

Creating a LUKS device

```
cryptsetup luksFormat /dev/sdp [keyfile]
```

Add a key to a LUKS device

```
cryptsetup luksAddKey /dev/sdp [keyfile]
```

Decrypt a LUKS device

```
cryptsetup luksOpen /dev/sdp cryptodevicename [--key-file keyfile]
```

Luks - 1

Linux Unified Key Setup or LUKS is a disk-encryption specification, it allows a user to encrypt a device with one ore more passphrase:

Creating a LUKS device

```
cryptsetup luksFormat /dev/sdp [keyfile]
```

Add a key to a LUKS device

```
cryptsetup luksAddKey /dev/sdp [keyfile]
```

Decrypt a LUKS device

```
cryptsetup luksOpen /dev/sdp cryptodevicename [--key-file keyfile]
```

As seen from the examples it is possible to use a keyfile.

Luks - 2

Following this procedure is then possbile to bond a LUKS device with a SSH private key using the Token hack:

Luks - 2

Following this procedure is then possbile to bond a LUKS device with a SSH private key using the Token hack:

- Create a random file used as plaintext file:

```
dd if=/dev/urandom of=plaintext bs=1 count=1024
```

Luks - 2

Following this procedure is then possbile to bond a LUKS device with a SSH private key using the Token hack:

- Create a random file used as plaintext file:
  `dd if=/dev/urandom of=plaintext bs=1 count=1024`
- Start the agent and store an identity: `eval 'ssh-agent' ; ssh-add`

Luks - 2

Following this procedure is then possbile to bond a LUKS device with a SSH private key using the Token hack:

- Create a random file used as plaintext file:
  `dd if=/dev/urandom of=plaintext bs=1 count=1024`
- Start the agent and store an identity: `eval 'ssh-agent' ; ssh-add`
- Create a token: `ssh-add -m plaintext > token`

Luks - 2

Following this procedure is then possbile to bond a LUKS device with a SSH private key using the Token hack:

- Create a random file used as plaintext file:
  `dd if=/dev/urandom of=plaintext bs=1 count=1024`
- Start the agent and store an identity: `eval 'ssh-agent' ; ssh-add`
- Create a token: `ssh-add -m plaintext > token`
- Use the token to format the device or add it to an existing one:
  `cryptsetup luksFormat /dev/devn token` or
  `cryptsetup luksAddKey /dev/devn token`

Luks - 2

Following this procedure is then possbile to bond a LUKS device with a SSH private key using the Token hack:

- Create a random file used as plaintext file:
  `dd if=/dev/urandom of=plaintext bs=1 count=1024`
- Start the agent and store an identity: `eval 'ssh-agent' ; ssh-add`
- Create a token: `ssh-add -m plaintext > token`
- Use the token to format the device or add it to an existing one:
  `cryptsetup luksFormat /dev/devn token` or
  `cryptsetup luksAddKey /dev/devn token`
- Delete the token.

Luks - 2

Following this procedure is then possbile to bond a LUKS device with a SSH private key using the Token hack:

- Create a random file used as plaintext file:
  `dd if=/dev/urandom of=plaintext bs=1 count=1024`
- Start the agent and store an identity: `eval 'ssh-agent' ; ssh-add`
- Create a token: `ssh-add -m plaintext > token`
- Use the token to format the device or add it to an existing one:
  `cryptsetup luksFormat /dev/devn token` or
  `cryptsetup luksAddKey /dev/devn token`
- Delete the token.
- From now on the device may be unlocked recreating the token and with:
  `cryptsetup luksOpen /dev/devn crytodevn –with-keyfile token`

Tripwire

Tripwire is a free software to monitor the data integrity on systems. Acording to a configurable policy it keeps a cryptographic database of data and alert for changes against the policy. It is an HIDS (host-based intrusion detection system).

Tripwire

Tripwire is a free software to monitor the data integrity on systems. Acording to a configurable policy it keeps a cryptographic database of data and alert for changes against the policy. It is an HIDS (host-based intrusion detection system).

It works with two passphrases called “local” and “site”. The “site” passphrase is meant to be the main one, used to modify the policy and the configuration and work site-wide. The “local” is used to update the cryptographic database.

Tripwire

Tripwire is a free software to monitor the data integrity on systems. Acording to a configurable policy it keeps a cryptographic database of data and alert for changes against the policy. It is an HIDS (host-based intrusion detection system).

It works with two passphrases called “local” and “site”. The “site” passphrase is meant to be the main one, used to modify the policy and the configuration and work site-wide. The “local” is used to update the cryptographic database.

Tripwire is a sofisticated system and need continuos adjustments so bonding the passphrases to the SSH keys may simplify the operations. Moreover with this method the “site” passphrase may travel accross servers using the forwarding capabilities of SSH

Implement a remote HIDS

The following scripts show how to implement a remote HIDS with the improved agent:

Creating the database

```
#!/bin/bash
CONFLIST=""
CONFLIST="$CONFLIST /bin/ps"
CONFLIST="$CONFLIST /usr/bin/ssh-agent"
DBLOC=/tmp/dbloc

for i in $CONFLIST; do
  REMOTE_TOKEN=‘ssh -o ForwardAgent=yes
      myremotehost "ssh-add -m $i"‘
  echo "$i -> $REMOTE_TOKEN"
  touch $DBLOC/$REMOTE_TOKEN
done
```

Check the remote system

```
#!/bin/bash
CONFLIST=""
CONFLIST="$CONFLIST /bin/ps"
CONFLIST="$CONFLIST /usr/bin/ssh-agent"
DBLOC=/tmp/dbloc

for i in $CONFLIST; do
  REMOTE_TOKEN=‘ssh -o ForwardAgent=yes
      myremotehost "ssh-add -m $i"‘
  REMOTE_TOKEN2=‘ssh myremotehost "cat $i"
      | ssh-add -m -‘
  if [ -f $DBLOC/$REMOTE_TOKEN ]; then
    echo "$REMOTE_TOKEN -> $i (ok remote)"
  else
    echo "Alert !"
  fi
  if [ -f $DBLOC/$REMOTE_TOKEN2 ]; then
    echo "$REMOTE_TOKEN2 -> $i (ok local)"
  else
    echo "Alert !"
  fi
done
```

Conclusion

In this work i tried to give to the openssh agent a central role in my systems security and to make easier the every day security checks.

Conclusion

In this work i tried to give to the openssh agent a central role in my systems security and to make easier the every day security checks.

Some work has to be done to put the sources in an elegant way (some docs, comments, check for error states, etc) but everything works smoothly.

Conclusion

In this work i tried to give to the openssh agent a central role in my systems security and to make easier the every day security checks.

Some work has to be done to put the sources in an elegant way (some docs, comments, check for error states, etc) but everything works smoothly.

Some time may be spent to make some improvement such as:

Conclusion

In this work i tried to give to the openssh agent a central role in my systems security and to make easier the every day security checks.

Some work has to be done to put the sources in an elegant way (some docs, comments, check for error states, etc) but everything works smoothly.

Some time may be spent to make some improvement such as:

- Let the user choose the digest algorithm.

Conclusion

In this work i tried to give to the openssh agent a central role in my systems security and to make easier the every day security checks.

Some work has to be done to put the sources in an elegant way (some docs, comments, check for error states, etc) but everything works smoothly.

Some time may be spent to make some improvement such as:

- Let the user choose the digest algorithm.
- Let the user choose the symmetric cryptography algorithm.

Conclusion

In this work i tried to give to the openssh agent a central role in my systems security and to make easier the every day security checks.

Some work has to be done to put the sources in an elegant way (some docs, comments, check for error states, etc) but everything works smoothly.

Some time may be spent to make some improvement such as:

- Let the user choose the digest algorithm.
- Let the user choose the symmetric cryptography algorithm.
- Improve the regression tests.

Conclusion

In this work i tried to give to the openssh agent a central role in my systems security and to make easier the every day security checks.

Some work has to be done to put the sources in an elegant way (some docs, comments, check for error states, etc) but everything works smoothly.

Some time may be spent to make some improvement such as:

- Let the user choose the digest algorithm.
- Let the user choose the symmetric cryptography algorithm.
- Improve the regression tests.
- Introduce a random sequence whose token is used as symmetric key.

Conclusion

In this work i tried to give to the openssh agent a central role in my systems security and to make easier the every day security checks.

Some work has to be done to put the sources in an elegant way (some docs, comments, check for error states, etc) but everything works smoothly.

Some time may be spent to make some improvement such as:

- Let the user choose the digest algorithm.
- Let the user choose the symmetric cryptography algorithm.
- Improve the regression tests.
- Introduce a random sequence whose token is used as symmetric key.
- Manage multiple keys and other then DSA.