Zwei-Faktor-Athentifizierung mit Yubikey-Token

Ein kostengünstiges Verfahren

Passwörter alleine sind zu unsicher um damit Benutzer zu Authentifizieren. Yubikey-Token ermöglichen ein vergleichsweise kostengünstiges Zwei-Faktor-Verfahren für verschiedene Unix-Dienste sowie eine Smartcard-Lösung für GnuPG-Schlüssel.

von **Stefan Schumacher**

Passwörter sind wie Unterwäsche. Du darfst sie keinen sehen lassen, musst sie regelmäßig wechseln und solltest sie nicht mit Fremden tauschen.
Chris Pirillo

Die Zugänge zu Computersystemen sollten vor unbefugten Benutzern geschützt werden. Dies erfolgt in der Regel über ein Verfahren der Authentifizierung. Authentifizierung bezeichnet hierbei den Nachweis einer bestimmten Eigenschaft. Dies ist in der Regel ein Passwort, dass zu einem bestimmten Benutzernamen gehört und beim Login eingegeben werden muss. Andere Authentifizierungsverfahren sind zum Beispiel Einmalpasswörter oder Transaktionsnummern sowie Ausweise, wo das aufgedruckte Foto zur jeweiligen Person passen muss oder Anruf-Parole-Verfahren, wie man sie aus Agentenfilmen der 80er Jahre kennt.

Abzugrenzen von der Authentifizierung ist die Autorisierung. Damit werden einer Person oder auch Maschine bestimmte Rechte eingeräumt. Auf Unix-Systemen wird dies von Anfang an für Dateien über die Kombination von Eigentümer und Gruppe sowie den darauf basierenden Dateirechten implementiert. Ähnliche Verfahren, nur wesentlich komplexer, sind die Access Control Lists (ACL), die unter anderem mehrere Gruppen mit jeweils eigenen Rechten für Dateien zulassen.

Zwei-Faktor-Authentifizierung

Authentifizierungsverfahren nutzen einen oder mehrere von drei grundlegenden Faktoren:

- **Wissen** ist dabei eine geheimgehaltene Kenntnis, zum Beispiel ein Passwort, eine PIN oder eine TAN.
- **Besitz** ist etwas, dass der Benutzer hat. Dabei kann es sich um einen Türschlüssel, eine EC-Karte, ein USB-Token oder eine Schlüsseldatei handeln.
- **Sein** ist hier eine körperliche Eigenschaft des Benutzers wie der Fingerabdruck, Irismuster, Stimme oder das Gesicht, welches auf aktuellen Smartphones als Passwort verwendet wird.

Diese drei Faktoren lassen sich auch beliebig kombinieren, so kann der Zutritt zu einem Server mit einer Sicherheitstür gesichert sein, die einen Schlüssel benötigt, den nur die Administratoren und Geschäftsführer haben. Zusätzlich ist dann zum eigentlichen Login noch ein Benutzername, Passwort, Fingerabdruckscan sowie eine Smartcard notwendig.

Der größte Nachteil der reinen Benutzername-Passwort-Authentisierung ist die Unsicherheit des Passworts. Kennt ein Angreifer das Passwort, kann er sich mit diesem Wissen im System einloggen und hat somit sofort alle Zugriffsrechte des kompromittierten Kontos zur Verfügung. Dies ist insbesondere problematisch, da viele Benutzer keinen Wert auf sichere Passwörter legen oder gar nicht wissen was ein sicheres Passwort überhaupt ausmacht.

Listen von geknackten Passwörtern zeigen, dass ein großer Teil der Benutzer überhaupt keinen Wert auf sichere Passwörter legt. So lauten die häufigsten Passwörter *123456*, *12345678*, *hallo*, *passwort*, *secret* oder schlicht `0000`. Durch die Verfügbarkeit extrem schneller Grafikkarten oder günstiger Rechenzeit von Cloud-Anbietern wächst die Verwundbarkeit derartiger Authentifizierungsverfahren steil an. Bei vielen Appliances werden auch die Default-Passwörter nicht geändert.

Deshalb ist es sinnvoll, neben dem Faktor Wissen (Passwort) auch den Faktor Besitz (USB-Token) in das Authentifizierungsverfahren zu integrieren. Der Vorteil hierbei ist, dass ein Angrei-

fer neben dem Passwort des Benutzers auch Zugriff auf das Token haben muss, wenn er sich erfolgreich einloggen will. Das kann zwar nicht jeden Angriff verhindern, aber die dafür notwendigen Ressourcen massiv erhöhen. Ein Token kann vor schwachen Passwörtern, vor dem Ausspähen des Passworts bei der Eingabe („Shouldersurfing-Attack“) und vor Man-in-the-Middle-Angriffen schützen.

Der Einsatz eines Tokens hat aber auch Nachteile. Passwörter lassen sich einfach ändern, zurücksetzen oder ausrollen. Für Tokens sind die Beschaffungskosten relevant. Diese können je nach Hersteller und eingesetzter Technik sowie Lizenzierungsmodell bei dreistelligen Beträgen je Benutzer liegen. Darüber hinaus muss eine Verwaltungsinfrastruktur aufgebaut werden. Die Token müssen individualisiert und an die Benutzer übergeben werden. Schulungen können notwendig werden und es muss ein Procedere für den Verlust oder Ausfall eines Tokens eingerichtet werden. Vergisst jemand sein Token zu Hause, kann er sich nicht mehr mit dem Firmen-VPN verbinden.

One Time Password

Ein „One Time Password“ (OTP) kann nur genau ein einziges Mal verwendet werden. Jeder Authentifizierungsvorgang verlangt ein neues OTP, egal ob der Vorgang erfolgreich abgeschlossen wurde oder nicht. Dieses Einmal-Passwort ist sicher gegen passive Angriffe wie Mitschneiden und gegen Replay-Attacken, nicht jedoch gegen eine Man-in-the-Middle-Attacke, bei der die Verbindung zum Zielsystem auf ein manipuliertes System umgeleitet wird.

Die technische Herausforderung hierbei ist, dass beide beteiligten Systeme das entsprechende OTP kennen müssen. Dfür gibt es mehrere Verfahren:

- Das älteste und am einfachsten zu implementierende Verfahren ist eine Liste mit Einmal-Passwörtern, die auf beiden Systemen hinterlegt bzw. dem Anwender zugestellt wird. Bei jedem Login wird chronologisch oder zufällig ein OTP abgefragt. Dieses Verfahren wird z. B. beim Online-Banking als TAN-Liste eingesetzt und ist relativ kostengünstig realisierbar. Nachteilig sind zum einen die Übermittlung der Passwortliste an den Anwender, die über einen sicheren Kanal erfolgen muss und zum anderen der mögliche Verlust der Liste.

- Sicherer, aber auch komplexer sind Passwortgeneratoren. Hier wird auf beiden Systemen ein OTP generiert und das Ergebnis einer Berechnung als Beweis der Identität übertragen. Dafür gibt es drei mögliche Verfahren:

  – **Zeitgesteuerte Generatoren** erzeugen in einem definierten Zeitintervall von ein bis fünfzehn Minuten Einmal-Passwörter mit demselben Algorithmus und gegebenenfalls Seed auf dem Server und dem Token. So besitzen beide zu einem bestimmten Zeitpunkt das gleiche OTP. Problematisch ist hierbei, dass das Token und der Server synchron laufen müssen, was sich durch Netzwerkzeitprotokolle wie NTP oder durch Funkuhren wie DCF77 erreichen lässt. Das Token benötigt dafür eine Energieversorgung, was den Preis erhöht und unter Umständen zu einer Begrenzung der Lebensdauer oder Erhöhung der Komplexität für den Anwender führt. Im RFC 6238 (TOTP: Time-Based One-Time Password Algorithm) wurde ein entsprechender Algorithmus definiert und publiziert.

  – **Ereignisgesteuerte Generatoren** funktionieren im Prinzip genauso. Sie generieren ein neues OTP jedoch nicht nach einer definierten Zeitspanne, sondern aufgrund eines bestimmten Ereignisses. Dies ist meistens ein Authentifizierungsvorgang. Das aktuell berechnete Passwort bleibt daher bis zum nächsten Authentifizierungsvorgang gültig. Bei diesem Verfahren werden die Batterien im Token geschont und das Token kann auch für eine längere Zeit abgeschaltet werden. Es muss dazu lediglich der letzte verwendete Datensatz

gespeichert werden. Wird ein Token an mehreren Servern genutzt, müssen die Server im Falle einer Authentisierung synchronisiert werden, um zu erfahren, dass das aktuelle OTP veraltet ist.

– Bei **Challenge-Response-gesteuerten Generatoren** gibt der Server eine Aufgabe vor (Challenge), die das Token beantworten muss (Response). Es wird also ein Initialwert auf das Token übertragen und basierend darauf ein OTP berechnet. Der größte Vorteil ist hierbei, dass sich Token und Server nicht gegenseitig synchronisieren müssen. Dies reduziert den Aufwand und das Token kann zwischendurch ausgeschaltet bleiben. Auch für dieses Verfahren liegt mit RFC 4226 (HOTP: An HMAC-Based One-Time Password Algorithm) eine Standardisierung vor. Es wird ein Keyed-Hash Message Authentication Code (HMAC) verwendet – eine kryptographische Hash-Funktion, die über einer Nachricht und einem geheimen Schlüssel abgeleitet wird.

Standards und Betriebsmodi von Token

Der Einsatz von Token zur Zwei-Faktor-Athentifizierung ist keine neue Entwicklung. Bereits in den 1980er Jahren setzte unter anderem das US-Militär Token bzw. sogenannte Dongle zur Authentifizierung ein. Diese Verfahren waren aber in der Regel proprietär und an einen Hersteller gebunden, der oftmals eine jährliche Lizenzgebühr je Benutzer erhebt. Um diesen Einschränkungen entgegenzuwirken, haben sich zwei Organisationen gegründet. Die „Initiative For Open Authentication“ (OATH) und die nicht kommerzielle Allianz „Fast Identity Online“ (FIDO).

Die OATH ist ein branchenübergreifender Zusammenschluss von Unternehmen, um eine offene Referenzarchitektur für sichere Authentifizierungsverfahren zu entwickeln. Dabei sollen offene Standards verwendet werden, um Kosten zu sparen und die Nutzung zu vereinfachen. Dazu hat sie unter anderem mehrere RFCs veröffentlicht, die zum Beispiel Algorithmen für Einmal-Passwörter definieren.

Die FIDO-Allianz wurde 2013 gegründet, um offene und lizenzfreie Industriestandards für die weltweite sichere und schnelle Authentifizierung im Internet zu entwickeln. Inzwischen sind der Allianz viele große Unternehmen wie Google, Microsoft, Samsung oder Lenovo beigetreten. Auch das Bundesamt für Sicherheit in der Informationstechnik ist seit Oktober 2015 Mitglied. FIDO hat bereits zwei Standards entwickelt: U2F (Universal Second Factor) und UAF (Universal Authentication Framework) und dazu die entsprechenden Spezifikationsrahmen für Biometrie, Trusted Platform Modules, Smartcards und NFC.

Challenge-Response bei Yubikey

Token und Server einigen sich durch einen Konfigurationsprozess auf ein geteiltes Geheimnis. Als Nachricht dient der interne Zählerstand der erfolgten Authentifizierungsversuche. Beim Authentifizierungsvorgang generiert das Token dann basierend auf dem geteilten Geheimnis als Schlüssel und dem Zählerstand als Nachricht mit SHA1 einen HMAC, der übertragen wird. Der Server generiert ebenfalls diesen HMAC und vergleicht den erhaltenen mit dem selbst berechneten Wert. Sind beide Werte gleich, ist die Authentisierung erfolgreich. SHA1 ist in diesem Verfahren noch als sicher zu betrachten, da durch den geheimen Schlüssel die Möglichkeit, Kollisionen zu generieren, nicht so einfach ist, wie bei einem einfachen SHA1-Hash.

Das TOTP-Verfahren funktioniert ähnlich, nur dass statt des Zählers als Nachricht ein Zeitwert verwendet wird. Hierzu wird intern die Unix-Epoche (Sekunden seit 01.01.1970) verwendet und durch ein definiertes Gültigkeitsintervall (z. B. fünf Minuten) geteilt. Dazu müssen der Athentifizierungsserver und das Token über eine interne Uhr verfügen. Dadurch werden Token teurer in der Herstellung, da sie neben der Uhr auch über eine Stromversorgung verfügen müssen. Da die Yubikey-Token über keine eingebaute Uhr verfügen, ist ein externes Programm (Yubico Authenticator) bzw. eine App auf dem Smartphone notwendig, um TOTP zu nutzen.

Weiter unterstützen Yubikey-Token auch den Challenge-Response-Modus via Human Interface Device (HID). Das Token verhält sich dabei wie eine USB-Tastatur und antwortet selbständig ohne Benutzerinteraktion auf Challenge-Anfragen. Dieser Modus eignet sich zum Beispiel besonders gut, wenn das Token als Kopierschutzmechanismus genutzt werden soll oder sehr häufige Abfragen bei Login-Prozessen erfolgen sollen.

Yubico bietet außerdem ein eigenes Challenge-Response-Verfahren namens Yubico OTP an. Dazu

verfügen die Token bereits ab Werk über einen eigenen geheimen Schlüssel in der Hardware. Dieser wird genutzt, um sich gegenüber den Yubikey-Servern, der sogenannten YubiCloud, zu authentifizieren. Die Yubicloud stellt dazu eine API bereit, um eigene Anwendungen wie OpenSSH oder OpenVPN mit ihr zu verbinden.

Ein weiterer Challenge-Response-Modus ist HMAC-SHA1. Hier wird aus einer Nachricht und einem geheimen Schlüssel ein Hash-Wert berechnet. Ein von Yubico bereitgestelltes PAM-Modul kann hier das Challenge-Response-Verfahren mit einem Yubikey offline durchführen. Mit dieser Variante kann man alle PAM-fähigen Authentifizierungsverfahren *offline* mit einem Yubikey verheiraten, beispielsweise den Anmeldeprozess auf einem Linux- oder OSX-Rechner, den `su(1)`-Befehl oder den `xscreensaver`-Bildschirmschoner. Auch mit LUKS verschlüsselte Festplatten lassen sich damit zusätzlich absichern.

In diesem Modus muss der Server den geheimen Schlüssel des HMAC-SHA1-Algorithmus gar nicht kennen. Ein Yubikey kann diesem Modus mit dem Programm `ykpamcfg` verwendet werden. `ykpamcfg` sendet eine Challenge an den Yubikey und fängt die entsprechende Response ab. Beides wird dann in einer Datei gespeichert. Beim ersten Anmeldeversuch wird die Challenge aus der Datei an den Yubikey gesendet und die Response mit der gespeicherten Antwort verglichen. Stimmt beides überein, kennt der Yubikey den richtigen privaten Schlüssel und ist damit authentifiziert. Für das nächste Login wird erneut eine Challenge gesendet und die Response gespeichert. Bei jedem Login-Prozess muss sich der Yubikey also mit dem gespeicherten Challenge-Response-Paar erneut authentifizieren und ein neues Paar für das nächste Login generieren, welches in einer Datei gespeichert wird.

Universal Second Factor (U2F) wurde von der FIDO-Allianz entwickelt und seit 2014 ausgerollt. Konzipiert wurde es für den einfachen Einsatz bei Webseiten und im Webbrowser. Bei U2F handelt es sich um ein erweitertes Challenge-Response-Verfahren mit asymmetrischer Kryptographie. Der geheime Schlüssel verbleibt dabei in der Hardware des Tokens und kann dieses nicht verlassen, es wird nur der öffentliche Schlüssel publiziert.

Für jeden Webdienst wird dabei ein eigenes Schlüsselpaar abgeleitet, wobei die TLS-Connection-ID und die URL des Webdienstes, sowie eine zufällig generierte Sitzungsnummer bei der Erzeugung des Schlüsselpaares mit einbezogen werden, um Phishing und Man-in-the-Middle-Attacken zu erschweren. Damit ist es außerdem möglich, ein Token an verschiedenen Webdiensten zu nutzen, ohne dass die Betreiber der Webdienste erkennen können, dass für verschiedene Benutzer oder Dienste das selbe Token verwendet wird.

Betriebsmodi des Yubikey

Außer dem einfachen U2F-Yubikey kommen alle Yubikeys mit zwei verfügbaren Slots, die verschieden konfiguriert werden können. Neben dem U2F-Modus kann ein Slot auch im OTP-Modus oder CCID-Modus (Chip Card Interface Device) betrieben werden. Der OTP-Modus nutzt einen der oben beschriebenen OTP-Standards. Der CCID-Modus ermöglicht es, Smartcards per USB mit dem PC zu verbinden. In diesem Modus kann man beispielsweise GnuPG-Schlüssel auf dem Yubikey speichern.

Mittels `ykpersonalize` kann man den gewünschten Modus über einen Zahlencode auswählen. Mit `ykpersonalize -m6` wird der Yubikey in den OTP/U2F/CCID-Modus versetzt, so dass er mit allen in diesem Artikel beschriebenen Konfigurationen funktioniert.

Abb. 1 zeigt vier verschiedene Yubikeys, davon sind zwei normal große Modelle und zwei Mini-Modelle, die im USB-Schacht stecken bleiben können.

Abbildung 1: Verschiedene Yubikey-Ausführungen (Quelle: Yubico)

Webseiten mit nativer U2F-Unterstützung

Webseiten die U2F nativ unterstützen, sind auf der Webseite https://www.dongleauth.info/ gelistet. Zu den größeren Unterstützern gehören unter anderem Google mit allen Diensten, Amazon Web Services, Microsoft Azure, Amazon, Mailbox.org oder Facebook.

Yubikeys werden nativ von den Webbrowsern Google Chrome sowie Chromium, Iridium und Opera unterstützt. Der Firefox unterstützt ab Version 60 FIDO U2F zumindest experimentell, erfordert dazu aber noch etwas Handarbeit in der Konfiguration. Man muss die Seite `about:config` aufrufen und den Wert `security.webauth.u2f` aktivieren. Unter https://demo.yubico.com/u2f kann man dann mit einem Yubikey prüfen, ob der eigene Browser U2F unterstützt.

Die Einrichtung eines Yubikeys ist dann vergleichsweise problemlos, auf den unterstützten Webseiten findet man in der Regel unter den Konto-Optionen einen Punkt zur Zwei-Faktor-Athentifizierung. Dort kann man den U2F-Yubikey mit dem eigenen Konto verbinden, so dass man ihn bei jedem Login einstecken muss. Meistens verlangen die Webseiten aus Sicherheitsgründen die Hinterlegung einer Handy-Nummer oder bieten eine TAN-Liste an, um sich trotzdem noch ohne Yubikey einloggen zu können. Dies ist sinnvoll, da man bei Verlust oder Defekt des Yubikeys nicht mehr auf das Konto zugreifen kann.

Webseiten ohne U2F-Unterstützung via Passwort-Manager

Über einen Umweg lassen sich auch Webseiten ohne U2F-Unterstützung mit einem Token absichern. Dazu muss lediglich der Passwortmanager KeePass ab Version 2.0 mit dem Plugin `OtpKeyProv` von Dominik Reichl zwischengeschaltet werden. Damit lässt sich die KeePass-Datenbank mit einem Passwort und zusätzlich mit einem Yubikey im OATH-HOTP-Modus absichern.

Durch ein Plugin können Webbrowser wie Firefox, Chrome und Safari direkt auf die KeePass-Datenbank zugreifen und so Benutzernamen und Passwörter aus der geschützten Datenbank holen, anstatt ihre eigenen unsicheren Passwortspeicher zu verwenden.

Hat man KeePass mit dem eingesetzten Browser und dem Yubikey verheiratet, kann man sich in die Webseiten die kein U2F unterstützen einloggen und dort ein neues Passwort vergeben. Da man sich dieses neue Passwort dank KeePass nicht merken muss, nutzt man den Passwort-Generator von KeePass und erzeugt ein 256-Bit-Hex-Passwort. Dies ist ein String, der aus 64 hexadezimalen Ziffern besteht. Diese Zeichenkette wird dann in der Webseite als neues Passwort hinterlegt. Dieses zufällig generierte Passwort ist in der Regel wesentlich sicherer als ein vom Menschen ausgedachtes. Möchte man sich nun auf der Webseite einloggen, benötigt man das KeePass-Master-Passwort sowie den Yubikey für das Einmal-Passwort um die KeePass-Datenbank freizuschalten.

GnuPG-Schlüssel auf dem Yubikey

Der Yubikey Neo und alle Modelle der 4er-Reihe unterstützen OpenPGP-Schlüssel über eine OpenPGP-Smartcard-Funktion. Das heißt, dass das Schlüsselpaar direkt vom Yubikey selbst in der Hardware generiert und in der Smartcard gespeichert wird. Der öffentliche Schlüssel kann auch aus der Smartcard exportiert werden, um ihn auf einen Keyserver hochzuladen. Der private OpenPGP-Schlüssel hingegen lässt sich nicht ohne Eingabe der PIN aus dem Yubikey auslesen. Damit besteht auch an einem nicht vertrauenswürdigen Rechner die Möglichkeit, Dateien oder Mails mittel OpenPGP zu verschlüsseln, ohne deinen eigenen privaten Schlüssel als Datei auf das System kopieren zu müssen. Die Verschlüsselung der Datei oder Mail wird direkt vom Yubikey durchgeführt und dem System dann verschlüsselt zur Verfügung gestellt.

Zuerst muss ein GnuPG-Hauptschlüssel mit Unterschlüssel generiert werden. Um den privaten GnuPG-Schlüssel insbesondere auf mobilen Rechnern zu schützen, bietet sich zur Schlüssel-Erzeugung der Reinraum-Ansatz mit Unterschlüsseln an. Dazu wird auf einem nackten System ohne Netzwerkzugang ein Schlüsselpaar samt Unterschlüsseln generiert. Diese Unterschlüssel können dann z. B. auf dem Laptop, einer Smartcard oder dem Yubikey genutzt werden. Auf Keysigning-Parties wird der Hauptschlüssel von den anderen Teilnehmern signiert und steigt so im Web of Trust auf. Sollte der Laptop mit den Unterschlüsseln gestohlen werden, muss man nur diese Unterschlüssel widerrufen und kann ein neues Paar für den Ersatz-Laptop generieren. Diese erben dann automatisch alle anderen Signaturen des Hauptschlüssels.

Als Reinraum-Rechner zur Erzeugung der Schlüssel kann man entweder einen dedizierten Rechner mit einer einfachen Linux-Distribution nutzen oder man setzt ein Live-System wie TAILS von einem USB-Stick ein. Als Hardware reicht ein älteres oder einfaches System wie ein Netbook oder auch ein Raspberry Pi vollkommen aus.

Zuerst erstellen wir mit `gpg2 --expert --full-gen-key` einen neuen Schlüsselsatz und wählen als erste Option `(8) RSA (set your`

own capabilities), um dann mit (E) für Encryption die Verschlüsselungsfähigkeit zu deaktivieren. Damit ist der Hauptschlüssel nur noch in der Lage, zu signieren und zu zertifizieren. Mit (Q) beenden wir den Dialog und setzen die Schlüssellänge auf 4096 Bit. Als weitere Optionen müssen noch die Benutzer-ID, ein Verfallsdatum sowie ein Kommentar und die Passphrase nach eigenem Ermessen gesetzt werden.

Im nächsten Schritt werden Unterschlüssel generiert, dazu ruft man gpg2 --expert --edit-key mit der ID des Hauptschlüssels auf. Mittels addkey können beliebig viele Unterschlüssel generiert werden. Mit der Option (6) RSA (encrypt only) erzeugen wir einen Unterschlüssel zum verschlüsseln. Dazu können wir wieder die Bitlänge auf 4096 und auf Wunsch ein Verfallsdatum setzen.

Danach wird erneut addkey aufgerufen, um einen Signatur-Schlüssel zu erzeugen. Mittels (8) RSA (set your own capabilities) kann man mit der Option E für Encrypt die Verschlüsselungsfähigkeit deaktivieren und erhält so einen Unterschlüssel, der nur noch signieren kann.

OpenPGP-Schlüssel können auch zur Athentifizierung, beispielsweise mit OpenSSH, genutzt werden. Dazu erstellt man mit addkey einen weiteren Unterschlüssel, diesmal wieder mit der Option (8) RSA (set your own capabilities). Anschließend wird mit S und E die Signier- und Verschlüsselungsfähigkeit deaktiviert und mit A die Authentifizierungsfähigkeit aktiviert. Das Schlüsselpaar sieht nun beispielsweise folgendermaßen aus:

```
[Anton@x201 ~]# gpg2 {-}-list-secret-keys
/Anton/.gnupg/pubring.kbx
------------------------
sec  rsa2048 2018-06-13 [SC] [expires: 2018-07-11]
     1CB9CD288E41E4E323275ECD10A84A0A43BF4FB2
uid          [ultimate] Anton Gorodezki (Yubikey
             Testschluessel) <nachtwache@moscow.ru>
ssb  rsa2048 2018-06-13 [E] [expires: 2018-07-11]
ssb  rsa2048 2018-06-13 [S] [expires: 2018-07-11]
ssb  rsa2048 2018-06-13 [A] [expires: 2018-07-11]
```

Mittels adduid können wir weitere Benutzer-IDs mit jeweils neuen E-Mail-Adressen erstellen. Somit ist es möglich, für jede vorhandene E-Mail-Adresse weitere Unterschlüssel zu erzeugen.

Um die Schlüssel zu sichern, gibt es drei Export-Optionen die jeweils die privaten Schlüssel, alle privaten Unterschlüssel sowie die öffentlichen Schlüssel exportieren, wie in folgendem Listing gezeigt. Mit dem vierten Befehl generiert man ein Widerruf-Zertifikat, für den Fall, dass das Schlüsselpaar kompromittiert wurde. Die so entstandenen Dateien kann man offline an einem sicheren Ort aufbewahren. Alternativ kann man die öffentlichen, sowie die Unterschlüssel auch auf den Laptop übertragen, auf dem sie eingesetzt werden sollen. Dazu werden die beiden Dateien Unter.sec und Oeffentliche.asc übertragen und mit gpg --import importiert.

```
gpg2 --armor --output Private.sec \
     --export-secret-key <KeyID>
gpg2 --armor --output Unter.sec \
     --export-secret-subkeys <KeyID>
gpg2 --armor --output Oeffentliche.asc \
     --export <KeyID>
gpg2 --output Widerruf.asc --gen-revoke <KeyID>
```

Nun muss der Master-Schlüssel aus dem Schlüsselring entfernt werden, um zu verhindern, dass dieser mit den Unterschlüsseln zusammen exportiert wird. Dies geht am einfachsten, indem man mit gpg2 --delete-secret-key <KeyID> den privaten Schlüssel komplett löscht und mit gpg2 --import Oeffentliche.asc Unter.sec die öffentlichen Schlüssel sowie die Unterschlüssel wieder importiert. Lässt man sich mit gpg2 -list-secret-keys die privaten Schlüssel anzeigen, erscheint nun in der ersten Zeile hinter dem sec des Master-Schlüssels eine Raute #. Dies indiziert, dass der entsprechende Master-Schlüssel *nicht* im Schlüsselring vorhanden ist und somit auch nicht kompromittiert werden kann.

Nun wird der GnuPG-Schlüssel mittels GnuPG (gpg) auf den Yubikey transferiert. Der Yubikey Neo unterstützt dabei jedoch nur RSA-Schlüssel mit 2048 Bit Länge. Dies ist der NFC-Funktionialität geschuldet. Der Yubikey Neo kann über NFC auch mit Smartphones und Tablets kommunizieren. Allerdings limitiert der hohe Energiebedarf den Einsatz von RSA-Schlüsseln auf 2048 Bit Schlüssellänge. Yubikeys der Reihe 4 unterstützen hingegen 4096 Bit lange RSA-Schlüssel, benötigen dafür aber zur Konfiguration GnuPG Version 2 (gpg2).

Das Kommando gpg2 --card-edit startet GnuPG mit der Verwaltungsoberfläche für Smartcards und zeigt die aktuell gesetzten Daten wie Hersteller, ID und Fingerprints der Schlüssel an. Die Smartcard wird mit zwei PIN-Nummern geschützt. Erstens die normale PIN-Nummer, um auf die OpenPGP-Schlüssel zuzugreifen und eine zweite PIN, welche die Adminfunktionalität schützt. Die erste PIN ist im Auslieferungszustand 123456, die Admin-PIN ist 12345678. Über die Befehle admin aktiviert man den Admin-Modus und mit passwd ändert man dann die beiden PINs. Anschließend wählt man mit der folgenden Befehlsfolge den entsprechenden Schlüssel

aus und kopiert ihn mit `keytocard` auf den Yubikey.

```
gpg2 {-}-expert {-}-edit-key <KeyID>
gpg> key 1
gpg> keytocard
gpg> key 1
gpg> key 2
gpg> keytocard
gpg> key 2
gpg> key 3
gpg> keytocard
gpg> quit
Save changes? (y/N) y
```

Mittels `key 1` wird dabei der entsprechende Unterschlüssel ausgewählt und nach dem Export wieder abgewählt. GPG fragt nach, welchen Slot es benutzen soll. In der Regel wird der jeweils passende für Verschlüsselung und Signaturen bereits vorausgewählt. Außerdem muss man vor dem Export auf die Smartcard die Passphrase für den Hauptschlüssel und die Admin-PIN des Yubikey angeben.

Die beiden Unterschlüssel zum Verschlüsseln und Signieren befinden sich nun auf dem Yubikey und können von dort aus benutzt werden. Auf dem Rechner wurden die Unterschlüssel durch den `keytocard`-Befehl aus dem Schlüsselring entfernt und durch einen sogenannten *Stub* ersetzt. Dies ist quasi eine Art Symlink im Schlüsselring, damit GnuPG die Schlüssel nicht direkt im Schlüsselring sucht, sondern auf der Smartcard bzw. dem Yubikey.

Soll der Yubikey auf anderen Rechnern ebenfalls mit den OpenPGP-Schlüsseln eingesetzt werden, ist es notwendig, dort die öffentlichen Schlüssel und die Stubs zu importieren. Dazu müssen die Stubs zuerst auf dem Reinraum-Rechner exportiert werden. Dies geht mit dem Befehl .

```
gpg2 --armor --output Stubs.sec \
     --export-secret-subkeys <KeyID>
```

Die beiden Dateien `Oeffentliche.asc` und `Stubs.sec` kopiert man dann auf den Zielrechner und importiert sie mit dem Befehl `gpg2 --import Oeffentliche.asc Stubs.sec`. Versucht man nun diese Schlüssel z. B. zum Signieren einer Datei zu benutzen, erhält man die Fehlermeldung, dass der private Schlüssel nicht vorhanden ist. Denn es ist noch notwendig, den Yubikey für GnuPG zu initialisieren. Dies geschieht mit `gpg2 -card-status`. GnuPG fragt nun beim Signieren bzw. Verschlüsseln nach der 6-stelligen Yubikey-PIN. Mit `gpg2 --card-status` kann man nun den Zustand der Smartcard anzeigen lassen, wie in Abb. 2 gezeigt:

Programme und Login-Prozesse mittels PAM ansteuern

PAM (Pluggable Authentication Modules) ist eine Schnittstelle mit der es möglich ist, Benutzer über anpassbare Module zu authentifizieren. Yubico bietet ein eigenes Open-Source-PAM-Modul zur Nutzung mit den Yubikeys an. Mit diesem PAM-Modul kann man alle Logins die PAM unterstützen, mit dem Yubikey verheiraten, ein normales Login an der Konsole ebenso wie die Passworteingabe an einem gesperrten Bildschirmschoner oder die Festplattenverschlüsselung mit LUKS.

Bei den meisten Distributionen ist das Yubico-PAM-Modul bereits im Paketmanager vorhanden, z. B. als `libpam-yubico` in Debian/Ubuntu oder als `yubico-pam` in Arch Linux. Es geht aber auch ohne Paketmanager, die manuelle Installation ist unter https://developers.yubico.com/yubico-pam/ beschrieben und recht einfach nachzuvollziehen.

Nach der Installation kann das Yubico-PAM-Modul in der PAM-Konfigurationsdatei beliebiger Dienste eingebunden werden. Es unterstützt verschiedene Betriebsmodi wie die Online-Authentifizierung über die YubiCloud oder eine Offline-Authentifizierung über eine lokale Challenge-Response-Abfrage.

```
gpg2 --card-status

Reader ...........: 1050:0116:X:0
Application ID ...: D2760006380102000006027145080000
Version ..........: 2.0
Manufacturer .....: Yubico
Serial number ....: 04754119
Name of cardholder: Anton Sergejewitsch Gorodezki
Language prefs ...: [not set]
Sex ..............: unspecified
URL of public key : http://www.moscow.ru/F43274A5C81AE64F.asc
Login data .......: [not set]
Signature PIN ....: forced
Key attributes ...: rsa2048 rsa2048 rsa2048
Max. PIN lengths .: 127 127 127
PIN retry counter : 3 3 3
Signature counter : 4
Signature key ....: 0C2E B278 ED10 469F E5A7  BC49 F432 74A5 C81A E64F
      created ....: 2018-06-13 16:54:21
Encryption key....: B0AC CF5A 0820 A1C6 F40F  32A0 E153 096D 7B0B 7C45
      created ....: 2018-06-13 16:53:46
Authentication key: DD45 4E87 2CA3 D6A7 257C  B0F4 CB77 D9BA 4B41 8153
      created ....: 2018-06-19 09:11:01
General key info..: sub rsa2048/F43274A5C81AE64F 2018-06-13 Anton Gorodezki
                          (Yubikey Testschluessel) <nachtwache@moscow.ru>
sec   rsa2048/10A84A0A43BF4FB2  created: 2018-06-13  expires: 2018-07-11
ssb>  rsa2048/E153096D7B0B7C45  created: 2018-06-13  expires: 2018-07-11
                                card-no: 0006 03814508
ssb>  rsa2048/F43274A5C81AE64F  created: 2018-06-13  expires: 2018-07-11
                                card-no: 0006 03814508
ssb>  rsa2048/CB77D9BA4B418153  created: 2018-06-13  expires: 2018-07-11
                                card-no: 0006 03814508
```

Abbildung 2: Ausgabe des Kommandos `gpg2 --card-status`

Auch die YubiCloud lässt sich relativ einfach einbinden und kostenfrei nutzen, allerdings funktioniert ein Login dann nur, wenn die YubiCloud auch erreichbar ist. Befindet man sich in einem fremden Netzwerk könnte der Zugriff auf die YubiCloud-Server möglicherweise geblockt werden. Damit ist dann ein Login nicht mehr möglich, so lange man keine Fallback-Lösung hat. Neben

der YubiCloud kann man die benötigten Server-Dienste auch lokal betreiben wofür Yubico zwei PHP-Module anbietet, die über einen Webserver wie Apache ins Internet gebracht werden können.

Die zu verwendenden Yubikeys müssen auf der Webseite https://upgrade.yubico.com/getapikey/ für die YubiCloud registriert werden. Der erzeugte API-Key muss dann dem Yubico-PAM-Modul als Option übergeben werden, kann aber auch zentral im `authfile` für alle Nutzer hinterlegt werden. Eine Alternative zur Authentifizierung gegenüber einem Server ist der Challenge-Response-Modus mit dem oben bereits beschriebenen HMAC-SHA1-Algorithmus, welcher auch offline funktioniert.

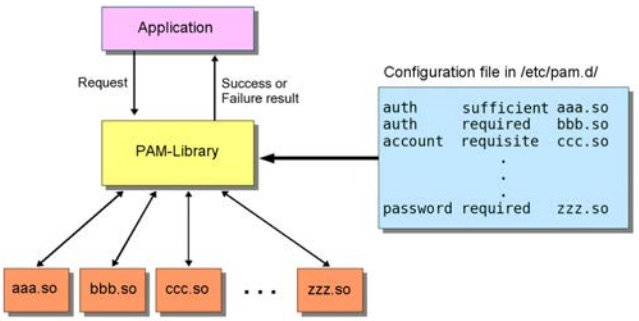
Für jeden Benutzer, der sich per Yubikey authentifizieren soll, wird eine Datei für die Challenge-Response-Authentifizierung angelegt. Dazu dient der Befehl `ykpamcfg -v -1`, wobei `-1` den ersten Slot aktiviert, so wie es im obigen `ykpersonalize`-Befehl gezeigt wurde. Das Programm erzeugt dann eine Datei unter `/.yubico/challenge-3965310`, wobei die Zahl am Ende des Dateinamens die Seriennummer des Yubikeys bezeichnet. Damit es es auch möglich, mehrere Yubikeys für einen Benutzer zu initialisieren.

Statt im Heimatverzeichnis des Benutzers kann die Challenge-Datei auch an einem zentralen Ort hinterlegt werden, beispielsweise unter `/etc/yubico/challenges/`. Dazu müssen die Challenge-Dateien nur in dieses Verzeichnis kopiert und umbenannt werden. Aus den Dateinamen `challenge-SERIENNR.` wird dann `benutzername-SERIENNR.`, also z. B. `/etc/yubico/challenges/anton-3965310`. Dem Yubico-PAM-Modul muss dann lediglich via `chalresp_path` mitgeteilt werden, wo die Challenge-Dateien liegen. Dies kann in größeren Netzwerken die Administration vereinfachen und ist zwingend erforderlich, wenn die Heimatverzeichnisse mittels eCryptFS verschlüsselt sind, da in diesem Fall ein Zugriff auf die lokale Challenge-Datei vor dem Login nicht möglich ist.

Mit `ykpersonalize` konfiguriert man den Yubikey so, dass im zweiten Slot die Challenge-Response-Athentifizierung mit HMAC-SHA1 aktiviert wird. Dabei werden nur die ersten 64 Bytes für die Berechnung des HMAC-SHA1-Hashes verwendet und es wird der Anwendung gestattet, die Seriennummer des Yubikeys auszulesen. Mit der Option `-o chal-btn-trig` kann man einstellen, dass der Yubikey die Antwort im Challenge-Response-Verfahren erst sendet, wenn der Anwender den Hardware-Button gedrückt hat. Ohne diese Option sendet der Yubikey die Antwort sofort.

```
ykpersonalize -2 -ochal-resp -ochal-hmac \
  -ohmac-lt64 -oserial-api-visible
```

Der Yubikey ist nun im Slot 2 für die Offline-Authentifizierung mit HMAC-SHA1 konfiguriert und kann genutzt werden. Um ihn für einen Benutzer bzw. das Yubico-PAM-Modul zu initialisieren, muss der Yubikey eingesteckt sein und noch mittels `ykpamcfg -2` die persönliche Challenge-Response-Datei des Benutzers generiert werden, welche standardmäßig unter `/.yubico/challenge-SERIENNR.` abgelegt wird. Mit der `-p PFAD` kann man auch den zentralen Challenge-Response-Pfad angeben, dann speichert `ykpamcfg` die Datei gleich in diesem Pfad als `USERNAME-SERIENNR.` ab. Damit kann das Yubico-PAM-Modul diesen Yubikey für den konfigurierten Nutzer authentifizieren und das Login gewähren.



Das Yubico-PAM-Modul selbst unterstützt verschiedene Optionen, die in der jeweiligen PAM-Konfigurationsdatei übergeben werden können. Die wichtigsten Optionen sind:

- `mode`: `client` zur Authentifizierung gegenüber einem Online-Server wie der YubiCloud; `challenge-response` für die Offline-Authentifizierung mittels HMAC-SHA1
- `debug`: aktiviert den Debugging-Modus
- `debug_file`: spezifiziert eine Datei, in die die Debugging-Ausgaben umgelenkt werden sollen
- `verbose_otp`: zeigt das Einmalpasswort während der Eingabe an; nur zu Debugging-zwecken sinnvoll, ansonsten eine schwerwiegendes Sicherheitsproblem.

- `authfile`: zentrale Datei in der die Token-IDs für die Benutzer gemappt werden in der Art `Benutzername:TokenID:TokenID2` usw.
- `id` und `key`: gibt die API-Client-ID bzw. den -Key an
- `nullok`: erlaubt ein Login ohne Token, wenn kein Token hinterlegt wurde; damit können sich bestimmte Benutzer ohne Token einloggen, andere nur mit
- `chalresp_path`: Pfad für das zentrale Challenge-Response-Verzeichnis, `/etc/yubico/challenges/` im og. Beispiel

su und ssh mit PAM und Challenge-Response

Um die Offline-Authentifizierung mittels HMAC-SHA1 zu nutzen, muss nur in der entsprechenden PAM-Konfiguration des gewünschten Dienstes das Yubico-PAM-Modul aktiviert werden. Für `su(1)` ist dies beispielsweise `/etc/pam.d/su`, für `sudo(1)` `/etc/pam.d/sudo` und für SSH `/etc/pam.d/sshd`. Das PAM-Modul `pam_yubico.so` befindet sich bei verschiedenen Distributionen an verschiedenen Orten, unter Arch Linux liegt es unter `/usr/lib/security/` und unter Ubuntu unter `/lib/security/`. Der Pfad ist dann abhängig von der verwendeten Distribution anzupassen.

Sollte die PAM-Konfiguration fehlschlagen, kann man sich unter Umständen auf der Maschine nicht mehr einloggen oder per `su(1)` zu Root werden. Deshalb ist es sinnvoll, während der Konfigurationsarbeiten ein zweites Root-Login auf der Zielmaschine offen zu haben und einen `screen` zu öffnen, um im schlimmsten Fall die PAM-Konfiguration zu reparieren.

Um für `su(1)` den Yubikey im Challenge-Response-Mode per PAM zu aktivieren, muss folgende Zeile an `/etc/pam.d/su` angehängt werden:

```
auth required /usr/lib/security/pam\_yubico.so \
   mode=challenge-response debug
```

Die Option `debug` ist für den Anfang empfehlenswert und kann später entfernt werden. Nach der Konfiguration erwartet der `su`-Befehl wie bisher auch das Benutzerpasswort und den eingesteckten Yubikey. Hat man den Yubikey mit `-o chal-btn-trig` konfiguriert, muss man nach Eingabe des Passworts zusätzlich zweimal kurz(!) den Knopf am Yubikey drücken.

Auch OpenSSH lässt sich mittels PAM mit dem Yubikey koppeln, sofern das OpenSSH-Login per Passwort erfolgt und nicht per SSH-Schlüsseldatei. Ob diese Kombination sinnvoll ist, muss jeder Admin für sich selbst entscheiden. Alternativ bietet sich das Verfahren an, einen GnuPG-Authentifizierungsschlüssel zu erstellen und auf den Yubikey hochzuladen, wie im vorangegangenen Abschnitt zu GnuPG beschrieben.

OpenVPN mit Benutzername/Passwort und Challenge-Response vom Yubikey

Auch OpenVPN ab Version 2.4 (ältere Versionen funktionieren nicht) kann Benutzer mit einem Yubikey authentifizieren, entweder via PAM direkt oder im Verbund mit FreeRADIUS. Dazu muss auf dem System das von Yubico angebotene PAM-Modul installiert werden und wie oben beschrieben mittels `ykpamcfg` der Yubikey für den jeweiligen Benutzer im Challenge-Response-Modus initialisiert werden.

Anschließend werden in der OpenVPN-Server-Konfiguration das PAM-Modul aktiviert, das Client-Zertifikat deaktiviert und der Nutzername als Common-Name fürs Protokoll aktiviert. Dies geschieht mit nachfolgenden Optionen, dabei bezeichnet `openvpn` den Namen der Konfigurationsdatei für das PAM-Modul, in der Regel liegt sie in `/etc/pam.d/`:

```
plugin  openvpn-plugin-auth-pam.so  openvpn
client-cert-not-required
username-as-common-name
```

In der OpenVPN-PAM-Konfiguration wird das Yubico-PAM-Modul wie gehabt mit `auth required /usr/lib/security/pam_yubico.so mode=challenge-response` eingebunden. Nun wird beim Verbindungsaufbau zum OpenVPN-Server nach dem Benutzernamen und dem Passwort gefragt sowie automatisch mit dem eingesteckten Yubikey eine Challenge-Response-Authentifizierung durchgeführt.

OpenVPN mit dem Yubikey als PKCS11-Smartcard mit OpenVPN-RSA-Schlüsseln

Die Konfiguration mit Challenge-Response-Verfahren funktioniert ja nur in Verbindung mit einem Benutzername/Passwort-Login. OpenVPN unterstützt jedoch schon seit geraumer Zeit die Authentifizierung über Zertifikate bzw. RSA-Schlüssel. Diese können einerseits als Datei im Dateisystem liegen, oder aber auch vom Yubikey mittels PKCS11 als Smartcard ausgelesen werden. Dann liegen die privaten Schlüssel nicht im Dateisystem, sondern nur in der Hardware des Yubikey vor und sind von dessen PIN geschützt.

Zuerst müssen für den OpenVPN-Server die Zertifikate generiert werden. Dies kann mit dem OpenSSL-Tool EasyRSA geschehen, wie es auf der Webseite unter „../77-rsa-key-management.html“ beschrieben wird (siehe Links). Nach der Erzeugung der Zertifikate liegen folgende Dateien vor:

- `ca.crt`: die Root-Certificate-Authority
- `server.crt`: das signierte lokale Zertifikat
- `server.key`: der lokale private Schlüssel
- `tlsauth.key`: der Pre-Shared-Key für TLS-Auth bzw. TLS-Crypt
- `dh2048.pem`: Diffie-Hellmann-Parameter für die Server-Seite

Der Server muss nun für den Einsatz der Zertifikate konfiguriert werden. Dies geschieht mit den in Abb. 3 gezeigten Befehlen.

```
## das Zertifikat der Root-Certificate-Authority
ca /etc/openvpn/server/easy-rsa/keys/ca.crt

## das signierte lokale Zertifikat
cert /etc/openvpn/server/easy-rsa/keys/server.crt

## der lokale private Schlüssel
key /etc/openvpn/server/easy-rsa/keys/server.key

## das Diffie-Hellmann-Parameter
dh /etc/openvpn/server/easy-rsa/keys/dh2048.pem

## der Pre-Shared-Key für die TLS-Verschlüsselung
## damit wird der TLS-Kontrollkanal authentifiziert
## und verschlüsselt
tls-crypt /etc/openvpn/server/easy-rsa/keys/tlsauth.key

## die zu verwendenden Verschlüsselungsalgorithmen für
## die symmetrische Verschlüsselung des Datenkanals
## Galois/Counter Mode (GCM) bietet zusätzlich neben
## Vertraulichkeit auch Authentizität und Integrität
## der Daten
ncp-ciphers AES-256-GCM:AES-192-GCM:AES-128-GCM

## Authentifizierungsalgorithmus für tls-crypt
auth SHA512

## erlaubte Algorithmen für den TLS-Kontrollkanal
tls-cipher \
TLS-DHE-RSA-WITH-AES-256-GCM-SHA384:\
TLS-DHE-RSA-WITH-AES-256-CBC-SHA256:\
TLS-DHE-RSA-WITH-AES-128-GCM-SHA256:\
TLS-DHE-RSA-WITH-AES-128-CBC-SHA256:\
TLS-DHE-RSA-WITH-AES-256-CBC-SHA 19 tls-server

## erzwingt die Mindestversion von TLS
tls-version-min 1.2

## erlaubt den Gebrauch eines Zertifikats durch
## mehrere Benutzer/Verbindungen
duplicate-cn
```

Abbildung 3: Der Server muss nun für den Einsatz der Zertifikate konfiguriert werden.

Clientseitig sind folgende Optionen analog zu setzen. Mit diesen Optionen kann sich der Benutzer über Zertifikat und Schlüssel (`cert` und `key`) authentifizieren und muss kein Passwort angeben, um eine OpenVPN-Verbindung zu öffnen.

```
ncp-ciphers AES-256-GCM:AES-192-GCM:AES-128-GCM
auth SHA512

tls-crypt "/home/stefan/.openvpn/Beere/tlsauth.key"
ca "/home/stefan/.openvpn/ca.crt"

cert "/home/stefan/.openvpn/client.crt"
key "/home/stefan/.openvpn/client.key"
```

Da die Sicherheit der Authentifizierung von den beiden Dateien mit dem `cert` und `key` abhängt, können diese auch auf den Yubikey im Smartcard-Modus hochgeladen werden und müssen nicht mehr als Datei vorliegen. Dazu benötigen wir auf dem Client neben OpenVPN noch `opensc` und `yubico-piv-tool`.

Zuerst werden der Schlüssel, das Client-Zertifikat und das CA-Zertifikat mittels OpenSSL in das PKCS12-Format konvertiert (1. Kommando). OpenSSL verlangt für die `.p12`-Datei ein Passwort, welches leer bleiben darf. Anschließend wird mit dem `yubico-piv-tool` das p12-

Zertifikat auf den Yubikey hochgeladen (2. Kommando). Mit dem 3. Kommando kann man sich die PKCS11-IDs des eingesteckten Yubikey anzeigen lassen:

```
openssl pkcs12 -export -out certkey.p12 \
  -inkey client.key -in client.crt \
-certfile ca.crt -nodes

yubico-piv-tool -s 9c -i certkey.p12 -K PKCS12 \
  -a import-key -a import-cert -p ""

openvpn {-}-show-pkcs11-ids \
  /usr/lib/x86_64-linux-gnu/opensc-pkcs11.so
```

Wichtig in der Anzeige ist die Zeile `Serialized id:`, deren Wert dann in die OpenVPN-Konfigurationsdatei des Clients eingetragen werden muss. Dabei sind die Backslashes wie im Beispiel unten durch je einen weiteren Backslash zu escapen. Zusätzlich muss mit `pkcs11-providers ...` der PKCS11-Provider angegeben werden. Die `cert`- und `key`-Option kann man auskommentieren bzw. entfernen und die jeweilige Datei löschen, so dass die Client-Konfiguration folgendermaßen geändert wird:

```
#cert "/home/stefan/.openvpn/client.crt"
#key "/home/stefan/.openvpn/client.key"

pkcs11-providers \
 /usr/lib/x86_64-linux-gnu/opensc-pkcs11.so

pkcs11-id \
 piv_II/PKCS\\x2315\\x20emulated/00000000/PIV_II\\x20\\ \
 x28PIV\\x20Card\\x20Holder\\x20pin\\x29/02
```

Nun kann man sich wieder mit dem OpenVPN-Server verbinden. Während des Verbindungsaufbaus wird die PIN des Yubikey abgefragt, so dass OpenVPN auf das hinterlegte Zertifikat zugreifen kann.

Festplattenverschlüsselung mit Yubikey und LUKS

Auch das Festplattenverschlüsselungssystem `LUKS` unter Linux unterstützt Yubikey-Token. Dazu ist es notwendig, die Pakete `yubikey-luks` und `yubikey-personalization` zu installieren. Ersteres aktualisiert `LUKS` dahingehend, dass es den Yubikey unterstützt, das zweite Programm wird benötigt um den Yubikey entsprechend zu konfigurieren.

`LUKS` wird üblicherweise bei der Installation von Linux eingerichtet und zusammen mit dem Logical Volume Manager `LVM` eingesetzt um eine Festplatte komplett zu verschlüsseln. Während des Boot-Prozesses wird dann ein Passwort abgefragt, um die Festplatte zu entschlüsseln und das Betriebssystem vollständig hochzufahren. Ansonsten verhält sich `LUKS` für den Anwender vollkommen transparent, so dass insbesondere die Datenträger in Laptops oder auch externe Festplatten und USB-Sticks unbedingt mit `LUKS` geschützt werden sollten. `LUKS` unterstützt bis zu 8 verschiedene Schlüssel. Die Schlüssel sind in der Regel einfache Passwörter, es kann aber mittels `yubikey-luks` auch ein Challenge-Response-Passwort über den Yubikey eingerichtet werden.

Bei der Installation des Systems lässt man die Festplatte mittels `LVM` und `LUKS` verschlüsseln. Dabei wählt man zunächst ein Passwort als Schlüssel. Da dieses Passwort während der folgenden Konfiguration des Systems mehrfach eingegeben werden muss, bietet sich vorerst ein einfaches und kurzes Passwort an, welches dann später gegen ein sicheres Passwort ausgetauscht wird. Nach der Installation des Systems werden dann noch die Pakete `yubikey-luks` und `yubikey-personalization` installiert.

Mit `cryptsetup` kann man nun in einem der freien sieben Slots den Yubikey als Schlüssel für `LUKS` einrichten. Dazu aktiviert man zuerst den gewünschten Slot, hier ist es der siebte, mit dem Befehl

```
cryptsetup luksAddKey --key-slot 7 /dev/sdb2
```

und nutzt dann `yubikey-luks-enroll` aus dem Paket `yubikey-luks` um den Schlüssel-Slot mit dem Yubikey zu verheiraten:

```
yubikey-luks-enroll -d /dev/sdb2 -s 7 -c
```

Dabei wird man zuerst nach dem `LUKS`-Passwort gefragt, welches man bei der Installation vergeben hat und bei der Konfiguration des Yubikeys nach einem neuen Passwort für das Yubikey Challenge-Response-Verfahren.

Startet man nun den Rechner neu, wird nach dem `LUKS`-Passwort oder dem Yubikey gefragt. Man kann nun entweder das einfache Passwort aus der Installation nutzen oder bereits den Yubikey einstecken und das Challenge-Response-Passwort nutzen. Es empfiehlt sich zumindest einmal, den Yubikey bei einem Neustart zu testen.

Funktioniert der Yubikey wie gewünscht, kann man auch weitere Slots vergeben, so können sich zum Beispiel Alice und Bob einen Laptop teilen und diesen mit einem einzigen Yubikey absichern. Dazu bekommt Alice im Slot 7 ein Passwort vergeben und Bob im Slot 6 ein anderes Passwort für den Yubikey. Der Systemadministrator kann bei Bedarf immer noch mit dem bei der Installation vergebenen Passwort die Festplatte freischalten.

Da dieses Passwort aber noch vergleichsweise unsicher ist und selten gebraucht wird, bietet sich an, dieses durch ein sehr langes zufallsgeneriertes Passwort zu ersetzen. Das Zufallspasswort kann man beispielsweise über einen Passwortgenerator wie `APG` mit `apg -m 128` generieren lassen oder man nutzt `/dev/urandom` und ein Hash-Verfahren wie im folgenden Kommando, um ein hinreichend sicheres 128-stelliges Zufalls-Passwort zu erzeugen.

```
dd if=/dev/urandom bs=1M count=1 2>/dev/null | \
  sha512sum
```

Das unsichere Passwort kann nun wieder mit `cryptsetup` geändert werden:

```
cryptsetup luksChangeKey --key-slot 0 /dev/sdb2
```

Man wird wieder zuerst nach dem unsicheren Installationspasswort und danach nach dem neuen Passwort gefragt. Das neue Zufallspasswort kann man dann auf einem anderen System in einem verschlüsselten Passwortmanager hinterlegen oder auch ausgedruckt an einem sicheren Ort (im Tresor, nicht in der Nähe der Festplatte) aufbewahren.

Links

Webseiten die U2F nativ unterstützen
https://www.dongleauth.info/
Testseite für Browser mit U2F
https://demo.yubico.com/u2f
Yubico-PAM-Modul
https://developers.yubico.com/yubico-pam/
YubiCloud-Registrierung
https://upgrade.yubico.com/getapikey/
OpenSSL-Tool EasyRSA
https://openvpn.net/index.php/open-source/documentation/miscellaneous/77-rsa-key-management.html

Über Stefan

Stefan Schumacher ist Direktor des Magdeburger Instituts für Sicherheitsforschung und Herausgeber des Magdeburger Journals zur Sicherheitsforschung. Er begann seine Hackerkarriere Ende der 80er Jahre auf einem Robotron KC85/3 und arbeitete später einige Jahre im NetBSD-Projekt mit. Er erforscht Sicherheit aus pädagogisch/psychologischer Sicht mit einem Schwerpunkt auf Social Engineering, Security Awareness und Didaktik der Sicherheit.