Developing a QRNG ECU for automotive security: Experience of testing in the real-world

Nguyen, H. N., Tavakoli, S., Shaikh, S. & Maynard, O.

Original citation & hyperlink:
Nguyen, HN, Tavakoli, S, Shaikh, S & Maynard, O 2019, Developing a QRNG ECU for automotive security: Experience of testing in the real-world. in 2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), Xi'an, China, 2019. IEEE, pp. 61-68, 14th Workshop on Testing, Xi'an, China, 22/04/19.
https://dx.doi.org/10.1109/ICSTW.2019.00033

DOI 10.1109/ICSTW.2019.00033
ISBN 978-1-7281-0889-6
ISBN 978-1-7281-0888-9

Publisher: IEEE

Developing a QRNG ECU for automotive security: Experience of testing in the real-world

Hoang Nga Nguyen
Coventry University
Coventry, UK
Hoang.Nguyen@coventry.ac.uk

Siamak Tavakoli
Crypta Labs Limited
London, UK
Siamak@cryptalabs.com

Siraj Ahmed Shaikh
Coventry University
Coventry, UK
Siraj.Shaikh@coventry.ac.uk

Oliver Maynard
Crypta Labs Limited
London, UK
Oliver@cryptalabs.com

***Abstract*—Over the last decade, automotive components and systems have become increasingly connected and digital in nature. This trend has significantly increased the risk of malicious interference with car components, vehicles and infrastructure, and cybersecurity defences have generally proven to be lacking. The success of, and trust in, connected and autonomous vehicles (CAVs) relies upon these security gaps being closed as soon as possible. To this end, Crypta Labs introduced a novel electronic control unit (ECU) prototype for enabling secure digital communication in the transport domain. Due to its novelty, it is a challenge to evaluate its functionality, robust and reliable for automotive platforms. In this paper, we introduce the novel ECU, apply a testing methodology specially adapted to this product to achieve the evaluation goal, and conclude with a discussion.**

***Index Terms*—Automotive, Security, Testing, ECU, Quantum Random Number Generator, Testbeds, Denial of Service (DoS)**

I. Introduction

Automotive components and systems have become increasingly connected and digital. This trend has significantly increased the risk of malicious interference with car components, vehicles and infrastructure, and cybersecurity defences have generally proven to be lacking. For example, traditional telematics capability for vehicles has evolved to support critical functionality including firmware updates for onboard Electronic Control Units (ECUs). Known as Over-The-Air (OTA) updates, this is significant to remotely address feature updates and performance flaws; indeed Tesla demonstrated this to address problems with braking systems on their recent model [11]. Supporting safety-critical functions through such connectivity, however, brings to fore the concerns around security of such components [13]. However, a key concern here is remote exploitation of vulnerabilities on the communication units onboard vehicles. Miller and Valasek brought attention to this [18] when they managed to perform remote code execution due to authentication flaws in the Uconnect System [9]. A further concern is attacks due to infiltration of software supply chains [12]. Essentially, third-party software could carry a 'backdoor' to bypass authentication measures. Therefore, the success of, and trust in, CAVs relies upon these security gaps being closed as soon as possible.

To this end, a key and traditional solution is encryption which is vital to protect confidential data by converting it so that cannot be read by humans or computers without authorised access. Kerckhoffs's principle states that the security of a cipher must reside entirely in the key, that is it must be entirely random. High-quality random numbers are therefore the essential foundation and enabler of encrypted-communication systems and products. Currently, the vast majority of cryptographic systems use algorithmically based random number generators (RNG), so-called Pseudo-RNG (PRNG). The weakness of this approach is that PRNGs are entirely deterministic: an eavesdropper who obtains the initialisation settings of the algorithm will be able to predict all possible outcomes, thus compromising the entire protocol.

In a true random number generator (TRNG) randomness comes from a naturally occurring source of randomness, i.e. entropy. Because the outcome of quantum-mechanical events cannot be predicted, they are considered the gold standard for random number generation (QRNG). QRNGs have typically been based on specialised hardware, such as single-photon sources and detectors resulting in cumbersome, large and often high-end/expensive systems [8]. Using the quantum properties of light, Crypta Labs have developed QRNG technology for deployment in small, remote and/or portable devices in a dynamic environment. They have implemented this foundation of secure digital communication in an electronic control unit (ECU) prototype for the automotive industry. The prototype requires evaluation in terms of technical and commercial suitability of Crypta Labs' QRNG technology to CAVs.

The rest of this paper is structured as follows. Section II introduces the design and implementation of the QRNG prototype. Section III presents our methodology for evaluating the prototype. Section IV analyses the test results, and Section V concludes the paper with a discussion.

II. Prototyping QRNG ECU

This section introduces the ECU that generates Quantum Random Numbers (QRNs). We describe the QRNG module responsible for generating random numbers followed by implementation of the QRNG ECU prototype.

A. QRNG module

The basis for our QRNG module is a controlled light source, such as an LED, shining on a sensor of a CMOS digital camera. The camera takes a picture and stores a frame in an uncompressed file format. This matrix of raw numerical values corresponds to the number of photons which were detected

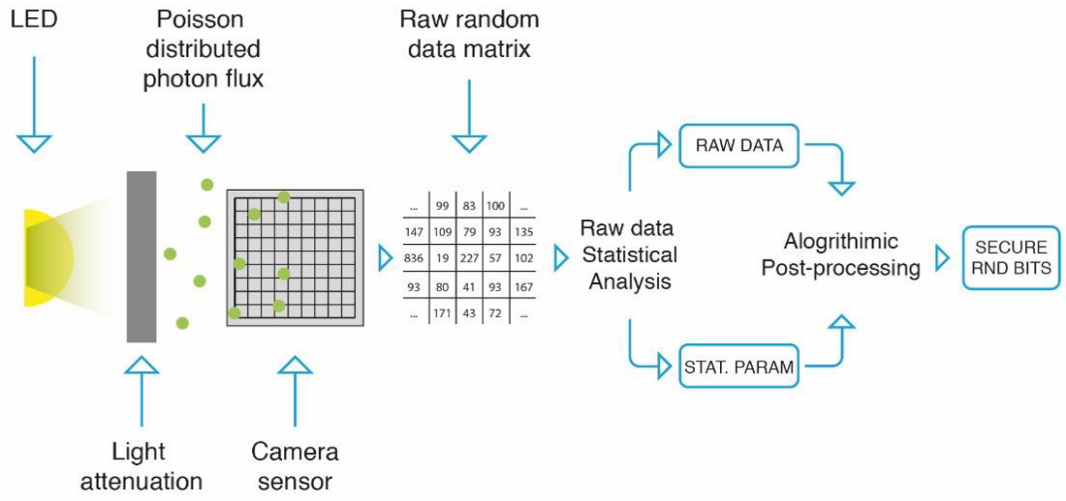

Fig. 1. A high-level overview of the QRNG mechanism proposed by Crypta Labs: The LED provides a source of photons which are captured by the Camera sensor; the number of detected photons during exposure time is stored in the raw random data matrix; This matrix is, later, analysed to generate the final random numbers.

during the exposure time of the picture. The raw data is then processed by an algorithm to enhance the entropy per bit of the final random numbers. The overall process is depicted in Fig. 1. The QRNG firmware provides three main activities. Firstly, it controls the camera, allowing the user to set the desired capture parameters. The software also allows the user to set the luminosity of the light source.

Secondly, it reads the value of the pixels from the image and converts this into a digital value. This is our raw random number. We developed a post-processing library in ANSI C that can take the input from the raw images and generate the random data. This library should be portable to most operating systems and platforms.

Finally, the software applies a post-processing algorithm which performs several functions on the input to produce the usable random number.

B. Implementing the QRNG ECU prototype

Within the controller area network (CAN) of nodes that would at some point in the network communicate with the entities outside the network, implementation of secure communication seems to be of the essence. It is important to establish that not every single node inside CAN would require to facilitate secure data to exchange with the other nodes inside the same CAN. It is equally important that those nodes that establish connections with outside CAN and exchange data do so in its most secure way in order to protect the privacy of data, communication protocol, as well as the security of the rest of the CAN nodes in the same network.

Prototype Architecture

The prototype system is split into two sub-systems each of which is realised by an ECU. The first ECU, containing the QRNG module, is in charge of receiving QRN requests, generating and providing QRNs for the second. In turn, the second ECU is responsible for requesting and using the provided QRNs such as in a hardware security module (HSM) for encrypting or authenticating a data. However, we shall not discuss the HSM in details and its implementation is left for future work.

Fig. 2 shows the building blocks of both sub-systems of the prototyping system. In the diagram, each block has operation access on its lower block. The thin lines between some of the blocks show the data communication links between them.

Furthermore, the interaction between two sub-systems is illustrated by the sequence diagram in Fig. 3. It shows the order of the interaction between the main building blocks of the software in the two ECUs. From top to bottom, the Custom code on a secondary ECU would decide to start one round of such process by issuing a request for a QRN on the CAN bus. The request would be understood and processed by the QRNG ECU by chaining the requests towards the QRN Engine and from there the camera device. Custom code, then upon the receipt of the QRN decides to use it by invoking an encryption function from the HSM block. Furthermore, the Fixed Environment part of the software in QRNG ECU would be wrapped in the application code, i.e. wrapper part of the Custom Environment code. Fixed firmware would also reside on the second ECU where it would need to access the encryption functionality, for example from the HSM block.

System Implementation

Each ECU in the prototype is implemented by a Raspberry Pi[1] extended with a PiCAN2 board[2].

This PiCAN2 board extends the Raspberry Pi with CAN-Bus communication ability. Connections to physical CAN bus from the PiCAN2 are enabled via a DB9 socket or a screw terminal. The two ends of the CAN bus use termination

[1]https://www.raspberrypi.org/

[2]http://skpang.co.uk/catalog/pican2-canbus-board-for-raspberry-pi-23-p-1475.html

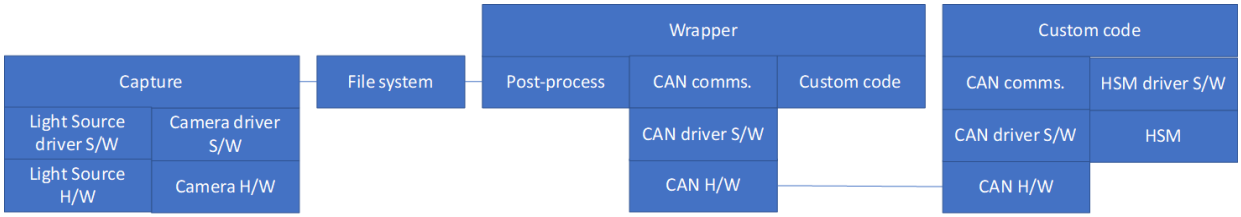

Fig. 2. The building blocks of the prototype QRNG system. Each block has operation access on its lower block. The thin lines between some of the blocks show the data communication links between them.

resistors in order to avoid attenuation of the signal due to mismatch of line impedance.

In Fig. 5, the Raspberry Pi board is coloured in green; the Pi-CAN2 board, coloured blue, is mounted on the Raspberry Pi board as a shield. CryptaLabs QRNG firmware operates on the Raspberry Pi device.

Fig. 4 shows the schematic block diagrams of the QRNG ECU and the secondary ECU, i.e., other ECU in Fig. 3. In Fig. 4, the main computing platform, Raspberry Pi is connected to the PiCAN2 device through its main on-board Input/Output pins that also provide power to the PiCAN2 board. The camera is connected to the Raspberry Pi board through USB port for both powering the camera and the data exchange. The light source that is coupled to the camera is powered separately but is connected via USB to the Raspberry Pi board to allow commands to set up the colour strength of the light source.

Fig. 6 shows provides a physical view of the components of the two ECUs. QRNG ECU is connected on its USB (Universal Serial Bus) port to the camera device which in turn is coupled to the LED (Light Emitting Diode) light source. The characteristics of the camera and its light source are set to the settings that guaranty the quality of the randomness of the generated random number. Obviously, this is also supported by the post-processing software that QRN Engine runs.

Fig. 7 shows a view of the interactions of the ECU at execution. The exchange of the CAN messages between the two ECUs is also monitored by means of standard CAN software.

Both operating systems of the two Raspberry Pi ECUs are Linux based. They support the software drivers that the camera and the PiCAN2 require.

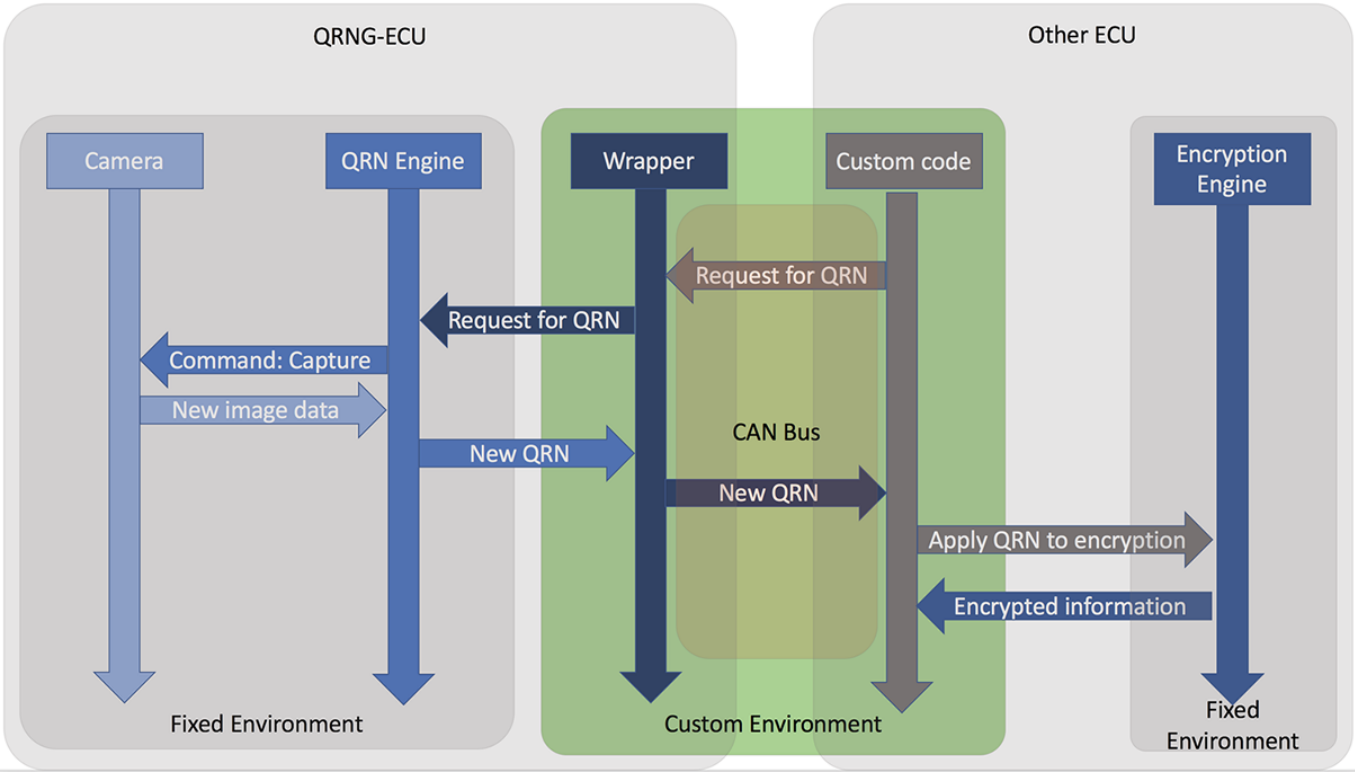

Fig. 3. Functional architecture of QRNG ECU which will served as a means for transmitting of confidential data between other ECUs. For example, upon a over the air update request, a telematics ECU on a Tesla [11] wants to send to a brake ECU an encrypted firmware for update. The QRNG ECU can provide secure encryption key to both ECUs to enable a secure transmittion of the firmware.

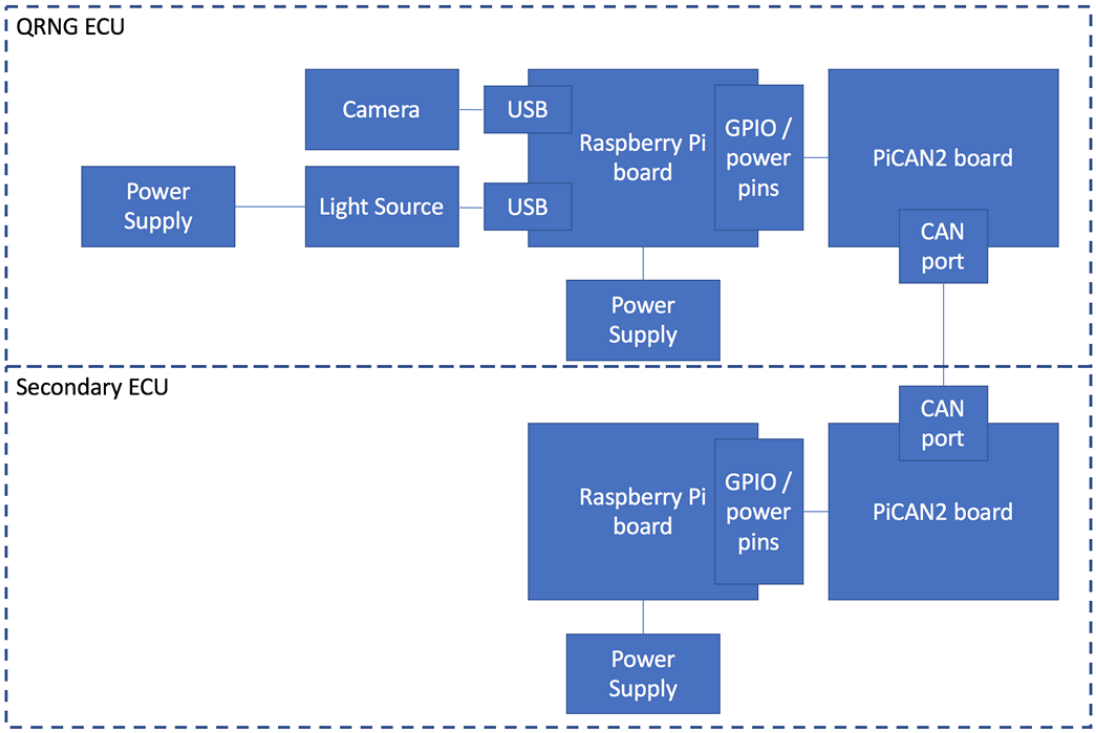

Fig. 4. Schematic diagrams of the QRNG ECU prototype and the secondary ECU. The QRNG ECU is responsible for receiving QRN requests, generating and providing QRNs. The secondary EC is responsible for requesting QRNs.

Fig. 5. PiCAN2 board[2] (blue) connected to a Raspberry-Pi (green). PiCAN2 board extends the Raspberry Pi with CAN-Bus communication ability. Connections to physical CAN bus from the PiCAN2 are enabled via a DB9 socket or a screw terminal.

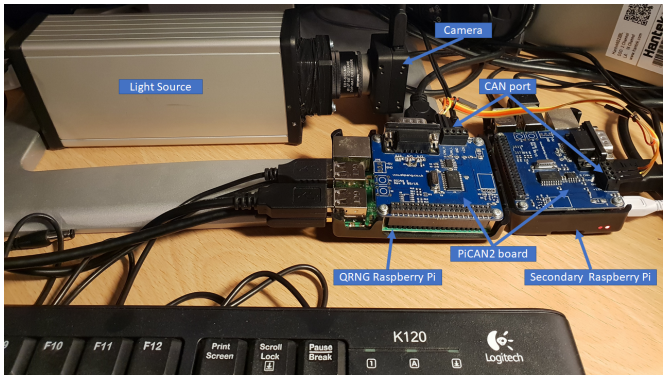

Fig. 6. A physical view of the prototype. The QRNG ECU is on the left. The camera is connected to it via USB port. The secondary ECU is on the right. It is connected with the QRNG ECU via an orange-yellow CAN bus cable.

III. Testing Methodology

Given a prototype QRNG-ECU, our testing objectives are to validate its functionality and evaluate its performance as well as its fault tolerance. Towards functionality, we check if the QRNG-ECU serves requests for random numbers as expected. Towards performance, our aim is to determine the maximal capability of serving requests. Finally, towards fault tolerance, we evaluate the fault tolerance of the QRNG ECU. We are most concerned with the influence that an attacker can have on disrupting the functioning of QRNG ECU via the CAN bus network and on reducing the quality of random numbers generated by the ECU. To this end, we employ a threat model where an attacker has access to the CAN bus to carry out a Denial of Service (DoS) attack or ability to guess numbers to be generated in advance, e.g., by the technique described in [17]. Therefore, we carry out two security tests: an availability test where DoS attacks are used and a quality test for random numbers generated by the QRNG ECU.

Our testing methodology is based on a testbed for automotive security testing introduced in [7]. This testbed uses a commercial Hardware In the Loop (HIL) tool which is capable of faithfully reproducing a CAN bus network. HIL

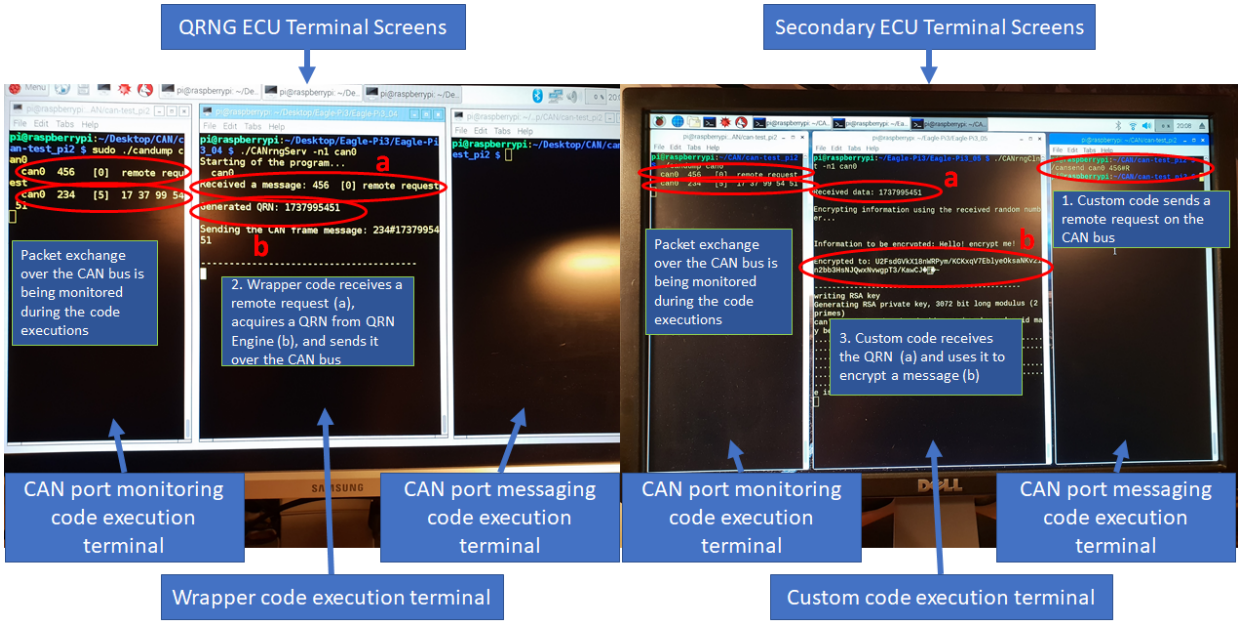

Fig. 7. QRNG ECU and the secondary ECU in execution. (1) The secondary ECU sent a request to the QRNG EC. (2a) The QRNG ECU received the request. (2b) It generated a QRN and replied back. (3a) The secondary ECU received the QRN. (3b) It then used the random number to encrypt some data.

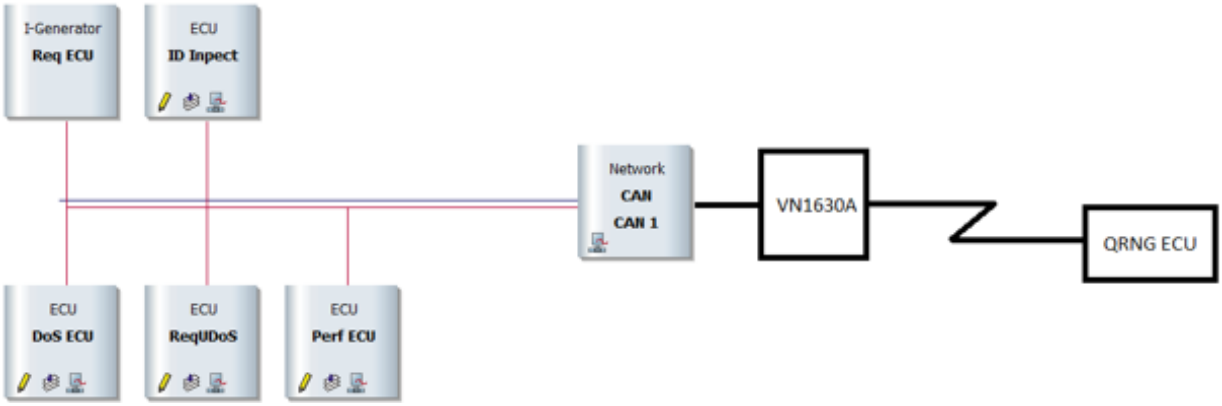

Fig. 8. The logical CAN-bus network setup. Req ECU emulates a manual request for a random number. ID Inspect ECU sequentially sends request messages with IDs from 0x0 to 0x7FF to the QRNG ECU. Perf ECU sends request messages at a high frequency. DoS ECU carries out attacks by emitting CAN messages. ReqUDoS ECU sends requests during a DoS attack.

tool used in our QRNG-ECU testing is commercially provided by Vector Informatik GmbH and consists of CANoe, a software to configure and simulate real-time CAN bus networks, and VN1630, a hardware to provide a physical interface to the simulated network. Then, the QRNG-ECU under test is connected to the simulated CAN bus network via VN1630. This ECU is provided by Crypta Labs with their firmware installed. Due to lack of the camera during testing, Crypta Labs also provided pre-captured images by the camera and customised their firmware to used these images to generate QRNs. This setting does not affect the testing purposes for functionality, performance and availability. For each of these tests is facilitated by the creation of several virtual ECUs in the simulation software CANoe. They are summarised in Table I.

TABLE I
VIRTUAL ECUS SET-UP.

Test type	Virtual ECU	Purpose
Functionality	Req ECU	to emulate a manual request for a random number
	ID Inspect	to sequentially send request messages with IDs from 0x0 to 0x7FF
Performance	Perf ECU	to send request messages at a high frequency
Availability	DoS ECU	to carry out DoS attacks to the QRNG
	ReqUDoS	to benignly send request during DoS attacks

Virtual ECUs are configured in CANoe to facilite tests.

Finally, in the test for randomness quality, Crypta Labs used the prototype system, see Fig. 6, to log about 55M generated

random numbers as input in our test.

The simulated CAN bus network is configured to operate at the standard rate 500kbps. We configured CANoe to connect all 5 ECUs to the CAN-bus. On the physical side, the QRNG-ECU is connected to the simulated CAN-bus network via one of the provided sub-D ports on VN1630A box. The logical setup is depicted in Fig. 8. The physical setup is demonstrated in Figure 9 where three virtual ECUs are hosted by the laptop.

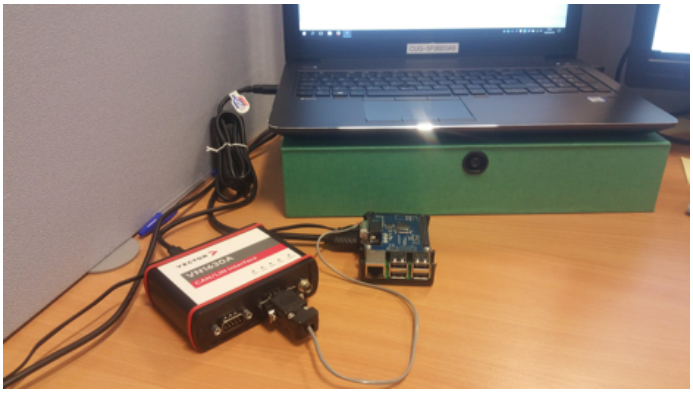

Fig. 9. The physical setup for security testing. CANoe and the virtual ECUs run on the laptop. The QRNG ECU (Raspberry and PiCAN2), in the middle, is connected to the laptop via the VN1630 box (coloured red, while and black).

IV. Test Result and Analysis

We treated the QRNG ECU as a blackbox where only the firmware was updated on the Raspberry Pi. Through its description provided by Crypta Labs, the functionality of the QRNG ECU is to receive random number request, which is processed as follows:

1) If the request has no data, a random number is generated.
2) The random number is then replied back to the CAN bus within a message with ID of 0x234.
3) Otherwise, no random number will be generated.

Functional test

There are two tests for validating the functionality of the QRNG-ECU as discovered by the code analysis.

First, a manual test is carried out. We used Req ECU to send requests with 5 different IDs to QRNG ECU. For each test, we recorded CAN id, data length, expected effect (if QRNG ECU responds to the request with a message whose ID is 0x234), observed effect and finally a verdict (passed if expected is the same as observed, or failed otherwise). The records are detailed in Table II.

In this test, it is clear that all the tests result as expected. However, in some case, the response message with ID of 0x234 from the ECU contains no random number. This must be classified as an error and it will be analysed more in the performance testing.

Second, an automated test is scheduled. We use ID Inspect ECU to send request message with ID ranging from 0x1 to 0x7FF. The ID 0x7FF corresponds to using all 11-bit in the ID field of a CAN message. In this test, a request is sent

TABLE II
Results from functional testing

Test	CAN id	DL	Expected	Actual	Status
1	0	0	Yes	Yes	Passed
2	0	8	No	No	Passed
3	1	0	Yes	Yes	Passed
4	123	0	Yes	Yes	Passed
5	20	8	No	No	Passed

Req ECU sends 5 requests with IDs to QRNG ECU. Each test is recorded with CAN id, data length (DL), expected effect (wherether QRNG ECU to response with a message of ID 0x234), observed effect (yes if QRNG ECU responded, no otherwise) and a verdict (passed/failed).

every 10ms with IDs starting from 0x0. If the response from QRNG ECU contains a random number, the ID of the next request is increased by 1. Otherwise, the next request will repeat the last ID. The test confirms that QRNG ECU responds to all request messages with ID ranging from 0x0 to 0x7FF. However, repeated requests with the same ID as the last one are indeed needed due to the presence of response message from QRNG ECU without a random number.

Performance test

We used Perf ECU to send 30,000 requests to QRNG ECU in 1/2 minute, i.e., a request is sent every 1ms. All requests are valid for QRNG ECU and have the same configuration:

1) CAN id is 60;
2) Data length is 0.

We recorded the response from QRNG ECU. In total, 29998 responses are received. This means 99.99% of the requests are processed. Only 13499 of them contain a random number. This means only 45% of the requests are actually served. These results are illustrated in Figure 10. We expect the low percentage of actual serves to come from a software bug, which gives room for further improvement.

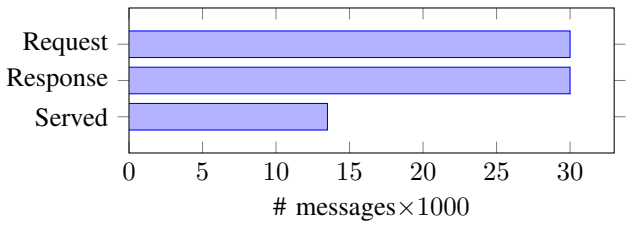

Fig. 10. Performance testing result: With 30000 requests sent, 29998 responses are received (99.99%) and only 13499 of them contain a random number (45%).

Availability test

We use the DoS ECU to carry out three DoS attacks where

1) DoS ECU constantly emits n CAN messages with ID 0 every 1ms;
2) Req U Dos ECU repeatedly emits a request message with ID 60 every 100ms.

In the first attack, n=100. For the subsequent attacks, we relax the attack messages by 10 times. In particular, for the 2nd, n=10, and, for 3rd, n=1. In each test, we scheduled requests by each ECU to send on the CAN bus network, then observed and recorded the actual requests by each ECU registered on the CAN. The results are presented in Table III.

In Test 1 the number of attacking messages by DoS ECU is invasive; messages by Req U Dos ECU are hardly sent on the CAN bus. It is improved in the second and third tests as the attacks are relaxed. Tests 1 and 2 reveal the maximal number of requests that can be placed successfully on the simulated CAN bus. For test 1, the number is $284,284 + 8 + 332 = 284,624$; for test 2, it is $284,030 + 269 + 332 = 284,631$. This indicates a baseline for the maximal amount of data frames that the simulated CAN bus network can handle in 30 seconds. Therefore, the low performance of the QRNG-ECU under DoS Attack is closely related to the nature of CAN bus network.

Randomness test

The Crypta Labs used the prototype system, as in Fig. 6, to collect about 55M generated random numbers. They are used as input for our test of the quality of the random number generator. This is a statistical test which is done by the random number generator testing suite Dieharder [2]. The Dieharder suit incorporates the diehard tests and Statistical Test Suite by the National Institute for Standards and Technology. A test named *runs* checks the correspondence between the number of identical bit runs in a generated bit stream and that of a random one, and the frequency of bit flips. Each test in the suit computes a *p-value* indicating the likelihood that the sequence of tested random numbers can be generated by a purely random number generator. The pass thresholds of these p-values vary for each test. In total, there are 114 different tests carried out with 104 passed, 4 weak and 6 failed. Table IV shows the results. However, we are confident the failures are down to the fact that not enough data has been tested and further tests are planned for larger datasets.

TABLE IV
RANDOMNESSS TESTING RESULT.

Test	Ntup	Tsamples	Psamples	p-value	Assess.
birthdays	0	100	100	0.98314746	PASSED
operm5	0	1000000	100	0.98856574	PASSED
rank32x32	0	40000	100	0.99367566	PASSED
rank6x8	0	100000	100	0.27602164	PASSED
bitstream	0	2097152	100	0.36259481	PASSED
opso	0	2097152	100	0.13629271	PASSED
oqso	0	2097152	100	0.24398521	PASSED
dna	0	2097152	100	0.59938792	PASSED
cnt1s_str	0	256000	100	0.70084853	PASSED
cnt1s_byt	0	256000	100	0.50112894	PASSED
parkinglot	0	12000	100	0.54444845	PASSED
2dsphere	2	8000	100	0.65623976	PASSED
3dsphere	3	4000	100	0.36259517	PASSED
squeeze	0	100000	100	0.08852922	PASSED
sums	0	100	100	0.36526772	PASSED
runs	0	100000	100	0.59823491	PASSED
runs	0	100000	100	0.05432660	PASSED
craps	0	200000	100	0.55273023	PASSED
craps	0	200000	100	0.89971117	PASSED

This table is drawn by Dieharder [2] where *p-values* indicate the likelihood that the sequence of tested random numbers can be generated by a purely random number generator and the pass thresholds of these p-values vary for each test.

V. RELATED WORK AND DISCUSSION

We have presented an ECU prototype for generating random numbers based on the quantum technology by Crypta Labs. This has revealed a number of practical insights into how to apply and extend existing approaches to security testing and analysis for automotive ECUs. The prototype has then been preliminarily assessed in a CAN security testbed for evaluating its functionality, performance and reliability. The implemented prototype managed to function with occasional miss, and keep service up to certain level (45%) of speed performance. While the test result shows the weakness of the ECU prototype against DoS attacks, our baseline analysis pointed out that this is related to the nature of automotive CAN.

Functional testing for automotive ECU development has been a key activity for the automotive industry. While aspects of safety are well understood for such testing, emerging functionality around connectivity, sensing, autonomy and cybersecurity has thrown up considerable challenges for functional and non-functional assurance on new ECU designs. The challenge is compounded by the manufacturers' desire for cost reduction and test automation of testing; cybersecurity is in itself a new challenge. The state-of-the-art on ECU security testing has offered a number of approaches: some based on formal methods [4], [10], some on simulation [1], [5], [7], [19], some on combination of both [3], and some on fuzzy approaches to penetration testing [6].

Simulation approaches generally concern with the need for

TABLE III
AVAILABILITY TESTING RESULT.

Test	DoS ECU		ReqUDoS ECU		QRNG ECU	Percent
	Scheduled	Actual (1)	Scheduled	Actual (2)	Response (3)	
1	3000000	284284	300	8	332	0.12%
2	300000	284030	300	269	332	0.12%
3	30000	30000	300	300	30300	100%

In each test, we recorded the number of requests scheduled by DoS ECU; the actual number (1) of requests by DoS ECU registered on the CAN bus; the number of scheduled requests by ReqUDoS; the actual number (2) of registered requests by ReqUDoS; the number of registered responses from QRNG ECU; and computed the serving percent $= (3)/((1) + (2)) \times 100$.

a testing environment where a large number of test cases can be executed on ECUs under test without making damages to an actual car. Boot and Richert propose a hardware-in-the-loop simulation to create a testing environment for engine ECUs [1]. The simulation emulates internal combustion engines which are modelled in MATLAB/Simulink. These models are implemented on the dSpace real-time hardware by automatic C code generation. In essence, the models receive input from an ECU under test such as the throttle valve position, injection amount and ignition angle and then compute the output of an engine such as the engine speed and sensor signals from the crankshaft, the camshaft and the exhaust system. Similarly Drolia *et al.* developed a testbed for remote ECU recalls [5]. Remote ECU recalls are used to analyse and update ECUs with software bugs. The analysis of ECU software is realised by a monitor implemented in MATLAB. The monitor is validated in the testbed which simulates a cars environment and comprises of nine ECUs (for pedals, control, cabin comfort, transmission, console and propulsion).

Our approach is similar to those using CANoe from Vector. For example, [19] carried out a feasibility study where CANoe is used to test the design scheme of a CAN bus network. Such a design scheme includes the network topology, its components, and speeds. They are implemented in CANoe to derive an environment for testing node functionalities, network load and the occurrence of error data frames. Flower *et al.* have utilised CANoe and Vectors hardware simulators to create a testbed for automotive security [7].

Formal methods are used for automated test case generation. Santos *et al.* model a CAN bus and formalise a threat model in CSP to generate test cases via model checking [4]. Similarly, Cheah *et al.* propose to transform attack tree threat models into CSP specifications to generate test cases automatically [3], making use of the simulation testbed proposed in [7] to provide a testing environment from generating test cases to test execution. Kaindl *et al.* are working on formalising ECU functionality and test environments in semantic specifications to automate test cases [10].

While simulation approaches are powerful and provide for an ideal environment to emulate on-board automotive architectures, specifying security properties remains a challenge. Approaches based formal methods, on the other hand, have the power to predicate security constraints effectively, but are yet to demonstrate effective translation from one environment to the other. This paper only serves to demonstrate the complex nature of such real-world security testing. Emerging approaches to detecting intrusions on CAN [14] hold further promise in baselining component behaviour, and detecting malicious anomalous behaviour [16] [15].

VI. Acknowledgement

Collaboration between Coventry University and Crypta Labs Ltd. has been funded by Innovate UK under the Connected and Autonomous Vehicles 2 Stream 3 FS project titled "Quantum-based secure communications for CAVs" (Reference 132999). Support was provided by The Centre for Connected and Autonomous Vehicles (CCAV) through Innovate UK.

References

[1] R. Boot, J. Richert, H. Schutte, and A. Rukgauer. Automated test of ECUs in a hardware-in-the-loop simulation environment. In *Proceedings of the 1999 IEEE International Symposium on Computer Aided Control System Design*, pages 587–594. IEEE, 1999.

[2] Robert G. Brown. Dieharder: A Random Number Test Suite. https://webhome.phy.duke.edu/˜rgb/General/dieharder.php, Acc. 12/2018.

[3] Madeline Cheah, Hoang Nga Nguyen, Jeremy Bryans, and Siraj A. Shaikh. Formalising Systematic Security Evaluations Using Attack Trees for Automotive Applications. In *Information Security Theory and Practice*, volume 10741, pages 113–129. Springer, 2018.

[4] Eduardo dos Santos, Andrew Simpson, and Dominik Schoop. A Formal Model to Facilitate Security Testing in Modern Automotive Systems. *In Proceedings of Theoretical Computer Science*, 271:95–104, May 2018.

[5] Utsav Drolia, Zhenyan Wang, Yash Pant, and Rahul Mangharam. AutoPlug: An automotive test-bed for electronic controller unit testing and verification. In *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 1187–1192. IEEE, October 2011.

[6] Daniel S. Fowler, Jeremy Bryans, Siraj Ahmed Shaikh, and Paul Wooderson. Fuzz testing for automotive cyber-security. In *48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops, DSN Workshops 2018, Luxembourg, June 25-28, 2018*, pages 239–246. IEEE Computer Society, 2018.

[7] Daniel S. Fowler, Madeline Cheah, Siraj Ahmed Shaikh, and Jeremy Bryans. Towards a Testbed for Automotive Cybersecurity. In *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pages 540–541. IEEE, March 2017.

[8] M. Herrero-Collantes and J. C. Garcia-Escartin. Quantum random number generators. *Reviews of Modern Physics*, 89(1):015004, 2017.

[9] ICS-CERT. Harman-Kardon Uconnect Vulnerability. https://ics-cert.us-cert.gov/advisories/ICSA-15-260-01, August 2018.

[10] Hermann Kaindl, Franz Lukasch, Matthias Heigl, Sevan Kavaldjian, Christoph Luckeneder, and Sebastian Rausch. Verification of Cyber-Physical Automotive Systems-of-Systems: Test Environment Assignment. In *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops*, pages 390–391, April 2018.

[11] Aarian Marshall. TESLA'S QUICK FIX FOR ITS BRAKING SYSTEM CAME FROM THE ETHER. https://www.wired.com/story/tesla-model3-braking-software-update-consumer-reports/, Acc. 12/2018.

[12] NCSC. Example supply chain attacks. https://www.ncsc.gov.uk/guidance/example-supply-chain-attacks, Created 28/1/2018.

[13] Dennis K. Nilsson, Ulf E. Larson, and Erland Jonsson. Creating a Secure Infrastructure for Wireless Diagnostics and Software Updates in Vehicles. In *Computer Safety, Reliability, and Security*, volume 5219, pages 207–220. Springer, 2008.

[14] A. Tomlinson, J. Bryans, and S. A. Shaikh. Towards viable intrusion detection methods for the automotive controller area network. In *2nd Computer Science in Cars Symposium – Future Challenges in Artificial Intelligence Security for Autonomous Vehicles*. ACM, 2018.

[15] A. Tomlinson, J. Bryans, and S. A. Shaikh. Using a one-class compound classifier to detect in-vehicle network attacks. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion 2018, Kyoto, Japan, July 15-19, 2018*, pages 1926–1929. ACM, 2018.

[16] A. Tomlinson, J. Bryans, S. A. Shaikh, and H. K. Kalutarage. Detection of automotive CAN cyber-attacks by identifying packet timing anomalies in time windows. In *48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops, DSN Workshops 2018, Luxembourg, June 25-28, 2018*, pages 231–238. IEEE Computer Society, 2018.

[17] Nhan Duy Truong, Jing Yan Haw, Syed Muhamad Assad, Ping Koy Lam, and Omid Kavehei. Machine Learning Cryptanalysis of a Quantum Random Number Generator. *IEEE Transactions on Information Forensics and Security*, 14(2):403–414, February 2019.

[18] Chris Valasek and Charlie Miller. A Survey of Remote Automotive Attack Surfaces. https://ioactive.com/pdfs/IOActive_Remote_Attack_Surfaces.pdf, 2014.

[19] Fang Zhou, Shuqin Li, and Xia Hou. Development method of simulation and test system for vehicle body CAN bus based on CANoe. In *2008 7th World Congress on Intelligent Control and Automation*, pages 7515–7519, Chongqing, China, 2008. IEEE.