LinuC

LinuC レベル1 技術解説セミナー
2022/3/12 開催

暗号の利用

主題	1.10：セキュリティ
副題	1.10.3 暗号化によるデータの保護

三澤 康巨

LPI-JAPAN

本日の講師

三澤 康巨

- KDDI株式会社で、電話等のネットワークサービス設備のエンジニアリングをはじめ様々な業務を担当しました。
- 2017年10月から2年半、KDDIグループ内のサーバ研修講師を務めました。
- サーバ研修受講者の中から、200名を超える LinuC レベル1 合格者を出しました。
- 2020年3月、KDDIを定年退職しました。
- LinuCレベル1技術解説セミナーの講師を担当
  - 2020年7月18日「ブートプロセスとsystemd」
  - 2021年1月23日「ハードディスクのレイアウトとパーティション」
  - 2021年3月　6日「ファイルシステムの作成と管理、マウント」
  - 2021年6月26日「テキストデータ処理」
  - 2022年1月15日「セキュリティ管理」
- その他
  - 2020年11月28日、オープンソースカンファレンス２０２０ オンライン/福岡「Linuxマシンを作ってみよう　～LinuC レベル１／レベル２ 学習環境構築ガイド～」の講師を担当

LinuCレベル１／レベル２ Version10.0とは

■LinuCとは

クラウド時代の即戦力エンジニアであることを証明するLinux技術者認定

✓現場で「今」求められている新しい技術要素に対応

- オンプレミス／仮想化・コンテナを問わず様々な環境下でのサーバー構築
- 他社とのコラボレーションの前提となるオープンソースへの理解
- システムの多様化に対応できるアーキテクチャへの知見

✓全面的に見直した「今」身につけておくべき技術範囲を網羅

今となっては使わない技術やコマンドの削除、アップデート、新領域の取り込み

✓Linuxの範疇だけにとどまらない領域までカバー

セキュリティや監視など、ITエンジニアであれば必須の領域もカバー

クラウドを活用できるITエンジニアに必須の技術がまとまっている

AWSなどのパブリッククラウドを活用するための技術

間が欠けている状態

オンプレミスのサーバーサイドLinux技術

【今まで／その他】

AWSなどのパブリッククラウドを活用するための技術

仮想マシン／コンテナ技術、クラウドセキュリティ、アーキテクチャ、ほか

オンプレミスのサーバーサイドLinux技術

LinuC Version10.0

暗号の利用

1.はじめに

2.暗号の種類と特徴

3.SSHによる安全なリモート接続

4.GnuPGによるファイルの保護

1.はじめに

2.暗号の種類と特徴

3.SSHによる安全なリモート接続

4.GnuPGによるファイルの保護

はじめに

- 本セミナーでは、主題1.10の中から、「暗号の利用」について解説します。
- 盗聴や改竄から情報を守るための仕組みが暗号です。本セミナーでは、暗号を利用するための基礎知識として、暗号の種類と特徴、安全なリモート接続、ファイルの保護について学習します。

- 学習効果を高めるため、実行例の出てくる部分では、ご自分でも実行してみることをお勧めします。

- Linuxには多数のディストリビューションが存在しますが、本セミナーの実行例では、CentOS 7 を使用します。
  - ビジネス用サーバーの多くで稼働している Red Hat Enterprise Linux 7 (RHEL7) と互換性があります。
  - RHEL7は有料ですが、CentOS 7 は無料で利用できます。
  - CentOSのバージョンは「Stream 9」までありますが、本セミナーでは、安定版であるバージョン「7」を使います。

はじめに

■CentOS 7 に基づく学習環境の構築方法を、LPI-Japanのサイトでご紹介しています。

LinuC レベル１／レベル２ Version 10.0 学習環境構築ガイド

https://linuc.org/docs/v10/guide_text.pdf

■学習環境構築ガイドでは、2種類の環境の構築方法を紹介しています。

【環境A】

- 用意したコンピュータの内蔵ストレージを上書きして、Linux専用コンピュータを構築します。
- WindowsやMacOS等の既存OSは使えなくなります。
- 不要になった古いPC等がある場合に、それを使ってください。

【環境B】

- WindowsやMacOS等の既存OSを壊すことなく、外付けSSDにLinuxをインストールします。
- これによって、既存OSとLinuxとの間を切り替えて利用することができます。
- 但し、既存OSとLinuxとを同時に利用することはできません。

※他に、VirtualBox等も紹介しています。

1.はじめに

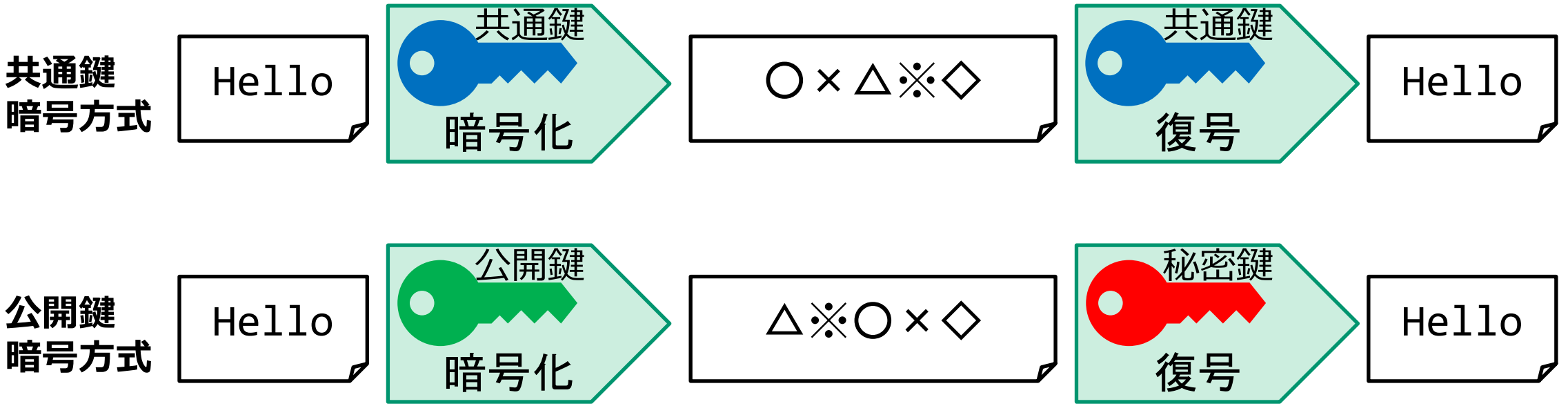
2.暗号の種類と特徴

3.SSHによる安全なリモート接続

4.GnuPGによるファイルの保護

共通鍵暗号方式と公開鍵暗号方式

- 暗号の仕組みには、共通鍵暗号方式と公開鍵暗号方式とがあります。
- 共通鍵暗号方式では、暗号化と復号とに同じ鍵（共通鍵）を使用します。単純なので、処理負荷が軽いという利点がありますが、鍵を安全に共有する方法に課題があります。
- 公開鍵暗号方式では、秘密鍵と公開鍵という鍵のペアを使用します。一方の鍵で暗号化された情報は、ペアのもう一方の鍵でしか復号できません。秘密鍵は安全に保管する必要がありますが、公開鍵は公開可能で容易に共有できるのが利点です。一方、処理負荷は重くなります。



主要な暗号化アルゴリズム

■公開鍵暗号方式では、下表の暗号化アルゴリズムがよく利用されています。

暗号化アルゴリズム	説　明
ed25519	新しいアルゴリズム。短い鍵で強い暗号を実現。
ecdsa	dsaを改良。短い鍵で高速処理を実現。
rsa	SSHに最初に採用されたアルゴリズム。
dsa	古いアルゴリズム。

1.はじめに

2.暗号の種類と特徴

3.SSHによる安全なリモート接続

4.GnuPGによるファイルの保護

SSHの概要

- リモートホストに接続するために、古くはTelnetが使われていました。しかし、Telnetでは通信経路上をデータがそのまま（平文で）流れるため、盗聴による情報漏洩のリスクがあります。
- 今では、盗聴への対策を施したSSH (Secure Shell) プロトコルが多く使われています。
- Linuxの主なディストリビューションは、SSHプロトコルの実装として OpenSSH を採用しています。

- OpenSSHを利用してリモートホストへ安全にログインするには、sshコマンドを使用します。

ssh コマンド

`ssh [オプション] [ユーザ名@]リモートホスト　※ユーザ名を省略するとローカルホストと同じユーザ名`

オプション	説　明
-l ユーザ名	ユーザ名を指定（ユーザ名@ と同じ）
-i 秘密鍵	秘密鍵を指定
-L	ポート転送を指定
-A	ssh-agentによる鍵転送を指定

SSHの概要

- OpenSSHを利用してリモートホストとの間でファイルを安全に送受信するには、scpコマンドを使用します。

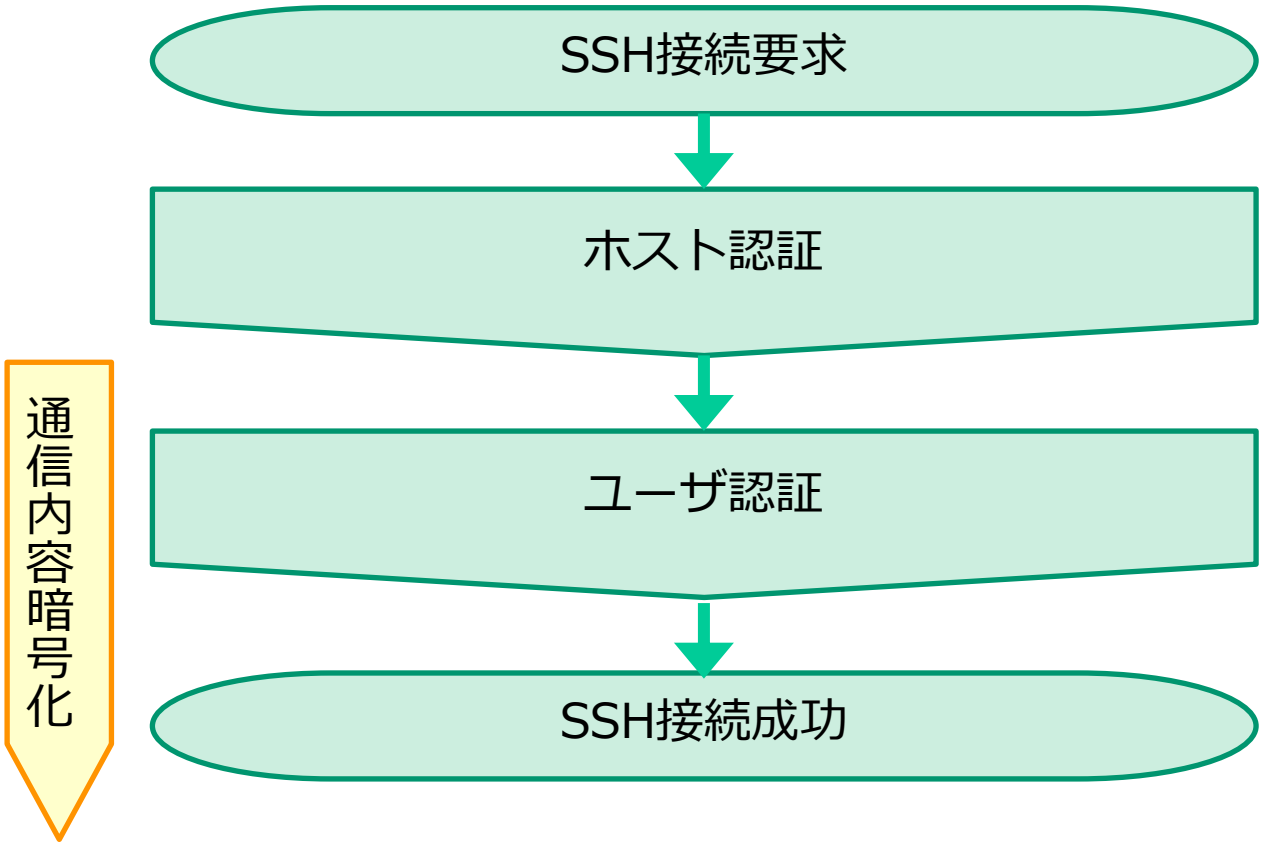
scp コマンド

```
scp [オプション] コピー元 コピー先
```

オプション	説　明
-i 秘密鍵	秘密鍵を指定
-p	コピー元ファイルの属性を維持してコピーする
-r	ディレクトリを再帰的にコピーする

SSHリモート接続フロー

- SSHプロトコルによるリモート接続では、リモート接続を要求する側をSSHクライアント、リモート接続を受け入れる側をSSHサーバと呼びます。
- 接続の際には、ホスト認証、ユーザ認証の順で認証が行われ、安全性を確保します。ホスト認証が成功すると、以降のクライアント～サーバ間通信内容が暗号化されて、情報漏洩を防止します。



sshコマンド

正しいサーバである（なりすましていない）ことを確認

正当なユーザであることを確認

ホスト認証

- ホスト認証は、SSHサーバが正しいサーバである（なりすましていない）ことを確認する手続きです。

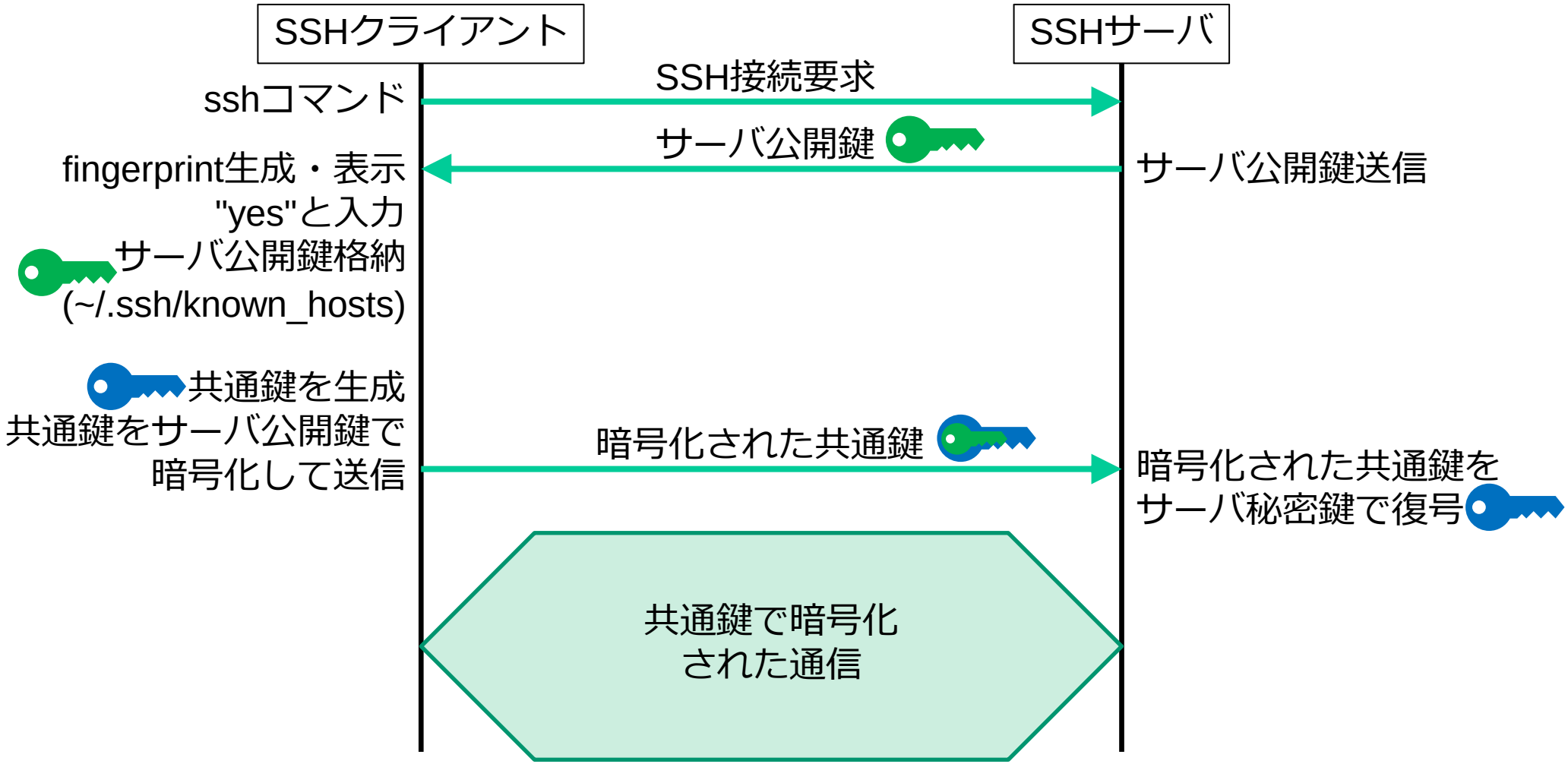
（以降、次スライドの図を参照）

- SSHサーバが接続要求を受け付けると、サーバの公開鍵（※）を送信します。この公開鍵から生成されたfingerprintがクライアント側に表示され、確認を求められます。何らかの手段で予め入手しておいた正しいfingerprintと一致していれば、サーバは正しいサーバなので、クライアントユーザは"yes"と入力して、認証を完了します。
- ホスト認証が完了すると、サーバの公開鍵がクライアント側の ~/.ssh/known_hosts ファイルに格納されます。次回以降の接続時には、格納された公開鍵を参照してホスト認証が自動的に行われ、クライアントユーザへの確認が省略されます。

※SSHサーバの公開鍵と秘密鍵は、OpenSSHのインストール時に作成されます。

ホスト認証

■クライアントでは、共通鍵をランダムに生成し、サーバの公開鍵で暗号化して、サーバへ送信します。サーバは自身の秘密鍵で復号して、共通鍵を入手します。以降、共通鍵を使ってクライアント～サーバ間通信内容が全て暗号化されます。



ユーザ認証

- ホスト認証が成功すると、次にユーザ認証が行われて、正当なユーザであることが確認されます。
- ユーザ認証には、パスワード認証と公開鍵認証とがあります。
- サーバ側にクライアントユーザの公開鍵が格納されている場合、公開鍵認証によるユーザ認証が自動的に行われ、接続が許可されます。（後述）
- サーバ側にクライアント公開鍵が格納されていなければ、クライアント側でパスワード入力が要求されます（パスワード認証）。

SSHリモート接続例（パスワード認証）

SSHリモート接続（パスワード認証）

```
$ ssh demouser@demo-server
The authenticity of host 'demo-server (192.168.122.8)' can't be established.
ECDSA key fingerprint is SHA256:B3sKdV+R5BS9uSGe+Xz3HOP6s7sCjje1+hlOpRrJIw8.
ECDSA key fingerprint is MD5:56:22:83:b8:04:22:52:a1:a9:06:e6:8f:dd:01:7f:40.
Are you sure you want to continue connecting (yes/no)? yes      ※fingerprintを確認
Warning: Permanently added 'demo-server,192.168.122.8' (ECDSA) to the list of known hosts.
demouser@demo-server's password:       ※パスワード認証
Last login: Sat Feb  5 20:51:08 2022 from gateway

demo-server$ uname -n
demo-server

demo-server$ id
uid=1000(demouser) gid=1000(demouser) groups=1000(demouser) context=・・・

demo-server$ exit
ログアウト
Connection to demo-server closed.
```

SSHリモート接続例（パスワード認証）

SSHリモート接続（パスワード認証）

```
$ cat ~/.ssh/known_hosts
demo-server,192.168.122.8 ecdsa-sha2-nistp256
AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBKYwwYZx0kr5/FbHNUFEnlu9gwFuqfkKGnORC
qg1NY8nNcTLmMoHiDGBkNaAr05qst2FPR4cTluvMwQPxn33B9Q=      ※サーバの公開鍵が格納されている

$ ssh demouser@demo-server
demouser@demo-server's password: Ctrl-c      ※fingerprint確認を求められない

$
```

公開鍵認証によるユーザ認証

（次スライドの図を参照）

- 公開鍵認証によるユーザ認証を利用するには、クライアント側でクライアントユーザの鍵ペアを生成し、その公開鍵をサーバ側のログインアカウントの ~/.ssh/authorized_keys ファイルに格納します。
- SSH接続を要求して、ホスト認証が終了すると、公開鍵認証によるユーザ認証が行われます。認証が成功すると、パスワード入力なしでログインが完了します。
- 公開鍵認証の原理は概ね次の通りです。クライアントが乱数を生成し、クライアントの秘密鍵で暗号化（電子署名）して、サーバへ送信します。サーバは格納されているクライアント公開鍵で復号します。復号（電子署名を検証）できれば、クライアントユーザはペアの秘密鍵を持っている正当なユーザなので、ログインを許可します。

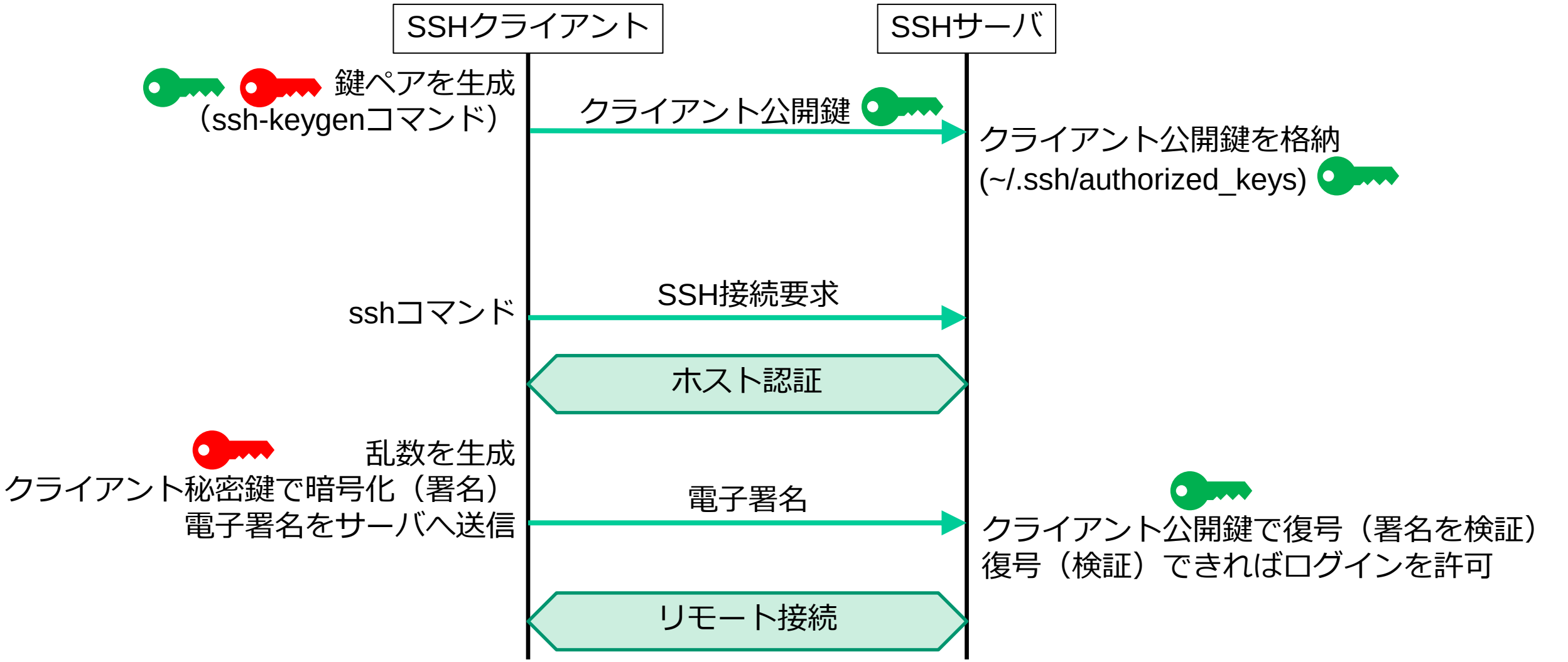
- 鍵ペアを生成するには、ssh-keygenコマンドを使用します。

ssh-keygen コマンド

```
ssh-keygen [オプション]
```

オプション	説　明
-t 暗号化アルゴリズム	鍵ペアの暗号化アルゴリズムを指定
-b ビット数	鍵のビット数を指定

公開鍵認証によるユーザ認証



公開鍵認証によるユーザ認証

SSHリモート接続（公開鍵認証：鍵ペア生成）

```
$ ssh-keygen -t ecdsa -b 521      ※鍵ペアを生成
Generating public/private ecdsa key pair.
Enter file in which to save the key (/home/user1/.ssh/id_ecdsa): (ENTER)
Enter passphrase (empty for no passphrase): (パスフレーズを設定)
Enter same passphrase again: (パスフレーズを再度入力)
Your identification has been saved in /home/user1/.ssh/id_ecdsa.
Your public key has been saved in /home/user1/.ssh/id_ecdsa.pub.
The key fingerprint is:
SHA256:xIRaFnBBuyC8WoyUFRhwQhZes6zsZNsRZxxAlzsk3Bk user1@SSD2-CentOS7
The key's randomart image is:
+---[ECDSA 521]---+
|==BB=EO+.        |
 . . .
+----[SHA256]-----+

$ ls ~/.ssh
id_ecdsa  id_ecdsa.pub  known_hosts
 ※秘密鍵    ※公開鍵
```

公開鍵認証によるユーザ認証

SSHリモート接続（公開鍵認証：サーバにクライアント公開鍵を格納）

```
$ scp ~/.ssh/id_ecdsa.pub demouser@demo-server:      ※公開鍵をサーバ側へ送付
demouser@demo-server's password:
id_ecdsa.pub
100%  272   453.3KB/s   00:00

$ ssh demouser@demo-server
demouser@demo-server's password:
Last login: Sun Feb  6 19:57:24 2022 from gateway
demo-server$ mkdir ~/.ssh      ※格納ディレクトリを作成
demo-server$ chmod 700 ~/.ssh      ※格納ディレクトリを保護
demo-server$ ls
id_ecdsa.pub
demo-server$ cat id_ecdsa.pub >> ~/.ssh/authorized_keys      ※クライアント公開鍵を格納
demo-server$ chmod 600 ~/.ssh/authorized_keys      ※公開鍵を保護
demo-server$ ls -l ~/.ssh
合計 4
-rw-------. 1 demouser demouser 272  2月  6 20:03 authorized_keys
demo-server$ exit
ログアウト
Connection to demo-server closed.
```

公開鍵認証によるユーザ認証

SSHリモート接続（公開鍵認証：公開鍵認証によるログイン）

```
$ ssh demouser@demo-server
（別windowでパスフレーズ入力）
Last login: Sun Feb  6 20:03:13 2022 from gateway      ※パスワード入力なしでログイン

demo-server$ uname -n
demo-server

demo-server$ id
uid=1000(demouser) gid=1000(demouser) groups=1000(demouser) context=・・・

demo-server$ exit
ログアウト
Connection to demo-server closed.
```

ssh-agent

■鍵ペア生成時にパスフレーズを設定した場合、前項の実行例のように、SSH接続毎にパスフレーズを入力する必要があります。ssh-agentを利用して、秘密鍵とパスフレーズを登録しておくと、入力を省略することができます。

ssh-agent コマンド： ssh-agentを起動

```
ssh-agent [オプション] コマンド
```

ssh-add コマンド： ssh-agentに秘密鍵を追加

```
ssh-add [オプション] 秘密鍵ファイル
```

オプション	説　明
-l	登録済秘密鍵を表示

ssh-agent

ssh-agentの利用

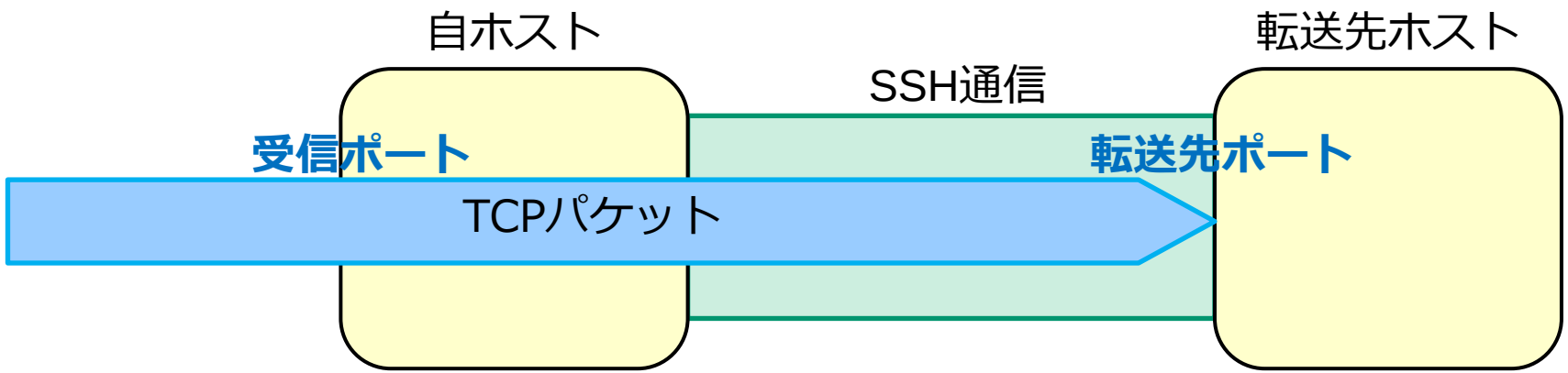
```
$ ssh-agent bash

$ ssh-add ~/.ssh/id_ecdsa
Enter passphrase for /home/user1/.ssh/id_ecdsa: (パスフレーズ入力)
Identity added: /home/user1/.ssh/id_ecdsa (/home/user1/.ssh/id_ecdsa)

$ ssh-add -l
521 SHA256:xIRaFnBBuyC8WoyUFRhwQhZes6zsZNsRZxxAlzsk3Bk /home/user1/.ssh/id_ecdsa (ECDSA)

$ ssh demouser@demo-server
Last login: Sun Feb  6 20:11:17 2022 from gateway      ※パスフレーズ入力なしでログイン

demo-server$ exit
ログアウト
Connection to demo-server closed.

$ exit

$
```

SSHポート転送

■SSHポート転送は、特定ポートで受信したTCPパケットを、SSHを使った安全な通信路を経由して、リモートホストへ転送する仕組みです。

ssh コマンドによるポート転送

```
ssh -L 受信ポート:転送先ホスト:転送先ポート [転送先ユーザ名@]転送先ホスト
```



SSHポート転送

SSHポート転送の利用

```
$ curl demo-server      ※Webサーバへ接続
curl: (7) Failed connect to demo-server:80; ホストへの経路がありません

$ sudo nmap demo-server
 . . .
PORT   STATE SERVICE
22/tcp open  ssh      ※Webサーバの80番ポートが開いていない
 . . .

$ ssh -f -N -L 10080:demo-server:80 demouser@demo-server
         ※SSHポート転送：自ホストの10080番ポートで受信したパケットををWebサーバの80番ポートへ

$ curl localhost:10080      ※自ホストの10080番ポートへ接続
Hello from web server       ※Webサーバの80番ポートへ転送された
```

1.はじめに

2.暗号の種類と特徴

3.SSHによる安全なリモート接続

4.GnuPGによるファイルの保護

GnuPGの概要

■GnuPG (GNU Privacy Guard) は、ファイルを暗号化したり、ファイルに電子署名を付与したりするユーティリティです。gpgコマンドを使います。

gpg コマンド

gpg［オプション］

オプション	説　明
--gen-key	鍵ペアを生成
--list-keys	キー情報を表示
--export メールアドレス	鍵をエクスポート
-o ファイル名	出力ファイルを指定
--import ファイル名	鍵をインポート
-a	インポート、エクスポートをASCII形式で行う（デフォルトはバイナリ）
--sign-key メールアドレス	鍵に署名する
-e	ファイルを暗号化
-r メールアドレス	鍵を指定
-b	署名ファイルを作成（分離署名）
--verify	署名ファイルを検証

鍵の生成と受渡し

■GnuPGを利用するには、まず鍵ペアを生成します。

GnuPG鍵ペアの生成

```
$ gpg --gen-key      ※鍵ペア生成
・・・
ご希望の鍵の種類を選択してください:
・・・
あなたの選択は? (ENTER)
RSA 鍵は 1024 から 4096 ビットの長さで可能です。
鍵長は? (2048) (ENTER)
・・・
鍵の有効期間は? (0) (ENTER)
(null)は無期限です
これで正しいですか? (y/N) y
・・・
本名: misawa
電子メール・アドレス: misawa@linuc.jp
コメント: (ENTER)
・・・
```

鍵の生成と受渡し

■鍵は ~/.gnupg/ 下のキーリングに格納されます。

GnuPG鍵ペアの生成

```
名前(N)、コメント(C)、電子メール(E)の変更、またはOK(O)か終了(Q)? O
秘密鍵を保護するためにパスフレーズがいります。
（別windowでパスフレーズ設定）
・・・
フィンガー・プリント = E57B 48CF 6235 AC91 A1BA  4DFB 6812 F1BF 4F35 793F
uid                  misawa <misawa@linuc.jp>
sub   2048R/32AB3571 2022-02-08

$ ls ~/.gnupg/
S.gpg-agent  gpg.conf  private-keys-v1.d  pubring.gpg  pubring.gpg~  random_seed
secring.gpg  trustdb.gpg

$ gpg --list-keys    ※キー情報を表示
/home/user1/.gnupg/pubring.gpg
------------------------------
pub   2048R/4F35793F 2022-02-08
uid                  misawa <misawa@linuc.jp>
sub   2048R/32AB3571 2022-02-08
```

鍵の生成と受渡し

■公開鍵をexportしてから、相手ユーザに渡します。

公開鍵のexportと引渡し

```
$ gpg -o user1.pubkey -a --export misawa@linuc.jp      ※公開鍵をexport

$ cp user1.pubkey /tmp      ※便宜上、/tmp 経由で受渡し

$ ls -l /tmp/user1.pubkey
-rw-rw-r--. 1 user1 user1 1707  2月   8 17:26 /tmp/user1.pubkey
```

鍵の生成と受渡し

■相手ユーザ側でも、まず鍵ペアを生成します。

公開鍵の受取りとimport

```
（別端末を開き、ユーザ切替え）
$ su - user9
パスワード:

user9$ cp /tmp/user1.pubkey .

user9$ gpg --gen-key
 . . .

user9$ gpg --list-keys
/home/user9/.gnupg/pubring.gpg
------------------------------
pub   2048R/7C850510 2022-02-08
uid                  kishida <fumio@linuc.jp>
sub   2048R/80039B0C 2022-02-08
```

鍵の生成と受渡し

■もらった公開鍵をimportします。

公開鍵の受取りとimport

```
user9$ gpg --import user1.pubkey      ※もらった公開鍵をimport
gpg: 鍵4F35793F: 公開鍵"misawa <misawa@linuc.jp>"をインポートしました
gpg:          処理数の合計: 1
gpg:            インポート: 1  (RSA: 1)

user9$ gpg --list-keys
/home/user9/.gnupg/pubring.gpg
------------------------------
pub   2048R/7C850510 2022-02-08
uid                  kishida <fumio@linuc.jp>
sub   2048R/80039B0C 2022-02-08

pub   2048R/4F35793F 2022-02-08
uid                  misawa <misawa@linuc.jp>
sub   2048R/32AB3571 2022-02-08
```

鍵の生成と受渡し

■importした相手の公開鍵に署名して、信頼できる公開鍵であることを記録します。

もらった公開鍵への署名

```
user9$ gpg --sign-key misawa@linuc.jp      ※もらった公開鍵へ署名
・・・
主鍵フィンガー・プリント: E57B 48CF 6235 AC91 A1BA  4DFB 6812 F1BF 4F35 793F

     misawa <misawa@linuc.jp>

本当にこの鍵にあなたの鍵"kishida <fumio@linuc.jp>"で署名してよいですか
(7C850510)

本当に署名しますか? (y/N) y

次のユーザの秘密鍵のロックを解除するには
パスフレーズがいります:"kishida <fumio@linuc.jp>"
2048ビットRSA鍵, ID 7C850510作成日付は2022-02-08

（別windowでパスフレーズ入力）
```

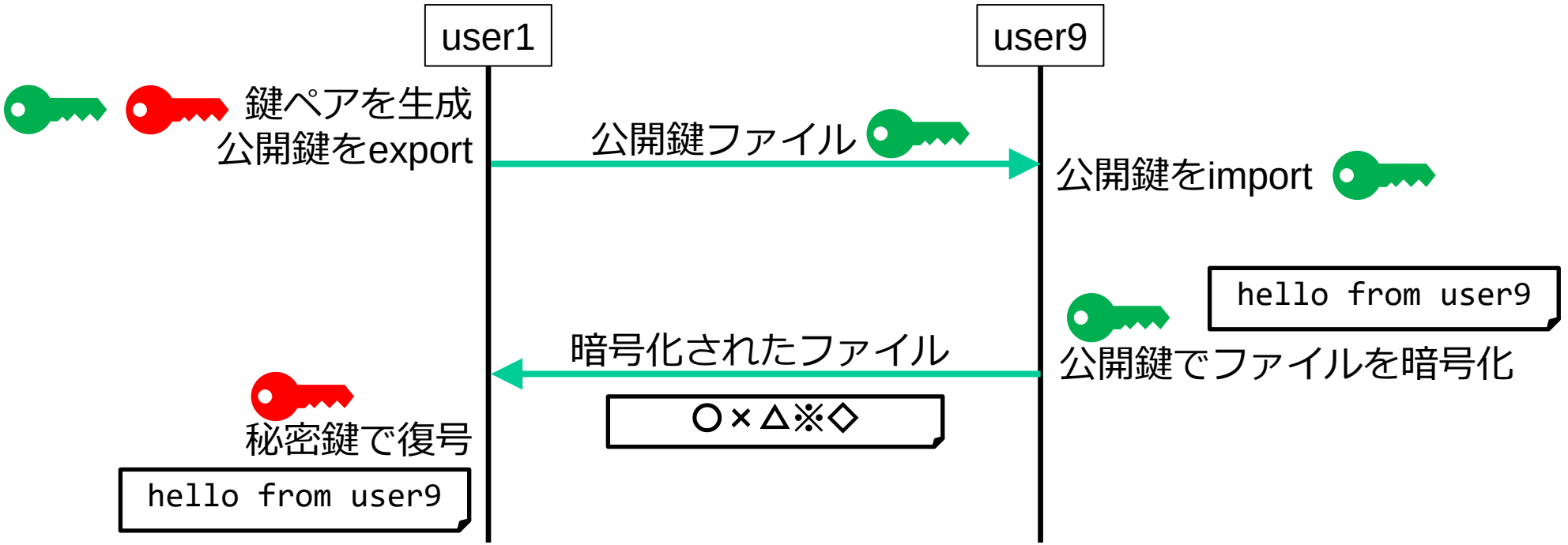
鍵の生成と受渡し

もらった公開鍵への署名

```
user9$ gpg --list-keys
gpg: 信用データベースの検査
gpg: 「ギリギリの信用」3、「全面的信用」1、PGP信用モデル
gpg: 深さ: 0  有効性:   1  署名:   1  信用: 0-, 0q, 0n, 0m, 0f, 1u
gpg: 深さ: 1  有効性:   1  署名:   0  信用: 1-, 0q, 0n, 0m, 0f, 0u
/home/user9/.gnupg/pubring.gpg
------------------------------
pub   2048R/7C850510 2022-02-08
uid                  kishida <fumio@linuc.jp>
sub   2048R/80039B0C 2022-02-08

pub   2048R/4F35793F 2022-02-08
uid                  misawa <misawa@linuc.jp>
sub   2048R/32AB3571 2022-02-08
```

ファイルの暗号化

■相手から受取った公開鍵を使ってファイルを暗号化し、ファイルを安全に送ります。



ファイルの暗号化

ファイルの暗号化

```
（user9で）
user9$ echo "hello from user9" > hello9.txt

user9$ cat hello9.txt
hello from user9

user9$ gpg -e -a -r misawa@linuc.jp hello9.txt      ※相手の公開鍵でファイルを暗号化

user9$ ls hello9*
hello9.txt  hello9.txt.asc

user9$ cat hello9.txt.asc
-----BEGIN PGP MESSAGE-----
Version: GnuPG v2.0.22 (GNU/Linux)

hQEMA8SXuiwyqzVxAQf/YxIiEP9ItW8bgs/bAS4atikRj0EKrK0L0kJ6nOIFWj7a
・・・
-----END PGP MESSAGE-----

user9$ cp hello9.txt.asc /tmp      ※便宜上、/tmp 経由で受渡し
```

ファイルの暗号化

■暗号化されたファイルを受取ったら、自身の秘密鍵で復号します。

ファイルの復号

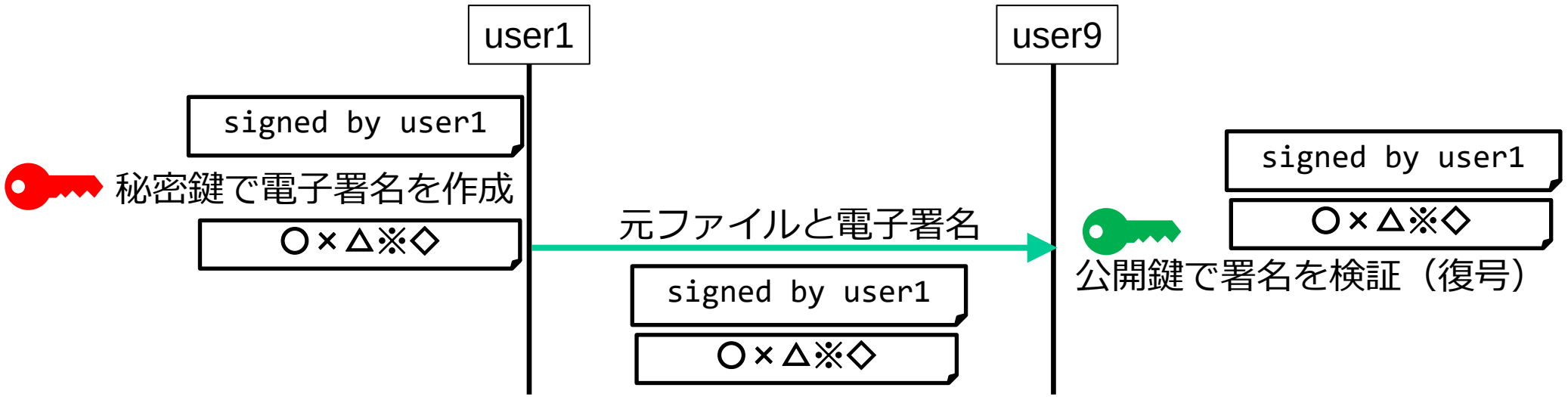
```
（user1で）
$ cp /tmp/hello9.txt.asc .

$ cat hello9.txt.asc
-----BEGIN PGP MESSAGE-----
. . .
-----END PGP MESSAGE-----

$ gpg hello9.txt.asc      ※自身の秘密鍵で復号
. . .
（別windowでパスフレーズ入力）
. . .

$ ls hello9*
hello9.txt  hello9.txt.asc

$ cat hello9.txt
hello from user9
```

gpg-agent

- gpg-agentは、GnuPGの秘密鍵を管理します。パスフレーズのキャッシュも行います。

電子署名

- 電子署名は、ファイルの送信者が本人であることを証明する仕組みです。
- ファイルを秘密鍵で暗号化することによって、ファイルの電子署名を作成します。
- 署名付ファイルの受信者は、送信者の公開鍵を使って検証（復号）します。正常に検証（復号）できれば、送信者は正しい秘密鍵を持っている本人であることの証明になります。



電子署名

電子署名の作成

```
（user1で）
$ echo "signed by user1" > sign1.txt
$ cat sign1.txt
signed by user1

$ gpg -b -a sign1.txt      ※電子署名作成
・・・
（別windowでパスフレーズ入力）

$ ls sign1*
sign1.txt  sign1.txt.asc
$ cat sign1.txt.asc
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v2.0.22 (GNU/Linux)

iQEcBAABAgAGBQJiAi7hAAoJEGgS8b9PNXk/jF8H/2hryCmeWs4WWInKHP2ZjQCs
・・・
-----END PGP SIGNATURE-----

$ cp sign1* /tmp      ※便宜上、/tmp 経由で受渡し
```

電子署名

電子署名の検証

```
（user9で）
user9$ cp /tmp/sign1* .

user9$ ls sign1*
sign1.txt  sign1.txt.asc

user9$ gpg --verify sign1.txt.asc sign1.txt      ※電子署名を検証
gpg: 2022年02月08日 17時50分41秒 JSTにRSA鍵ID 4F35793Fで施された署名
gpg: "misawa <misawa@linuc.jp>"からの正しい署名

user9$ cat sign1.txt
signed by user1
```

本セミナーのまとめ

1. CentOS 7 に基づく学習環境を自分で構築してみましょう。

2. 暗号の仕組みには、共通鍵暗号方式と公開鍵暗号方式とがあります。

3. SSHを利用してリモートホストへ安全にアクセスできます。
4. SSHで公開鍵認証によるユーザ認証を利用すると、パスワードなしでリモートホストへアクセスできます。

5. GnuPGを使って、ファイルの暗号化と電子署名を行えます。

ご清聴ありがとうございました