ＩＴ特別講座
SSH 再入門

～ssh サーバーと高度な利用～

Ver 1.0

LA-Linux 専任講師 矢越昭仁

2011/07/23

ネットワークを介したコマンド操作はセキュリティ上 SSH を使う事が常識となっています。このコースではサーバーの設定や、より高度な利用方法について解説します。

目次

1.SSH とは

1-1. SSH の必要性

TELNET を用いたリモート操作では､すべてのデータが暗号化されないまま送信されます。このため、重要なアカウント情報が盗聴されやすいという危険性がありました。ログイン時にユーザー名とパスワードだけ暗号化したとしても、su コマンド使う時などはパスワードがネットワークを流れてしまいます。

そのため通信全てを暗号化し､盗聴されても内容が漏れないようにする SSH（Secure Shell）というプロトコルが開発されました。SSH は、サーバ・クライアント双方で SSH プログラムを動かし、SSH 同士が暗号化した経路を確立します。情報はのこ暗号化された経路の中でやりとりされます。

図 1：SSH 暗号化経路概念図

OpenSSH

SSH は 1995 年ヘルシンキ大学のタトゥ・ウネロン氏によりフリーソフトとして配布されましたが、後に SSH 社を設立、商用化されました。その後、OpenSSH プロジェクトが新た互換ソフトを開発し無償で公開しています。現在では Linux を含む、多くの UNIX 系 OS での動作がサポートされています。CentOS では以下のパッケージがあります。

```
・openssh             Client/Server 両方に共通する機能、ファイル
・openssh-askpass     X11 用の認証 GUI
・openssh-clients     SSH クライアント機能
・openssh-server      SSH サーバー機能
・openssl             共有ライブラリなど
```

必要に応じ yum 等でインストールして下さい。

1-2. OpenSSH クライアント

SSH クライアントの使用

SSH サーバーに接続するためには、OpenSSH パッケージの SSH クライアントプログラムとして、ssh コマンドを利用します。数々のオプションがサポートされていますが、基本的な用法は以下のとおりです。

```
ssh  [ ユーザー名 @ ] 接続先のホスト

例）
$ ssh student@172.16.0.1
```

ログイン先のホスト名または IP アドレスを指定します。接続先ユーザー省略すると、接続

元のユーザー名が使用されます。

ホスト認証

通信を暗号化しても、すべてが安全になるというわけではありません。例えば接続先のホストになりすまし、重要なデータを奪うとい事が考えられます。

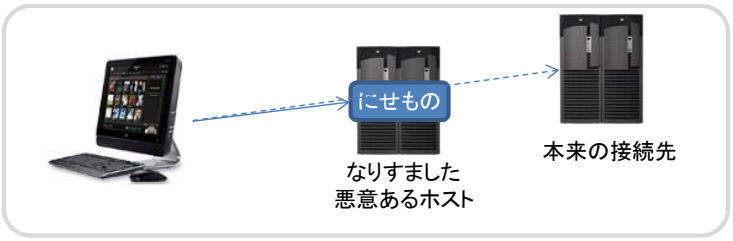


図 2：なりすましホスト例

このなりすまし[1]回避するには、ログイン先が意図しているホストかどうかを確認する必要があります。SSH では、通信の開始時に SSH サーバー側からクライアントに対してホスト認証鍵と呼ばれるデータが送信されます。

ホスト認証鍵は、ホストに固有のもので、なりすましホストと正規のホストとでは異なり、「なりすまし」を検出することができます。

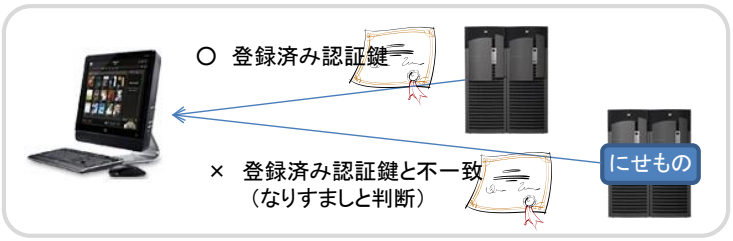


図 3：ホスト認証概念図

SSH サーバーに対して、ssh コマンドを使用して通信を開始する場合、初回のログイン時に、以下のようなメッセージが表示されます。

```
$ ssh student@172.16.0.100
The authenticity of host‘172.16.0.100’can’t be established.
DSA key fingerprint is 97:02:85:2f:a1:c2:ca:b2:47:7d:11:d3:fc:14:fa:6a.
Are you sure you want to continue connecting (yes/no)?
```

これは、ホスト認証鍵がサーバー側から送られてきたことを示し、このホストを信用するか否を確認しています。

問題がなく[2]「yes」と入力すると、更に以下のようなメッセージが表示されます。

```
Warning: Permanently added ‘172.16.0.100’ (DSA) to the list of known hosts.
```

これは接続先のホストの認証鍵がクライアント側に登録され、次回からはこの認証鍵とサーバーから送られる認証鍵をチェックするということを示しています。

登録された認証鍵は、ユーザーのホームディレクトリにある「.ssh」ディレクトリ内の「known_hosts」ファイルに追記されていきます。

---

[1] なりすましたコンピュータが、正規のホストから取得したホスト認証鍵をそのまま送信することでなりすましを行うことが出来ないように工夫されています。

[2] サーバー管理者に fingerprint の値が正しいかどうかを確認します。詳しくは p.18「ホスト認証鍵」参照

```
$ cat ~/.ssh/known_hosts
172.16.0.100 ssh-dss
AAAAB3NzaC1kc3MAAACBAJe3ZBierTVC4ZUdibZxdYuJ86kujEV93jsVKS/9n8yYM6XvKWE3q
aj41TAl1YeJIz3oEUzmTwFCy+FKnW+6dzr6Ux/Xl5nDbNcsrpKwRrDql8U8ddOVeHGl/zeydL
LhA4cWlMS8n7eNaL9jYN9AWh8FA10LItPsTufVzEMU/i0jAAAAFQDR+KxGf/injm6PvhP6did
otHOdRwAAAIAeFFzM9C6Ci8fo+w6jrRL9ajkazcgcQTGlRovfp8h5CoaS8iaqsr8m5zJlCF4C
kvOUUhmjMcLpIpZxYWgwm4OhK6mAIcdCN39x96ATBwUkNVWtpOD93u20Fuxl5X3357Z2td8vl
D6d8Qrx/mTIXYw2RRCjSqLaw3qFtbL5W+dJdgAAAIB6VmdyxvpGT69p4rQePejTtpO1EUtr5I
O5+9dYbzbFhJwm3iDXnYmASAWbGt2a+Skn64+3ySCDwZB+z6NUKjkVuHmNPPtCBM83IX+8gCo
6CZUIj4m9iLqfPb5E6ghqvX1WMDT/8PRKbpVPs9AenghBCQrbGfSz8KJpOH4ztqgkmA
```

これらのファイルには、

```
[ホスト名 1，ホスト名 2,...] (ssh-dss or ssh-rsa) [認証鍵のデータ]
```

という書式で、登録された全てのホストの認証鍵が記述されています。

2 回目以降、同じホストに接続したつもりで、この認証鍵が異なると「なりすまし」の疑いがあるとしてログインがエラーになります。

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that the RSA host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
6e:23:af:20:ef:89:66:aa:72:7a:0d:78:fe:57:26:3f.
Please contact your system administrator.
Add correct host key in /root/.ssh/known_hosts to get rid of this message.
Offending key in /root/.ssh/known_hosts:4
RSA host key for localhost has changed and you have requested strict checking.
Host key verification failed.
```

暗号化されたパスワード認証

SSH を用いた通信は、サーバーのホスト認証が終了した後、ユーザーの認証が行われます。認証は複数の方法がサポートされていて、最もシンプルな方法は Linux のパスワードを使う方法です。これをパスワード認証と呼びます。

```
$ ssh student@172.16.0.100
student@172.16.0.100's password: [パスワードを入力]
Last login: Sun Jan  9 10:00:00 2005 from 172.16.0.200
[student@h001 student]$
```

ログイン時に求められるパスワードはログイン先のホスト（ここでは 172.16.0.100）でのパスワードであることに注意してください。接続先のホストに登録されているアカウントでログインします。

1-3. OpenSSH サーバー

SSH サーバーの起動

OpenSSH サーバーは、openssh-server パッケージによりインストールされます。実行プログラムは「/usr/sbin/sshd」であり、sshd デーモンが SSH サーバーの役割を担います。sshd に関する設定ファイルは、「/etc/ssh/sshd_config」ファイルですが、デフォルトの設定[3]で、パスワード認証が可能になっています。多くの Linux では標準で利用できるようになっています。

sshd を起動、停止はサーバー起動用スクリプトによります。

```
# /etc/init.d/sshd start              （起動）
# /etc/init.d/sshd stop               （停止）
```

SSH サーバーの設定

OpenSSH でのサーバーデーモン sshd の設定は設定ファイル sshd_config で行います。CentOS などでは、ディレクトリ/etc/ssh/以下に配置されています[4]。

```
#    $OpenBSD: sshd_config,v 1.73 2005/12/06 22:38:28 reyk Exp $

# This is the sshd server system-wide configuration file.  See
# sshd_config(5) for more information.

# This sshd was compiled with PATH=/usr/local/bin:/bin:/usr/bin

# The strategy used for options in the default sshd_config shipped with
# OpenSSH is to specify options with their default value where
# possible, but leave them commented.  Uncommented options change a
# default value.

#Port 22
#Protocol 2,1
Protocol 2
```

「#」で始まる行はコメント行でデフォルトの設定ファイルにおいては、設定項目についてコメントアウトされている部分が見受けられますが、これはデフォルトの設定値を示しています。例えば、sshd サーバーがクライアントからの要求を受け付ける TCP ポートに関する設定である「Port」については、以下の記述となっています。

```
    #Port 22
```

これは sshd が起動するときのデフォルト値として 22 が指定されている事を表しています。

[3] デフォルト設定には脆弱性があるので、運用には注意が必要です。
[4] ディレクトリ/etc/ssh 以下には、sshd_config 以外にも ssh_config といった設定ファイルもあります。これは ssh クライアントの設定ファイルです。混同しないように注意してください。

主な設定項目は以下の通りです（カッコ内は省略値)。

表1： sshd_config の主な設定項目

項目名	解説
Port	sshd が要求を LISTEN するポートのポート番号(22)
Protocol	sshd がサポートする SSH プロトコルのバージョン SSH1 と SSH2 のいずれも許可する場合と、SSH2 のみを許可する場合のいずれかが一般的である(2)
HostKey	クライアント側に送信するホスト認証鍵を収めたファイル名。通常は変更しない(/etc/ssh/ssh_host_key, ssh, ssh_host_rsa_key, ssh_host_dsa_key)
LoginGraceTime	クライアントが認証のために用いることができる時間[秒/分] 接続をした後、この時間を過ぎても認証に成功しない場合は、自動的に接続が切断される。(2m)
PermitRootLogin	ssh を利用したユーザーroot の直接ログインを許可する場合は「yes」、拒否する場合は「no」を指定する。特段の理由がない限り、拒否するのが望ましい(yes)
PasswordAuthentication	パスワード認証を許可するかどうかを指定する。セキュリティレベルを向上させるために、許可しない場合が増えている(yes)
ChallengeResponse Authentication	SSH1 のチャレンジ＆レスポンス認証の有無(yes)
AllowUsers	利用可能な Linux ユーザのリスト(なし)
X11Forwarding	X11 のポート転送の有無(yes)

パスワード認証を許可にするためには、「PasswordAuthentification」を「yes」に指定する必要があります。デフォルトの設定で「yes」が指定されています。

/etc/ssh/sshd_config の設定を変更した場合は、sshd を再起動して、新しい設定を有効にする必要があります。

```
# /etc/init.d/sshd restart
        または
# service sshd restart
```

1-4. SSH1 と SSH2

SSH プロトコルには 2 つのバージョンがあります。SSH1(バージョン１)はぜい弱性が報告されており利用は推奨されません。SSH2 は、SSH1 に比べて認証方式が改良され、サポートする暗号のアルゴリズムが DSA, RSA の 2 つに増強されています。

サポートしているバージョンをクライアント・サーバー双方で合わせる必要がありますが、これは通信の開始時に行われます。

ssh コマンドに-v オプションをつけると、サーバーにログインするまでのサーバーとクライアント間でなされるやり取りを逐一表示することができます。

```
$ ssh -v localhost
OpenSSH_4.3p2, OpenSSL 0.9.8e-fips-rhel5 01 Jul 2008
debug1: Reading configuration data /etc/ssh/ssh_config
debug1: Applying options for *
debug1: Connecting to localhost [127.0.0.1] port 22.
debug1: Connection established.

debug1: permanently_set_uid: 0/0
```

```
debug1: identity file /root/.ssh/identity type -1
debug1: identity file /root/.ssh/id_rsa type -1
debug1: identity file /root/.ssh/id_dsa type -1
debug1: loaded 3 keys
debug1: Remote protocol version 2.0, remote software version OpenSSH_4.3
debug1: match: OpenSSH_4.3 pat OpenSSH*
debug1: Enabling compatibility mode for protocol 2.0
debug1: Local version string SSH-2.0-OpenSSH_4.3
                        : （後略）
```

OpenSSH サーバーのプロトコルバージョンは「2.0」、ソフトウエアのバージョンは 4.3。クライアントのプロトコルも「2.0」で、ソフトウエアは 4.3 である事を示しています。
ssh に-1 オプションを付け起動すると、プロトコルバージョン 1.0 で動作します。以下はその例（抜粋）です。

```
$ ssh -v -1 192.168.78.130
                        :
debug1: loaded 1 keys
debug1: Remote protocol version 1.99, remote software version OpenSSH_4.3
debug1: match: OpenSSH_4.3 pat OpenSSH*
debug1: Local version string SSH-1.5-OpenSSH_4.3
                        :
```

今回は、サーバー側プロトコルが「1.99」クライアント側が「1.5」で動作しています。
このように、通信開始時に両者がプロトコルをそろえることにより、異なるバージョンのソフトウエアでも確実な通信を行う事ができます。

1-5. 空パスワード入力の許可

パスワード入力の省略 (空パスワード) を許可するのはセキュリティ上問題となりますが、完全に外部と遮断された環境下や、人ではなくプログラム通信するといった、特異な場合に必要となる場合があります。
例)パスワードのないユーザーで ssh 利用

```
[student@server1 ~]$ su - emptypass
[emptypass@server1 ~]$ id
uid=502(emptypass) gid=502(emptypass) groups=502(emptypass)
[emptypass@server1 ~]$ exit
logout

[student@server1 ~]$ ssh emptypass@localhost
emptypass@localhost's password:
Permission denied, please try again.
```

空パスワードを許可する設定は PermitEmptyPasswords によりますが、ssh だけでなく Linux のセキュリティ設定 PAM の修正も必要となる場合があります。
PAM(Pluggable Authentication Modules)は、ユーザー認証を集中的に行う Linux の機能でパスワード情報の管理や、他の認証サービスとの連携を司ります。PAM を用いることでアプリケーションの修正をすることなく、新しい認証サービスへの移行や、情報の一元管

理が可能となります。

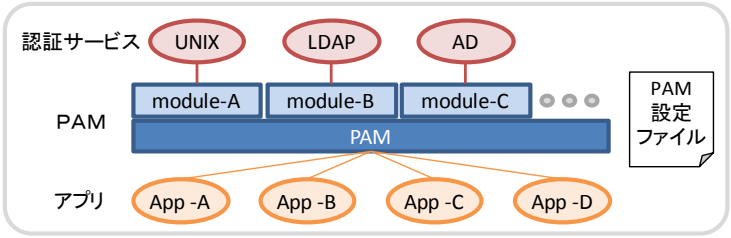


図 4：PAM 概念図

なお、個々のモジュールは/lib/security 下に格納されています。

PAM の設定変更

PAM はシステム全体の認証を制御しています。設定ファイルは/etc/pam.d の下にアプリケーション（コマンドやデーモン）と同じ名称のテキストファイルとして格納されています。

例）/etc/pam.d/sshd(抜粋)

```
#%PAM-1.0
auth       include      system-auth
account    required     pam_nologin.so
password   include      system-auth
session    optional     pam_keyinit.so force revoke
```

はコメント行で、#%は特別に PAM のバージョンを示します。

設定内容は 1 項目 1 行で、空白で区切られた 3 つの領域からなります。左からタイプ、コントロール、モジュール名および引数です。

タイプ　コントロール　　モジュール名（パス）　　[引数…]

タイプとコントロールには以下の種類があります。

表2： PAM のタイプ

タイプ	解説
auth	ユーザー認証（ユーザー名、パスワード、IC カード等で本人かどうかを確認する）
account	ユーザーが有効かどうかを判定する（有効期限、権限など）
password	パスワードの変更と確認方法
session	ユーザー認証の前後に行う処理

タイプは 1 ファイル中に複数記述することができますが、その場合、全てのが成功して初めて処理が成功します。たとえば auth が 3 行あれば、その 3 種類すべての関門を通過して初めてユーザー認証が許可されます。

表3： PAM のコントロール

コントロール	解説
requisite	モジュール認証が NG ならば、即認証プロセスを中止し、認証失敗を返す。
required	モジュール認証が NG なら、残りの同タイプを処理した後、失敗を返す。
sufficient	モジュール認証が OK かつ、それ以前の required が OK ならば成功を返す
optional	同一タイプが 1 つのみ(自分自身)以外は、認証可否に影響を与えない。

CentOS は既定値として空パスワードを許可していますが、SELinux や他のディストリビューションの場合は制限している場合があります。
CentOS の場合は、以下の場所で空パスワードを制御しています。
/etc/pam.d/system-auth

```
#%PAM-1.0
# This file is auto-generated.
# User changes will be destroyed the next time authconfig is run.
auth       required     pam_env.so
auth       sufficient   pam_unix.so nullok try_first_pass
                                    :
```

pam_unix.so モジュールのオプションとして、nullok が設定されていると空パスワードを許可します。このキーワードがない場合は、空パスワードを許可しません。
ディストリビューションやバージョンによっては、別のファイルを用いる場合もありますが内容はかわりません。
/etc/security/sshd

```
auth       required     pam_unix.so shadow nullok nodelaly
```

PAM は定義ファイルを修正すると、即時でその内容が反映されます。

空パスワードアカウントの用意

passwd(1)や useradd(8)では空パスワードを設定できないので、vipw(8)を用いて passwd(5)、shadow(5)のパスワード情報を削除します。
vipw を実行すると最初に/etc/passwd が表示され、修正・保存を行うと、続いて/etc/shadow の編集が始まります。passwd, shadow とも非常に重要なファイルのため、vipw はバックアップを作成した上、他に編集していないか排他制御を行います。これらのファイルを直接編集する場合は必ず vipw を使ってください。
例)ユーザ emptypass を作成し、パスワード情報を削除

```
# useradd emptypass
# grep emptypass /etc/passwd /etc/shadow
/etc/passwd:emptypass:x:502:502::/home/emptypass:/bin/bash
/etc/shadow:emptypass:!!:15163:0:99999:7:::
# vipw
        ( /etc/passwd, /etc/shadow を編集し、パスワード情報を削除)
# grep emptypass /etc/passwd /etc/shadow
/etc/passwd:emptypass::502:502::/home/emptypass:/bin/bash
/etc/shadow:emptypass::15163:0:99999:7:::
```

/etc/passwd の 2 番目の領域を「:x:」から「::」に修正、/etc/shadow も同様に 2 番目の領域を「:!!:」から「::」に修正します。すでに利用を開始しているユーザーの場合はパスワード情報として例えば「:1MGWT0TYG$TKs6kYG51PK27pfVGafRZ/:」といった非常に長い文字列となっています。
この時点で、ユーザーemptypass にコンソールからログインすると、パスワードは聞かれずに、すぐログインできます。

```
CentOS release 5.6 (Final)
Kernel 2.6.18-238.12.1.el5 on i686

Server1  login: emptypass
[emptypass@server1 ~]$
```

・sshd 設定変更

空のパスワードを有効にするため、/etc/ssh/sshd_config を修正します。

sshd_config の設定：

```
PermitEmptyPasswords    yes
```

sshd_config 修正後　sshd を再起動し、動作を確認します。

```
[root@server1 work]# /etc/init.d/sshd restart
Stopping sshd: [  OK  ]
Starting sshd: [  OK  ]


[root@server1 work]# ssh emptypass@server1
Last login: Fri Jul  8 13:51:28 2011 from localhost.localdomain

[emptypass@server1 ~]$ id
uid=502(emptypass) gid=502(emptypass) groups=502(emptypass)
```

2.OpenSSH サーバー

2-1. 公開鍵暗号認証

パスワード認証を用いた SSH によるログインでは、ログイン時に送信されるアカウント情報や通信内容が暗号化されるため、TELNET などに比べ安全性が高いといえます。しかし、暗号化されているとはいえ、ユーザー名とパスワードがネットワーク上を流れるため、盗聴された暗号化データからパスワードを解読される可能性があります。

公開鍵暗号とは

一般に、暗号化のための鍵と復号のための鍵が同じ暗号を用いる方法を「秘密鍵暗号方式」とよびます。

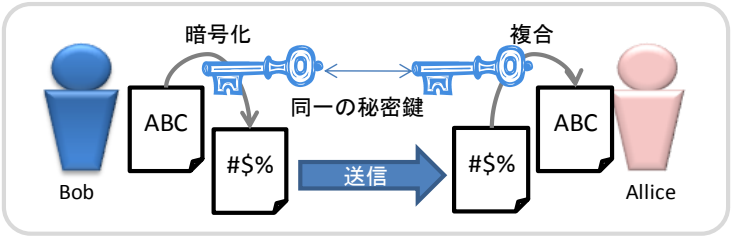


図 5：秘密鍵暗号方式の概念図

送信者は、暗号化したデータとは別に「鍵」を受信者に送る必要があります。送信者と受信者が鍵データを保存したフロッピーディスクなどを直接手渡しできればよいですが、電子メールで送信する場合などには、盗聴にされてしまう可能性があります。安全な通信経路が存在しない状況で、暗号化・復号化に必要な鍵を渡さなければなりません。これを「鍵配送問題」といい、暗号の歴史の中で非常に多くの人を悩ませました。

この鍵配送問題を解決したのが、「公開鍵暗号方式」という種類の暗号です。公開鍵暗号では、暗号化するための鍵（公開鍵）と復号化するための鍵（秘密鍵）の 2 つを用います。公開鍵はその名の通り、公開して誰もが入手できるようにします。暗号化したデータは受信者だけが持っている秘密鍵によってのみ復号できます。Bob が Alice に暗号化したメッセージを送信する場合を考えます。

図 6：公開鍵暗号方式（1）

1. Alice は自分の公開鍵と秘密鍵を作成し、公開鍵を公開します
2. Bob は Alice の公開鍵を入手します。

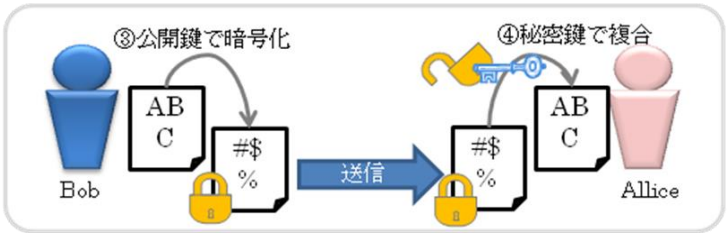


図 7：公開鍵暗号方式（2）

3. Bob はメッセージを、公開鍵で暗号化し Alice に送信します
4. Alice は自分の秘密鍵を用いて、Bob からのメッセージを復号します

公開鍵暗号を用いた認証

SSH ではユーザー認証を行うために公開鍵暗号を用いることができます。公開鍵暗号を利用した認証の手続きは、以下のようなプロセスで行われます。

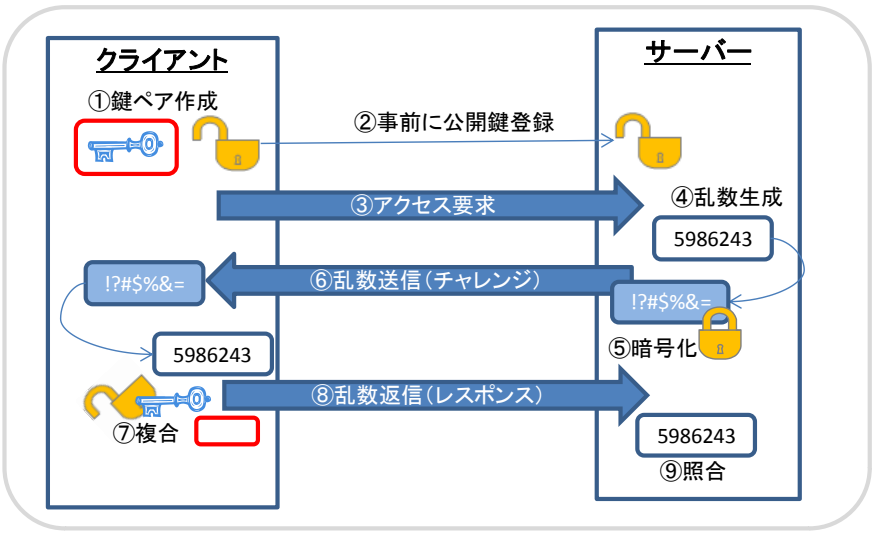


図 8：SSH 通信手順(公開鍵方式)

1. ログイン先の SSH サーバーにあらかじめ、自分の公開鍵を登録する。
   （秘密鍵はコピーされても利用できないよう「パスフレーズ」を使って暗号化）
2. 公開鍵を事前にサーバーへ登録しておく。
3. SSH クライアントで SSH サーバーにアクセスする。
4. SSH サーバーは乱数を発生する。
5. 乱数をクライアントから預かった公開鍵で暗号化する。
6. 暗号化データをクライアントへ送付する。
7. クライアントで受領データを複合し乱数を得る。このとき秘密鍵を使う際に「パスフレーズ」が要求される。
8. 復号した乱数を送り返す（実際にはホスト鍵を使って暗号化する）。
9. 返信された乱数と送ったときの内容を照合し、同じであれば認証ＯＫ

この認証方法は、「ある公開鍵で暗号化されたデータはペアとなる秘密鍵によってのみ、復号される」という性質を利用しています。

2-2. OpenSSH における公開鍵暗号認証

鍵の生成

SSH サーバーに対し、公開鍵暗号認証を行うためには、まず最初にクライアント側で公開鍵と秘密鍵のペアを作成する必要があります。OpenSSH で作成できる鍵のアルゴリズムには以下のようなものがあります。

表4： SSH プロトコルと暗号化アルゴリズム

	SSH1	SSH2
アルゴリズム	RSA	RSA、DSA

現在 SSH1 は脆弱性が報告されており、基本的に SSH2 を用います。
公開鍵と秘密鍵を作成するためには、ssh-keygen コマンドを用います。DSA というアルゴリズムを用いた暗号鍵のペアを作成する場合は、以下のように作成します。

鍵ファイル名の指定を行うことができますが、通常はデフォルトの「~/.ssh/id_dsa」のままにしておきます。

```
$ ssh-keygen -t dsa
Generating public/private dsa key pair.
Enter file in which to save the key (/home/student/.ssh/id_dsa):
```

続いて、パスフレーズの入力を求められます。これは秘密鍵を用いる時に必要となるパスワードのようなものです。パスフレーズを設定することにより、たとえ鍵ファイルが盗まれたとしても、本人以外のユーザーには使用できないことになります。パスフレーズを空文字列にすることもできますが、セキュリティ向上のために他人には容易にわからないような文字列を登録しておきます。

```
Enter passphrase (empty for no passphrase): [パスフレーズを入力]
Enter same passphrase again: [再度パスフレーズを入力]
Your identification has been saved in id_dsa.
Your public key has been saved in id_dsa.pub.
The key fingerprint is:
35:69:a1:27:ae:62:0d:f9:cf:46:12:3c:1a:a8:2c:5a student@172.16.0.200
```

ユーザーのホームディレクトリ以下の.ssh というディレクトリの中に公開鍵「id_dsa.pub」と秘密鍵「id_dsa」というペアが作成できます。

```
$ ls ~/.ssh
id_dsa  id_dsa.pub  known_hosts
$ ls -l ~/.ssh
-rw-------   1 student       student 744     3月  9 16:07 id_dsa
-rw-r--r--   1 student       student 613     3月 15 14:05 id_dsa.pub
-rw-r--r--   1 student       student 2636    3月 15 14:19 known_hosts
```

「id_dsa」のパーミッションは「600(rw-------)」、「id_dsa.pub」のパーミッションは「644(rw-r--r--)」として作成されます。誤って「id_dsa」を削除、変更してしまわないように「400(r--------)」にするのが望ましいとされています。

id_dsa の例

```
-----BEGIN DSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: DES-EDE3-CBC,266C1C2D512E46FB
RbZGyVERCTCxgzxYpEYsIvbm8hGlg1cvVMZYigBIx/KRXu6SsB/PMf70/3YYvM4N
XkznkEznTHUQ930LK3QH9GqQFVuxG3AsfTDxERV5f32PAkeN0Jan9DM+VxKB5uCN
                              :（省略）
w2H3a1dWYw9JrXqtrcahhn0NLehUEHDPNI4rNlGVCxMB8EkKoacdvtSqtGqVlbAZ
4pHqEyoFMcIs2e1Wh5LaQriFwvAw67gV4MHmMiX3a5f+J7fkHUzwW/1pPdrnQPGt
-----END DSA PRIVATE KEY-----
```

id_dsa.pub の例

```
ssh-dss AAAAB3NzaC1kc3MAAAEBAO2/69TZAGteal18auh1dedODSETJuI1Fhia7C6O
8iKrPi6b9CtefE8ppyEpY2UAzHI5Unnn9yerzjakptckKcGbvQRaiU8+RkueDHM21byh
4l7R3xjj/ss6/G265pYay8uSqmu5CUJrXC41vcqjw2zb0C0GiWL4VMm0Ky1FCBwpqmvH
                              :（省略）
LWWaCAcgac2rFe6fgVwHKWgK4uzw0oxJPIgZWiRekSoHxMzeO2a34WNF2+mcMtdIG31g
7uKQHwq2uMK+PxC5NNsW6Qr04LERBll2xtSxQvutozDOxFZ+CQsoUN02RZguhTEkFUHU
dCHERWSu0Wd/rgLoGcAFF9EAHQwFEO+mUEyudjXBNQpF0uA== student@pc001
```

アルゴリズム DSA を用いた場合の秘密鍵・公開鍵ファイルは上のようなテキストファイルで記述された文字列データとなっています。

アルゴリズム RSA を用いる場合もこれと同様で、鍵生成の際に鍵のアルゴリズムとして、「-t rsa」を指定します。なお SSH1 で生成される鍵ファイルのデフォルト値は「identity」と「identity.pub」となります。

SSH サーバーへの公開鍵の登録

公開鍵認証を行うには、ログイン先の SSH サーバーに生成した公開鍵を前もって登録しておく必要があります。管理者の公開鍵ファイルは、フロッピーディスクなどの安全な媒体を用いて SSH サーバー側に移すのがよいとされています。これは、公開鍵が悪意のある人に盗まれると、なりすましをされる危険性が高くなるためです。

ログイン先のユーザーのホームディレクトリ以下の.ssh/authorized_keys というファイルに公開鍵ファイルの内容を書き込むことで、公開鍵の登録が完了します。

```
$ cat ~/.ssh/id_dsa.pub >> ~/.ssh/authorized_keys
```

上のコマンドを入力することにより、authorized_keys ファイルがない場合、新たに作成され、既にある場合は内容が追加されます。

authorized_keys ファイルは上のように作成した場合、パーミッションは「664」で作成されます。グループ内の他のユーザーに変更されないようにするために「644」にする必要があります。

公開鍵認証を用いた SSH サーバーへのログイン

SSH サーバーに公開鍵を登録したら、ログイン準備の完了です。ログインのためには、前回と同様に ssh コマンドを用います。

すると、自分の秘密鍵を使うためのパスフレーズが求められます。パスフレーズを入力すれば、ログインが完了します。秘密鍵は~/.ssh ディレクトリ内の id_ds または id_rsa です。

```
$ ssh 172.16.0.200
Enter passphrase for key '/home/student/.ssh/id_dsa':
Last login: Fri Mar 15 08:01:31 2002 from 172.16.0.100
[student@200 student]$
```

別の秘密鍵を指定して、SSH サーバーにログインしたい場合は、以下のように「-i」オプションに続けて、鍵ファイルを指定します。

```
$ ssh -i ~/.ssh/student.dsa 172.16.0.200
```

上の例では、秘密鍵の名前を「student.dsa」というファイル名を指定した場合を示しています。

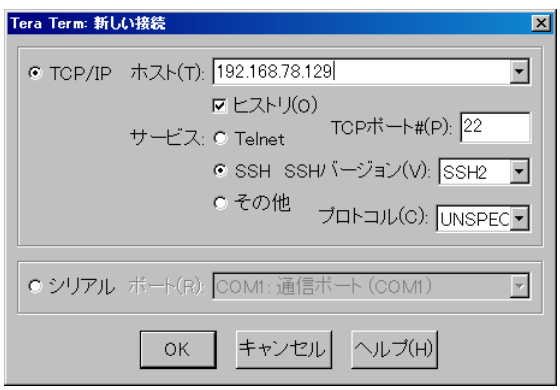
Windows 上での SSH クライアントの利用

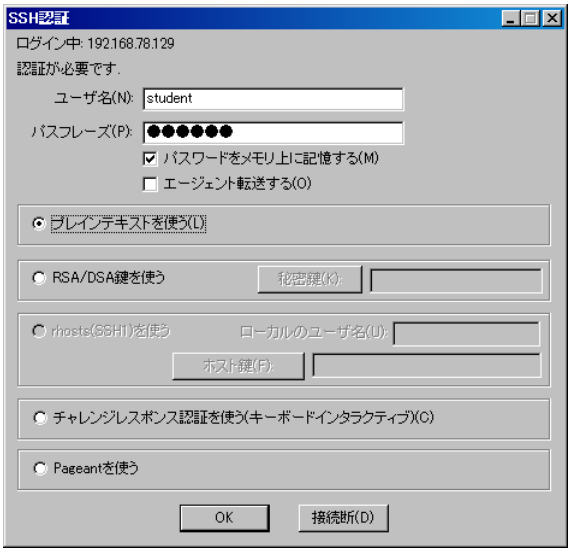
Windows マシンから SSH を使ってサーバーにログインする際には、Windows 用の SSH クライアント（ターミナルエミュレータ）が必要となります。よく利用されているソフトウエアとしては、TeraTerm と Putty があります。
両者ともフリーで、インターネット上から入手することができます。

・TeraTerm（テラターム）

http://sourceforge.jp/projects/ttssh2/

TeraTerm のログイン画面は以下のようになっており、Telent と SSH が選択できるようになっています。また SSH での接続を選択すると、さらにプレンテキスト(パスワード認証)、共通鍵認証を選択できる画面が表示されます。





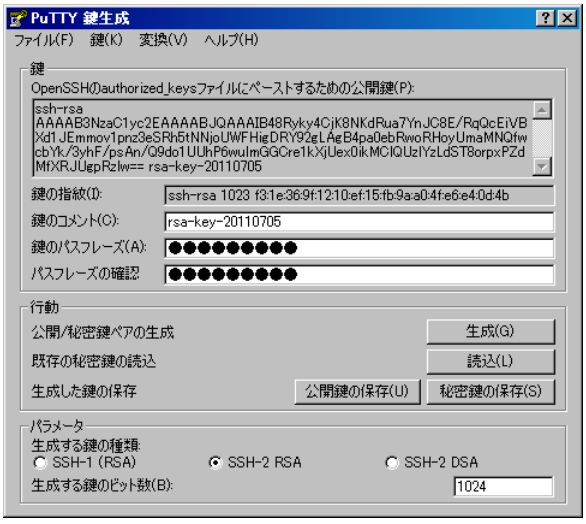
鍵ペアの作成は、メニュー設定（S）＞SSH 鍵生成(N)...から鍵生成画面を呼び出し、生成後、2 つの鍵をそれぞれファイルとして保存します。

TeraTerm には他にも、SCP によるファイル転送など数多くの機能が含まれます。

・PuTTY(パティ)

http://yebisuya.dip.jp/Software/PuTTY/

TeraTerm は非常に歴史のターミナルエミュレータでしたが、SSH への対応が遅かったため、PuTTY も良く使われるようになりました。現在ではこの 2 者がよく使われています。鍵の生成は、専用のプログラムで作成するようになっています（Pu TTYgen)。



SSH サーバー側の設定

インストールされた SSH サーバーはデフォルトで公開鍵認証方式を許可する設定となっています。これらの設定も SSH サーバーの設定ファイルである、「/etc/ssh/sshd_config」ファイルにて行います。

表5： 公開鍵認証を許可するための設定項目

項目名	説明
RSAAuthentication	SSH1 の RSA 認証を許可する。 yes または no で指定する（デフォルトは yes）
PubKeyAuthentication	SSH2 の DSA 認証または RSA 認証を許可する yes または no で指定する（デフォルトは yes）
AuthorizedKeysFile	SSH サーバーが参照するユーザーの公開鍵のパス ホームディレクトリからの相対パスで指定する

デフォルトの設定ファイルでは、これらの設定はコメントアウトされているか、記述されていないかのいずれかになっています。

ホスト認証鍵

ホスト認証は、sshd が生成した公開鍵を用いて通信しています。このキー・ペアは HostKey で指定され、既定値は以下の値で、sshd 起動時になければ生成されます。

- /etc/ssh/ssh_host_key, ssh_host_key.pub　　SSH1 鍵
- /etc/ssh/ssh_host_dsa_key, ssh_host_dsa_key.pub　　SSH2 DSA 鍵
- /etc/ssh/ssh_host_rsa_key, ssh_host_rsa_key.pub　　SSH2 RSA 鍵

/etc/ssh/sshd_config：

```
# HostKey for protocol version 1
#HostKey /etc/ssh/ssh_host_key
# HostKeys for protocol version 2
#HostKey /etc/ssh/ssh_host_rsa_key
#HostKey /etc/ssh/ssh_host_dsa_key
```

例）ホスト認証鍵の再作成

```
[root@server1 ~]# rm -f /etc/ssh/ssh_host*
[root@server1 ~]# /etc/init.d/sshd restart
Stopping sshd: [OK]
Generating SSH1 RSA host key: [OK]
Generating SSH2 RSA host key: [OK]
Generating SSH2 DSA host key: [OK]
Starting sshd: [OK]
```

公開鍵の所有者が正しいかどうかを確認するには、その要約である「指紋(fingerprint)」を用いるのが便利です。ssh-keygen の l オプションにより指紋を確認することができます。

```
[root@server1 ~]# ssh-keygen -lf /etc/ssh/ssh_host_rsa_key
2048 35:69:a1:27:ae:62:0d:f9:cf:46:12:3c:1a:a8:2c:5a
  /etc/ssh/ssh_host_rsa_key.pub
```

2-3. 安全なファイル転送 (scp, sftp)

SSH クライアントと SSH サーバー間の暗号化された通信路を用いると、盗まれる危険性の低いファイル転送を行うことができます。これを実現するためのツールが scp と sftp です。

scp コマンド

scp コマンドは、暗号化された経路を用いて、ローカルのファイルとリモートのファイルの間でファイルのやりとりを行うためのコマンドです。cp コマンドの様に使えますが、ファイルにリモートホストのファイルを指定することができます。

scp コマンドは、クライアント側で実行します。サーバー側からクライアント側に送ることも、クライアント側からサーバー側に送ることも可能です。

```
scp ［コピー元のファイル］ ［コピー先のファイル］
```

リモートファイルの指定は以下のように行います。

```
［ユーザー名］@［ホスト名］:［ファイル名］
```

例：ホスト 172.16.0.100 のファイル/etc/hosts をカレントディレクトリ（.）にコピーする

```
scp student@172.16.0.100:/etc/hosts .
```

scp コマンドを用いるときにユーザー認証が行われますが、これは ssh コマンドと同様です。

sftp

sftp も scp コマンドと同様に OpenSSH 付属のツールです。scp とは異なり、sftp は対話式でファイル転送を行えます。sftp サーバーは SSH サーバーの付属の機能として、実行することができます。

/etc/ssh/sshd_config ファイルの末尾に以下のような行の記述があります。

```
Subsystem   sftp  /usr/libexec/openssh/sftp-server
```

このように記述されていると、sshd 起動時に sftp サーバー「/usr/libexec/openssh/sftp-server」も実行されます。

クライアントから sftp サーバーに接続するためには、sftp コマンドを用います。sftp コマンドは ftp コマンドと同様にログイン終了後、get コマンドや put コマンドを利用することにより、ファイルのやりとりが可能になります。

3.SSH の応用的な利用

3-1. 認証エージェントの利用

認証エージェント(ssh-agent) を利用すればパスフレーズを記憶し、ssh コマンド実行時にユーザーに代わってパスフレーズを代入してくれます。ssh と接続するためにいくつか環境変数を設定する必要がありますが、ssh-agent 実行時に必要な処理が画面に表示されるので、eval コマンドを使って実行します。

```
[student@server2 ~]$ ssh server1 uname –a
Enter passphrase for key '/home/student/.ssh/id_dsa':
Linux server1.localdomain 2.6.18-238.12.1.el5 #1 SMP Tue May 31 13:23:01 EDT
2011 i686 i686 i386 GNU/Linux

[student@server2 ~]$ eval `ssh-agent`
Agent pid 7433
[student@server2 ~]$ ssh-add ~/.ssh/id_dsa
Enter passphrase for /home/student/.ssh/id_dsa:
Identity added: /home/student/.ssh/id_dsa (/home/student/.ssh/id_dsa)
[student@server2 ~]$ ssh server1 uname –a
Linux server1.localdomain 2.6.18-238.12.1.el5 #1 SMP Tue May 31 13:23:01 EDT
2011 i686 i686 i386 GNU/Linux
```

パスワードエージェントを作動させ、ssh-add でパスフレーズを登録すれば、以降パスフレーズの入力は省略できます。

終了するには、ssh-agent 実行時に表示された PID を元に kill するか、-k オプション付きの ssh-agent を実行します。

```
[student@server2 ~]$ eval `ssh-agent –k`
Agent pid 7433 killed
```

-t オプションを指定し認証エージェントの有効期限を設定することもできます。指定しない場合は無限大（デフォルト。有効期限を設定しない）となります。

```
[student@server2 ~]$ eval `ssh-agent -t 1m`
Agent pid 13225
[student@server2 ~]$ ssh-add ~/.ssh/id_dsa
Enter passphrase for /home/student/.ssh/id_dsa:
Identity added: /home/student/.ssh/id_dsa (/home/student/.ssh/id_dsa)
[student@server2 ~]$ derver1 date
Fri Jul 15 13:57:36 JST 2011
[student@server2 ~]$ ssh server1 date
Enter passphrase for key '/home/student/.ssh/id_dsa':
Fri Jul 15 13:59:38 JST 2011
```

なお、X11 環境では ssh-askpass を用います。

3-2. ポートフォワーディング

SSH は暗号化した通信経路を形で提供することで、リモートホストへのログインやリモートホスト上での操作の安全性を高めます。この暗号化された通信経路を他のサービスに応用する機能をポートフォワーディング（ポート転送）と言います。

たとえば、メールサーバーとクライアント間の通信経路を暗号化することができれば、より安全なメールの送受信が可能になります。最も安全なメールの送受はメッセージ自体を暗号化[5]することですが、ポートフォワーディングにより通信経路自体を暗号化することができます。

ポートフォワーディングは接続先のホストのポートとローカルホストのあるポートの間に予め暗号化された通信経路を用意しておくものです。

図 9：POP における通信経路

POP3 プロトコルによるメールの読み出しは、メールクライアントと POP3 サーバーの間の通信により行われます。上図はリモートサーバーの POP3 のポートと、ローカルホストのあるポート（動的に割り当）の間に通信経路を張る様子を示しています。

同様にポートフォワーディングを使った場合は数のようになります。

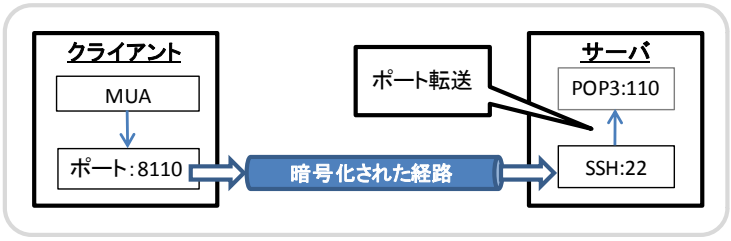


図 10：ポートフォワーディングを使った POP の通信経路

この例ではサーバーの 110 番ポートと、クライアントの 8110 番ポートの間に暗号化した通信経路を用意し、メールプログラムはサーバーの POP ポートではなく、すでに確立している暗号化経路の入り口（自分のポート 8110）と接続しています。

Linux での操作例

ポートフォワーディングは、ssh コマンドの-L オプションを用います。

```
ssh -L [localhost 上の適当なポート]:[接続先のホスト]:[接続先のポート] 接続先のホスト
```

例と同じように、自身のポート 8110 と POP サーバーの 110 ポート間に暗号化経路を確立させる場合、以下のようになります。

---

[5] メールメッセージ本文を暗号化するためのツールとしては、PGP（Pretty Good Privacy、http://www.pgpi.org/ 、日本サイト http://www006.upp.so-net.ne.jp/naoki-s/pgpi/）が有名。

```
$ ssh -L 8110:server1:110 server1
Enter passphrase for key '/home/student/.ssh/id_dsa':
Last login: Fri Jul  8 15:49:09 2011 from client.localdomain

[student@server1 ~]$
```

このコマンドを実行すると、ssh により server1 へログインします。この時点で暗号化された経路が確立しています。終了するには、ログアウトします。

コンソールが接続されたままの状態になるため、動作確認するには、別のコンソールを使います。まずポートの利用状況を確認します。

```
$ netstat –atn
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address      Foreign Address     State
tcp       0      0 127.0.0.1:8110        0.0.0.0:*           LISTEN
tcp       0      0 0.0.0.0:5560          0.0.0.0:*           LISTEN
tcp       0      0 192.168.78.130:48599  192.168.78.129:22   ESTABLISHED
tcp       0      0 ::1:8110              :::*                LISTEN
tcp       0      0 :::22                 :::*                LISTEN
```

ローカルの 8110 ポートが待ち受けている事がわかります。また SSH のセッションが張られている事もポイントです。

続いて telnet を使い POP に接続します。直接 110 ポートに接続した場合と、8110 ポートを経由した場合を確認します。

例）POP サーバーに直接接続した場合

```
[student@client ~]$ telnet server1 110
Trying 192.168.78.129...
Connected to server1.localdomain (192.168.78.129).
Escape character is '^]'.
+OK Dovecot ready.
```

例）POP サーバーに暗号化ポート(8110)経由で接続した場合

```
[student@server2 ~]$ telnet localhost 8110
Trying 127.0.0.1...
Connected to localhost.localdomain (127.0.0.1).
Escape character is '^]'.
+OK Dovecot ready.
```

暗号化ポートで接続した場合は、サーバー側の内部でポート転送されているため、POP サーバーはローカル接続された状態になっています（Connected to localhost）。

Windows での操作

TeraTerm でポート転送を使うには、一旦 SSH でログインし、SSH のセッションを確立してから、メニュー：設定(S)＞SSH 転送(O)...を選びます。

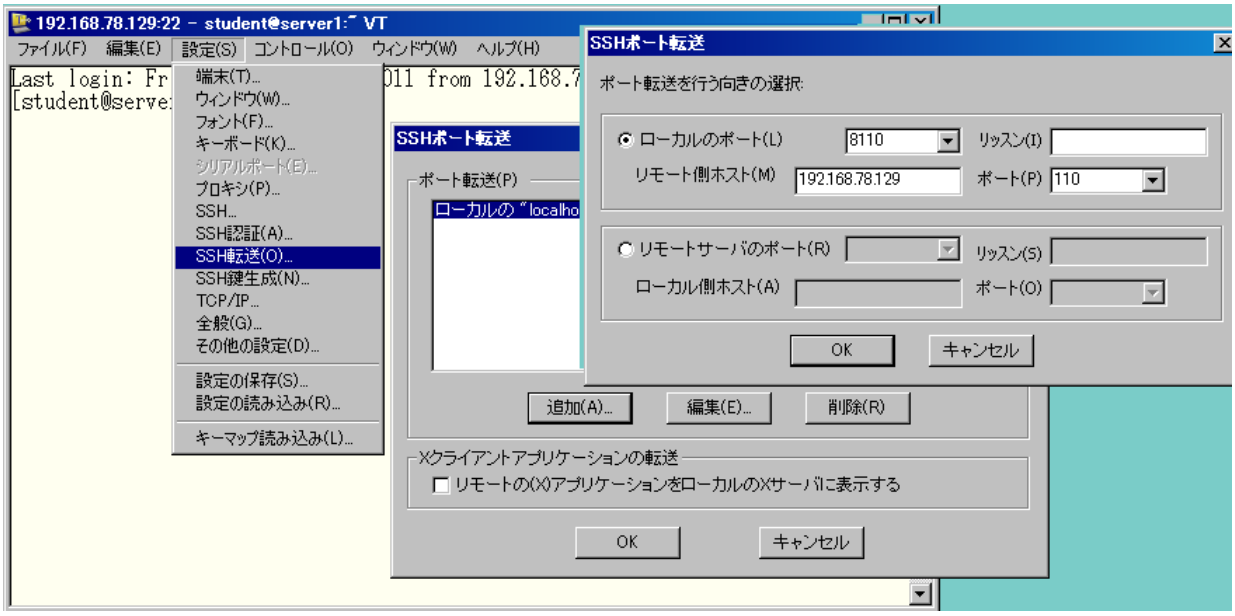


図 11：TeraTerm ポートフォワーディング設定

「SSH ポート転送」画面で、[追加(A)...]ボタンを押し、「ポート転送を行う無期の選択」にて、ローカルのポートを選択し、以下の項目を入力します。

・ローカルのポート　　：8110
・リモート側ホスト　　：POP サーバー名または IP アドレス
・ポート　　：110

この状態で、コマンドプロンプトを起動しローカルの 8110 に接続します。

DOS> telnet localhost 8110

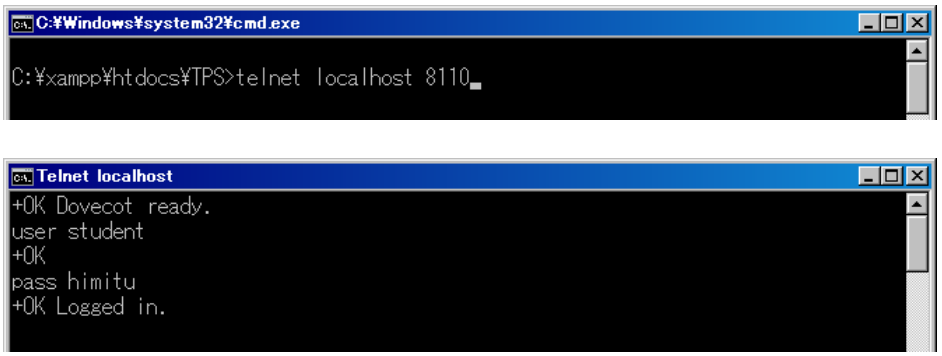


図 12：DOS でのポートフォワード実行例

3-3. tcpdump による動作確認

実際に暗号化されているかどうか、サーバー側で通信パケットを追跡してみます。
tcpdump はNICのデータを直接解析するツールでホストやポートを指定し必要なパケットを抽出します。たとえば 110 ポートを監視するのであれば、次のようにタイプします。

```
# tcpdump -l -x –i eth0 port 110
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 1600 bytes
16:31:09.660809 IP server2.localdomain.36312 > server1.localdomain.pop3: S
782458348:782458348(0)  win  5840  <mss  1460,sackOK,timestamp  20920126
0,nop,wscale 7>
        0x0000:  4510 003c 6ad2 4000 4006 b185 c0a8 4e82
        0x0010:  c0a8 4e81 8dd8 006e 2ea3 5dec 0000 0000
        0x0020:  a002 16d0 bf88 0000 0204 05b4 0402 080a
        0x0030:  013f 373e 0000 0000 0103 0307
```

主なオプション

- -l　　標準出力のバッファリングを行う（パイプライン処理時に必要）
- -x　　パケットの内容を 16 進数ダンプする
- -i NIC　インターフェースを指定

オプションに続いて、抽出条件を指定します。今回はアドレスに関係なく 110 番ポートを監視しています。文字列のダンプオプションがないため、tcpdump の解析には Wireshark などの専用ソフトを用います。今回は手製のスクリプト(tcpdfilter.pl)を用い、実際の処理を監視してみます。

```
# tcpdump -l  -x port 110 | ./tcpdfilter.pl
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 1600 bytes
IP server2.localdomain.36313 > server1.localdomain.pop3: S 833952330

IP server1.localdomain.pop3 > server2.localdomain.36313: S 175169766
E__<__@_@_Fu__N___N____n1__J________g___________?__________
                        :（中略）
IP server2.localdomain.36313 > server1.localdomain.pop3
E__H__@_@__t__N___N__n___p__1__K___.y___________?__+OK Dovecot ready.__
IP server2.localdomain.36313 > server1.localdomain.pop3: P 1
E__4__@_@_F{__N___N____n1__K_p_____._________?______
IP server1.localdomain.pop3 > server2.localdomain.36313
E__B__@_@_Fl__N___N____n1__K_p_____._>_______?_?____user student__
IP server1.localdomain.pop3 > server2.localdomain.36313: P 21
E__4__@_@_____N___N__n___p__1__Y___._6__________?_?
IP server2.localdomain.36313 > server1.localdomain.pop3
E__9__@_@_____N___N__n___p__1__Y___.I___________?_?+OK__
IP server2.localdomain.36313 > server1.localdomain.pop3: P 15
E__4__@_@_Fy__N___N____n1__Y_p_____._1_______?_?____
IP server1.localdomain.pop3 > server2.localdomain.36313: P 26
E__A__@_@_Fk__N___N____n1__Y_p_____.kI_______?______pass himitu__
                        :（後略）
```

tcpdump の 16 進ダンプを文字列に変換するプログラム

```
#!/usr/bin/perl
# tcpdump -l -x -i eth0 src host 192.168.78.1 and port 22 | $0
while(<STDIN>){
     if(m/^[ ¥t]/){
     # save hex dump data
             s/^.*:[ ]+//;
             $store .= $_;
     } else {
             s/^[^ ]* //;    # delete Time Stamp
             s/:[^:]*$//;    # delete :xxxxx string
             print "$_¥n";   # print line (not starting with TAB)

             $_ = $store;

             s/¥s//g;        # remove spaces and join
             # s/^.{120}//;
             s/[0-9A-Fa-f][0-9A-Fa-f]/pack("C", hex $&)/eg; #   hex   ->
char
             s/[¥x00-¥x1f¥x7f-¥xff]/_/g;
             print "$_¥n";
             $store = "";
     }

}
exit;
```

http://ycos.sakura.ne.jp/LA/Special/tcpdfilter.pl

3-4. ssh サーバー設定時の留意点

ssh サーバ設定の既定値には脆弱性があるため、以下の設定を施すことが推奨されています。

ssh サーバ設定推奨値

設定名	設定値	解説
PasswordAuthentication	no	パスワード認証の禁止 =公開鍵認証のみ
ChallengeResponseAuthentication	No	同上
PermitRootLogin	no	ルートでの直接ログイン禁止
Protocol	2	SSH1 の利用禁止
Port	22 以外の値 (1022 など)	外部からの攻撃対策
X11Forwarding	no	X11 用ポート転送禁止
AllowUsers	利用ユーザ	ssh を利用できるユーザを限定する。