ORACLE

Key-Management

Oracle Cloud Infrastructure

Cloud built for Enterprises

Technology Cloud Engineering (TCE)
Oracle Deutschland
Dezember 2021

Safe harbor statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.

Agenda

1. Key Management – aber richtig!
2. Nutzung (ssh / putty)

Es gibt zwei Nachrichten…

Die gute Nachricht:

Wenn man keine Passphrase verwendet, ist alles recht einfach.

Die schlechte Nachricht:

Wenn man „sichere“ Keys nutzen möchte (mit Passphrase verschlüsselt), dann ist relevant welches Tool man zur Key-Generierung verwendet, welche Crypto-Libraries installiert bzw. welche Crypto-Policies aktiviert sind und ob „crypto.fips_enabled“ gesetzt ist. In der Cloud Shell ist fips enabled und daher können bestimmte Cipher und MD5 nicht genutzt werden:

```
sysctl crypto.fips_enabled  # Ausgabe: crypto.fips_enabled = 1
openssl version             # Ausgabe: OpenSSL 1.0.2k-fips  26 Jan 2017
```

Erklärung

Federal Information Processing Standard (FIPS) ist ein Standard um sensitive Daten zu schützen (und in den USA bei Behörden vorgeschrieben).

ssh-keygen (OpenSSL) erzeugte bis ca. 2000 private Schlüssel mit dem Header „BEGIN ENCRYPTED **RSA** PRIVATE KEY“ (Traditionelles Format SSLeay). Dabei wird ein MD5 Hash genutzt. Mit FIPS ist MD5 nicht mehr erlaubt und es wird ein SHA1 Hash genutzt. Das neue Format heißt PKCS#8 und der Header beginnt mit „BEGIN ENCRYPTED PRIVATE KEY“.

Auf einer VM mit fips enabled, kann ein verschlüsselter SSLea private key nicht erzeugt, umgewandelt (in ein anderes Format) oder die Passphrase geändert werden. Möchte man per „ssh“ auf eine andere VM, werden nur unverschlüsselte SSLeay keys oder PKCS#8 keys unterstützt. FIPS 140-2 Non-Proprietary Security Policy:

https://www.oracle.com/technetwork/topics/security/140sp3757-7142108.pdf

Weitere Infos und Fazit

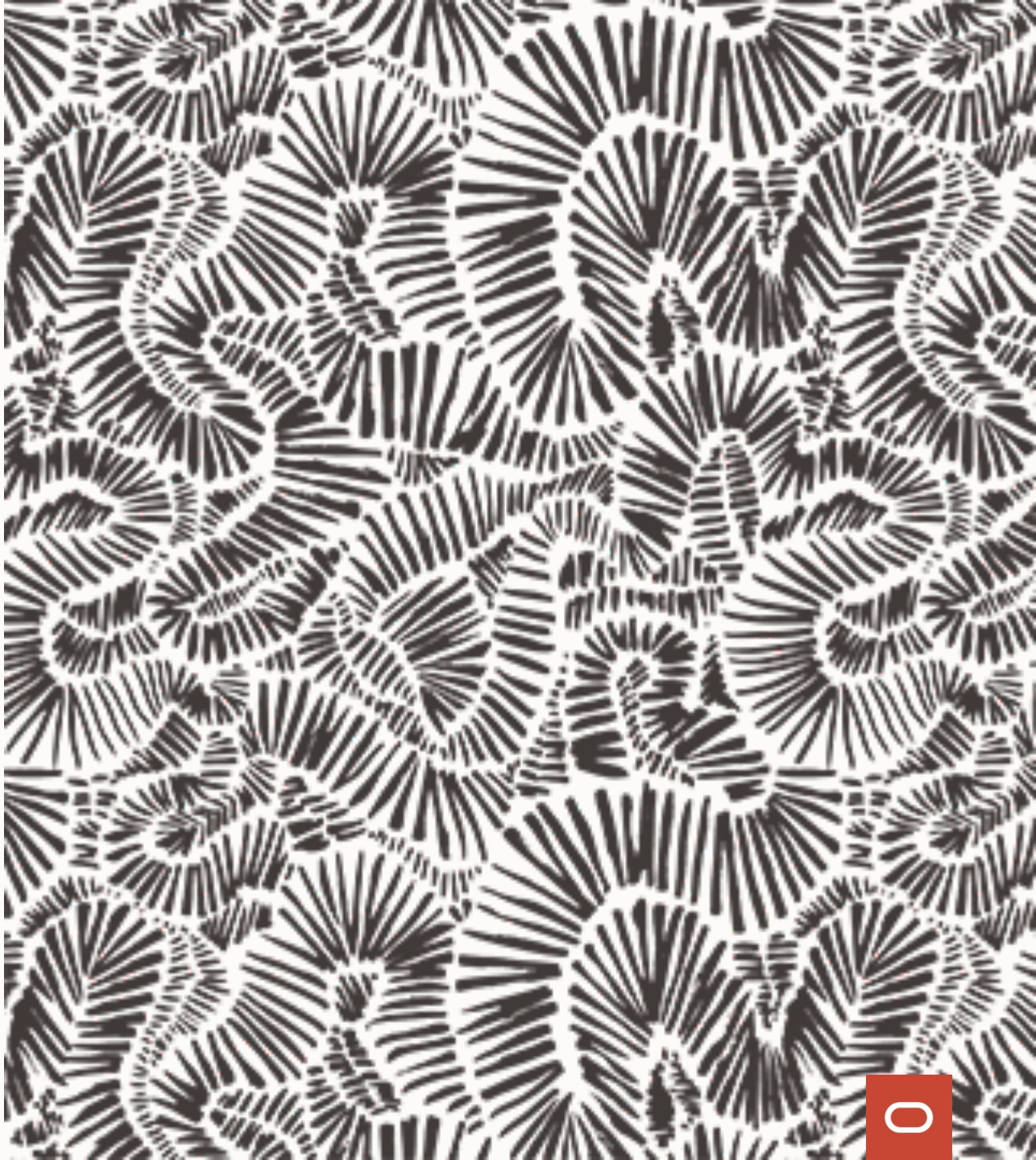
- oci cli (Version 3.3.2) funktioniert sowohl mit SSLeay als auch mit PKCS#8 keys problemlos bei fips enabled als auch disabled.
- ssh funktioniert mit mit SSLeay keys nur bei fips disabled – PKCS#8 keys funktionieren bei fips enabled und disabled. Dies gilt auch für die direkte Nutzung des REST-APIs.
- terraform (Version 1.0.11) funktioniert nur mit SSLeay (RSA) oder ECDSA und nicht mit PKCS#8 keys – alternativ kann man aber „Instance Pricipal“ Authorisierung nutzen.
- Alles funktioniert problemlos mit unverschlüsselten Keys.

Fazit:

Möchte man verschlüsselte Keys nutzen, sollte man für oci cli und terraform SSLeay und für ssh und REST-API PKCS#8 keys nutzen.

Key-Managment

- Keys generieren (ein Key für alle Tools?)
  - oci cli
  - ssh-keygen
  - openssl
  - puttygen
  - Externe Tools
- Passphrase oder Cipher ändern
- SSH Keys rotieren



Key-Generierung mit oci cli (bis Version 3.3.3)

```
mkdir ~/keys 2>/dev/null; cd ~/keys
oci setup keys --overwrite --output-dir . --passphrase "12345" >/dev/null 2>&1

head -n 6 oci_api_key.pem            # private key im RSA Format

-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 5,ENCRYPTED
DEK-Info: AES-256-CBC,94DEEBD1CF1BAC31104BBEABCD2B0386

Korj+axQ1IzN2EKCE0uWkcMJnaMs5/vdWRLUZpjwtXQTX/cOWJqZflDQqTxTI8cX
HJrhIMfBbDcAgUXDGR+WKraLfRr8dTuFFStsYXKtzuRtTfszZNfxgxEWZPi0d78K

head -n 4 oci_api_key_public.pem     # public key (PEM format)

-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAqL1DNJbriyvizELUBEBQ
YLrhIMfBbDcAgUXDGR+WKraLfRr8dTuFFStsYXKtzuRtTfszZNfxgxEWZPi0d25L
lEjEUFrQz8q/63ju8WqQx874p/Uj7suEud1/ra2i1+7EGC11LnHJx9nWtH0EKTpw
```

Key-Generierung mit oci cli (bis Version 3.3.3)

```
oci setup keys --overwrite --output-dir .  # ohne passphrase

head -n 4 oci_api_key.pem            # private key

-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAwLp7HKl8P4tP1ERE6zQbn6OGpUpveNBHomjFQ4ng3+g7Z6EG
hbfHHVwebBp9S6PrDEZ40/I2CQ6pQQRsTaC47n9tZ5k2Ocw/HX7e2qon+jE6qR0G
arOOKmWeLAaoELlXB8lt0173yCgBERdNxwnEyM2WKoASwB0pETG0QwiQ1iD9YN25

head -n 4 oci_api_key_public.pem    # public key (PEM format)

-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAwLp7HKl8P4tP1ERE6zQb
n6OGpUpveNBHomjFQ4ng3+g7Z6EGhbfHHVwebBp9S6PrDEZ40/I2CQ6pQQRsTaC4
7n9tZ5k2Ocw/HX7e2qon+jE6qR0GarOOKmWeLAaoELlXB8lt0173yCgBERdNxwnE
```

Erzeugung des RSA Public keys (aus Private Key)

```
# Generate rsa public key – does not work with passphrase if fips enabled

username="opc"; passphrase=""

result=`ssh-keygen -y -f oci_api_key.pem -P "$passphrase"`
echo "$result $username" > oci_api_key.pub

cat oci_api_key.pub

ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDOxOjcO7d5ZStVNa9Ji7SfobjgXygOJgDBTKCCodIeYIMLed
HOURJjEMrcfpvVIQqqKJjVUJkcrh0gsLTFWD2gZH68WbtzsHT+3tRxaIgm+x1opJM1+oLX1vKBwUCr
z7KoY4b0G7PDWtjQMnQFCqqgL0x+9mwJ3XvFNXV2svD1zp0vjw5yzkZjb0TKBTW2Q9Hmm8DeAvGvyV
241HwHMmkiiXdjWEaIhRg7y2P/o+pMzbns4CTrke/vJ0dB7eWpSKE46JV6oINV5TvUpp1sxQ01J/wX
Z77sKeiSQyhcvEmcbTrJMSpfaQSPGN1yqEsjRXbFSUFEbW5dZhmBy608Qd3f opc
```

Erzeugung des RSA Public keys (aus Public Key)

```
# Generate rsa public key from public key – works everytime

username="opc"

result=`ssh-keygen -i -m PKCS8 -f oci_api_key_public.pem`
echo "$result $username" > oci_api_key.pub

cat oci_api_key.pub

ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDOxOjcO7d5ZStVNa9Ji7SfobjgXygOJgDBTKCCodIeYIMLed
HOURJjEMrcfpvVIQqqKJjVUJkcrh0gsLTFWD2gZH68WbtzsHT+3tRxaIgm+x1opJM1+oLX1vKBwUCr
z7KoY4b0G7PDWtjQMnQFCqqgL0x+9mwJ3XvFNXV2svD1zp0vjw5yzkZjb0TKBTW2Q9Hmm8DeAvGvyV
241HwHMmkiiXdjWEaIhRg7y2P/o+pMzbns4CTrke/vJ0dB7eWpSKE46JV6oINV5TvUpp1sxQ01J/wX
Z77sKeiSQyhcvEmcbTrJMSpfaQSPGN1yqEsjRXbFSUFEbW5dZhmBy608Qd3f opc
```

Fingerprint auslesen (aus Private Key)

```
# Get fingerprint – does not work correct with passphrase if fips enabled

pp=""
keyfile="oci_api_key.pem"

result=`openssl rsa -pubout -outform DER -inform PEM -in "$keyfile" \
  -passin "pass:$pp" 2>/dev/null \
  | openssl md5 -c -non-fips-allow 2>/dev/null | sed 's|(stdin)= ||g'`

echo "$result"

55:9b:91:5c:e5:d3:4f:9b:4a:c2:e8:c8:58:a7:d2:43
```

Fingerprint auslesen (aus Public Key)

```
# Get fingerprint from public key – works everytime

keyfile="oci_api_key_public.pem"

result=`openssl pkey -pubin -pubout -outform DER -in "$keyfile" 2>/dev/null \
  | openssl md5 -c -non-fips-allow 2>/dev/null | sed 's|(stdin)= ||g'`

echo "$result"

55:9b:91:5c:e5:d3:4f:9b:4a:c2:e8:c8:58:a7:d2:43
```

Key-Generierung mit oci cli

Vorteile:

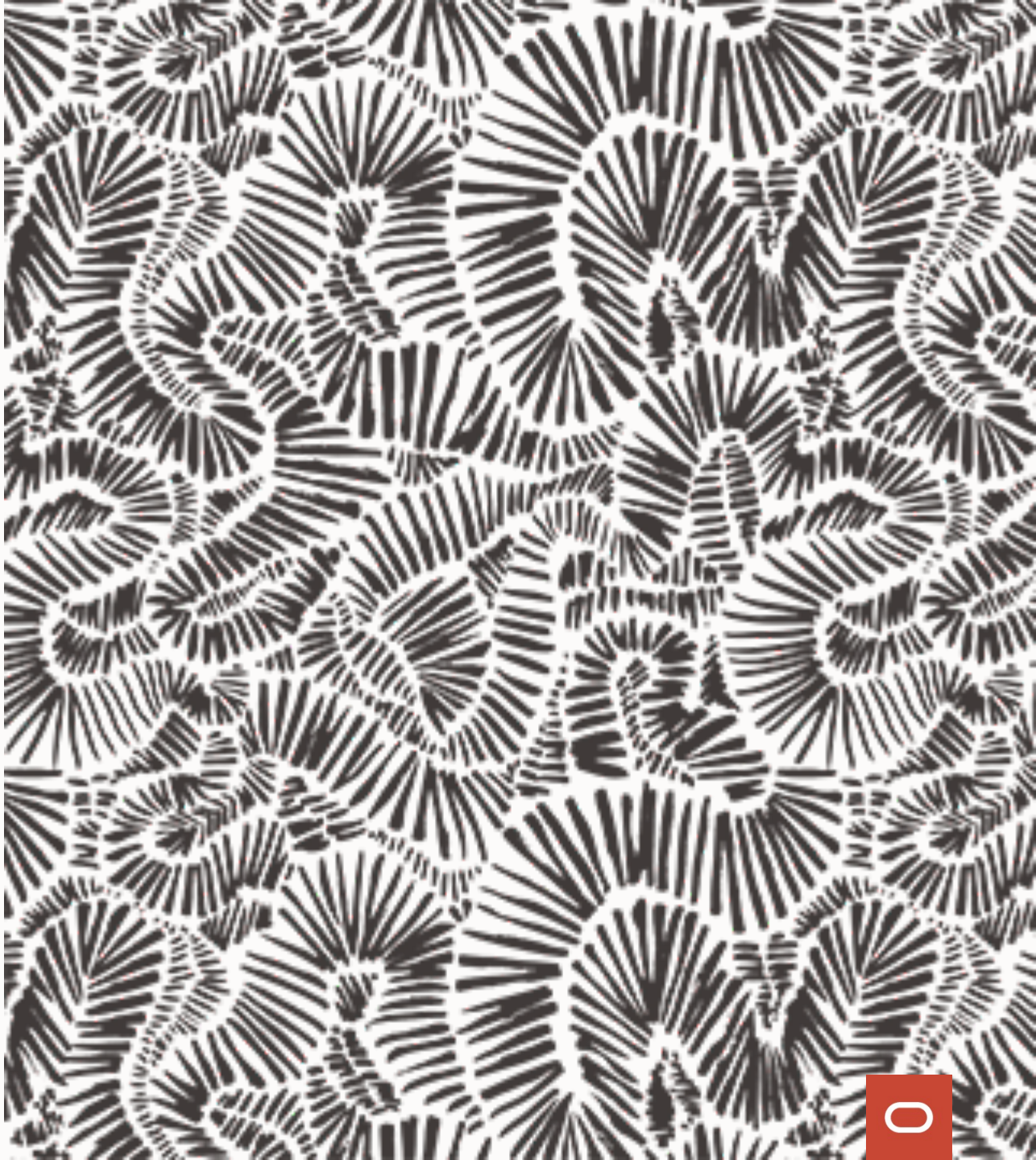
- Einfache Key-Generierung für API Zugriff. Läuft perfekt (mit und ohne Passphrase) mit oci cli (terraform bisher nur mit RSA SSLeay).
- Bis Version 3.3.3: Erstellung auf Cloud Shell (fips enabled) und Linux Instanz (OCI Oracle Linux 7.9 – fips disabled) gleich, da anscheinend Crypto-Libraries unabhängig davon verwendet werden. Ab Version 3.4.0: PRIVATE-KEY HMACWITHSHA256

Nachteile:

- Passphrase darf **keine Sonderzeichen** wie Umlaute enthalten und kann in der Cloud Shell nicht geändert werden, wenn er verschlüsselt erstellt wurde (Cipher oder Passphrase ändern).
- Key wird nur mit 2048 Bits Länge erstellt – AES-256-CBC ist aber relativ sicher verschlüsselt.
- Falls verschlüsselt, kann damit „ssh“ (wenn fips enabled) nicht genutzt und man darf diesen Key auch ggf. nicht verwenden (mit terraform oder CLI / SDK), wenn man **FIPS zertifiziert** bleiben möchte.

Key-Managment

- Keys generieren (ein Key für alle Tools?)
  - oci cli
  - ssh-keygen
  - openssl
  - puttygen
  - Externe Tools
- Passphrase oder Cipher ändern
- SSH Keys rotieren



Key-Generierung mit ssh-keygen (**fips disabled**)

```
# Generate RSA key on fips disabled platform

bits=4096
pp="12345"
username="opc"
keyfile="id_rsa"

ssh-keygen -q -t rsa -N "$pp" -b $bits -C "$username" -m PEM -f "$keyfile"
2>/dev/null

head -n 6 $keyfile

-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: AES-128-CBC,8EDC065A8DA3456DFE3C27286FB1A489

Korj+axQ1IzN2EKCE0uWkcMJnaMs5/vdWRLUZpjwtXQTX/cOWJqZflDQqTxTI8cX
HJrhIMfBbDcAgUXDGR+WKraLfRr8dTuFFStsYXKtzuRtTfszZNfxgxEWZPi0d78K
```

Key-Generierung mit ssh-keygen (**fips enabled**)

```
head -n 6 $keyfile

-----BEGIN ENCRYPTED PRIVATE KEY-----
MIIJnzBJBgkqhkiG9w0BBQ0wPDAbBgkqhkiG9w0BBQwwDgQIJP3Yz+hLrisCAggA
MB0GCWCGSAFlAwQBAgQQpNhGeRL5u3UNDoI6zM2aLwSCCVD32TjRSU3tsdcY7yEr
2BtTs3pZQcf8HsE23km9TSf8TkRfwqYD2Mtk7HF5vir/zxCh8LdzpGXV2e4IbjSr
MmSlPG0e5gi8Wbbt6Cuh6yx1tWzOisFgq0daVmhBinlBmPc2O3PO6NVsl7XLz2GQ
J4sPRjr+Y43mdNbpPuCXPfMwTeWvNKe50izHDv35MBunUKoDFVcDXIE04RE/qZzo

cat ${keyfile}.pub

ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAACAQCQKjtrpb7HPA8ZeD91myrinI2qMVjhaGYSwu0m0MtfoJMxi54rl0
/VrXA11GynR9wDhjbIkVywizhWSn653AnFMRWLpFw3CL8E4Pyf/sqfq44HTAJ/vmWNCJcRIW8LK+7Z+9Aj
HGNJFzjTdD7x9iTy3SqCFk13WSNrgU5M9w8URkjHun1WXR8I/YFPdBz7uP7p66Gj0jDIYjkDdFAP/l6ohy
nlSjsIgwcH4rQ3D2w6O3EoKlPxA+Sx2obPPlHPuF2MP4JF/vva2o7HyLWKsw7wN+lGNKxkABnHNtGLyJ3G
MNRPLqu+FoRUWoYEk7BOI6Qz4SzHlpGYVNH8CEtRJXUzrEy0wUulaBRh4R4JhWKjE4DRr8VFcZ3M9wETfR
ZxDL+yMJDtYjYfZSVSM72Sb0LVg0F1M/9YLs3PRyyq2ciPRiVpZ/m349H67SlbDLiPCEBX/tpxrN2hq5Fg
cGEUTCq4Tzpka/w== opc
```

Key-Generierung mit ssh-keygen

Vorteile:

- Einfache Key-Generierung für ssh Zugriff. Länge des Schlüssel kann frei gewählt werden.
- Erstellung auf Cloud Shell (fips enabled) und Linux Instanz (OCI Oracle Linux 7.9 – fips disabled) unterschiedlich. Auf disabled wird ein SSLeay key erstellt und auf enabled wird ein PKCS#8 key erstellt.
- Läuft perfekt (mit und ohne Passphrase) mit ssh und REST-API (wenn der Key auf der Zielplattform genutzt wird, wo er erstellt wurde).

Nachteile:

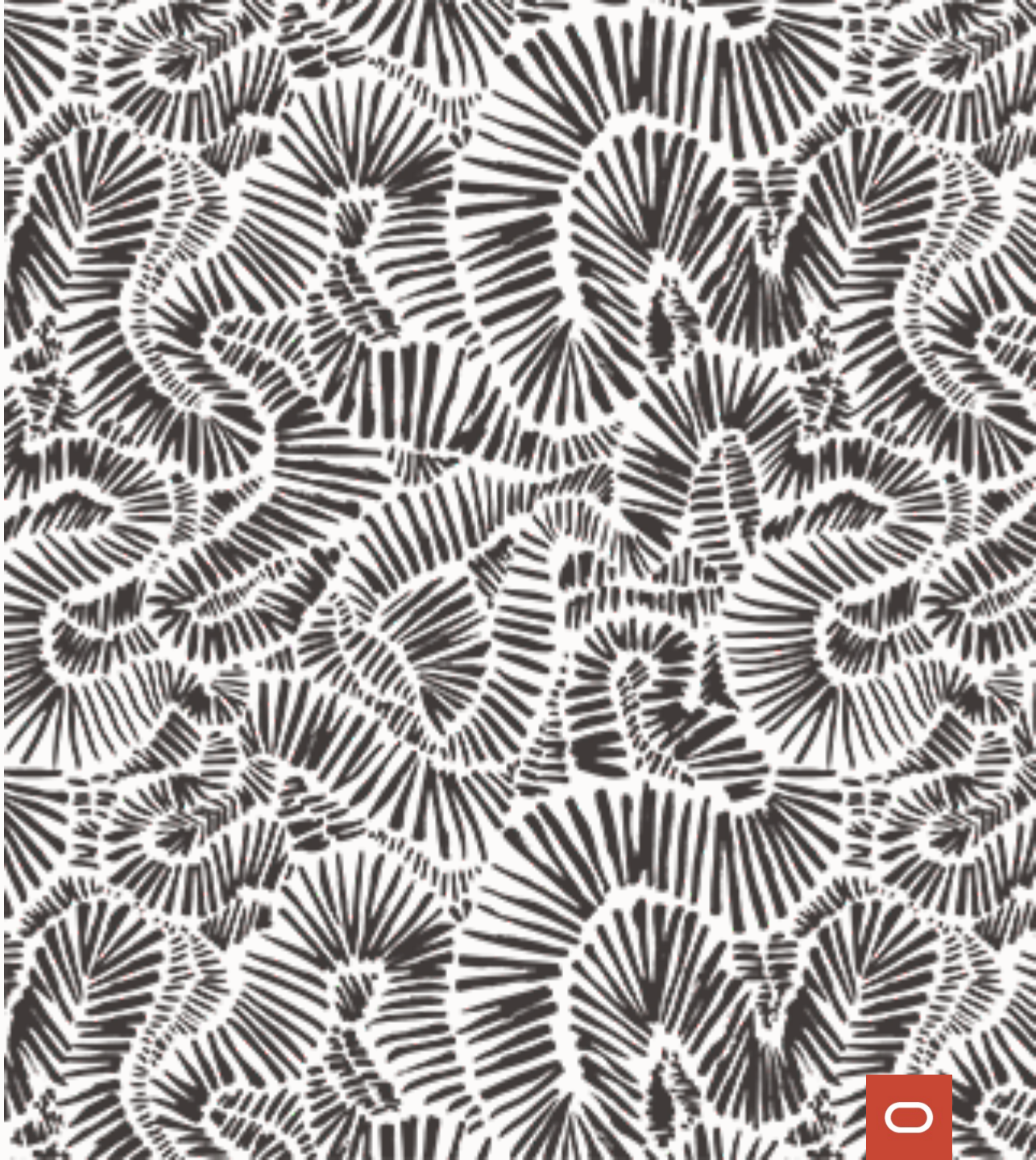
- PKCS#8 key kann nicht von terraform genutzt werden. Cipher AES-128-CBC ist relativ unsicher.

Fazit bisher:

Man könnte zwei Keys für die unterschiedlichen Zwecke erstellen. Möchte man das?

Key-Managment

- Keys generieren (ein Key für alle Tools?)
  - oci cli
  - ssh-keygen
  - openssl
  - puttygen
  - Externe Tools
- Passphrase oder Cipher ändern
- SSH Keys rotieren



Key-Generierung mit openssl (**fips disabled**)

```
# Generate RSA key

bits=4096
pp="12345"
username="opc"
keyfile="id_rsa"

openssl genrsa -aes256 -out "$keyfile" -passout "pass:$pp" $bits 2>/dev/null

head -n 6 "$keyfile"

-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: AES-256-CBC,FF4B97008525FC8E89BFF8DBC852A328

LoxwbE3PpG8vDPazT93aJlYhDUQfpBr3nPPvl9k1pKkb3XAoFrdmFEQYY3YRm4Ld
HJrhIMfBbDcAgUXDGR+WKraLfRr8dTuFFStsYXKtzuRtTfszZNfxgxEWZPi0d78K
```

Key-Generierung mit openssl (**fips enabled**)

```
# Generate RSA key

bits=4096
pp="12345"
username="opc"
keyfile="id_rsa"

openssl genrsa -aes256 -out "$keyfile" -passout "pass:$pp" $bits 2>/dev/null

head -n 5 "$keyfile"

-----BEGIN ENCRYPTED PRIVATE KEY-----
MIIJnzBJBgkqhkiG9w0BBQ0wPDAbBgkqhkiG9w0BBQwwDgQIWEEPvXebnVwCAggA
MB0GCWCGSAFlAwQBKgQQ7Gk3YY9mUxkC7u2C0d6k3ASCCVCyrdWi+9OlaM1S3r5L
GiWWAiruzRxmPcQirnmpFI6gp7qtdcVrKJNrAR2R/kZPy6005Md3dX8hDG4tFut/
FmoLtW0xkv2t/GU4ASEp6cCgZayiH7QMZTXGcG5HswtGYpSlNRYSz33kOlTYJowt
```

Key-Generierung mit openssl

```
# Public key erzeugen (im PEM Format für API Nutzung)

pp="12345“
keyfile="id_rsa"

openssl rsa -in "$keyfile" -passin "pass:$pp" -pubout -out "${keyfile}.pem" 2>/dev/null

head -n 5 "${keyfile}.pem“

-----BEGIN PUBLIC KEY-----
MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEA5xQBqLFY+CAheOCi/xtQ
4khy0pOxkUHRxuhlPjNlCkQf3eOHXVNbEfl1qohK4A7I6TRFJdIH7MHyz6ll7Hfd
pC8T9S/9/DO39TZwRt13FpfMgAcGwvTuD8xQbpTc+1p1laCSPOshoLH8US7RAcSh
NXQnVn0M3KyDYV1JY+cs5IWBDCNeR473k4EtG5xB5cHejZZ+esWUR1nDH/7DsY98
```

Key-Generierung mit openssl

```
# PKCS#8 key erzeugen (im PEM Format für ssh Nutzung unter fips enabled und disabled)

pp="12345"
keyfile="id_rsa"

openssl pkcs8 -topk8 -v2 aes256 -inform pem -in "$keyfile" -passin "pass:$pp" \
  -passout "pass:$pp" -outform pem -out "${keyfile}.pk8" 2>/dev/null

head -n 5 "${keyfile}.pk8“

-----BEGIN ENCRYPTED PRIVATE KEY-----
MIIJnzBJBgkqhkiG9w0BBQ0wPDAbBgkqhkiG9w0BBQwwDgQIc01NOxUUaxQCAggA
MB0GCWCGSAFlAwQBKgQQhDBDGYHjT4BAGxxttG+X3wSCCVDUosT38kV04n7SaqIw
Mr0Jyok6DJSWnCFOJ+iCl6oQOfMHN+hENlwFSsE4OtVPGvPTP69Ece4gh7BsHO9s
ntiZxIiAF73SfUSPkFyl0mMz04eqtWOnxxEmbjVwqHjQcxjyxtFjndkwhTc+GmQX
```

Key-Generierung mit openssl

Vorteile:

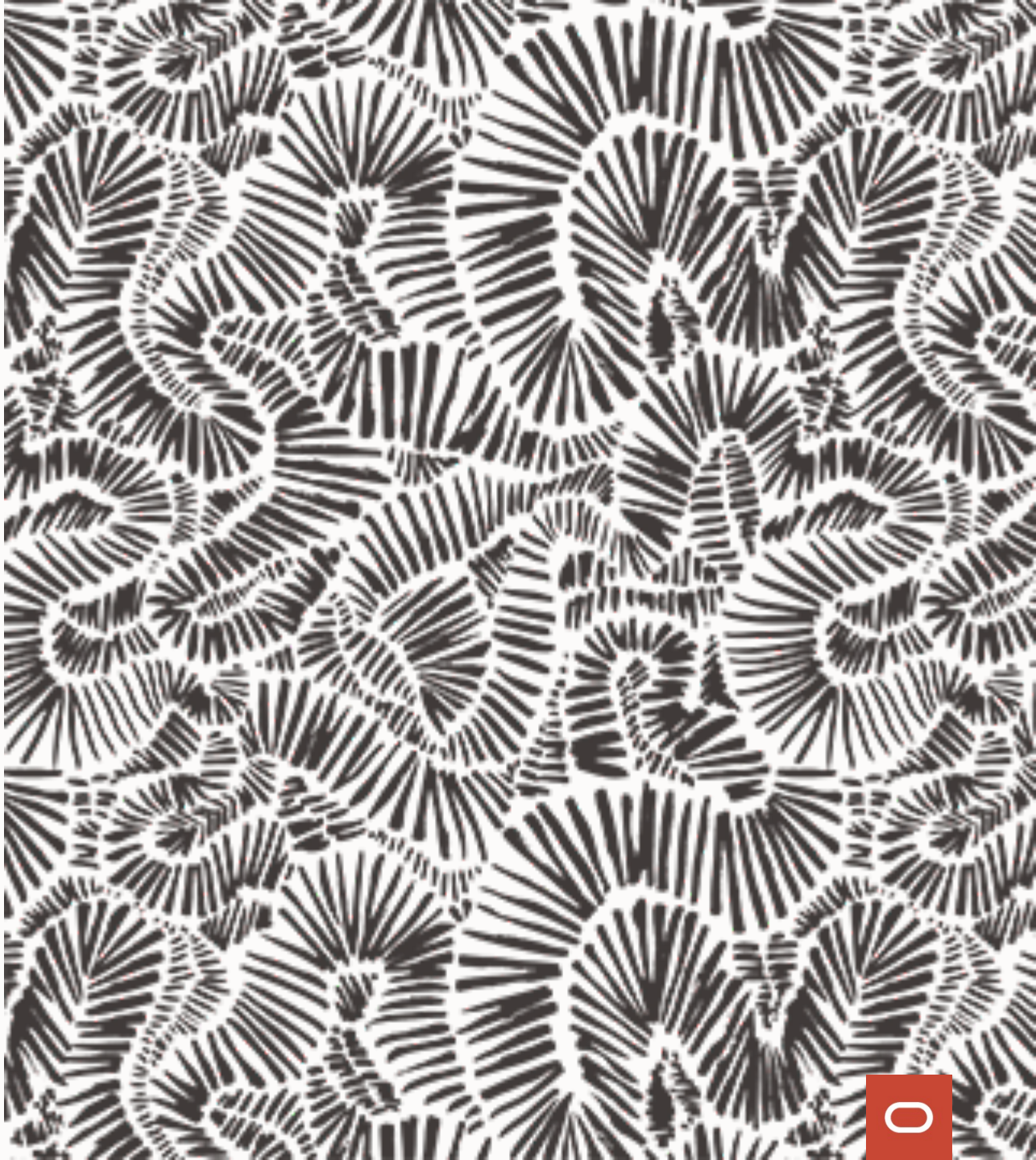
- Vielfache Möglichkeiten für die Key Erstellung und Konvertierung. Länge kann frei gewählt werden.
- Cipher kann frei gewählt werden (ssh-keygen nur AES-128-CBC).
- Konvertierung von SSLeay nach PKCS#8 möglich (auf fips disabled Plattform).
- Es kann ein Key für alle Tools erstellt werden (falls auf fips disabled Plattform erstellt).
- Generierung RSA Public Key (für ssh) siehe Folie 11.
- Sonst die gleichen Anmerkungen wie bei ssh-keygen.

Nachteile:

- Es können nicht alle benötigten Formate in der Cloud Shell erzeugt werden (kein SSLeay key).

Key-Managment

- Keys generieren (ein Key für alle Tools?)
  - oci cli
  - ssh-keygen
  - openssl
  - puttygen
  - Externe Tools
- Passphrase oder Cypher ändern
- SSH Keys rotieren



Key-Generierung mit puttygen

```
# Install puttygen

if [ "$OCI_CLI_CLOUD_SHELL" = "True" ]; then
  mkdir ~/bin 2>/dev/null
  curl -skL https://standby.cloud/download/cmd/i386/64/putty/0.76/Linux/puttygen \
    -o ~/bin/puttygen
  chmod 755 ~/bin/puttygen
else
  sudo yum install putty
fi

# Check if puttygen is in PATH

result=`which puttygen 2>/dev/null`
echo "Found: $result"
```

Key-Generierung mit puttygen

```
# Create non encrypted key

mkdir ~/keys 2>/dev/null
cd ~/keys

bits=4096;
username="opc"
keyfile="id_rsa"
oldpp=""; echo "$oldpp" > oldppfile

puttygen -q -t rsa -b $bits -P --new-passphrase oldppfile -O private-
openssh \
  -C "$username" -o "$keyfile" 2>/dev/null

head -n 3 "$keyfile"

-----BEGIN RSA PRIVATE KEY-----
MIIJKAIBAAKCAgEAkZ+A6BlDxN4vlf3SMP990VINwifou7Qcakrp91QwYT7luTA9
ezZqIVVrCzGmxc2lZmCxkxixCYfoZVWwCmkAOOWCqV30rltqFtGpgBplRhk0MWKP
```

Key-Generierung mit puttygen

```
# Create API public key and PKCS#8 key (encrypted)

newpp="12345"

openssl rsa -in "$keyfile" -passin "pass:$oldpp" -pubout \
  -out "${keyfile}.pem" 2>/dev/null
openssl pkcs8 -topk8 -v2 aes256 -inform pem -in "$keyfile" -passin "pass:$oldpp" \
  -passout "pass:$newpp" -outform pem -out "${keyfile}.pk8" 2>/dev/null

head -n 4 "${keyfile}.pk8"

-----BEGIN ENCRYPTED PRIVATE KEY-----
MIIJnzBJBgkqhkiG9w0BBQ0wPDAbBgkqhkiG9w0BBQwwDgQIFSRlNlI5TqMCAggA
MB0GCWCGSAFlAwQBKgQQmxmRORvBtQtnGEq3EP4sQASCCVDnurk8kyupSD5gn1fx
WacsOyPCzK/8eio15+V+Zc1IpRU136KOSiP4q7MtMaCsu7lrYjzWk6+9oFCB0CFd
```

Key-Generierung mit puttygen

```
# Create putty private keys (version 2 and 3) and encrypt SSLeay key

echo "$newpp" > newppfile

puttygen -q "$keyfile" -P --old-passphrase oldppfile --new-passphrase newppfile \
  -C "$username" --ppk-param version=2 -o "${keyfile}.ppk" 2>/dev/null

puttygen -q "$keyfile" -P --old-passphrase oldppfile --new-passphrase newppfile \
  -C "$username" --ppk-param version=3 -o "${keyfile}.pp3" 2>/dev/null

puttygen "$keyfile" -P --old-passphrase oldppfile --new-passphrase newppfile \
  -O private-openssh -o "$keyfile" 2>/dev/null
```

Key-Generierung mit puttygen

```
# Delete pwd files and show header of SSLeay key

rm –f oldppfile newppfile

head "$keyfile"

-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: DES-EDE3-CBC,78B43885AA0B9510

y+C6ARx4p3lNpTm2O7jmVqpC4Nh4U+MzxikKEhhTQVrLtn2BnSFHozl3Bndegv1z
CIIAAPsmQI/iYsXFEkAQf6DTtDGScNck7pAzpY1S6oyZWMtfyuuIeWndvI5SBb13
1cjqudcILKH81Witl/LKKD+0/E59L35+yCVkuzb7yZ9bZkDBbJT3zbpzKYYFB39R
WAC1KZjtH0FOhUZETK5m0Z/5xFYYEPmOkoS+BAABqSv6+pX7TDJw8bti+KD/l/qN
FpADiGBaJmF8eFZPwrFvakYKuG+FahOV/Yp1RbeJpIYD8eWB3Gq0vtaWv94ZoqXW
18FW1QTHupDfljrUq0ys1c2y/bzGlGjO67Ra8sA39/U9K69NWw9sVFwqkOI1Gk8L
```

Erzeugung des RSA Public keys (aus Public Key)

```
# Generate rsa public key from api public key

result=`ssh-keygen -i -m PKCS8 -f "${keyfile}.pem"`
echo "$result $username" > "${keyfile}.pub"

cat "${keyfile}.pub"

ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDOxOjcO7d5ZStVNa9Ji7SfobjgXygOJgDBTKCCod
IeYIMLedHOURJjEMrcfpvVIQqqKJjVUJkcrh0gsLTFWD2gZH68WbtzsHT+3tRxaIgm+x1o
pJM1+oLX1vKBwUCrz7KoY4b0G7PDWtjQMnQFCqqgL0x+9mwJ3XvFNXV2svD1zp0vjw5yzk
Zjb0TKBTW2Q9Hmm8DeAvGvyV241HwHMmkiiXdjWEaIhRg7y2P/o+pMzbns4CTrke/vJ0dB
7eWpSKE46JV6oINV5TvUpp1sxQ01J/wXZ77sKeiSQyhcvEmcbTrJMSpfaQSPGN1yqEsjRX
bFSUFEbW5dZhmBy608Qd3f opc
```

Key-Generierung mit puttygen

Vorteile:

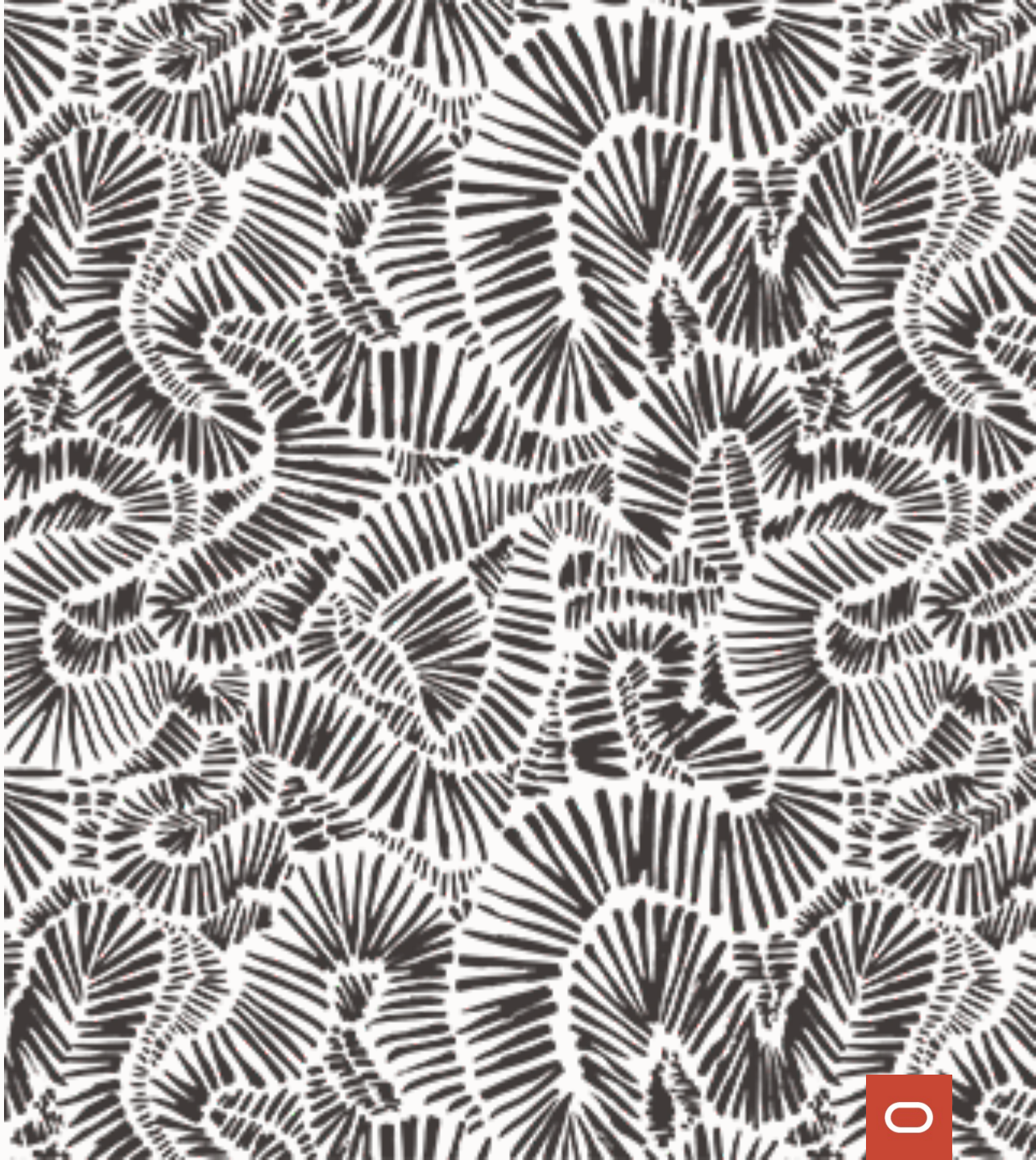
- Es kann ein Key mit allen benötigten Formaten für alle Tools mit puttygen und mit Konvertierung durch openssl und ssh-keygen in Kombination (egal ob fips enabled oder disabled) erstellt werden.
- Zusätzlich kann man auch den PuTTy private key (.ppk) für die Nutzung unter Windows erstellen.
- Der SSLeay key wird auf allen Plattformen erstellt und mit Tripple DES (des3) verschlüsselt (160 bits oder 192 bits – sicherer als AES-128 aber weniger sicher als AES-256). Die Länge des Keys ist frei wählbar (sollte nie kleiner als 2048 bits sein).

Nachteile:

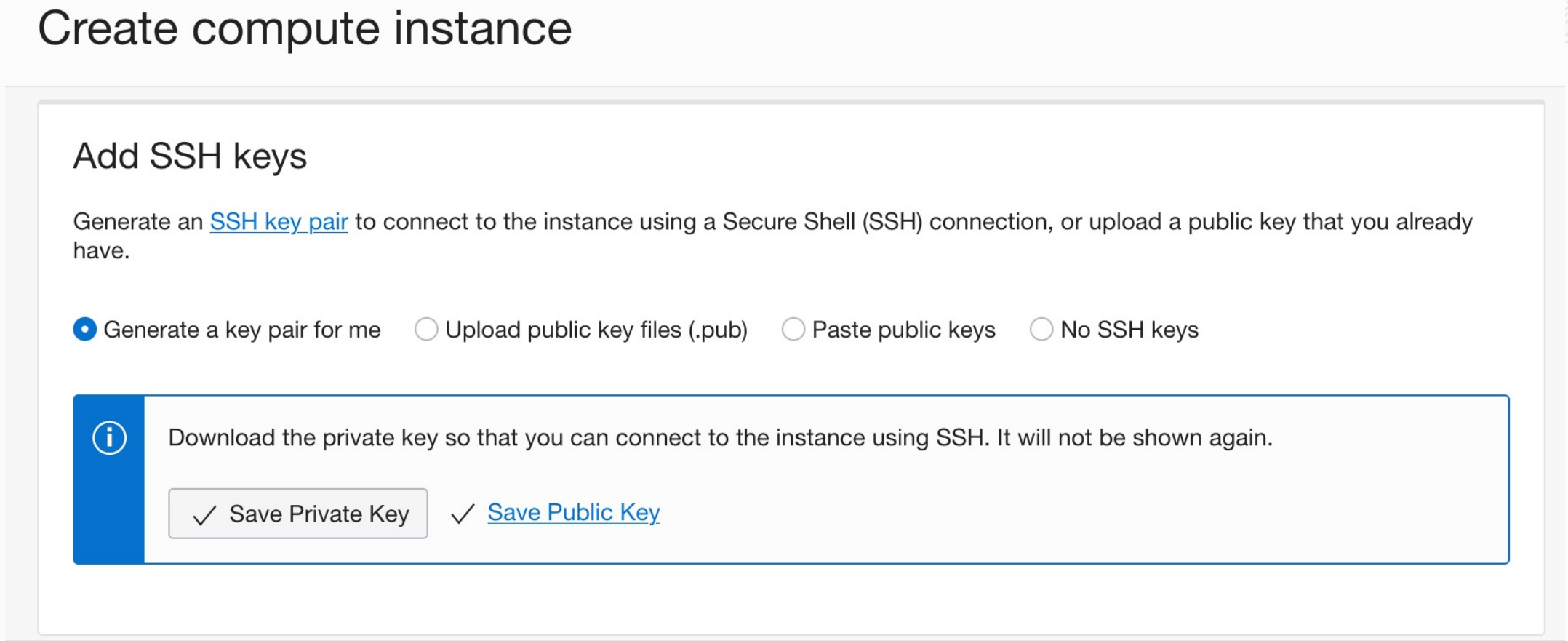
- puttygen muss zuerst lokal installiert werden.
- Mehrere Kommandos müssen hintereinander und in richtiger Reihenfolge abgesetzt werden.

Key-Managment

- Keys generieren (ein Key für alle Tools?)
  - oci cli
  - ssh-keygen
  - openssl
  - puttygen
  - Externe Tools
- Passphrase oder Cipher ändern
- SSH Keys rotieren



Key-Generierung per OCI Web-GUI…



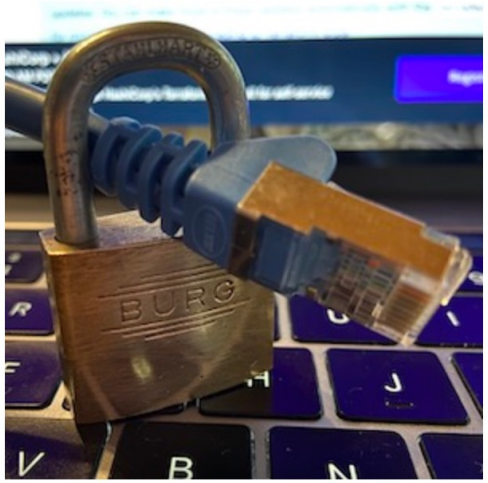
Erzeugt nur einen unverschlüsselten private und public SSLeay key, den man lokal speichern kann.

Wird daher nicht weiter betrachtet.

… und per STANDBY.CLOUD Web-GUI

SSH Key Generator

Verwendungszweck



Sie können dieses Tool nutzen um ein Private- / Public-Key Pair zu generieren. Laden Sie den Public-Key "id_rsa.pub" in die Oracle Cloud und nutzen Sie den Private-Key um per SSH oder PUTTY eine gesicherte Verbindung zu Ihrer Instanz in der Oracle Cloud herzustellen. Die Keys funktionieren sowohl für OPC (Oracle Public Cloud oder auch OCI-C) als auch für OCI (Oracle Cloud Infrastructure).

Geben Sie zwingend einen Benutzernamen (z.B. opc oder opc@hostname) und optional eine Passphrase (mit mindestens 5 Zeichen) an. Wenn Sie eine Passphrase angeben, wird der Private-Key mit dieser verschlüsselt und bei der Anmeldung an Ihrer Instanz werden Sie nach dieser Passphrase gefragt. So ist Ihr Private-Key noch etwas sicherer.

Wenn Sie auf den "Generate" Button klicken, werden die Keys generiert (inklusive einer Info-Datei), gezipt und für den Download zur Verfügung gestellt. Die Schlüssel haben eine Länge von 4096 Bit.

Username:

Passphrase:

Generate | Reset

URL: https://standby.cloud/ssh-keygen/

Erzeugt alle benötigten Keys verschlüsselt oder unverschlüsselt.

Key-Generierung per API zu STANDBY.CLOUD

```
# Download API from https://standby.cloud/ssh-keygen

mkdir ~/bin ~/keys 2>/dev/null

curl -skL https://standby.cloud/ssh-keygen/createkeys-api -o ~/bin/create-keys
chmod 755 ~/bin/create-keys

# Create keys

username="opc"
passphrase="12345"

create-keys "$username" "$passphrase"
if [ $? -eq 0 ]; then
  unzip -q /tmp/"${username}.zip" -d ~/keys
  rm -f /tmp/"${username}.zip"
  cd ~/keys/"$username"
fi
```

Key-Generierung mit STANDBY.CLOUD

Vorteile:

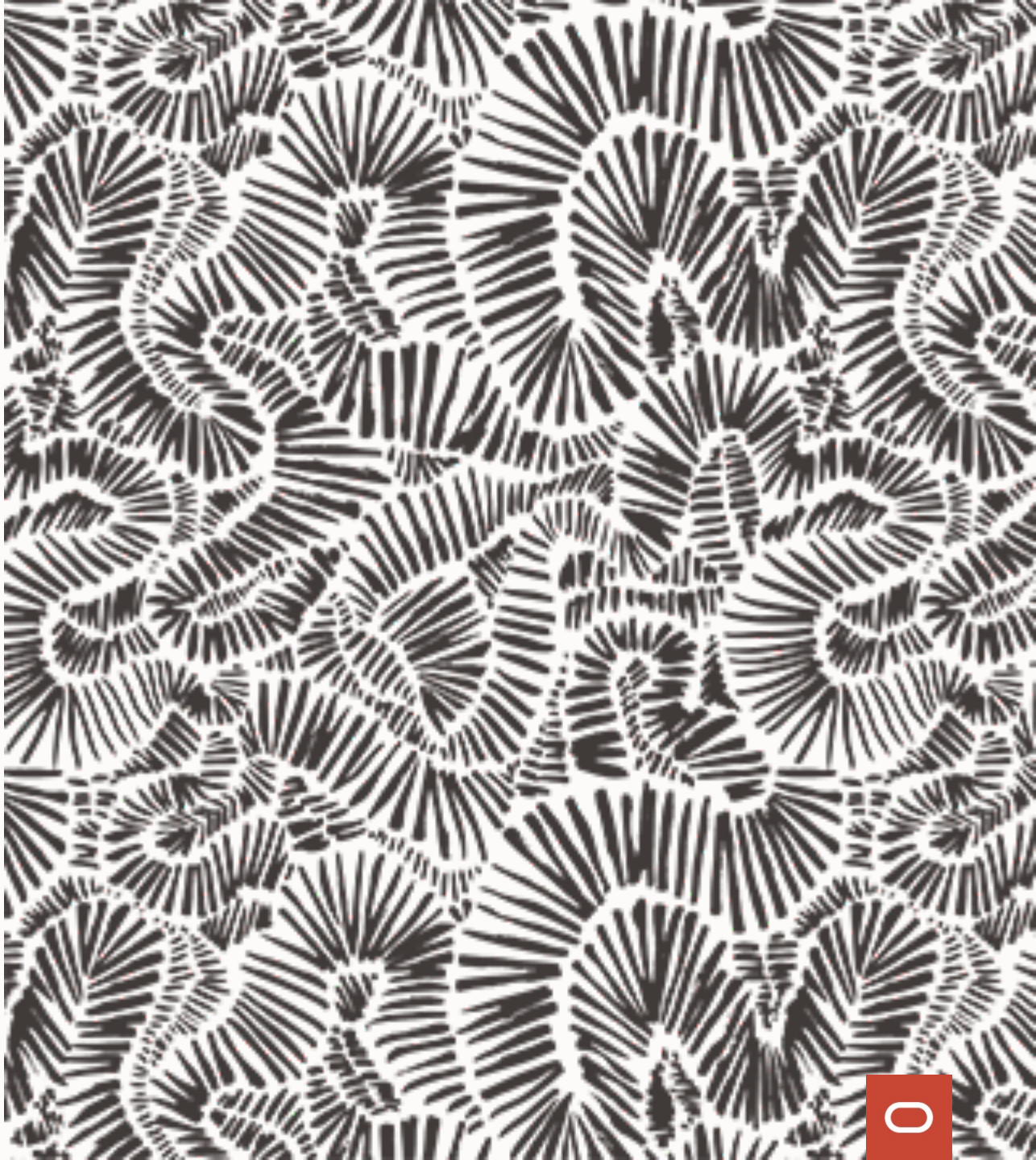
- Einfachste Handhabung. Es wird ein Key in allen benötigten Formaten geliefert.
- Auf der externen Plattform werden alle Keys korrekt generiert. Die benötigten Tools sind vorinstalliert und in der benötigten Version vorhanden.

Nachteile:

- Man muss der externen Plattform vertrauen. Es könnten die Passphrase geloggt oder der Key gespeichert werden.
- Wenn man statt dessen die Keys lokal selbst erzeugen möchte, kann man aber anstelle des APIs auch direkt das Script zum Erzeugen der Keys laden (einfach in dem Script auf der vorherigen Seite „createkeys-api“ durch „createkeys“ ersetzen). Dann muss aber puttygen (und auch ssh-keygen und openssl) lokal installiert sein.

Key-Managment

- Keys generieren (ein Key für alle Tools?)
  - oci cli
  - ssh-keygen
  - openssl
  - puttygen
  - Externe Tools
- Passphrase oder Cipher ändern
- SSH Keys rotieren



Passphrase und Cipher ändern (**openssl**)

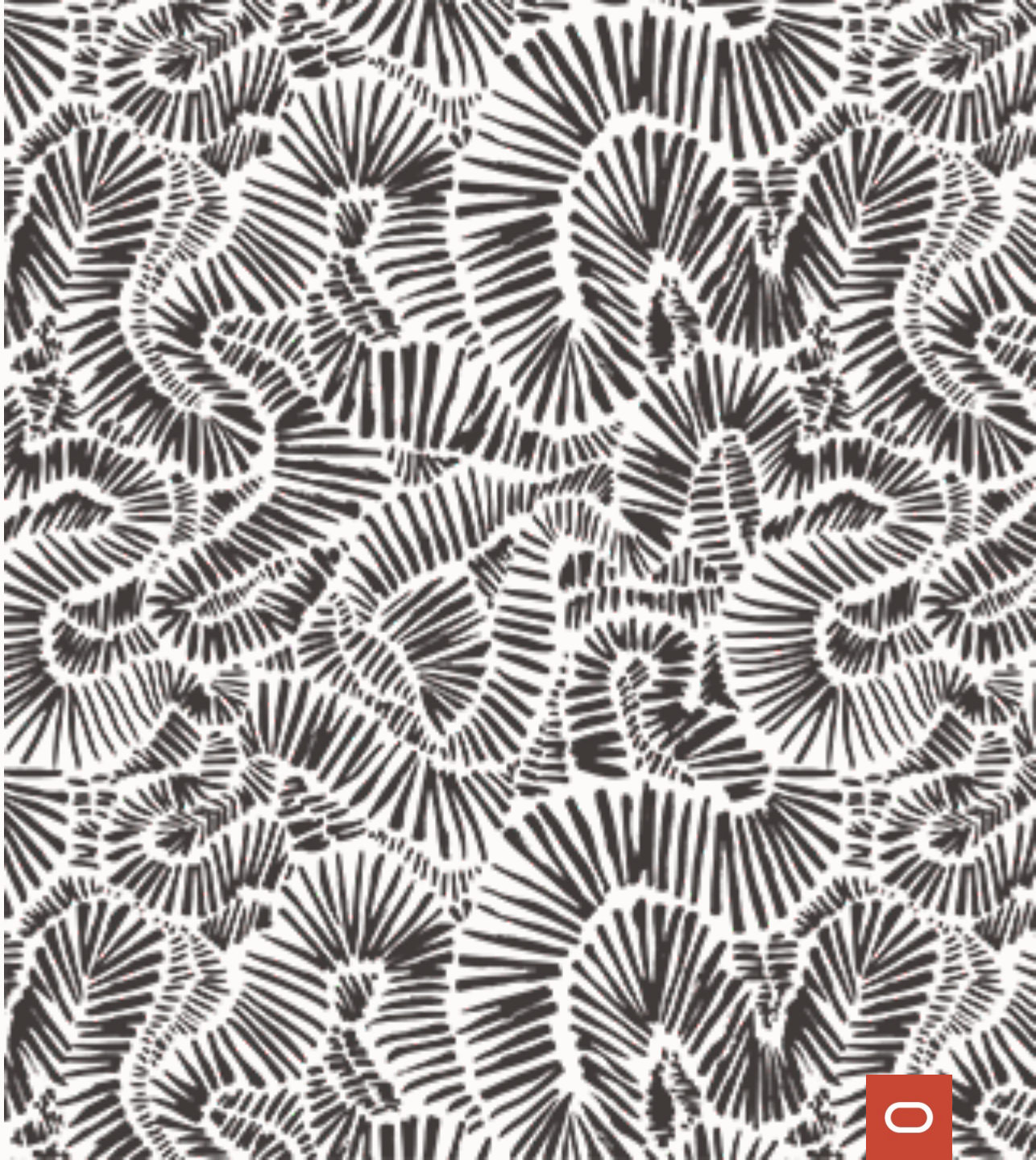
```
# Works only if fips disabled or key was unencrypted
# Without cipher: Key will not be encrypted

keyfile="id_rsa"; cipher="aes256"; oldpp="12345"; newpp="georg"
openssl rsa -$cipher -inform PEM -in "$keyfile" -passin "pass:$oldpp" \
  -outform PEM -out "${keyfile}.new" -passout "pass:$newpp"

head -n 5 "${keyfile}.new"

-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: AES-256-CBC,94DEEBD1CF1BAC31104BBEABCD2B0386

Korj+axQ1IzN2EKCE0uWkcMJnaMs5/vdWRLUZpjwtXQTX/cOWJqZflDQqTxTI8cX

# On fips enabled and passphrase was empty:

-----BEGIN ENCRYPTED PRIVATE KEY-----
```

Passphrase und Cipher ändern (**puttygen**)

```
# Works everytime but only with cipher des3 (Tripple DES)

keyfile="id_rsa"
echo "$oldpp" > oldppfile
echo "$newpp" > newppfile

puttygen "$keyfile" -P --old-passphrase oldppfile --new-passphrase newppfile \
  -O private-openssh -o "${keyfile}.new"

rm –f oldppfile newppfile
head -n 6 "${keyfile}.new"

-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: DES-EDE3-CBC,A946815A2463AF65

83/8GTUdnJUYUN2ltmU43BZ3lhDovmhBaF9gVddQoUkZn8y4h37ugjIP5VJ/fBjV
/h/U8myLEexZbqlhsvlzGjyEERPGjv72ZwkETVNaSc37ezca0ww975I67mJB6Jw4
```

Key-Managment

- Keys generieren (ein Key für alle Tools?)
  - oci cli
  - ssh-keygen
  - openssl
  - puttygen
  - Externe Tools
- Passphrase oder Cipher ändern
- SSH Keys rotieren



SSH-Keys rotieren

Die Keys sollten regelmäßig erneuert / ausgetauscht werden. Die Nutzung der Keys sollte zwischen 90 und 180 Tagen liegen (abhängig von der Company-Policy). Der Public Key wird auf dem Zielrechner in der Datei ~/.ssh/authorized_keys des jeweiligen Users eingetragen, der SSH nutzen soll.

Der Austausch des Schlüssels sollte möglichst automatisiert erfolgen (z.B. per Ansible), damit nicht durch einen Fehler (z.B. bei der manuellen Eingabe) die Instanz nicht mehr zugänglich ist. Sollte der Austausch doch einmal manuell durchgeführt werden, dann den neuen Schlüssel in der „authorized_keys“ eintragen, den Schlüssel testen und bei Erfolg den alten Public Key aus der Datei löschen. Beispiel (vorher):

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQCZ6pZUodui/NxpbVngul8cjq6TpimYfBxT1gy3dLI4T6CImR2W+liOq
lKh7JJVSPodAEIL57dH7X8azQ9By93njzv1qUTusZih+s31Iuo3cRN3B9cDIGBmx/jj5gmWH6Z/I5ePdTZUMa
noneDLE0jkspUw18vuXu5c1P7DlIu995DlDdiR7AGx9QAlwLoLNl6iQTbkVLLuDKJD95r48OA6xhMsfj6tmJx
7APNLMdydCXtjmmmS2jdSii42wYdUZ6Hz+JvUk6s5uHHrYGhYSED9+hHDvNBF5V92pp3/n52shA7Ripc7+UrN
xCY6tLJgHUweo0y1yKd9208xTU9Zdd29 oci
```

SSH-Keys rotieren (Schritt 2)

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQCZ6pZUodui/NxpbVngul8cjq6TpimYfBxT1gy3dLI4T6CImR2W+liOq
lKh7JJVSPodAEIL57dH7X8azQ9By93njzv1qUTusZih+s31Iuo3cRN3B9cDIGBmx/jj5gmWH6Z/I5ePdTZUMa
noneDLE0jkspUw18vuXu5c1P7DlIu995DlDdiR7AGx9QAlwLoLNl6iQTbkVLLuDKJD95r48OA6xhMsfj6tmJx
7APNLMdydCXtjmmmS2jdSii42wYdUZ6Hz+JvUk6s5uHHrYGhYSED9+hHDvNBF5V92pp3/n52shA7Ripc7+UrN
xCY6tLJgHUweo0y1yKd9208xTU9Zdd29 oci

ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAACAQDw+SHcrKbNCQzdjZMPnydfduokFz4tJJgCCDA2Smdi65bFZtXxPuNeK
9r8gsDniOkiZDIKEeop/r7c0T1aRwFGfEdKmQ0I3Qk0GYiDWH35exRvGXAq9v6P4xyDS00aIgZJqcLdKrjuQM
5XysaJqqjWLjloIOF7n/R4eyq0mVLjsT/stuD72S7CBGSocNCjMZCTLeKaw2P1eJ7PEi6jsspV6pSgeIjNuCq
eeoGlg+EK+N3Uixs/P80f/v6fynFmfGeuQTgaeZzw4V2BLe0vzQlrmGSxlsscFHsG0iv9iAm0wwbhySQ+Tp00
uxneDIcs8tpsDVV9Pf7U2sRiQAgJeY0itV5cjT3V1NDwP015lnwmwoPXfMEgtMG1Q8CQeDnSOd8ZgXbv9RYJu
qbbOjfbJ3xzra/4pADIOrxK71caYHwzwCjq8nNovPMQYFMD925NfWZclkPvoHMEBtNfFBUTD2F5c8EEelYZyE
OBAnFOGGg9gv59P9XeS7FhkybgDDhmbWzBzL4wEdmBEyLMz6wuFASJnJYglv2CK3mLn/Rcm5PVpAdPMWnUI8Q
rkyGANINp5KQnMjTz0PCQvQrzyprUtfAF2rqlw6l4/qOPBmzh8oVpTbTNtMKwYemymzsNUjzjuOrG8Q1v9ttw
PKBlSk5+xp4Ah3A890XQ== oci2
```

SSH-Keys rotieren (Schritt 3)

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAACAQDw+SHcrKbNCQzdjZMPnydfduokFz4tJJgCCDA2Smdi65bFZtXxPuNeK
9r8gsDniOkiZDIKEeop/r7c0T1aRwFGfEdKmQ0I3Qk0GYiDWH35exRvGXAq9v6P4xyDS00aIgZJqcLdKrjuQM
5XysaJqqjWLjloIOF7n/R4eyq0mVLjsT/stuD72S7CBGSocNCjMZCTLeKaw2P1eJ7PEi6jsspV6pSgeIjNuCq
eeoGlg+EK+N3Uixs/P80f/v6fynFmfGeuQTgaeZzw4V2BLe0vzQlrmGSxlsscFHsG0iv9iAm0wwbhySQ+Tp00
uxneDIcs8tpsDVV9Pf7U2sRiQAgJeY0itV5cjT3V1NDwP015lnwmwoPXfMEgtMG1Q8CQeDnSOd8ZgXbv9RYJu
qbbOjfbJ3xzra/4pADIOrxK71caYHwzwCjq8nNovPMQYFMD925NfWZclkPvoHMEBtNfFBUTD2F5c8EEelYZyE
OBAnFOGGg9gv59P9XeS7FhkybgDDhmbWzBzL4wEdmBEyLMz6wuFASJnJYglv2CK3mLn/Rcm5PVpAdPMWnUI8Q
rkyGANINp5KQnMjTz0PCQvQrzyprUtfAF2rqlw6l4/qOPBmzh8oVpTbTNtMKwYemymzsNUjzjuOrG8Q1v9ttw
PKBlSk5+xp4Ah3A890XQ== oci2
```

Tipp: Der Public Key darf keine Zeilenumbrüche enthalten (ein neuer Key beginnt in einer neuen Zeile). Deswegen den Key nicht per „more“ sondern „cat“ anzeigen und dann z.B. per Copy & Paste in die Datei eintragen.

SSH-Keys rotieren (Public Keys zentral ablegen)

Der SSH Daemon (sshd) kann so konfiguriert werden, daß beim Anmelden an der Instanz anstatt (oder zusätzlich) der authorized_keys eine zentrale Datei (entweder z.B. im Object Storage für alle Instanzen oder auf einer Instanz für alle ssh User auf der Instanz) ausgewertet wird.

Das Script sollte ausführbar sein und Root-Rechte haben (oder Sie nutzen den User „nobody“). Das einfachste Script (z.B. „get-authkeys“) macht ein „cat“ auf eine zentrale Datei, die wie „authorized_keys“ aufgebaut ist. In der /etc/ssh/sshd_config werden 3 Zeilen (die vorher auskommentiert sind) mit den entsprechenden Werten befüllt z.B.:

```
PubkeyAuthentication yes
AuthorizedKeysCommand /etc/ssh/get-authkeys "%u"     # %u = Username (login user)
AuthorizedKeysCommandUser root
```

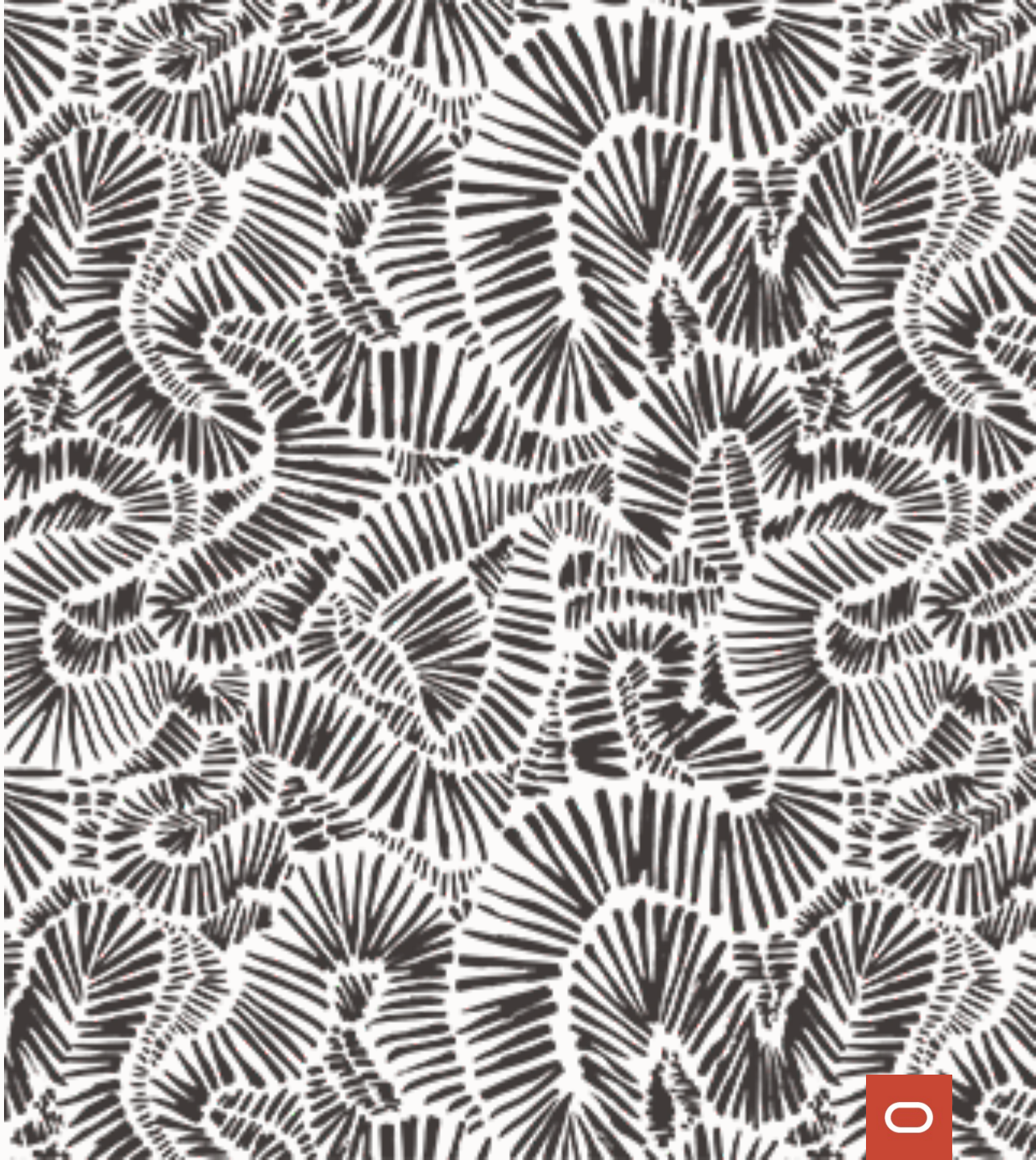
Anschließend den sshd neu starten mit: „sudo systemctl restart sshd“.

Agenda

1. Key Management – aber richtig!
2. Nutzung (ssh / putty)

Nutzung

- ssh / scp
- putty



SSH

Unter Linux (Solaris, MacOS) gelangt man mit dem Kommando „ssh“ und benötigten Parameter auf die Ziel-Instanz z.B.:

```
ssh -i /PathToPrivateKey/privatekey opc@130.68.12.100
```

oder

```
ssh -i "$sshprivkey" "${destuser}@$destip" -J $bastionname  # J = ProxyJump
```

oder

```
ssh -i "$sshprivkey" "${destuser}@$destip" \
  -o ProxyCommand="ssh -W %h:%p $bastionname -i $sshprivkey"
```

Dabei werden Einstellungen aus /etc/ssh/ssh_config verwendet. Diese kann man mit einer lokalen Datei „config“ unter ~/.ssh überschreiben und auch einfach alle Zielrechner und benötigte Keys dort hinterlegen.

Auch ein Port-Forwarding oder ein Jump über einen Bastion- oder Jump-Host sind so einfach definierbar.

SSH (config part 1)

```
cat ~/.ssh/config

Host *
        ForwardAgent no
        ForwardX11 no
        ForwardX11Trusted no
        Port 22
        Protocol 2
        ServerAliveInterval 60
        ServerAliveCountMax 30
        StrictHostKeyChecking no
        UserKnownHostsFile /dev/null
        LogLevel error
```

SSH (config part 2)

```
Host bastion
        HostName 130.162.53.164
        User ubuntu
        IdentityFile "~/keys/bastion.pk8"

Host operator
        HostName 192.168.0.132
        User opc
        ProxyJump bastion
        IdentityFile "~/keys/operator.pk8"

Host essbase
        HostName 192.168.0.150
        User opc
        ProxyJump bastion
        IdentityFile "~/keys/id_rsa.pk8"
        LocalForward 9001 localhost:9001
        LocalForward 7002 localhost:7002
```

SSH /SCP

Sind die Ziel-Instanzen in der config Datei definiert, gelangt man auf den Operator Host (per ProxyJump über den Bastion), in dem man einfach dieses Kommando absetzt:

```
ssh operator
```

Möchte man eine Datei auf den Operator kopieren, kann man dies mit „scp“ erzielen:

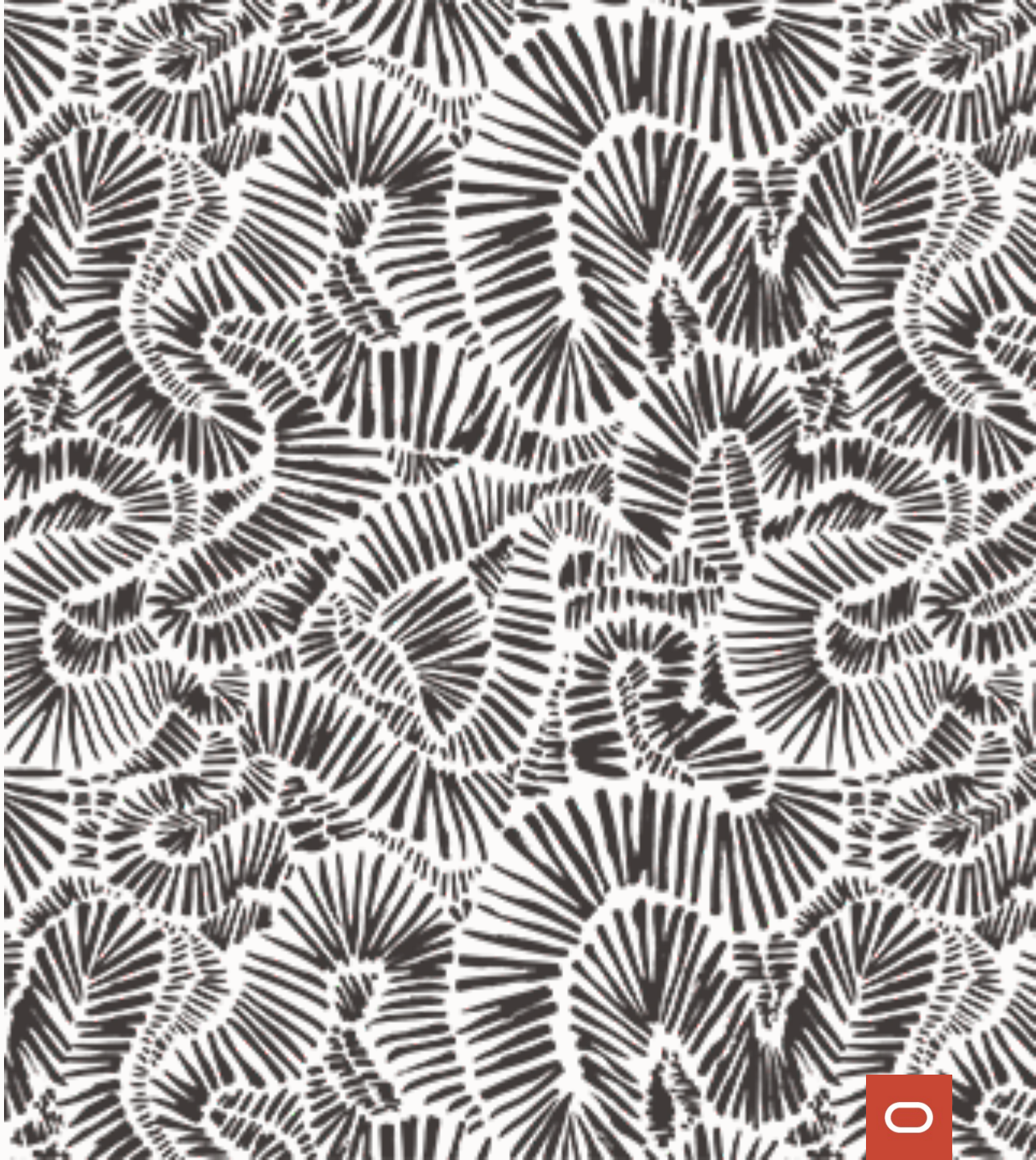
```
scp myfile operator:/tmp
```

Hiermit wird die loakle Datei „myfile“ auf den Operator-Host in das Verzeichnis „/tmp“ kopiert. Die andere Richtung:

```
scp operator:/tmp/myfile .
```

Nutzung

- ssh / scp
- putty



PuTTy

Unter Windows kann man entweder mit Git Bash (https://gitforwindows.org/) wie unter Linux arbeiten oder man nutzt PuTTy (https://www.putty.org/) um sich mit der Ziel-Instanz zu verbinden.

Einen schnellen Einstieg zu ssh / scp und PuTTy / WinSCP findet man hier:
https://standby.cloud/download/pdf/SSH.pdf

Um einen ProxyJump über den Bastion Host zu machen, konfigurieren Sie PuTTy z.B. mit der privaten IP des Operators (und entsprechendem Key) und tragen Sie unter Connection->Proxy diese Werte ein:

Proxy type: Local
Proxy hostname: <PublicIP-Bastion>
Port: 22
Username: <User>
Telnet command: plink.exe %user@%proxyhost -nc %host:%port -i <PathToKey>/<KeyName>.ppk

PuTTy (ProxyJump)

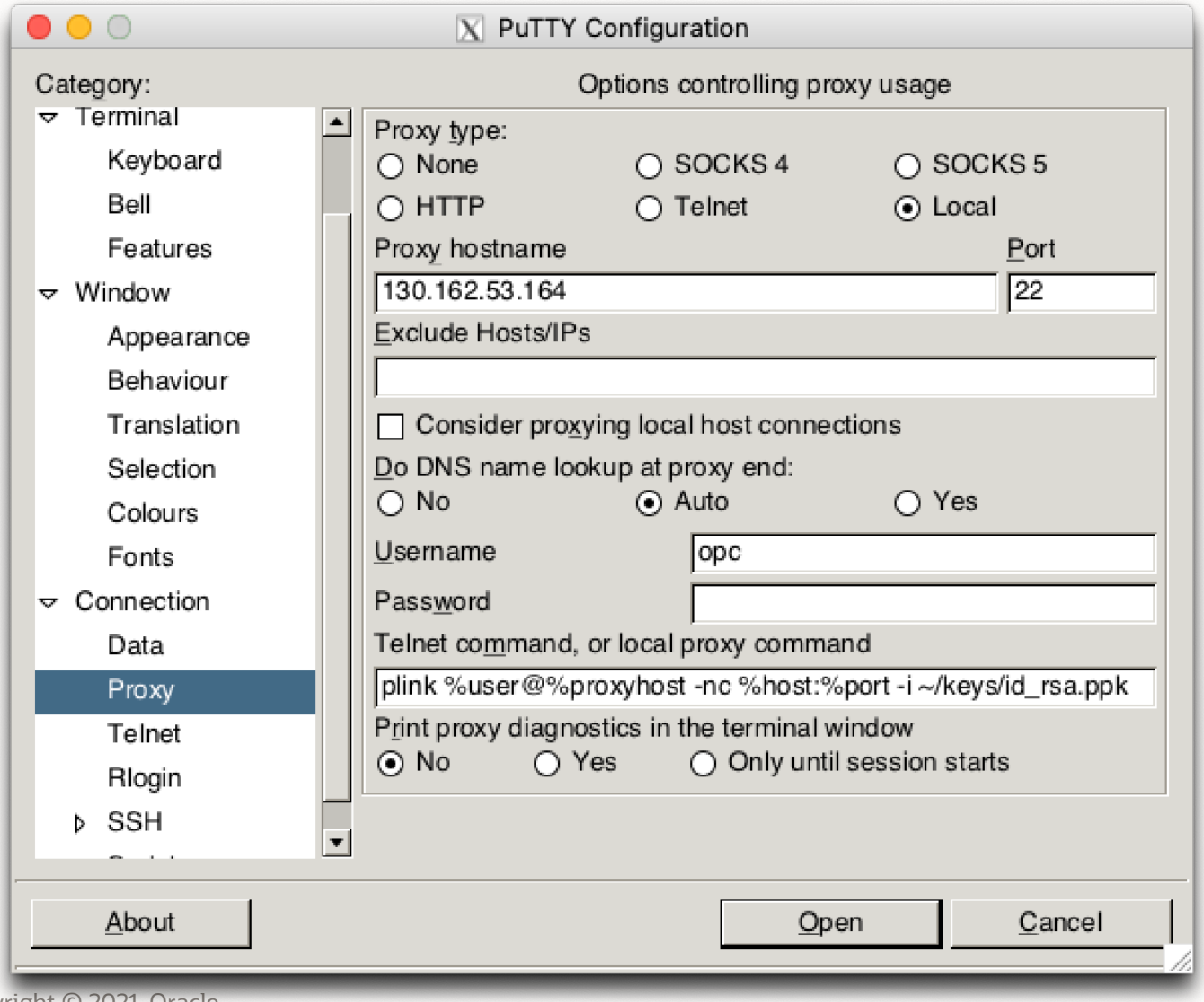


Bild zeigt Demo Proxy-Einstellungen von PuTTy auf MaxOS.

Georg Völl

Principal Account Cloud Engineer (ACE)

Oracle Technology Cloud Engineering (TCE)

eMail: georg.voell@oracle.com