IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2020, No. 2, pp. 99–127. DOI:10.13154/tches.v2020.i2.99-127

Dismantling DST80-based Immobiliser Systems

Lennert Wouters[1], Jan Van den Herrewegen[2], Flavio D. Garcia[2], David Oswald[2], Benedikt Gierlichs[1] and Bart Preneel[1]

[1] imec-COSIC, KU Leuven Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
firstname.lastname@esat.kuleuven.be

[2] School of Computer Science, University of Birmingham, UK,
{jxv572,f.garcia,d.f.oswald}@cs.bham.ac.uk

Abstract. Car manufacturers deploy vehicle immobiliser systems in order to prevent car theft. However, in many cases the underlying cryptographic primitives used to authenticate a transponder are proprietary in nature and thus not open to public scrutiny. In this paper we publish the proprietary Texas Instruments DST80 cipher used in immobilisers of several manufacturers. Additionally, we expose serious flaws in immobiliser systems of major car manufacturers such as Toyota, Kia, Hyundai and Tesla. Specifically, by voltage glitching the firmware protection mechanisms of the microcontroller, we extracted the firmware from several immobiliser ECUs and reverse engineered the key diversification schemes employed within. We discovered that Kia and Hyundai immobiliser keys have only three bytes of entropy and that Toyota only relies on publicly readable information such as the transponder serial number and three constants to generate cryptographic keys. Furthermore, we present several practical attacks which can lead to recovering the full 80-bit cryptographic key in a matter of seconds or permanently disabling the transponder. Finally, even without key management or configuration issues, we demonstrate how an attacker can recover the cryptographic key using a profiled side-channel attack. We target the key loading procedure and investigate the practical applicability in the context of portability. Our work once again highlights the issues automotive vendors face in implementing cryptography securely.

Keywords: Vehicle immobilisers, Digital Signature Transponder, DST80, key diversification, side-channel attacks

1 Introduction

A vehicle immobiliser system is an imperative anti-theft device in any modern car. By requiring cryptographic authentication from a Radio Frequency IDentification (RFID) transponder embedded within the key fob, the immobiliser prevents an attacker from hot-wiring a car. The car typically authenticates the low-frequency transponder through a challenge-response protocol based on symmetric key cryptographic primitives. However, the use of proprietary cryptography in several widespread immobiliser systems has rendered these practically ineffective [BGS+05, VGB12, VGE15, WMA+19, HGO18]. In order to safeguard the security of the immobiliser, it is crucial that the community can publicly scrutinise the underlying cryptographic primitives.

Texas Instruments (TI) produced their first cryptographically enabled transponder in 1995, named Digital Signal Transponder 40 (DST40) [Kai08]. Bono et al. reverse engineered the DST40 cipher using a black-box approach in 2005 and showed that an attacker could exhaustively search the key space using an FPGA cluster [BGS+05]. As a consequence, TI released DST80 as a successor to the insecure DST40 in 2008 in order to prevent exhaustive search attacks. DST80 transponders use an 80-bit cryptographic

Received: 2019-10-15 Accepted: 2019-12-15 Published: 2020-03-02

key to encrypt a 40-bit challenge generated by the reader. To this date, DST80 remains confidential and the immobiliser systems relying on it (shown in Table 1) have not been publicly scrutinised.

Table 1: Non-exhaustive list of vehicles affected by the research in this paper. The indicated production period is based on the information available in [traa, trab]. The models in bold point out the specific vehicles we inspected.

Make	Period	Model
Toyota	2009-2013	**Auris (2011)**
	2010-2013	Camry
	2010-2014	Corolla
	2011-2016	FJ Cruiser
	2009-2015	Fortuner
	2010+	Hiace
	2008-2013	Highlander
	2009-2015	**Hilux (2014)**
	2009-2015	Land Cruiser
	2011-2012	RAV4
	2010-2014	Urban Cruiser
	2011-2013	Yaris
Tesla	06/2018-07/2019[1]	**Model S (2018)**

Make	Period	Model
Kia	2012+	**Ceed (2016)**
	2014	**Carens (2014)**
	2011-2017	Rio
	2013+	Soul
	2013-2015	Optima
	2011+	Picanto
Hyundai	2008+	I10
	2009+	I20
	2009+	I20
	2010+	Veloster
	2016	**IX20 (2016)**
	2013	**I40 (2013)**

The immobiliser is a mandatory anti-theft device in any motorised vehicle sold in the European Union since 1995 [Com95]. Its primary purpose is to prevent unauthorised use of the vehicle, which it achieves by requiring an RFID transponder to authenticate before starting the vehicle. Figure 1 depicts the general layout of such an immobiliser system, comprising an ignition coil, the immobiliser ECU, the Engine Control Unit and a transponder embedded in a key fob. The immobiliser identifies the transponder, which it powers through the ignition coil, through a Transponder Base-Station IC such as the TMS3705 [Tex12b]. Subsequently, it challenges the transponder chip, which authenticates by generating a cryptographic response. Only when this response is correct, the immobiliser authenticates to the Engine Control Unit in similar fashion to [Bok17] and enables it to start the engine. The security of the link between the immobiliser and engine ECU is beyond the scope of this paper.

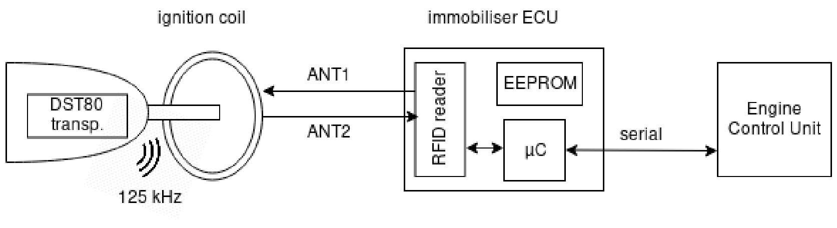

Figure 1: A typical immobiliser system

1.1 Contributions

The contributions of this paper can be summarised as follows:

- **Recovering and reverse engineering immobiliser firmware.** We present a novel fault attack to recover the firmware of the Renesas 78K0 microcontroller, reducing the complexity of the so far most efficient known attack introduced in [BFP19] from 15,000 to two successful power glitches. We reverse engineered the proprietary

[1]Tesla resolved the issue using an OTA update allowing affected customers to self-service their key fob.

DST80 cipher from this immobiliser firmware, and we publicly document this cipher for the first time in full detail. Additionally, we recovered the key diversification schemes from three major manufacturers.

- **Security analysis of key diversification schemes.** We analyse the security of the key diversification schemes used in Toyota, Kia and Hyundai immobiliser systems. We reduce the complexity for recovering the cryptographic key from 2^{80} to 2^{24} off-line encryptions for Kia and Hyundai. For Toyota immobiliser systems, we show how the key is based on publicly readable information such as the transponder serial number, therefore losing all of its entropy.

- **Practical attacks on DST80 transponders.** We show how a downgrade attack can reduce the key space from 2^{80} to 2^{41}. Depending on the exact configuration, this can lead to recovery of the DST80 key using two lookup tables and only four challenge response pairs. This attack affects the second version of the Tesla Model S key fob. Moreover, we describe Denial-Of-Service attacks, which can render the key fob unusable.

- **A portable profiled side-channel attack on the key loading procedure.** We propose a profiled side-channel attack to recover the cryptographic key from a properly configured transponder. Specifically, we target the key loading procedure performed before each invocation of the cipher. Attacking static data loads using a profiled attack is considered one of the most difficult side-channel targets [ZOB18]. In this paper, we document how this can be done in practice and take portability issues into account. Previous work proposes correcting for DC offset and using normalization techniques to overcome portability issues. These techniques do not solve the issues encountered in this scenario, instead we create a model for the family of devices and attack a previously unseen device to evaluate the model. Using this technique, we can reduce the key space from 2^{80} to approximately 2^{40} using only 10 traces.

1.2 Related work

Immobiliser and Keyless Entry systems

Because of their security critical function, immobilisers have been a hot topic of research ever since they became an integral part of the car's internal network. As early as 2005, Bono et al. published and cracked the DST40 cipher used in immobiliser systems of many vehicles by exhaustively searching the keyspace with an FPGA cluster [BGS+05]. More recently, Wouters et al. uncovered the use of the same DST40 cipher in immobiliser systems of several high-end contemporary supercars, including certain Tesla and McLaren models [WMA+19]. In contrast, in this work we analyse the security of multiple real world immobiliser systems using the DST80 cipher and demonstrate multiple practical attacks. In 2012, Verdult et al. found several flaws in the Hitag2 cipher, allowing an attacker to bypass the electronic protection within minutes [VGB12]. One year later, researchers proposed several attacks on the Megamos Crypto transponder, another cipher widely used in car immobilisers [VGE15]. Finally, Hicks et al. disclosed several vulnerabilities in AUT64, a 64-bit feistel network block cipher used in immobiliser systems of Mazda, Ford and Proton cars [HGO18].

The immobiliser, Passive Keyless Entry (PKE), and Remote Keyless Entry (RKE) systems are often intertwined in modern vehicles. When combined, they are referred to as a Passive Keyless Entry and Start (PKES) system, which allows the vehicle to unlock and start automatically when the legitimate user is near-by the vehicle. Francillon et al. showed that several models with PKES systems are vulnerable to relay attacks in [FDC11].

In [GOKP16], Garcia et al. presented a novel correlation-based attack on the Hitag2 cipher in order to bypass the RKE system in vehicles of several manufacturers. They also revealed the lack of key diversification in the VW Group RKE systems. Furthermore, it has been demonstrated that the Keeloq cipher, used in both immobiliser and RKE, is vulnerable to a plethora of physical and cryptanalytical attacks [KKMP09, Bog07, IKD+08, CBW08].

Side-Channel Analysis

With respect to the physical security of transponder chips, Side-Channel Analysis (SCA) is a well established field of research in which side-channel emanations of a target device are captured and analysed to extract secret information. In contrast to an exhaustive search or a cryptanalytic attack the goal of SCA is to recover (part of) the cryptographic key by observing one or more side-channels such as power consumption, electromagnetic radiation, time, temperature and photonic emissions. The attacks discussed in this paper use power side-channel measurements to recover the cryptographic key from DST80 transponders.

Side-channel attacks are generally divided into non-profiled and profiled attacks. Non-profiled attacks such as Differential Power Analysis (DPA) do not assume any prior knowledge about the device being attacked [KJJ99]. In contrast, a profiled side-channel attack consists of two phases commonly referred to as the profiling and the attack phase. During the profiling phase an adversary is assumed to have access to one or multiple unlocked devices which they can use to acquire side-channel measurements. These profiling measurements are used to create a model for the device. During the attack phase the attacker will acquire measurements from a Device Under Test (DUT) for which he cannot modify the key. The model created during the profiling phase is now used to recover the key from the DUT. Multivariate Gaussian templates and stochastic models are the most common types of profiled attacks [CRR02, SLP05].

Recently machine learning techniques such as Multi-Layer Perceptron (MLP) models and Convolutional Neural Networks (CNNs) have been shown to perform well in the profiled attack scenario, in some cases outperforming classical profiled attacks. For example, the translation invariability of CNNs is an advantage when measurements suffer from misalignment [PSB+18, CDP17]. Carbone et al. showed that machine learning can be an effective tool when evaluating cryptographic implementations with countermeasures [CCC+19].

The practical applicability of profiled attacks is often overestimated by using the same device, and measurements from the same acquisition campaign during both the profiling and attack phases. In this setting the inter-device variability and environmental factors such as changes in the measurement setup or ambient temperature are neglected. Montminy et al. demonstrated that using the zero mean, unit variance normalization technique on both the training and attack dataset significantly improved the effectiveness of their template attack on the AES `AddRoundKey` operation [MBTL13]. Choudary and Kuhn have shown that template attacks can be made portable in certain situations by using dimensionality reduction techniques and manipulating the DC content of the eigenvectors [CK18]. In an artificial scenario they were able to recover the exact value of an 8-bit data load across multiple devices with a success rate of 85% using 1000 attack traces. In their experiments the main difference between devices and acquisition campaigns were due to DC offset, their results show that compensating for this DC offset and using multiple devices in the profiling step greatly improves the template efficiency [CK14].

Recent work by Golder et al. has shown that using Principal Component Analysis in combination with Dynamic Time Warping (DTW) and a Multi-Layer Perceptron (MLP) model trained on traces from multiple devices can achieve an accuracy up to 99.43% on unseen devices [GDD+19]. Similarly to the work by Choudary et al. they targeted the Atmel 8-bit XMEGA family of microcontrollers but instead of focusing on a single byte load their target was the AES block cipher. In contrast, in this work, we target multiple subsequent byte loads in a real-world scenario in which we do not know the implementation and have no control over it.

1.3 Notation

Unless stated otherwise, we will use the following notation. S_i denotes bit i of a variable S, with S_0 being the least significant (rightmost) bit. Likewise, $S[i]$ stands for byte i of S, with $S[0]$ the least significant (rightmost) byte. We depict a rotation of S by i bits to the left by $S \lll i$. $S_{i...j}$ characterises a range from bit i to bit j of variable S. Similarly, a shift of S by i bits to the left is denoted by $S \ll i$. Finally, $(b \,|\, c)$ indicates a concatenation of bytes b and c, with c the least significant byte.

2 DST80

In this Section, we present the full details of DST80. Unlike the black-box approach taken to recover DST40 in [BGS+05], we reverse engineered the DST80 cipher from immobiliser firmware. Additionally, we give an an overview of the DST80 transponder chips, and describe how to recover this immobiliser firmware.

2.1 DST80 transponders

The Texas Instruments DST transponders are available in different package types; the classical wedge type transponder (TMS37145 [Tex12a]), a TSSOP package (TMS37126 [Tex12c]) and a TSSOP package containing both the TMS37126 and an MSP430 microcontroller (TMS37F128 [Tex12d]). All of these incorporate an 80-bit authentication key, which is internally stored in two 40-bit chunks, key_L and key_R. The transponders can be configured either for fast authentication (the reader is not authenticated) or mutual authentication. DST80 transponders are uniquely identified by a 32-bit serial number stored in EEPROM consisting of a 3-byte unique identifier and a 1-byte manufacturer code. According to the datasheet, pages 1, 2 and 3 are always publicly readable. Additionally, the transponder incorporates several pages of EEPROM user memory, all of which can be write-locked. Table 2 gives an overview of several significant EEPROM pages of a DST80 transponder [Tex12a].

Table 2: DST80 transponder memory layout

Page	Size (bytes)	Description
1	1	Selective addressing[2]
2	1	User data[3]
3	4	Transponder ID
4	10	Encryption key
30	5	Configuration page

Authentication protocol. DST80 transponders follow the same challenge-response protocol as DST40, described in [BGS+05]. Specifically, the reader generates a 5-byte challenge and transmits it to the transponder. The transponder in turn calculates and sends the 5-byte response based on the shared key, consisting of a 16-bit Block Check Character (BCC) and a 24-bit signature, as can be seen in Figure 2.

[2]Page 1 is also referred to as the *password* page. For example, in transponders used with Toyota immobilisers, it indicates whether the key is a master key or valet key.

[3]According to [Tex12a], this page contains the key number in the application. In Toyota transponders, it is called the *identity* page.

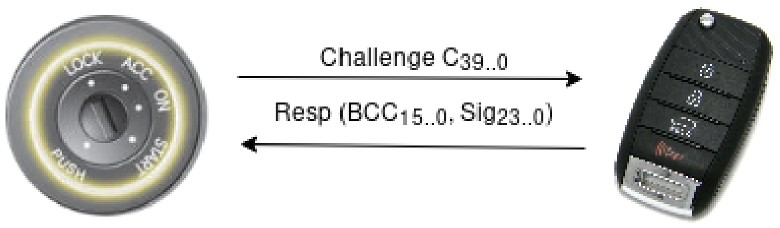

Figure 2: The DST80 authentication protocol

2.2 Reading out immobiliser firmware

We found the same immobiliser ECU in both Kia and Hyundai cars. An 8-bit microcontroller (the STM8AF6266 [STM]) controls the ignition coil and enables or disables the Engine Control Unit over a serial connection depending on whether the authentication was successful. This particular microcontroller is part of the STM8AF series and has 8kB of flash and an internal EEPROM of 384 bytes. It also features memory read-out protection, however, in this case, the protection was not enabled. Thus, we could recover the firmware and internal EEPROM by connecting to the SWIM interface and issuing a Read-On-The-Fly (ROTF) command, which allows to read any byte in the 24-bit address space. If the read-out protection was enabled, an attacker could potentially recover the firmware by adapting existing physical attacks as described in [BFP19, OT17].

On the other hand, in Toyota cars, a dedicated ECU produced by Tokai Rika handles the immobilisation process. It contains an external ST96320 EEPROM chip connected to a main MCU with only a Tokai Rika part number on it, making it hard to identify the chip. In order to read out the firmware of this MCU, we had to determine its model and debug interfaces. By looking at the locations of standard pins, such as V_{cc} and V_{ss}, we narrowed down the search to the Renesas 78K0 series [Renb]. By taking the location of the oscillator pins (`X1` and `X2`) into account, the 78K0/Kx2 resulted as the most likely candidate. The 78K0 microcontrollers contain an on-chip bootloader, which is accessible through the *Renesas Flash Programming Interface* [Rena]. By setting the FLMD0 pin high on reset, an external programmer can access this serial interface to write, erase, and verify the internal flash. However, the 78K0 series does not provide a command to read the memory. We used a Raspberry Pi 3 to communicate with the chip using a combination of GPIO pins (for RESET and FLMD0 signals) and the UART pins. After issuing the *signature* command, we recovered the exact model of the MCU, namely the *µPD78F0515A*, a 64kB A-grade 78K0/KC2 MCU [Renb].

Bozzato et al. show several ways to read out the internal flash of such MCUs through voltage glitching attacks in [BFP19]. One attack is based on overwriting an empty flash section of the MCU with a custom piece of code that outputs the memory over a serial interface, and finally overwriting the reset vector to boot from this section. We improved on this attack by targeting the *Security Set* command. The 78K0/Kx2 series provides several levels of security: *block erase*, *chip erase*, *write* and *boot sector overwrite* protection, which cannot be reversed once enabled. The *Security Set* command sends a byte containing each security bit individually, after which the Renesas bootloader checks that none of the given bits reverse the current security settings. By voltage glitching this command, we disabled all security bits on the MCU, giving us the ability to erase and overwrite the boot section with our custom program to dump the firmware over a serial interface. This attack only requires a single successful glitch. To read the boot section as well, we repeated the attack with a second, identical immobiliser ECU, however now overwriting a different, reachable section with our own program. Once the microcontroller executes this

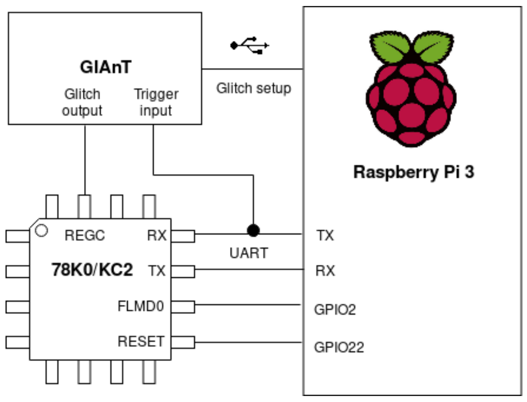

Figure 3: Our setup for the voltage glitching attack

section, the boot section is dumped and we have recovered the whole firmware. Because we need to perform the attack twice in this case, we require a total of two successful glitches, improving significantly on the so far most efficient known attack in [BFP19], which required a total of 15,000 successful glitches.

Hardware setup and glitch parameters. For glitching the firmware protection of the 78K0 immobiliser MCU, we used the Generic Implementation ANalysis Toolkit (GIAnT) [Osw16], an open-source FPGA-based board used for glitching and hardware analysis. We used a Raspberry Pi 3 running Raspbian to interface with the microcontroller's bootloader of the immobiliser ECUs and to control the glitching hardware. In order to read out the immobiliser ECU firmware, we implemented large parts of the Renesas Flash Programming Interface in Python. We acquired a 2011 Toyota Auris and a 2014 Toyota Hilux immobiliser ECU. We desoldered the chips from the ECUs, placed each of them on a breakout board, and connected a 16 MHz resonator to the oscillator pins `X1` and `X2`. As shown in Figure 3, we connect the voltage output from the GIAnT to the *REGC* pin in order to bypass the internal regulatory capacitor of the 78K0. Next. we confirmed that both chips contain the same firmware by comparing the values obtained by the *checksum* command.

With a glitch voltage of 0 V and a normal operating voltage of 2.7 V, we triggered on the first transmitted bit of the *Security Set* message from the Raspberry Pi, resulting in a successful glitch of 100 ns width at an offset of 596.78 μs on the first and 818.05 μs on the second chip. We attribute the significant difference in glitch offsets to the initial value of the security byte: in the Hilux MCU, the write protection for non-boot sectors was already disabled, while in the chip desoldered from the Auris immobiliser, all protection bits were initially set.

2.3 Reverse engineering the cipher

Due to most of the DST80 transponders being backwards compatible with DST40, our first assumption was that large parts of the cipher would be quite similar. Therefore, we statically analysed the firmware searching for the well known DST40 Feistel network. We located the equivalent of the *f*-boxes, *g*-boxes and *h*-box as used in DST40 in the firmware, and from there we could start reverse engineering the internals of the cipher. Due to the atypical number of rounds (200), we managed to locate the round function and more importantly, the round key schedule.

2.4 Cipher details

The DST80 cipher is, like DST40, an unbalanced Feistel network which runs for 200 rounds to calculate the response. It uses an 80-bit cryptographic key, split over two independent Key State Registers. The 40-bit internal state is initialised with the 40-bit challenge generated by the reader and updated with 2 bits each round.

Definition 1. Let $b_7 \dots b_0$ be the bit representation of byte b. Then the permutation $P_1(b) : \mathbb{F}_2^8 \to \mathbb{F}_2^8$ is defined as follows.

$$P_1(b_7 \dots b_0) = \begin{pmatrix} b_7 & b_6 & b_5 & b_4 & b_3 & b_2 & b_1 & b_0 \\ b_7 & b_3 & b_5 & b_1 & b_6 & b_2 & b_4 & b_0 \end{pmatrix}$$

Definition 2. Let $(S_{n-1}, S_{n-2} \dots S_1, S_0)$ be the byte representation of S. Given Permutation P_1 as defined in Definition 1, then we define the operation $P_2(S)$ as follows.

$$P_2(S) \leftarrow (P_1(S_{n-1}), P_1(S_{n-2}), \dots, P_1(S_1), P_1(S_0))$$

At the start of each round as depicted in Algorithm 2, the function *DST80_Merge(l, r)* combines the two 40-bit key state registers into one 40-bit round key as shown in Algorithm 1. It permutes the bytes of both keys according to P_1, after which it inverts each register depending on the value of its most significant bit. Finally, it returns a 40-bit round key which serves along with the internal state as input to the Feistel function F. This Feistel function is identical to its DST40 equivalent, resulting in a pair of bits which is fed back into the challenge register. For reference, we included the specification of the Feistel function in Appendix C. Unlike DST40 (where the key state register is shifted every three rounds) the LFSRs containing the two 40-bit keys are shifted each round, with tapped bits on positions 0, 2, 19 and 21.

Algorithm 1 Generation of the DST80 round key

```
1:  function DST80_Merge(keyL, keyR)
2:      keyL ← $P_2$(keyL)
3:      keyR ← $P_2$(keyR)
4:      if $\mathtt{keyL}_{39} == 1$ then
5:          keyL ← keyL ⊕ 0x7FFFFFFFFF
6:      end if
7:      if $\mathtt{keyR}_{39} == 1$ then
8:          keyR ← keyR ⊕ 0x7FFFFFFFFF
9:      end if
10:     return $(\mathtt{keyL}_{39\dots20} \mid \mathtt{keyR}_{39\dots20})$
11: end function
```

Algorithm 2 The DST80 round function

```
1: function DST80_Round(keyL, keyR, $s$)                ▷ With $s$ = internal state
2:     $k \leftarrow DST80\_Merge(\mathtt{keyL}, \mathtt{keyR})$
3:     $s \leftarrow (s \gg 2) \mid ((F(k, s) \oplus s_{1\dots0}) \ll 38)$
4:     $\mathtt{keyR} \leftarrow (\mathtt{keyR} \gg 1) \mid ((\mathtt{keyR}_0 \oplus \mathtt{keyR}_2 \oplus \mathtt{keyR}_{19} \oplus \mathtt{keyR}_{21}) \ll 39)$
5:     $\mathtt{keyL} \leftarrow (\mathtt{keyL} \gg 1) \mid ((\mathtt{keyL}_0 \oplus \mathtt{keyL}_2 \oplus \mathtt{keyL}_{19} \oplus \mathtt{keyL}_{21}) \ll 39)$
6: end function
```

3 Practical attacks on DST80 systems

This section describes several practical attacks on real-world DST80 systems. Through reverse engineering the immobiliser firmware, we present and analyse the security of two key diversification schemes used by three major car manufacturers. Furthermore, we illustrate how configuration issues of a transponder can lead to Denial-of-Service and downgrade attacks.

3.1 Uncovering key diversification schemes from immobiliser firmware

The security of the DST80 cipher relies on 80 bits of entropy in the cryptographic key, split up in two 40-bit chunks: `keyL` and `keyR`. Ideally, either the car manufacturer or Original Equipment Manufacturer (OEM) generates a random and unique 80-bit secret key for each paired transponder. However, in practice we have discovered that the key generation process is not random at all. We recovered the secret keys from the immobiliser ECUs of three different manufacturers and find recurring patterns among keys of each of them. In this Section, we bring to light the key diversification schemes we have recovered from studying the immobiliser firmware. With knowledge of this key generation scheme hidden in the immobiliser firmware an attacker can recover cryptographic keys from a DST80 transponder with only a single challenge response pair.

Attacker model. For the attacks described in this Section, we assume that the adversary can communicate with the DST80 transponder, either wirelessly or over a serial interface.

3.1.1 Kia and Hyundai low-entropy keys

While the microcontroller's flash memory is typically quite large and thus used for program code, the EEPROM usually contains small amounts data, specific to the car or even ECU. After having located serial numbers of the paired transponders in the EEPROM, we deemed it very likely that the corresponding DST80 keys would be stored there as well. Since the internal EEPROM of the microcontroller on the immobiliser is only 384 bytes, it is plausible to recover the DST80 key by doing an exhaustive search on the EEPROM data. First, we generate a challenge and acquire the corresponding signature from the transponder. Assuming the `keyL` and `keyR` are adjacent in memory, we set each sequence of 10 bytes as the cryptographic key and check whether they produce the same response. In order to eliminate any problems caused by endianness, we compute a signature with every key byte at every position in the 10-byte key. Note that for larger EEPROMs we can perform an entropy analysis to identify the areas most likely to contain a cryptographic key. Furthermore, if the preceding procedure does not produce a candidate key, we can enhance it by searching the binary in blocks of five bytes, or in the worst case we can search by byte. This can however result in state explosion and becomes infeasible for larger EEPROM sizes.

Using these techniques, we recovered the ten key bytes displayed in Table 3 from the internal EEPROM. A first observation is that even though two different physical transponders are paired to the immobiliser, they share the same cryptographic key. We can confirm this without knowledge of the DST80 key by sending the same challenge to both transponders, which generate identical responses.

Furthermore, Table 3 suggests that two bytes of both the `keyL` and the `keyR` are identical in all four of the DST80 keys, leaving six unknown bytes. Algorithm 3 clarifies the key generation scheme used by Kia and Hyundai, leaving only three bytes of entropy. Thus, provided they have brief access to a legitimate key fob, an attacker can immediately recover a valid DST80 key from a Kia/Hyundai transponder with only one challenge-response pair.

Table 3: Kia and Hyundai immobilisers and their respective DST80 keys. The two 2-byte constants X and Y have been redacted on the manufacturers request.

Make	Year & Model	`keyL`	`keyR`
Kia	2014 Carens	`955A3E`$\|\|$ X	Y $\|\|$ `C1A56A`
	2016 Ceed	`724560`$\|\|$ X	Y $\|\|$ `9FBA8D`
Hyundai	2013 I40	`756F1E`$\|\|$ X	Y $\|\|$ `E1908A`
	2016 IX20	`357B13`$\|\|$ X	Y $\|\|$ `EC84CA`

Algorithm 3 Recovering DST80 keys for transponders configured for Hyundai and Kia vehicles. The two 2-byte constants X and Y have been redacted on the manufacturers request.

```
1: function SEARCH_KEY(C, S)                    ▷ With C - Challenge; S - Signature
2:     for $i$ in $\{0 \ldots 2^{24}\}$ do
3:         $key_L[4], key_L[3], key_L[2] \leftarrow i[0], i[1], i[2]$
4:         $key_L[1], key_L[0] \leftarrow$ X
5:         $key_R[4], key_R[3] \leftarrow$ Y
6:         $key_R[2] \leftarrow \neg key_L[2]$
7:         $key_R[1] \leftarrow \neg key_L[3]$
8:         $key_R[0] \leftarrow \neg key_L[4]$
9:         $crc, sig \leftarrow DST80(C, key_L, key_R)$
10:        if $sig == S$ then return $key_L, key_R$
11:        end if
12:    end for
13: end function
```

Attack complexity. This attack only requires one challenge-response pair. We reduced the computational complexity to recover a cryptographic key from a Kia/Hyundai DST80 transponder from 2^{80} to 2^{24} encryptions.

3.1.2 Toyota key diversification scheme

We reverse engineered the Toyota immobiliser firmware to recover and analyse the security critical procedures contained within. Listing 4 depicts one of these procedures we reverse engineered from the firmware, namely the key derivation scheme for the DST80 cryptographic keys. Every time a transponder is presented, the immobiliser derives the DST80 key from its serial number (stored on page 3) and the 1-byte values stored on pages 1 and 2. Since the least significant byte of the DST80 transponder ID is the manufacturer code, this byte does not affect the key generation. Besides these three always readable transponder pages, the key derivation scheme relies on three security constants stored in the internal flash memory of the immobiliser. From our experiments, these security constants are identical across all Toyota DST80 immobilisers analysed. The fact that they are located in flash memory strengthens this hypothesis, since flash memory is less commonly used to store unique secrets.

Attack complexity. This attack only requires reading out three *public* transponder pages. Therefore, using the uncovered key derivation scheme, the security of this system is effectively reduced from 2^{80} to a few operations.

Algorithm 4 The Toyota key generation algorithm reverse engineered from immobiliser firmware. The three 5-byte constants $X_0 \dots X_2$ have been redacted on the manufacturers request.

```
$X \leftarrow [X_0, X_1, X_2]$
function GEN_KEY(page 1, page 2, id)
    $key_L \leftarrow (id \ll 16) \mid (page1 \ll 8) \mid page2$
    $key_R \leftarrow key_L$
    for $i$ in $\{0 \dots 7\}$ do
        $S_1 \leftarrow key_R$
        $S_2 \leftarrow key_R \oplus X[i \mod 3]$
        $S_3[0] \leftarrow ((S_2[0] + S_2[2] + S_2[3])\ \&\ \texttt{ff}) \lll 1$
        $S_3[1] \leftarrow ((S_2[2] + S_2[3] + S_2[4])\ \&\ \texttt{ff}) \lll 3$
        $S_3[2] \leftarrow ((S_2[0] + S_2[1] + S_2[3])\ \&\ \texttt{ff}) \lll 1$
        $S_3[3] \leftarrow ((S_2[0] + S_2[1] + S_2[4])\ \&\ \texttt{ff}) \lll 3$
        $S_3[4] \leftarrow ((S_2[1] + S_2[2] + S_2[4])\ \&\ \texttt{ff}) \lll 1$
        $key_R = S_3 \oplus key_L$
        $key_L = S_1$
    end for
     return $key_L, key_R$
end function
```

3.2 Transponder configuration issues

In Section 2.1 we outlined the different DST transponders available. The main difference in these transponders lies in the available interfaces. The wedge type transponder has a Low Frequency (LF) interface whereas the TMS37126 has both a Serial Peripheral Interface (SPI) and an LF interface. The TMS37F128 has both interfaces but the SPI is bonded internally to the MSP430 making it more difficult to access for an adversary.

These interfaces can be used to configure the transponder, to read and write values from and to the EEPROM storage and to request a cryptographic response to a provided challenge. All accesses to these interfaces are in fact accesses to an EEPROM page, each of which is five bytes in size and can be locked to prevent modification. For example, a cryptographic key can be set by writing to page 4 of the transponder. Afterwards, this page can be locked to prevent modification. By default a new DST transponder is configured to compute DST40 responses to a provided challenge. During car manufacturing or key fob pairing the transponder can be configured to use "80-bit mode" (DST80) by setting the correct bit in page 30 of the transponder.

In this section we will discuss multiple vulnerabilities which can be exploited using these interfaces when the transponder is not correctly configured during key fob manufacturing or pairing. We will assume that the transponder was configured to use DST80 and will discuss how configuration issues can reduce the security level provided by these transponders.

3.2.1 Downgrade attacks

We identified three distinct scenarios in which a downgrade attack could be used to reduce the security provided by these transponders. The first two scenarios do not require physical access to the key fob whereas the third scenario does require physical access.

Page 30 unlocked. A transponder configured to use DST80 with page 30 left unlocked can be downgraded to use DST40 instead. In this scenario, the transponder uses `keyL`, one of the two 40-bit keys used in DST80, to compute a DST40 response. This means that an adversary can downgrade the transponder and recover half of the 80-bit key using only two challenge response pairs, effectively reducing the computational complexity of the attack from 2^{80} to 2^{41} encryptions. After reverting back to 80-bit mode the transponder will again use the original 80-bit key. This attack requires short-range LF communication.

Page 4 and page 30 unlocked. Recent work shows that a DST40 key can be recovered using a 5.4 TB precomputed table, the response to a chosen challenge, a second challenge response pair and 2^{15} DST40 operations on average [WMA+19]. Our experiments show that we can recover a DST80 key with only twice the amount of resources if, in addition to page 30, page 4 is left unlocked. Similar to the previous scenario this attack requires short-range LF communication.

Specifically, an adversary can first alter the transponder's configuration to use DST40 instead of DST80 (because page 30 is left unlocked). The transponder will now use `keyL` with the DST40 cipher to compute the response to a provided challenge. At this point the adversary can use the attack as described in [WMA+19] to recover `keyL`. After recovering the 40-bit key we can change it to a chosen value (because page 4 was left unlocked) and we can reconfigure the transponder to use DST80. Empirical results show that in this scenario the transponder will compute DST80 responses using the original `keyR` in combination with the chosen `keyL`. The last step is now to recover `keyR` which can be achieved using a second precomputed table. We can generate this second table by computing the responses to a chosen 40-bit challenge for all possible values of `keyR` while keeping `keyL` fixed to a chosen value. In fact, we can do this more efficiently if we choose `keyL` to be the all 0 key. In that case the contents of the LFSR will remain constant throughout the 200-round execution of DST80 and most of the f-functions can be simplified because of a constant input, resulting in a more efficient software implementation.

We discovered that the Tesla Model S key fob released as a response to the attacks shown in [WMA+19] was not properly configured. These key fobs had both page 4 and page 30 unlocked, allowing for key recovery with approximately twice the effort compared to the previous attack. The downgrade attack reduces the computational complexity from 2^{80} to 2^{41} encryptions. Additionally, we only require 4 challenge response pairs and two 5.4 TB precomputed tables to recover the full 80-bit key in a matter of seconds.

Tesla was able to resolve the issue using an Over-The-Air software update that allowed customers to self-service their key fobs.

Page 4 and page 30 locked. The final scenario requires physical access to the key fob and arises when both page 4 and page 30 are locked down by the manufacturer. If the target chip has an easily accessible SPI interface the adversary can control the transponder from this interface and request DST40 responses even if the transponder is configured to use DST80, again reducing the computational complexity of the attack from 2^{80} to 2^{41}. While this attack is the easiest to apply in the case of a TMS37126 transponder it can also be applied to the TMS37F128 by exposing the bond wires interconnecting the MSP430 and TMS37126 or by replacing the firmware image running on the MSP430.

If the MSP430 does not have its JTAG fuse blown, a Commercial-Off-The-Shelf (COTS) MSP430 programmer can be used to create a backup of the firmware before replacing it with a malicious version. The malicious firmware image can query the transponder for the required challenge response pairs after which the adversary can restore the original firmware image. If the JTAG fuse is blown the adversary will first have to obtain a legitimate firmware image from a single key fob. They could achieve this by exploiting a vulnerability in the MSP430 bootloader, through a fault attack, or by outsourcing this firmware recovery step to a reverse engineering service provider [Goo08, BFP19]. Once the adversary has obtained a copy of the firmware, they can use the Interrupt Vector Table (IVT) stored at the end of flash as a password to unlock the bootloader in subsequent attacks. The remainder of the attack is the same as before; replace the firmware to get challenge response pairs and rewrite the original firmware image using the MSP430 bootloader instead of the JTAG interface.

3.2.2 Denial-of-Service attacks

Leaving transponder pages unlocked can lead to trivial Denial-of-Service (DoS) attacks. For example, if the configuration (page 30) is not write protected an adversary could change the transponder to use DST40 instead of DST80 (or vice versa) to prevent a user from starting their vehicle. Similarly, if the cryptographic key is not write protected an adversary can overwrite it to prevent a user from starting their vehicle. If their only goal is DoS they can additionally lock these pages permanently, preventing a dealership from repairing the key fob.

Performing this type of DoS attack can be automated by building a device which repeatedly broadcasts the required commands. While there might be little incentive for someone to do this type of attack it could lead to bad publicity for the affected car manufacturers and increased revenue for local garage owners.

3.3 Discussion and mitigation

With regard to the weak key generation based attacks described in Sections 3.1.1 and 3.1.2, we believe that one reason why manufacturers choose to implement such a scheme is to facilitate the key pairing process. For instance, certain Toyota models allow programming a new key fob by inserting it into the ignition and performing a precise sequence of actions [toy]. This is only possible because the immobiliser can derive the DST80 key from transponder pages 1-3 as described in Section 3.1.2. Instead of relying on a weak key generation scheme, the manufacturer can mitigate this by implementing a secure-diagnostics based solution for pairing a new key, such as the solution proposed in [dHG18].

It is of the utmost importance that DST transponders are configured correctly to offer any security guarantees. Any car manufacturer or OEM using these transponders should ensure that page 4 and page 30 are locked. Furthermore, locking other transponder pages should be considered if they are not used. While migrating the key programming procedure is an involved task for the manufacturer, the misconfigured transponders described in Section 3.2.1 are easier to fix. For instance, Tesla fixed the issues with their DST80 key fobs with a software update issued a few months after disclosure. Finally, enabling mutual authentication on the transponder would mitigate transponder-only attacks as described in Sections 3.1.1 and 3.2.1. These require a challenge response pair, which an attacker could not obtain without knowing the encryption key. However, if the adversary has access to the car they can still acquire a challenge response pair by eavesdropping the wireless communication between the transponder and immobiliser ECU.

4 Side-Channel Analysis

The attacks described in Section 3 require knowledge of the manufacturer's key generation scheme or transponders with pages 4 and 30 unlocked. In this section we describe how we use power analysis to recover a *randomly generated* cryptographic key from a securely configured DST transponder. In Subsection 4.1 we show how to recover the initial 40-bit round key in DST80 (or the full key in DST40) using Differential Power Analysis (DPA) with a difference of means distinguisher. In Subsection 4.2 we describe a profiled side-channel attack which targets the key loading procedure performed before every invocation of the cipher.

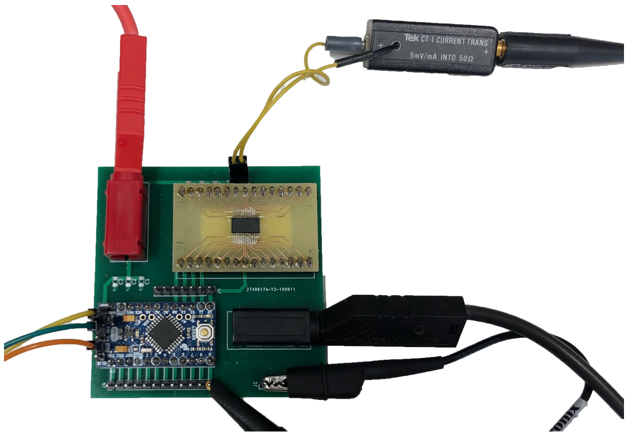

Figure 4: Measurement setup for acquiring power traces from a TMS37126 IC (on the yellow breakout board). Top-right: Tektronix CT-1 current probe. Bottom-left: Arduino Pro Mini functioning as a serial to SPI translator.

Hardware setup. To acquire power measurements for the side-channel attacks we used a Tektronix DPO 7254C oscilloscope in combination with a Tektronix CT-1 current probe, Mini-Circuits ZFL-1000LN+ amplifier and 20 MHz lowpass filter. Our measurement setup is using a TMS37126 IC as the target and is shown in Figure 4. We captured the traces using a sample rate of 200 Megasamples Per Second (MSPS). Similar results could likely be obtained using low-cost and open source side-channel evaluation tools such as the ChipWhisperer Lite, which can sample up to 105 MSPS [OC14].

4.1 Differential Power Analysis

In this section we describe a DPA attack which allows recovering the 40-bit DST40 key or the initial 40-bit round key in DST80. This initial 40-bit round key reveals 20 bits of each of the 40-bit keys `keyL` and `keyR`.

We mainly focus on the first round of the DST ciphers, as recovering the initial round key reveals the entire secret key in the case of DST40 or half of it in the case of DST80. As the attack is identical for both DST80 and DST40 we will use a simplified model of the cipher in which a 40-bit known value is combined with a 40-bit secret to produce a 2-bit round output (i.e. $F(b, c)$ as shown in Listing 2 and defined in [BGS+05]).

The output of the round function is based on four functions (or g-boxes, reverse engineered in [BGS+05]), each of which combines 10 bits from the known input with 10 bits from the secret. During the DPA attack we will acquire power traces in which only the 10 bits of the input affecting the output of one of the four functions is varied. Depending on the state of the remaining three g-boxes, varying this input will result in two possible output values. Therefore, based on a 10-bit subkey guess, the acquired traces can be split into two groups. Different key guesses can be evaluated using the difference of means distinguisher. For each 10-bit subkey there are two keys which produce the same division of traces. Therefore, we end up with two candidate keys for each of the four 10-bit subkeys. The remaining key bits can be recovered by means of exhaustive search ($2^4 \cdot 2^{40}$) or by extending the attack to more rounds of the cipher.

Empirical results show that approximately 7500 traces are required for each of the four subkeys. With our current setup, we can measure approximately four traces per second. Hence, it would be rather cumbersome and time-consuming (around two hours) to acquire 30,000 traces from a key fob during the attack phase. Therefore, we consider a more practical profiled attack in the following sections.

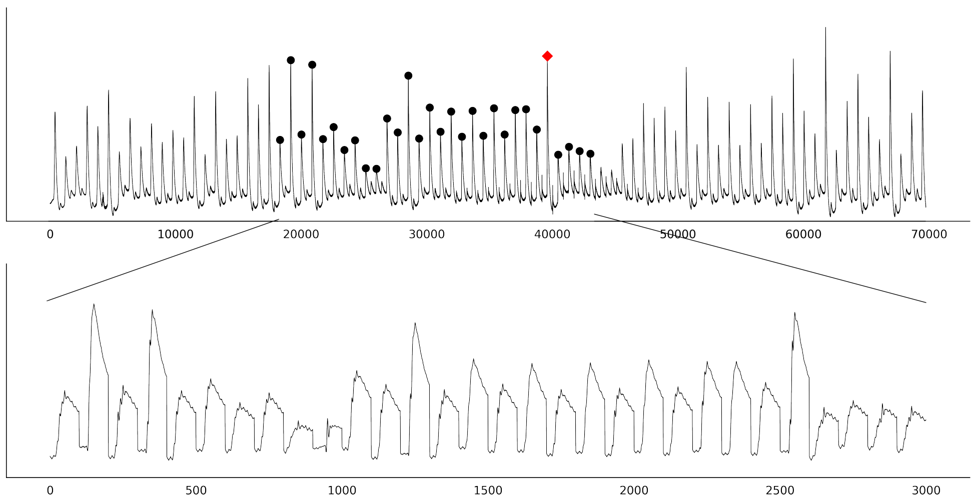

Figure 5: Average power trace over 1000 measurements (top). The traces were aligned on the highest peak close to time sample 40000, indicated by a red diamond. To achieve alignment across multiple devices we identified the peaks of interest (black dots) and selected 100 time samples around those to form artificial traces (bottom).

4.2 Profiled Side-Channel Attack

While investigating the power traces from a transponder we noticed clear differences in the power consumption based on the value of the key. To further investigate we acquired multiple traces with many different keys. The Hamming weight correlation of each key byte value clearly showed that each byte of the key is loaded sequentially before the DST80 operation starts.

In this section we describe how we preprocessed the raw traces to form aligned traces of 3000 samples each. Afterwards, we consider two distinct scenarios, in the first scenario the adversary attempts to recover the key from the same device as the one being profiled (single device scenario). In the second scenario, the multiple devices scenario, the adversary has multiple devices for profiling but has to attack a device which was not part of the profiling set. In this work we explore using machine learning techniques to tackle both these scenarios.

We show that in the case of a single device the exact value of each key byte can be easily recovered using a basic MLP model and as little as ten traces on average. Using these same models on an unseen device does not give good results. Therefore, we explore training the model on five devices before attacking a sixth, unseen device. In Section 4.2.4 we investigate which samples are used by the MLP models to perform classification, exploring the differences between the models.

4.2.1 Preprocessing and identifying regions of interest

For both the single device and multiple device scenario we preprocess the traces using the same technique. This consists of a rough alignment based on a single power trace peak followed by peak extraction (Figure 5). During the last step, samples close to the peaks of interest are selected to form new and aligned traces. Using this technique we obtain aligned traces even though each target chip has a slightly different operating frequency and suffers from clock drift. In the end we reduce each raw trace consisting of 70k samples to a new trace of 3k samples. Using the sum of squared pairwise t-differences (SOST) we identified the regions of interest in the power traces [GLP06]. Specifically, we used the

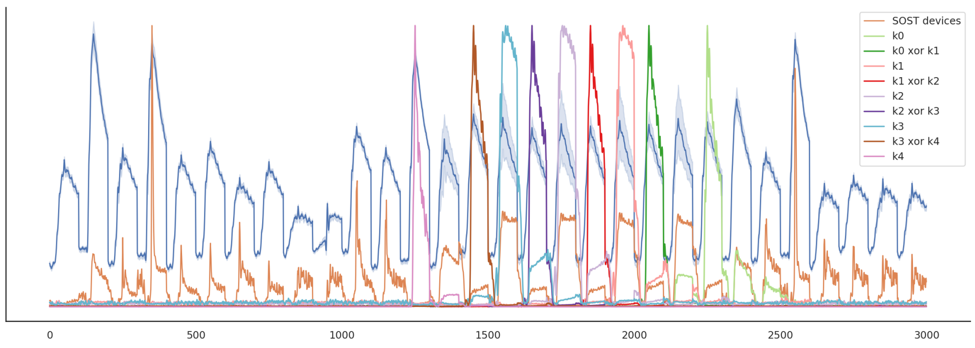

Figure 6: The average preprocessed trace with error bars indicating the spread over the different devices (in blue). Additionally, for each of the nine labels, this plot shows the regions of leakage identified by the SOST. The SOST result shown in orange indicates the locations in the trace containing information on the inter-device difference. All of the SOST results are scaled to fit the plot. Best viewed on screen.

SOST to uncover regions in the power traces which contain information on the loaded key byte and regions which contain information on the inter-device variability. Using the SOST technique we identified leakage for each byte of the key (k_i) and the difference between subsequent bytes ($k_i \oplus k_{i+1}$). Additionally, we identified regions which contain information identifying the different devices.

Figure 6 shows the average power trace including error bars indicating the spread over six distinct transponder chips. While we can observe a slight difference in DC offset it is clearly not the only difference between the devices. Therefore, correcting for the DC offset as proposed by Choudary and Kuhn [CK18] will not solve all problems in this case.

Figure 6 additionally shows which areas of the power trace contain information on the inter-device differences. Having this information will hopefully allow to generalise to unseen devices. Finally, Figure 6 also shows the SOST results for each of the nine labels. While we observe leakage for each of the byte values (k_i) we also observe leakage for the distance between consecutive byte loads ($k_i \oplus k_{i+1}$). Note that there is a rather large spread in the mean power traces in the regions containing information on k_1, k_2 and k_3. For the remainder of this paper, we will focus on attacking the first half of the DST80 key, as `keyL` and `keyR` are loaded sequentially.

4.2.2 Single device

In the single device scenario an adversary is allowed to create a leakage model (profiling phase) on the same device as the target device (attack phase). For DST transponders this type of attack is applicable if the target transponder does not have the key (page 4) locked. In this scenario an adversary can first acquire a set of attack traces. Afterwards, the key can be varied to acquire profiling traces. A classifier trained on the profiling traces can then be used to recover the key from the attack traces, if necessary the key can also be reprogrammed to the key fob to restore its functionality.

To test the applicability of a single device attack we acquired one million traces with random keys from a single transponder. After preprocessing the traces we end up with a dataset of 950k traces of which 850k traces are used for training, 50k for validation and 50k as a test or attack set.

We trained the same model on each of the nine labels identified in Section 4.2.1. The Multi-Layer Perceptron (MLP) model we employed is listed in Appendix A, this model was obtained by manual tuning and is by no means the best possible model. Similarly

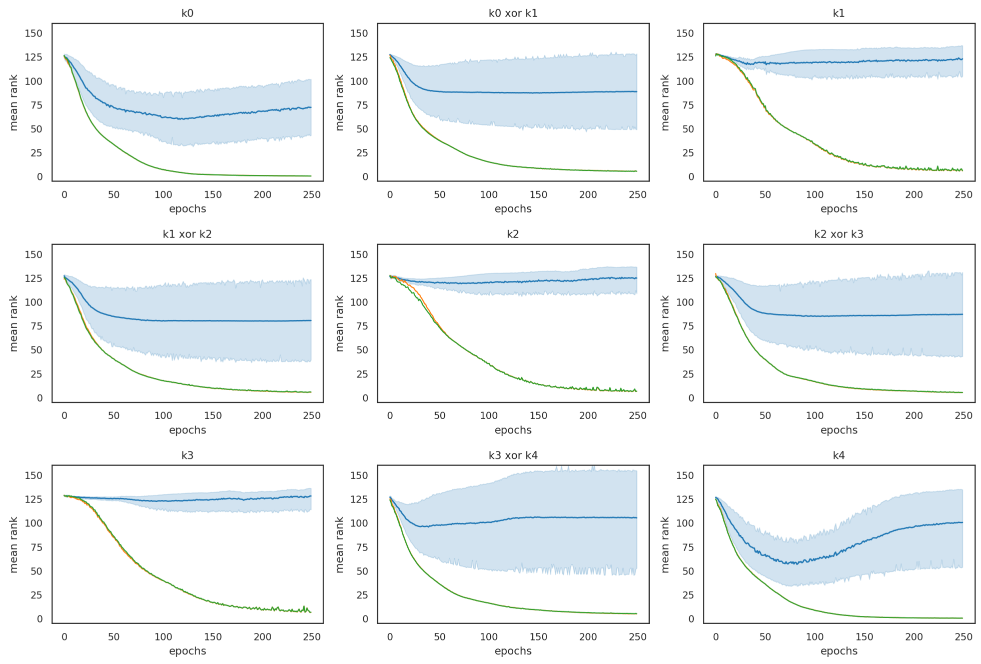

Figure 7: Average validation (orange) and attack rank (green) versus the number of epochs for each of the labels, the two lines mostly overlap. Using the model trained on a single device to attack any of the other five devices we obtain the average rank depicted in blue (with error bands).

to Picek et al. we observed that accuracy is often not the best metric to evaluate the trained models [PHJ+19]. We observe that selecting the best model solely on the loss and accuracy values is unlikely to result in selecting the model which performs best during an attack in which traces are combined. Therefore, we propose to evaluate the trained models by looking at the average rank of the correct label at each epoch for a fixed number of attack traces.

To that end we classify 50 traces with the same label for each of the 256 possible values of each label, the classifier outputs are combined using the log-likelihood method. For each label the results of these 256 experiments are averaged to obtain the mean rank over all possible label values using 50 attack traces. We repeat this evaluation at each epoch of the MLP training process. The results for training a model on traces from a single device are depicted in Figure 7, the figure also shows the results for using that same model to try and attack an unseen device. From these results it is clear that attacking the same device as the profiling device is trivial, for each of the nine labels the models perform well even when the number of traces used for the attack is as low as 50. In fact, using as little as 10 traces seems to be sufficient to achieve similar results, adding more traces does not significantly increase the performance.

Figure 7 also shows the results when using the models trained on a single device to attack five unseen devices. In this case, a model trained on a single device cannot be used to attack an unseen device unless it happens to match the training device very closely. Note that the models trained on labels k_1, k_2 and k_3 do not seem to generalise across devices. Recall from Section 4.2.1 that samples containing information on these labels significantly differ across different devices.

In the hope of obtaining a more general model over the family of devices we explore training our models on multiple devices in the following section.

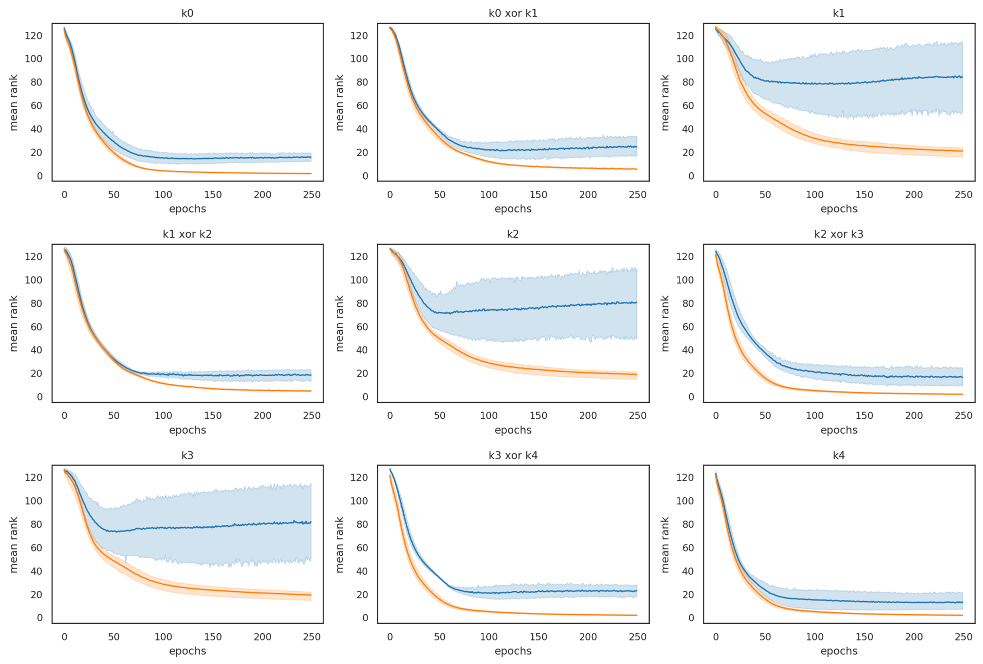

Figure 8: Average validation (orange) and attack rank (blue) versus the number of epochs for each of the labels over the 6-fold cross-validation indicated by the error bands.

4.2.3 Multiple devices

The single device scenario is often used in the literature and in security evaluations to evaluate the worst-case scenario. However, in most cases this overestimates the model's ability to generalise to a different device.

In the multiple devices scenario the adversary has access to multiple unlocked transponder chips and can capture as many power traces as necessary during the profiling stage. Their end goal is to use the model generated during the profiling phase to recover the cryptographic key from an unseen device.

To evaluate this scenario we acquired one million power measurements from six transponders using random keys. After preprocessing the traces we end up with 950k traces for each of the devices. During our experiments we used 900k traces from each device for training and 50k traces for evaluation, giving us a training set of 4.5 million traces and a validation set of 250k traces. To ensure that the models generalise to an unseen device we use the traces from the remaining device as an attack set. We perform this experiment six times, each time using the traces from a different device as the attack set (cross-validation).

Similar to Section 4.2.2 we use 50 traces of the validation and attack set to evaluate our models at each training epoch. Figure 8 shows the average validation and attack rank at each training epoch for each target label. The error bands indicate the results over the six experiments, each using a different device as the attack set. While the improvement over the single device scenario is remarkable, the spread in the average attack rank for some of the labels indicates that the models would likely benefit from more distinct devices during the training process. It is clear that predicting k_1, k_2 and k_3 is more difficult than the other labels, this was to be expected from the results obtained in the single device scenario (Figure 7) and by observing the large spread in the average traces in Figure 6. We will disregard the models trained on k_1, k_2 and k_3 in the remainder of the paper, as they cannot be reliably used for classification.

To evaluate the relationship between the number of attack traces and the mean rank we select the weights for the model at a specific training epoch for each of the labels. This selection is performed solely based on the validation rank, as would be the case in a real world scenario. For each of the six remaining labels we thus select the model at a certain epoch where the validation rank is the lowest. As we did notice slight overfitting in some cases we also bound the model selection to not include models which were trained for more than 175 epochs. After selecting the best model for each label we used this model to classify an increasing amount of attack traces for each possible value of that label and recorded the average rank. For each number of traces we averaged the results over five distinct sets of traces. The results of these experiments indicate that using as little as 10 power traces provides stable results in the output of the models. Furthermore, adding more traces does not significantly impact the rank of the correct value.

4.2.4 Single device vs. multiple devices

In Section 4.2.1 we identified regions of interest using the SOST method. In this section we visualise the samples which receive most attention from the MLP models to perform the classification.

To visualise which samples are considered more important by the MLP model we analysed the weights of the input layer. In general, for a trace of n samples and an MLP model with p input neurons, the model will learn $n \cdot p$ input weights. In this case the model learns one weight for each input sample (3000) in each of the first layer neurons (500), hence the input layer contains 1.5 M $(3000 \cdot 500)$ weights.

To visualise the attention given to each input sample we use the L_∞ norm to reduce the $3000 \cdot 500$ weight matrix to 3000 values of which the magnitude is an indication for the sample's importance. We repeat this for each of the trained models at each epoch, in this way we get information on the important samples and how this evolved during training of the model. Appendix B contains Figure 9 which depicts the results for the models trained on a single device and Figure 10 the results for models trained on five devices. For the single device case the areas of interest identified by the MLP model are mostly consistent with the SOST from Figure 6. Alternatively, an occlusion or gradient based visualization technique can be used.

For the multiple device scenario we initially expected the models to combine the samples containing most information on the inter-device variation (Figure 6) with those containing information on the target labels. However, in the multiple device scenario the models for $k_i \oplus k_{i+1}$ seem to mostly combine the samples identified to contain information on $k_i \oplus k_{i+1}$ and k_{i+1}. This observation indicates that a multi-label time-series classifier could perform better as it might be able to learn the relationship between labels k_i, $k_i \oplus k_{i+1}$ and k_{i+1}.

4.2.5 Key enumeration

During the attack phase an adversary will collect traces from a target transponder. After preprocessing the attack traces, the trained models can be used to predict each of the six labels. At this point the adversary has six lists of labels, each ranked according to the likelihood provided by the classifier. However, as our classifiers are not perfect the adversary will have to enumerate some of the remaining key candidates.

We use the classifiers trained on the following six labels: k_0, $k_0 \oplus k_1$, $k_1 \oplus k_2$, $k_2 \oplus k_3$, $k_3 \oplus k_4$, k_4. The fact that we obtain six labels for only five target variables (key bytes) can be used to our advantage as we can easily eliminate keys using a few xor operations instead of executing the full cipher. In practice we achieve this by bounding the number of candidates for each label based on the accuracy of our model on the independent attack set. Concretely, we can form tuples of candidates from the first five labels and check which of those hold according to the candidates in the bounded list of k_4. By keeping track of the product of the individual probabilities for each of the labels we can compute an overall likelihood for this combination of labels and thus the candidate key.

As on average the correct label for each of the six used target labels is close to 20 (Figure 8), we would have to test 20^5 or 2^{21} keys on average, of which a substantial portion will be invalid and thus eliminated. Assuming we can obtain similar results for the second 40-bit key which is being loaded sequentially, this would result in a practical key recovery attack requiring 10 power measurements and about 2^{40} DST80 operations.

In scenarios where the models perform worse or for larger key sizes it could be worthwhile to extend the work by Li et al. on key enumeration with dependent score lists [LWWW17]. Alternatively, if the models for k_1, k_2 and k_3 performed better (e.g. by having more training devices or a multi-label classifier) this enumeration strategy could be applied directly, further decreasing the required effort.

5 Conclusion

In this paper we demonstrated the insecurity of two different immobiliser systems used by three major car manufacturers. By glitching the firmware protection of the microcontroller we read the firmware of several immobiliser ECUs. From this firmware, we reverse engineered the proprietary Texas Instruments DST80 cipher. We present the cipher here for the first time, which enables other researchers to scrutinise its security. Additionally, we demonstrate that Toyota derives an 80-bit cryptographic key for each transponder based on its serial number and other publicly readable transponder data. Similarly, we show how transponders used in Kia and Hyundai cars rely on three bytes of entropy in the encryption key. In each of these cases an attacker can recover the full cryptographic key in a matter of milliseconds. Additionally, we show how insufficient diligence from the manufacturer can lead to a vulnerable configuration of the transponder, resulting in practical key recovery and Denial of Service attacks. Moreover, we propose several side-channel attacks to reduce the key space from 2^{80} to 2^{40} for properly configured transponders with a randomly generated key. More specifically, we propose a Differential Power Analysis attack to recover the first 40-bit round key containing 20 bits of each initial key state register. The attack targets a hardware implementation [Kai08] of the cipher and requires 30k power traces. Finally, we present a profiled side-channel attack in which we target the key being read from the transponder's EEPROM. We show that this is a trivial task in the single device scenario, while in the more realistic scenario in which we target an unknown device, we reduce the key space to approximately 2^{40} with as little as 10 power traces.

This wide range of attacks summarised in Table 4 demonstrates the large attack surface of security critical systems like the vehicle immobiliser. An effective immobiliser system does not only require secure cryptographic primitives, but also a secure key generation scheme, rigorously configured transponders and secure hardware implementations. In this paper we address each of these elements and describe several countermeasures to mitigate the proposed attacks.

Responsible disclosure. We informed Toyota, Kia, Hyundai and Tesla of the identified issues and provided each with a tailored report. Additionally, we informed Texas Instruments about our intention of publishing the DST80 cipher and provided them with details on the downgrade and side-channel attacks. Per request of the manufacturers we redacted the constants in Algorithms 3 and 4.

Table 4: Attacks on DST80 transponders described in this paper. In the scenarios marked by an asterisk lookup tables can be employed to speed up the key recovery process.

Affected	Prerequisites	Consequences
Toyota	Communication with the transponder	Key recovery (Algorithms 3 and 4)
Kia	Single authentication trace	
Hyundai		
Tesla	Write access to transponder pages 4 and 30	complexity 2^{41}*
	Write access to transponder page 30	complexity 2^{41}*
All key fobs using TMS37F128 or TMS37126	Physical access (DPA $\pm 30k$ traces)	complexity 2^{44}
	Physical access (Profiled SCA ± 10 traces)	complexity $\pm 2^{40}$
	Physical access (SPI interface)	complexity 2^{41}*

Acknowledgments

This work was supported in part by the Research Council KU Leuven C1 on Security and Privacy for Cyber-Physical Systems and the Internet of Things with contract number C16/15/058. In addition, this work was supported in part by the Flemish Government through FWO Sancus G.0876.14N and in part by the European Commission through the Horizon 2020 research and innovation programme under grant agreement Cathedral ERC Advanced Grant 695305. Additionally, this work was partly funded by EPSRC Fellowship EP/R008000/1.

References

[BFP19] Claudio Bozzato, Riccardo Focardi, and Francesco Palmarini. Shaping the Glitch: Optimizing Voltage Fault Injection Attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(2):199–224, 2019.

[BGS+05] Steve Bono, Matthew Green, Adam Stubblefield, Ari Juels, Aviel D Rubin, and Michael Szydlo. Security Analysis of a Cryptographically-Enabled RFID Device. In *USENIX Security Symposium*, volume 31, pages 1–16, 2005.

[Bog07] Andrey Bogdanov. Linear Slide Attacks on the KeeLoq Block Cipher. In Dingyi Pei, Moti Yung, Dongdai Lin, and Chuankun Wu, editors, *Information Security and Cryptology, Third SKLOIS Conference, Inscrypt 2007, Xining, China, August 31 - September 5, 2007, Revised Selected Papers*, volume 4990 of *Lecture Notes in Computer Science*, pages 66–80. Springer, 2007.

[Bok17] Wouter Bokslag. An assessment of ECM authentication in modern vehicles. Master's thesis, Eindhoven University of Technology, 2017.

[CBW08] Nicolas Courtois, Gregory V. Bard, and David A. Wagner. Algebraic and Slide Attacks on KeeLoq. In Kaisa Nyberg, editor, *Fast Software Encryption, 15th International Workshop, FSE 2008, Lausanne, Switzerland, February 10-13, 2008, Revised Selected Papers*, volume 5086 of *Lecture Notes in Computer Science*, pages 97–115. Springer, 2008.

[CCC+19] Mathieu Carbone, Vincent Conin, Marie-Angela Cornelie, François Dassance, Guillaume Dufresne, Cécile Dumas, Emmanuel Prouff, and Alexandre Venelli. Deep Learning to Evaluate Secure RSA Implementations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(2):132–161, 2019.

[CDP17] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional Neural Networks with Data Augmentation Against Jitter-Based Countermeasures - Profiling Attacks Without Pre-processing. In *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, pages 45–68, 2017.

[CK14] Omar Choudary and Markus G. Kuhn. Template Attacks on Different Devices. In *COSADE*, volume 8622 of *Lecture Notes in Computer Science*, pages 179–198. Springer, 2014.

[CK18] Marios O. Choudary and Markus G. Kuhn. Efficient, Portable Template Attacks. *IEEE Trans. Information Forensics and Security*, 13(2):490–501, 2018.

[Com95] European Commission. Commission Directive 95/56/EC, Euratom of 8 November 1995 adapting to technical progress Council Directive 74/61/EEC relating to devices to prevent the unauthorized use of motor vehicles. 1995.

[CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template Attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002.

[dHG18] Jan Van den Herrewegen and Flavio D. Garcia. Beneath the Bonnet: A Breakdown of Diagnostic Security. In Javier López, Jianying Zhou, and Miguel Soriano, editors, *Computer Security - 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, September 3-7, 2018, Proceedings, Part I*, volume 11098 of *Lecture Notes in Computer Science*, pages 305–324. Springer, 2018.

[FDC11] Aurélien Francillon, Boris Danev, and Srdjan Capkun. Relay Attacks on Passive Keyless Entry and Start Systems in Modern Cars. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2011, San Diego, California, USA, 6th February - 9th February 2011*. The Internet Society, 2011.

[GDD$^+$19] Anupam Golder, Debayan Das, Josef Danial, Santosh Ghosh, Shreyas Sen, and Arijit Raychowdhury. Practical Approaches Toward Deep-Learning-Based Cross-Device Power Side-Channel Attack. *IEEE Trans. VLSI Syst.*, 27(12):2720–2733, 2019.

[GLP06] Benedikt Gierlichs, Kerstin Lemke-Rust, and Christof Paar. Templates vs. Stochastic Methods. In Louis Goubin and Mitsuru Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings*, volume 4249 of *Lecture Notes in Computer Science*, pages 15–29. Springer, 2006.

[GOKP16] Flavio D. Garcia, David Oswald, Timo Kasper, and Pierre Pavlidès. Lock It and Still Lose It - on the (In)Security of Automotive Remote Keyless Entry Systems. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 929–944. USENIX Association, 2016.

[Goo08] Travis Goodspeed. Practical Attacks against the MSP430 BSL. In *Twenty-Fifth Chaos Communications Congress. Berlin, Germany*, 2008.

[HGO18] Christopher Hicks, Flavio Garcia, and David Oswald. Dismantling the AUT64 Automotive Cipher. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(2):46–69, May 2018.

[IKD$^+$08] Sebastiaan Indesteege, Nathan Keller, Orr Dunkelman, Eli Biham, and Bart Preneel. A Practical Attack on KeeLoq. In Nigel P. Smart, editor, *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, volume 4965 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2008.

[Kai08] Ulrich Kaiser. Digital Signature Transponder. In Paris Kitsos and Yan Zhang, editors, *RFID Security: Techniques, Protocols and System-on-Chip Design*, pages 177–189. Springer US, Boston, MA, 2008.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, pages 388–397, 1999.

[KKMP09] Markus Kasper, Timo Kasper, Amir Moradi, and Christof Paar. Breaking KeeLoq in a Flash: On Extracting Keys at Lightning Speed. In Bart Preneel, editor, *Progress in Cryptology - AFRICACRYPT 2009, Second International Conference on Cryptology in Africa, Gammarth, Tunisia, June 21-25, 2009. Proceedings*, volume 5580 of *Lecture Notes in Computer Science*, pages 403–420. Springer, 2009.

[LWWW17] Yang Li, Shuang Wang, Zhibin Wang, and Jian Wang. A Strict Key Enumeration Algorithm for Dependent Score Lists of Side-Channel Attacks. In *Smart Card Research and Advanced Applications - 16th International Conference, CARDIS 2017, Lugano, Switzerland, November 13-15, 2017, Revised Selected Papers*, pages 51–69, 2017.

[MBTL13] David P. Montminy, Rusty O. Baldwin, Michael A. Temple, and Eric D. Laspe. Improving cross-device attacks using zero-mean unit-variance normalization. *J. Cryptographic Engineering*, 3(2):99–110, 2013.

[OC14] Colin O'Flynn and Zhizhang (David) Chen. ChipWhisperer: An Open-Source Platform for Hardware Embedded Security Research. In Emmanuel Prouff, editor, *Constructive Side-Channel Analysis and Secure Design - 5th International Workshop, COSADE 2014, Paris, France, April 13-15, 2014. Revised Selected Papers*, volume 8622 of *Lecture Notes in Computer Science*, pages 243–260. Springer, 2014.

[Osw16] David Oswald. Generic Implementation ANalysis Toolkit, 2016. available online at https://sourceforge.net/projects/giant.

[OT17] Johannes Obermaier and Stefan Tatschner. Shedding too much Light on a Microcontroller's Firmware Protection. In William Enck and Collin Mulliner, editors, *11th USENIX Workshop on Offensive Technologies, WOOT 2017, Vancouver, BC, Canada, August 14-15, 2017*. USENIX Association, 2017.

[PHJ$^+$19] Stjepan Picek, Annelie Heuser, Alan Jovic, Shivam Bhasin, and Francesco Regazzoni. The Curse of Class Imbalance and Conflicting Metrics with Machine Learning for Side-channel Evaluations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(1):209–237, 2019.

[PSB+18] Emmanuel Prouff, Rémi Strullu, Ryad Benadjila, Eleonora Cagli, and Cécile Dumas. Study of Deep Learning Techniques for Side-Channel Analysis and Introduction to ASCAD Database. *IACR Cryptology ePrint Archive*, 2018:53, 2018.

[Rena] Renesas Electronics. *78K0/Kx2 Flash Memory Programming.*

[Renb] Renesas Electronics. *78K0/Kx2 User's manual: Hardware.*

[SLP05] Werner Schindler, Kerstin Lemke, and Christof Paar. A Stochastic Model for Differential Side Channel Cryptanalysis. In *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, pages 30–46, 2005.

[STM] STMicroelectronics. *STM8AF Series.*

[Tex12a] Texas Instruments. *Digital Signal Transponder With DST80 Authentication, EEPROM, and LF Immobilizer*, 2012.

[Tex12b] Texas Instruments. *TMS3705 Transponder Base Station IC*, 2012.

[Tex12c] Texas Instruments. *TMS37126 - Remote access identification device with integrated 3D wakeup receiver and immobilizer interface*, 2012.

[Tex12d] Texas Instruments. *TMS37F128 - Controller Remote Access Identification (CRAID) with integrated microcontroller, 3D wakeup receiver, and immobilizer interface*, 2012.

[toy] Toyota Transponder Chip Key & Remote Key Fob Programming Instructions. http://www.diy-time.com/automotive/toyota/program-chip-toyota-key-remote-fob/.

[traa] Car Keys Online. https://www.car-keys-online.com.

[trab] Transpondery. http://www.transpondery.com/transponder_catalog/toyota_transponder_catalog.html.

[VGB12] Roel Verdult, Flavio D. Garcia, and Josep Balasch. Gone in 360 Seconds: Hijacking with Hitag2. In Tadayoshi Kohno, editor, *Proceedings of the 21th USENIX Security Symposium, Bellevue, WA, USA, August 8-10, 2012*, pages 237–252. USENIX Association, 2012.

[VGE15] Roel Verdult, Flavio D. Garcia, and Baris Ege. Dismantling Megamos Crypto: Wirelessly Lockpicking a Vehicle Immobilizer. In Samuel T. King, editor, *Proceedings of the 22th USENIX Security Symposium, Washington, DC, USA, August 14-16, 2013*, pages 703–718. USENIX Association, 2015.

[WMA+19] Lennert Wouters, Eduard Marin, Tomer Ashur, Benedikt Gierlichs, and Bart Preneel. Fast, Furious and Insecure: Passive Keyless Entry and Start Systems in Modern Supercars. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(3):66–85, 2019.

[ZOB18] Yevhenii Zotkin, Francis Olivier, and Eric Bourbao. Deep Learning vs Template Attacks in front of fundamental targets: experimental study. *IACR Cryptology ePrint Archive*, 2018:1213, 2018.

A Multi-Layer Perceptron model

We used the following MLP model for each of the nine labelings of the data. This is by no means the best possible model as we did not perform an exhaustive parameter search. The model was implemented using Keras and uses the categorical cross-entropy loss function with the Stochastic Gradient Desecent (SGD) optimizer with a learning rate of 0.0001, a momentum of 0.95 and the Nesterov momentum enabled. We used a batch size of 10000 during training over multiple devices and a batch size of 2500 when training on a single device.

- Fully connected layer consisting of 500 perceptrons with relu activation.
- Fully connected layer consisting of 100 perceptrons with relu activation.
- Fully connected layer consisting of 50 perceptrons with relu activation.
- Fully connected layer consisting of 25 perceptrons with relu activation.
- Fully connected layer consisting of 16 perceptrons with relu activation.
- Fully connected layer consisting of 256 perceptrons with softmax activation.

B MLP input layer weight visualizations

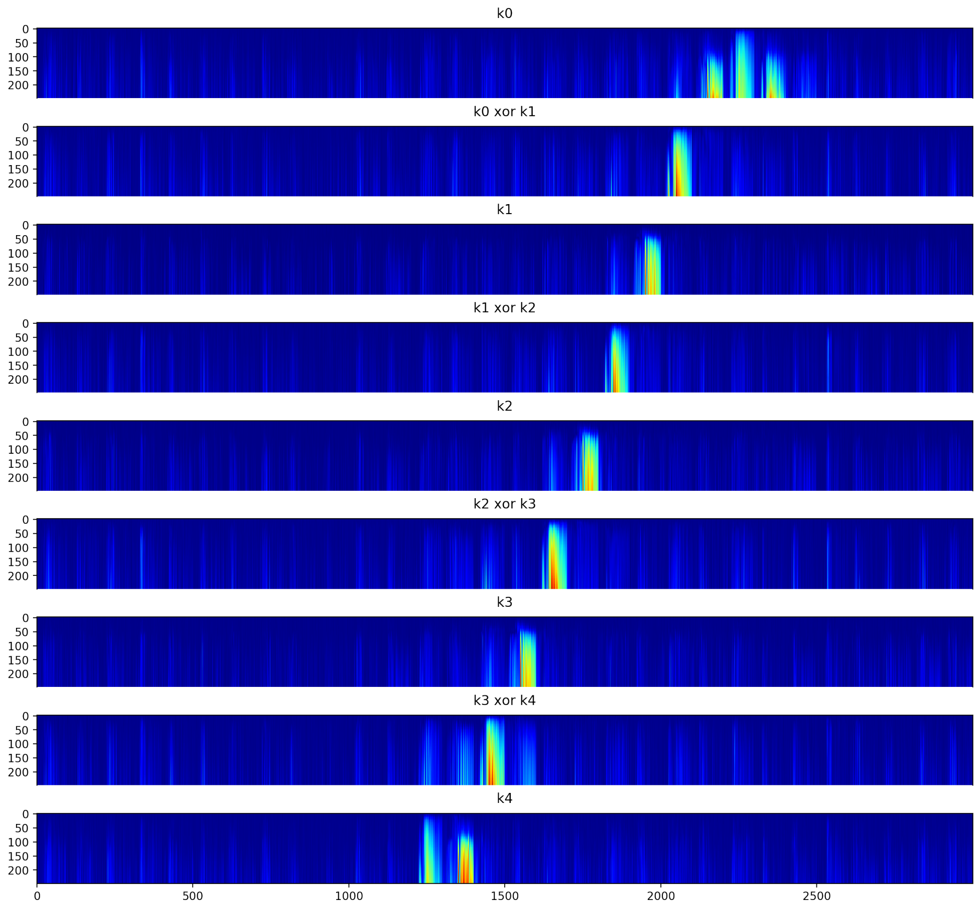

Figure 9: L_∞ norm of the input weights over the 250 training epochs (y-axis) for models trained on a single device.

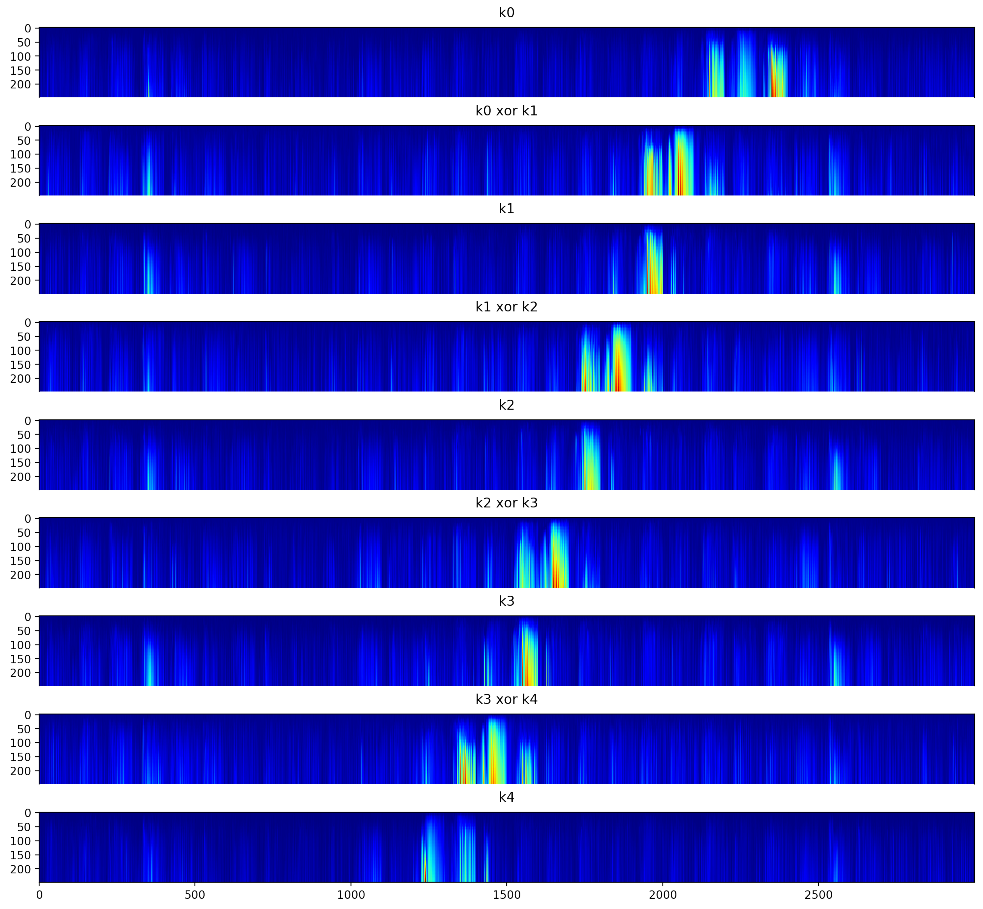

Figure 10: L_∞ norm of the input weights over the 250 training epochs (y-axis) for a model trained on five distinct devices.

C Specification of the DST80 function $F(k, s)$

In this Section we specify the Feistel function F, originally defined in [BGS+05], again. We present it here as we reverse engineered it from the immobiliser firmware.

Table 5: The tables used in the Feistel function F

index	f_α	f_β	f_γ	f_δ	f_ϵ	g	h
0	1	0	1	0	1	0	0
1	0	1	0	1	1	1	1
2	1	1	0	0	1	0	3
3	0	1	0	1	0	0	3
4	0	0	1	1	0	1	2
5	0	1	0	0	0	1	1
6	1	0	1	1	1	1	2
7	1	0	1	0	0	0	0
8	0	0	1	1	0	0	0
9	0	0	1	1	1	1	2
10	1	1	0	0	0	1	1
11	1	0	1	0	0	1	2
12	0	1	0	1	0	0	3
13	1	1	0	1	1	0	3
14	0	1	0	0	1	1	1
15	1	0	1	0	1	0	0
16	1	1	1	0			
17	0	0	0	0			
18	1	1	1	1			
19	0	0	1	1			
20	1	1	1	0			
21	1	1	0	0			
22	0	0	0	1			
23	0	0	0	1			
24	0	0	0	0			
25	1	0	0	1			
26	0	1	0	0			
27	1	1	1	1			
28	1	0	1	1			
29	1	1	1	0			
30	0	0	0	1			
31	0	1	1	0			

$$\begin{aligned}
F(k,s) &= h(g_4, g_3, g_2, g_1) \\
\\
g_1 &= g(f_{16}, f_{15}, f_{14}, f_{13}) \\
g_2 &= g(f_{12}, f_{11}, f_{10}, f_9) \\
g_3 &= g(f_8, f_7, f_6, f_5) \\
g_4 &= g(f_4, f_3, f_2, f_1) \\
\\
f_1 &= f_\delta(s_{32}, k_{32}, s_{24}, k_{24}, s_{16}) \\
f_2 &= f_\epsilon(k_{16}, s_8, k_8, k_0) \\
f_3 &= f_\beta(s_{33}, k_{33}, s_{25}, k_{25}, s_{17}) \\
f_4 &= f_\epsilon(k_{17}, s_9, k_9, k_1) \\
\\
f_5 &= f_\delta(s_{34}, k_{34}, s_{26}, k_{26}, s_{18}) \\
f_6 &= f_\gamma(k_{18}, s_{10}, k_{10}, s_2, k_2) \\
f_7 &= f_\beta(s_{35}, k_{35}, s_{27}, k_{27}, s_{19}) \\
f_8 &= f_\alpha(k_{19}, s_{11}, k_{11}, s_3, k_3) \\
\\
f_9 &= f_\delta(s_{36}, k_{36}, s_{28}, k_{28}, s_{20})) \\
f_{10} &= f_\gamma(k_{20}, s_{12}, k_{12}, s_4, k_4) \\
f_{11} &= f_\beta(s_{37}, k_{37}, s_{29}, k_{29}, s_{21}) \\
f_{12} &= f_\alpha(k_{21}, s_{13}, k_{13}, s_5, k_5) \\
\\
f_{13} &= f_\delta(s_{38}, k_{38}, s_{30}, k_{30}, s_{22}) \\
f_{14} &= f_\gamma(k_{22}, s_{14}, k_{14}, s_6, k_6) \\
f_{15} &= f_\beta(s_{39}, k_{39}, s_{31}, k_{31}, s_{23}) \\
f_{16} &= f_\alpha(k_{23}, s_{15}, k_{15}, s_7, k_7)
\end{aligned}$$