z/TPF secure file transfer

2023 TPF Users Group Conference
April 24-26, Dallas, TX
Operations and Coverage Subcommittee

—

Dan Gritter

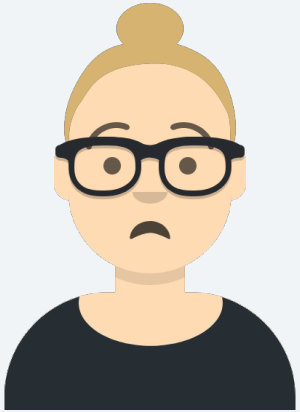
Disclaimer

Any reference to future plans are for planning purposes only. IBM reserves the right to change those plans at its discretion. Any reliance on such a disclosure is solely at your own risk. IBM makes no commitment to provide additional information in the future.

Problem statement

Security concerns continue to increase in all aspects of technology. With these concerns, customers need a way to secure all file traffic in and out of z/TPF. Current options of FTP and TFTP servers are not secure. Under the covers, loadtpf uses FTP. These options leave gaps in the area of security.

Pain points



Christine
Security Administrator

All my connections to z/TPF are not secure. There is a risk when files are transferred using FTP and TFTP. In addition, loadtpf uses FTP under the covers. I need to deploy a solution to fill these security gaps.

My test and production systems require different security considerations. I need a solution that will offer multiple configuration options for my systems.

The developers and testers in my organization are very busy. I also need a solution that will not disrupt their typical workflow.

As-is user story

Files can be transferred into the z/TPF system by using FTP or TFTP, but these options do not encrypt the data during the transfer. Because loadtpf uses FTP under the covers, loadsets are also not encrypted when transferred to z/TPF.

To-be user story

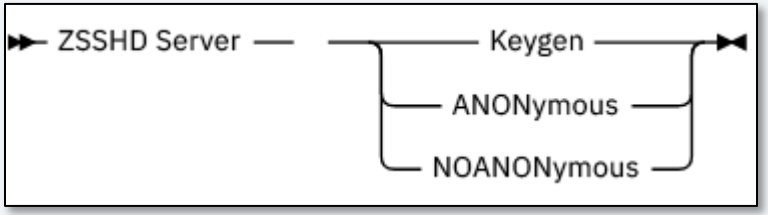
A stand-alone SSH server on z/TPF, defined as a JAM, provides the ability to securely transfer files between z/TPF and remote platforms. User authentication and authorization for the SSH server on z/TPF provide another layer of security to z/TPF connections.

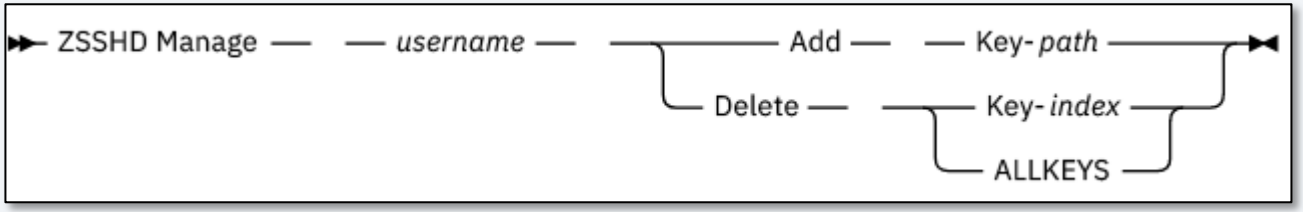
SSH server on z/TPF – overview

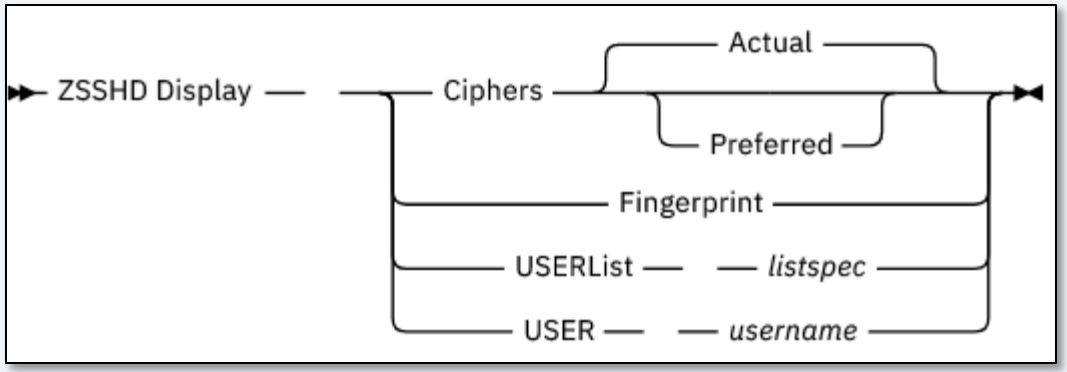
- Implemented as a stand-alone SSH server by using the SSHD subproject of Apache Mina (https://mina.apache.org/sshd-project)
- Configured as a JAM to utilize:
  - ZJAMC commands
  - JAM descriptor customization
  - Message logging
  - JVM monitoring with z/TPF system monitoring for Java applications
- Managed sockets with InetD by using a connection port defined during setup (**ZINET**)
- Managed functionality with new **ZSSHD** commands
- Authenticated user sessions established by exchanging keys
- Bootstrapped initial connection with anonymous login to transfer the first key

No client-side functionality is included with this project

ZSSHD commands - overview







SSH server startup process

Administrator

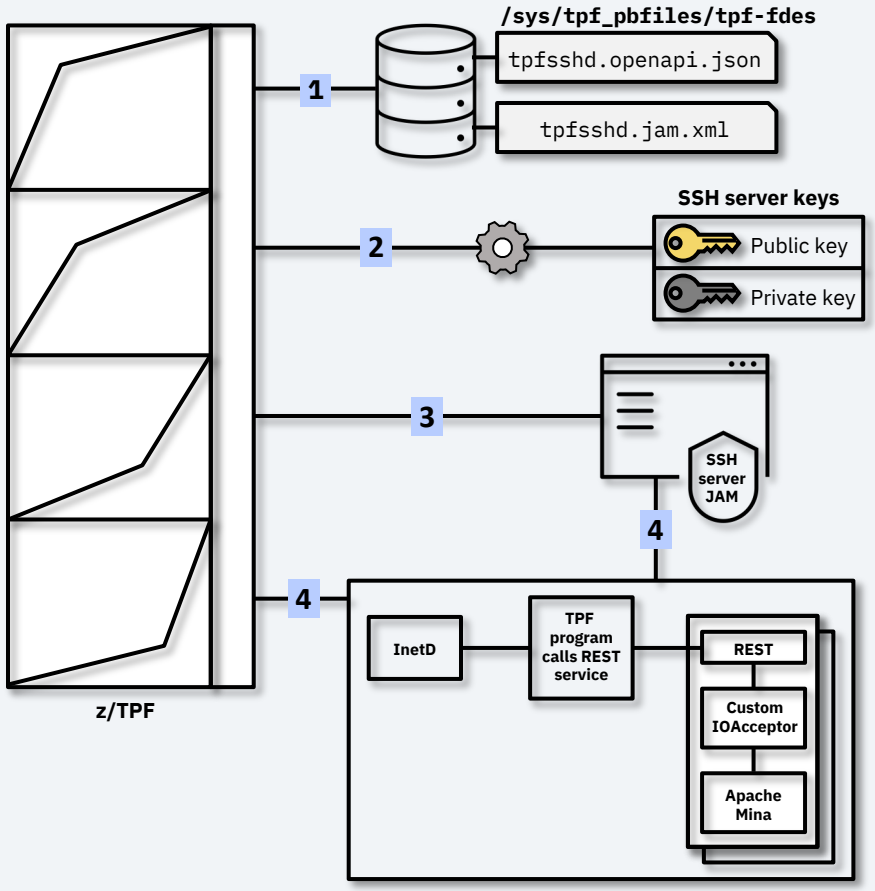
Assumption: SSH server JAM is configured.

1. Deploy the OpenAPI and JAM descriptors (*one-time startup step*):
   ```
   ZMDES DEPLOY FILE-tpfsshd.openapi.json
   ZMDES DEPLOY FILE-tpfsshd.jam.xml
   ```

2. Generate the server's public-private key pair (*initial startup step and periodic maintenance*):
   ```
   ZSSHD SERVER KEYGEN
   ```

3. Start the SSH server JAM (*initial startup step and stop, start, and recycle as needed*):
   ```
   ZJAMC START N-tpfsshd
   ```

4. Add and start the InetD configuration (*one-time startup step*):
   ```
   ZINET ADD S-TPFSSHD PGM-CJM6 MODEL-AOA2
   PORT-22 P-TCP
   ZINET START S-TPFSSHD
   ```



SSH server bootstrap process

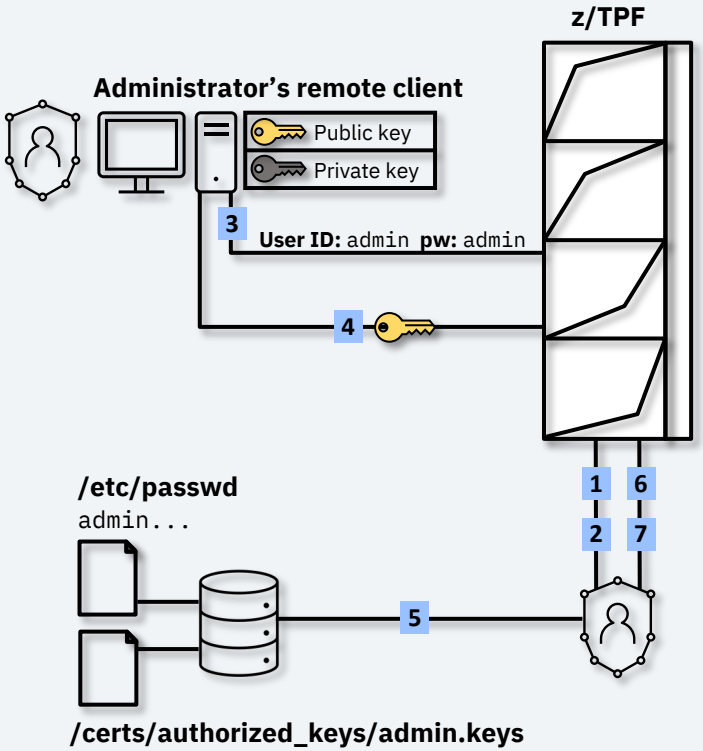
- Enables an anonymous login mode
- Intended for securely transferring the first public key to z/TPF
- First public key transferred to z/TPF should be the administrator's public key
- **ZSSHD SERVER (NO)ANONYMOUS** toggles anonymous login on and off, which controls the **anon** property in the properties file
- Changes to the **anon** property are activated when the SSH server JAM is started (if not previously started) or recycled (if already started)
- With anonymous login enabled, the login credentials for the SSH server are the remote client's user ID as the password – no keys are exchanged

SSH server bootstrap process - example

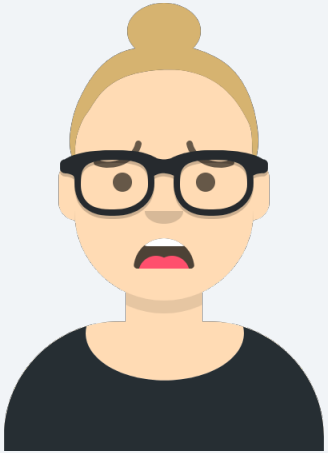
Administrator

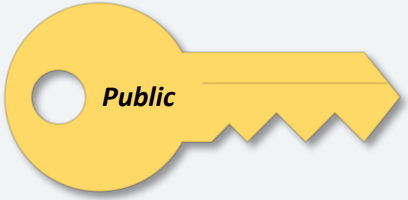
Assumption: SSH server JAM is configured and running.

1. From z/TPF, issue the **ZSSHD SERVER ANONYMOUS** command to switch on anonymous login.
2. From z/TPF, issue the **ZJAMC RECYCLE N-tpfsshd** command to recycle the SSH server JAM and enable anonymous logins.
3. From the remote client, connect to the SSH server using SFTP with the remote client user ID as the password.
4. From the remote client, ***securely*** transfer the administrator's public key (`admin.pub`) to a directory on z/TPF (/tmp).
5. From z/TPF, issue the **ZSSHD MANAGE admin ADD KEY-/tmp/admin.pub** command to add the administrator's public key.
6. From z/TPF, issue the **ZSSHD SERVER NOANONYMOUS** command to switch off anonymous login.
7. From z/TPF, issue **ZJAMC RECYCLE N-tpfsshd** to recycle the SSH server JAM and disable anonymous logins.

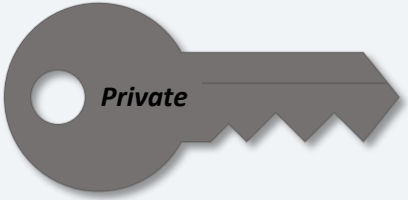


Key management





z/TPF secure file transfer seems like a great option for securing all file transfers to z/TPF, but key management could be a nightmare.



SSH server user keys - POSIX user ID prerequisite

- Any user key added to the SSH server must have an associated POSIX user ID and password defined on z/TPF
- User IDs are associated with group names
- z/TPF provides a set of default user IDs and group names
- Additional user IDs and passwords can be added to z/TPF with or without file system security support enabled
- User IDs are defined in the `/etc/passwd` file
- See Adding user and group names in IBM Documentation for more information

SSH key management – basics

Client side (user)

- Generates a public-private key pair from their remote client, using the **ssh-keygen** command, if the key pair does not already exist
- Shares their public key with the administrator using a customer-defined process

Server side (administrator)

- Generates the SSH server public-private key pair with the **ZSSHD SERVER KEYGEN** command
- Adds user public keys to z/TPF with the **ZSSHD MANAGE *username* ADD KEY-*path*** command to associate the user's public keys with the user's Posix user ID on z/TPF
- Deletes specific keys or all keys for a user with the **ZSSHD MANAGE *username* DELETE** command to prevent unauthorized connections
- Updates the SSH server public-private key pair with the **ZSSHD SERVER KEYGEN** command for periodic maintenance (SSH server JAM must be recycled to pick up the new key)

SSH key management – beyond the basics

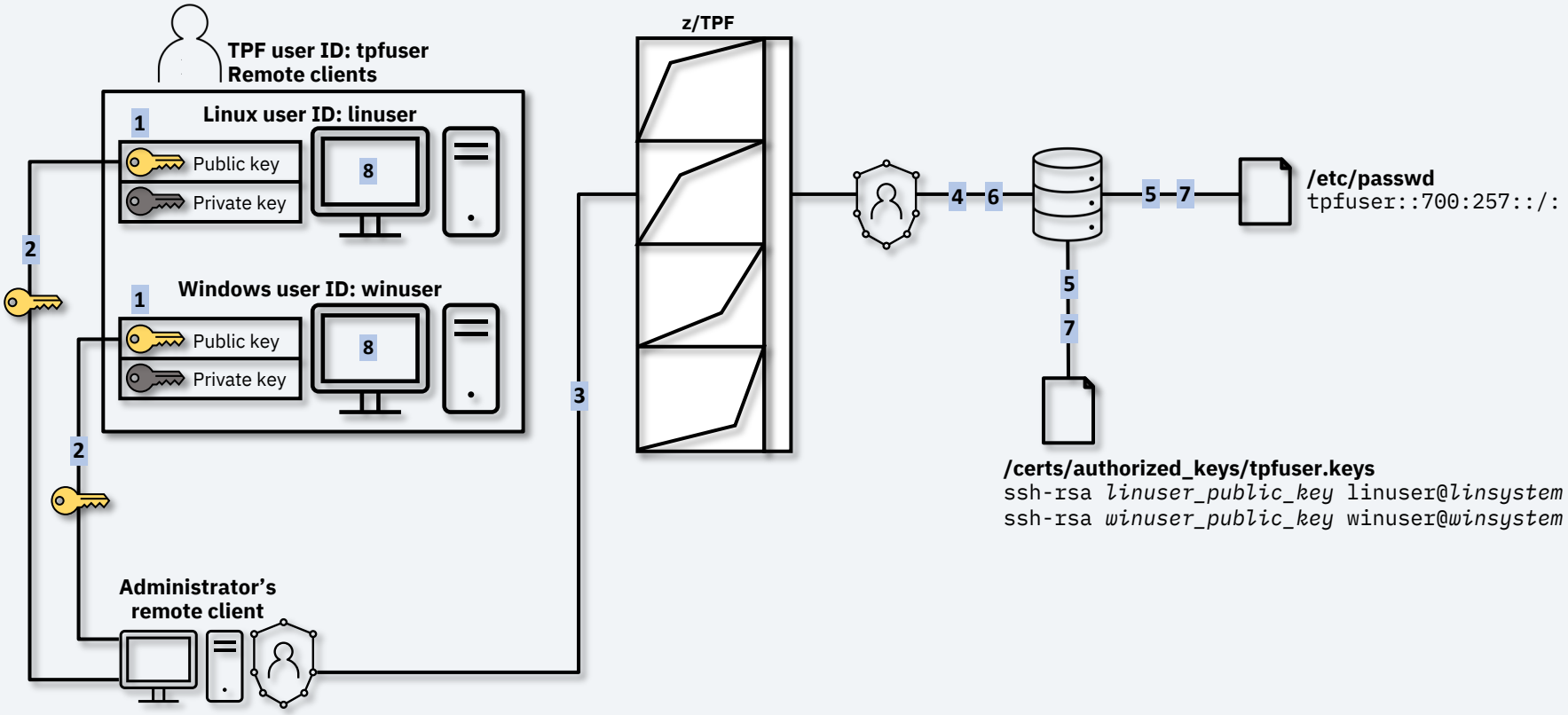
- Key management can be customized to suit your needs
- Test and production systems can have different key management strategies
- The following sample key management options will be presented:
  - User driven individual keys
  - Administrator driven individual keys
  - Administrator driven role-grouped keys
  - Persistent anonymous mode (only intended for test environments)

Sample key management – user driven individual keys

Assumption: SSH server JAM is configured and running, the bootstrap process is complete, and the user's POSIX ID is defined on z/TPF in `/etc/passwd`.

1. User: Generates a public-private key pair from each of the remote systems they'll use to connect to z/TPF (Linux and Windows in this example).
2. User: Shares their multiple public keys (`linuser.pub` and `winuser.pub`) with the administrator using a customer-defined process.
3. Administrator: ***Securely*** transfers the user's public keys to the `/tmp` directory on z/TPF using SFTP.
4. Administrator: Adds the user's public key from Linux to the SSH Server:
   `ZSSHD MANAGE tpfuser ADD KEY-/tmp/linuser.pub`
5. System: Validates tpfuser is defined in `/etc/passwd`, then adds the public key to a `/certs/authorized_keys/tpfuser.keys` file, which is created if it does not exist.
6. Administrator: Adds the user's public key from Windows to the SSH Server:
   `ZSSHD MANAGE tpfuser ADD KEY-/tmp/winuser.pub`
7. System: Validates tpfuser is defined in `/etc/passwd`, then adds the public key to the existing `/certs/authorized_keys/tpfuser.keys` file.
8. User: Can connect to the SSH server on z/TPF from Linux or Windows and transfer files securely.

Sample key management – user driven individual keys



Sample key management – administrator driven individual keys

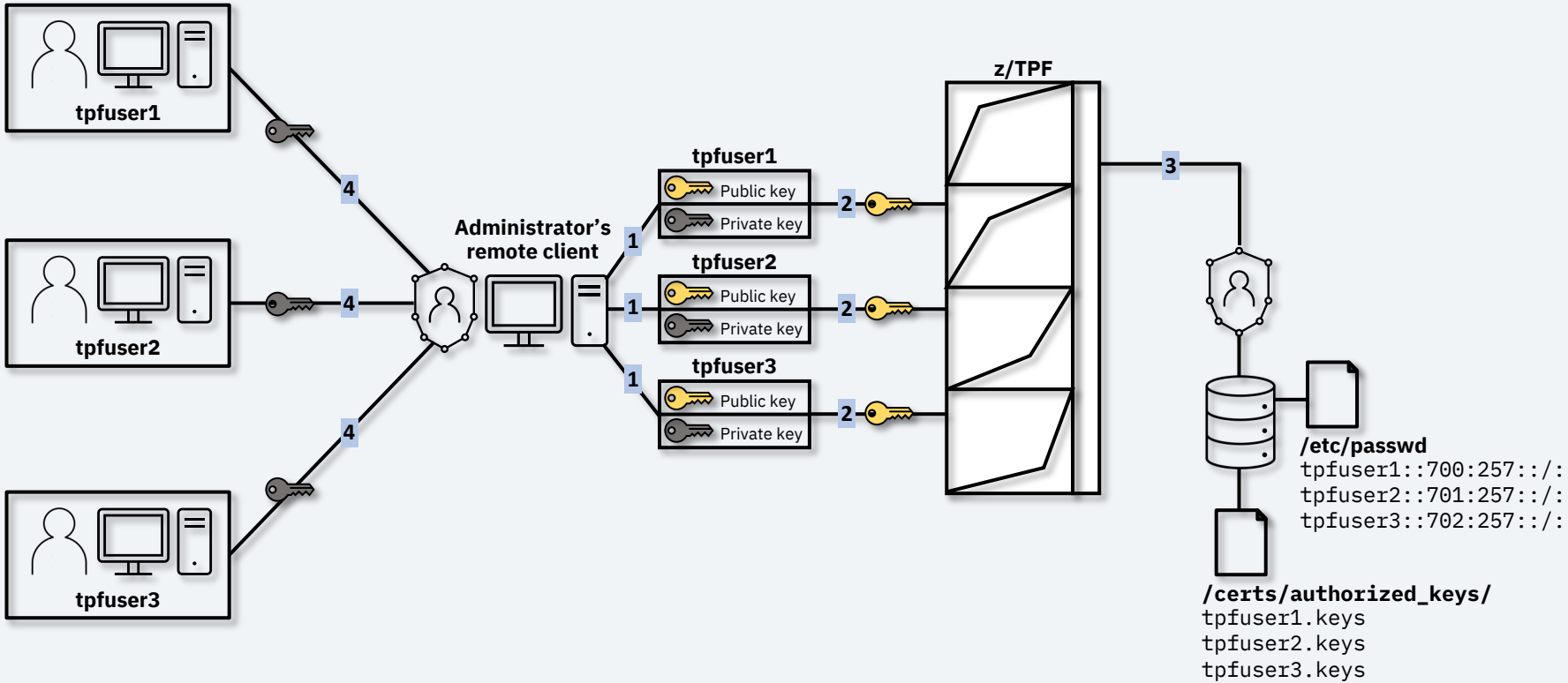
Administrator

Assumption: SSH server JAM is configured and running, the bootstrap process is complete, and the appropriate POSIX IDs are defined on z/TPF in `/etc/passwd`.

1. From a remote client, generate multiple public-private key pairs based on z/TPF POSIX user IDs, for example:
   ```
   ssh-keygen –f /home/admin/keys/tpfuser1
   ssh-keygen –f /home/admin/keys/tpfuser2
   ssh-keygen –f /home/admin/keys/tpfuser3
   ```
2. From that same remote client, use SFTP to ***securely*** transfer the public keys (tpfuser*.pub) to a directory on z/TPF, for example `/tmp`.
3. From z/TPF, add the public keys to the SSH server, for example:
   ```
   ZSSHD MANAGE tpfuser1 ADD KEY-/tmp/tpfuser1.pub
   ZSSHD MANAGE tpfuser2 ADD KEY-/tmp/tpfuser2.pub
   ZSSHD MANAGE tpfuser3 ADD KEY-/tmp/tpfuser3.pub
   ```
4. With a customer-defined process, ***securely*** share the appropriate private key with the matching user.

Hint: Having the same user ID on z/TPF and the remote client will simplify the login process for users.

Sample key management – administrator driven individual keys



Sample key management – administrator driven role-grouped keys

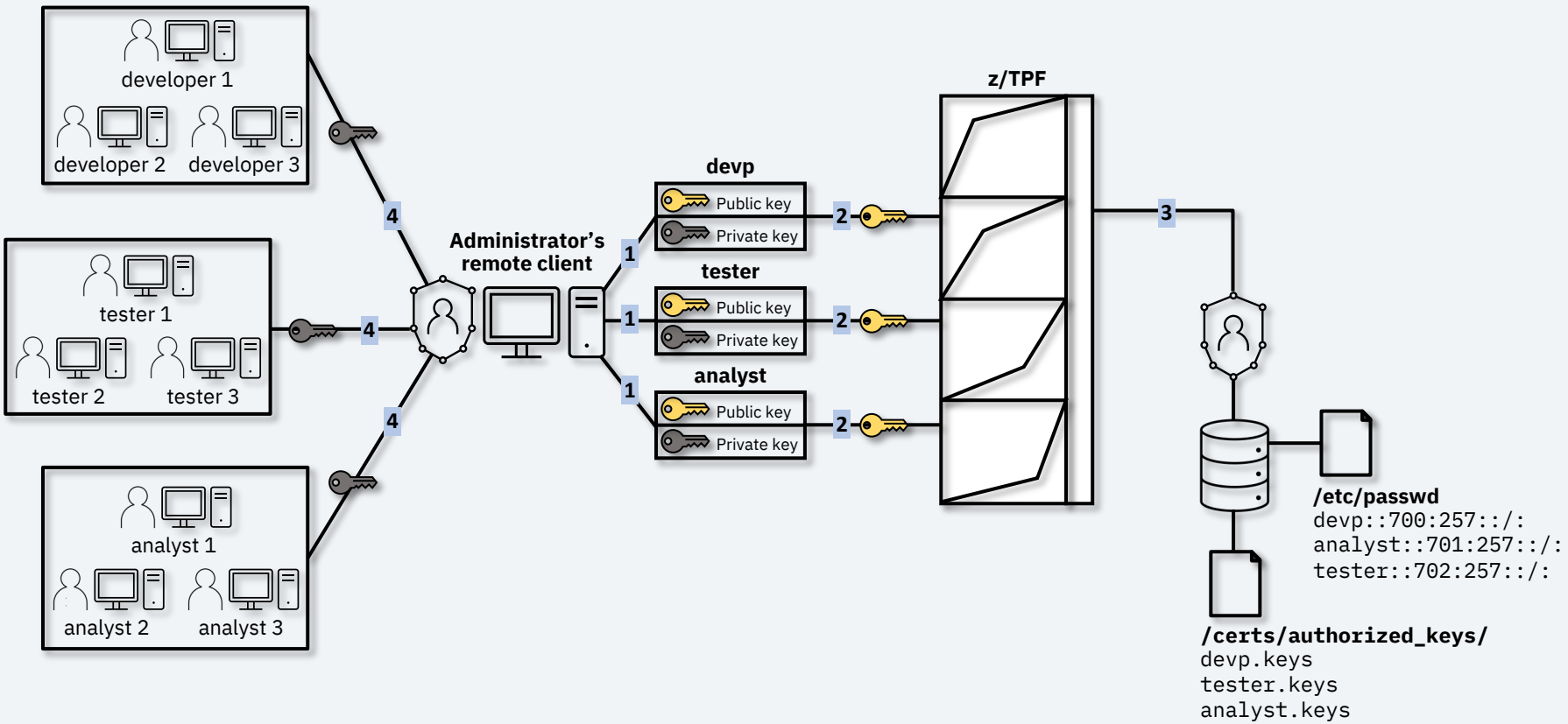
Administrator

Assumption: SSH server JAM is configured and running, the bootstrap process is complete, and the **devp**, **tester**, and **analyst** POSIX IDs are defined on z/TPF in `/etc/passwd`.

1. From a remote client, generate multiple public-private key pairs based on *role*, for example:
   ```
   ssh-keygen -f /home/admin/keys/devp
   ssh-keygen -f /home/admin/keys/tester
   ssh-keygen -f /home/admin/keys/analyst
   ```
2. From that same remote client, use SFTP to transfer the public keys (*role**.pub) to a directory on z/TPF, for example `/tmp`.
3. From z/TPF, add the public keys to the SSH server, for example:
   ```
   ZSSHD MANAGE devp ADD key-/tmp/devp.pub
   ZSSHD MANAGE tester ADD key-/tmp/tester.pub
   ZSSHD MANAGE analyst ADD key-/tmp/analyst.pub
   ```
4. With a customer-defined process, ***securely*** share the appropriate private key with the authorized users by role, for example, everyone on the development team gets the **devp** private key.

Note: Revoking access to a single user in a group would require repeating this process to ensure that only authorized users can continue connecting to the SSH server.

Sample key management – administrator driven role-grouped keys



Sample key management – persistent anonymous mode

Test environments only

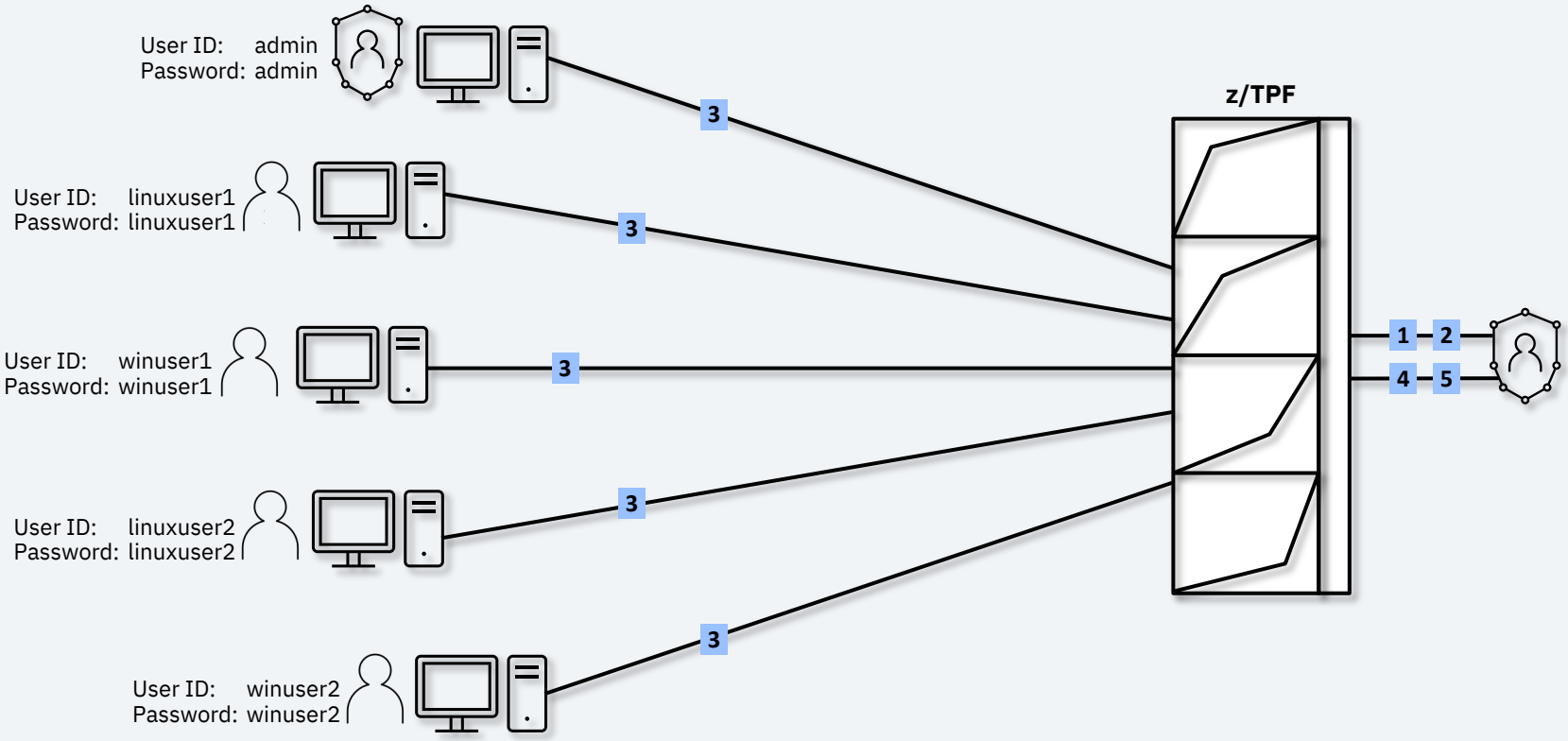
Administrator

Assumption: SSH server JAM is configured and running.

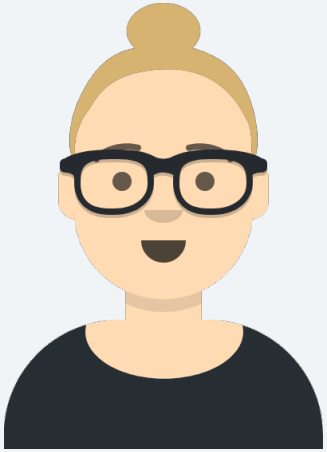
1. From z/TPF, issue the **ZSSHD SERVER ANONYMOUS** command to switch on anonymous login.
2. From z/TPF, issue the **ZJAMC RECYCLE N-tpfsshd** command to recycle the SSH server JAM and enable anonymous logins.
3. Any client can connect to the SSH server using their remote client user ID as their password. ***No keys are exchanged in this configuration.***
4. *Optional:* To stop using anonymous login on the system, from z/TPF, issue the **ZSSHD SERVER NOANONYMOUS** command to switch off anonymous login.
5. *Optional:* From z/TPF, issue **ZJAMC RECYCLE N-tpfsshd** to recycle the SSH server JAM and disable anonymous logins.

Sample key management – persistent anonymous mode

Test environments only



SSH server – details and behavior



Maybe key management is not that difficult. Are there other aspects of the SSH server that I should understand?

SSH server properties file

- The default SSH server input properties are defined in the `/base/tpfsshd/sshd.properties` file, which is loaded to the `/sys/tpf_pbfiles/apps/tpfsshd` directory on z/TPF
- The input properties are validated during the SSH server startup process to create a server version of the properties
- The default input properties, such as **ciphers** and **anon** mode, can be overridden to suit your needs
- To change the default input properties:
  - Modify the input properties file as needed
  - Load the input properties file to the `/sys/tpf_pbfiles/apps/tpfsshd` directory in binary format
  - Stop and restart the SSH server JAM (ZJAMC STOP and ZJAMC START) or recycle the JAM (ZJAMC RECYCLE) for the new properties to take effect
  - The customized properties are validated again during that JAM startup process, and the validated properties will override the previously validated server properties file

Cipher configuration for file transfers

- Apache Mina supports multiple ciphers for file transfers
- Ciphers on the server and the client are ordered by priority from most preferred to least preferred
- The first cipher from the client's cipher list that is available on the server determines the cipher in both directions for file transfers
- Client to server connections fail if a cipher cannot be negotiated
- Default server cipher priority:

```
aes256-cbc
aes192-cbc
aes128-cbc
aes256-gcm@openssh.com
aes128-gcm@openssh.com
aes256-ctr
aes192-ctr
aes128-ctr
chacha20-poly1305@openssh.com
```

- Server cipher priorities are customizable with the `/sys/tpf_pbfiles/apps/tpfsshd/sshd.properties` file

Cipher configuration for file transfers - details

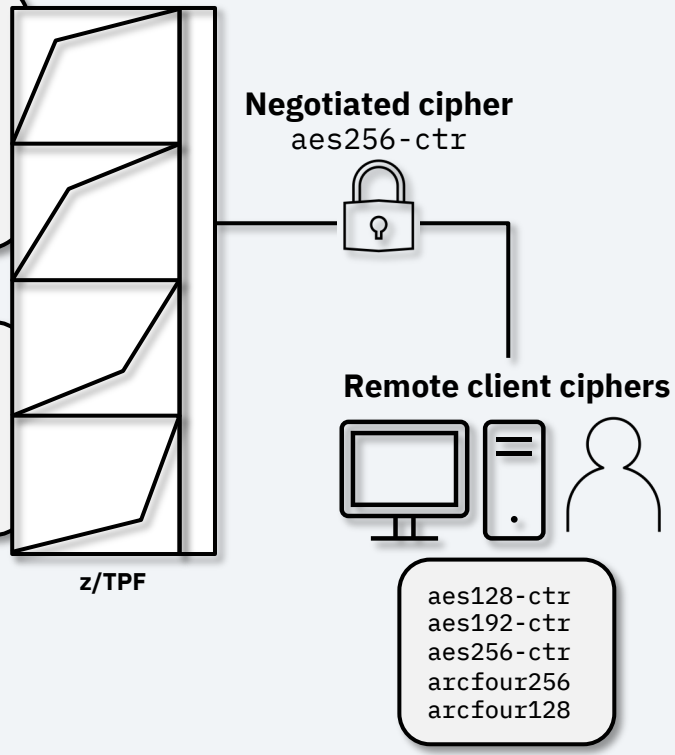
SSH server ciphers requested: ZSSHD DISPLAY CIPHERS PREFERRED

```
SSHD0003I 09.06.03 THE CIPHERS SPECIFIED FOR SECURE FILE TRANSFER DISPLAY
  aes256-ctr
  aes256-cbc
  invalid-cipher
  aes192-cbc
  3des
  aes128-cbc
END OF DISPLAY+
```

invalid cipher value
unsupported cipher value

SSH server ciphers set: ZSSHD DISPLAY CIPHERS

```
SSHD0002I 09.06.11 THE CIPHERS SET FOR SECURE FILE TRANSFER DISPLAY
  aes256-ctr
  aes256-cbc
  aes192-cbc
  aes128-cbc
END OF DISPLAY+
```



During SSH server startup, the JAM log file will record issues with ciphers:

```
WARNING: Ignoring Unsupported cipher(s) ([invalid-cipher, 3des]) in
aes256-ctr,aes256-cbc,invalid-cipher,aes192-cbc,3des,aes128-cbc
Mar 15, 2023 2:36:30 PM com.ibm.tpf.sshd.SSHDServer setCiphers
INFO: Ciphers set for the server in priority order:
[aes256-ctr, aes256-cbc, aes192-cbc, aes128-cbc]
```

SSH server fingerprint verification

- The SSH server fingerprint is established when the server's key pair is generated
- Initial remote connections to the SSH server are presented with the server's fingerprint
- The **ZSSHD DISPLAY FINGERPRINT** command on z/TPF can verify the server's fingerprint
- Mismatched fingerprints are a sign to stop the client/server connection

SSH server fingerprint verification - details

Linux client initial connection to the z/TPF SSH server

```
/home/someuser> sftp ip_address
The authenticity of host 'ip_address (ip_address)' can't be established.
RSA key fingerprint is SHA256:XJNxkm/dAfMV6FX3NbR9qTETKh4HkplH7QwxWpCM/aU.
Are you sure you want to continue connecting (yes/no)?
```

Windows client initial connection to the z/TPF SSH server

```
C:\>sftp ip_address
The authenticity of host 'ip_address (ip_address)' can't be established.
RSA key fingerprint is SHA256:XJNxkm/dAfMV6FX3NbR9qTETKh4HkplH7QwxWpCM/aU.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

z/TPF ZSSHD DISPLAY FINGERPRINT command

```
==> ZSSHD DISPLAY FINGERPRINT
SSHD0020I 15.59.51 THE SSH SERVER FINGERPRINT DISPLAY
SHA256:XJNxkm/dAfMV6FX3NbR9qTETKh4HkplH7QwxWpCM/aU
END OF DISPLAY+
```

z/TPF secure file transfer requirements

- System must be configured for Java with a minimum of these refresh APARs installed:
  - PJ46945 for IBM SDK, Java Technology Edition, Version 8
  - PJ46984 for IBM Semeru Runtime Certified Edition for z/TPF
- File system permissions must be in place
- RSA 2048-bit keys must be used for authentication

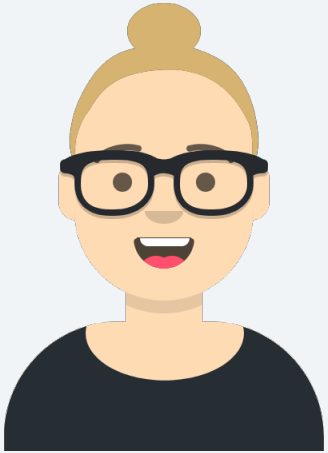
File transfer options

- SFTP for transferring files in binary format
- loadtpf for transferring loadsets
- TPF Toolkit for securely transferring loadsets

Conclusion

z/TPF secure file transfer:

- Secures file transfers with SFTP and loadtpf
- Manages users and keys with the new **ZSSHD** commands
- Allows for periodic rotation of server and user keys
- Adds to the growing library of Java applications on z/TPF



Wow, z/TPF secure file transfer sounds like a great solution! I'd like to start planning my implementation.

Target delivery – end of 2Q2023

Thank you to our sponsor users!

Our development cycle is driven by your feedback.

We began engaging with sponsor users for this project in August 2022. We have incorporated changes into our design based on sponsor user feedback.

Thank you for helping us improve this project!