Hardware-assisted Security in Electronic Control Units

Secure Automotive Communications by Utilizing One-Time-Programmable Network-on-Chip and Firewalls

George Kornaros
Technological Educational Institute of Crete – Department of Informatics Engineering, Heraklion, Crete, GR

Othon Tomoutzoglou
Technological Educational Institute of Crete – Department of Informatics Engineering, Heraklion, Crete, GR

Marcello Coppola
STMicroelectronics, Grenoble, FR

With emerging smart automotive technologies, vehicle-to-vehicle communications and software-dominated enhancements for enjoyable driving and advanced driver assistance systems, the complexity of providing guarantees in terms of security, trust and privacy in a modern cyber-enabled automotive system are significantly elevated. New threat models emerge that require efficient system-level countermeasures. This article introduces synergies between on- and off-chip networking techniques to ensure secure execution environment for electronic control units. The proposed mechanisms consist of hardware firewalling and on-chip network physical isolation, which mechanisms are combined with system-wide cryptographic techniques in automotive CAN-bus communications to provide authentication and confidentiality.

As automotive firms increasingly implement a full range of emerging smart-car strategies to produce effective systems that deliver sophisticated environments for safe and enjoyable driving, so are increasing the possibilities for hackers and counterfeiters not only to gain access to important layers of software but also to exploit idiosyncrasies in hardware and break into devices ranging from infotainment, cameras, and vehicle communications. Today’s vehicles integrate System-on-Chip (SoC) with powerful CPUs to handle not only control functions but also the processing of

This article has been accepted for publication in a future issue of IEEE Micro, but has not been fully edited. Content may change prior to final publication.

web content, DVD playback, and various A/V entertainment formats. Modern vehicles prominently use CAN-bus due to low-cost and reliability, but communications become complex, e.g., to support multimedia distribution in all seats via the Ethernet AVB, Flexray or MOST protocols. SoCs need to specialize functions and dedicate cores to ensure the prompt execution of safety and other tasks that mandate real-time processing.

Each new feature or electronic module included in a modern car increases the attack surface, which involves the signals transmitted among Electronic Control Units (ECUs) from the in-vehicle network perspective, the functionality performed in software that range from critical tasks to infotainment, and the security assets (e.g., keys). As shown in Figure 1, automotive electronic systems become more distributed, more software-controlled and connected than ever, with in-vehicle architecture or Vehicle-to-X (V2X) communication. Due to the increasing potential for malicious attacks using the multitude of attack surfaces in modern vehicles, it is mandatory to develop hooks and methods that work synergistically to secure both on-chip network and external vehicle communications.

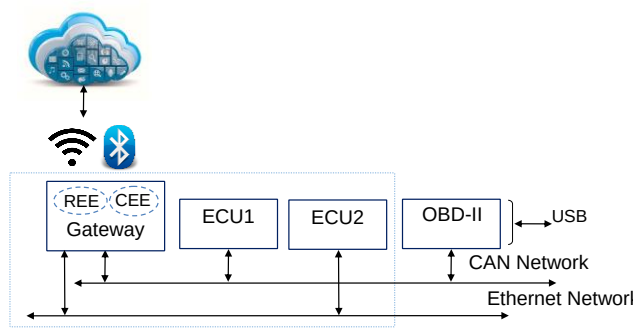

Figure 1 Connectivity along with diverse software of open-platform automotive systems increases attack surface; a gateway today can support multiple execution environments (e.g., rich- and critical-) to provide protection, but light-weight ECUs, OBD-II hubs are traditionally more exposed to attacks

Given the limited bandwidth and payload size of the CAN protocol together with the low computation ability of microcontrollers, security and authentication mechanisms tend to be lightweight in order to keep other system design requirements satisfied. Even though today's gateway nodes that enable wireless communications to have the capacity to run advanced vehicular mixed-critical runtime environments, while dependably enforcing the security and reliability of the underlying information technology, still, most ECUs lack the capability to host isolated execution, run cryptographic and anti-malware services. Even if manufacturers attempt to consolidate the big amount of network interfaces towards the vehicle within only one single point of access and to apply firewalling services against attacks from outside and against information drain from inside, the increased surface that comes with smart cars with highly complex cyber-physical interactions, as well as with increased communication channels requires more efficient security solutions.

Embedded devices' Operating Systems (OSes) are usually equipped with sandbox mechanisms (e.g., "DroidBox: Android application sandbox", http://code.google.com/p/droidbox/) to prevent malicious applications illegally gaining access to sensitive data or compromising other applications, i.e., to provide a Trusted Execution Environment (TEE) for mobile applications. In this scope ARM's TrustZone technology enables secure services to run in the secure world of the processor. However, a CPU with TrustZone security extensions only provides an isolated execution environment, but not a trusted one, since it can't attest to the user or an external verifier that the software running inside the environment is untampered and trustworthy. To guarantee the execution of authentic code a.k.a., *secure boot*, it is mandatory to trust the initial hardware and boot code, which then this trust can be extended to code loaded and executed later by using cryptographic techniques such as digital code signing. No single technique will prevent compromise; security must be built up in layers upon a secure foundation. The secure foundation is the root of trust and starts with secure hardware. Software protection methods are always good to use for

flexibility, such as asymmetric cryptographic exchange of initial keys or session keys, but hardware primitives and mechanisms should be used to strengthen the software methods.

In this work we introduce I/O Secure Communication Unit (IOSCU) to provide hardware-assisted protection of resources and data by means of ensuring filtering of transactions by illegal or unauthorized software running on an ECU. Through physical breaking of on-chip communication links we achieve isolation of resources. One such resource involves preconfigured rules that enforce discrimination of accesses depending on the authentication credentials of the initiator, i.e., benign or malware software. The key idea is to enhance the chained root of trust in booting with one-time programmable routing paths in a NoC, together with hardware-assisted firewalling and combining with authentication mechanisms of in-vehicle ECUs communications. In practice, the mechanisms described in this work go beyond conventional security approaches by linking on-chip hardware firewalling and authentication mechanisms with anonymizing and encryption of CAN-bus frames and system-wide authentication.

Next, we identify past research efforts to achieve security using hardware techniques. Then, we present our IOSCU architecture and discuss the details and ramifications of the synergistic on-chip NoC-based protection mechanisms with CAN-bus authentication-encryption, which scheme to the best of our knowledge is the first to bind and design strong joint on-chip and off-chip security methods. Finally, we comment on orthogonality of our approach with other solutions for secure in-vehicle communications.

RELATED WORK

Hardware-assisted protection techniques are increasingly adopted to provide strong security services. Table 1 summarizes related works. As vehicle networks become increasingly open and software-dominated, application vulnerabilities can be exploited to take control of processing elements to perform malicious actions (e.g., via buffer or stack overflows[1]). Further, physical access to vehicle's networks, through OBD-II ports for example, designate an ideal point for hackers to gain access. Hence, works enforce isolation, through security-aware routing using the on-chip interconnect NoC "firewalls",[2-4] or memory protection via hardware-assisted secure inter-process communication.[5] Since there can be diverse security requirements by different applications that run on the processor and by different cores inside the SoC, security regions can be applied in the system to provide isolation.

Researchers also propose authentication support by matching the source or routing history of the packet. Works have presented an integrated approach to application authentication,[6-7] while Shreejith[7] also supports security at the network interface by integrating data encryption and network access restrictions using synchronized timestamping and cross-layer encryption using light weight ciphers within the controller, for configurable data security and obfuscation of the protocol header information.

The goal of our work is to provide hardware-assisted protection against powerful software attackers, including those who could gain system privileges; in addition to related works, we support one-time-programmable firewall rules configuration and an authentication method isolated even from privileged software. Our attacker model, however, excludes hardware attacks such that attackers with physical access on each microcontroller are potentially able to acquire sensitive data while circumventing the protection of IOSCU, e.g., by chip probing or fault injection.

Table 1: Summary of hardware-based protection solutions

Protection Technique	Security Service	On-chip Network	Security Regions	Programmability	Overheads
Sepulveda et al.[2]	Access control Authentication Confidentiality	Yes	Yes	Dynamic via Security Manager co-processor (software)	Large (per Router)

Fiorin et al.[3]	Access control	Yes	Yes	Dynamic	Modest (per Target using TCAM)
Grammatikakis et al.[4]	Access control	Yes	Yes	Dynamic	Modest (per Initiator)
Shreejith et al.[7]	Authentication, lightweight encryption	No	No	Boot-time authentication	Custom Communication Controller
Kornaros et al.[6]	Authentication Confidentiality	No	No	Dynamic authentication, Firmware-over-the-Air	Modest
IOSCU	Access control Authentication	Yes	Yes	One-Time-Programmable (at boot-time)	Low (per Target: Memory region or Memory-mapped Device)

SYSTEM ARCHITECTURE

The proposed synergistic security solution overcomes the inherent low computational capability of ECUs to handle complex security algorithms and additionally our mechanisms allow open-access components to make security decisions without requiring continuous connectivity to external parties.

Figure 2 shows the IOSCU internal architecture, that is, the integrated Network-on-Chip with the Firewall and the Authentication Finite State Machine (AFSM), which provide secure and authenticated communication between CPU and any memory mapped peripherals. The logical memory regions that are mapped to the address space of the NoC initiator are all be subject to access rules written to the Firewall. The accesses are differentiated based on the privilege level of the transaction initiator and on the type of access. During reset time the *Config-Reroute* unit allows the CPU to access the Firewall configuration port. Therefore, the CPU can configure the Firewall with the desired rules at boot time. When the CPU commands the *Config-Reroute* unit to disconnect the routing paths to the Master Interface MS1, then, all routers are re-programmed to route all transactions to Master Interface MS2. This transaction to the *Config-Reroute* unit is a one-time write operation, where the circuitry automatically disables any additional transaction. Hence, this memory region does not need further firewalling during operation afterwards, since each router performs packet forwarding based on the destination memory address. Therefore, the *Config-Reroute* essentially remaps the address space when reprogrammed to divert packets to MS2 interface.

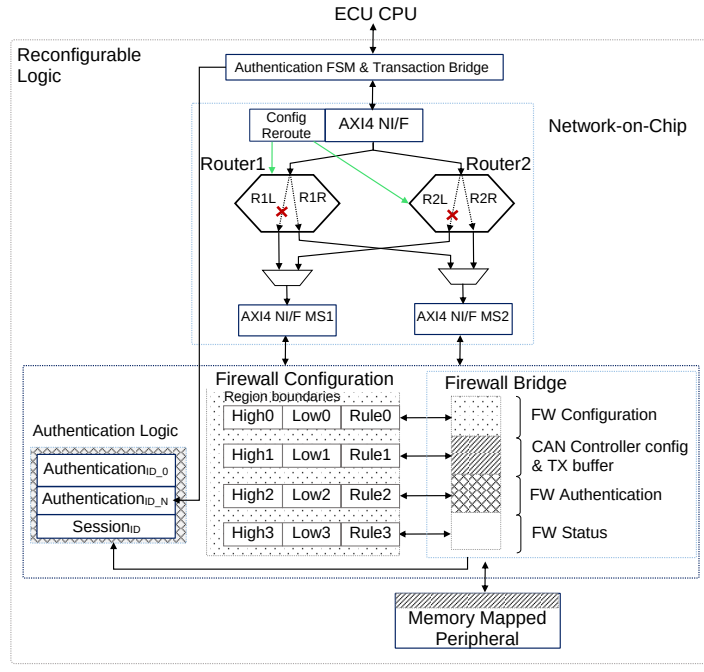

Figure 2. Embedding IOSCU: a smart one-time-programmable NoC extended with a firewall service to provide access control to any memory-mapped peripheral, such as a CAN-bus controller

SYSTEM OPERATION

We consider the OBD-II node shown in Figure 1 to integrate the IOSCU extensions. The proposed enhancements stem from the dual-fold requirement to ensure secure networking among authenticated nodes at system level and at the same time to isolate authentication and protection mechanisms from software layers. At boot time the OBD-II node requires the Communication-Certification Authority (CCA) node, i.e. Gateway, to send the secret *AuthenticationID*. This *AuthenticationID* serves to enable the OBD-II node to communicate with the network (otherwise the CAN frames that the OBD-II may send will be discarded as untrusted) and additionally, to enable the application to authenticate itself through the credentials it provides; these credentials are used to construct the *SessionID* throughout the active interval of the external connection. Multiple *AuthenticationIDs* enable differentiation of the privilege/access level by activating different rules and consequently access to different services in the vehicle enabled by the CCA. The first bootloader initially triggers the AFSM to request from the CCA node and fetch one *AuthenticationID* in the Firewall's *AuthenticationID register* without forwarding it or even notifying the software. All *AuthenticationID* registers are totally isolated from the software (i.e., non-memory mapped). After boot time, when rerouting is complete, any transaction to the Firewall's configuration address range will appear in the Firewall Bridge port. This address range is banned by default through filtering by the permissions that are set in the first Firewall configuration entry (*High0*, *Low0* and *Rule0* registers). Thus, after boot time, any attempt to change the rules is not possible.

Alternatively, one device-ECU *AuthenticationID* can be generated in the backend of the vehicle manufacturer or of the specialized service provider and stored inside the device in a tamper resistant manner during the design/manufacturing process. This removes the need to request an *AuthenticationID* at boot time.

During normal operation, an application should log in to the IOSCU unit by providing credentials (i.e., the identification) that are transferred to the special *SessionID* register. This register is mapped in the third firewall region (FW Authentication Logic) that is only writeable, while the read permission is disabled by the corresponding rule. Thus, once the *SessionID* is stored internally in the Firewall, it cannot be retrieved to verify its value or format.

The authentication control logic is responsible to unlock *Rule1* for the corresponding peripheral, i.e., the CAN controller configuration register and transmit logic. Unless the authentication controller enables *Rule1*, all accesses to the CAN controller are prohibited. If the given *SessionID* matches one of the *AuthenticationIDs* and the auxiliary bus signals (e.g., AXI4 AxPROT) match the rules, then the read and/or write attributes are enabled. Hence, the connection allows the reception and/or transmission of CAN frames over the network.

The fourth region, protected by *Rule3*, controls the access to status register that accounts for transaction violation attempts. For instance, if an adversary software issues transactions to send a CAN frame, then the firewall will prohibit such unauthorized access to CAN control register and will only log this event. The CPU is never notified, nor has access to this protected status register, unless has proven its authenticity and the corresponding privileges that comes out of *Rule3* enables accessing the register.

CAN Network Identification

The network may be an untrusted and/or compromised network. To protect sensitive information like one *AuthenticationID* that travels in the network during boot time, the data need to be encrypted before sending it over the CAN bus. The receiver that exploits the IOSCU in synergy with the authentication FSM prevents forwarding such sensitive data to the software stack.

In particular, to ensure confidentiality and anonymization our security protocol involves a set of pre-shared securely stored encryption keys. The OBD-II node generates one 128-bytes-long true random number (TRN) at boot time and once per session and stores it in secure flash together with the relative time T_n. Essentially, the relative time stands for the amount of times of communicating specific CAN frames such as the *AuthenticationID*. It is essential for the protocol to operate correctly the counter-relative time T_n on every IOSCU-enabled node to be synchronized with the relative time T_n on the CCA, otherwise the *AuthenticationID* distribution is not feasible.

The OBD-II encrypts the TRN with the pre-shared encryption key corresponding to time T_n and transmits it to the CCA node. After decryption, the CCA conceals the 64 bytes of the *AuthenticationID* by padding 32 bytes of random data before and after, inspired by steganography techniques. Then, the constructed data frame is scrambled based on the relative time T_n and the SHA256 hash of the received TRN. Finally, the CCA encrypts and sends the resulting data frame. Sixteen CAN frames, with eight bytes each, must be sent to get the secured credentials.

When the OBD-II node acquires the secured CAN frames, performs symmetrically the same operations via the authentication FSM and then stores the hash (SHA256) of the *AuthenticationID* in the Firewall's internal isolated register. An attacker does not gain anything by intercepting the captured frames (i.e., replay attacks are not feasible). After an application stores the hash of its encrypted credentials to the S*essionID* register, then the application gains the appropriate access level that this *SessionID* provides.

Hardware Firewall

In addition to memory compartments protection, the objective of the Firewall is to guarantee that each application utilizes the resources (devices) that were initially specified by the bootloader. Thus, design flaws, e.g., a memory leak in a lower-criticality memory region, or a virus that got into the system via the In-Vehicle Infotainment (IVI) environment, for example, cannot adversely affect another independent application in the system; and moreover, other applications running in protected logical partitions cannot be attacked or starved of memory or CPU time.

Based on the topology, two appealing options exist, for different reasons each. When integrating the Firewall at the NoC initiator network interface the non-conforming transactions are not allowed to enter the NoC, thus saving throughput and energy. Alternatively, when placing a Firewall after the target network interface, this offers a single point of protection for the target, thus avoiding consistency issues and synchronization overheads of updating the protection rules across different Firewalls. If the access rules for a logical partition change, then all Firewalls must be updated while safeguarding against transactions that may be on the fly. Furthermore, if different Firewall

instances are used, then communication between them may take place via a channel which is cryptographically or otherwise secure.

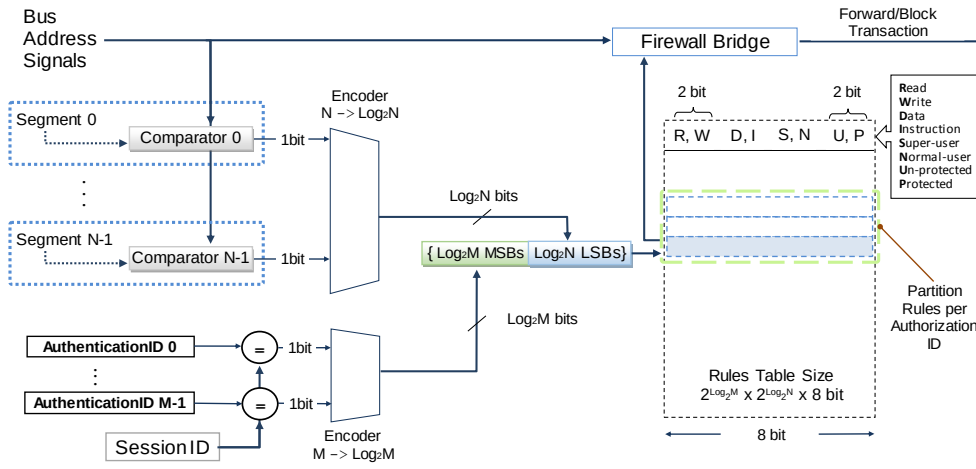

Figure 3. General Firewall organization for N memory-mapped regions; the Firewall decision can be a valid/non-valid in-band signal with the transaction, or even the full rule match result

The Firewall processes the address field of an incoming transaction as shown in Figure 3, searching whether it is subject to preconfigured rules. There are N segments describing address ranges and N parallel comparators. If the physical address matches any of the segments, then the transaction is processed by the Firewall's bridge FSM using the corresponding rule. The Firewall provides support for *separation of roles and rules per role*. Each role requires a different authentication level. Essentially, the system can support different security domains. The role that an application pertains to defines the security domain that is accessible by providing its session ID. If there is no match between the *SessionID* and any of the *AuthenticationIDs*, then, a partition with default rules is used.

The format of a rule is an eight-bit value where every bit represents an attribute and zero means deny, while one means allow. As Figure 3 shows, the attributes indicate whether the transaction issuer can read or write, if the transaction concerns data or instruction, if the issuer is super or normal user and finally if the transaction is protected or not. All the attributes except the first two are available only if the bus that the CPU uses to communicate with the peripherals can carry auxiliary signals containing information about the transaction type or transaction initiator, e.g. AXI4 AxPROT signal.

IMPLEMENTATION AND EVALUATION

We used the SEcube (Secure Environment cube) platform to evaluate the IOSCU. The SEcube provides a Cortex-M4 CPU alongside a low-power non-volatile FPGA (Lattice MachXO2-7000) in a single chip. We implemented the design shown in Figure 2, where, in the place of the memory-mapped device, we have integrated a CAN-2B compatible controller. IOSCU introduces a latency of 6 clock cycles for every read and 5 cycles for every write transaction due to the NoC; the decision making of the Firewall is constant, 5 clock cycles.

Table 2 depicts an overall evaluation of the impact of IOSCU components when realized in the MachXO2-7000 device, operating at 20MHz and consuming in total 36.3mW (6.9mW static, 29.4mW dynamic) at 1.2V.

Table 2. IOSCU breakdown of implementation cost

Hardware Components	Area Cost		Latency
	LUTs	Regis-ters	
NoC with OTP Programmable Routing	4811	3478	Reroute latency: 4 clock cycles for write path, 4 clock cycles for read path. NoC latency Slave-to-Master: 5 clock cycles for write path, 6 clock cycles for read path
Hardware Firewall (includes bridge, rules, authentication, control and glue logic)	195	107	5 clock cycles
CAN-2A/B Controller	1365	644	1 frame buffer
GPIO2AXI Bridge	130	65	2 clock cycles
Total	6501	4294	

We developed a custom bootloader under FreeRTOS-v.10 that utilizes the Memory Protection Unit (MPU) present in Cortex-M4 architecture. During boot-time the bootloader initializes the IOSCU (as described above), preventing unauthorized access to the secured peripherals (i.e., CAN controller, Firewall configuration). After system's initialization is complete, the bootloader creates the initial task in Supervisor (privileged) mode using the *xTaskCreateRestricted()* function call. This task is responsible to create any new task (i.e. user application) in User (un-privileged) mode in a way that has its own memory address space, isolated from the memory region where the supervisor or other user tasks store sensitive data (i.e., login credentials, cryptography keys). Any I/O command to the IOSCU secured space is done through the supervisor task that is responsible for updating the *SessionID* based on the user task, then forwarding the required transaction and finally clearing the *SessionID*.

The use cases to validate the IOSCU functionality consist, at first, of an application that prompts, via the UART interface, the user (vehicle maintenance engineer) application, to access the CAN network by providing the appropriate credentials. Only upon authorized login, is it possible to access the secured peripherals, while any attempt to access the peripherals using unauthorized credentials is successfully blocked. Second, after booting, we intentionally reload an application that tries to dump the memory to gain access to other users' credentials. All attempts are unsuccessful since the sensitive memory regions are accessible only by tasks with the appropriate privileges; any attempt to access the IOSCU secured peripherals without the appropriate *SessionID* is blocked by the Firewall.

If the appropriate *AuthenticationIDs* are not hardcoded to the FPGA, then, they are fetched over the CAN network using the encryption procedure described in CAN Network Identification section. If an attacker tries to inject his custom *AuthenticationID*, this ID will be rejected by the Firewall if it does not match with any ID that is stored locally at the *AuthenticationID* registers; thus, an attacker may only try to sniff the IDs over the bus and try to decrypt the data. As shown in figure 4, we were able to sniff CAN-bus traffic (i.e., *AuthenticationIDs* exchange), using a PCAN-to-USB adapter. The CCA transmits encrypted traffic that consists of the same *AuthenticationID* and is received by the OBD-II node (SEcube); we logged the payload of 16000 CAN frames, i.e., 1024000 bit. Based on the amount of the logged data, 1000 different versions of the same encrypted *AuthenticationID* can be constructed.

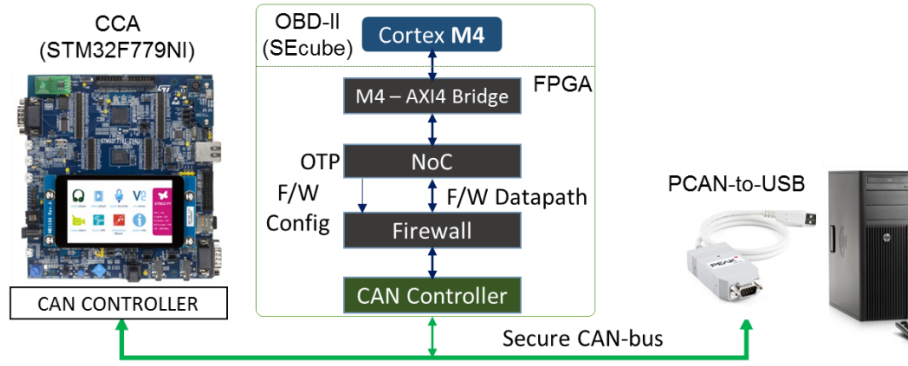

Figure 4. Secured CAN-bus network using the SEcube ECU as an OBD-II node with hardware firewalling and authentication control paired with the Communication-Certification Authority (CCA) node

To validate the randomness of the logged encrypted data we used the NIST-SP-800-22rev1a statistical test suite. We applied seven of the randomness tests to the binary expansions of e, π, √2, √3 and the logged payloads, for 10^6 bits. Table 3 presents the probability value (P-value) for all the tests and every binary expression. The selected significance level for the tests is 1%; thus, any P-Value ≥ 0.01 leads to the conclusion that the binary expression can be considered random. The randomness of the binary sequences produced by our custom *AudentificationID* (512 bit) encryption guarantees that the necessary security level has been achieved. Additionally, a steganography-like method over cryptography scheme is used (i.e., 256 bit carry valid data, out of the 512 bit transmitted for the construction of one single *AudentificationID*), which, in combination with the proof-of-randomness makes sensitive data even more secure.

Table 3. Binaries expression NIST-SP-800-22rev1a test results for the Block Lengths and LoIBS: Length of the Individual Bit Stream/s over BSA: BitStream Amount. The m-bit pattern (B) for the non-overlapping template test is equal to 000000001. The rest of the input parameters are set as default in the test suite.

Test	Block Length (bit)	LoIBS / BSA	Π	e	√2	√3	Custom encryption
Block Frequency	128	10^5/10	0.739918	0.213309	0.911413	0.534146	0.534146
Long Run	-	10^6/10	0.024390	0.718945	0.012117	0.446726	0.016831
Rank	-	10^6/10	0.083553	0.306156	0.823810	0.314498	0.819091
FFT	-	10^5/10	0.739918	0.122325	0.739918	0.350485	0.035174
Non-overlapping Template	9	10^6/10	0.165757	0.078790	0.569461	0.532235	0.488873
Approximate Entropy	10	10^5/10	0.534146	0.534146	0.122325	0.350485	0.739918
Maurer's Universal	-	10^6/10	0.669012	0.282568	0.130805	0.165981	0.993341
Linear Complexity	500	10^6/10	0.255475	0.826335	0.317127	0.346469	0.875101
Serial 1	16	10^6/10	0.143005	0.766182	0.861925	0.157500	0.394489
Serial 2	16	10^6/10	0.034354	0.462921	0.629225	0.171100	0.697556

DISCUSSION

Automotive systems are increasingly exposed to new communication channels. In modern vehicles to address increasing complexity due to the large number of ECUs and their software code, a promising solution is domain-based networking, where a domain controller isolates a number of devices and sub-networks. From a threat perspective, communication segmentation off-chip allows better visibility, management and isolation. As automotive bus technologies hardly provide an option for secure in-vehicle communication in the protocol definition and as the secure communication with external authentic devices is becoming a major concern, multiple layers of security need to be constructed harmonically together. Intruders can not only try to read or write data from the vehicle network but also internally in an ECU by using DMA-based attacks. By incorporating a hash into a CAN frame alone, despite the quality of the hashing algorithm, does not protect against playback attacks where bus activity is recorded during an event, and subsequently played back over the bus; additionally, malicious connections through open-access hubs need to be prevented by the hub software that is also vulnerable to attacks. Hence, intelligent hardware firewalling is required and connected entities (devices or applications) need to be authenticated whenever they access the vehicle. The proposed NoC Firewalling mechanism jointly with the in-vehicle CAN-bus authentication-securing protocol thus provide a multilevel solution for secure communications, while causing negligible overhead to ECUs real-time requirements.

The trend of real embedded systems is to use a mix of cores and accelerators centered on an application's needs and thus, these integrated SoCs require a shared underlying network infrastructure (possibly with requirements for non-interference).[9] However, physical isolation based on hardware mechanisms offers immense security that is hard to achieve with software IDS (e.g., anti-malware) or other integrity and authentication solutions. By increasing the footprint size and complexity of code running inside protected containers, like using TrustZone, increases also its vulnerabilities. Hence, IOSCU-based hardware components can serve to provide multi-level differentiation and multi-level authorization on multiple isolated regions in ECUs. The original manufacturer can use the highest authorization level – only this level would allow a factory reset or to activate a possible bootloader to re-program the flash memory in the device or reset the CAN controller configuration. The next authorization level down can be the "owner" or technician that needs to be able to replace a single device within the system.

In addition, methods have been proposed through secure key exchange, allowing the distribution of keys to ECUs, in terms of enabling the protection of all aspects of communicating messages in automotive networks, i.e., confidentiality, integrity and authenticity.[8] To prevent targeted network attacks from external devices, different techniques have also been introduced, such as IDs-randomization, ID-anonymization, ID-hopping,[13] or obfuscated CAN IDs[12] to prevent attacks on vehicle fleets that use the same platform. The goal is to allow only authentic ECUs to send and receive frames using IDs that only they can know.

Regarding on-chip networks some research proposals aim to use access control or data scrambling[10] to symmetric-key based cryptography design for securing the on-chip communication network.[11] A security wrapper and a key-keeper core can be included in the NoC to protect encrypted private and public keys and thus ensuring cores run trusted software. Orthogonal to these solutions we introduced smart synergies of on-chip and off-chip networking to strengthen current state-of-the-art security of ECUs, which revolves around protecting information traveling in the network, against software and even side-channel and physical attacks. IOSCU minimizes the need for enhancing the CAN network protocol with security mechanisms, which poses several challenges given the very limited data rates available since bus utilization may significantly increase.

CONCLUSIONS

Automotive ECUs, with increasingly diverse access modes must be made secure against attacks and must not permit any remote malicious stimulus. The coalition of on-chip NoC-based protection mechanisms with software and off-chip network protocols for security can offer compelling advantages against modern open-platform automotive systems that support driver-assistance applications, cloud connectivity or adjacent vehicle-infrastructure connectivity. Smart hardware primitives can essentially provide a secure execution environment to ECUs that are exposed to software or hardware adversaries. In the future, it is envisioned to use these integrated security techniques in tandem with new cryptographic methods such as Quantum Key Distribution for more robust vehicle security, to ensure non-compromised safety.

REFERENCES

1. L. Szekeres, M. Payer, L. T. Wei, and R. Sekar, "Eternal War in Memory. Security & Privacy," IEEE 12, 3, 2014, pp. 45-53.
2. J. Sepúlveda, D. Flórez and G. Gogniat, "Reconfigurable security architecture for disrupted protection zones in NoC-based MPSoCs," Proc. 10th Int'l Symp. on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2015, pp.1-8.
3. L. Fiorin, S. Lukovic, G. Palermo and P. di Milano, "Implementation of a reconfigurable data protection module for NoC-based MPSoCs," Proc. Int'l Symp. on Parallel and Distributed Processing, 2008, pp. 1-8.
4. M. D. Grammatikakis et al., "Security in MPSoCs: A NoC Firewall and an Evaluation Framework," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 34, no. 8, 2015, pp. 1344-1357.
5. F. Brasser, B. El Mahjoub, A.-R. Sadeghi, C. Wachsmann, and P. Koeberl, "TyTAN: Tiny Trust Anchor for Tiny Devices," Proc. 52nd Ann. Des. Aut. Conference (DAC'15), pp. 34:6.
6. G. Kornaros and S. Leivadaros, "Securing Dynamic Firmware Updates of Mixed-Critical Applications," Proc. 3rd IEEE Int'l Conf. on Cybernetics (CYBCONF), 2017, pp. 1-7.
7. S. Shreejith and S. A. Fahmy, "Security aware network controllers for next generation automotive embedded systems," Proc 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), 2015, pp. 1-6.
8. P. Mundhenk, et al., "Security in Automotive Networks: Lightweight Authentication and Authorization," ACM Trans. Des. Autom. Electron. Syst. 22, 2, Art. 25, 2017, pp. 25:1-25:27.
9. H.M.G. Wassel et al., "SurfNoC: A Low Latency and Provably Non-Interfering Approach to Secure Networks-on-Chip," Proc. 40th Ann. Int'l Symp. Computer Architecture (ISCA'13), 2013, pp. 583-594.
10. D. M. Ancajas, K. Chakraborty and S. Roy, "Fort-NoCs: Mitigating the Threat of a Compromised NoC", Proc. 51st Ann. Des. Aut. Conference, 2014, pp. 158:1-158:6.
11. C.H.Gebotys and R.J.Gebotys, "A Framework for Security on NoC Technologies," Proc. of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI'03), 2003, pp. 113-.
12. M. Lukasiewycz, P. Mundhenk, and S. Steinhorst, "Security-Aware Obfuscated Priority Assignment for Automotive CAN Platforms", ACM Trans. Des. Autom. Electron. Syst. 21, 2, Art. 32, 2016.
13. A. Humayed and B. Luo, "Using ID-Hopping to Defend Against Targeted DoS on CAN", Proc. 1st International Workshop on Safe Control of Connected and Autonomous Vehicles (SCAV'17).

This article has been accepted for publication in a future issue of IEEE Micro, but has not been fully edited. Content may change prior to final publication.