Power Analysis and Fault Attacks against Secure CAN: How Safe Are Your Keys?

Colin O’Flynn, Greg d’Eon
NewAE Technology Inc, Halifax, Canada.

Abstract

Designers of automotive systems find themselves pulled in an impossible number of directions. Systems must use the most advanced security features, but at the same time run on low-cost and resource-constrained hardware. Ultimately, an engineering trade-off will eventually be made regarding how encryption and key management is used on these systems, potentially leaving them vulnerable to attack.

In this paper, we detail the applicability of side-channel power analysis and fault injection on automotive electronic systems, showing how these dangerous techniques can be used to break an otherwise secure system. We build a small example network using AES-CCM to implement an encrypted, authenticated CAN protocol. We demonstrate how open-source hardware and software can easily recover the encryption keys from some of these nodes with side-channel power analysis, and we recover a full firmware image from one device with a fault-injection attack using the same tools. We also discuss how these attacks can be improved to bypass some common countermeasures and be applied against devices in the real world, bypassing security on in-vehicle communication or over-the-air firmware updates.

With these demonstrations in mind, we emphasize the importance of using strong encryption and authentication keys with proper key management and distribution methods. We discuss methods for mitigating these side-channel and fault attacks, and we use these methods to provide guidelines for creating a system architecture that is secure against these hardware attacks.

Introduction

The growing complexity of automotive systems is putting a large amount of pressure on their designers. The industry constantly pushes to add new features for comfort, entertainment, and safety, requiring a single vehicle to use an increasing number of communicating electronic devices. Embedded devices with safety-critical roles are often connected to a vehicle's CAN bus alongside customer-connected modules such as the entertainment system [1]. Attackers have managed to abuse these complex networks, allowing them to break into or remotely control vehicles [1]. Now, there is a research effort to develop encrypted in-vehicle communication to mitigate these flaws, but no standards are in wide use within production vehicles.

Additionally, some of the automotive electronics in use require firmware updates after leaving the factory. Over-the-air (OTA) updates have become possible with internet-connected systems, but these update methods can reveal more vulnerabilities. Manufacturers would like to stop customers from loading their own firmware into a car's systems to avoid issues with counterfeit parts, regulatory problems, and safety issues. Additionally, the official firmware image may include secret communication keys and algorithms that should not be revealed to attackers. To this end, most OTA firmware updates will be encrypted and signed by the manufacturer to ensure that only legitimate updates can be loaded onto the system.

This paper investigates encrypted and authenticated in-vehicle communication protocols, showing how they are vulnerable to side-channel power analysis and fault injection attacks. To demonstrate these security flaws, an example CAN bus with an AES-CCM protocol is set up between three embedded devices. Then, two possible attacks are demonstrated: a side-channel attack is used to recover encryption keys, and a fault injection attack successfully dumps a device's complete firmware. Finally, several countermeasures are discussed, and recommendations are made to help engineers prevent these attacks.

Background

This paper will focus mainly on in-vehicle attacks, where an attacker has some access to some or all nodes on a given network. Such an attacker may have a variety of objectives, but the typical cryptographic methods used in stopping the attacks are often similar.

The variety of nodes is important: an attacker may be able to break a critical node using an attack on a less critical node. For instance, if poor key provisioning is used, attacking a lighting or infotainment module could provide an attacker with secrets that provide access to the main ECU, allowing them to find additional security vulnerabilities or modify engine parameters.

Symmetric cryptography is often used in secure networks and for firmware encryption. Firmware signing would normally use asymmetric cryptography. While there are some power analysis attacks possible against asymmetric cryptography, the fact that the private signing key is not stored on the target device makes power analysis against asymmetric cryptography less valuable. Bypassing signature verification steps with fault attacks is more commonly used when breaking asymmetric cryptography

Secure CAN

The most popular communication system in automobiles today is the CAN bus, which allows many devices to send and receive messages using a single 2-wire line [2]. This protocol allows commands and data to be sent up to 1 Mbit/s, but is still simple enough for inexpensive low-power microcontrollers to use. However, the CAN standard does not contain any significant level of security. All messages sent on a CAN bus are easily readable, and there is no form of authentication to confirm the identity of a message's sender.

There are two components that would improve the security level of the CAN protocol. First, the messages could be encrypted before they are sent on the CAN bus. Encrypting the CAN messages obfuscates traffic on the CAN bus, which may complicate reverse engineering of the messages, helping prevent in-vehicle attacks [3]. Second, authentication tags could be sent with each message. These tags are calculated by performing a cryptographic signing algorithm with the message and a secret authentication key. Then, any node knowing the secret key can confirm that the message was sent by a privileged device with the key, stopping attackers from injecting their own messages.

Firmware Updates

Manufacturers have several things at stake in their vehicles' firmware. One issue is that the firmware often contains several important secrets: it may have cryptographic keys or algorithms embedded in the source code. Additionally, manufacturers want to ensure that they have control over the firmware running on their vehicles. If end users could freely reprogram an ECU, they could adjust the car's tuning parameters, disable critical safety features, or destroy important data logs.

Currently deployed automotive devices provide various levels of security for their firmware update procedure. For instance, the Unified Diagnostic Services (UDS) protocol involves an authentication step: the vehicle generates a seed, and the programmer is to perform some secret operation on the seed, calculating a key that unlocks the security-critical components [4]. This type of challenge-response system can sometimes be reverse-engineered or bypassed by abusing other programming features, such as a JTAG interface. Other vehicles are beginning to use over-the-air (OTA) firmware updates; bypassing security checks on OTA updates could allow firmware to be adjusted without physical access to the vehicle. Mitigating these security issues requires firmware signing techniques using well-known cryptographic standards.

Key Distribution

In symmetric cryptography, a shared secret key is required both for encryption and authentication, and safely distributing this key is a critical aspect of any secure system. There is a spectrum of solutions to the key distribution problem. The simplest method of key distribution is to use a single fixed key for all devices, allowing the OEM to pre-program this secret into each device. This ensures all nodes on a network can easily send messages to each other, but an outside attacker cannot. However, this method of key sharing is risky - if the key is revealed, the entire network is compromised. On the other end, the most robust method is to allow the devices to generate random secrets for each communication link and securely share these keys with other devices on the network. This approach is safer, but generating and distributing keys on a low-resource embedded device can be difficult and slow. Recently, CAN-specific key distribution has also been proposed taking into account these limitations [3].

An automotive environment provides some unique security challenges. It is typical for nodes on a CAN bus to have greatly different levels of processing power: the central ECU in a car will likely have more resources than a door control unit. An attacker can use these asymmetric security levels to their advantage. If a single key is used for the entire network, finding the key on the weakest node is enough to compromise the entire vehicle's security. This paper demonstrates how easily these keys can be leaked, showing the necessity of a proper key management and distribution system.

Example CAN

This section describes an example network using several different devices running a modified CAN protocol with encryption and authentication. Note that this protocol is not the focus of this paper, and a basic protocol was only used to have a concrete example without disparaging a specific existing secure CAN protocol. Trade-offs made in this example protocol (such as the maximum input message size, and small MAC size) will not be discussed. Instead, we wish to focus on fundamental weaknesses which are applicable among almost all secure networking proposals. This includes both secure CAN, and similar networking protocols such as CAN-FD or Ethernet. An example of the applicability of these attacks to other protocols can be seen when fault attacks are introduced, where the same type of attack which we demonstrate on a CAN network is performed on a USB device [5].

Physical Network

The network is designed to demonstrate a mix of full-feature (such as a gateway device) and low-cost (such as an end device) nodes. Three different devices are present: a gateway using a NXP MPC5748G, an end-node using a ST STM32F415, and an end-node using an Atmel AT90CAN128. The devices have a variety of hardware features: the MPC5748G is a Triple-Core PowerPC device with a Hardware Security Module (HSM) including hardware AES. The STM32F415 is a single-core ARM Cortex M4 microcontroller with hardware AES. Finally, the AT90CAN128 is a low-cost device without any hardware accelerated cryptographic protocols. These devices are all connected on a single physical CAN bus, where messages are sent back and forth between them. The devices are on commercially available development boards that are instrumented to simplify measuring of side-channel power analysis leakage, but the attacks to be discussed can easily be performed on OEM hardware. Several of the development boards are manufactured by NewAE Technology Inc., and are part of the ChipWhisperer project. The physical network is shown in Figure 1.

In addition to the expected nodes, one of the devices has a ChipWhisperer Capture CW1200 connected which is used for power analysis and fault attacks, and another node has the ChipWhisperer-Lite for power analysis attacks. The ChipWhisperer-Lite is specifically included due to its completely open-source nature (including PCB files, FPGA design, firmware, computer software, documentation) making it a useful device for academic research and teaching purposes. Under normal operating conditions, these measurement devices would not be present. Finally, a PEAK PCAN-USB is used as a computer interface for monitoring the network, and can also be used in injecting packets. While specific hardware is used for demonstration purposes here, the attacks discussed in this paper are not specific to any given measurement or control hardware - for example, the power analysis attacks in this paper can be applied with general-purpose oscilloscopes.

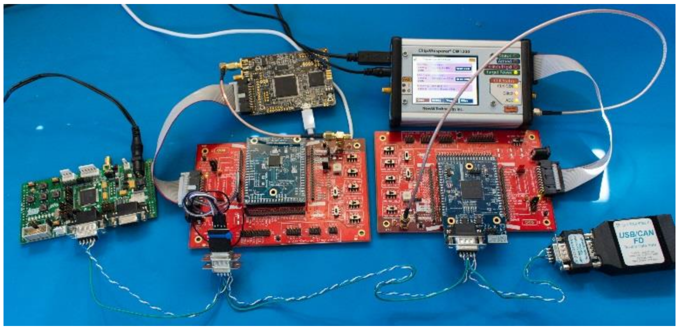


Figure 1. An example network includes a low-end node (AT90CAN128, left), medium-performance node (STM32F415, middle), 32-bit triple core PowerPC gateway device (MPC5748G, right), and a computer interface. For power analysis and glitch insertion various parts of the ChipWhisperer platform are shown attached to the nodes.

Encrypted & Authenticated CAN Protocol

The devices in this network communicate with a modified CAN protocol. This protocol uses AES-CCM to encrypt the input data and produce a MAC tag for authentication. A block diagram of the encryption and authentication process is shown in Figure 2. The use of AES-CCM in this protocol has the advantage of only requiring a single AES-ECB encryption primitive, remaining relatively fast and lightweight. Full details of the message format are given in Appendix A.

Additionally, one common problem with securing CAN is that the CAN bus is limited to a data frame size of 8 bytes. Rather than using CAN-FD or ISO 15765-2 as previous work has done, this protocol uses the ability of AES-CTR to encrypt a frame smaller than 16 bytes. It also uses the extended addressing bits of CAN 2.0B mode as a frame counter, incrementing this value for each message. These bits are used in the nonce for the AES-CTR input and the MAC tag calculation to prevent replay attacks. The receiving node must validate the nonce to confirm it is unique, and the secret keys must be changed once this frame counter ('msg #') rolls over.

Despite the high level of cryptographic security in this system, the following section will show how certain nodes can be trivially compromised given physical access to the hardware. These demonstrations will even assume that the system is otherwise secure - that there are no other flaws in the firmware or hardware.

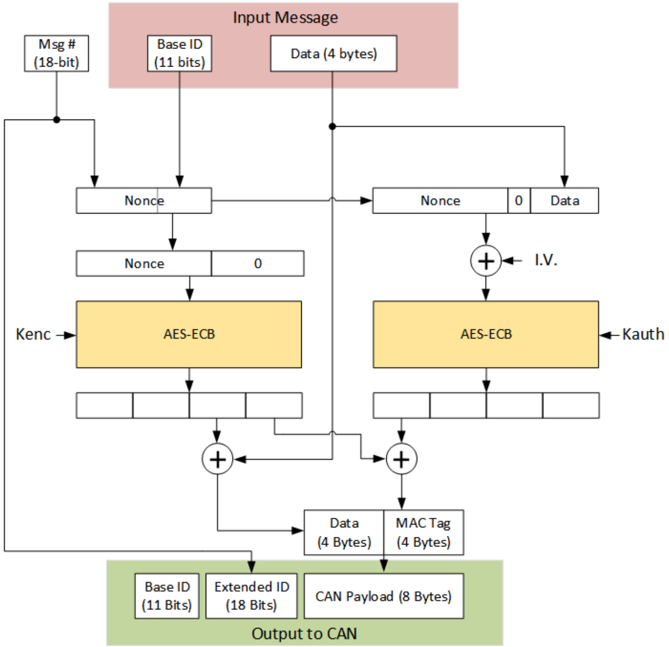


Figure 2. A modified CAN protocol using AES-CCM to provide both encryption and authentication. The input data is limited to 4 bytes long, and a 4-byte message authentication code (MAC) tag is appended. The extended ID field of the CAN message is used to transmit the nonce. This message format is described in detail in Appendix A.

Side Channel Power Analysis

Side channel power analysis is a method of breaking otherwise secure cryptographic algorithms implemented on physical devices. This field is often referred to by the name of the original technique, Differential Power Analysis (DPA), proposed by Kocher et al. [6]. This work uses a more recent technique called Correlation Power Analysis (CPA) [7]. Both DPA and CPA rely on the fact the power consumption of a digital device on specific clock cycles has some dependence on the actual data being transferred on the internal data bus. For many microcontrollers, this specifically results from the fact the internal data-bus lines are first set to a pre-charge state before every new piece of data transferred on the data bus. The amount of power required to set the data-bus from the pre-charge state to the final state will depend on the number of bits set to '1' on the data bus, typically referred to as a 'Hamming Weight' (HW) leakage model. This leakage model can be used to break software implementations of cryptographic functions, since we can learn a small amount of information about internal states of the algorithm. Based on the known input (or output) of the algorithm, we can determine the secret key information a single byte at a time, since we can determine what value the secret key must take to make the observed Hamming weight measurement valid. A comprehensive discussion of these attacks is given in [8].

Taking a specific example, this attack is possible on everything from small 8-bit microcontrollers [7] up to full computer systems [9, 10]. This attack can be performed with low-cost hardware such as the open-source ChipWhisperer [11] which can be built for approximately $200 USD (see documentation and link to source code held on GITHub at ChipWhisperer.com), and is applicable to real products. Example of products broken with it include recovering the secret key used to sign & encrypt over-the-air firmware update images for the Philips Hue lights [12], recovering the secret key in the Yubikey 2 [13], recovering encryption keys used for bitstream protection on Xilinx [14], Altera [15], and Microsemi FPGAs [16], and breaking key fobs using the Keeloq algorithm [17].

CPA Attacks on AES-128

With side-channel power analysis, it is straightforward to break an AES encryption key. In a CPA attack, each byte of the key is recovered by considering all possible values {0x00, 0x01, ..., 0xFF} of each byte. For each possible value, the correlation is calculated between an intermediate encryption state and a number of power traces recorded with known inputs (either plaintext or ciphertext). Then, at one point in time, the hypothetical intermediate value of the encryption state will have a high correlation with the traces. To provide a specific example, Figure 3A shows a power trace recorded from the STM32F415 device (as shown in Figure 1). In this plot, the AES hardware encryption is happening between sample 360 and sample 500. Then, Figure 3B shows the calculated correlation at each point in time for all 256 guesses, with the correct value of the key byte highlighted in red. This correlation peak would tell the attacker that this is the correct guess. This attack uses a Hamming distance leakage model targeting the difference between the 9th and 10th round states (as in [12]).

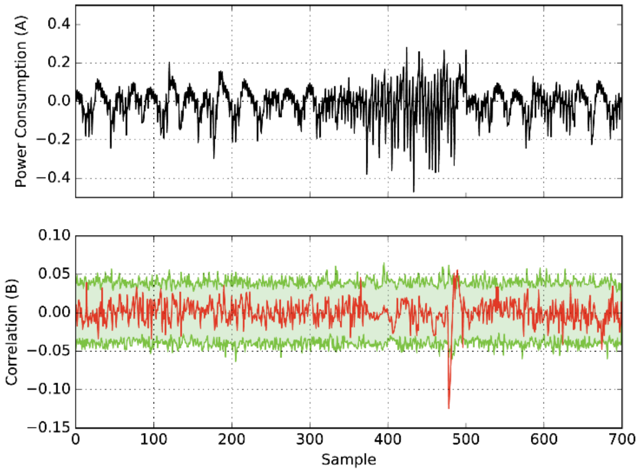


Figure 3. The top trace at (A) shows the power consumption during a hardware AES encryption on the STM32F415. The bottom trace (B) shows a correlation of every wrong key-guess (in light green) compared to the correct key-guess (in dark red) for each point in the power trace.

The nodes on our CAN bus have a variety of AES implementations - the smallest device is using a software implementation, whereas the other two are using hardware implementations. We attempted CPA attacks on the three devices to provide approximate information on leakage characteristics of the various devices. Table 1 summarizes these results, including the number of ‘traces’ (that is, encryption operations to be observed) to recover the secret key. Note the wide range – the software implementation on the AT90CAN128 takes only 40 traces, the STM32F415 takes 1900 traces, and the MPC5748G device was not able to be broken with a CPA attack using up to 200 000 traces. The 200 000 upper limit was chosen as was approximately the maximum number of traces expected to be allowed with the same encryption key (see Appendix A). This analysis was a basic CPA attack, so it is not claimed the device actually contains specific countermeasures. A more complete analysis is required to understand the security level of this device before making decisions about the number of encryptions one should allow before changing the encryption key.

Table 1. The number of traces required to recover the secret key from each device running AES-128 on each device. Note that the key was not recovered from the MPC5748G after 200 000 traces.

Device	**Leakage Model**	**Traces**
AT90CAN128	HW: Round 1 SubBytes Output	40
STM32F415	HD: Final Round State to Ciphertext	1 900
MPC5748G	Unknown (not broken)	> 200 000

Extension to Secured CAN Protocol

The proposed secured CAN bus is using AES-CCM (CTR with CBC MAC) mode. This slightly complicates the power analysis attack, as the attacker no longer has the exact input to the AES algorithm. However, it is still possible to attack either of the two encryption blocks. In Figure 2, the input to the actual AES-ECB algorithm is the nonce and counter value. The output of the AES-ECB algorithm is XORed with the plaintext in performing the AES-CTR encryption. A basic side-channel power analysis attack is recovering the key based on the input to the AES-ECB algorithm, so instead the attacker can perform a power analysis attack of later rounds of AES, as described in [18]. This technique was previously practically demonstrated in [19], and in the specific case of AES-CCM there is a shortcut due to the key reuse between the CTR and CBC modes described in [11].

To perform this attack in the real world, the attacker needs to be able to monitor a number of encryptions with varying inputs, where the approximate number of encryptions for breaking the AES key is given in Table 1. The attacker does not need to control these inputs (plaintexts and message IDs) as long as they change. Some real devices always decrypt all received messages, discarding them once realizing they are invalid after decryption. If this is the case, an attacker can simply send arbitrary messages to the target - it is irrelevant that the messages are ignored, as long as they are decrypted. Other devices might first authenticate a message before decrypting. Here, the attacker will need to first break the authentication algorithm or monitor valid messages sent to the target device from within the network.

Finally, the attacker may not always need to recover both of the secret keys: the encryption key is all that is needed to view the unencrypted content of the messages. The authentication key might only need to be broken if an attacker wants to inject their own malicious messages onto the network.

Use of H-Field Probe

While all the test boards in Figure 1 have been modified to include resistive shunts to simplify power measurement, it is well known that a non-contact electromagnetic (EM) probe can also be used. The EM probe is typically designed for sensing the magnetic field (H-Field), and suitable probes are widely available from commercial suppliers as they are used in EM emissions testing. When using a H-Field probe, the probe is simply held above the target chip as in Figure 4. The changing current consumption generates a changing magnetic field, and this field is picked up by the probe, amplified, and sent to the same measurement/capture equipment. The effectiveness of the attacks is on the same order of magnitude between the EM probe and the physical shunt measurement [20].

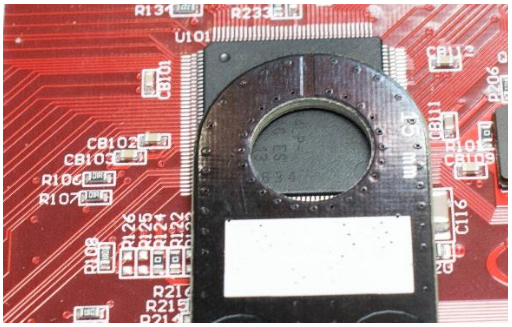


Figure 4. A magnetic-field probe can be used to measure the power consumption of a target device, and has roughly the same effectiveness as the shunt resistor without requiring any modifications to the target.

Fault Attacks

Fault (or 'glitch') attacks are used to cause a device to perform unintended operations. In a fault attack, the system is brought outside of its regular operating conditions for a short amount of time. These faults could include short pulses into the clock signal, violating setup and hold times and causing instructions to be executed incorrectly. They could also include voltage fault injection, where the internal core's voltage supply is changed momentarily. Previous work has shown that causing ringing on the internal power network of the target chip is one of the mechanisms which causes effective fault injections [21].

While both clock and voltage glitching offer surprisingly selective fault effects (being able to cause errors in a single instruction even), more granularity may be needed on advanced targets. For these targets, either electromagnetic or optical glitching would be preferred, as this glitching mechanism can target specific areas of the chip surface using a X-Y table to mechanically change the glitch location. Both of these methods have proven effective against a variety of advanced targets [22, 23], and low-cost solutions have been presented for both types of glitching [24, 25].

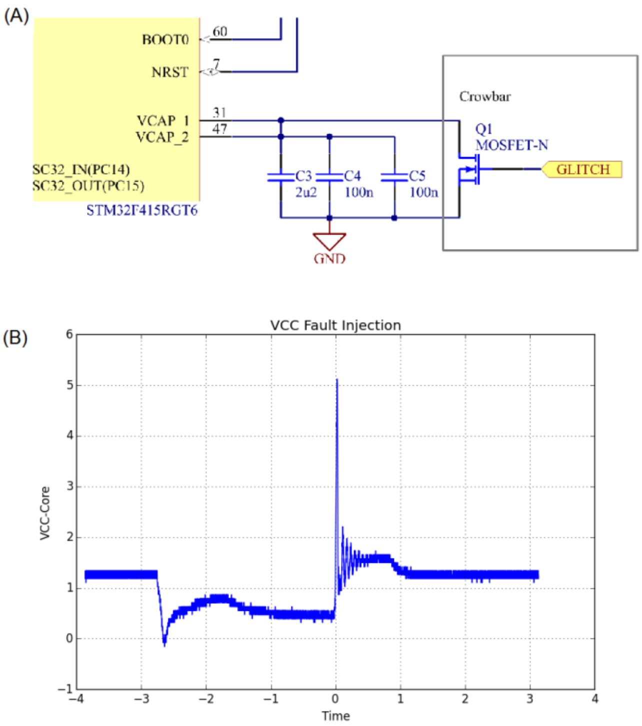
Fault attacks have the potential to be more powerful than side-channel analysis. Rather than recovering specific secret values within a device, fault attacks allow us to entirely bypass or modify certain operations. For example, to attack a device with signed firmware images, fault injection can cause the target to skip the signature check, making it load an unsigned image. This type of attack allows a malicious firmware image to be loaded onto the device, opening up additional attacks, including recovering secrets stored within the device memory.

One particularly weak point in many embedded devices is in their communication systems. One common code structure is to use a loop to transmit one byte at a time over an interface such as CAN or UART. With fault injection, the counter in this loop can be corrupted, causing the code to miss the exit condition and continue sending much more data than intended. Such an attack was demonstrated on a practical platform by Micah Scott, who successfully used a fault attack against a USB device causing it to dump the entire memory contents over the USB port[1] [5]. Of particular interest was the fact this fault glitched a higher-level communication loop - the memory was dumped in valid USB packets that respected the maximum memory size and waited for appropriate acknowledgment signals.

Example of Fault Attacks on STM32F415

We performed a fault attack against the STM32F415 target node as a demonstration of how an attacker could recover code memory from an otherwise secure device. The target of this fault is a CAN communication loop, as in the previous description. Both code and data memory are present in the same memory space on the STM32F415 (as on other ARM devices), and depending on the location of the data being printed to the communications interface may result in data and/or code memory being dumped.

We used a crowbar voltage injection attack to achieve our effective fault ***[26]***. This crowbar circuitry simply shorts the power rail of the device for a short amount of time with a MOSFET, as in Figure 5A. The resulting waveform on the internal core voltage pin of the STM32F415 is shown in Figure 5B. This glitch was timed to occur shortly after the beginning of the message, which indicates that the communication loop is running. Then, the ChipWhisperer software was used to automatically search for a precise glitch length and timing that caused the memory to be dumped. In practice, an attacker could run this automatic search over a long period of time, so it is likely that they could find a successful fault.



[1] In addition to the referenced paper, Scott has a detailed video of this attack at https://www.youtube.com/watch?v=TeCQatNcF20

Figure 5. A simple MOSFET circuit in (A) pulls down the core power voltage. The resulting waveform in (B) causes a fault in the STM32F415, resulting in a memory dump.

Fault Attack and SCA Combinations

Fault attacks have a wide variety of use-cases, and may often be combined creatively with side-channel attacks (SCA). For example, one common security measure is to use a key derivation function (KDF) with a single master key. A KDF allows a device to generate ephemeral keys without revealing the master key, giving it the ability to change keys during operation. Typically, it would be complicated to perform a side-channel attack on a KDF, as it is difficult to predict when the KDF will run. However, with fault injection, it may be possible to recover the device's code memory including the master key. Furthermore, an advanced device with secure write-only key storage may also be vulnerable to a combined attack: by loading custom firmware, an attacker could trigger a key derivation function many times, allowing a side-channel attack to succeed.

Another example of a combination attack was recently presented by Veredas et al **[23]**. There, power analysis was used to determine the location in time that a JTAG lock bit was set in a microcontroller, and an EM fault injection attack was used to prevent the correct value of this bit from being read. This left the device operational but with an unlocked JTAG port, allowing an attacker access to the internal memory of the device.

Where SCA attacks may have a specific target, such as a secret key or password, fault attacks have a more varied range of effects, and it is difficult to predict an attacker's path. For instance, fault attacks on a JTAG password may first be demonstrated with high-end equipment, but without additional experimentation it is unknown what the ability of an attacker armed with only a low-cost device would be capable of. While safety-critical design practices greatly complicate a fault injection attack, they are not a guaranteed preventative measure (as demonstrated in [23]).

Countermeasures

Software/Hardware Cryptography Implementations

Several countermeasures have been proposed to stop side-channel attacks. One common countermeasure is to add random delays before or during the cryptographic algorithm, but these delays are almost never effective. Recall from Figure 3A that the encryption operation has a visible power signature - it is visually obvious when the encryption is happening, and it is simple to realign a set of traces to remove these delays [27]. In addition, many oscilloscopes are capable of triggering based on patterns in the analog data, such as the Sum of Absolute Difference (SAD) trigger in the ChipWhisperer project [11]. More robust signal processing techniques, such as Dynamic Time-Warping [27], can also handle complex delays at any point in the algorithm. Another countermeasure is to enable additional hardware peripherals to increase the amount of noise on the power rails; this increased noise level only increases the number of traces required for the key recovery to succeed, and it is rarely enough protection to prevent a side-channel attack.

In general, software implementations of cryptographic functions tend to be routinely broken, despite many proposed countermeasures to power analysis. As a high-profile example, the DPA Contest V4.2[2] presented an AES implementation including random masking of plaintexts and shuffled operations [28]. Despite these efforts, 10 different research groups recovered the secret key, with several of the attacks requiring only a single encryption trace to recover the key. It should now be clear that software encryption cannot easily be protected against SCA.

In hardware, there are a few more methods that can be used to prevent these attacks. First, a dedicated cryptography peripheral can be set up to calculate an entire round in parallel. Note from Table 1 the STM32F415 hardware peripheral takes considerably more traces than a software implementation[3]. This alone is unlikely to be sufficiently difficult, as an attacker is likely able to cause several thousand encryption operations in a short time. To further improve results, some additional hardware features can make SCA difficult. One example is Masked Dual-rail Precharge Logic [29], which attempts to make the overall power consumption constant regardless of the input data. Through careful design, it is possible to create a device that is difficult to break with SCA. To this end, there is a growing movement for standardized security tests of encryption modules (such as ISO/IEC 17825), which would give engineers the ability to make intelligent designs preventing key leakage through side-channel attacks.

These countermeasures increase the difficulty of a SCA to succeed by requiring more traces. This can be used in combination with the system architecture to provide security – if a device can be broken with 500 000 traces (encryption operations), but the secret key is guaranteed to be changed after 10 000 uses, the SCA attack is of little practical use.

Security Through Architecture

Automotive systems require a variety of devices with different levels of complexity, and it would be prohibitively expensive to protect every single one from hardware attacks. Instead, the best form of protection is to design a key distribution architecture that protects the most critical components of the vehicle.

To ensure a high level of hardware security, an automotive system requires two main design considerations. First, the safety-critical devices must be safe against the attacks demonstrated in this paper. Care should be taken to select processors that have strong countermeasures against SCA, and these electronics should include anti-tamper features to prevent (or at least detect) these other types of invasive attacks. Second, the keys used in these secure communication links must be strong and unique. This requirement means that at minimum each vehicle should have its own OEM-generated key, and knowledge of the vehicle's VIN should not be enough to recover the key. Further segregations of the CAN bus should mean different groups also have unique keys. This way, if an end node has its firmware leaked or its keys are recovered with SCA, the critical components of the vehicle remain safe. Using asymmetric cryptography is one method of enforcing this security: each node has a separate key, so leaking an end node’s keys does not compromise the entire system.Using separate keys for each node in a CAN bus

[2] See http://www.dpacontest.org/v4/

[3] Software AES on the STM32F415 can be broken in around 40 traces, similar to the AT90CAN128 software.

may be too much work for small devices in these real-time systems. However, the previous solution suggests that the safety-critical nodes must use relatively high-power processors with strong cryptographic countermeasures. In this case, it should be possible to offload most of the key-management work to gateway devices, which are powerful enough to handle the extra work of tracking per-node keys. Recent work on methods of distributing keys to end nodes aligns with this requirement [3].

If using instead a KDF with a per-vehicle key, this key could be generated by combining a unique per-vehicle ID (such as the VIN) with some OEM secret known to the OEM, allowing the manufacturer to quickly recalculate each vehicle's key. Also, this OEM secret should vary over time to ensure that a leak of an old OEM secret does not compromise newer models.

Asymmetric cryptography is of great value in avoiding issues with shared keys. Typically asymmetric algorithms are too slow for use in real-time encryption/decryption on communication links, but are helpful for both key distribution, and verification or transfer of critical data blocks (e.g., firmware signing). Where asymmetric cryptography can be used in such a way the private key is not stored on the vulnerable device (i.e., when performing signature verification) removes sensitive data that can be leaked via side-channel or other means.

For embedded devices, ECC with Curve25519 has become a popular choice due to patent-free status combined with excellent performance possible even on embedded devices [30]. An excellent example of side-channel attack on Curve25519 implementations is available in [31], which also includes references to other work of interest.

Conclusions

This paper has presented a cryptographically secure CAN protocol and shown how it can be trivially broken using side-channel power analysis and fault injection. Open-source tools were used to recover secret encryption keys and full firmware images on a variety of hardware platforms, and the full extent of these attacks in practice was discussed.

With these demonstrations in mind, future work on secure CAN standards should focus on designing systems that are resilient against the effects of SCA and fault injection to ensure a high level of vehicle safety. To achieve this safety, automotive system designers must be aware of the challenges present in embedded security. These systems must be designed to ensure that safety-critical devices are protected against hardware attacks, and careful key management is central to ensuring that these protected components are not left open to other vulnerabilities.

References

[1] C. Miller and C. Valasek, "Remote Exploitation of an Unaltered Passenger Vehicle," IOActive White Paper, 2015.

[2] International Organization for Standardization, *Road vehicles -- Controller area network (ISO 11898),* 2015.

[3] S. Jain and J. Guajardo, "Physical Layer Group Key Agreement for Automotive Controller Area Networks," *Workshop on Cryptographic Hardware and Embedded Systems (CHES),* pp. 85-105, 2016.

[4] International Organization for Standardization, *Road vehicles -- Unified diagnostic services (ISO 14229),* 2013.

[5] M. Scott, "The FaceWhisperer for USB Glitching," *PoC || GTFO,* vol. 13, pp. 30-37, 2016.

[6] P. Kocher, J. Jaffe and B. Jun, "Differential Power Analysis," *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology,* pp. 388-397, 1999.

[7] E. Brier, C. Clavier and F. Olivier, "Correlation Power Analysis with a Leakage Model," *Proceedings of Workshop on Cryptographic Hardware and Embedded Systems,* pp. 16-29, 2004.

[8] E. Oswald, S. Mangard and T. Popp, Power Analysis Attacks: Revealing the Secrets of Smart Cards, Springer US, 2007.

[9] D. Genkin, I. Pipman and E. Tromer, "Get Your Hands Off My Laptop: Physical Side-Channel Key-Extraction Attacks on PCs," *International Workshop on Cryptographic Hardware and Embedded Systems (CHES),* pp. 242-260, 2014.

[10] J. Balasch, B. Gierlichs, O. Reparaz and I. Verbauwhede, "DPA, Bitslicing and Masking at 1 GHz," *Workshop on Cryptographic Hardware and Embedded Systems (CHES),* pp. 599-619, 2015.

[11] C. O'Flynn and C. Zhizhang, "ChipWhisperer: An Open-Source Platform for Hardware Embedded Security Research," in *COSADE,* Paris, France, 2014.

[12] E. Ronen, C. O'Flynn, A. Shamir and A.-O. Weingarten, "IoT Goes Nuclear: Creating a ZigBee Chain Reaction," *IEEE Symposium on Security and Privacy (SP),* pp. 195-212, 2017.

[13] D. Oswald, B. Richter and C. Paar, "Side-Channel Attacks on the Yubikey 2 One-Time Password Generator," *In International Workshop on Recent Advances in Intrusion Detection (RAID),* pp. 204-222, 2013.

[14] A. Moradi and T. Schneider, "Improved Side-Channel Analysis Attacks on Xilinx Bitstream Encryption of 5, 6, and 7 Series," *In Workshop on Constructive Side-Channel Analysis and Secure Design (COSADE),* pp. 71-87, 2016.

[15] A. Moradi, D. Oswald, C. Paar and P. Swierczynski, "Side-channel attacks on the bitstream encryption mechanism of Altera Stratix II: facilitating black-box analysis using software reverse-engineering," *In Proceedings of the ACM/SIGDA*

international symposium on Field programmable gate arrays, pp. 91-100, 2013.

[16] S. Skorobogatov and C. Woods, "In the blink of an eye: There goes your AES key," 2012.

[17] C. Paar, T. Eisenbarth, M. Kasper, T. Kasper and A. Moradi, "KeeLoq and Side-Channel Analysis-Evolution of an Attack," in *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, 2009.

[18] J. Jaffe, "A First-Order DPA Attack Against AES in Counter Mode with Unknown Initial Counter," *Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, pp. 1-13, 2007.

[19] National Institute of Advanced Industrial Science and Technology (AIST), "Power Analysis Attacks on SASEBO," 2010.

[20] D. Agrawal, J. Rao and P. Rohatgi, "Multi-channel Attacks," in *Workshop on Cryptographic Hardware and Embedded Systems*, 2003.

[21] L. Zussa, J.-M. Dutertre, J. Clediere and B. Robisson, "Analysis of the fault injection mechanism related to negative and positive power supply glitches using an on-chip voltmeter," *Symposium on Hardware-Oriented Security and Trust (HOST)*, 2014.

[22] R. Carpi, S. Picek, L. Batina, F. Menarini, D. Jakobovic and M. Golub, "Glitch It If You Can: Parameter Search Strategies for Successful Fault Injection," in *Smart Card Research and Advanced Applications (CARDIS)*, 2014.

[23] R. P. Veredas and N. Wiersma, "Safety != Security. A security assessment of state of the art ASIL-D certified microcontrollers," in *ESCAR USA*, 2017.

[24] O. Guillen, M. Gruber and F. De Santis, "Low-cost Setup for Localized Semi-invasive Optical Fault Injection Attacks - How Low Can We Go?," *Workshop on Constructive Side-Channel Analysis and Secure Design (COSADE)*, 2017.

[25] A. Cui and R. Housley, "BADFET: Defeating Modern Secure Boot Using Second-Order Pulsed Electromagnetic Fault Injection," in *USENIX Workshop on Offensive Technology (WOOT)*, 2017.

[26] C. O'Flynn, "Fault Injection using Crowbars on Embedded Systems," IACR E-Print, 2016.

[27] J. J. van Woudenberg, M. Witteman and B. Bakker, "Improving Differential Power Analysis by Elastic Alignment," in *Topics in Cryptology – CT-RSA*, 2011.

[28] S. Bhasin, N. Bruneau, J.-L. Danger, S. Guilley and Z. Najm, "Analysis and Improvements of the DPA Contest v4 Implementation," in *Conference on Security, Privacy, and Applied Cryptography Engineering (SPACE)*, 2014.

[29] T. Popp and S. Mangard, "Masked Dual-Rail Pre-charge Logic: DPA-Resistance Without Routing Constraints," in *Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, 2005.

[30] D. J. Bernstein, "Curve25519: New Diffie-Hellman Speed Records," *PKC*, pp. 207-228, 2006.

[31] D. Genkin, L. Valenta and Y. Yarom, "May the Fourth Be With You: A Microarchitectural Side Channel Attack on Several Real-World Applications of Curve25519," in *ACM Conference on Computer and Communications Security*, 2017.

[32] C. O'Flynn and C. Zhizhang, "Power analysis attacks against IEEE 802.15.4 nodes," in *Workshop on Constructive Side-Channel Analysis and Secure Design (COSADE)*, 2016.

[33] A. Greenberg, "Hackers remotely kill a jeep on the highway—With me in it," *Wired*, no. 7, p. 21, 2015.

Contact Information

Colin O’Flynn, PhD
C.T.O.
NewAE Technology Inc.
1083 Queen St., Suite 196. Halifax, NS., Canada.
coflynn@newae.com

Definitions/Abbreviations

AES	Advanced encryption standard
CAN	Controller area network
CBC	Cipher block chaining
CCM	CTR Mode with CBC MAC
CPA	Correlation power analysis
CTR	Counter
DPA	Differential power analysis
ECB	Electronic code book
EM	Electromagnetic

H-Field	Magnetic field
HD	Hamming distance
HW	Hamming weight
KDF	Key derivation function
MAC	Message authentication code
OEM	Original Equipment Manufacturer
OTA	Over-the-air
SCA	Side channel analysis

Appendix A: Details of Secure CAN Protocol

This appendix provides technical details of the secure CAN protocol, which is described in Figure 2. Providing a secure communications link will require additional data to be transferred beyond just the original payload. Since the standard CAN data payload is limited to 8 bytes, adding security to an 8-byte data payload would require splitting the message. To avoid this issue, the protocol here has a maximum user data payload size of 4 bytes. The remaining 4 bytes available in the 8-byte CAN payload are reserved for a Message Authentication Code (MAC), which allows the receiver to confirm a given message was sent from the claimed node and has not been modified.

The encryption/authentication method is simply a slight modification of the standard AES-CCM (see RFC 3610) encryption mode, which uses AES-CTR mode for encryption and AES-CBC for authentication. The modification here is that (1) different keys are supported for encryption and authentication, and (2) only a single block is ever operated on due to the short message size, so the counter inputs to the AES-CTR mode are fully fixed.

Our secure CAN block can be seen as taking as an input a given message which has a standard 11-bit ID (with no extended ID), and four payload data bytes. This example protocol may be limited in real-life use cases, but is designed primarily to demonstrate aspects of a secure protocol without adding the complexity involved in using CAN-FD or ISO 15765-2 to support longer messages. The secure CAN protocol transmits the message on the CAN network in such a manner that the message is encrypted, and someone spoofing the CAN network could not replay messages (i.e., they could not record encrypted messages of a door unlock command and simply replay them as-is, without breaking the encryption).

Providing replay protection requires some “message counter”. We hijack the “extended ID” bits for this purpose, giving us an 18-bit counter. A transmitting node simply increments this on every message sent, and the receiving node will verify for every message that the counter has been incremented from the last valid received message. The receiver MUST only update their internal state of what constitutes a valid counter-ID after verifying the message came from an authentic source (i.e., MAC verification passes OK), as otherwise an attacker can perform a simple denial of service attack on the CAN bus by sending messages with the message counter set to the maximum value. Under normal operations the keys must change before that counter overflows, as once the counter overflows the current keys can no longer be used for communication. With an 18-bit counter this means 2^{18} (262 144) messages can be exchanged with a given key pair.

In addition to the message counter, a message authentication code (MAC) is used such that the receiver can verify the message authenticity. The MAC code is 4 bytes appended to the data payload. The MAC is dependent on the contents of the message counter, message ID, data payload, and a secret “authentication key” known only to authorized nodes on the network.

Tables A1 and A2 show the generation of the encrypted payload and MAC tag. In this example the input message is DE AD BE EF (4 bytes), which is converted to 28 BF 24 96 95 A2 89 87 . The first four bytes are the encrypted payload, the next four bytes are the MAC tag. This message must be transmitted with a standard message ID of 0x2D0 (the original message ID), and with the extended ID bits set to 0x00456 (the message counter). In Table A2 a non-zero I.V. is used as a calculation example to validate byte ordering, but if following NIST SP800-38C this should instead all 0’s if a fixed value is used.

Table A1. This table shows the encryption of the payload portion of the CAN message (all values in hex).

Original Message ID	2D0
Original Message Payload	DE AD BE EF
Message Counter	00456
Kenc (use with AES-CTR)	2B 7E 15 16 28 AE D2 A6 AB F7 15 88 09 CF 4F 3C
AES-CTR Nonce (AES-ECB Input)	00 04 56 02 D0 00 00 00 00 00 00 00 00 00 00 00
AES-ECB Output (for AES-CTR)	5D 30 A9 47 12 24 9F 84 F6 12 9A 79 5C 57 CE 02
XOR of AES-ECB output byte 8-11 with payload (AES-CTR encryption of Payload)	(F6 12 9A 79) $\oplus$ (DE AD BE EF) = (28 BF 24 96)

Table A2. This table shows the generation and encryption of the MAC tag. Reference table A1 for payload information (all values in hex).

Kauth	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
AES-CBC Input	00 04 56 02 D0 00 00 00 00 00 00 00 DE AD BE EF
AES-CBC I.V. (Example to validate byte order)	00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
AES-CBC Output	C9 F5 47 85 2F EE 25 43 8F 5C 8A B7 68 B0 8D BA
MAC Tag	C9 F5 47 85
XOR of AES-ECB output byte 12-15 with MAC (AES-CTR encryption of MAC Tag)	(5C 57 CE 02) $\oplus$ (C9 F5 47 85) = (95 A2 89 87)

Power Analysis Considerations

Power analysis against plain AES-ECB is used in the paper as a reference for the number of traces required. Recovering all 16 bytes of the AES-128 encryption key requires us to perform encryption (or decryption) operations, where we know or control all 16 bytes of the input data. This data must be non-constant in order for us to perform a standard CPA attack. When other modes are used (such as AES-CTR mode used here), most of the input data is fixed (i.e., the nonce values are the input to the AES-ECB block). We can still apply CPA attacks against other modes (such as the AES-CCM mode used here) with some modifications. The direct application to AES-CTR mode was first described in [18]. Due to the diffusion property of AES, the small changes of input text will result in all bytes being recoverable at a later round of the AES algorithm. This does not even require us to know the value of all constant bytes.

The paper in [18] was based on bytes 14-15 changing (as would happen in AES-CTR when multiple blocks are encrypted). Our protocol instead has an ability to change the frame counter bytes, which are mapped into bytes 0-2 of the AES-ECB input, so would require a slight modification to how the attack is handled. Such modifications were used in [32] and [12].

In [12] a specific attack against AES-CCM was used, which works assuming the AES-CCM authentication and encryption key are the same. A power analysis attack against AES-CBC mode may be easier than AES-CTR mode, since the input data will be fed more directly into the AES-ECB primitive for AES-CBC. This would allow an attacker to perform power analysis on only the authentication operation, and recover the encryption key also used for encryption. The secure CAN protocol used in this paper has separate authentication & encryption keys to avoid this attack, requiring an attacker to perform power analysis against both the encryption and authentication operations.

Note the selection of I.V. makes little difference for the power analysis attack against AES-CBC. If a known I.V. is used this can be accounted for by the attacker. If an unknown but fixed I.V. is used, we can consider the I.V. as being XOR'd with the Kauth bytes instead of the input data, since the first step in the AES algorithm will be to XOR each of the AES-ECB input bytes with the key bytes. That is the algorithm as written is [(Input $\oplus$ I.V.) $\oplus$ Kauth], but we instead consider the secret key the combination of (Kauth $\oplus$ I.V.). An attacker can use this recovered combination without needing to separate the I.V. and Kauth portions, since when using the recovered combination they will be calculating [Input $\oplus$ (Kauth $\oplus$ I.V.)], which will provide the same output as if the Kauth and I.V. portions were known separately.

The remainder of the AES-CBC attack will need to proceed similarly to the AES-CTR attack, since some of the AES-CBC bytes are fixed and cannot be changed.