Security Aware Network Controllers for Next Generation Automotive Embedded Systems

Shanker Shreejith
School of Computer Engineering
Nanyang Technological University
shreejit1@ntu.edu.sg

Suhaib A. Fahmy
School of Computer Engineering
Nanyang Technological University
sfahmy@ntu.edu.sg

ABSTRACT

Modern cars incorporate complex distributed computing systems that manage all aspects of vehicle dynamics, comfort, and safety. Increased automation has demanded more complex networking in vehicles, that now contain a hundred or more compute units. As these networks were developed as silos, little attention was given to security early on. However, this has become a key challenge in the automotive domain, as these systems have been shown to be susceptible to various attacks, with sometimes catastrophic consequences. Addressing security in such systems requires consideration of the network and compute units, both hardware and software, and complex real-time constraints. We present an approach that integrates application authentication, message encryption and network access control into a smart network interface, without compromising network determinism. A custom interface with partial reconfiguration support on FPGAs enables seamless integration of security at the interface, offering a level of security not possible with standard layered approaches.

Categories and Subject Descriptors

C.2 [**Computer Communication Networks**]: General—*Security*

1. INTRODUCTION

Modern cars now employ a hundred or more electronic control units (ECUs) that together take care of many vehicle features, from entertainment and comfort, to increasingly critical aspects of the drive train. These networks evolved over time and are built in a modular fashion with each ECU connected to the required network through a network controller. As connectivity within the car and to the outside world increases, concerns have arisen about the security of these previously siloed networks. Indeed, multiple invasive and non-invasive attacks have been shown to be effective in modern vehicles, ranging from simple techniques like fuzzing and replay attacks to more complex attacks involving compromised software on ECUs [1, 2, 3, 4]. Fuzzing attacks involve injection of commands/data on the network in a bid to force ECUs into certain vulnerable modes (like enabling flashing) or to cause them to halt due to unexpected command-data combinations. Replay attacks involve capturing network messages and re-using them at a later time to spoof commands to an ECU. As wireless communication becomes increasingly common in modern vehicles, whether in wireless sensor systems, internet-enabled services, or (future) vehicle-to-vehicle (V2V) communication, new pathways become available to potential attackers without even requiring physical access.

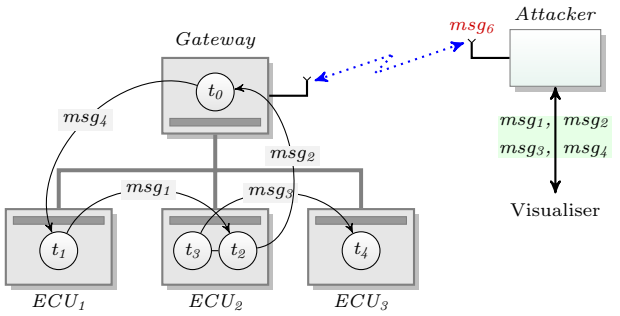

Figure 1: An attacker exploiting a compromised gateway in a vehicle to gain access to internal network messages either as a passive hacker (observer) or an active one (injecting messages/commands).

Automotive networks were originally closed, with only the mandatory on-board diagnostics (OBD) port providing an interface to the outside world. Since these OBD ports provide direct and/or bridged access to both critical and non-critical networks, they represent an ideal point for a hacker to gain access via an (infected) OBD dongle. Malicious software or hardware provides another pathway, mostly introduced through non-approved after-market upgrades. These can be used to launch internal attacks (observations or manipulations) on messages or other ECUs since the network provides implicit full bus access to all components. Fig. 1 shows an example scenario where an attacker is able to observe messages through a compromised network gateway. Once access to the network is gained, it might be possible to install defective software on safety-critical ECUs over the

This work was supported by the Singapore National Research Foundation under its Campus for Research Excellence And Technological Enterprise (CREATE) programme.

DAC '15 June 07 - 11, 2015, San Francisco, CA, USA

http://dx.doi.org/10.1145/2744769.2744907 .

network, compromising their functionality, as was demonstrated in [1]. These exploits are possible since the vehicular networks offer no mechanism to authorise devices that integrate and communicate on a specific network (other than physical connectivity) and no way to prevent compromised ECUs from accessing the network. Attempting to solve these weaknesses in software is challenging because of the logical separation between computation in the ECU, and physical bus access in the network controller.

In this paper, we present a novel approach to this problem with the use of a reconfigurable network controller that incorporates application authentication and network access regulation. The authenticity of an ECU's boot-ROM contents (bootloader, application and configuration) are verified before the network interfaces are enabled, thus preventing a compromised ECU from accessing the network. Furthermore, the network interface integrates data encryption and imposes network access restrictions using obfuscation and cross-layer techniques, thus preventing unauthorised devices from integrating (and thus communicating) over the network. By integrating this functionality in a reconfigurable controller, the scheme does not impact the real-time characteristics of the network and is abstracted from the application. We demonstrate this approach on the time-triggered FlexRay network, with a Xilinx Zynq platform. The proposed method establishes a configurable hardware security layer, which can be built upon to provide adaptable security schemes.

2. BACKGROUND

Vehicular networks are physically separated into high and low performance networks, based on several factors like criticality of functions, latency requirements and communication bandwidth, and are typically bridged via a central gateway. High performance networks link together safety-critical ECUs and sensors, like engine control and drive-by-wire systems, while low-performance networks connect non-critical ECUs like window controls and door locks. These networks are also served by different protocols, like high-speed CAN (HS-CAN) and FlexRay for high performance networks and low-speed CAN (LS-CAN) and local interconnect network (LIN) for low performance networks, each providing the required bandwidth and reliability guarantees necessary for the respective applications. However, telematics and driver assistance systems often require information from both networks, and are often connected directly to both network classes, creating unwanted bridging [1, 2].

Security of in-vehicle communication (IVC) has recently been subject to investigation. In [1, 2], the authors analyse the security of computation and communication systems by exploring different attack vectors on actual vehicles. Their experiments highlight some common weaknesses in existing network architecture like network bridging and configuration request acceptance from a low priority network, as well as protocol limitations, all of which provide multiple attack planes for a potential hacker. While the experimental approach exposed many practical attack possibilities, analytical evaluation of threats and effects has also been explored [3, 4, 5]. These analyses evaluate the threats and damages based on multiple factors (severity, success probability) [3] or classify ECUs based on safety effect levels of threats [4] with extensions to wireless interfaces [5]. Similarly, the *EVITA* project focuses on evaluating the security of vehicular systems, focusing on futuristic V2V and the enabled-vehicles concept. The project evaluates possibilities for formal verification and efficient hardware/software co-design for future communication systems [6, 7].

Techniques have also been proposed to address these challenges to some extent. Standard methods like cryptography and anomaly detection were proposed in [5] to provide data security, though these are not resistant to replay attacks. A scheme based on trust and access control lists to verify message authenticity on CAN-based ECU systems was proposed in [8]. This method provides a higher level of security, but does not prevent an unauthorised device (either newly plugged in or compromised) from accessing network traffic. Software based automotive security solutions are also proposed in [9], but such software must be protected against tampering and manipulation from invasive and internal attacks. Alternatively, automotive hardware-based security modules (HSMs) and secure hardware extensions (SHEs) have been proposed, providing high levels of tamper protection for V2V and IVC [10]. Such HSMs are attached as co-processors, and the security is invoked at a higher level (i.e., after the messages have been received) incurring a latency and power cost.

We aim to integrate application security and network protection within the network interface (NI) of the ECU system, thus abstracting such details from the application, enabling seamless migration to the secure domain. Extending the NI using extensions offers features that do not work at the software level, such as the scheme in [11] which improves energy efficiency using smart network interfaces that can sleep their respective ECUs or [12] that provides data layer extensions for enhanced communication.

Our extensions to a standard NI on reconfigurable hardware enable integration of network protection without incurring additional latency, thus preserving timing guarantees. Integrating security at the interface also enables us to leverage network properties to provide dynamic run-time security and eliminates software intervention during application authorisation.

3. CONCEPT

An effective security mechanism should (1) authenticate and authorise the software on all ECUs (in the secure network) and (2) must only allow authorised devices to access the broadcast network. While HSMs/SHEs provide methods to implement software authentication/authorisation, the broadcast nature of bus protocols allows any plugged in off-the-shelf component to observe the bus and integrate on to it by deducing network parameters. Our method aims to circumvent these issues with the enhanced NI that tightly integrates security within and around the communication controller (protocol implementation).

To authenticate software, an agent external to the ECU must be involved, like the HSM unit which is attached as a co-processor. We integrate this functionality within the custom NI whereby, the interface logic reads the contents of the boot-ROM and verifies the authenticity of the contents using a one-way hash function, whose expected value is embedded in the hardware at production. If the authenticity is verified, the authorisation to use this hardware by the application is verified against the unique identifier of the hardware, which is generated at run-time by a circuit that maps intrinsic properties of the device to its unique identifier. Since intrin-

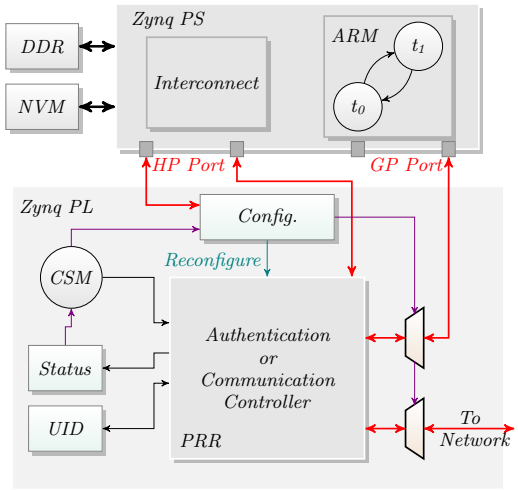

Figure 2: High-level system architecture on Xilinx Zynq.

sic properties cannot be exactly controlled by manufacturing processes and are difficult to predict, no two identical devices generate the same identifier using such circuits, thus providing unique method to tie down software for a particular hardware platform. Only after the boot-ROM contents are authenticated and authorised for this hardware, will the communication controller (CC) be enabled, thus preventing a compromised ECU from accessing the network.

Once an ECU has been authorised by its NI, it can start normal communication over the shared bus. However, since the bus is of broadcast nature, malicious hardware on the bus could still decode the communication and integrate on to the bus. To circumvent this, we integrate synchronised timestamping and cross-layer encryption using light weight ciphers within the CC, which provides configurable data security and obfuscates protocol header information. The timestamps prevent replay attacks since stale data is rejected at the CCs of receiving ECUs. The obfuscated headers mean that an unauthorised device cannot decode the protocol parameters by observing them. Hence, only devices authorised with a pre-shared key (from the OEM) can integrate and communicate over the network. We also incorporate a method to update the keys at run-time using symmetric cryptography, which can be further improved using light weight asymmetric schemes.

4. CASE STUDY

In this section, we describe the system architecture of our case study for a FlexRay-based ECU on a Xilinx Zynq platform. We use partial reconfiguration (PR) to dynamically invoke the authentication and CC modules as and when required, thus optimising area and power consumption. We integrate an SHA-1 hashing function and the PRESENT lightweight cipher [13] for software encryption and data ciphering. The proposed concept can be used with other time-triggered network architectures which may eventually replace FlexRay in vehicles and using other standard hashing/cipher functions.

Hardware Architecture: A high level architecture of the proposed system based on the Xilinx Zynq platform is shown in Fig. 2. The Zynq processing system (PS) integrates highly capable ARM cores and a number of peripheral devices like DDR Memory Interface, Non-volatile memory interface (over SPI), Ethernet and others. The PS is tightly coupled with programmable logic (PL), which can be used to implement custom functions and/or accelerators, communicating over High Performance (HP) or General Purpose (GP) ports. The application code, boot-loader and the PL bitstream(s) are stored in non-volatile memory (NVM), as is common for automotive ECUs.

Within the PL, we instantiate the software authentication mode or the FlexRay CC in a partially reconfigurable region (PRR). During startup, the PRR instantiates the authentication logic by default. The authentication logic comprises an SHA-1 one-way hashing function and a Ring Oscillator-based Physically Unclonable Function (RO-PUF), along with the supporting logic in the static region to support communication with the PS. The *UID* register is a configurable width register (8-bit to 128-bit) that stores the unique identifier for the hardware-software combination on this ECU and is later used by the FlexRay network as the identifier of the specific ECU. The *Status* register holds the status of the software authentication process and is used for enabling the interface MUXes that connect the PRR to the FlexRay bus and to the PS (for configuring the FlexRay CC). The control state machine (*CSM*) is responsible for initialising data movement between the PS and PL (for authorisation) and also for initialising the reconfiguration of the PRR to load the FlexRay interface, once ECU software is authorised. The *Config.* module manages reconfiguration of the PRR.

Authentication Function: During system initialisation, after the Zynq PS has completed the boot sequence and programmed the PL with the default bitstream, the *sys_init()* function initialises the interfaces to the PL logic. The Control State Machine (CSM) in the PL logic then initialises a DMA read from the non-volatile memory to compute the hash value of the memory content, including the bootloader, the default bitstream for the PL and the application software (called the *boot image*). The DMA reads are directed into the hashing function and are double buffered to improve the performance. The SHA-1 core is custom designed and operates on 16 32-bit blocks of data (size of DMA burst) to iteratively compute the SHA hash of the entire *boot image*.

In parallel, the physically unclonable function (PUF) module generates a 128-bit hardware identifier (HID), which is combined with the SHA hash to authorise the software on this hardware. We use a configurable RO-PUF function with 128 instances of the configurable ring oscillator (RO), each instance contained within a single logic block (CLB), based on the design in [14]. This design allows for accurate reproduction of the hardware signature, since the routing within each RO is completely constrained to the CLB and its associated switch box (interconnect). Once the SHA-1 hash and HID are generated, the software hash is authenticated against the hard-coded *SAR* register value, while the HID and software hash are combined and hashed to determine if the software is authorised to run on this specific hardware by matching it against the hard-coded *SHAR* register value. Once the software and hardware are authenticated and authorised (as valid or invalid), the status register is updated and a reconfiguration may be triggered depending on its value. If authorised, the CSM triggers the *Config. Controller* to read the (cached) bitstream data corresponding to the FlexRay CC and enables the interfaces to the bus, which are otherwise disabled to prevent even a forced recon-

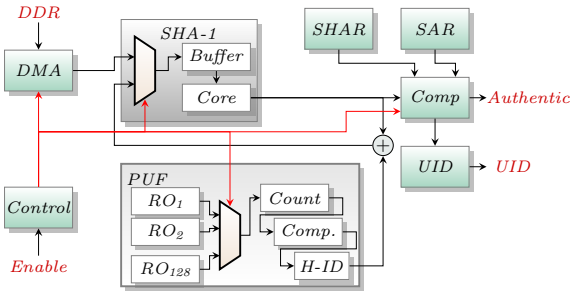

Figure 3: The Hardware-Software Authentication logic

figuration from the PS to be futile. The architecture of the authentication function, expanded out from the PRR block is as shown in Fig. 3.

Once authenticated, corrupt software could still be forced onto the PS at run-time (via JTAG for example); however, such programming triggers the PS reset, which is also wired out into the hardware logic. This run-time reset disables the interface multiplexers (from CC to PL and CC to FlexRay bus) which stay disabled until the hardware is power cycled, forcing the system to boot from the NVM. This prevents invasive attacks from employing non-persistent run-time manipulations on application code. Any persistent alteration (requiring the code to be changed in the NVM) will be detected as a violation by the hashing function, preventing the system from authenticating the software and loading the CC.

Secure FlexRay Network Interface: The FlexRay CC is the implementation of the FlexRay communication protocol, composed of the Protocol Engine (PE) and the Host Interface (HI) modules. The PE implements the protocol specifics like clock synchronisation and encoding/decoding, while the HI implements generic address-data interface to the application/host processor.

To integrate data security into the communication system, we have integrated a timestamp module and a light weight PRESENT cipher into the datapath of the encoding/decoding blocks in an otherwise standard FlexRay CC. By utilising custom data widths and extensive pipelining, we have ensured that this does not introduce additional latency to the data flow in the system, and is confined within protocol boundaries, as discussed in [15]. Even though introduction of timestamps increases the entropy of data being exchanged making attacks difficult, it does not prevent a newly plugged-in (or compromised) unauthorised device from accessing communication on the network.

To achieve this, we must prevent unauthorised devices from integrating onto the network. The protocol headers encompass information about the communication schedule and the configuration of the network in plain text, which can be observed by a unauthorised device to recover the protocol parameters. Knowledge of these and the communication schedule can be used to generate a valid configuration for the unauthorised device, which could then integrate on to the network and manipulate the messages.

To circumvent this, we extend the controller so that the protocol headers are also obfuscated by the cipher logic. The first byte of the protocol header containing the flag bits is left untouched, but the following 4 bytes that comprises the slot number, cycle number and payload length (along with header CRC) are combined with the 4 byte timestamp

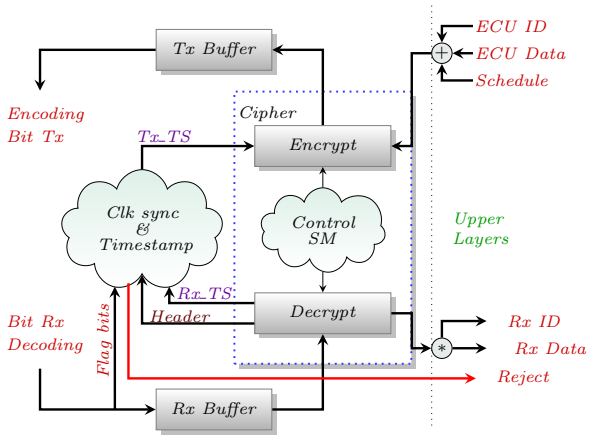

Figure 4: Enhanced datapath at the cipher logic for incorporating header obfuscation.

(Tx_TS) to form the first 64-bit block. This is then encrypted with the pre-shared key, over 8 cycles (configurable, up to 32), while the first 64-bits of ECU data is prefetched into the buffer. To increase entropy, the data may be (optionally) padded with the timestamp (2 or 4-bytes), sacrificing bandwidth. This data is encrypted with a timestamp-based-key which is a combination of the header timestamp and the pre-shared key. This approach ensures that there is large entropy even with identical data (and no timestamp padding) with further increased entropy in the case of data padded with timestamp. This is possible in time-triggered networks since all nodes are synchronised. The data may be encrypted with a larger number of cycles (up to 482 cycles) without affecting the latency of the system, since this can be hidden within the encoding/transmission delay of the preceding 64-bits.

At the receiving end, the byte-decoded data is read directly into the PRESENT decoding buffer. The first byte of the frame header is passed through untouched, allowing the protocol defined startup and synchronisation logic to work without changes. The remaining 4-bytes along with the timestamp are decrypted with the same pre-shared key to determine the actual protocol information (slot, cycle, and payload size), which is used by nodes to integrate onto the network. Since only authorised devices are preconfigured with the pre-shared key, this scheme ensures that unauthorised devices cannot integrate on to the network, as they cannot determine network parameters from observing the bus.

The remaining received data is decrypted in a similar manner, but by using the timestamp-based-key regenerated from the decrypted timestamp (Rx_TS) and pre-shared key to recover the original data. Since the encryption/decryption latencies are deterministic, the timestamp validation system can add this deterministic offset to time-validate the received message and may discard the same if messages are older than a configured threshold, protecting the ECU from replay attacks. The altered datapath for the timestamp-based header obfuscation and data encryption is shown in Fig 4.

Run-time alteration of cipher properties: As mentioned, a pre-shared key and default cipher configuration is loaded into the CC of the critical ECUs by the vendor to ensure that only authorised parts are used. This key is used

to obfuscate the schedule and data during the network integration phase. To further enhance security, it is possible to change the encryption scheme periodically at runtime. Once integrated, the secure gateway can trigger a new cipher configuration using an (encrypted) network management vector (NMV) message, that is decoded by the CC. The NMV provides the new number of rounds for each segment (header/-data) and the new pre-shared key (which can be generated using standard algorithms). The nodes adopt this new configuration at the start of the next cycle, ensuring synchronised operation. Handling the alteration of cipher properties within the CC abstracts these details from the application and does not provide a path for software hacks to access this information.

5. EXPERIMENT SETUP AND RESULTS

The proposed system is evaluated in two steps. First the software tamper protection is evaluated on a Zedboard featuring a Xilinx Zynq XC7Z020 to measure the boot time, determinism of the hardware identifiers, overall latency and resource overheads. Next, the run-time network access control, data encryption and enhanced entropy of frames are evaluated on a Xilinx AC701 board by integrating multiple ECUs with authentic software (authentication block is removed due to area constraints).

Table 1 shows the comparison of resource overheads of the proposed system, compared to a standard implementation of the FlexRay NI (without any extensions) on the Zynq. Incorporating the cipher and datapath extensions to achieve network access control in both channels results in a 115% overhead in flipflops (FFs) and 59% in Logic (LUTs) at the NI alone. However, the overall implementation (NI including PR support) still utilises under 33% of resources on the low capacity XC7Z020 device with only a marginal increase in power over the standard NI (38 mW increase). PR allows the non-concurrent authorisation and NI blocks to be located in the same physical region, reducing the overall resource requirements.

To evaluate the hardware-software authentication, the implementation was tested on multiple Zedboards and the Xilinx ZC702 development board featuring the same XC7Z020 device. It was observed that the HID generated by the RO-PUF differed by 34 bits on average (16 bits minimum) between the different boards for the same configured challenge value, whereas the generated HID had no appreciable variation on the same board for over a few thousand runs at room temperature. Tampered software on the other hand resulted in large variations in the hash value, with a single line edit resulting in more than 80 bits difference. With the *boot image* read directly from the NVM, the authentication function took 118.3 ms to authorise the software while the reconfiguration operation took 62.5 ms to load the secure CC (once authenticated). Alternatively, the *sys_init()* routine can buffer the NVM contents (including CC bitstream) in DDR memory, and overlap data movements allowing authentication to be completed in 66.5 ms, and reconfiguration in 4.4 ms (to load the secure CC). However, this could be prone to tampering with DDR contents (via software or otherwise). The default scheme buffers the CC bitstream only, which provides high reconfiguration speeds (4.4 ms) with reasonable authentication latency (118 ms) without compromising security.

We evaluate the network security and data encryption

Table 1: Comparison of Resources on XC7Z020.

Function	Submodule	FFs	LUTs	BRAMs	DSPs
standard CC	PMM	225	550	0	0
	CSS	1861	3508	2	1
	MIL(x2)	851	1393	2	0
	CHI	1676	2060	10	1
	Total	5491	8939	16	2
secure CC	PMM	225	555	0	0
	CSS	1861	3525	2	1
	Cipher(x2)	2548	2379	4	0
	MIL(x2)	1380	1635	3	0
	CHI	1676	2086	10	1
	Total	11618	14194	26	2
S/W Auth.	PUF/SHA	4665	6574	1	0
	Static	3895	3619	1	0
	Total	8560	10193	2	0
Overhead (%)	CC	6127	5255	10	0
		115%	58.8%	62.5%	0%

scheme by integrating 4 ECUs (3 authorised and one attacker, all based on MicroBlaze soft cores and a FlexRay CC), each running tasks that trigger data exchange over the network in every cycle, on a single FPGA device (XC7A200T on the AC701 board). For this evaluation, we have assume that the ECUs are running authentic software, since the single chip does not offer sufficient resources to manage reconfiguration for multiple ECUs. The FlexRay schedule uses a 10 ms cycle at the full 10 Mbps bitrate, with the CCs operating at 80 MHz and the MicroBlaze at 100 MHz.

Fig. 5 shows the normalised entropy (correlation) of the transmitted frames (with 1 ms timestamp accuracy), when the authenticated secure CCs integrate and start communicating over the network. The timestamped header, when encrypted over 5,8 and 16 rounds with the same pre-shared key (cases 2 to 4) results in much higher entropy than the standard header from the same device (single slot, all cycles). The frame headers appear as noise to any unauthorised device, preventing it from integrating onto the network, since the clock synchronisation parameters cannot be extracted from a pair of frames (adjacent cycles) without decrypting them, as observed in our experiment with the attacker ECU. In fact, attacker is forced to quit the integration process after exceeding the number of failed attempts, as required by the protocol. Fig. 5 also shows the improvement over entropy of static data (case 5) achieved by timestamping alone (case 6), encrypting with time-varying key without and with timestamped data (case 7 and 8 respectively). The non-timestamped data when encrypted with the time-varying key produces nearly the same impact as encrypting the timestamped data, without consuming additional bandwidth required to incorporate the timestamp in the data segment.

We observe that the header obfuscation causes a delay of 125 ns (for the default 8 round decryption of header) from the arrival of header bytes to the decoding of protocol parameters like the slot number, cycle number, and payload length, compared to the standard CC. However, these pa-

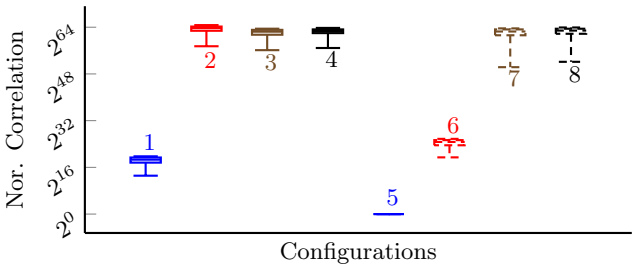

Figure 5: Entropy of obfuscation scheme 1: Normal Header, 2-4: Header via different rounds, 5: Static Data by Static key, 6: Timestamped Static Data (unencrypted), 7 : Static data with time-varying key 8: Timestamped Data with time-varying key

rameters are only used by the upper layers of the protocol (like medium access) and need only be validated after receiving a complete frame (including the frame CRC), unlike the flag bytes. Hence this delay does not impact protocol behaviour.

Finally, we triggered a run-time cipher adaptation by forcing one of the ECUs to send a NVM message changing the key and number of rounds. We observed that the NVM was decoded by CCs on the fly and applied changes within 5 cycles of reception of the entire frame. The round-key generation, consumed n cycles (for n rounds) over this reception delay, with the double key buffer enabling the new round keys to be generated without affecting the current encryption scheme. This enabled instant migration to the new cipher parameters at the end of current cycle, with all authorised ECUs moving to the new parameters synchronously at the cycle boundary.

6. CONCLUSION

Securing vehicular networks is of paramount importance as vehicles increase in connectivity. While software and hardware-based security mechanisms have been proposed, they are restricted to either the computational or network planes, and often incur additional latencies. We have presented an integrated approach to application authentication and network security, that crosses the layers in automotive networks. Malicious software is counteracted through an authentication scheme that prevents network communication in the case of a mismatch. Attacker ECUs are prevented from joining the network through a robust obfuscation scheme that hides network timing parameters from unauthorised ECUs. We demonstrated this system using the Xilinx Zynq with partial reconfiguration allowing sharing of resources, and demonstrate that the scheme does not impact communication latencies and hence the determinism of the system. The same method can be extended to other time-triggered protocols which might eventually supersede FlexRay, and we aim to further enhance the system by integrating novel key-exchange mechanisms and policies at higher layers for true dynamic security.

7. REFERENCES

[1] 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 Symp. on Security and Privacy (SP)*, pages 447–462, May 2010.

[2] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, Tadayoshi Kohno, et al. Comprehensive Experimental Analyses of Automotive Attack Surfaces. In *USENIX Security Symposium*, 2011.

[3] Olaf Henniger, Ludovic Apvrille, Andreas Fuchs, Yves Roudier, Alastair Ruddle, and Benjamin Weyl. Security requirements for automotive on-board networks. In *Proc. Int. Conf. on Intelligent Transport System Telecommunications (ITST)*, 2009.

[4] Dennis K Nilsson, Phu H Phung, and Ulf E Larson. Vehicle ECU classification based on safety-security characteristics. In *Proc. Conf. on Road Transport Information and Control*, 2008.

[5] Ivan Studnia, Vincent Nicomette, Eric Alata, Yves Deswarte, Mohamed Kaaniche, and Youssef Laarouchi. Survey on security threats and protection mechanisms in embedded automotive networks. In *Proc. Conf. on Dependable Systems and Networks Workshop (DSN-W)*, 2013.

[6] Hendrik Schweppe, Benjamin Weyl, Yves Roudier, Muhammad Sabir Idrees, Timo Gendrullis, Marko Wolf, Gabriel Serme, Santana Anderson De Oliveira, Herve Grall, Mario Sudholt, et al. Securing car2X applications with effective hardware software codesign for vehicular on-board networks. *VDI Automotive Security*, 27, 2011.

[7] Gabriel Pedroza, Muhammad Sabir Idrees, Ludovic Apvrille, and Yves Roudier. A Formal Methodology Applied to Secure Over-The-Air Automotive Applications. In *Proc. Vehicular Technology Conference (VTC Fall)*, pages 1–5. IEEE, 2011.

[8] Andre Groll and Christoph Ruland. Secure and authentic communication on existing in-vehicle networks. In *IEEE Intelligent Vehicles Symp.*, 2009.

[9] Tim Leinmüller, Levente Buttyan, Jean-Pierre Hubaux, Frank Kargl, Rainer Kroh, Panos Papadimitratos, Maxim Raya, and Elmar Schoch. Sevecom-secure vehicle communication. In *Proc. of IST Mobile Summit*, volume 2006, 2006.

[10] Marko Wolf and Timo Gendrullis. Design, implementation, and evaluation of a vehicular hardware security module. In *Proc. Information Security and Cryptology (ICISC)*, pages 302–318. Springer, 2012.

[11] C. Schmutzler, A. Lakhtel, M. Simons, and J. Becker. Increasing energy efficiency of automotive E/E-architectures with Intelligent Communication Controllers for FlexRay. In *Proc. International Symposium on System on Chip (SoC)*, 2011.

[12] S. Shreejith and S.A. Fahmy. Extensible FlexRay Communication Controller for FPGA-Based Automotive Systems. *IEEE Transactions on Vehicular Technology*, 64(2):453–465, 2015.

[13] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: An ultra-lightweight block cipher. In *Proc. Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, 2007.

[14] Xin Xin, J Kaps, and Kris Gaj. A configurable ring-oscillator-based PUF for Xilinx FPGAs. In *Proc. Euromicro Conf. on Digital System Design (DSD)*, pages 651–657. IEEE, 2011.

[15] S. Shreejith and S .A. Fahmy. Zero Latency Encryption with FPGAs for Secure Time-Triggered Automotive Networks. In *Proc. Int. Conf. on Field Programmable Technology (FPT)*, pages 256–259, 2014.