Extremely Secure Communication
Research Project 2

Daniel Romão - daniel.romao@os3.nl
System and Network Engineering, MSc
University of Amsterdam

July 5, 2015

Abstract

Companies nowadays rely on the Internet for various purposes. One of these, is communication between employees or work teams at different locations. Suspicions of governmental agencies performing traffic collection and decryption exist, posing the threat of companies having their Internet communication leaked. This is specially relevant when confidential information is transmitted and their business secrets might be at risk.
This research shows how an extremely secure communication on the Internet can be deployed for work teams and individuals around the globe. Having the knowledge that random number generators used for cryptographic operations might contain backdoors, a true random number generator was implemented. This random number generator is based on an older true random number generator project, which uses a single PN junction for noise generation, and will include new features and more noise sources: a second PN junction and a jitter-based noise generator. This device improves the entropy of the Linux system used by clients, improving the cryptographic operations. Companies, often have Virtual Private Networks (VPNs), which employees can use when are outside of the company. These not only provide an isolation layer, but also access to internal services. However, this means that communication between employees will always go through a central location, making all communication easier to obtain. This happens because only the Internet link to the location of the VPN server needs to be tapped. In order to avoid this, a multipoint VPN is used. The technology used is Dynamic Multipoint Virtual Private Network (DMVPN). With a multipoint VPN, the VPN clients communicate directly with each other. The operating system used for the clients is Tails Linux. Because Tails Linux always boots from a clean state, a bootstrap software was created to allow easy deployment of the multipoint VPN and random number generator software.
Testing results of the random number generator obtained from the tools Ent and rngtest show good performance, favoring the dual PN junction without filtering setting. Throughput tests shown that the throughput of the jitter-based noise generator is significantly lower than the throughput of the PN junction noise generators, diminishing its usefulness as a suitable noise generator. Tests of the bootstrap software shown that the deployment of the multipoint VPN and random number generator software is not only possible, but easy when the software created is used. It is possible to conclude that the scenario proposed is valid and both random number generator and multipoint VPN integrate well.

Contents

1 Introduction

Privacy on the Internet is a huge topic of discussion nowadays. Not only data people supply voluntarily is subject of analysis by companies and governments, such as pictures on social networks, but also private and confidential data which is meant to be kept secret. For example, the National Security Agency (NSA)[1] is known for collecting Internet traffic from several links[2], store the traffic collected, and attempt to decrypt it or at least collect metadata from the traffic[1]. In addition, suspicion of backdoors on current encryption-related software and random number generators exist[2]. Those backdoors allow decryption of captured traffic to be made easier, allowing access to data.

In this research, a scenario to enable remote work teams and individuals to communicate and share data securely is proposed and developed. Examples where this would be useful, are an oil company finding a new source of oil, or a company in process of a due diligence assessment on taking over another company and does not want to influence the market value. In order to tackle the issue of possible backdoors, an open-source, verifiable, operating system was used and a true random number generator (TRNG)[3], also known as hardware random number generator (HRNG), was implemented. Further, this device will also be referred simply as RNG. To reduce the possibility of having a link tapped, where the sensitive data is transported, all data is not only encrypted, but also travels over a multipoint VPN. A multipoint VPN ensures that all traffic is isolated and no single tap point exists, as individual VPN tunnels are created between all participants.

1.1 Problem statement

The main research question is:

> How can an extremely secure communication on the Internet be deployed for work teams and individuals around the globe?

This question leads to the following sub-questions:

1. Given the hardware random number generators on computer hardware not being secure, how feasible is it to use an external RNG device?
2. What kind of TRNGs designs exist and what are their advantages/disadvantages?
3. How feasible is it to implement a TRNG design using easily available, low cost, resources?
4. What implementations of open-source multipoint VPNs exist?
5. How feasible is the deployment of one of this technologies on volatile operating systems?
6. How feasible is the implementation of this multipoint VPN using the TRNG built for sub-question 3?

1.2 Research methods

In this sub section, the research methods applied for each sub question will be described.

1.2.1 Given the hardware random number generators on computer hardware not being secure, how feasible is it to use an external one?

For this question, the process of building the RNG and interfacing it with a computer will show how easily this can be done.

[1] https://www.nsa.gov
[2] http://www.theguardian.com/world/2013/jul/31/nsa-top-secret-program-online-data

1.2.2 What kind of TRNGs designs exist and what are their advantages/disadvantages?

Research on currently developed noise generator circuits and TRNGs exist and investigate how those compare. A noise generator circuit design will be used for the TRNG created.

1.2.3 How feasible is it to implement a TRNG design using easily available, low cost, resources?

The process of implementation of the TRNG will show how easy it is. This will also show if it can be build using easily available parts, and how expensive it is.

1.2.4 What implementations of open-source multipoint VPNs exist?

Suitable open-source multipoint implementations will be searched and one will be picked for the proof-of-concept.

1.2.5 How feasible is the deployment of one of this technologies on volatile operating systems?

The process of deploying and configuring the multipoint VPN software will show how easy it is to deploy and configure the software.

1.2.6 How feasible is the implementation of this multipoint VPN using the TRNG built for sub-question 3?

To answer this question, the integration of the TRNG built with the multipoint VPN will be evaluated.

2 Related work

2.1 True random number generator

Encrypted connections are essential for secure communication. However, those can still be exploited in case of a presence of a backdoor in the implementation of security-related protocols or on what will be explored, a randomization source of key material generation.
Several TRNGs have been produced over the years. Good examples are the OneRNG[3], the TrueRNG[4], and the entropy key[5]. From those, OneRNG is the only having an open hardware design. This is important as a user can assess how the device operates and look for possible flaws that might reduce its performance in producing random numbers. Unfortunately, it is not available for sale at the time of this research. The OneRNG and the entropy key are known to use PN junctions[4] as noise source.
While those products are great, the possibility of building a TRNG from scratch allows better control and understanding of a TRNG, adding the possibility for experimentation, such as adding noises sources or try different components. Arduino[6] is a popular micro-controller development project, where several development boards are produced within the project.
The Arduino is a good option for interfacing an analog circuit to a computer. Other options, such as Field Programmable Gate Arrays (FPGAs), have been used for the same purpose. An example where an FPGA was used for same purpose can be seen in the paper "An embedded true random number generator for FPGAs" by Paul Kohlbrenner and Kris Gaj[5]. Even though good results can be possible to obtain using FPGAs, the Arduino platform is not only more affordable, but also more widespread.
As for the circuit itself, a few designs exist, which follow the same technology as the TRNGs mentioned before, using PN junctions. An interesting one, because it can be easily implemented and capable of good results, and base for the TRNG that will be implemented, was created by Rob Seward[6], while his work was based on work of Will Ware[7] and Aaron Logue[8]. A different approach for noise generation, is to use ring oscillators[9][10]. While ring oscillators could also be an option, the PN junctions will be preferred as the implementation is easier.

2.2 Multipoint VPN

A large number of initiatives for Internet privacy have been started. A well known one is Tor[7]. Tor is a network aimed at providing secure and anonymous access to the Internet. While Tor is a great complement to the target scenario, it does not provide the isolation desired. It is relevant to notice that in this project, anonymity is not a requirement, while data confidentiality and an easy to use private environment is. Virtual Private Networks are commonly used by companies who wish remote employees or sites, to be able to connect to a trusted company network. However, having a central point of connectivity that can be exploited is not a desirable situation. Networking vendors have been putting an effort in developing technologies to provide secure multipoint VPNs. Using a multipoint allows remote sites/users to communicate with each other without having their traffic going trough a central location. The most relevant multipoint VPN technology found is the DMVPN[11], developed by Cisco[8]. Good detail of this technology can be found on their patent, named "Method and apparatus for dynamically securing voice and other delay-sensitive network traffic"[12]. Information about deploying a secure DMPVN can be seen on the paper "Design and implementation of secure enterprise network based on DMVPN" by Huaqi Chen[13].
A very interesting alternative is N2N[14], as it also has desired characteristics for this research, such as a peer-2-peer topology. N2N is a secure peer-2-peer VPN, which uses UDP for traffic encapsulation. Unfortunately, N2N is not being developed at the moment, and for that reason, it was not considered as a suitable solution for the networking component of this project.

[3] http://onerng.info/
[4] http://ubld.it/products/truerng-hardware-random-number-generator/
[5] http://www.entropykey.co.uk/
[6] http://www.arduino.cc/
[7] https://www.torproject.org/
[8] http://www.cisco.com

3 Scope

3.1 Multipoint VPN and Operating System

The planned multipoint VPN technology for deployment in this project is DMVPN. An open-source implementation of this technology was used. On the hosts, the Tails Linux[9] operating system was used, and where the DMVPN implementation run on. Because this operating system always starts from its installed state, a set of scripts was created to deploy the software with minimal user interaction. Creating a new distribution with the VPN software pre-installed is out of scope, not only for the sake of time, but also because of the maintenance effort that would be needed to keep the new distribution up-to-date. Using a bootstrap script will allow the software to be deployed and run on new versions of the Tails Linux operating system, requiring at most, minimal changes that might be required for compatibility with a new version of the operating system or VPN-related software.

3.2 True random number generator

The purpose of implementing a true random number generator in this project, is to assess the feasibility of the implementation, along with the assessment of the advantages over a pseudo random number generator. Research was done into the circuits typically deployed, and one was deployed. Designing new circuits/methods, as well as deep statistical/mathematical analysis is out of the scope of this project. However, basic statistical analysis were done in order to do an assessment of the RNG deployed, such as entropy calculation.

[9] https://tails.boum.org/index.en.html

4 Approach

4.1 True Random Number Generator

The first part of the approach was implementing a TRNG, which is primarily based on a PN junction circuit for avalanche noise[15] generation and an Arduino for data processing and interfacing with a PC. In order to experiment and attempt to improve the performance of the RNG, two equal PN junction circuits were built. The output of these circuits were connected to an Arduino development board, which performs proper acquisition and processing of the noise by converting the analog signal to digital, and comparing it with a baseline which leads to assessment of the bit value of the analog signal sampled. On the Arduino board, an internal noise generator was also used in conjunction with the analog circuits. This internal RNG uses the jitter of the watchdog timer of the Arduino, to generate a random stream. The library Entropy[16] implements this technique and was used in this project. Figure 1 shows how the components are connected.

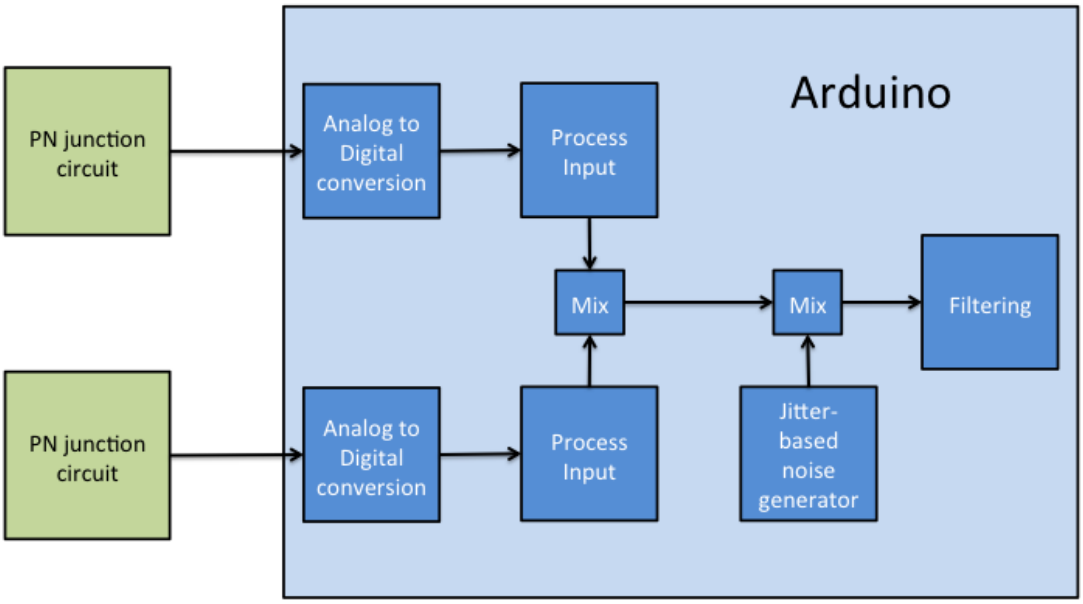

Figure 1: Block diagram of the RNG

After the signal from the PN junctions is converted to a digital value, between 0 and 255 (8 bit) and mixed, which is achieved by a xor operation, the bit value is mixed with the the jitter-based noise generator and filtering is performed. After the filtering, the the result is sent over a serial interface. The output of the RNG is used to feed the (Linux) kernel entropy pool of the host where it is connected, in order to be used by /dev/random and /dev/urandom. For this task, rng-tools[10] was used. The Arduino used in this project is a Duemilanove clone. Even not being an original board from the Arduino brand, it will be continued to be referred as "Arduino", as the boards are compatible. Other Arduino and Arduino-like boards might be compatible with the noise generator circuits and code, however proper checking is advised, specially because the output of the PN junction noise generators might damage lower voltage (3.3V) boards.

4.1.1 PN Junction circuit

The RNG implementation includes two PN junction circuits for noise generation. A diagram of the circuit implemented can be seen on figure 2. This circuit was originally designed by Aaron Logue[8], and later modified by Rob Seward[6].

[10] https://www.gnu.org/software/hurd/user/tlecarrour/rng-tools.html

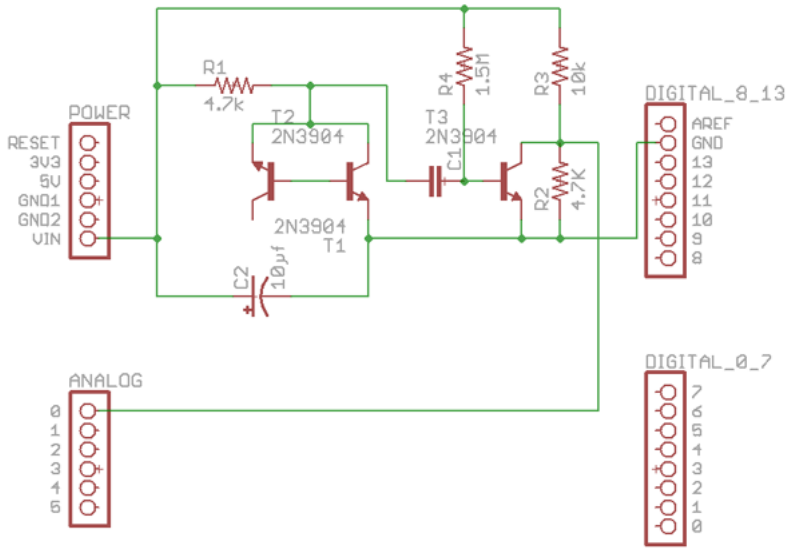

Figure 2: PN Junction circuit[11]

During the practical implementation of the circuits, the 1.5MΩ resistor was replaced by a 1.2MΩ resistor in series with a 220KΩ resistor, because of the lack of availability of resistor of the original value, and a single 10μF capacitor was used for both circuits. The assembly was done on a traditional breadboard and can be seen on figure 3. The RNG was powered with a 12V adapter, as 12V is required by the PN junction circuits.

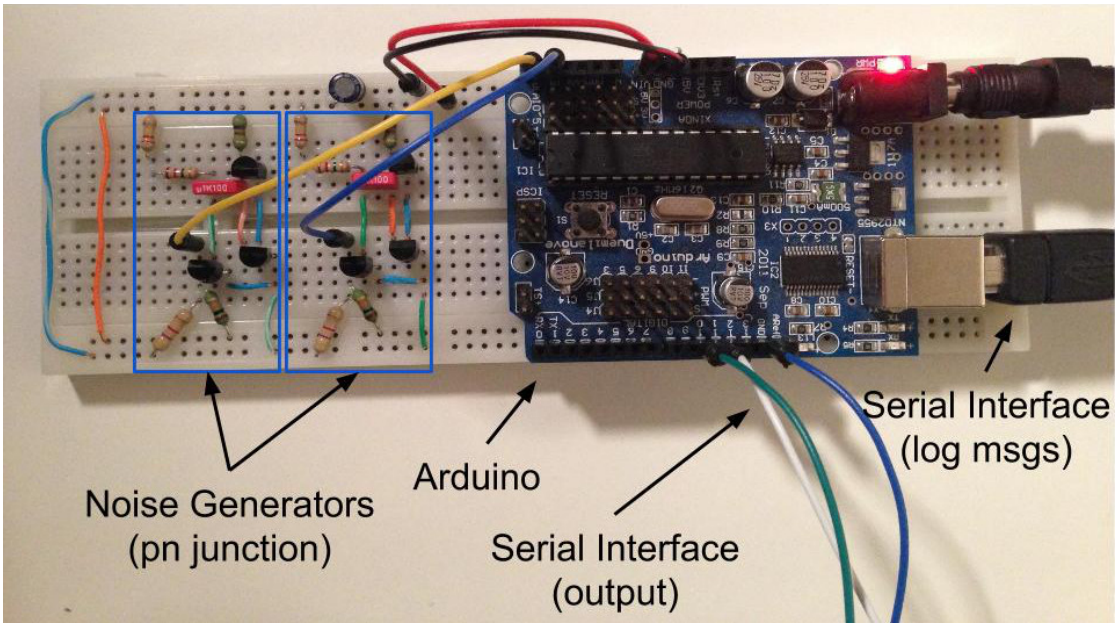

Figure 3: Picture of the RNG created

[11]Image retrieved from: `http://robseward.com/misc/RNG2/RNG_Version_2_images/rng2_circuit_small.png`

4.1.2 Arduino code

The Arduino code was based on work done by Rob Seward. His code supports a single PN junction noise generator and can perform two different filtering functions: exclusive or (xor) and Von Neumann. In order to find a baseline for the noise generator, before any data is sent to the PC, a large (50000) amount of samples is taken, and the median is found. The code will then use this median to assess if the value of a certain sample should be translated into a 0 or 1. The Arduino model used support analog signals between 0V and 5V. The analog to digital converter, convert the analog signal to a digital value.

The code implements the following output modes:

- ASCII byte
- Binary
- ASCII Boolean

Even though the code already developed seemed to work well, it was too limited for the RNG intended. The code was then extended to support more noise generators and a raw byte output mode for rng-tools, and a few other improvements. A summary of the new features contributed is:

- Support for a second PN junction noise generator
- Support for the watchdog timer jitter-based noise generator provided by the Entropy library
- New raw byte output mode
- Continuous calibration
- Support for a second serial interface for logging messages
- Send log messages, such as baseline values, mode of operation, etc, over the second serial interface

In an attempt to improve the performance of the RNG, two more sources of noise were added: a second PN junction circuit and a different noise source are now supported. This new noise source (watchdog time jitter) is handled by the Entropy library, which made its integration easy. The rng-tools expects to receive raw bytes from a RNG, an output mode missing in the original code. This output mode was implemented and used for all tests.

One issue with the original code, is the calibration operation is only done once, when the device is powered on. This is fine if the device is unplugged often, but if the device is left running for several months, the bias of the circuit will drift over time, and the threshold will no longer be valid. This results in a reduced performance of the RNG. Factors that might cause this bias drift are, for example, the room temperature, aging components, and voltage fluctuations. To allow the device to run for long periods of time, for example, when connected to a server, without performance degradation, besides the calibration on start-up, the device will be continuously calibrating itself and adjust the threshold as needed. This is achieved by using the same procedure done when the device boots, in simultaneous with the regular operation of the RNG. This is possible because the result of the analog to digital converters is used for both calibration and regular operation. After each calibration operation is performed, the structure containing the data captured and used for the median calculation, is re-initialized and the calibration operations starts from the beginning. Each PN junction has its own baseline.

With the addition of a second serial interface, one serial interface can be used as RNG output and one for logging purposes. The additional serial interface is software emulated, as this device only has one hardware serial interface. In contrast with the original code, which uses the

hardware serial port for output, the output of the RNG implemented is sent over the software serial interface and log messages over the hardware serial interface. The reason for this is closing and opening operations on the hardware serial interface triggers a reset on the micro-controller, a feature used for programming the device. If the software reading the RNG interrupts the connection because, for example, no data from the RNG is needed at a certain moment, the Arduino will restart and go through the boot process over and over. This behaviour is non-existent on the software serial interface and in case of interruptions, the Arduino will continue running without interruptions.
An example log output can be seen below:

```
TRNG Starting...
2 external noise generator(s) will be used
The internal noise generator is disabled
Filtering applied: von Neumann
The output format is byte
*********
The threshold 1 is: 59

The threshold 2 is: 24
```

4.1.3 Tests

In order to validate and assess the performance of the RNG, several statistical tests were performed. First, 512 Kb samples of raw data from the RNG are planned to be collected, for all possible modes of operation and filtering. For collection of these samples, a Linux machine was used, and the following example commands can be used for the effect:

```
time head -c 512K </dev/ttyUSB0 >rng_single_xor

time head -c 512K </dev/ttyUSB0 >jitter_no_filtering
```

The time command was used for time measurement. This allows the calculation of the throughput of the RNG in the different modes of operation and filtering options.
For performing the statistic tests, the tools rngtest and Ent were used. The rngtest tool checks if the data complies with the FIPS 140-2 standard. It checks every 2500 byte blocks individually, and as output shows how many of these blocks meet the requirements of the standard, and in case of blocks not meeting the requirements, it shows the category of the reason.
Ent performs several checks and calculations on the data. These tests are listed on table 1.

Test	Description	Ideal Result
Entropy	Density of the information in a file Measured in bits per character	Should be as close to the number 8 as possible
Optimum compression	Perform data compression	Ideally, data compression will not be possible, showing a totally unpredictable sequence
Chi square	Calculate the Chi square distribution[17]	Should be between 10% and 90%
Arithmetic mean	Calculate the arithmetic mean of the data: Sum all bytes and divide by file size	Should be as close to 127.5 as possible
Monte Carlo value for Pi	Calculates the value of Pi using the Monte Carlo method[18]	Should as close to the value of Pi as possible: 0% error
Serial correlation coefficient	Measures how much a byte depends on the previous	Should be as close to zero as possible

Table 1: Tests performed by the Ent utility

4.2 Multipoint VPN

Having a good source of entropy, the next step is to make the communication over the Internet as secure and private as possible. As one the goals is to have a single computer system to be autonomous, OpenNHRP, an open-source implementation of the DMVPN protocol was used, as it can run on a standard Linux system. OpenNHRP implements the Next Hop Resolution Protocol and aims to be Cisco compatible. Using other Linux software, ipsec-tools and racoon, it is possible to build an entire secure DMVPN topology.

The operating system users run is the Debian-based Tails Linux. Tails Linux was chosen for this research, as it is an operating system aimed at providing a secure, private access to the Internet. Since the functionality of Tails Linux is very likely of interest of the target audience of this research, the multipoint VPN makes it a great addition.
Opposite to the way DMVPN is typically deployed, a DMVPN spoke, traditionally a border router on a site, is self contained within the users' system. This allows single, independent users, to access the VPN without any extra equipment. Figure 4 shows an example topology.

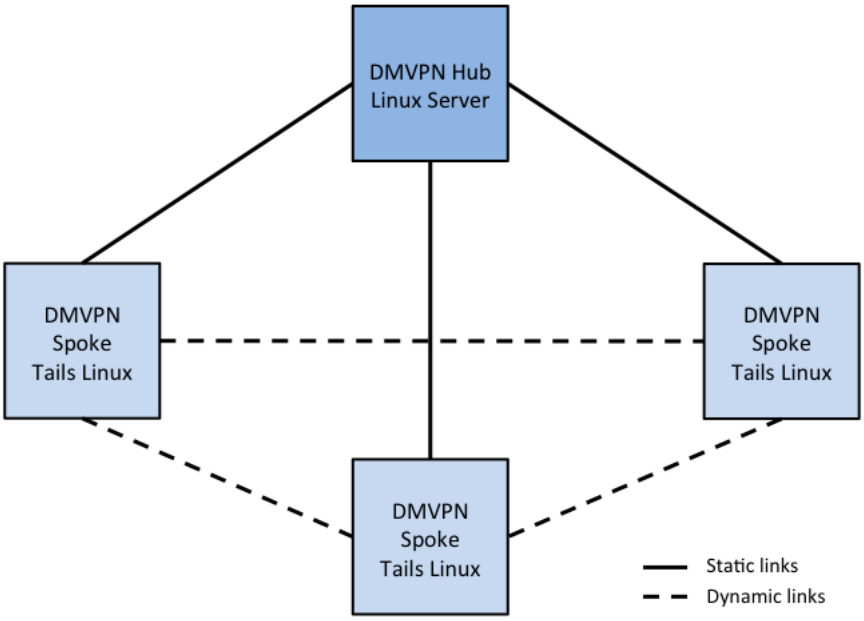

Figure 4: Example topology

The DMVPN hub in the experimental setup is an Ubuntu Linux 14.04 server machine. The hub authenticates users (spokes) and register them after successful authentication. The hub handles ARP-like messages from the spokes, requesting to know what is the outside IP address of another spoke with whom they wish to communicate with. After receiving the address resolution reply from the hub, a spoke can contact the other spoke in order to establish a dedicated tunnel.

Deploying a DMVPN spoke on Tails Linux poses extra challenges when comparing with an ordinary Linux distribution:

1. Tails Linux is made to no leave traces behind. All data changes on a machine are lost on shutdown/reboot and after every boot a clean system is presented.

2. The operating system typically runs from a USB stick, without persistent storage.

3. To ensure anonymity, only traffic to the Tor and I2P networks is allowed. All other traffic is dropped.

In order to solve the first two points, one of the functionalities of Tails Linux, is the possibility to create an encrypted persistent storage drive on the USB stick where it runs. The existence of such persistent storage space will allow for user data to be placed, and ultimately, bootstrap scripts/software that can customize the operating system, accommodating the changes required for a DMVPN spoke to operate. This is covered in the section 4.3.
To solve the third point, a customized set of firewall rules is loaded, replacing the standard rule-set. This customized set of rules keep the standard set of rules, with the addition of allowing traffic required for the multipoint VPN.

Essentially, the DMVPN is supported by three software components: OpenNHRP, ipsec-tools, and racoon.

- OpenNHRP: Implementation of the NHRP protocol
- IPsec-tools: IPsec-related utilities
- Racoon: IKE key management, required for IPsec authentication

The main software for the DMVPN deployment is OpenNHRP. OpenNHRP can be used on both DMVPN hub and spokes. For a DMVPN spoke, the configuration is mainly composed by:

- IP address of the DMVPN hub
- Network settings of the VPN tunnel
- Authentication string: A string used as form of password that need to match with the hub configuration for successful registration
- Destination interface: In this case, the host itself, represented by the loopback interface

The (GRE) tunnel interface used for the VPN is configured separately. Example of configuration:

```
ip tunnel add gre1 mode gre key 1234 ttl 64
ip addr add 10.255.255.10/24 dev gre1
ip link set gre1 up
```

This configures a GRE tunnel interface, with IP address 10.255.255.10 and network mask 255.255.255.0.

Configuring ipsec-tools is a straight forward process: All GRE traffic should be enabled and Encapsulating Security Payload (ESP) should be used for packet authentication, which in the experimental setup was configured in transport mode. The full configuration for ipsec-tools, as well as for all other components can be seen in the appendix section.
The last main component of the DMVPN network is racoon. Racoon handles the security associations with the other hosts. Racoon can use either pre-shared keys or certificates for this purpose. In the deployed scenario, x509 certificates were used. A certificate authority (CA) was created and used to sign the hosts' certificates. For validation, each host not only has its own key and certificate, but also the CA certificate.

The most important details of the configuration are:

- Exchange mode: main
- NAT transversal enabled. Presence of NAT is automatically detected, allowing a spoke to operate behind NAT.

- Encryption algorithm for phase 1: AES
- Hash algorithm for phase 1: SHA-1
- Authentication method for phase 1: RSASIG

4.3 Bootstrap software

The possibility of storing data on the same USB stick where Tails Linux is installed, allow storing a bootstrap software that can easily configure and deploy a DMVPN spoke. In order to minimize time of deployment and user effort, the bootstrap software is modular, and is composed by the following:

- Bootstrap DMVPN spoke setup script
- Bootstrap DMVPN spoke start script
- Pre-made configuration files for rng-tools, OpenNHRP, racoon, ipsec-tools and ferm[12]

The setup script only has to be run once on each spoke. This script takes user input, where the network parameters are asked, as well as the DMVPN authentication secret. The pre-made configuration files contain place holders, where the spoke configuration will need to be present. As the user inserts the parameters, these place holders are filled with the right parameters, leading to at the end, a set of configuration files adjusted for the specific spoke.
The setup script also handles the key and certificate generation for the spoke. Prior to the key and certificate generation, rng-tools is installed and configured. Having rng-tools operational allows the key and certificate generation process to benefit from the added entropy from the RNG. It is advised then, that the RNG is plugged to the spoke machine before starting the setup script. In order to save time when actually deploying the DMVPN spoke, OpenHNRP, which is not available in the Tails Linux repositories, is downloaded and compiled. The source code and result of the compilation are stored in the persistent storage. When the deployment is performed, OpenNHRP only needs to be installed.

After successful configuration, the certificate generated will need to be signed by the CA, and the certificate of the CA will need to be copied to the persistent storage.
The most relevant part of the setup script output where user input is requested can be seen below, and a diagram of the tasks performed by the setup script can be seen on figure 5.

```
Insert IP address of the hub tunnel interface (ex. 10.255.255.1): 10.255.255.1

Insert IP address of the server/router that is the DMVPN hub: 145.100.104.48

Insert IP address for this spoke tunnel interface (ex. 10.255.255.10): 10.255.255.10

Insert netmask of the tunnel network (ex. 24): 24

Insert DMVPN authentication string: secret
```

[12] http://ferm.foo-projects.org

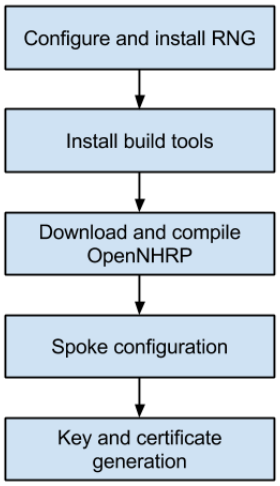

Figure 5: Tasks performed by the spoke setup script

When the spoke certificate is signed, and the CA certificate is included, the DMVPN spoke can finally be deployed. For this, the start script can be used.
The start script installs all software and deploy the configuration files on the Tails Linux system. No user interaction is required. The script starts by installing and configuring rng-tools, for RNG operation. After, it installs all remaining software and copies the configuration files. It also configures the GRE interface for the VPN. The process is finished by restarting all services, for the new configuration be loaded, and start the OpenNHRP software. A diagram of the tasks performed by can be seen on figure 6.

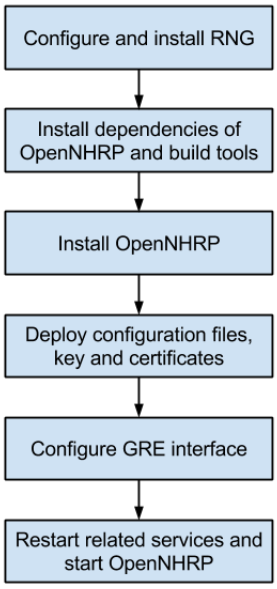

Figure 6: Tasks performed by the spoke start script

5 Results

5.1 True Random Number Generator

In order to verify the performance of the RNG and compare operation modes and filtering options, samples were collected and analysed. Even though the desired sample size was 512Kb, it was not possible to collect samples of this size for the operation modes where the jitter-based noise generator is used. The reason of this is the very low throughput of the jitter-based noise generator, lowering significantly the throughput of the RNG. On table 2, the throughput of the RNG for all modes and filtering options can be seen. Samples from /dev/urandom where collected without the RNG.

RNG	**Filtering**	**Throughput (byte/sec)**
Single PN junction	None	586.59
Single PN junction	XOR	587.12
Single PN junction	Von Neumann	208.24
Dual PN junction	None	383.39
Dual PN junction	XOR	375.59
Dual PN junction	Von Neumann	115.46
Jitter	None	0.85
Jitter	XOR	0.85
Jitter	Von Neumann	0.21
Single PN junction + jitter	None	0.85
Single PN junction + jitter	XOR	0.85
Single PN junction + jitter	Von Neumann	0.21
Dual PN junction + jitter	None	0.85
Dual PN junction + jitter	XOR	0.85
Dual PN junction + jitter	Von Neumann	0.22
/dev/urandom	n.a.	2730666.67

Table 2: Throughput of all operation modes and filtering options of the RNG, and /dev/urandom

As it can be observed, the performance of the RNG, when the jitter-based noise generator is used, is very low, specially when the Von Neumann filtering is used, which is expected to reduce the performance. Even using no filtering or the xor filtering, the amount of time needed to gather one 512Kb sample is well beyond the duration of this research, which is limited to five weeks. For this reason, 2504 byte samples where gathered for all operation modes and filtering options, as well as for /dev/urandom for comparison purposes. The reason of the sample size chosen is the minimum size required by rngtest. Rngtest splits data into 2500 byte samples (4 bytes are wasted) and analysis each individually. Having samples of this size, the rngtest can run one test on each sample. As will be seen later, 512Kb samples were also captured for the non jitter-based operation modes for better confidence in the results. Table 3 shows the results of the Ent tests for the 2504 byte samples.

RNG	Filtering	Entropy	Optimum compression	Chi Square (2504 samples)	Arithmetic mean	Monte Carlo Pi	Serial correlation
Single PN junction	None	7.920038	0%	252.70 (50.00%)	129.5843	3.069544365 (2.29%)	-0.016443
Single PN junction	XOR	7.902325	1%	322.84 (0.50%)	130.6138	3.050359712 (2.90%)	-0.012846
Single PN junction	Von Neumann	7.903012	1%	305.25 (2.50%)	129.4093	3.059952038 (2.60%)	-0.004145
Dual PN junction	None	7.909748	1%	290.12 (10.00%)	126.6058	3.155875300 (0.45%)	-0.034352
Dual PN junction	XOR	7.905288	1%	306.07 (2.50%)	128.8694	3.098321343 (1.38%)	0.010862
Dual PN junction	Von Neumann	7.920281	0%	257.20 (50.00%)	129.9812	3.223021583 (2.59%)	-0.013156
Jitter	None	7.907869	1%	290.73 (10.00%)	130.4720	3.031175060 (3.51%)	0.007691
Jitter	XOR	7.919676	1%	260.68 (50.00%)	128.6118	3.031175060 (3.51%)	0.023880
Jitter	Von Neumann	7.919558	1%	255.36 (50.00%)	129.3962	3.194244604 (1.68%)	0.024030
Single PN junction + jitter	None	7.908752	1%	288.69 (10.00%)	128.5555	3.117505995 (0.77%)	0.008573
Single PN junction + jitter	XOR	7.923098	0%	240.23 (50.00%)	127.3027	3.165467626 (0.76%)	-0.006640
Single PN junction + jitter	Von Neumann	7.909490	1%	299.73 (5.00%)	131.0276	3.184652278 (1.37%)	-0.020073
Dual PN junction + jitter	None	7.920467	0%	250.25 (50.00%)	125.5395	3.165467626 (0.76%)	-0.008814
Dual PN junction + jitter	XOR	7.924311	0%	242.68 (50.00%)	128.7999	3.107913669 (1.07%)	0.027533
Dual PN junction + jitter	Von Neumann	7.911900	1%	282.35 (25.00%)	130.8427	3.194244604 (1.68%)	0.000254
/dev/urandom	n.a.	7.932694	0%	228.37 (75.00%)	126.3586	3.050359712 (2.90%)	0.002745

Table 3: Ent tests of 2504 byte samples of all operation modes and filtering, and /dev/urandom

For better understanding of table 3, the results were color coded according to the following rules:

- Entropy: The best result has the color green and the weakest red. Yellow shows the intermediate results.
- Optimum compression: Green was used for 0% of compression and red for 1%, which shows data is compressible.
- Chi Square: For this test, color were used to show the differences, as indicated on the Ent manual[13]
  - Less than 1% or greater than 99%: Red, as the sequence is almost certainly not random
  - Between 1% and 5% or between 95% and 99%: Orange, as the sequence is suspect
  - Between 5% and 10% or between 90% and 95%: Yellow, as the sequence is almost suspect
  - Between 10% and 90%: Green
- Arithmetic mean: The best result has the color green and the weakest red. Yellow shows the intermediate results.
- Monte Carlo Pi: The best result has the color green and the weakest red. Yellow shows the intermediate results.
- Serial Correlation: The best result has the color green and the weakest red. Yellow shows the intermediate results.

[13] http://www.fourmilab.ch/random/

The results of each test will be analyzed individually.

Entropy

Most of the operation modes produced similar results, all slightly below the pseudo random number generator /dev/urandom. The operation settings that produced the best results are single and dual PN junctions with jitter and xor filtering. The setting with two PN junctions produced the best result.

Optimum compression

The results of this test are rather inconclusive. All performed similar, having the possibility of compression up to 1%.

Chi Square

Most of the operation modes and filtering performed well in this test. Exceptions are when the jitter-based noise generator was not used and the filtering applied is xor. Also, when the Von Neumann filtering was used in combination with a single PN junction or a single PN junction and jitter, the results where rather poor.

Arithmetic mean

In this test, the best setting was single PN junction with jitter and xor filtering, producing a very interesting result, very close to the ideal value, 127.5. Even /dev/urandom shown a lower performance on this test.

Monte Carlo Pi

Most of the settings produced a better results then /dev/urandom in this test, particularly the dual PN junction mode without filtering. The modes where the jitter-based noise generator was combined with either one or two PN junctions, produced consistently good results.

Serial Correlation

In this test, only one mode produced a result better than /dev/urandom, that is dual PN junctions with jitter and Von Neumann filtering. This is also the most complex mode, where the input data is taken from the three noise generators and the more complex Von Neumann filtering is applied. Data manipulation seems to be key in this test.

As written above, even though it was not possible to obtain 512 Kb samples of the RNG when the jitter-based noise generator was used, samples of this size for the other modes were taken. In this situation, it is also possible to compare results of the tests when samples of different sizes are used. Table 4 show the results obtained for the 512Kb samples. The colors used on table 4 have the same meaning as the color used on table 3.

RNG	Filtering	Entropy	Optimum compression	Chi Square (524288 samples)	Arithmetic mean	Monte Carlo Pi	Serial correlation
Single PN junction	None	7.987673	0%	4868.18 (0.01%)	128.0038	3.145489294 (0.12%)	0.000724
Single PN junction	XOR	7.987554	0%	4957.42 (0.01%)	128.1143	3.123058789 (0.59%)	-0.001192
Single PN junction	Von Neumann	7.988300	0%	4410.29 (0.01%)	128.8105	3.119213559 (0.71%)	0.000148
Dual PN junction	None	7.988324	0%	4392.78 (0.01%)	128.2537	3.137615729 (0.13%)	0.000895
Dual PN junction	XOR	7.988355	0%	4370.00 (0.01%)	128.2837	3.131390119 (0.32%)	0.000916
Dual PN junction	Von Neumann	7.988333	0%	4386.37 (0.01%)	128.3335	3.129467504 (0.39%)	-0.001960
/dev/urandom	n.a.	7.999602	0%	288.92 (10.00%)	127.3549	3.140408098 (0.04%)	-0.000704

Table 4: Ent tests of 512Kb samples of all operation modes and filtering, excluding jitter, and /dev/urandom

When using the bigger samples, all settings produced very similar results in most of the tests. Exceptions were the Monte Carlo Pi and Serial Correlation tests. The results of the Monte Carlo Pi test shows that either when using one or two PN junctions, the absence of filtering produces better results. This result is consistent for the dual PN junction without filtering setting, analysed previously, but not for the single PN junction mode, which produces a very different results when the smaller sample was used.

Subjecting the 2504 byte samples to the FIPS 140-2 test, performed by rngtest, all samples were successful. As written above, in the case of this sample size, only one test is performed per sample, due to the 2500 byte block size. Table 5 shows the result of the same test for the 512Kb samples.

RNG	Filtering	Number of Successes	Number of Failures	Percentage of Success
Single PN junction	None	199	10	95.22%
Single PN junction	XOR	206	3	98.56%
Single PN junction	Von Neumann	209	0	100.00%
Dual PN junction	None	209	0	100.00%
Dual PN junction	XOR	209	0	100.00%
Dual PN junction	Von Neumann	209	0	100.00%
/dev/urandom	n.a.	209	0	100.00%

Table 5: FIPS 140-2 tests on 512Kb samples of operation modes and filtering, excluding jitter, and /dev/urandom

The tests performed show that, besides the single PN junction mode when using no filtering or xor filtering, the RNG is capable of producing a byte stream that for the sample size analyzed, 100% of blocks pass the test. It is, of course, possible that for larger sample sizes, the result might not be 100%, but from these tests, it is possible to predict a good quality byte stream. Rng-tools, which receives the byte stream from the RNG, performs the FIPS 140-2 test on the data received before feeding the entropy pool of the Linux kernel. This means that not all data received from the RNG when it is operating on the single PN junction mode without filtering, or with xor filtering, would be used. The byte blocks that fail the test are discarded.

By combining the results of the 512Kb samples and throughput, it is possible to obtain a better overview of the results. Table 6 shows this combination. The columns "Ent Green", "Ent Yellow", and "Ent Red", are related to the color code used in table 4, describing the best, average and the weakest score(s), respectively.

RNG	Filtering	Ent Green	Ent Yellow	Ent Red	FIPS 140-2	Throughput (byte/sec)
Single PN junction	None	1	4	1	95.22%	586.59
Single PN junction	XOR	1	3	2	98.56%	587.12
Single PN junction	Von Neumann	2	1	3	100%	208.24
Dual PN junction	None	1	4	1	100%	383.39
Dual PN junction	XOR	1	4	1	100%	375.59
Dual PN junction	Von Neumann	1	4	1	100%	115.46
/dev/urandom	n.a.	4	1	9	100%	2730666.67

Table 6: Results of the 512Kb samples combined with the throughput

The single PN junction settings, either with no filtering or with xor filtering, provide the best throughput, even having in mind that not all data will be used to feed the entropy pool, as those settings achieved a sub-100% mark on the FIPS 140-2 tests. Right next on the throughput capacity, the dual PN junction mode with no filtering or with XOR filtering achieved 100% on the FIPS 140-2 test and still are capable of a relatively high throughput. Also, both performed similarly on the Ent tests.
When the Von Neumann filtering is used, both single and dual PN junction performed well on the FIPS 140-2 test, obtaining a 100% success rate. These, however are capable of less throughput when comparing with the other filtering options. The results for the Ent tests were mixed, where the dual PN junction mode produced average results, when comparing with the other settings, and the single PN junction mode produced good results in two tests, but weak results in three.

5.2 Multipoint VPN

Running the proof-of-concept software has shown it is possible to deploy a DMVPN spoke with minimum effort. The time taken for deployment is heavily influenced by the hardware specs of the machine and by the speed of the Internet connection. Even a non technical user could configure a DMVPN spoke, just by inserting the network configuration and secret as told by an administrator. In case a user receives a USB stick with Tails and the DMVPN spoke already configured, then all that is needed is to run the start script and wait for the procedure to finish, as user interaction is not required.

As it can be expected, the performance tests show a speed degradation when the multipoint VPN is in use. Table 7 shows the bandwidth achieved when using the multipoint VPN to communicate between two spokes over a common ADSL home connection and over a 1Gb link. For these tests, iperf[14] version 2 with the default settings was used.

Connection type	Multipoint VPN	Direct
Home connection	682 Kbit/s	743 Kbit/s
1Gb link	254 Mbit/s	949 Mbit/s

Table 7: Result of the bandwidth tests using iperf version 2.

The result of the bandwidth measurements show a high performance degradation when the tunnel is used to communicate between two hosts over the 1Gb link. However, over the domestic connection, the results were very similar, showing that there are other factors contributing to the difference, besides the overhead of the VPN.

[14] https://iperf.fr/

6 Conclusion

The outcome of this research shows that the scenario proposed is feasible and capable of achieving good results.
To start, it is possible to conclude about the performance and throughput of the RNG in the different settings. The results of the throughput of the RNG, when the samples were collected, show that the operation modes where the jitter-based noise generator is used, the throughput is very low, making the contribution of the RNG to the kernel entropy pool significantly smaller when comparing to the operation modes where this noise generator is disabled. For this reason, only operation modes without the jitter-based noise generator will be considered.
Rng-tools, which receives random bytes from the RNG, performs the FIPS 140-2 test before feeding the kernel entropy pool. Because of this, a RNG setting that produces high throughput, but does not have a probability of success on the FIPS 140-2 test close to 100%, might not contribute as much to the kernel entropy pool as a setting that has a smaller throughput, but has a very high probability of success. This was not, however, the case for the settings where 100% of success was not achieved, since the throughput was high enough to make up for that. On the other hand, because having blocks being discarded will have a negative consequence when the system is in need of entropy, the settings with 100% of success rate are preferred. The setting for which 100% of success was obtained and has the highest throughput, is dual PN junction without filtering, being closely followed by dual PN junction with xor filtering.
When looking at the results of the remaining tests, comparing the dual PN junction mode with xor filtering and without filtering, the setting where filtering is not used, provided better results on the entropy, Chi square, arithmetic mean and Monte Carlo tests, when looking at the results of the 2504 byte samples. If comparing the results of the 512Kb samples, there is an inconsistency on the entropy test, where the setting with xor filtering performed better. Still, the setting without filtering performed better in the majority of the tests, making the dual PN junction without filtering overall, the best setting found for the RNG implemented. Different users might however, value more certain tests than others, and some users might make a different choice on the operation setting for the RNG.

The bootstrap software developed to deploy the multipoint software on Tails Linux has proven to allow, very likely even non-technical users, to deploy a DMVPN spoke easily. A company can either distribute Tails Linux with the software pre-configured (run spoke setup beforehand, have the certificate signed and include the CA certificate), or it can simply tell a user where to get the software and provide the network setting and authentication string, to be inserted by the user when running the spoke setup script. In this situation, the user would then have to provide the certificate generated to a system administrator to have it signed by the CA, as well as obtain the CA certificate, required for validation of other certificates signed by the same CA.
While there was a high bandwidth loss when spokes were connected via a 1Gb link, the same was not observed when a regular domestic ADSL connection was used. Taking into account that the end-users would be dispersed around the world, the connection type mostly used would likely be similar to the second type, a standard ADSL connection. In this case, the overhead of operating on such VPN will not be significant. Users can then have a more secure connection between them, without compromising the usability.

Considering the RNG and multipoint VPN as a whole, it can be concluded that both integrate well with each other, and while the DMVPN is the basis for better data confidentiality, the plug-and-play RNG provides a valuable contribution to the scenario by improving the quality of the encryption.

7 Future Work

The work done in in this research is a step towards data confidentiality on the Internet. There is, however, work that can be done to improve the proposed scenario and create alternative scenarios with the same purpose. Focusing on the scenario proposed, not only more testing could be done, but some improvements could be made. A company willing to use the RNG of this project, would have to manufacture it or have it manufactured, as using the RNG mounted on a breadboard is not practical. Having this in mind, a pcb design would be required. Also, the code of the RNG could be modified to exclude the logging serial interface and the jitter-based noise generator, as its performance is very low. The integration of the RNG with rng-tools can also be optimized. In this research, the default options of rng-tools were used, however, there seems to be room for improvement that could lead to a better sourced kernel entropy pool.

Going beyond what was already described, implementing the RNG with a faster microcontroller could lead to significant throughput enhancements, which is one of the weak points of the current RNG.
On the multipoint VPN, the configuration, primarily related IPsec might have some room for improvement.
The deployment software, at the moment, assumes that all operations run as expected. Error handling code could be a first addition to the bootstrap software. For large companies intending to use the prototype developed, a way to easily generate Tails USB pen drives with the spoke pre-configured, including certificates, could be developed.

8 Acknowledgement

This research project was done at KPMG, in Amstelveen, The Netherlands. I would like to sincerely thank my supervisors Ruud Verbij, Jarno Roos, and Martijn Sprengers for their great support during my research project and for the opportunity to do this project with them at KPMG.

References

[1] Susan Landau, "Making Sense from Snowden: What's Significant in the NSA Surveillance Revelations," IEEE Security & Privacy, vol. 11, no. 4, pp. 54-63, July-Aug., 2013 http://www.computer.org/csdl/mags/sp/2013/04/msp2013040054-abs.html

[2] Checkoway, Stephen, et al. "On the practical exploitability of Dual EC in TLS implementations." USENIX Security. Vol. 1. 2014. http://web.elastic.org/~fche/mirrors/www.cryptome.org/2014/03/DualECTLS.pdf

[3] True Random Number Generator http://en.wikipedia.org/wiki/Hardware_random_number_generator

[4] PN junction http://en.wikipedia.org/wiki/Pn_junction

[5] Paul Kohlbrenner and Kris Gaj. 2004. An embedded true random number generator for FPGAs. In Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays (FPGA '04). ACM, New York, NY, USA, 71-78. DOI=10.1145/968280.968292 http://doi.acm.org/10.1145/968280.968292

[6] Make your own True Random Number Generator 2 http://robseward.com/misc/RNG2/

[7] Hardware Random Bit Generator https://web.jfet.org/hw-rng.html

[8] Hardware Random Number Generator http://www.cryogenius.com/hardware/rng/

[9] Ring Oscillator http://en.wikipedia.org/wiki/Ring_oscillator

[10] Sunar, B.; Martin, W.J.; Stinson, D.R., "A Provably Secure True Random Number Generator with Built-In Tolerance to Active Attacks," Computers, IEEE Transactions on , vol.56, no.1, pp.109,119, Jan. 2007 http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4016501&isnumber=4016489

[11] Cisco IOS DMVPN Overview http://www.cisco.com/c/dam/en/us/products/collateral/security/dynamic-multipoint-vpn-dmvpn/DMVPN_Overview.pdf

[12] R. Kalimuthu, Y. Kalley, M.L. Sullenberger, and J. Vilhuber. Method and apparatus fordynamically securing voice and other delay-sensitive network traffic, April 29 2008. URLhttps://www.google.com/patents/US7366894. US Patent 7,366,894

[13] Huaqi Chen, "Design and implementation of secure enterprise network based on DMVPN," Business Management and Electronic Information (BMEI), 2011 International Conference on , vol.1, no., pp.506,511, 13-15 May 2011 doi: 10.1109/ICBMEI.2011.5916984 http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5916984&isnumber=5916840

[14] Luca Deri and Richard Andrews. N2n: A layer two peer-to-peer vpn. Resilient Networks and Services, volume 5127 of Lecture Notes in Computer Science, pages 53-64. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-70586-4. http://dx.doi.org/10.1007/978-3-540-70587-1_5

[15] Avalanche noise http://en.wikipedia.org/wiki/Avalanche_diode

[16] Entropy Library https://sites.google.com/site/astudyofentropy/project-definition/timer-jitter-entropy-sources/entropy-library

[17] Biebighauser, Dan. "Testing Random Number Generators." University of Minnesota (2000). http://math.umn.edu/~garrett/students/reu/pRNGs.pdf

[18] M. Hnon, "The Monte Carlo method", Astrophysics and Space Science, November 1971, Volume 14, Issue 1, pp 151-167 http://dx.doi.org/10.1007/BF00649201

9 Appendix

9.1 RNG code

arduino-rng.ino

```
/*******************************/
/*  Rob Seward 2008-2009       */
/*  v1.0                       */
/*  4/20/2009                  */
/*                             */
/*  Extended by Daniel Romao   */
/*  6/12/2015                  */
/*******************************/

#include <SoftwareSerial.h>
#include <Entropy.h>

SoftwareSerial rngSerial(10, 11);

#define BINS_SIZE 256
#define CALIBRATION_SIZE 50000

#define NO_BIAS_REMOVAL 0
#define EXCLUSIVE_OR 1
#define VON_NEUMANN 2

#define ASCII_BYTE 0
#define BINARY 1
#define ASCII_BOOL 2
#define BYTE 3

#define INTERNAL_ONLY 0
#define SINGLE 1
#define DUAL 2

/***  Configure the RNG **************/
int rng_mode = DUAL;
int rng_internal = false;
int bias_removal = EXCLUSIVE_OR;
int output_format = BYTE;
int baud_rate = 19200;
/************************************/

unsigned int bins[2][BINS_SIZE];
int adc_pins[] = {4, 5};
int led_pin = 13;
boolean initializing = true;
unsigned int calibration_counter = 0;
byte threshold[2];


void setup(){
  pinMode(led_pin, OUTPUT);
  Serial.begin(baud_rate);

  for (int i=0; i < rng_mode; i++){
      clear_bins(i);
  }

  if (rng_internal){
    Entropy.initialize();
  }

  rngSerial.begin(baud_rate);
  log_starting();
}

void loop(){

  byte adc_byte[2];
  int adc_value;
  int i;

  for (i=0; i < rng_mode; i++){
    adc_value = analogRead(adc_pins[i]);
    adc_byte[i] = adc_value >> 2;
  }

  if (calibration_counter >= CALIBRATION_SIZE){

    for (i=0; i < rng_mode; i++){
      threshold[i] = findThreshold(i);
      clear_bins(i);
    }

    calibration_counter = 0;
    initializing = false;
  }

  for (i=0; i < rng_mode; i++){
    calibrate(adc_byte[i], i);
  }
```

```
  if (rng_mode != INTERNAL_ONLY){
    calibration_counter++;
  }

  if (initializing == false || rng_mode == INTERNAL_ONLY){
    processInput(adc_byte, threshold);
  } else {
      printStatus();
    }
}

void processInput(byte adc_byte[], byte threshold[]){
  boolean input_bool[2];
  boolean input_bool_after;
  int i;

  for (i=0; i < rng_mode; i++){
    input_bool[i] = (adc_byte[i] < threshold[i]) ? 1 : 0;
  }

  if (rng_mode == DUAL){
    input_bool_after = input_bool[0] ^ input_bool[1];
  } else if (rng_mode == SINGLE) {
      input_bool_after = input_bool[0];
    }

  if (rng_internal){
    uint8_t bool_internal = Entropy.random(2);
    if (rng_mode == INTERNAL_ONLY){
      input_bool_after = bool_internal;
    } else {
        input_bool_after ^= bool_internal;
      }
  }

  switch(bias_removal){
    case VON_NEUMANN:
      vonNeumann(input_bool_after);
      break;
    case EXCLUSIVE_OR:
      exclusiveOr(input_bool_after);
      break;
    case NO_BIAS_REMOVAL:
      buildByte(input_bool_after);
      break;
  }
}


void buildByte(boolean input){
  static int byte_counter = 0;
  static byte out = 0;

  if (input == 1){
    out = (out << 1) | 0x01;
  }
  else{
    out = (out << 1);
  }
  byte_counter++;
  byte_counter %= 8;
  if(byte_counter == 0){
    if (output_format == ASCII_BYTE) rngSerial.println(out, DEC);
    if (output_format == BINARY) rngSerial.print(out, BIN);
    if (output_format == BYTE) rngSerial.write(out);
    out = 0;
  }
  if (output_format == ASCII_BOOL) rngSerial.print(input, DEC);
}
```

aux_functions.ino

```
//Blinks an LED after each 10th of the calibration completes
void printStatus(){
  unsigned int increment = CALIBRATION_SIZE / 10;
  static unsigned int num_increments = 0; //progress units so far
  unsigned int threshold;

  threshold = (num_increments + 1) * increment;
  if(calibration_counter > threshold){
    num_increments++;
    Serial.print(F("*"));
    blinkLed();
  }
}

void blinkLed(){
  digitalWrite(led_pin, HIGH);
  delay(30);
  digitalWrite(led_pin, LOW);
}

void clear_bins(int ng){
  int i;
  for(i=0; i < BINS_SIZE; i++){
      bins[ng][i]=0;
  }
```

```
}
```

calibration.ino

```
void calibrate(byte adc_byte, int ng){
  bins[ng][adc_byte]++;
}

unsigned int findThreshold(int ng){
  unsigned long half;
  unsigned long total = 0;
  int i;

  for(i=0; i < BINS_SIZE; i++){
    total += bins[ng][i];
  }

  half = total >> 1;
  total = 0;
  for(i=0; i < BINS_SIZE; i++){
    total += bins[ng][i];
    if(total > half){
      break;
    }
  }

  print_threshold(ng, i);

  return i;
}
```

filtering.ino

```
void exclusiveOr(byte input){
  static boolean flip_flop = 0;
  flip_flop = !flip_flop;
  buildByte(flip_flop ^ input);
}

void vonNeumann(byte input){
  static boolean previous = 0;
  static boolean flip_flop = 0;

  flip_flop = !flip_flop;

  if(flip_flop){
    if(input == 1 && previous == 0){
      buildByte(0);
    }
    else if (input == 0 && previous == 1){
      buildByte(1);
    }
  }
  previous = input;
}
```

logging.ino

```
void log_starting(){
  Serial.println(F("\nTRNG Starting..."));

  Serial.print(rng_mode);
  Serial.println(F(" external noise generator(s) will be used"));

  Serial.print(F("The internal noise generator is "));

  if (rng_internal) {
    Serial.println(F("enabled"));
  } else {
      Serial.println(F("disabled"));
    }

  Serial.print(F("Filtering applied: "));

   switch(bias_removal){
    case VON_NEUMANN:
      Serial.println(F("von Neumann"));
      break;
    case EXCLUSIVE_OR:
      Serial.println(F("xor"));
      break;
    case NO_BIAS_REMOVAL:
      Serial.println(F("none"));
      break;
  }

  Serial.print(F("The output format is "));

  switch(output_format){
    case ASCII_BYTE:
      Serial.println(F("ASCII byte"));
      break;
    case BINARY:
      Serial.println(F("binary"));
      break;
```

```
    case ASCII_BOOL:
      Serial.println(F("ASCII boolean"));
      break;
    case BYTE:
      Serial.println(F("byte"));
      break;
  }
}

void print_threshold(int ng, int threshold){
  Serial.print(F("\nThe threshold "));
  Serial.print(ng+1);
  Serial.print(F(" is: "));
  Serial.println(threshold);
}
```

9.2 DMVPN spoke bootstrap code

Spoke setup

```
#!/bin/bash
# Intial setup script
# Tested with Tails 1.4

# Configure serial interface
stty -F /dev/ttyUSB0 19200 clocal cs8 -cstopb -parenb

#Install rng-tools and configure RNG
apt-get update
apt-get install rng-tools -y

cp files/configuration/rng-tools /etc/default/rng-tools
service rng-tools restart

# Download build tools and dependecies
apt-get install build-essential libc-ares-dev pkg-config -y

# Download and compile OpenNHRP
wget http://downloads.sourceforge.net/project/opennhrp/opennhrp/opennhrp-0.14.1.tar.bz2
tar xf opennhrp-0.14.1.tar.bz2
cd opennhrp-0.14.1
make
cd ..
rm opennhrp-0.14.1.tar.bz2

# Configure Hub
echo -e "\nInsert IP address of the hub tunnel interface (ex. 10.255.255.1): \c"
read TUNNEL_HUB_IP

sed -i "s/TUNNEL_HUB_IP/$TUNNEL_HUB_IP/g" files/configuration/opennhrp.conf

echo -e "\nInsert IP address of the server/router that is the DMVPN hub: \c"
read HUB_IP

sed -i "s/HUB_IP/$HUB_IP/g" files/configuration/opennhrp.conf
sed -i "s/HUB_IP/$HUB_IP/g" files/configuration/ferm.conf

# Configure Spoke
echo -e "\nInsert IP address for this spoke tunnel interface (ex. 10.255.255.10): \c"
read TUNNEL_SPOKE_IP

sed -i "s/TUNNEL_SPOKE_IP/$TUNNEL_SPOKE_IP/g" spoke-start.sh

# Configure netmask
echo -e "\nInsert netmask of the tunnel network (ex. 24): \c"
read TUNNEL_NETMASK

sed -i "s/TUNNEL_NETMASK/$TUNNEL_NETMASK/g" files/configuration/opennhrp.conf
sed -i "s/TUNNEL_NETMASK/$TUNNEL_NETMASK/g" spoke-start.sh

#Configure secret
echo -e "\nInsert DMVPN authentication string: \c"
read SECRET

sed -i "s/SECRET/$SECRET/g" files/configuration/opennhrp.conf

# Create keys
echo -e "\nA key and certificate will be created now for this spoke\n"

openssl genrsa -des3 -out key_encrypted.key 4096
openssl rsa -in key_encrypted.key -out key.pem
openssl req -new -key key.pem -out cert.csr

mkdir -p files/certs
mv key.pem files/certs/key.pem
mv cert.csr files/certs/cert.csr
rm key_encrypted.key

echo -e "\nEnd of configuration!\n"
echo "The file ./files/certs/cert.csr will need to be signed using the Root CA key and certificate."
echo "The file name of the certificate should be: cert.pem"
echo -e "The certificate of the CA will also have to be included in the certs directory. The file should be named ca.pem\n"
```

Spoke start

```
#!/bin/bash
# Deployment script
# Tested with Tails 1.4

export DEBIAN_FRONTEND=noninteractive

# Configure serial interface
stty -F /dev/ttyUSB0 19200 clocal cs8 -cstopb -parenb

#Install rng-tools and configure RNG
apt-get update
apt-get install rng-tools -y

cp files/configuration/rng-tools /etc/default/rng-tools
service rng-tools restart

# Install dependencies and build tools
apt-get install racoon ipsec-tools build-essential libc-ares-dev pkg-config -y

# Install OpenNHRP
cd opennhrp-0.14.1
make install
cd ..

# Copy configuration files
cp files/configuration/opennhrp.conf /etc/opennhrp/opennhrp.conf
cp files/configuration/racoon.conf /etc/racoon/racoon.conf
cp files/configuration/ipsec-tools.conf /etc/ipsec-tools.conf
cp files/configuration/ferm.conf /etc/ferm/ferm.conf

# Copy keys' directory
cp files/certs/* /etc/racoon/certs/

# Load GRE kernel module
modprobe ip_gre

# Create GRE interface
ip tunnel add gre1 mode gre key 1234 ttl 64
ip addr add TUNNEL_SPOKE_IP/TUNNEL_NETMASK dev gre1
ip link set gre1 up

# Restart services
service rng-tools restart
service racoon restart
service setkey restart
service ferm restart

# Start OpenNHRP
/usr/sbin/opennhrp -d

echo -e "\nCompleted!\n"
```

ferm.conf

```
# -*- mode: conf[space] -*-
#
#  Configuration file for ferm(1).
#

# I2P rules that grant access to the "i2psvc" user (those with $use_i2p) will
# only be enabled if the string "i2p" is entered at the boot prompt.
# Deny or reject rules affecting "i2psvc" will always be set.
def $use_i2p = `test -d /usr/share/i2p && echo 1 || echo 0`;

# IPv4
domain ip {
    table filter {
        chain INPUT {
            policy DROP;

            # Established incoming connections are accepted.
            mod state state (RELATED ESTABLISHED) ACCEPT;

            # Traffic on the loopback interface is accepted.
            interface lo ACCEPT;

            # Allow GRE traffic
            proto gre ACCEPT;

            # Allow all traffic on the tunnel interface
            interface gre1 ACCEPT;
        }

        chain OUTPUT {
            policy DROP;

            # Established outgoing connections are accepted.
            mod state state (RELATED ESTABLISHED) ACCEPT;

            # White-list access to local resources
            outerface lo {
                # White-list access to Tor's SOCKSPort's
                daddr 127.0.0.1 proto tcp syn dport 9050 {
                    mod owner uid-owner root ACCEPT;
                    mod owner uid-owner proxy ACCEPT;
```

```
        mod owner uid-owner nobody ACCEPT;
    }
    daddr 127.0.0.1 proto tcp syn mod multiport destination-ports (9050 9061 9062 9150) {
        mod owner uid-owner amnesia ACCEPT;
    }
    daddr 127.0.0.1 proto tcp syn dport 9062 {
        mod owner uid-owner htp ACCEPT;
        mod owner uid-owner tails-iuk-get-target-file ACCEPT;
        mod owner uid-owner tails-upgrade-frontend ACCEPT;
    }

    # White-list access to Tor's ControlPort
    daddr 127.0.0.1 proto tcp dport 9051 {
        mod owner uid-owner tor-launcher ACCEPT;
        # Needed by a workaround in tordate (NM's 20-time.sh hook)
        # for temporarily changing Tor's logging severity.
        mod owner uid-owner root ACCEPT;
    }

    # White-list access to the Tor control port filter
    daddr 127.0.0.1 proto tcp dport 9052 {
        mod owner uid-owner amnesia ACCEPT;
    }

    # White-list access to Tor's TransPort
    daddr 127.0.0.1 proto tcp dport 9040 {
        mod owner uid-owner amnesia ACCEPT;
    }

    # White-list access to system DNS and Tor's DNSPort
    daddr 127.0.0.1 proto udp dport (53 5353) {
        mod owner uid-owner amnesia ACCEPT;
    }

    # Whitelist access to Tor's DNSPort so I2P can resolve hostnames when bootstrapping
    daddr 127.0.0.1 proto udp dport 5353 {
        @if $use_i2p mod owner uid-owner i2psvc ACCEPT;
    }

    # White-list access to ttdnsd
    daddr 127.0.0.2 proto udp dport 53 {
        mod owner uid-owner amnesia ACCEPT;
    }
    daddr 127.0.0.2 proto tcp syn dport 53 {
        mod owner uid-owner amnesia ACCEPT;
    }

    # White-list access to I2P services for the amnesia user (IRC, SAM, POP3, SMTP, and Monotone)
    # For more information, see https://tails/boum.org/contribute/design/I2P and https://geti2p.net/ports
    daddr 127.0.0.1 proto tcp syn mod multiport destination-ports (6668 7656 7659 7660 8998) {
        @if $use_i2p mod owner uid-owner amnesia ACCEPT;
    }

    # Whitelist access to I2P services for the i2psvc user,
    # otherwise mail and eepsite hosting won't work. The mail ports (7659 and 7660) are
    # accessed by the webmail app
    daddr 127.0.0.1 proto tcp syn mod multiport destination-ports (7658 7659 7660) {
        @if $use_i2p mod owner uid-owner i2psvc ACCEPT;
    }

    # Whitelist access to the i2pbrowser user
    daddr 127.0.0.1 proto tcp syn mod multiport destination-ports (4444 7657 7658) {
        @if $use_i2p mod owner uid-owner i2pbrowser ACCEPT;
    }

    # White-list access to the java wrapper's (used by I2P) control ports
    # (see: http://wrapper.tanukisoftware.com/doc/english/prop-port.html)
    # If, for example, port 31000 is in use, it'll try the next one in sequence.
    daddr 127.0.0.1 proto tcp sport (31000 31001 31002) dport (32000 32001 32002) {
        @if $use_i2p mod owner uid-owner i2psvc ACCEPT;
    }

    # White-list access to CUPS
    daddr 127.0.0.1 proto tcp syn dport 631 {
        mod owner uid-owner amnesia ACCEPT;
    }

    # White-list access to Monkeysphere
    daddr 127.0.0.1 proto tcp syn dport 6136 {
        mod owner uid-owner amnesia ACCEPT;
    }
}

# clearnet is allowed to connect to any TCP port via the
# external interfaces (but lo is blocked so it cannot interfere
# with Tor etc) including DNS on the LAN. UDP DNS queries are
# also allowed.
outerface ! lo mod owner uid-owner clearnet {
    proto tcp ACCEPT;
    proto udp dport domain ACCEPT;
}

# Local network connections should not go through Tor but DNS shall be
# rejected. I2P is explicitly blocked from communicating with the LAN.
# (Note that we exclude the VirtualAddrNetwork used for .onion:s here.)
daddr (10.0.0.0/8 172.16.0.0/12 192.168.0.0/16) @subchain "lan" {
    proto tcp dport domain REJECT;
```

```
                proto udp dport domain REJECT;
                mod owner uid-owner i2psvc REJECT;
                ACCEPT;
            }

            # Tor is allowed to do anything it wants to.
            mod owner uid-owner debian-tor ACCEPT;

            # i2p is allowed to do anything it wants to on the internet.
            outerface ! lo mod owner uid-owner i2psvc {
                @if $use_i2p proto (tcp udp) ACCEPT;
            }

            # Allow IPSEC + GRE
            proto udp dport (500 4500) daddr HUB_IP ACCEPT;
            proto esp daddr HUB_IP ACCEPT;
            proto gre ACCEPT;

            # Allow all traffic on the tunnel interface
            outerface gre1 ACCEPT;

            # Everything else is logged and dropped.
            LOG log-prefix "Dropped outbound packet: " log-level debug log-uid;
            REJECT reject-with icmp-port-unreachable;
        }

        chain FORWARD {
            policy DROP;
        }
    }

    table nat {
        chain PREROUTING {
            policy ACCEPT;
        }

        chain POSTROUTING {
            policy ACCEPT;
        }

        chain OUTPUT {
            policy ACCEPT;

            # .onion mapped addresses redirection to Tor.
            daddr 127.192.0.0/10 proto tcp REDIRECT to-ports 9040;

            # Redirect system DNS to Tor's DNSport
            daddr 127.0.0.1 proto udp dport 53 REDIRECT to-ports 5353;
        }
    }
}

# IPv6:
domain ip6 {
    table filter {
        chain INPUT {
            policy DROP;
        }

        chain FORWARD {
            policy DROP;
        }

        chain OUTPUT {
            policy DROP;
            # Everything else is logged and dropped.
            LOG log-prefix "Dropped outbound packet: " log-level debug log-uid;
            REJECT reject-with icmp6-port-unreachable;
        }
    }
}
```

ipsec-tools.conf

```
#!/usr/sbin/setkey -f

spdflush;
   spdadd 0.0.0.0/0 0.0.0.0/0 gre -P out ipsec esp/transport//require;
   spdadd 0.0.0.0/0 0.0.0.0/0 gre -P in ipsec esp/transport//require;
```

opennhrp.conf

```
interface gre1
  holding-time 3600
  map TUNNEL_HUB_IP/TUNNEL_NETMASK HUB_IP register
  cisco-authentication SECRET
  shortcut
  redirect
  multicast dynamic
  non-caching

interface lo
  shortcut-destination
```

racoon.conf

```
path certificate "/etc/racoon/certs";
remote anonymous {
        exchange_mode main;
        lifetime time 2 hour;
        certificate_type x509 "/etc/racoon/certs/cert.pem" "/etc/racoon/certs/key.pem";
        ca_type x509 "/etc/racoon/certs/ca.pem";
        my_identifier asn1dn;
        nat_traversal on;
        script "/etc/opennhrp/racoon-ph1dead.sh" phase1_dead;
        dpd_delay 120;
        proposal {
                encryption_algorithm aes 256;
                hash_algorithm sha1;
                authentication_method rsasig;
                dh_group modp4096;
        }
        proposal {
                encryption_algorithm aes 256;
                hash_algorithm sha1;
                authentication_method rsasig;
                dh_group 2;
        }
}

sainfo anonymous {
        pfs_group 2;
        lifetime time 2 hour;
        encryption_algorithm aes 256;
        authentication_algorithm hmac_sha1;
        compression_algorithm deflate;
}
```

rng-tools

```
HRNGDEVICE=/dev/ttyUSB0
```

opennhrp.conf for the DMVPN Hub

```
interface gre1
  holding-time 3600
  multicast dynamic
  shortcut
  redirect
  non-caching
 cisco-authentication secret
```