**BRNO UNIVERSITY OF TECHNOLOGY**
**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**

**FACULTY OF INFORMATION TECHNOLOGY**
**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**

**DEPARTMENT OF INFORMATION SYSTEMS**
**ÚSTAV INFORMAČNÍCH SYSTÉMŮ**


# PORTING TANG TO OPENWRT

**PORTOVANIE TANG NA OPENWRT**

**TERM PROJECT**
**SEMESTRÁLNÍ PROJEKT**


**AUTHOR** — **TIBOR DUDLÁK**
**AUTOR PRÁCE**

**SUPERVISOR** — **Ing. ONDREJ LICHTNER**
**VEDOUCÍ PRÁCE**



**BRNO 2017**

## Abstract
TBD



## Abstrakt
TBD



## Keywords
porting, Tang, OpenWrt, Clevis, Escrow, encryption



## Klíčová slova
portovanie, Tang, OpenWrt, Clevis, Escrow, kryptovanie


## Reference
DUDLÁK, Tibor. *Porting Tang to OpenWRT*. Brno, 2017. Term project. Brno University of Technology, Faculty of Information Technology. Supervisor Lichtner Ondrej.

# Porting Tang to OpenWRT

## Declaration

Hereby I declare that this term project was prepared as an original author's work under the supervision of Ing. Ondrej Lichtner. The supplementary information was provided by Jan Pazdziora, Ph. D.

. . . . . . . . . . . . . . . . . . . . . . . .
Tibor Dudlák
January 30, 2017

# Contents

# Chapter 1

# Introduction

Nowadays, the whole world uses information technologies to communicate and simply to spread knowledge in form of bits to the other people. But there are personal information such as photos from a family vacation, videos of our children as they grow, contracts, testaments and so on which we would like to protect.

Encryption protects our data even when we do not know that. It protects our conversation privacy, personal information stored all over governmental authorities, bank accounts, in general data transmitted around the internet and stored on our hard drives. Encryption provides process of transforming our information in such way that only trusted person can decrypt data and access it. An unauthorized person might be able to access secured data but will not be able to read the information from them without the proper key. The most important thing is keeping the encryption key (the password) a secret.

The goal of this bachelor thesis will be to port *Tang* server 4, its dependencies, and *Clevis* framework 5 to *OpenWrt* system 6. With accomplishing this, we will be able to automatize process of unlocking encrypted drives on our private respectively home network. There will be no need for any decryption server but only *OpenWrt* device running the *Tang* server itself.

# Chapter 2

# Encryption

Nowadays, it is common to have a password protected system. But the encrypted disk requires another password (key) to decrypt. Imagine you come home in a mood to enjoy your time and your system asks for password, not once, but twice. I think this is the reason why most of us do not use encryption, even when we know it will protect our data. This is what
Tang 4 is used for, since we want to automatize things.

Lets take a look on how the encryption is typically done. As you can see in the image Figure 2.1 it all starts with desire to keep our data to ourselves and as a secret to the other people. More often than not, these secrets are stored on our hard drives. Usually we encrypt this secret by using an encryption key. However, our secret data might grow in size, and it is time and resource consuming to decrypt and encrypt the secret every time encryption key changes or it is compromised. Because of that, we wrap encrypted data in the key encryption key and this is what the system prompts from us when booting. So, changing the key encryption key does not affect encrypted data. We can change it whenever we desire to and redistribute the new key to all users who are supposed to access this data.

Figure 2.1: How we encrypt data

To automatize this, we could generate something cryptographically stronger than user provided password. Then, we store this cryptographically stronger random key on a remote system, from where we can get it later. This is basically how the Escrow 3 model works.

# Chapter 3

# Escrow

Before Tang, automated decryption was usually handled by Key Escrow [22] (also known as a "fair" cryptosystem). A client using Key Escrow usually generates a key, encrypts data with it and then stores the key encryption key on a remote server. However, it is not as simple as it sounds.

To deliver these keys we want to store on Escrow server, we have to encrypt the channel on which we distribute the key. When transmitting keys over not secure network without encrypted link, anyone listening to the network traffic could immadiately fetch the key. This should signal security risks, and, of course, we do not want any third party to acces our secret data. Usually we encrypt a channel with TLS [41](Transport Layer Security) or GSSAPI [24] (Generic Security Services Application Program Interface) as you can see on a Figure 3.1 below. Unfortunatelly, this is not enough to call the communication secure.

We cannot just start sending these keys to a server, if we do not know whether this server is the one it acts to be. This server has to have its own identity to trust, and we have to authenticate to this server too. Increasing amount of keys implicates a need for Certification Authority server (CA)[16] or Key Distribution Center (KDC) [31] to manage all of them. With all these keys, and at this point only, server can verify if the person is permitted to get their key, and the client is able to identify trusted server. This is a fully stateful process. To sum up, an authorized third party may gain access to keys stored on Escrow server under certain circumstances only.

Figure 3.1: Escrow model

Complexity of this system increases the attack surface and for this complex system it would be unimaginable not to have backups. Escrow server may store lots of keys from lots of different places and basically we can not afford to lose them.

# Chapter 4

# Tang

Tang server is an open source project implemented in C [17] programming language, and it binds data to network presence. What does binding data to network presence really mean? Essentially, it allows us to make some data to be available only when the system containing the data is on a particular, usually secure, network.

Tang server advertises asymmetric keys[37] and a client is able to get the list of these signing keys 4.5.3 by HTTP [29] GET request. The next step is the provisioning step. With the list of these public keys the process of encrypting data may start. A client chooses one of the asymmetric keys to generate a unique encryption key. After this, the client encrypts data using the created key. Once the data is encrypted, the key is discarded. Some small metadata has to be produced as a part of this operation. The client should store this metadata to work with it when decrypting. Finally, when the client wants to access the encrypted data, it must be able to recover encryption key. This step starts with loading the stored metadata and ends with simply performing a HTTP[29] POST to Tang server. Server performs its mathematical operation and sends the result back to the client. Finally, the client has to calculate the key value, which is better than when server calculates it. So the Tang server never knew the value of the key and literraly nothing about its clients.

Figure 4.1: Tang model

On Figure 4.1 you can see the Tang model. It is very similar to Escrow model 3.1 but there are some thing missing. In fact, there is no longer a need for TLS channel to secure comunication between the client and the server, and that is the reason why Tang implements the McCallum-Relyea exchange 4.1 as described below.

## 4.1 Binding with Tang

A client performs an ECDH [20] key exchange using the McCallum-Relyea algorithm 4.1 in order to generate the binding key. Then the client discards its own private key so that the Tang server is the only party that can reconstitute the binding key. To blind the client's public key and the binding key Tang uses a third, ephemeral key [21]. Ephemeral key is generated for each execution of a key establishment process. Now only the client can unblind his public key and binding key.

| Provisionong | | Recovery | |
|---|---|---|---|
| used client's side | server's side | client's side | server's side |
| | $S\epsilon_R[1, p-1]$ | $E\epsilon_R[1, p-1]$ | |
| | $s = gS$ | $x = c + gE$ | |
| | < − s | x − > | |
| $C\epsilon_R[1, p-1]$ | | | $y = zS$ |
| $e = gC$ | | | < − y |
| $K = gSC = sC$ | | $K = y - sE$ | |
| Discard: K, C | | | |
| Retain s, c | | | |

Table 4.1: McCallum-Relyea exchange

## 4.2 Provisioning

The client selects one of the Tang server's exchange keys (we will call it sJWK; identified by the use of deriveKey in the sJWK's key_ops attribute). The small „s" stands for server's key pair and JWK[6] is used format of the message. The client generates a new (random) JWK (cJWK; c stands for client's key pair). The client performs its half of a standard ECDH exchange producing dJWK which it uses to encrypt the data. Afterwards, it discards dJWK and the private key from cJWK.

The client then stores cJWK for later use in the recovery step. Generally speaking, the client may also store other data, such as the URL [42] of the Tang server or the trusted advertisement signing keys.

$$s = g * S \tag{4.1}$$

$$c = g * C \tag{4.2}$$

$$K = s * C \tag{4.3}$$

## 4.3 Recovery

To recover dJWK after discarding it, the client generates a third ephemeral key (eJWK). Using eJWK, the client performs elliptic curve group addition of eJWK and cJWK, producing xJWK. The client POSTs xJWK to the server.

The server then performs its half of the ECDH key exchange using xJWK and sJWK, producing yJWK. The server returns yJWK to the client.

The client then performs half of an ECDH key exchange between eJWK and sJWK, producing zJWK. Subtracing zJWK from yJWK produces dJWK again.

Mathematically (capital is private key; g stands for generate) client's operation:

$$e = g * E \tag{4.4}$$

$$x = c + e \tag{4.5}$$

$$y = x * S \tag{4.6}$$

$$z = s * E \tag{4.7}$$

$$K = y - z \tag{4.8}$$

## 4.4 Security

We can now compare Tang and Escrow. In contrast, Tang is stateless and doesn't require TLS [41] or authentication. Tang also has limited knowledge. Unlike escrows, where the server has knowledge of every key ever used, Tang never sees a single client key. Tang never gains any identifying information from the client.

| | Escrow | Tang |
|---|---|---|
| Stateless | No | Yes |
| SSL/TLS | Required | Optional |
| X.509 | Required | Optional |
| Authentication | Required | Optional |
| Anonymous | No | Yes |

Table 4.2: Comparing Escrow and Tang

Let's think about the security of Tang system. Is it really secure without an encrypted channel or even without authentication? So long as the client discards its private key, the client cannot recover dJWK without the Tang server. This is fundamentally the same assumption used by Diffie-Hellman (and ECDH)[20].

### 4.4.1 Man-in-the-Middle attack

In this case [34], the eavesdropper in this case sees the client send xJWK and receive yJWK. Since, these packets are blinded by eJWK, only the party that can unblind these values is the client itself (since only it has eJWK's private key). Thus, the MitM attack fails.

### 4.4.2 Compromise the client to gain access to cJWK

It is of utmost importance that the client protects cJWK from prying eyes. This may include device permissions, filesystem permissions, security frameworks (such as SELinux [39]- Security-Enhanced Linux) or even the use of hardware encryption such as a TPM. How precisely this is accomplished is an exercise left to the client implementation.

### 4.4.3 Compromise the server to gain access to sJWK's private key

The Tang server must protect the private key for sJWK. In this implementation, access is controlled by filesystem permissions and the service's policy. An alternative implementation might use hardware cryptography (for example, an HSM [28]) to protect the private key.

## 4.5 Building Tang

Tang is originally packaged for Fedora OS[23] version 23 and later but we can build it from source of course. It relies on few other software libraries:

- http-parser 4.5.1
- systemd / xinetd 4.5.2
- jose 4.5.3
  - jansson 4.5.4
  - openssl 4.5.5
  - zlib 4.5.6

To build it from source, download source from poject's GitHub [25] or clone it. Make sure you have all needed dependencies installed and then run:

```
$ autoreconf -if
$ ./configure -prefix=/usr
$ make
$ sudo make install
```

Optionally to run tests:

```
$ make check
```

### 4.5.1 http-parser

Tang uses this parser for both parsing HTTP requests and HTTP responses. You can find this parser on its own GitHub [13].

### 4.5.2 systemd / xinetd

systemd [15] is a suite of basic building blocks for a Linux [32] system. It provides a system and service manager that runs as PID [36] 1 and starts the rest of the system. systemd provides aggressive parallelization capabilities, uses socket [43] and D-Bus [18] activation for starting services, offers on-demand starting of daemons, keeps track of processes using Linux control groups, maintains mount and automount points, and implements an elaborate transactional dependency-based service control logic.

### 4.5.3 José

José [14] is a C-language implementation of the Javascript [30] Object Signing and Encryption standards. Specifically, José aims towards implementing the following standards:

- RFC 7515 - JSON Web Signature (JWS) [8]
- RFC 7516 - JSON Web Encryption (JWE) [5]
- RFC 7517 - JSON Web Key (JWK) [6]
- RFC 7518 - JSON Web Algorithms (JWA) [10]
- RFC 7519 - JSON Web Token (JWT) [9]

- RFC 7520 - Examples of ... JOSE [7]
- RFC 7638 - JSON Web Key (JWK) Thumbprint [6]

JOSE (Javascript Object Signing and Encryption) is a framework intended to provide a method to securely transfer claims (such as authorization information) between parties.

Tang uses JWKs in comunication between client and server. Both POST request and reply bodies are JWK objects.

### 4.5.4 jansson

Jansson [11](licenced under MIT [35] licence) is a C library for encoding, decoding and manipulating JSON data. It features:

- Simple and intuitive API and data model
- Comprehensive documentation
- No dependencies on other libraries
- Full Unicode support (UTF-8)
- Extensive test suite

### 4.5.5 OpenSSL

OpenSSL [4] contains an open-source implementation of the Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols. It is used by network applications to secure communication between two parties over network.

### 4.5.6 zlib

Library zlib [1] is used for data compression.

## 4.6 Server enablement

Enabling a Tang server is a two-step process. First, enable and start the service using systemd.

```
$ sudo systemctl enable tangd-update.path

$ sudo systemctl start tangd-update.path

$ sudo systemctl enable tangd.socket

$ sudo systemctl start tangd.socket
```

Second, generate a signing key and an exchange key.

```
$ sudo jose gen -t '{"alg":"ES256"}' -o /var/db/tang/sig.jwk

$ sudo jose gen -t '{"kty":"EC","crv":"P-256","key_ops":["deriveKey"]}' \
      -o /var/db/tang/exc.jwk
```

Now we are up and running. Server is ready to send advertisment on demand.

# Chapter 5

# Clevis

Clevis provides a pluggable key management framework for automated decryption [12]. It can handle even automated unlocking of LUKS [33] volumes. To do so, we have to encrypt some data with simple command:

```
$ clevis encrypt PIN CONFIG < PLAINTEXT > CIPHERTEXT.jwe
```

In clevis terminology, a *pin* is a plugin which implements automated decryption. We simply pass the name of supported pin here. Secondly *config* is a JSON object which will be passed directly to the *pin*. It contains all the necessary configuration to perform encryption and setup automated decryption.

## 5.1 PIN: Tang

Clevis has full support for Tang. Here is an example of how to use Clevis with Tang:

```
$ echo hi | clevis encrypt tang '{"url": "http://tangserver"}' > hi.jwe
The advertisement is signed with the following keys:
    kWwirxc5PhkFIH0yE28nc-EvjDY

Do you wish to trust the advertisement? [yN] y
```

In this example, we encrypt the message „hi“ using the Tang pin. The only parameter needed in this case is the URL of the Tang server. During the encryption process, the Tang pin requests the key advertisement from the server and asks you to trust the keys. This works similarly to SSH [38].

Alternatively, you can manually load the advertisment using the adv parameter. This parameter takes either a string referencing the file where the advertisement is stored, or the JSON contents of the advertisment itself. When the advertisment is specified manually like this, Clevis presumes that the advertisement is trusted.

## 5.2 PIN: HTTP

Clevis also ships a pin for performing escrow using HTTP. Please note that, at this time, this pin does not provide HTTPS support and is suitable only for use over local sockets. This provides integration with services like Custodia.

## 5.3 PIN: SSS - Shamir Secret Sharing

Clevis provides a way to mix pins together to provide sophisticated unlocking policies. This is accomplished by using an algorithm called Shamir Secret Sharing (SSS) [40].

## 5.4 Binding LUKS volumes

Clevis can be used to bind a LUKS volume using a pin so that it can be automatically unlocked.

How this works is rather simple. We generate a new, cryptographically strong key. This key is added to LUKS as an additional passphrase. We then encrypt this key using Clevis, and store the output JWE inside the LUKS header using LUKSMeta.

Here is an example where we bind `/dev/vda2` using the Tang ping:

```
$ sudo clevis bind-luks /dev/sda1 tang '{"url": "http://tang.local"}'
The advertisement is signed with the following keys:
        kWwirxc5PhkFIH0yE28nc-EvjDY

Do you wish to trust the advertisement? [yN] y
Enter existing LUKS password:
```

Upon successful completion of this binding process, the disk can be unlocked using one of the provided unlockers.

### 5.4.1 Dracut

The Dracut unlocker [19] attempts to automatically unlock volumes during early boot. This permits automated root volume encryption. Enabling the Dracut unlocker is easy. Just rebuild your initramfs after installing Clevis:

```
$ sudo dracut -f
```

Upon reboot, you will be prompted to unlock the volume using a password. In the background, Clevis will attempt to unlock the volume automatically. If it succeeds, the password prompt will be cancelled and boot will continue.

### 5.4.2 UDisks2

Our UDisks2 unlocker [3] runs in your desktop session. You should not need to manually enable it; just install the Clevis UDisks2 unlocker and restart your desktop session. The unlocker should be started automatically.

This unlocker works almost exactly the same as the Dracut unlocker. If you insert a removable storage device that has been bound with Clevis, we will attempt to unlock it automatically in parallel with a desktop password prompt. If automatic unlocking succeeds, the password prompt will be dissmissed without user intervention.

# Chapter 6

# OpenWrt

OpenWrt is described as a Linux distribution for embedded devices(typically wireless routers). It provides a fully writable filesystem with package management, so we are not bound to application provided by the vendor. It allow us to customize the device through the use of packages to suit any application.

The OpenWrt project started in January 2004. The first OpenWrt versions were based on Linksys GPL [27] sources for WRT54G and a buildroot from the uClibc project. Today (January 2017) the stable 15.05.1 release of OpenWrt (codenamed „Chaos Calmer") released in March 2016 using Linux kernel version 3.18.23 runs on hundreds of routers.

## 6.1 OPKG Package manager

The opkg utility (an ipkg fork) is a lightweight package manager used to download and install OpenWrt packages from local package repositories or ones located in the Internet. GNU/Linux users already familiar with apt, aptitude, pacman, yum, dnf, etc. will recognize the similarities. It also has similarities with NSLU2's Optware, also made for embedded devices. OPKG is however a full package manager for the root file system, instead of just a way to add software to a seperate directory (e.g. /opt). This also includes the possibility to add kernel modules and drivers. OPKG is sometimes called Entware, but this is mainly to refer to the Entware repository for embedded devices.

Opkg attempts to resolve dependencies with packages in the repositories - if this fails, it will report an error, and abort the installation of that package.

Missing dependencies with third-party packages are probably available from the source of the package. To ignore dependency errors, pass the –force-depends flag.

# Chapter 7

# Conclusion

The Tang 4 server is a very lightweight program. It provides secure and anonymous data binding using McCallum-Relyea exchange **??** algorythm.

Clevis 5 is a client software with full support for Tang. It has minimal dependencies and it is possible to use with HTTP, Escrow 3, and it implements Shamir Secret Sharing[40]. Clevis has GNOME [26] integration so it is not only a command line tool. Clevis also supports early boot integration with dracut5.4.1 or even removable devices unocking using UDisks2 5.4.2.

To port Tang to OpenWrt system 6 it will be necesarry to port all its dependencies first. The OpenWrt system has already package openssl 4.5.5, zlib 4.5.6, and jansson 4.5.4 but only version 2.7 which is too old. So there will be a need for porting jansson. José will require porting and http-parser 4.5.1 too. The systemd 4.5.2 would be huge effort but tang's requirements are minimal and we should be able to work with xinetd [2]. Finally, some work will be required to port Tang itself.

# Bibliography


[1] Adler, M.: *zlib*. [Online].
Retrieved from: https://github.com/madler/zlib

[2] Braun, R.: *xinetd*. [Online].
Retrieved from:
http://web.archive.org/web/20051227095035/http://www.xinetd.org:80/

[3] Desktop, F.: *UDisk2*. [Online].
Retrieved from: https://udisks.freedesktop.org/docs/latest/

[4] Eric A. Young, T. J. H.: *OpenSSL*. [Online].
Retrieved from: https://github.com/openssl/openssl

[5] Jones, e. a.: *JSON Web Encryption*. [Online].
Retrieved from:
http://tools.ietf.org/html/draft-ietf-jose-json-web-encryption

[6] Jones, e. a.: *JSON Web Keys*. [Online].
Retrieved from: http://tools.ietf.org/html/draft-ietf-jose-json-web-key

[7] Jones, e. a.: *JSON Web Keys*. [Online].
Retrieved from: https://tools.ietf.org/html/rfc7520

[8] Jones, e. a.: *JSON Web Signing*. [Online].
Retrieved from:
http://tools.ietf.org/html/draft-ietf-jose-json-web-signature

[9] Jones, e. a.: *JSON Web Tokens*. [Online].
Retrieved from:
https://tools.ietf.org/html/draft-ietf-oauth-json-web-token

[10] Jones, e. a.: *JWT Authorization Grants*. [Online].
Retrieved from: http://tools.ietf.org/html/draft-ietf-oauth-jwt-bearer

[11] Lehtinen, P.: *jansson*. [Online].
Retrieved from: https://github.com/akheron/jansson

[12] McCallum, N.: *Clevis*. [Online].
Retrieved from: https://github.com/latchset/clevis

[13] McCallum, N.: *HTTP-parser*. [Online].
Retrieved from: https://github.com/nodejs/http-parser

[14] McCallum, N.: *jose*. [Online].
Retrieved from: https://github.com/latchset/jose

[15] Poettering, L.; Sievers, K.; Hoyer, H.; et al.: *systemd*. [Online].
Retrieved from: https://en.wikipedia.org/wiki/Systemd

[16] Wikipedia: *Certification Authority*. [Online].
Retrieved from: https://en.wikipedia.org/wiki/Certificate_authority

[17] Wikipedia: *C programming language*. [Online].
Retrieved from: https://en.wikipedia.org/wiki/C_(programming_language)

[18] Wikipedia: *Dbus*. [Online].
Retrieved from: https://en.wikipedia.org/wiki/D-Bus

[19] Wikipedia: *dracut (software)*. [Online].
Retrieved from: https://en.wikipedia.org/wiki/Dracut_(software)

[20] Wikipedia: *Elliptic curve Diffie–Hellman*. [Online].
Retrieved from:
https://en.wikipedia.org/wiki/Elliptic_curve_Diffie%E2%80%93Hellman

[21] Wikipedia: *Ephemeral key*. [Online].
Retrieved from: https://en.wikipedia.org/wiki/Ephemeral_key

[22] Wikipedia: *Escrow*. [Online].
Retrieved from: https://en.wikipedia.org/wiki/Escrow

[23] Wikipedia: *Fedora (operating system)*. [Online].
Retrieved from: https://en.wikipedia.org/wiki/Fedora_(operating_system)

[24] Wikipedia: *Generic Security Services Application Program Interface*. [Online].
Retrieved from: https://en.wikipedia.org/wiki/
Generic_Security_Services_Application_Program_Interface

[25] Wikipedia: *GitHub*. [Online].
Retrieved from: https://en.wikipedia.org/wiki/GitHub

[26] Wikipedia: *GNOME*. [Online].
Retrieved from: https://en.wikipedia.org/wiki/GNOME

[27] Wikipedia: *GPU General Public Licence*. [Online].
Retrieved from: https://en.wikipedia.org/wiki/GPL

[28] Wikipedia: *Hardware security module*. [Online].
Retrieved from: https://en.wikipedia.org/wiki/Hardware_security_module

[29] Wikipedia: *Hypertext Transfer Protocol*. [Online].
Retrieved from: https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

[30] Wikipedia: *JavaScript*. [Online].
Retrieved from: https://en.wikipedia.org/wiki/JavaScript

[31] Wikipedia: *Key distribution center*. [Online].
Retrieved from: https://en.wikipedia.org/wiki/Key_distribution_center

[32] Wikipedia: *Linux*. [Online].
Retrieved from: https://en.wikipedia.org/wiki/Linux

[33] Wikipedia: *Linux Unified Key Setup*. [Online].
Retrieved from: https://en.wikipedia.org/wiki/Linux_Unified_Key_Setup

[34] Wikipedia: *Man in the middle attack*. [Online].
Retrieved from: https://en.wikipedia.org/wiki/Man-in-the-middle_attack

[35] Wikipedia: *Massachusetts Institute of Technology*. [Online].
Retrieved from:
https://en.wikipedia.org/wiki/Massachusetts_Institute_of_Technology

[36] Wikipedia: *Process identifier*. [Online].
Retrieved from: https://en.wikipedia.org/wiki/Process_identifier

[37] Wikipedia: *Public key cryptography*. [Online].
Retrieved from: https://en.wikipedia.org/wiki/Public-key_cryptography

[38] Wikipedia: *Secure Shell*. [Online].
Retrieved from: https://en.wikipedia.org/wiki/Secure_Shell

[39] Wikipedia: *Security-Enhanced Linux*. [Online].
Retrieved from: https://en.wikipedia.org/wiki/Security-Enhanced_Linux

[40] Wikipedia: Shamir Secret Sharing. [Online].
Retrieved from: https://en.wikipedia.org/wiki/Shamir's_Secret_Sharing

[41] Wikipedia: *Transport Layer Security*. [Online].
Retrieved from: https://en.wikipedia.org/wiki/Transport_Layer_Security

[42] Wikipedia: *Uniform Resource Locator*. [Online].
Retrieved from: https://en.wikipedia.org/wiki/Uniform_Resource_Locator

[43] Wikipedia: *Unix domain socket*. [Online].
Retrieved from: https://en.wikipedia.org/wiki/Unix_domain_socket