Woodpecker, a Software-only True Random Generator for the CAN Bus

Tsvika Dagan and Avishai Wool

tdagan02@gmail.com, yash@eng.tau.ac.il
Tel Aviv University, Israel

***Abstract*—This paper describes *Woodpecker*, a software-only True-Random-Number-Generator (TRNG) for electronic control units (ECUs) connected to a vehicular CAN bus. *Woodpecker* follows the design of the Linux RNG mechanism (*LRNG*), and relies on the unpredictability of inter-arrival times of CAN messages as its primary source of input, mixed with the message IDs and message payloads.**

Our main contribution is demonstrating that despite the strong periodicity exhibited in CAN bus traffic, if time events are measured at a reasonably high fidelity, the inter-arrival times embed enough entropy to extract true random bits. We evaluated our method on vehicles from Jeep, Ford, and Subaru. Using a time measurement fidelity of 100μsec we extracted 634–770 entropy bits per second according to *LRNG* estimation. Using 1μsec fidelity we extracted 4944 entropy bits per second.

Beyond randomness within a single driving session, we evaluated the variability between different sessions. We compared the inter-arrival times captured in multiple sleep-to-ignition traces, taken in close succession in the same car, without any driver actions beyond pressing the ignition button. Even under such static conditions we found that about 25–30% of the inter-arrival times varied between traces.

Finally, we demonstrate that even if the attacker is connected to the same CAN bus and is measuring the inter-arrival times simultaneously with *Woodpecker* at the same 1μsec fidelity (twice as fast as the CAN bus speed of 500 Kbps), there is sufficient variation introduced by the measurement equipment to produce about 20% of differences between the *Woodpecker* and attacker traces.

I. Introduction

A. Motivation

Modern vehicles are susceptible to cyber-attacks: this is since they are controlled by multiple dedicated computers (electronic control units - ECUs) that are typically connected not only to each other (e.g., over a CAN bus) but also to the outside world—often by wireless protocols (WiFi, Bluetooth, Cellular, etc.). These conditions, and the introduction of new technologies, that allow remote access to the vehicle internal systems, make vehicles vulnerable to potential new attack vectors of increasing number. Researchers have already shown that these attacks can be both feasible and severe (e.g., attacks on Jeep [1], Tesla [2]) even when the vehicle engine is off (as shown by Cho et al. [3]).

Some of the defense mechanisms that were suggested to identify or block these attacks require a good source of random bits (e.g., to produce a nonce for a randomized cryptographic protocol, to enable secure key generation, etc.). The introduction of the V2X systems and their use of public key cryptography makes the need for strong random bits even more crucial as described by Henry et al. [4].

In standard computer systems random bits are typically gathered by the operating system (OS) in random-pools, and are used by the OS itself and by higher level applications. These pools are filled by a random generator (RNG) that is responsible for the quality (high entropy and non determinism) of the random bits. A typical RNG consists of two components: a TRNG—to gather high quality true random seeds, and a Pseudo-Random-Number-Generator (PRNG) that uses these seed as an input to generate a higher volume of pseudo-random bits. A good TRNG typically requires either special hardware, or a source of unpredictable environmental measurements, in order to produce non-deterministic, high-entropy bits at a reasonable rate. The PRNGs is generally a software implementation of a cryptographic primitive that expands a short true-random seed into a much longer pseudo-random sequence.

The growing need for cryptographically-secure random bits, together with the assumption that many ECUs do not incorporate a hardware-based TRNG, motivated us to look for a software only TRNG, based on the ECU's natural operating environment - the vehicle interconnected CAN bus.

B. Related Work

1) Vehicle Security: Research into vehicle cyber-security has been growing since the first publication of Koscher et al. [5] in 2010. Using sniffing, fuzzing and reverse engineering of ECU's code, the authors succeeded in controlling a wide range of vehicle functions, such as disabling the brakes, stopping the engine, etc.

Checkoway et al. [6] showed that a remote attack, without physical access to the vehicle, is also possible (via Bluetooth, cellular radio, etc.). Valasek and Miller [7] demonstrated actual attacks on Ford Escape and Toyota Prius cars via the CAN bus network. They affected the speedometer, navigation system, steering, braking and more. In 2015 it was reported [1], [8] that they remotely disabled a Jeep's brakes during driving, and caused Jeep to recall 1.4M vehicles. Foster and Koscher [9] have also reported of the potential vulnerabilities in relatively new commercial OBD-II dongles (such as those used by insurance companies to track one's driving) which support cellular communication and may be even exploited via SMS. In 2016, a team of researchers from Keen Security Lab demonstrated a successful attack on the Tesla electrical vehicle [2], taking control over the vehicle through a bug in the Infotainment unit's browser, forcing the company to release an over-the-air software update.

Several ideas were offered to secure vehicles against cyber-attacks, including both active and passive solutions. One approach is to try and secure the internal communication of the vehicle - typically a CAN bus, by adding authentication to the messages (e.g., by using a cryptographic Message Authentication Code (MAC)). Several ideas were suggested, ranging from adding part of a MAC tag to the actual message's data field, to splitting the MAC into several pieces and layers as offered by Glas and Lewis [10]. Van Herrewege et al. [11] suggested to use a new light-weight protocol to better fit the CAN bus limitations. Their *CANAuth* protocol, also relied on the *CAN+* protocol of Ziermann et al. [12], which allowed them to split the authentication bits in between the sampling points of the bus. A similar approach was adopted by the AUTOSAR standard, as defined by the Secure Onboard Communication (SecOC) mechanism [13], to add some authentication and replay prevention to the vehicle's internal networks. Note that all these cryptographic solutions require secret keys and/or random nonces—hence they rely on a good source of randomness to produce the keys.

A different, non-cryptographic, family of solutions is based on destroying non-legitimate spoofed messages. These include suggestions by Matsumoto et al. [14], Kurachi et al. [15], [16], Ujiie et al. [17], and the *Parrot* system of Dagan and Wool [18], [19].

Another approach is to try and identify un-authorized access to the internal network of the vehicle, by using Anomaly or Intrusion Detection Systems (IDS). Markovitz and Wool [20], [21] demonstrated the ability to classify the traffic over the CAN bus, where Marchetti et al. offered some anomaly detection mechanisms, based on an information theoretic algorithm [22] and on inspection of sequences of IDs [23]. Hamada et al. [24] offered to implement an IDS system that relies on the traffic density of some periodic messages.

Newer works offered to rely on some unique characteristics of the ECU to build an IDS for the CAN bus. Lee et al. [25] used the time of arrival of Remote-frames reply packets to identify potential attackers; whereas both Cho and Shin [26], and Choi et al. [27] used the voltage characteristics of an ECU to identify attacks.

Some leading manufacturers, such as NXP [28] and Bosch [29] offer a variety of products to secure the vehicles, ranging from Hardware Secure Modules (HSMs) to full fledged secure gateways. The existence of these products fits the wide-spreading holistic (in-depth / layered) approach for vehicle cyber-security, as described by Van Roermund et al. [30]—and many of them rely on a source of randomness.

2) Random Generation: All modern operating systems provide a standard API for a Pseudo Random Number Generator (PRNG), that is used mainly for cryptographic functions. Faults were found in the randomness properties of the PRNG both in Windows and in Linux [31], [32]. Faults or wrong usage of the OS PRNG can lead to severe security breaches in the cryptographic services provided by those operating systems (cf. [33]). Therefore modern OSes incorporate sources of *True Random* unpredictable events into their PRNG mechanisms.

Typical sources for randomness are based on events outside the CPU, such as hard-disk activity [34], network or user activity [35], accelerometer events [36], SRAM power-up effects [37], or CPU performance monitoring registers [38]. The CPU clock is also sometimes sampled during the system run time for added entropy (e.g., in the Windows CryptGenRandom() API function [39]) or as the sole entropy source of a random number generator [40]. Intel has designed a hardware TRNG (True Random Numbers Generator) [41] which may be incorporated into some of its CPUs.

Unfortunately these sources are not always available—and in particular most of them are not present in car ECUs. Besides hardware-based TRNGs that may be present in high-end secure modules ECUs, the only source of environmental unpredictability that is available to all ECUs is the network activity. Wan et al. [42] offered to use the wireless Channel Randomness to Generate Keys for Automotive Cyber-Physical System Security.

In this work, We followed the analysis that was offered by Gutterman et al. [32] - which analyzed the Linux RNG mechanism (*LRNG*), and adopt it for the CAN environment.

C. Contribution

Our starting point is the observation that a good source of randomness is required by ECUs for a growing

number of security tasks, and that many ECUs may lack a hardware-based TRNG. This leaves CAN bus activity as a possible source of entropy. However, CAN bus traffic is known to be highly periodic machine-to-machine traffic, and a-priori it was unclear how unpredictable it is. Our main contribution is demonstrating that despite the strong periodicity exhibited in CAN bus traffic, if time events are measured at a reasonably high fidelity, the inter-arrival times embed enough entropy to extract true random bits.

The *Woodpecker* mechanism follows the design of the Linux RNG (*LRNG*), and relies on the unpredictability of inter-arrival times of CAN messages as its primary source of input, mixed with the message IDs and message payloads.

We evaluated our method on vehicles from Jeep, Ford, and Subaru (having a 500Kbps bus). Using a time measurement fidelity of 100μsec, we extracted 634–770 entropy bits per second according to *LRNG* estimation. Using 1μsec fidelity, we extracted 4944 entropy bits per second.

Beyond randomness within a single driving session, we evaluated the variability between different sessions. We compared the inter-arrival times captured in multiple sleep-to-ignition traces, taken in close succession in the same car, without any driver actions beyond pressing the ignition button. Even under such static conditions we found that about 25–30% of the inter-arrival times varied between traces.

Finally, we demonstrate that even if the attacker is connected to the same CAN bus and is measuring the inter-arrival times simultaneously with *Woodpecker* at the same 1μsec fidelity (twice as fast as the CAN bus speed of 500 Kbps), there is still sufficient variation introduced by the measurement equipment to produce about 20% of differences between the *Woodpecker* and the attacker traces.

Organization: In the next section we describe some preliminaries. In Section III we introduce the *Woodpecker* mechanism. Section IV describes our testing and evaluation of the system. Section V describes some related problems and limitations, and offers some mitigations. We conclude with Section VI.

II. Preliminaries

A. The Linux RNG

The *LRNG*, as described by [32], uses several timed events (e.g., mouse movements, keyboard values, etc,) to update its pools of random bits, and measure its entropy gain.

Each gathered event is represented by two 32 bits words, which are added as an input to the *LRNG* main entropy pool. The first word includes the time of the event (in milliseconds, or CPU-cycles, since the last boot), and the second word includes the event's encoded value (8 bits for a keyboard press, 12 bits for a mouse movement, 3 bits for a disk event, and 4 bits for an interrupt).

Bits from the main entropy pool are fed into two LFSR-like pools (The mechanism for updating the pools is based on a TGFSR - Twisted Generalized Feedback Shift Register). The first pool is used for high-entropy random bits, and the second for lower quality bits (available through the `/dev/random` and the `/dev/urandom` devices, respectively).

The time difference in milliseconds between every two consecutive events is the basis for estimating the amount of entropy of the given event. Let t_{n-1}, t_n denote the times of the previous and current events, as 32-bit integers. Define

$$\begin{aligned} \delta_n &= t_n - t_{n-1} \\ \delta_n^2 &= \delta_n - \delta_{n-1} \\ \delta_n^3 &= \delta_n^2 - \delta_{n-1}^2 \end{aligned}$$

Note that both δ_n^2 and δ_n^3 may be negative, hence the use of their absolute value in the formula below. The entropy contribution of each event is defined as

$$\log_2(min(|\delta_n|, |\delta_n^2|, |\delta_n^3|)_{[19-30]}) \quad (1)$$

where $X_{[a-b]}$ denotes bits *a* to *b* of X (0 being the most significant bit). The entropy contribution is set to zero in case $min(|\delta_n|, |\delta_n^2|, |\delta_n^3|)_{[19-30]}$ is zero.

The entropy contribution of each event is added to an entropy counter. The counter is decremented by k, when k bits of random are extracted from the random pool.

B. CAN bus

The Controller Area Network (CAN) bus standard (developed by Robert Bosch GmbH [43]) is probably the most common protocol for in-vehicle communication. The protocol is a serial broadcast protocol which offers a reliable communication channel for the vehicle's Electronic Control Units (ECUs). The ECUs control the car's different subsystems (such as the engine control unit, the ABS system, etc). Modern vehicles typically have a few dozen ECUs.

Apart from the host processor, a typical ECU consists of a CAN controller, to implement and enforce the protocol. The controller is generally implemented by hardware, whereas the host processor is usually a microcontroller or full-fledged CPU running custom firmware and software.

Each CAN frame is identified by a message ID which is either 11 or 29 bits long (for *standard / extended-frame* format); However CAN messages do not carry an identifier of the destination: each ECU unilaterally

SOF	ID	RT	IDE	Res	DLC	Data	CRC	CDel	Ack	ADel	EOF
0	#11	0	0	0	#4	#DLC x 8	#15	1	#1	1	1111111

Fig. 1. A standard data frame, with an 11-bit ID and a 4-bit DLC (length) field. The most common case is of DLC=8, having 64 bits of data.

decides which message IDs to accept and act upon. Any ECU can monitor all the traffic that goes over the bus (including while it is transmitting).

The CAN protocol is a synchronous protocol, in which time is split into bit-time slots. The length of the slots depends on the (pre-configured) bus speed that can vary between 1Mbps (with a minimal slot of 1μsec) down to 250Kbps (having 4μsec slots).

Figure 1 describes a data frame in a *standard-frame* (11 bits ID) format, where the 4-bit DLC field describes the number of bytes (0-8) that the data-field should contain.

C. Adversary model

We present two adversary model variants that are relevant to different usages to our system, as further described in Section V. In both models we assume that the attacker is familiar with the *Woodpecker* implementation, and is familiar with the details and semantics of the CAN messages of the given vehicle. However, we assume the attacker has no direct control over the *Woodpecker* mechanism: Either the attacker cannot run code in *Woodpecker*'s ECU, or *Woodpecker* is protected by the ECU's OS or hardware, etc. We also assume that under both models the attacker has no ability to alter the time-stamp of the arriving messages, as provided by the *Woodpecker's* hosting ECU.

- **External adversary model:** In this model the adversary has no access to *Woodpecker*'s CAN bus: E.g., the attacker is external to the vehicle, or is connected to another bus, which is separated by some internal filtering gateway. The External attacker cannot observe the same traffic as of his potential victim.
- **Internal adversary model:** In this model the adversary has access to the same CAN bus as the *Woodpecker*, and has access to the same traffic.

III. The *Woodpecker* mechanism

A. Overview

The *Woodpecker* mechanism uses the CAN traffic as a source of unpredictability to gather, evaluate and update a pool of true random bits. *Woodpecker* is an adaptation of the *LRNG* to the specifics of the CAN bus: it uses the inter-arrival time between every two consecutive CAN messages as its primary source of randomness, to both estimate the gained entropy, and to update its pool of input seeds - the main entropy pool. The message ID and message Data are added to the pool of seeds as additional contributors.

As in the *LRNG*, *Woodpecker* maintains an entropy counter to estimate the number of random bits that is available in its pool. *Woodpecker* increases the counter on every received message (by its calculated entropy gain, see Section III-B), and reduces the counter whenever random bits are read from the pool (according to the number of extracted random bits).

The entropy pool is updated based on the following 32-bit words of each received CAN message:

- TimeStamp1,TimeStamp2: The current message's time of arrival, as a 64-bit quantity split into two 32-bit words[1], measured from the last boot-time of the hosting device, using the best available fidelity of the device, e.g., in μsec.
- messageID - The received message's ID, padded with leading zeros.
- messageData1, messageData2 - Usually the 64-bit Data field of the received message, or the Data field padded with leading zeros in case of shorter messages.

We note that both the message ID and Data are optional, and depend on *Woodpecker*'s ability to gather this data (see Section V-A for more details).

An additional processing step is required before extracting the actual random bits from the main entropy pool for external usage. This step can include a similar process as defined by the *LRNG*, or any other secured process (e.g., to include some LFSR and a hash function). The details of extracting randomness from the main entropy pool are out of the scope of this paper, whose main focus is on the evaluation of the amount of randomness in the main entropy pool.

Figure 2 describes the general mechanism of the *Woodpecker*.

B. Entropy Estimation

Woodpecker's entropy estimation measurements are based on the *LRNG* mechanism, as described in Section II-A, where the time difference between every two consecutive CAN messages replaces the *LRNG* time difference (δ_n) between every two gathered events (e.g., mouse movements).

To extract as much entropy as possible out of each event we let *Woodpecker* use the highest available time-measurement fidelity of its host. This allows us to raise the original *LRNG* time stamp fidelity of 1 millisecond,

[1]Note that it is possible to use a single 32-bit word containing the time difference between the current and the previous messages, but using the full 64-bit time stamp can be beneficial.

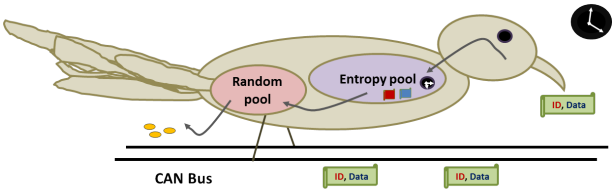

Fig. 2. The *Woodpecker* mechanism. Collecting the messages from the CAN bus and getting the related time stamp, before updating its main entropy pool (with the message ID, Data, and its time of arrival). The bits are further processed before passed into the random pool for external usage.

to higher levels. Some of our test devices allowed a fidelity of 100μsec, where others allowed higher fidelity of 1μsec.

In our experimentation (Section IV) we tracked the results of the *LRNG* entropy estimation, assuming no random bits are extracted from the pool. We further evaluated the entropy in the pool using the classical Shannon entropy definition:

$$Entropy = \sum_{x} -P(x)\log_2(P(x)) \qquad (2)$$

where $P(x)$ represents the probability of a value x (an inter-arrival time) to appear within the given sample.

Section IV-B1 summarizes our results for the gathered entropy under the *LRNG* methodology, and Section IV-B2 summarizes our findings using the Shannon definition.

C. Non determinism checks

Beyond evaluation of the amount of randomness within a single session, we wanted to measure the determinism of CAN sessions: if a CAN trace is highly deterministic (even if it has high entropy) than an attacker that can record one session may be able to predict the contents of the random pool in another session.

To evaluate the non-determinism we used two different tools: the Linux `diff` utility, and entropy measurement on the difference-of-traces.

The evaluation was done between pairs of traces taken under identical conditions, e.g., between two traces of the same length, and the same vehicle, at a similar state.

1) The Linux diff utility: We used the Linux `diff` utility to evaluate the difference between every two analyzed samples (trace files). The analysis was done on text files that include the list of the time differences, one per line, from the given sample (.trc file).

To quantify the difference level we count the number of different lines between the compared files as reported by `diff`, and calculate their relative percentage within the overall number of lines (which represent messages) in the given files. `diff` uses a "longest common subsequence" (LCS) algorithm to find matching blocks of lines. Different blocks of lines are reported with an 'a', 'c', and 'd' tags, for "added", "changed" and "deleted" blocks of lines.

The utility was executed on an Ubuntu 16.04.4 LTS (Release: 16.04), under Windows 10, using *GNU diffutils* 3.3. To count the number of differences we issued:

```
diff -d f1.txt f2.txt | egrep "[acd]" | wc
diff -d f2.txt f1.txt | egrep "[acd]" | wc
```

Note that we used the same number of lines for each comparison, and used the '-d' option of the utility (that try to minimize the differences), to reduce the differences to the minimum. We also compared each pair twice as presented above to remove symmetry related issues.

The results of these measurements are described in the first part of Section IV-C.

2) LRNG entropy measurements of the trace difference: In addition to the `diff` utility, we also calculated the *LRNG* entropy of the difference between the inter-arrival times of every two traces. We did this by subtracting the measured time differences of one sample from the other, obtaining an 'artificial' deltas file for analysis. The analysis was done using the same *LRNG* entropy check as described in Section II-A.

The overall sum of the *LRNG* entropy, per new delta file, can show us how different are the two given samples, where higher entropy represents bigger differences. The results of these checks are described in the second part of Section IV-C.

IV. Testing

This section describes the results from testing the entropy and the non-determinism of multiple samples from different vehicles.

A. Testing environment

The analysis was performed on samples that were taken from the CAN bus (through the OBD-II connector) of three different vehicles: Ford Focus 2012, Jeep 2015, and Subaru B4 2015. The Ford samples were gathered earlier by [20].

We note that unless specified otherwise, we used the same three main series of samples in all of our measurements, where each series contains three samples taken from the same vehicle, under similar conditions. The main series are called: *#CJ1-3* for the *Jeep*, *#FF2-4* for the *Ford Focus*, and *#SB4 81-83* for the Subaru samples.

The CAN bus speed of both the *Jeep* and the *Subaru* vehicles was 500Kbps, giving a bit-time slot of 2μsec.

We believe the same is true also for the Ford vehicle (this value was not specified by [20]). Our measurements were done either with 1μsec (double the bus speed), or with 100μsec (50 times slower than bus speed) fidelity.

The following equipment from *Peak-system* was used to gather the samples:

- PCAN-USB device [44] using the NXP SJA1000 CAN controller [45], [46], with 100μsec fidelity.
- PCAN-USB-FD device [47] using Peak's proprietary FPGA-based CAN controller, with 1μsec fidelity.
- PCAN-Diag-V2 hand tool device (HTD) [48] using the NXP LPC2292 built-in CAN controller, with 1μsec fidelity.

Both PCAN-USB devices were controlled via USB connections by a laptop running Windows, using the PCAN-View control software. The software was used to capture the traffic over the vehicle's CAN bus into *.trc* trace files (of either version 1.1, or 2.0). The trace files include the incoming CAN messages (ID, Data, Type) and their related time-stamp (with 100μsec fidelity for v1.1, and 1μsec fidelity for v2.0).

The hand tool device (HTD) was also used to capture some of the traffic into *.btr* files, that were later converted into *.trc* v2.0 files. When needed, two devices were connected to the same OBD-II port through a *Peak T-connector* (two-to-one D9 connector).

The following subsections summarize our results, first for the entropy measurements, and then for the samples consistency checks. We also add and describe some identified characteristics in relation to the time-stamp's fidelity and the vehicle's condition, when applicable.

At the end of this section (Section IV-C3), we include some preliminary evaluation for the differences between samples of different devices (having the same fidelity) to provide some answer to the internal adversary model (recall Section II-C).

B. Entropy measurements

1) LRNG measurements: Under this mechanism, we estimate the entropy by using a similar procedure as of the *LRNG's* one (recall Section II-A), with one major change—the time differences were calculated on time-stamps with higher fidelity (of either 100μsec or 1μsec) instead of the *LRNG's* 1 millisecond lower fidelity.

Figures 3 shows the *LRNG* calculated entropy-gain of two different samples (having 100μsec fidelity): one from the CAN bus of the Jeep, during the first five seconds from sleep (silent bus) to ignition, and the other from the Ford vehicle during driving. Note that many events have zero entropy, but enough of them carry 1-7 entropy bits. Furthermore, on the Jeep graph the values are higher during the first second, with up to 7 entropy bits on some samples. This is because we observe some very long time differences during the first second (maybe since not all of the ECUs are awake). In the Ford graph, and in seconds 2-5 of the Jeep graph, we see lower entropies, up to 4 bits per sampled event (the inter-arrival time of two consecutive messages).

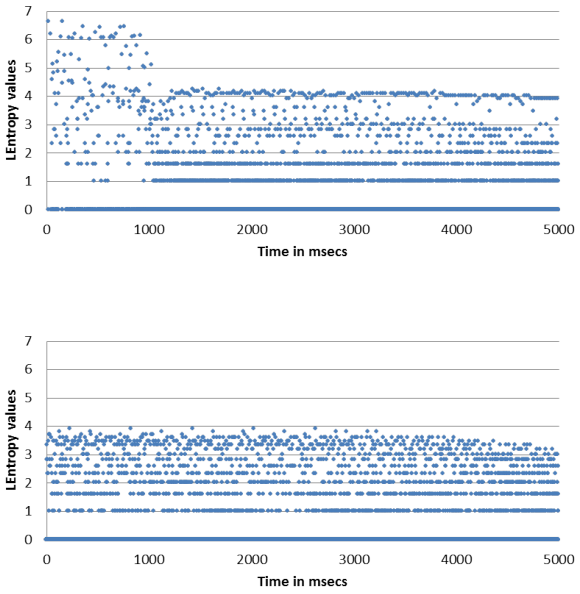

Fig. 3. The *LRNG* entropy per sampled event over time (100μsec fidelity). (Top) Jeep sample *#CJ1*: sleep to ignition. (Bottom) Ford sample *#FF2*: driving. Notice for the differences between the 1st second and the rest in the Jeep sample.

TABLE I
LRNG ENTROPY SUMMARY

	Jeep	**Ford**	**Subaru**
Sample's fidelity in μsecs	100	100	100
Time in milliseconds	4999.96	4999.8	4998.66
Number of Messages	9330	11467	4187
Total *LRNG* entropy	3174.72	3857.35	3491.55
Average entropy per second	634.95	771.50	698.49
Average entropy per message	0.34	0.33	0.83

Table I summarizes the results of the *LRNG* entropy for all three vehicles, each averaged over the three samples of the following series: *#CJ1–3*, *#FF2–4*, *#SB4 N1–3*, having the same fidelity of 100μsec. Note that the Subaru sends roughly half the number of messages per second, yet the entropy per second is quite similar to that measured on the other vehicles.

2) Shannon related measurements: Here we measure the entropy-gain using *Shannon* equation for entropy (recall Equation 2). To do so we first calculate and present the probability of every distinct delta (time gap)

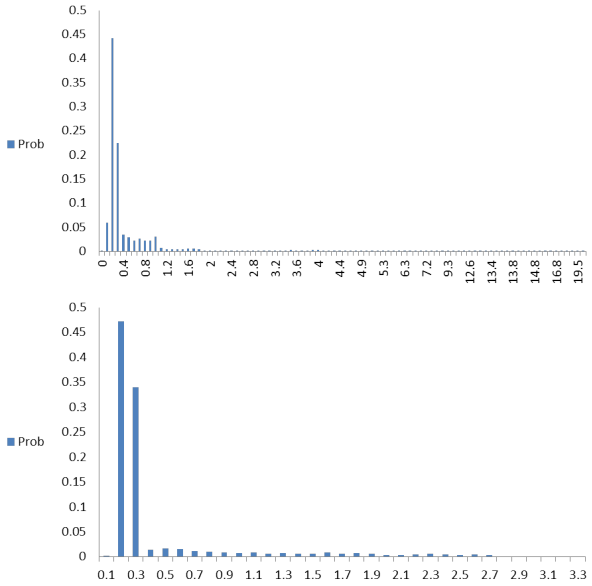

Fig. 4. Probability distribution of inter-arrival times during a 5 second interval (100μsec fidelity). (Top) *Jeep*: sleep to ignition, *#CJ1*. (Bottom) *Ford*: driving, *#FF2*.

TABLE II
DELTA VALUE DISTRIBUTION STATISTICS AND SHANNON ENTROPY

	Jeep	**Ford**	**Subaru**
Sample's fidelity (μsec)	100	100	100
Number of messages	9330	11467	4187
Number of distinct values	87.66	32.33	80.33
***Shannon* entropy**	2.95	2.34	3.62

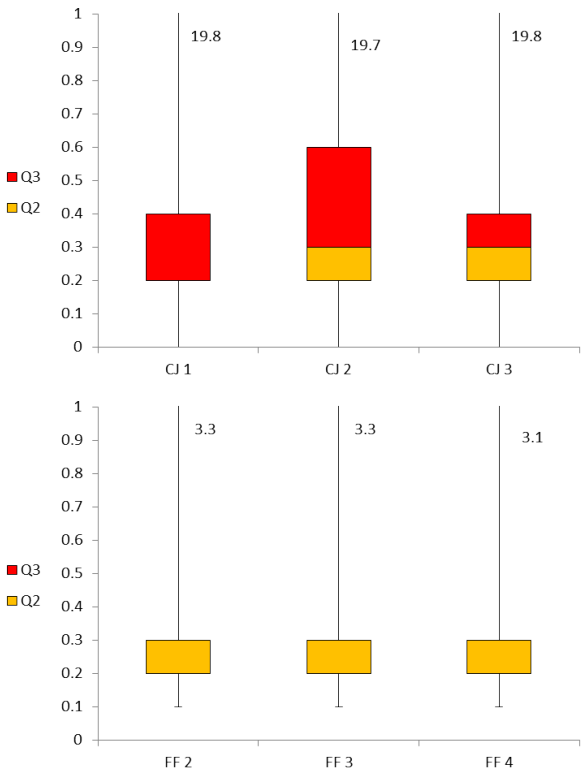

Fig. 5. The delta values distribution of samples over 5 seconds. (Top) *Jeep*: sleep to ignition, *#CJ1–3*. (Bottom) *Ford*: driving, *#FF2–4*. The graph is a box-and-whiskers graph, where the median is indicated by the border between the bottom and the top boxes that indicate the second and third quartiles; The whiskers show the minimum and maximum values (number tag only when exceeding the frame).

value per sample.

Figure 4 displays the distribution of inter-arrival times (delta) of the *Ford* (*#FF2*) and *Jeep* (*#CJ1*) samples. We note that the *Jeep* sample has more distinct delta values than the *Ford* one (90 compared to 33), probably since the *Jeep* sample was gathered during wake-up whereas the *Ford* sample was gathered during driving: recall the higher delta values observed during the 1st second of the *#CJ1* sample seen in Figure 3-top).

Figure 5 shows the delta-value distributions for the *Jeep* and *Ford* distribution, using a box-and-whiskers graph. Note that while the distribution is fairly concentrated around the median value, the distribution does have a long tail.

Table II summarizes the results of the average gathered entropy according to *Shannon* equation for all three vehicles, using the *#CJ1–3*, *#FF2–4*, and *#SB4 N1–3*, 100μsec series. According to this metric a typical delta measurement of the CAN bus of these vehicles yields between 2.34–3.62 bits of uncertainty: much higher than the conservative entropy-per-message estimate of the *LRNG* metric, recall Table I.

3) The Time-stamp's fidelity effect on the entropy: Here we show that the time-stamp's fidelity (e.g, of 1μsec vs. 100μsec) affects the results of the gathered entropy. We expect that when the time is measured at better fidelity, more of the random fluctuations in arrival time will be captured.

We demonstrate this, on both the *LRNG* and the *Shannon* measurements, by using two distinct devices: the regular *CAN-USB* with 100μsec fidelity, and the *CAN-USB-FD* with its 1μsec fidelity, sampling the same CAN traffic concurrently on the same vehicle.

Figure 6 shows the *LRNG* entropy of the same scenario (the first 5 seconds from sleep to switch-on), measured concurrently on the *Jeep* by the two devices.

Note that the total LNRG entropy (over 5 seconds) grew from 4290 to 29522 (from 858 to 5905 per second): a factor of 6.88 growth. This fits our expectation for a x100 improvement in fidelity (since $\log_2(100) = 6.64$).

Figure 7 shows that a higher fidelity also affects the time differences distribution: we observe more distinct delta values (1512 values in comparison to 65 for the 100μsec samples). This also influences the Shannon

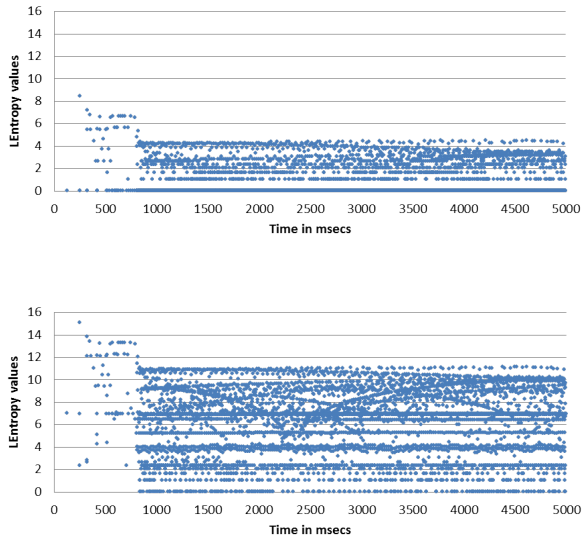

Fig. 6. The *LRNG* entropy per sample over time for the Subaru sleep to switch-on, sampled by two devices in parallel. (Top) Reg sample with 100μsec fidelity. (Bottom) FD sample - 1μsec. Notice the higher values in the lower graph.

TABLE III
THE TIME-STAMP'S FIDELITY EFFECT

Sample's fidelity in μsec	1	100
Time in milliseconds	4998.8	4998.3
Number of messages	4804	4804
Total *LRNG* entropy	29522.08	4290.62
Average *LRNG* entropy per second	5905	858
Average *LRNG* entropy per message	6.14	0.89
Number of distinct values	1512	65
***Shannon* entropy**	7.82	3.81

entropy which grew from 3.81 to 7.83.

Table III summarizes the results of this effect on both the *LRNG* and Shannon's measurements.

C. Non determinism checks

In this section we evaluate the level of determinism of related samples, using two different tools: the Linux `diff` utility, and entropy measurement on the difference-of-traces.

The evaluation was done between pairs of traces taken under identical conditions, e.g., between two traces of the same length, and the same vehicle, at a similar state, under the sleep-to-ignition scenario, which has minimal environmental variability and involves no user actions. We show that even in this extremely constrained scenario there are significant differences between traces.

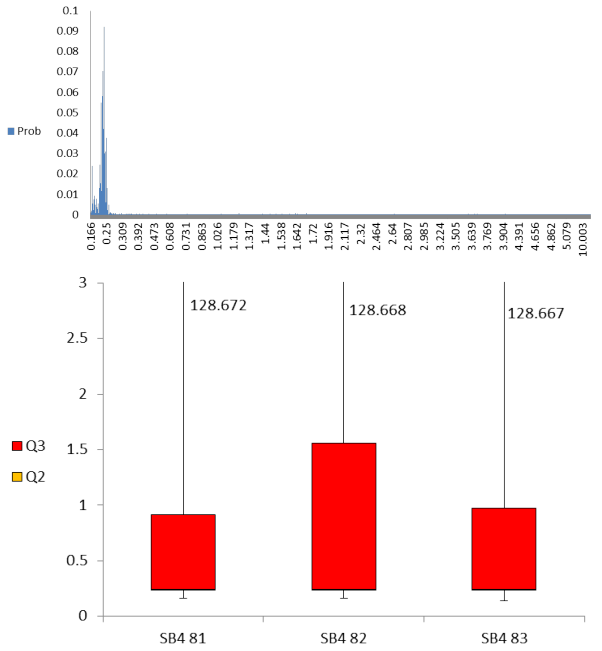

Fig. 7. (Top) *Subaru - #SB4 81* deltas distribution. (Bottom) *Subaru - #SB4 81–83*, box-and-whiskers graph. Notice the higher resolution of the horizontal axis on the distribution graph, in comparison to Figure 4.

TABLE IV
THE *Linux Diff* UTILITY RESULTS SUMMARY

	Jeep	**Ford**	**Subaru**
Sample's fidelity in μsec	100	100	1
Average Number of messages	9248	11464	4137
Average number of differences	2793.33	3267.5	870
Differences per message	0.30	0.28	0.21

1) The Linux Diff utility: Table IV shows the summary of the *Linux Diff* utility (recall Section III-C1) as calculated on the main sample series of each vehicle (*#CJ1–3, #FF2–4, #SB4 81–83*). Our results show an average difference of between 20 to 30 percent, between every two related samples (of the same vehicle, under similar conditions). This implies that the samples are indeed different enough, i.e., there is enough variability in the vehicle CAN bus traffic to make it a valid source of randomness input for the *Woodpecker*.

We note that here the higher fidelity of the *Subaru* samples did not seem to contribute to the overall number of differences—on the Subaru, with 1μsec fidelity we observed a lower number of trace-to-trace differences than the results for the other vehicles at 100μsec fidelity.

One can hypothesize that the beginning of a sleep-to-ignition scenario is more deterministic than later points in time. To test this hypothesis, Table V compares the *Diff* results between the 1st second and the 5th second (using the same number of messages for both

TABLE V
1ST VS 5TH SECOND, *Linux Diff* RESULTS (*#CJ1–3*)

	1st 700	last 700	Full sample
Number of messages	700	700	9248
Average number of diffs	175.33	214.5	2793.33
Differences per message	0.250	0.306	0.302

TABLE VI
LRNG ENTROPY AVERAGES ON THE 'SUBTRACTED' TRACES

	Jeep	Ford	Subaru
Sample's fidelity in μsec	100	100	1
Number of messages	9248	11464	4137
Total *LRNG* entropy	4816.29	6127.76	23843.41
Average per message	0.52	0.53	5.76

sets) using *Jeep's #CJ1–3* series. The table shows that while the hypothesis is generally true, the effect is not very strong: during the first second we measured about 25% differences whereas during the fifth second we measured about 30% differences.

2) LRNG entropy measurements: An alternative method of estimating the determinism (or lack thereof) across different traces of the same scenario relies on the *LRNG* entropy. However, now we measure the *LRNG* entropy in an artificial "trace" that is the subtraction of two related traces. We did this for every two related samples, recall Section III-C2. The assumption is that higher entropy in the subtracted trace indicates more non-determinism: in the extreme, two identical traces would yield an all-zero subtracted trace with an *LRNG*-entropy of 0.

Table VI summarizes the results of this check (on the main three sample series), showing relatively high entropy averages for all three vehicles, strengthening our results from the previous subsection, raising the chances that the CAN traffic can truly serve as a good random seed input for the *Woodpecker*. It is interesting to note that these results are even higher than those of the original trace files as presented in Table I

3) Preliminary evaluation of the difference between samples of different devices: Here we try to answer whether different devices can obtain different results even when having the same fidelity. This can be important if operating under the internal-adversary model - where the attacker has access to the same traffic, and therefore may possibly be able to calculate the same random bits as of the target ECUs.

For this purpose we sampled the Subaru CAN bus, at the same time, by two separate devices - the *PCAN USB FD* and the *HTD*. Both devices provide the same fidelity of 1μsec, which is twice as fast as the CAN bus speed (of 500Kbps).

TABLE VII
NON-DETERMINISM RESULTS OF PARALLEL SAMPLES USING DISTINCT DEVICES, IN RELATION TO THE *#SB4 81–3* RESULTS

	HTD vs FD	FD *#SB4 81–3* results
Number of msgs	4137	4137
Diff: Avg num of diffs	845.5	870
Diff: Avg diffs per msg	0.20	0.21
***LRNG* entropy sum avg**	23.18	23843.41
***LRNG* avg per message**	0.005	5.763

The left column of Table VII summarizes the results of this check (using the Subaru main sample series of both the *FD* and the *HTD* devices), where the right column is used as a reference to include the previously displayed results of the consistency level as calculated between the three related samples of Subaru (The Subaru's columns in Tables IV and VI).

The table shows that the *Diff* utility values of the *'FD vs HTD'* samples are quite high: similar to the values that were calculated between the different samples (recall Table IV). On the other hand the *LRNG* entropy values of the subtracted traces are quite low.

After inspecting the data more closely we saw that there were indeed many differences between the samples captured by the *FD* and *HTD* devices, but these differences were usually of 1μsec. A difference of 1 disappears under the *LRNG* calculation since the least-significant bit, bit 31, is discarded (recall Equation 1). However a +/-1 difference *is* counted by the *Diff* utility, hence the qualitative difference between the results of the two measures.

It seems that much of the variability in the message arrival times resides in the least-significant bits. Thus it may be better to include *all* the bits of the delta in case of using the *LRNG* equation for the *Woodpecker's* entropy evaluation. Using devices with higher fidelity (e.g., of more than twice as the CAN bus speed) could provide even better results—to improve *Woodpecker*'s resilience to an internal-attacker adversary. We leave this aspect for future work.

V. LIMITATIONS AND MITIGATIONS

The *Woodpecker* mechanism has some limitations we present below, together with possible mitigations. We recommend to take them into account when considering this solution.

A. Traffic filtering

ECUs may be configured to gather only some of the CAN messages (filtered by message ID), which restrict *Woodpecker*'s timing measurement; furthermore, an ECU may not support the transfer of the full message payload (message ID and Data) to *Woodpecker*,

for technical, or cost related issues. If *Woodpecker* is implemented inside the CAN Transceiver or Controller then traffic filtering may not affect it, however if it is implemented in the ECU's hosting system—traffic filtering could degrade the quality and rate of random bit output. Further investigation is needed to better understand this potential limitation.

B. Inside attackers

An adversary model in which the attacker has access to the same traffic seen by *Woodpecker*, e.g., when the attacker resides in a neighboring ECU, is more challenging; Such an attacker has the theoretical possibility of calculating the same random bits as the *Woodpecker* ECU.

Our preliminary results (Section IV-C3) show that different ECUs, observing the same CAN traffic, measure different inter-arrival times, when using the same measurement fidelity. These positive results lead us to believe that *Woodpecker* can operate effectively under this adversary model, but further investigation is required to better evaluate this. We further suggest several options to deal with inside attackers, that can strengthen the system's security:

- Maintain a random state to serve as the initial seed for *Woodpecker* on every boot—e.g., by periodically storing the current random pool in non-volatile memory. This mimics what the Linux RNG does. Note that this solution is still vulnerable if the attacker code is running inside the same ECU as *Woodpecker* and is able to access the current state or pool.
- Let each *Woodpecker* choose which message IDs to gather, serving as a potentially secret key. A CAN bus in a typical vehicle carries a few tens of different message IDs, so there is a large selection. We note that changing this selection on every boot (e.g., based on the saved state of *Woodpecker*) can probably make this solution even better.
- Maintain two separate random pools. The first pool is for external usage (e.g., for the VTX protocol) and uses the regular *Woodpecker* mechanism described in Section III-A), to answer the external adversary model only. The second pool is for internal usage (e.g., key generation, internal message authentication, etc.). The internal pool can be governed by a stricter minimum-entropy counter, much like the Linux `/dev/random` device, which blocks read attempts if the entropy counter is too low. Further investigation is required to evaluate whether this solution can indeed be effective.

C. Malicious traffic manipulation

An attacker that is able to transmit non legitimate messages over the bus can add new messages, or delay others, which may manipulate *Woodpecker's* timestamps measurements. We are unsure how much control the attacker may gain over the random pool's state by this method, but it may be a concern. Some scenarios may be especially challenging: E.g., if the attacker is able to raise the traffic density to 100%, and the system starts from a fixed state, *Woodpecker* may measure many small and predictable inter-arrival times—namely the CAN *intermission* gap—which may drive the pool to an attacker-chosen state. We note that this scenario may be problematic to the entire vehicle (it is a type of a Denial of Service attack which may shut down the CAN bus).

VI. Conclusion and Future Work

This paper described *Woodpecker*, a software-only True-Random-Number-Generator (TRNG) for electronic control units (ECUs) connected to a vehicular CAN bus. *Woodpecker* follows the design of the Linux RNG mechanism (*LRNG*), and relies on the unpredictability of inter-arrival times of CAN messages as its primary source of input, mixed with the message IDs and message payloads. Our main contribution is demonstrating that despite the strong periodicity exhibited in CAN bus traffic, if time events are measured at a reasonably high fidelity, the inter-arrival times embed enough entropy to extract true random bits. We evaluated our method on vehicles from Jeep, Ford, and Subaru. Using a time measurement fidelity of 100μsec we extracted 634–770 entropy bits per second according to *LRNG* estimation. Using 1μsec fidelity we extracted 4944 entropy bits per second.

Beyond randomness within a single driving session, we evaluated the variability between different sessions. We compared the inter-arrival times captured in multiple sleep-to-ignition traces, taken in close succession in the same car, without any driver actions beyond pressing the ignition button. Even under such similar conditions we found that about 25–30% of the inter-arrival times varied between traces.

We demonstrated that even if the attacker is connected to the same CAN bus and is measuring the inter-arrival times simultaneously with *Woodpecker* at the same 1μsec fidelity (twice as fast as the CAN bus speed of 500 Kbps), there is sufficient variation introduced by the measurement equipment to produce about 20% of differences between the *Woodpecker* and attacker traces.

We believe that using devices with higher time fidelity (e.g., an ECU with an 8MHz clock) could provide even better results, allowing *Woodpecker* to deal also with the more challenging internal adversary model. We note that

further investigation is needed (e.g., experiments with real ECUs) in order to check this.

ACKNOWLEDGMENTS

We would like to thank Yuval Montvelisky for helping with gathering some of the samples that were used for this work.

REFERENCES

[1] A. Greenberg, "Hackers remotely kill a Jeep on the highwaywith me in it," http://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/, 2015.

[2] D. Pauli, "Hackers hijack Tesla Model S from afar, while the cars are moving," http://theregister.co.uk/2016/09/20/tesla_model_s_hijacked_remotely, 2016.

[3] K.-T. Cho, Y. Kim, and K. G. Shin, "Who killed my parked car?" *arXiv preprint arXiv:1801.07741*, 2018.

[4] K. J. Henry., P. Bottinelli., and N. Ebeid, "Entropy and randomness in vehicular environments and v2x applications," in *5th Embedded Security in Cars (ESCAR USA)*, Ypsilanti, MI, USA, Jun. 2017.

[5] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage, "Experimental security analysis of a modern automobile," in *IEEE Symposium on Security and Privacy (SP)*, May 2010, pp. 447–462.

[6] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno, "Comprehensive experimental analyses of automotive attack surfaces," in *Proceedings of the 20th USENIX Conference on Security*, ser. SEC'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 6–6. [Online]. Available: http://dl.acm.org/citation.cfm?id=2028067.2028073

[7] D. C. Miller and C. Valasek, "Adventures in automotive networks and control units," http://www.ioactive.com/pdfs/IOActive_Adventures_in_Automotive_Networks_and_Control_Units.pdf, 2014, [Online; accessed 22-July-2015].

[8] A. Greenberg, "After Jeep hack, Chrysler recalls 1.4m vehicles for bug fix," http://www.wired.com/2015/07/jeep-hack-chrysler-recalls-1-4m-vehicles-bug-fix/, 2015.

[9] I. Foster and K. Koscher, "Exploring controller area networks," *USENIX ;Login: magazine*, vol. 40, no. 6, 2015.

[10] B. Glas and M. Lewis, "Approaches to economic secure automotive sensor communication in constrained environments," in *11th Int. Conf. on Embedded Security in Cars (ESCAR 2013)*, 2013.

[11] A. Van Herrewege, D. Singelee, and I. Verbauwhede, "CANAuth-a simple, backward compatible broadcast authentication protocol for CAN bus," in *ECRYPT Workshop on Lightweight Cryptography*, 2011.

[12] T. Ziermann, S. Wildermann, and J. Teich, "CAN+: A new backward-compatible controller area network (CAN) protocol with up to 16× higher data rates." in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2009, pp. 1088–1093.

[13] AUTOSAR, "AUTOSAR secure onboard communication (SecOC), version 4.3," https://www.autosar.org/standards/classic-platform, 2016.

[14] T. Matsumoto, M. Hata, M. Tanabe, K. Yoshioka, and K. Oishi, "A method of preventing unauthorized data transmission in controller area network," in *IEEE Vehicular Technology Conference (VTC Spring)*. IEEE, 2012, pp. 1–5.

[15] R. Kurachi, Y. Matsubara, H. Takada, N. Adachi, Y. Miyashita, and S. Horihata, "CaCAN—centralized authentication system in CAN (controller area network)," in *12th Int. Conf. on Embedded Security in Cars (ESCAR)*, 2014.

[16] R. Kurachi, H. Takada, T. Mizutani, H. Ueda, and S. Horihata, "SecGW secure gateway for in-vehicle networks," in *13th Int. Conf. on Embedded Security in Cars (ESCAR)*, 2015.

[17] Y. Ujiie, T. Kishikawa, T. Haga, H. Matsushima, T. Wakabayashi, M. Tanabe, Y. Kitamura, and J. Anzai, "A method for disabling malicious CAN messages by using a centralized monitoring and interceptor ECU," in *13th Int. Conf. on Embedded Security in Cars (ESCAR)*, 2015.

[18] T. Dagan and A. Wool, "Parrot, a software-only anti-spoofing defense system for the CAN bus," in *14th Int. Conf. on Embedded Security in Cars (ESCAR)*, Munich, Germany, Nov. 2016.

[19] ——, "Testing the boundaries of the Parrot anti-spoofing defense system," in *5th Embedded Security in Cars (ESCAR USA)*, Ypsilanti, MI, USA, Jun. 2017.

[20] M. Markovitz and A. Wool, "Field classification, modeling and anomaly detection in unknown CAN bus networks," in *13th Embedded Security in Cars (ESCAR'15)*, Cologne, Germany, Nov. 2015.

[21] ——, "Field classification, modeling and anomaly detection in unknown CAN bus networks," *Vehicular Communications*, vol. 9, pp. 43–52, 2017.

[22] M. Marchetti, D. Stabili, A. Guido, and M. Colajanni, "Evaluation of anomaly detection for in-vehicle networks through information-theoretic algorithms," in *2nd IEEE International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI)*, Sept 2016, pp. 1–6.

[23] M. Marchetti and D. Stabili, "Anomaly detection of can bus messages through analysis of id sequences," in *28th IEEE Intelligent Vehicle Symposium (IV)*, June 2017, pp. 1–6.

[24] Y. Hamada, M. Inoue, S. Horihata, and A. Kamemura, "Intrusion detection by density estimation of reception cycle periods for in-vehicle networks: A proposal," in *14th Int. Conf. on Embedded Security in Cars (ESCAR 2016)*, Munich, Germany, Nov. 2016.

[25] H. Lee, S. H. Jeong, and H. K. Kim, "Otids: A novel intrusion detection system for in-vehicle network by using remote frame," in *PST*, 2017.

[26] K.-T. Cho and K. G. Shin, "Viden: Attacker identification on in-vehicle networks," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 1109–1123.

[27] W. Choi, K. Joo, H. J. Jo, M. C. Park, and D. H. Lee, "Voltageids: Low-level communication characteristics for automotive intrusion detection system," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 8, pp. 2114–2129, 2018.

[28] NXP automotive, "http://www.nxp.com/applications/automotive," 2017, [Online; accessed July-2017].

[29] Bosch mobility solutions, "http://www.bosch-mobility-solutions.com/en," 2017, [Online; accessed July-2017].

[30] T. van Roermund, A. Birnie, R. Moran, and J. Frank, "Securing the in-vehicle network of the connected car," in *14th Int. Conf. on Embedded Security in Cars (ESCAR)*, Munich, Germany, Nov. 2016.

[31] L. Dorrendorf, Z. Gutterman, and B. Pinkas, "Cryptanalysis of the Windows random number generator," in *Proceedings of the 14th ACM conference on Computer and Communications Security*. ACM, 2007, p. 485.

[32] Z. Gutterman, B. Pinkas, and T. Reinman, "Analysis of the Linux random number generator," pp. 371–385, 2006. [Online]. Available: http://dl.acm.org/citation.cfm?id=1130235.1130388

[33] "Ubuntu security notice usn-612-1 may 13, 2008 openssl vulnerability cve-2008-0166."

[34] E. Schreck and W. Ertel, "Disk drive generates high speed real random numbers," *Microsystem Technologies*, vol. 11, no. 8, pp. 616–622, 2005.

[35] Q. Zhou, X. Liao, K. Wong, Y. Hu, and D. Xiao, "True random number generator based on mouse movement and chaotic hash function," *Information Sciences*, vol. 179, no. 19, pp. 3442–3450, 2009.

[36] J. Voris, N. Saxena, and T. Halevi, "Accelerometers and randomness: Perfect together," in *Proceedings of the fourth ACM conference on Wireless Network Security*, ser. WiSec '11. New York, NY, USA: ACM, 2011, pp. 115–126. [Online]. Available: http://doi.acm.org/10.1145/1998412.1998433

[37] D. Holcomb, W. Burleson, and K. Fu, "Power-up SRAM state as an identifying fingerprint and source of true random numbers," *IEEE Transactions on Computers*, vol. 58, no. 9, pp. 1198–1210, Sep. 2009.

[38] E. Ronen, "Security applications for hardware performance counters: Software attestation and random generation," Master's thesis, School of Electrical Engineering, Tel Aviv University, 2012. [Online]. Available: https://www.eng.tau.ac.il/~yash/ronen-thesis-2012.pdf

[39] M. Howard and D. Leblanc, *Writing Secure Code*, 2nd ed. Redmond, WA, USA: Microsoft Press, 2002.

[40] A. Seznec and N. Sendrier, "HAVEGE: A user-level software heuristic for generating empirically strong random numbers," *ACM Transactions on Modeling and Computer Simulation*, vol. 13, no. 4, pp. 334–346, 2003.

[41] S. Srinivasan, S. Mathew, R. Ramanarayanan, F. Sheikh, M. Anders, H. Kaul, V. Erraguntla, R. Krishnamurthy, and G. Taylor, "2.4GHz 7mW all-digital PVT-variation tolerant true random number generator in 45nm CMOS," in *IEEE Symposium on VLSI Circuits (VLSIC)*, Jun. 2010, pp. 203–204.

[42] J. Wan, A. B. Lopez, and M. A. Al Faruque, "Exploiting wireless channel randomness to generate keys for automotive cyber-physical system security," in *Cyber-Physical Systems (ICCPS), 2016 ACM/IEEE 7th International Conference on*. IEEE, 2016, pp. 1–10.

[43] Robert Bosch GmbH, "CAN specification, version 2.0," 1991. [Online]. Available: http://www.bosch-semiconductors.de/media/ubk_semiconductors/pdf_1/canliteratur/can2spec.pdf

[44] PEAK-System, "PCAN-USB: CAN interface for USB," 2015. [Online]. Available: http://www.peak-system.com/produktcd/Pdf/English/PCAN-USB_UserMan_eng.pdf

[45] Philips Semiconductors, "SJA1000 stand-alone CAN controller," Application Note AN97076, 1997. [Online]. Available: http://www.nxp.com/documents/application_note/AN97076.pdf

[46] ——, "SJA1000, stand-alone CAN controller," Data Sheet, 2000. [Online]. Available: http://www.nxp.com/documents/data_sheet/SJA1000.pdf

[47] PEAK-System, "PCAN-USB FD: CAN FD interface for high-speed USB 2.0," 2015. [Online]. Available: http://www.peak-system.com/produktcd/Pdf/English/PCAN-USB-FD_UserMan_eng.pdf

[48] ——, "PCAN-Diag 2: Handheld device for CAN bus diagnostics," 2015. [Online]. Available: http://www.peak-system.com/produktcd/Pdf/English/PCAN-Diag2_UserMan_eng.pdf