Installing D'Agents: Release 2.1

Robert S. Gray
Thayer School of Engineering
Hanover, NH 03755
E-mail: robert.s.gray@dartmouth.edu

March 11, 1998

1 Introduction

This document describes how to install D'Agents 2.1 and the Pretty Good Privacy (PGP) program such that D'Agents will use PGP to digitally sign and encrypt both migrating agents and inter-agent communication. Thus, the D'Agents server will be able to verify the identity of an incoming agent, and each agent will be able to verify the identity of those agents with which it is communicating.

If you do not want to take the time to install PGP, you should *stop reading now and switch to the other set of installation instructions*, which describe how to install D'Agents *without* installing PGP. Note that since D'Agents uses an old version of PGP, you *will* have a much easier time if you install D'Agents without PGP, sacrifice some of the security features, and simply run D'Agents inside a closed network environment. We recommend that you initially install D'Agents without PGP, and install D'Agents with PGP only if you like D'Agents and want to use it in a more substantial application.

You will find the installation process time-consuming the first time that you go through it. Based on initial user feedback, however, it will go much faster once you have done it a time or two.

If you need any help with installation, please send e-mail to robert.s.gray@dartmouth.edu.

2 Installation

The source code for D'Agents 2.1 is packaged as a tar file

`agent2.1.tar.gz`

which can be obtained via our World Wide Web page

`http://agent.cs.dartmouth.edu/`

or via anonymous FTP

`ftp::://agent.cs.dartmouth.edu/pub/agents/`

The source code distribution supports Tcl and Scheme agents only. An additional source code module for Java agents is available by request. If you are interested in the Java module, you should send e-mail to robert.s.gray@dartmouth.edu.

In addition, you need the source code for version 2.6.2 of MIT's implementation of PGP. MIT's PGP is packaged as a tar file

`pgp262s.tar.gz`

which can be obtained via the World Wide web page

`http://web.mit.edu/network/pgp.html`

Note: MIT's PGP 2.6.2 is the only encryption subsystem that D'Agents supports "out-of-the-box". Unfortunately, international users can not download MIT's PGP, since its cryptographic algorithms are considered munitions and can not be legally "exported" outside the United States or Canada. Therefore, the security mechanisms of D'Agents 2.1 will not be available to international users until D'Agents is modified so that it works with an international version of PGP. If anyone wants to (1) suggest a good international version of PGP or (2) (even better) write the necessary interface code themselves, please send mail to robert.s.gray@dartmouth.edu. The modifications should be quite simple and should involve only two files, generic/genPGP.cc and keys/template/keys.tcl.

The D'Agents and PGP tar files contain *all* necessary code. In particular, you do not need the source code for the standard Tcl/Tk interpreters, and you do not need to install the standard Tcl/Tk interpreters on your system. You also do not need the source code for any Scheme interpreter, and you do not need to install any Scheme interpreter on your system.

D'Agents 2.1 is a Unix-only system. It is known to compile and run correctly on most versions of Linux, FreeBSD and Solaris, and with most versions of GNU gcc/g++ and Sun CC.

D'Agents 2.1 should be easily portable to any modern Unix, and since it does not use C++ exceptions or templates, most C++ compilers should be able to compile it. Porting to non-Unix operating systems, such as Windows NT/XP/2000 or Windows 98, will require a significant amount of work.

2.1 Installing D'Agents 2.1

To install D'Agents 2.1:

1. Although D'Agents provides security mechanisms to protect a machine from malicious agents, these security mechanisms are not complete. For example, an agent can consume as much memory as it wants (up to the limit imposed by the underlying Unix system). Therefore, since incoming agents run under the same userid as the D'Agents server, you should *not* run the server as *root* or as any userid that has authority to access or overwrite sensitive data. In fact, it is *highly recommended* that you create a new account for the D'Agents system and run the server from this account. The account should have only as much authority as is needed for the desired prototype application(s). In this document, I will assume that you create an account with username *dagents*. For the greatest peace of mind, you should also create a new group, and make username *dagents* a member of that group only.

2. Log onto your new *dagents* account, and download, uncompress and untar agent2.1.tgz, This process will create a directory called *agent2.1*. Read the README file that is in the *agent2.1* directory before you continue.

3. Edit the *Makefile* and specify the desired compilation options. Most of the options should be self-explanatory.

   PORT specifies the TCP/IP port that the agent server uses to communicate with agents on remote machines. You *must* use the same TCP/IP port on every machine. SOCKET_NAME specifies the filename of the Unix domain socket that the server uses to communicate with agents on the local machine.

   LOG_DIRECTORY specifies the directory in which the server will store its log files.

   DATA_DIRECTORY specifies the directory in which the server will create the Unix domain socket SOCKET_NAME. In addition, all of the server configuration files will be installed in DATA_DIRECTORY.

USERNAME and GROUPNAME specify the user and group to which the server will switch if it is started as *root*. USERNAME should be the username of the account that you created in step 1 (e.g, *dagents*). GROUPNAME should be the associated group.

SERVER_HOME is the home directory for the server. This directory should be on the local disk since the server's database of encryption keys will be stored in this directory.

LOCAL_MACHINE specifies the name of the machine on which you are installing the server. This information is not used during compilation, but instead is used to generate some default configuration files.

ADMIN_KEYNAME is the PGP keyname of the server administrator. Since you will not create the administrator's PGP key until later, you can enter any desired keyname.

TEMP_DIRECTORY specifies the directory in which the server will create its temporary files.

SOURCE_PREFIX is the source code directory (i.e., the directory that contains the Makefile that you are currently editing). BUILD_PREFIX is the build directory (i.e., the directory in which .o files will be stored during compilation). Both SOURCE_PREFIX and BUILD_PREFIX should be fully qualified.

System management will be much easier if all of the installation subdirectories are in the same top-level directory. For example, in the default Makefile, all of the installation subdirectories are in the top-level directory */usr/agenttcl*.

Finally, do not change the values of PGP_NAME or PGP_KEY_DIRECTORY, since the rest of this document assumes that you are using the default values.

4. Compile the modified Tcl interpreter (*agent*) by typing

   ```
   make tcl_agent
   ```

   Compile the modified Tk interpreter (*agent-tk*) by typing

   ```
   make tk_agent
   ```

   Compile the server (*agentd*) by typing

   ```
   make serv
   ```

   The Makefile does not detect and recover from as many errors as it should. Watch the output for anything that looks like an error message.

5. Customize the configuration files and Tcl scripts by typing

   ```
   make customize
   ```

   This step creates reasonable default configuration files and changes the Tcl scripts so that the first line points to the correct executable, etc.

6. Install the system by typing

   ```
   make install
   ```

   This installs the executables, the libraries and the configuration files. If you want to install only the executables and libraries (i.e., if you do not want to overwrite existing configuration files), type

   ```
   make install-code
   ```

instead.

7. If you want to use Scheme agents (which I recommend only if you are familar with Scheme), you should compile the Scheme module

   ```
   make scheme_agent
   ```

   and then install the Scheme module

   ```
   make install-scheme
   ```

8. Type

   ```
   make ranbits
   ```

   to generate a random bits file that the server will use to initialize its cryptographically-secure random number generator. This step takes a while, but it provides significant security benefits.

9. Add the directory that contains the *agent* and *agent-tk* interpreters to the PATH environment variable on the *dagents* account – i.e., add BIN_INSTALL to the PATH variable. Add the directory that contains the shared libraries to the LD_LIBRARY_PATH environment variable on the *dagents* account – i.e., add LIB_INSTALL to the LD_LIBRARY_PATH variable. If you do not want other users to be able to run the interpreters, *agent* and *agent-tk*, go to the BIN_INSTALL directory and turn off the group and other permissions. If you do want other users to be able to run the interpreters, make sure that the other users know about adding BIN_INSTALL to their PATH and LIB_INSTALL to their LD_LIBRARY_PATH. Of course, you can go halfway instead, and allow only a certain group of users to execute the interpreters (e.g., perhaps the special agent group that you created back in step 1).

The entire installation process must be repeated for every distinct architecture on which you want to run the D'Agents system. On duplicate architectures, you do not need to recompile, but you will need to do everything else (i.e., edit the Makefile, *make customize*, *make install*, *make ranbits*, etc.). On every machine, you should use a different set of installation directories, even if two (or more) machines share a filesystem and have the same architecture. Although it is possible to share the *bin*, *lib* and *scripts* subdirectories, it gets confusing quickly.

2.2 Installing PGP and generating keys

To install PGP and generate keys:

1. For each machine on which you plan to run the D'Agents system, (a) log onto the *dagents* account, (b) download, uncompress and untar MIT's PGP version 2.6.2, (c) compile MIT's PGP, and (d) copy the resulting executable, *pgp*, into the PGP_INSTALL directory that you specified in the Makefile. Rename the executable as *agpgp* when you copy it. In other words, when you are done with this step, you should have an executable called *agpgp* in the PGP_INSTALL directory.

2. Use the *keys.tcl* utility, which was installed in the BIN_INSTALL directory, to generate a key for each machine, each user and the administrator. Generate all the keys on the same machine so that you end up with a *single* master database of keys. Generate a user key even if the user already has a PGP key for e-mail, since separate keys will provide more security, both for the D'Agents system and for the user. When generating a machine key, make sure that you enter the *complete* machine name, e.g., *bald.cs.dartmouth.edu* rather than just *bald*. When generating the administrator key, make sure that you enter the keyname that you specified in the ADMIN_KEYNAME field in the Makefile.

Note: Each key has a name and a password, both of which are case-sensitive. Make sure that you mention the case-sensitivity when you distribute the names and passwords to your D'Agents users.

3. Once you have generated the keys, use *keys.tcl* to create a package of keys for each machine and each user. To create a package, hit the "Import/Export Keys" button. When you hit this button, you will be asked for the name of the package. You can use any name that you want. If you are creating the package for a machine named *tioga*, for example, you might enter *tioga* as the package name. The package then would be saved to disk as two separate files, *tioga.sec.pgp*, which contains the package's private keys, and *tioga.pub.pgp*, which contains the package's public keys.

   For each machine, the package should contain the public/private keys of that machine and the public keys of all other machines, all users and the administrator. For each user, the package should contain the public/private keys of that user and the public keys of all other users, all machines and the administrator. (If a user is an administrator, the package should include the public/private keys of the administrator rather than just the public key.)

4. Use the *genrand* utility, which was installed in the SERV_INSTALL directory, to create a file called *randseed.bin* that contains 3072 random bits. In other words, enter the command

   ```
   genrand randseed.bin 3072
   ```

   at the Unix prompt. If you feel up to it, you can create a different *randseed.bin* file for each machine and each user. Per-user randseed.bin files provide additional security (but only a little, since the files will be regenerated as soon as they are used once.)

5. Copy the *config.txt* that came with MIT's PGP into the same directory as your new *randseed.bin* file. Having these two files in the same place will make the next step easier.

6. Now you need to put the correct package of keys onto each machine. For each machine, (a) create a subdirectory called *.agpgp* in the SERVER_HOME directory, (b) copy the *randseed.bin* and *config.txt* files from the previous two steps into this new *.agpgp* directory, and (c) copy the *XXX.sec.pgp* and *XXX.pub.pgp* files that contain the machine's keys into the *.agpgp* directory. When you make the copies in step (c), rename *XXX.sec.pgp* as *secring.pgp*, and rename *XXX.pub.pgp* as *pubring.pgp*. In other words, for a machine called *tioga*, you should end up with two files called *pubring.pgp* and *secring.pgp* in the *.apgp* directory. The *pubring.pgp* file should be a copy of *tioga.pub.pgp*, and the *secring.pgp* file should be a copy of *tioga.sec.pgp*. Finally, turn off group and other permissions for all four files, *randseed.bin*, *config.txt*, *secring.pgp* and *pubring.pgp*, and make sure that all four files and the *.agpgp* directory are owned by the special agent userid, *dagents*.

7. Finally, you need to give the correct package of keys to each user. For each user, (a) create a subdirectory called *.agpgp* in the user's home directory, (b) copy the *randseed.bin* and *config.txt* files into this new *.agpgp* directory, and (c) copy the *XXX.sec.pgp* and *XXX.pub.pgp* files that correspond to that user into the *.agpgp* directory. When you make the copies in step (c), rename *XXX.sec.pgp* as *secring.pgp*, and rename *XXX.pub.pgp* as *pubring.pgp*. Finally, turn off group and other permissions for all four files, *randseed.bin*, *config.txt*, *secring.pgp* and *pubring.pgp*, and make sure that all four files and the *.agpgp* directory are owned by the user. You must repeat this process if the user has different home directories on different machines. Of course, you can have your D'Agents users perform this step themselves, rather than doing it all yourself.

If you need to generate additional machine or user keys later, you can simply generate the new keys, export the new keys into any affected packages, and redistribute the packages.

3 Running the server

The server must run on every machine to which you want to send mobile agents. For each machine:

1. Log onto the *dagents* account. Remember that you want the server to run with the minimal permissions of the *dagents* account.

2. Edit the server configuration file *agentd.conf*. This configuration file is in the DATA_DIRECTORY that you specified in the Makefile. The configuration file is well commented. Here we just cover some of the more important parts.

   The *PassPhrase* field specifies the password for the server's PGP key. (The server's PGP key is the PGP key that you generated for the server's machine in the previous section.)

   The *MaxAgents* and *MaxAnonAgents* fields specify how many agents and how many *anonymous* agents are allowed on the machine at one time. If *MaxAnonAgents* is zero, the server will only accept agents that have been digitally signed by an authorized user.

   The *Owner* and *Admin* fields specify the PGP keynames of the machine's owner and administrator respectively. The owner and administrator have special authority. For example, they can forcibly terminate any running agent.

   You should add an *AllowedUser* field for each authorized user. An agent that comes from any other user will be considered *anonymous*, and if *MaxAnonAgents* is zero, these anonymous agents will be refused entry.

   You should add an *AllowedMachine* field for each machine from which the server is allowed to accept agents. As in D'Agents 1.1, the server checks whether an incoming agent came from an *AllowedMachine* using only the apparent IP address of the sender. Thus, if your network environment is vulnerable to TCP/IP spoofing attacks, you should not allow anonymous agents. If your network environment is not vulnerable to TCP/IP spoofing attacks (or is not *too* vulnerable), you should allow anonymous agents, since anonymous agents are much faster than authenticated agents. (Anonymous agents do not need to be digitally signed with PGP.)

   The *AllowSigned* field should be set to *yes* if you want to accept agents from authorized users even if the authorized users are not on an *AllowedMachine*. For example, this field should be set to *yes* if some of your authorized users want to launch agents from their laptop computers (which might have different network addresses at different times).

   The *TrustedMachine* fields specify the machines that the local machine trusts completely. Suppose that machine A has an agent that verifiably came from user *rgray*. Now the agent decides to migrate to machine B, which trusts machine A. Machine A digitally signs the agent with its PGP key and sends the agent to machine B. Machine B verifies that the agent came from machine A, and since machine B trusts machine A, machine B believes machine A when it says that the agent originally came from user *rgray*. Thus, machine B treats the agent as if it came directly from user *rgray*, rather than through an intermediate machine.

Note: If an agent migrates to an untrusted machine, and then returns later, it will be considered *anonymous* upon its return, simply because the untrusted machine might have modified the agent in a malicious way. (The current version of D'Agents does not protect an agent against malicious machines.) Keep this in mind when deciding (a) which machines trust each other and (b) whether to allow anonymous agents.

   The *ReservedName* fields specify that certain agent names are reserved for certain owners. The two reserved names in the current system are *resource-manager*, which is used by the resource managers, and *server-checker*, which is used by the *machine.tcl* script.

3. Change into the server directory. The server directory is the SERV_INSTALL directory that you specified in the Makefile.

4. Start the server by typing

   ```
   ./start_agentd
   ```

   You will see output that looks like the following:

```
Mon Jul 15 16:18:44 2002 - loadConfig: initial:
     INFO: AllowedMachine "localhost"
Mon Jul 15 16:18:44 2002 - loadConfig: initial:
     INFO: AllowedMachine "10.0.0.53"
Mon Jul 15 16:18:44 2002 - loadConfig: initial:
     INFO: TrustedMachine "localhost"
Mon Jul 15 16:18:44 2002 - loadConfig: initial:
     INFO: TrustedMachine "10.0.0.53"
Mon Jul 15 16:18:44 2002 - loadConfig: initial:
     INFO: name "resource-manager" reserved for owner
           "Agent Tcl administrator <cs.dartmouth.edu>"
Mon Jul 15 16:18:44 2002 - loadConfig: initial:
     INFO: name "server-checker" reserved for owner
           "Agent Tcl administrator <cs.dartmouth.edu>"
Mon Jul 15 16:18:44 2002 - loadConfig: initial:
     INFO: loading interpreter list
Mon Jul 15 16:18:44 2002 - loadConfig: initial:
     INFO: adding "STATE-TCL" to interpreter list
Mon Jul 15 16:18:44 2002 - loadConfig: initial:
     INFO: 3072 random bits read from file
           "/usr/agenttcl/access/random.bin"
```

After the last line, you will be returned to the command prompt, since the server is now running as a daemon. All further messages are logged in the file *agent.log*, which is in the LOG_DIRECTORY that you specified in the Makefile.

To verify that the server is running as a daemon, check for *agentd* processes. On the author's machine:

```
% ps ax | grep agentd
32080 pp4 S     0:00 agentd -host bald.cs.dartmouth.edu ...
32081 pp4 S     0:00 agentd -host bald.cs.dartmouth.edu ...
%
```

The first *agentd* maintains the server's internal tables, and the second *agentd* watches for incoming agents on the TCP/IP port. Both processes must be running for the server to function correctly. Check the screen output and the log file for error messages if one or both are missing. In the error log, the first *agentd* is referred to as "agentd", and the second *agentd* is referred to as "socketd".

The server can be run in the foreground by editing the file *start_agentd* and adding the flag

```
-nodaemon
```

to the line that actually starts *agentd*. All error and informational messages will be displayed on the terminal rather than written to the log file. Foreground mode is useful for server development and intense debugging.

5. Switch back to the DATA_DIRECTORY and edit the file *manager.conf*. This file determines what kind of system access is given to each authorized user. Access is controlled at a coarse level in the current release, e.g., all agents from a particular authorized owner either can or cannot access the local filesystem. We chose coarse-grained resource control for this initial release since it was (1) easier to get right and (2) sufficient for experimentation with both anonymous "client" agents (that do not need direct access to system resources) and authenticated "service" agents (that do need direct access).

6. Switch back to the SERV_INSTALL directory and start the resource manager. To do this, first set the AGENTKEY and AGENTPASS environment variables to the PGP keyname and password of the user

who is allowed to start the resource manager. (You specified this user in the *ReservedName* field in the *agentd.conf* file, and you chose the user's keyname and password during the key-generation phase.) Then start the resource manager by running the *start_man* script. On the author's machine:

```
setenv AGENTKEY "Agent Tcl administrator <cs.dartmouth.edu>"
setenv AGENTPASS "Agent Tcl is a banana"
./start_man
```

Note that environment variables are not the best way to specify the keyname and password. However, the agent system does unset these environment variables as soon as it has read their values, so the variables will not be accidentally propagated to child processes.

7. To bring down the server later, you need to kill all *agentd*, *agent*, and *agent-tk* processes (there will be quite a few if there are several agents executing on the local machine). Sending a SIGTERM to any *agentd* process will kill all *agentd* processes and most of the *agent* and *agent-tk* processes. The remaining *agent* and *agent-tk* processes must be killed individually.

8. If you change *agentd.conf*, you need to tell the server to reload its configuration information. You can either (1) send a SIGHUP to any *agentd* process or (2) stop and restart the server as per the previous steps.

9. Finally, it is okay to run *start_agentd* from a system bootup script. The server will switch to the userid specified in the configuration file *agentd.conf* as soon as it starts executing. It is also okay to run *start_man* from a system bootup script. In this case, however, the bootup script must first set the AGENTKEY and AGENTPASS environment variables, so the script should not be group- or other-readable (or writable).

4 Utilities

There are several useful utilities included with the system: *genrand*, *stirrand*, *seerand* and *machine.tcl*. The first three utilities, *genrand*, *stirrand* and *seerand*, are used to generate and stir the server's random bits file. These three utilities are in the SERV_INSTALL directory; the random bits file is the file *random.bin* in the DATA_DIRECTORY directory. To generate a new random bits file, use the *genrand* command.

```
genrand random.bin 3072
```

The second argument is the number of random bits to place in the file. Make sure that you use 3072 here, since the server will refuse to continue if less than 3072 random bits are available. To "stir" the random bits with the contents of an arbitrary file, use the *stirrand* command.

```
stirrand random.bin < filename
```

Finally, to see the random bits as human-readable hex numbers, use the *seerand* command.

```
seerand random.bin
```

You should not use the *seerand* command too often since it involves transmitting the random bits in "clear-text" at some point (e.g., between the remote disk and the local machine, between the local machine and the screen, etc.). You might use it occasionally, however, if you want to verify that the D'Agents developers have not done anything stupid, such as accidentally filling *random.bin* with zeros.

After using *stirrand* or *genrand*, you must tell the server to reload the random bits file. You can do this by sending a SIGHUP to any *agentd* process (or by stopping and restarting the server as described above). In

addition to any regeneration and stirring that you do, the server uses the types and arrival times of agent requests to continually stir the random bits as it executes, and writes out a new random bits file every time that it shuts down.

The last utility, *machine.tcl*, is found in the BIN_INSTALL directory. First, you must edit *machine.tcl* and change the four variables at the top to values that are appropriate for your site. In particular, the *machines* variable must list the machines on which *you* are running the D'Agents server. (The D'Agents logs at Dartmouth are filled with "access-denied" messages because many people run *machine.tcl* without replacing the Dartmouth machine names with their own machine names.) Now if you just type

```
machine.tcl
```

you will see a list of the machines on which the server *should* be running. If you type

```
machine.tcl all
```

instead, *machine.tcl* will *confirm* that the server is running on each machine by sending out child agents and waiting for responses. You will see informational messages as it submits the agents and receives the responses. Once *machine.tcl* has received all the responses or timed out, it will display a list of errors. Errors almost always indicate that the server is not running on the specified machine (or that your network is down). If you want to check only specific machines, you can list those machines on the command line. For example,

```
machine.tcl m q
```

confirms that the server is running on machines *m* and *q*. Finally, if you want to get reports periodically as to which servers are down, you can set up

```
machine.tcl all
```

as a cron job that e-mails its output to a given user.

Bug: The *machine.tcl* utility assumes that your servers will accept anonymous agents. If this is not the case, edit *machine.tcl* so that the line

```
security signatures on
```

is always executed, i.e., remove the enclosing `if` statement. Then, before running *machine.tcl*, set the AGENTKEY and AGENTPASS environment variables to the PGP keyname and password of an authorized user.

5 Agents and security

A full discussion of D'Agents security features (and the commands for using those security features) will appear in future documentation. Here is just enough to get started. When launching an agent, the user must first set the AGENTKEY and AGENTPASS environment variables to her PGP keyname and password. Then, before issuing the agent_begin command, the *agent* must issue the command

```
security signatures on
```

to turn on digital signatures. Optionally, the agent can also issue the command

```
security encryption on
```

to turn on encryption. (Digital signatures allow a server to authenticate the identity of an agent's owner; encryption prevents anyone from looking at the agent's contents while the agent is in transit across the network.)

Note: If the agent can be anonymous, i.e., if (1) the agent can perform its task without special resource access and (2) the servers accept anonymous agents, none of the previous steps are necessary.