

Wprowadzenie do cyberbezpieczeństwa

Temat 1:

Uwierzytelnianie klienta SSH
za pomocą kluczy prywatnych

Jakub Stachowicz, 198302

Jan Wiśniewski, 197662

26 maja 2025

Spis rzeczy

1 Omówienie SSH i konfiguracja serwera	3
1.1 Czym jest SSH?	3
1.2 Algorytmy kryptograficzne w SSH	3
1.3 Konfiguracja z wymuszonym użyciem kluczy	3
2 Klucze – generowanie i instalacja	4
2.1 Generowanie kluczy	4
2.1.1 W systemie Linux – ssh-keygen	4
2.1.2 W systemie Windows – PuTTYgen	4
2.2 Instalacja kluczy	5
2.2.1 Klient i serwer SSH Linux	5
2.2.2 Klient i serwer SSH Windows (OpenSSH)	5
2.2.3 Klient SSH PuTTY	5
3 Zastosowania SSH	6
3.1 Zdalny terminal	6
3.2 Przesył plików	6
3.2.1 SFTP (Secure FTP)	6
3.2.2 SCP (Secure Copy)	6
3.2.3 WinSCP	6
4 Bibliografia	7

1 Omówienie SSH i konfiguracja serwera

1.1 Czym jest SSH?

SSH (Secure Shell) to protokół sieciowy, który służy do zdalnego logowania się do innego komputera (serwera) w sposób bezpieczny. Umożliwia zarządzanie systemem operacyjnym, przesyłanie plików oraz wykonywanie poleceń na odległość – wszystko to przy użyciu szyfrowanego połączenia[1, The SSH protocol].

Uwierzytelnianie w SSH może odbywać się na dwa sposoby: za pomocą hasła lub kluczy kryptograficznych. Logowanie hasłem jest proste, ale mniej bezpieczne, ponieważ narażone jest na ataki typu brute-force. Znacznie bezpieczniejszą i częściej stosowaną metodą jest logowanie przy użyciu kluczy SSH, które opiera się na parze klucz prywatny-publiczny. Klucz publiczny umieszczany jest na serwerze, a prywatny pozostaje na komputerze użytkownika, dzięki czemu możliwe jest uwierzytelnienie bez podawania hasła. Metoda ta jest szczególnie przydatna w automatyzacji i pracy z wieloma serwerami[1, Automate with SSH keys, but manage them].

1.2 Algorytmy kryptograficzne w SSH

Temat algorytmów kryptograficznych w protokole SSH dotyczy zarówno szyfrowania samego ruchu sieciowego między klientem a serwerem, jak i logowania i autoryzacji. Niniejsze opracowanie skupia się na algorytmach stosowanych w autoryzacji za pomocą kluczy prywatnych.

W protokole SSH można używać kilku algorytmów generujących parę kluczy, w standardzie zdefiniowano RSA, DSS (DSA) oraz umożliwiono definiowanie innych kluczy[6, 6.6. s. 13]. Najpopularniejsze to RSA, ECDSA i Ed25519 (a dawniej używano też DSA, które dziś jest uważane za przestarzałe). OpenSSH wspiera następujące typy kluczy: DSA, RSA, ECDSA oraz Ed25519[2, User key generation]. Obecnie zaleca się wybór silniejszych algorytmów, np. RSA lub Ed25519 – OpenSSH 7.0 i nowsze domyślnie wyłączają słaby DSA, a od wersji 10.0 wsparcie dla tego algorytmu zostało usunięte w całości[3].

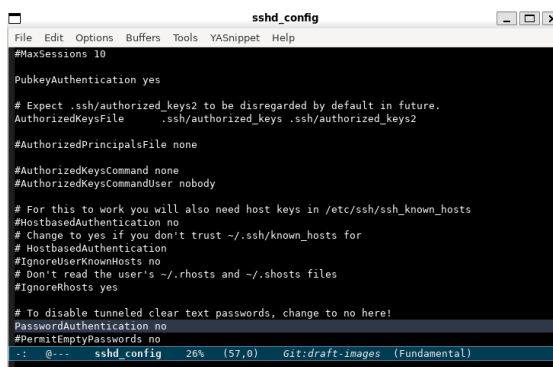
Dalsza część artykułu skupia się na użyciu kluczy RSA, jednak pozostałych algorytmów można użyć w bardzo podobny sposób. Najczęściej – poprzez zmianę parametru `rsa` na np.

`ed25519`.

1.3 Konfiguracja z wymuszonym użyciem kluczy

Aby serwer SSH (usługa `sshd`) zezwalał tylko na logowanie kluczem i wyłączał logowanie hasłem, należy zmodyfikować plik konfiguracyjny znajdujący się najczęściej pod ścieżką `/etc/ssh/sshd_config`[7]. Ważne dyrektywy to m.in.:

1. `PubkeyAuthentication yes` – włącza uwierzytelnianie kluczem publicznym (domyślnie zazwyczaj jest włączone).
2. `PasswordAuthentication no` – wyłącza logowanie hasłem. Po jego ustawieniu serwer nie akceptuje hasła SSH.
3. `AuthorizedKeysFile` – określa lokalizację pliku z kluczami publicznymi (domyślnie dla parametru `%h/.ssh/authorized_keys` będzie to `~/.ssh/authorized_keys`).



```
sshd_config
File Edit Options Buffers Tools YASnippet Help
#MaxSessions 10

PubkeyAuthentication yes

# Expect .ssh/authorized_keys2 to be disregarded by default in future.
AuthorizedKeysFile .ssh/authorized_keys .ssh/authorized_keys2

#AuthorizedPrincipalsFile none

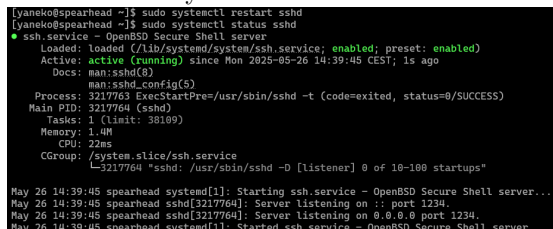
#AuthorizedKeysCommand none
#AuthorizedKeysCommandUser nobody

# For this to work you will also need host keys in /etc/ssh/ssh_known_hosts
#HostbasedAuthentication no
# Change to yes if you don't trust ~/.ssh/known_hosts for
# HostbasedAuthentication
#IgnoreUserKnownHosts no
# Don't read the user's ~/.rhosts and ~/.shosts files
#IgnoreRhosts yes

# To disable tunneled clear text passwords, change to no here!
PasswordAuthentication no
#PermitEmptyPasswords no

.: @--- sshd_config 26% (57,0) Git: draft-images (Fundamental)
```

Po edycji pliku `sshd_config` należy zapisać zmiany i ponownie uruchomić serwis SSH, np.: `sudo systemctl restart sshd` lub `sudo service sshd reload`. Spowoduje to zastosowanie nowych ustawień.



```
[yaneke@spearhead ~]$ sudo systemctl restart sshd
[yaneke@spearhead ~]$ sudo systemctl status sshd
● ssh.service - OpenBSD Secure Shell server
   Loaded: loaded (/lib/systemd/system/ssh.service; enabled; preset: enabled)
   Active: active (running) since Mon 2025-05-26 14:39:45 CEST; 1s ago
     Docs: man:sshd(8)
           man:sshd_config(5)
   Process: 3217763 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUCCESS)
    Main PID: 3217764 (sshd)
       Tasks: 1 (limit: 38109)
     Memory: 1.4M
        CPU: 22ms
   CGroup: /system.slice/ssh.service
           └─3217764 "sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups"

May 26 14:39:45 spearhead systemd[1]: Starting ssh.service - OpenBSD Secure Shell server...
May 26 14:39:45 spearhead sshd[3217764]: Server listening on :: port 1234.
May 26 14:39:45 spearhead sshd[3217764]: Server listening on 0.0.0.0 port 1234.
May 26 14:39:45 spearhead systemd[1]: Started ssh.service - OpenBSD Secure Shell server.
```

2 Klucze – generowanie i instalacja

2.1 Generowanie kluczy

Parę kluczy (klucz prywatny i publiczny) generuje użytkownik. Klucz prywatny przechowywany jest po stronie klienta, natomiast klucz publiczny – po stronie serwera.

2.1.1 W systemie Linux – ssh-keygen

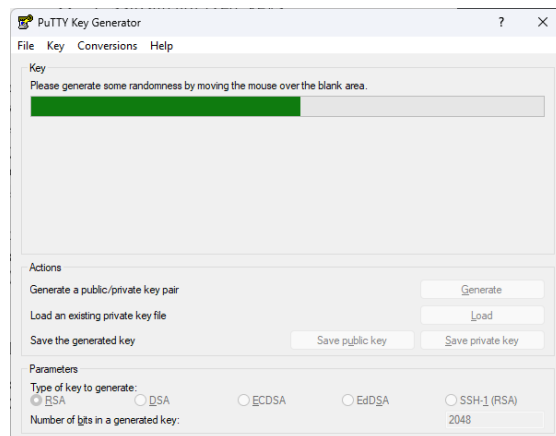
W Linuksie (oraz systemach UNIX/macOS) zwykle używa się polecenia `ssh-keygen`. Przykładowo: `ssh-keygen -t rsa -b 4096`. Polecenie `-t rsa -b 4096` tworzy klucz RSA o długości 4096 bitów[8].

Po uruchomieniu program zapyta o ścieżkę do pliku (domyślnie `~/.ssh/id_rsa` dla RSA lub `~/.ssh/id_ed25519` dla Ed25519) i opcjonalne hasło (passphrase). Gdy zakończy generowanie, w katalogu `~/.ssh/` powstaną dwa pliki: `id_rsa` (klucz prywatny) i `id_rsa.pub` (klucz publiczny).

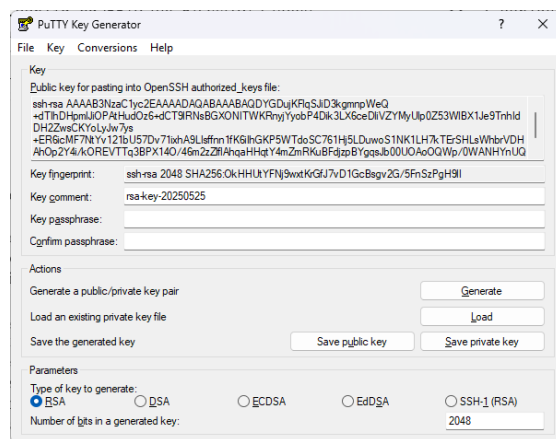
```
[yan3k@WinPC ~]$ ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/home/yan3k/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/yan3k/.ssh/id_rsa
Your public key has been saved in /home/yan3k/.ssh/id_rsa.pub
The key's fingerprint is:
SHA256:HzrPgLh20QhuIht35YtX7fD4B330vkDHBdXkRtqW+Q4 yan3k@WinPC
The key's randomart image is:
+--[RSA 4096]--+
  .+..+
  o.o..+.
  ..o.o..+.
  ..oS=.E+
  .ooo=+.+.
  .o.o+.+.
  ..+.o
  .o.o.o
  +--[SHA256]--+
```

2.1.2 W systemie Windows – PuTTYgen

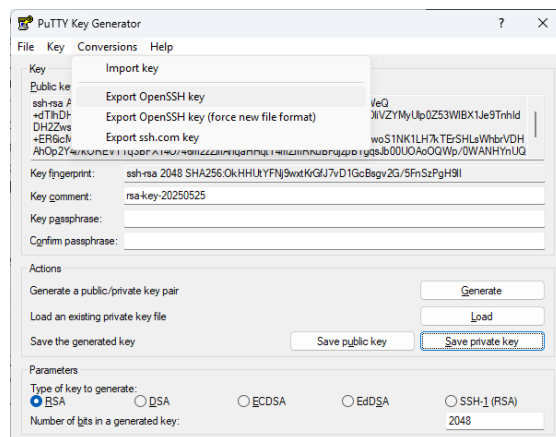
W systemie Windows często korzysta się z PuTTYgen (narzędzie wchodzące w skład pakietu PuTTY). Uruchamiamy PuTTYgen, wybieramy typ klucza (np. RSA lub Ed25519) i długość (np. 2048 bitów), a następnie klikamy przycisk Generate. PuTTYgen poprosi nas o losowe ruchy myszką w obrębie okna – to sposób na zebranie entropii do generacji klucza. Gdy pasek postępu dojdzie do końca, w oknie PuTTYgen pojawi się wygenerowany klucz publiczny (tekst zaczynający się np. `ssh-rsa AAAA...`).



Następnie należy wpisać (dwukrotnie) passphrase chroniący klucz prywatny (zalecane) i zapisać pliki: kliknąć Save public key (np. `mykey.pub`) oraz Save private key (plik `.ppk` dla PuTTY, np. `mykey.ppk`).



W PuTTYgenie można też przekonwertować prywatny klucz do formatu OpenSSH (menu Conversions → Export OpenSSH key) – przydaje się to, gdy chcemy używać tego samego klucza w programach innych niż PuTTY. Plik `.ppk` pozostaje natywnym formatem PuTTY.



2.2 Instalacja kluczy

Po wygenerowaniu kluczy należy je zainstalować – zarówno po stronie klienta, jak i serwera.

2.2.1 Klient i serwer SSH Linux

Aby klient logujący się z Linuksa mógł użyć swojego klucza, klucz publiczny klienta należy dodać do pliku `~/.ssh/authorized_keys` konta docelowego na serwerze. Najprościej to zrobić poleceniem `ssh-copy-id` – narzędzie automatycznie kopiuje nasz klucz publiczny do odpowiedniego pliku na serwerze. Przykład: `ssh-copy-id user@serwer`.



Powoduje to dodanie zawartości klucza `~/.ssh/id_rsa.pub` (lub innego domyślnego klucza) do `~/.ssh/authorized_keys` na serwerze. Jeśli narzędzie nie jest dostępne, można wykonać ręcznie: najpierw zalogować się hasłem, a potem na serwerze stworzyć katalog `.ssh` i dopisać klucz, np.:

```
mkdir -p ~/.ssh
echo "$(cat ~/.ssh/id_rsa.pub)"
    >> ~/.ssh/authorized_keys
chmod 700 ~/.ssh
chmod 600 ~/.ssh/authorized_keys
```



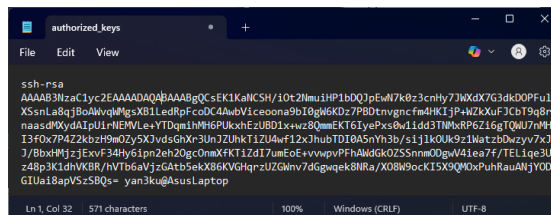
Następnie należy zadbać o instalację klucza prywatnego po stronie klienta. W tym celu po wygenerowaniu pary kluczy, plik z kluczem prywatnym (`id_rsa`) powinien zostać skopiowany lub przeniesiony tylko do katalogu domowego użytkownika klienta, w podkatalogu `.ssh`:

```
mv /path/to/id_rsa ~/.ssh/id_rsa.
```

Po dodaniu klucza można przetestować logowanie: `ssh user@serwer` już nie powinno prosić o hasło (chyba że nadaliśmy passphrase do klucza).

2.2.2 Klient i serwer SSH Windows (OpenSSH)

Windows (np. Windows 10) posiada wbudowany serwer OpenSSH (lub Win32-OpenSSH). Konfiguracja kluczy jest analogiczna: klucz publiczny wrzucamy do pliku `authorized_keys`. Dla konta użytkownika domyślnie jest to ścieżka: `C:\Users\\.ssh\authorized_keys`.



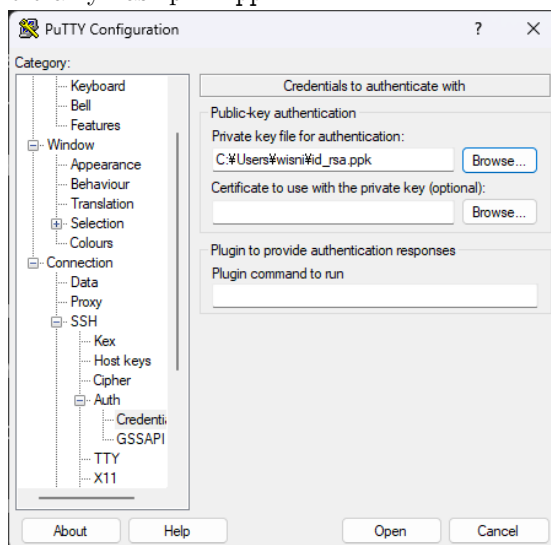
Dla kont administratora przewidziano specjalny plik w katalogu `C:\ProgramData\ssh\administrators_authorized_keys`. Zawartość tych plików można wprowadzić ręcznie np. przez `scp` lub nawet komendami PowerShell (przykład w dokumentacji Microsoft pokazuje użycie `ssh` z parametrem, który na zdalnym serwerze tworzy katalog `.ssh` i dopisuje klucz do `authorized_keys`). Ważne jest, aby katalog `.ssh` miał ograniczone prawa (tylko właściciel czy administrator), inaczej OpenSSH może odrzucić klucze.

W tym przypadku plik z kluczem prywatnym powinien trafić analogicznie jak na Linuxie do folderu `.ssh` w foledrze domowym użytkownika: `C:\Users\\.ssh\id_rsa`.

Po umieszczeniu klucza publicznego, logowanie SSH przebiega z użyciem klucza, bez hasła.

2.2.3 Klient SSH PuTTY

W przypadku PuTTY klucz prywatny musi być w formacie PuTTY (.ppk). Jeżeli mamy klucz w formacie OpenSSH (np. wygenerowany wcześniej ssh-keygenem), wystarczy go załadować w PuTTYgen (Load private key) i zapisać jako .ppk (Save private key). Następnie, aby PuTTY użył klucza, w oknie konfiguracji sesji PuTTY przechodzimy do Connection → SSH → Auth i w polu „Private key file for authentication” wybieramy nasz plik .ppk.



W razie potrzeby możemy też użyć Pagean-

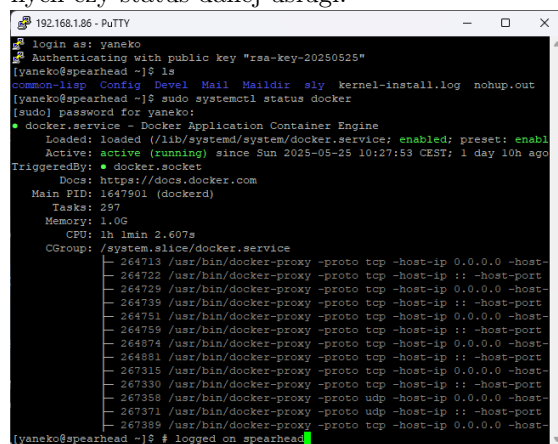
ta – agenta kluczy PuTTY – i dodać tam klucz. Klucz publiczny PuTTY (tekst z pola „Public key for pasting...” w PuTTYgen) należy skopiować na serwer do ~/.ssh/authorized_keys tak

samo, jak w poprzednich metodach. Po dodaniu klucza do authorized_keys należy skonfigurować PuTTY wskazując wygenerowany prywatny plik .ppk i przetestować połączenie.

3 Zastosowania SSH

3.1 Zdalny terminal

Najpopularniejszym zastosowaniem protokołu SSH jest zdalny terminal. SSH umożliwia to, co umożliwia „zwykły” terminal używany lokalnie, lecz przez sieć. Dzięki zdalnemu połączeniu nie trzeba znajdować się fizycznie przy serwerze, żeby sprawdzić np. obecność plików konfiguracyjnych czy status danej usługi.



```
192.168.1.86 - PuTTY
login as: yaneko
Authenticating with public key "rsa-key-20250525"
[yaneko@spearhead ~]$ ls
common-lisp  Config  Devel  Mail  Maildir  sly  kernel-install.log  nohup.out
[yaneko@spearhead ~]$ sudo systemctl status docker
[sudo] password for yaneko:
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; preset: enable)
   Active: active (running) since Sun 2025-05-25 10:27:53 CEST; 1 day 10h ago
   TriggeredBy: ● docker.socket
   Docs: https://docs.docker.com
   Main PID: 1647501 (dockerd)
   Tasks: 297
   Memory: 1.0G
   CPU: 1h 1min 2.607s
   CGroup: /system.slice/docker.service
           └─ 264713 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-
           └─ 264722 /usr/bin/docker-proxy -proto tcp -host-ip :: -host-port
           └─ 264729 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-
           └─ 264736 /usr/bin/docker-proxy -proto tcp -host-ip :: -host-port
           └─ 264751 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-
           └─ 264759 /usr/bin/docker-proxy -proto tcp -host-ip :: -host-port
           └─ 264874 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-
           └─ 264881 /usr/bin/docker-proxy -proto tcp -host-ip :: -host-port
           └─ 267315 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-
           └─ 267330 /usr/bin/docker-proxy -proto tcp -host-ip :: -host-port
           └─ 267358 /usr/bin/docker-proxy -proto udp -host-ip 0.0.0.0 -host-
           └─ 267371 /usr/bin/docker-proxy -proto udp -host-ip :: -host-port
           └─ 267389 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-
```

3.2 Przesył plików

Drugim popularnym zastosowaniem jest transfer plików. Istnieje kilka protokołów tego transferu, opartych na protokole SSH.

3.2.1 SFTP (Secure FTP)

SFTP to protokół transferu plików bazujący na SSH. Domyślnie używa tego samego szyfrowanego kanału SSH i tych samych metod uwierzytelniania co SSH. Oznacza to, że jeśli już mamy skonfigurowany klucz dla SSH, wystarczy wpisać w terminalu: `sftp user@host`.

Sesja SFTP rozpocznie się tak samo jak zwykłe połączenie SSH (możemy się spodziewać monitu o passphrase klucza prywatnego, jeśli go ustawiliśmy). Po zalogowaniu otrzymujemy interaktywny prompt `sftp>`, w którym można używać komend typu `ls`, `cd`, `get filename`, `put filename` itp., aby pobierać lub wysyłać pliki[4].

Ponieważ uwierzytelnienie odbywa się kluczem, SFTP nie będzie już prosić o hasło użyt-

kownika (chyba że klucz miał ustawione hasło).

3.2.2 SCP (Secure Copy)

Polecenie `scp` pozwala kopiować pliki między maszynami z wykorzystaniem SSH. Jeśli autoryzacja kluczem jest skonfigurowana, wystarczy użyć `scp` tak jak zwykle, np.:

```
scp lokalny_plik.txt
user@host:/zdalna/sciezka/
```



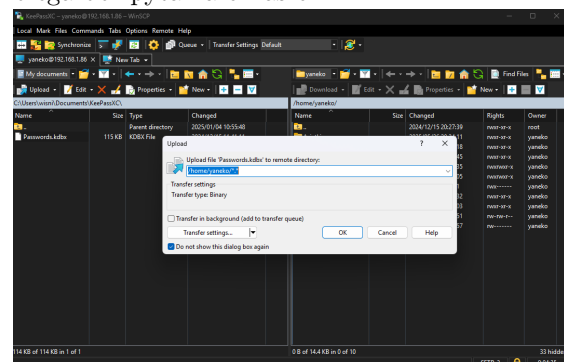
```
[yan3k@WinPC ~]$ scp spearhead:/etc/ssh/sshd_config ./
sshd_config 100% 3222 6.1MB/s 00:00
```

Ta komenda przeniesie plik na serwer bez potrzeby podawania hasła (chyba że nasz klucz ma passphrase). Po skopiowaniu pliku `scp` wyświetli komunikat o postępie, a przesłany plik znajdziemy w zadanym katalogu na hoście.

3.2.3 WinSCP

WinSCP to graficzny klient SFTP/SCP na Windows. Po wybraniu protokołu SFTP wpisujemy dane sesji (nazwa hosta, port, użytkownik). W Advanced (zaawansowanych ustawieniach) przechodzimy do SSH → Authentication i wskazujemy ścieżkę do pliku klucza prywatnego (PuTTY .ppk)[5]. WinSCP automatycznie użyje tego klucza przy nawiązywaniu połączenia.

Po udanym zalogowaniu widzimy interfejs drag&drop dwóch katalogów (lokalnego i zdalnego) – możemy wtedy przeciągać pliki albo używać komend Get i Put jak w SFTP. Dzięki zastosowaniu klucza prywatnego transfer plików przebiega bez pytania o hasło.



4 Bibliografia

Artykuły

- [1] Tatu Ylonen, *What is SSH (Secure Shell)?*, <https://www.ssh.com/academy/ssh>, [dostęp 24.05.2025].
- [2] Microsoft Learn, *Key-based authentication in OpenSSH for Windows*, https://learn.microsoft.com/en-us/windows-server/administration/openssh/openssh_keymanagement [dostęp 24.05.2025].
- [3] OpenSSH Release Notes, *OpenSSH 10.0*, <https://www.openssh.com/txt/release-10.0>, [dostęp 24.05.2025].
- [4] DigitalOcean Community Tutorial, *How To Use SFTP to Securely Transfer Files with a Remote Server*, <https://www.digitalocean.com/community/tutorials/how-to-use-sftp-to-securely-transfer-files-with-a-remote-server>, [dostęp 24.05.2025].
- [5] SuperUser forum, *How do I get WinSCP to connect to an SSH server with a private key that I specify?*, <https://superuser.com/questions/1644862/how-do-i-get-winscp-to-connect-to-an-ssh-server-with-a-private-key-that-i-specif>, [dostęp 24.05.2025].

Dokumentacje techniczne

- [6] Chris Lonvick, Tatu Ylonen, Styczeń 2006, *The Secure Shell (SSH) Transport Layer Protocol*, <https://www.rfc-editor.org/rfc/rfc4253>, [dostęp 24.05.2025].
- [7] OpenBSD manual page server, *ssh_config*, https://man.openbsd.org/ssh_config, [dostęp 24.05.2025].
- [8] OpenBSD manual page server, *ssh-keygen*, <https://man.openbsd.org/OpenBSD-current/man1/ssh-keygen>, [dostęp 24.05.2025].