Security Analysis of an Open Car Immobilizer Protocol Stack

Stefan Tillich and Marcin Wójcik

University of Bristol, Computer Science Department, Merchant Venturers Building, Woodland Road, BS8 1UB, Bristol, UK
{tillich,wojcik}@cs.bris.ac.uk

Abstract. An increasing number of embedded security applications—which traditionally have been heavily reliant on secret and/or proprietary solutions—apply the principle of open evaluation. A recent example is the specification of an open security protocol stack for car immobilizer applications by Atmel, which has been presented at ESCAR 2010. This stack is primarily intended to be used in conjunction with automotive transponder chips of this manufacturer, but could in principle be deployed on any suitable type of transponder chip. In this paper we re-evaluate the security of this protocol stack. We were able to uncover a number of security vulnerabilities. We show that an attacker with a cheap standard reader close to such a car key can track it, lock sections of its EEPROM, and even render its immobilizer functionality completely useless. After eavesdropping on a genuine run of the authentication protocol between the car key and the car, an attacker is enabled to read and write the memory of the car key. Furthermore, we point out the threats of relay attacks and session hijacking, which require slightly more elaborate attack setups. For each of the indicated attacks we propose possible fixes and discuss their overhead.

Keywords: Security, car immobilizer, protocols, openness, analysis.

1 Introduction

Embedded security systems like car immobilizers have traditionally relied on proprietary algorithms and protocols where the specifications have been kept confidential. On the one hand, this approach can indeed complicate an attacker's job for understanding the system's specifics. On the other hand, non-public specifications limit the exposure of the employed security mechanisms to expert scrutiny. Prominent attacks on such "closed" embedded security systems like on the MIFARE Oyster card for London public transport [7] and the KeeLoq algorithm used in remote control systems [8] demonstrate the risks of this security philosophy. In contrast, open evaluation efforts like the Advanced Encryption

This is the full version of the paper. Previous versions of this paper were presented at the industrial track of ACNS 2012 [17], at INTRUST 2012 [14], as an invited paper at WESS 2012 [15], and at ESCAR 2012 [16].

Standard (AES) competition [12] and the Secure Hash Algorithm-3 (SHA-3) competition [13] are widely recognized for yielding robust solutions. Therefore, it is encouraging to see emerging open security specifications for embedded systems like the Open Source Immobilizer Protocol Stack given in [1]. Facilitated by its openness, we were able to re-evaluate its security, point out a number of security vulnerabilities, and suggest countermeasures.

A car immobilizer is a system that requires the presence of a security token (often in the form of a key fob) to allow a car to run. If this token is not present, the car's Engine Control Unit (ECU) interrupts key components like the ignition, the starter motor circuit, or the fuel pump. The communication between car and key fob is typically done via RFID, where the car is fitted with an RFID reader and the key fob contains an RFID tag. While earlier models used a static code in the key fob, modern immobilizers utilize either rolling codes or cryptography to prevent duplication of the key fob. Communication between car and key fob involves the use of a protocol stack which defines frame sizes, data formats, error detection, data transformations, etc.

Our analysis shows that an attacker can track car keys, lock sections of their EEPROM, and even permanently disrupt their immobilizer functionality with the help of a cheap standard reader and provided that she is in communication range. Eavesdropping on a genuine run of the authentication protocol between the key and the car allows an attacker to subsequently read and write memory of the key. In addition we discuss the issues of relay attacks and session hijacking.

The rest of the paper is organized as follows. The investigated protocol stack is described in Section 2. The attacks and security issues are we discovered are discussed in Section 3 and conclusions are drawn in Section 4.

2 Description of the Immobilizer Protocol Stack

An open security protocol stack for car immobilizer applications has been presented in [9]. It is mainly intended for use with specific automotive transponder chips. According to [9], the stack consists of a physical layer, a logical layer, a protocol layer, and the AES crypto layer. The physical layer deals with modulation types, data encoding, and bit timing. The logical layer defines the functional behavior of the reader and the transponder and includes communication link controls, controls configuration, setup of functional dependencies and error resolution. The protocol layer allocates data frames and buffers for reading and writing. It implements the user command interface, authentication, and key learning (*i.e.* changing cryptographic keys before and after deployment). The AES crypto layer controls the data authentication results[1]. Both physical and AES crypto layer are already industry standards. The logical and protocol layer,

[1] The description of this protocol layer in [9] probably refers to the use of the AES block cipher in the execution of various commands by reader and key fob. As such it is debatable whether it constitutes a separate layer or should be considered as part of the protocol layer.

which are usually proprietary, are made open. This means the specification of these layers is available for inspection and modification.

The protocol stack implements a number of commands to be issued by the reader to the key fob. In most cases, the car featuring the immobilizer functionality acts as reader but the reader can also be a programming device used by the car manufacturer or distributor. The communication between reader and key fob uses the LF band at 125 kHz. In this band, the normal read range is usually very limited (commonly a few centimetres), but there are readers available which can extend it to up to one metre [3, 6] and thus allowing for attacks in close proximity of the key fob.

The command set out in the protocol stack's specification [1] encompasses eleven commands. They include reading of the key fob's unique ID (UID) and error status, initiation of authentication, setting of the used secret keys, initiation and leaving of the so-called enhanced mode (for RF communication powered by the battery), a request to repeat the last response, reading and writing of user memory as well as setting memory access protection to certain memory sections. Authentication can be configured to be unilateral (only key fob authenticates itself to the reader) or "bilateral" (both key fob and reader authenticate themselves to each other) [2]. If "bilateral authentication" is configured, some commands like reading and writing user memory can only be executed when there has been a previous successful authentication.

Authentication follows the challenge-response pattern [11]. The verifying party sends out a challenge (usually a random number) and the other participant transforms the challenge cryptographically using a secret or private key and returns the response. The first party then checks this result using its knowledge of the same secret key or the according public key. The point of the challenge is to prevent replay attacks, where messages recorded from a genuine protocol run are replayed by an attacker at a later time to achieve authentication. Therefore, the challenge must be non-repeating or only repeat with negligible probability.

The investigated protocol stack has the caveat that the key fob is not expected to be able to generate challenges. This is no problem for unilateral authentication, where the challenge is generated by the reader alone, but poses difficulties for "bilateral authentication". "Bilateral authentication" works by reusing the challenge from the reader for the challenge of the key fob. The cryptographic transformation involved in the authentication is AES encryption with one of two shared keys. Figure 1 shows the steps of "bilateral authentication" as given in [1].

The car and the key fob share two AES keys (Key 1 and Key 2). The "bilateral authentication" protocol works as follows. First, the car requests the key fob's unique ID (UID) via the "ReadUID" command. The key fob reads its UID from memory and returns it. The car checks whether this UID is paired with it (as a

[2] As we will show in the following, "bilateral authentication" as set out in [1] fails to achieve the goal of authenticating both parties which is central to protocols performing bilateral (mutual) authentication. We will thus use quotes to refer to this particular part of the protocol stack.

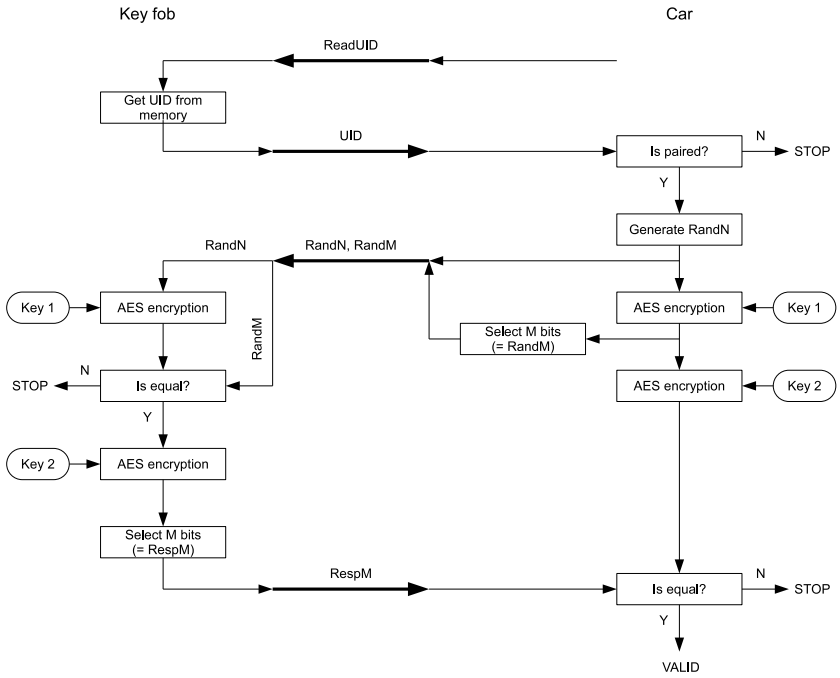

Fig. 1. "Bilateral authentication" between key fob and car.

mechanism for early termination if a foreign key fob is in communication range). The second part of the protocol consists of a "Start Authentication" command by the car and the key fob's response. The car generates the N-bit challenge RandN, encrypts it with Key 1 and selects M bits of the resulting ciphertext as RandM. N and M can be configured to be less than the AES block size of 128 bits in order to reduce communication overhead. RandN and RandM are sent to the key fob, which validates that RandM originated from RandN via encryption with Key 1. If this is successful, the car is authenticated to the key fob. The key fob uses the output of the first AES encryption as input for a second AES encryption with Key 2. As this value is not fully known to an eavesdropper (M being usually smaller than 128), it is also denoted as hidden challenge. M bits of the second encryption result are selected as RespM, which is sent to the car. The car then verifies that RespM resulted from encryption with Key 2. On success, the key fob is authenticated to the car and "bilateral authentication" is finished.

3 Security Issues

The concrete attacks are listed in the rest of this section. All attacks require a standard reader in the vicinity of the genuine key fob. The relay attack in Section 3.1 requires a second reader close to the car and connected to the first reader. The replay attack described in Section 3.4 also requires eavesdropping on

a valid authentication or separate communication with the genuine key fob and the car prior to the actual attack. The session hijacking outlined in Section 3.6 requires the presence of the attacker's reader during communication of the genuine key fob and car. Also in this case the reader must be able to selectively overpower the signal from the car.

3.1 Relay Attack with Genuine Key Fob

Unlike the other attacks described in this paper, this attack is not specific to the investigated protocol stack but can potentially be applied to any security system with token-based authentication. A relay attack tricks the car into thinking that the key fob is in its immediate vicinity when it is actually located further away, thus allowing an attacker to deactivate the immobilizer. Such relay attacks have been known as early as 1976 [2] and have been practically demonstrated, e.g. in [5] for Automotive Passive Keyless Entry and Start Systems and in [4] for the EMV chip and PIN setting. In the current setting, this attack relays messages between the genuine key fob and the car through a transparent reader (close to the genuine key fob) connected to a transparent key fob (close to the car) as shown in Figure 2. Such an attack would require two cooperating attackers, one bringing the transparent reader close to the genuine key fob and the other gaining entry to the car and bringing the transparent key fob close to the car's reader.

Fig. 2. Relay attack with transparent reader and key fob.

A potential countermeasure to this relay attack is to measure the communication delay between the reader's challenge and the key fob's response in order to detect the actual distance between the communicating endpoints. Alternatively, a dedicated protocol, like the distance bounding protocol used in [4] could be employed. However, the protocol stack includes a mechanism to defeat such countermeasures. If the transparent key fob fakes an uplink CRC error, this forces the car to send a "Repeat Last Response" command. The attacker can use the extra time for the repeated response to get the actual response from the genuine key fob.

This remote attack could be defended against with the measurement of the communication delay of the key fob by the car and by abandoning the mechanism of requesting a repeat of the the key fob's response in answer to a CRC error. Instead the whole sequence of commands and responses should be repeated when a CRC error is encountered. This gives the attacker no time to hide the extra communication delay introduced by the transparent reader and key fob. Measurement of the communication delay might require extra components (e.g. a high-precision oscillator) at the car's side.

3.2 Tracking

Generally speaking, tracking is potentially enabled through any command which can be executed without proper authentication and which returns a result both predictable by the attacker and unique to the key fob. Although tracking can be achieved by a large range of technologies (e.g. automated visual license plate recognition), tracking via electronic tokens tends to be more versatile (e.g. requires no direct line of sight), more reliable (inherent error resilience of the employed digital communication protocols) and cheaper (wide availability of standardized readers). Traceability can be seen as an enabler of a large array of undesired behaviors (ranging from mere nuisances as revealing personal habits to potential untrustworthy companies up to more sinister threats as determining which individual in a crowd has recently come by an ATM machine).

The protocol stack includes the "ReadUID" command to retrieve the 32-bit UID from the key fob. There is no security mechanism in place which would require authentication by the reader. Therefore, any reader can request the UID and the key fob can be potentially tracked by an attacker via a number of standard readers installed at various places.

The specification [1] does not state whether tracking is considered an issue to protect against. A possible argument against the feasibility of tracking could be the potential small communication range of the key fob. However, we still think it is important to point out this issue in order to caution system developers against redeploying the protocol layer on a different physical layer without considering the potential ramifications of a change in communication range.

Tracking via the "ReadUID" command could be prevented if the UID is not returned in cleartext, but dependent on a shared secret and a nonce generated by the key fob. A simple example is to use the existing AES encryption E_K with one of the pre-shared keys K in a tweakable block cipher construction $\tilde{E}_K$ [10] [3] .

$$\tilde{E}_K(\text{nonce}, \text{UID}) = E_K(\text{nonce} \oplus E_K(\text{UID})) \tag{1}$$

The result of $\tilde{E}_K$ will vary with the nonce and the UID will be protected even when the nonce is revealed. Thus, even though the key fob can be still queried by any reader, the result cannot be used any more to track it.

There are two options for the values returned by the key fob depending on the actual functional requirements. If the complete result of $\tilde{E}_K$ is returned alongside with the nonce, the reader can decrypt it and arrive at the original UID. Thus, the full functionality of the original "ReadUID" command is retained. This comes at the price of a relatively high communication overhead as the key fob needs to send the 128-bit ciphertext $\tilde{E}_K$ and the nonce. The computational overhead

[3] We have chosen a tweakable block cipher construction as example as it offers semantic security. Other constructions like $E_K(\text{nonce}|\text{UID})$ might be more efficient by eliminating the need for a separate transmission of the nonce. However, the analysis of the security of such construction is out of scope of this paper.

would essentially be the generation of the nonce and two AES encryptions on the key fob side and two AES decryptions on the reader side.

Alternatively, the reader could still check for a specific UID if only a part of the result of $\tilde{E}_K$ were returned with the nonce. This could be useful if the reader requires the "ReadUID" command exclusively to check for a specific UID. We denote this new command as "CheckUID" and its functionality is shown in Figure 3. It's advantage is a shorter response and a better response time of the key fob compared to the enhanced "ReadUID" command.

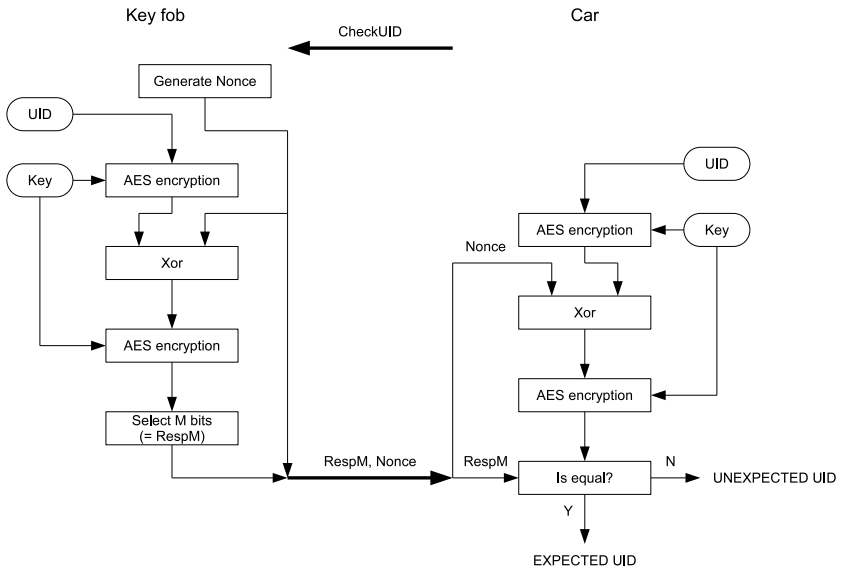

Fig. 3. Enhanced "CheckUID" command with resistance against tracking.

By varying the size of the nonce and the portion of $\tilde{E}_K$ to be checked (M-bit RespM), the security and communication overhead can be balanced. For example, using a 32-bit portion of $\tilde{E}_K$ for checking, a similar resilience against accidentally matching UIDs would be introduced as in the original protocol stack with 32-bit UIDs. The communication overhead would consist of the extra bits of the nonce and the computational overhead would be the generation of the nonce for the key fob and two extra AES encryptions for key fob and reader each. The encrypted UID ($E_K(\text{UID})$) could also be pre-computed and stored which would reduce the computational overhead by one AES encryption for each side.

In both cases, the key fob must be able to generate nonces. This might require a key fob with slightly higher capabilities as set out in the protocol stack specification. It might even be possible to include a software nonce generator (e.g. a LFSR with its state stored in non-volatile memory[4]). Note that generation of

[4] The generator may need to be cryptographically strong.

nonces is also required by one of the countermeasures to the attack described in Section 3.4. Considering the cost spent on implementation of a sound security primitive like AES, it appears logical to spend the extra effort for nonce generation in order to allow for sound protocols as well. It appears doubtful whether it is possible to defend against tracking without the ability of the key fob to generate nonces.

Other commands could also be used for tracking as well, although it would not be as versatile as using the "Read UID" command. In the following, we give a brief discussion of the use of other commands for tracking.

The command "Read Transponder Error Status" returns a status byte consisting of the last executed command and an error code for its execution. A tracking reader could induce a specific (error) state in the key fob, e.g. by sending a non-existing command. Other tracking readers could then query the key fob's state and thus track it. As there is only a limited number of possible (error) states, only a limited number of key fobs would be trackable at once. Also, any genuine errors could disturb the tracking process as key fobs would end up in different states as anticipated by the tracking readers.

The "Repeat Last Response" prompts the key fob to return the result to its last received command. As most responses contain a status byte (command code and error status), "Repeat Last Response" could be used similarly to "Read Transponder Error Status" for tracking. Also, if the last response contained a unique value (e.g. the UID in response to "Read UID" or the encrypted challenge RespM in response to "Start Authentication"), "Repeat Last Response" could be used for tracking.

Defenses against tracking via "Read Transponder Error Status" or "Repeat Last Response" could be a randomized encryption of the status byte or a limitation of these commands to authenticated readers. Also, for "Repeat Last Response", responses containing potentially trackable information should not be repeated.

The "Start Authentication" command could also be used for tracking, as its response to a fixed challenge will always be the same but will most likely differ amongst different key fobs if they use different AES keys. However, an attacker needs to know a valid challenge-response pair (RandN and RandM, cf. Figure 1), as otherwise the key fob would abort the authentication before providing a trackable response. To prevent tracking via "Start Authentication", the key fob could randomize its response to the car's challenge by including a self-generated random part to it. This would still allow for authentication by the car (provided the new random part of the challenge is also returned)[5].

3.3 Denial-of-Service Attacks

The protocol stack includes commands for writing new cryptographic keys to the key fob, which replaces the old keys used for authentication. There are two

[5] In the case of "Start Authentication", tracking could also be prevented by the adoption of one of the countermeasure against replay attacks in Section 3.4.

different modes for doing this: In open mode, a "Learn Secret Key1" or "Learn Secret Key2" can be issued by any reader in order to set new keys. In secure mode, an encrypted key is sent by the reader device, decrypted by the key fob and the result is set as new key as shown in Figure 4. The key used for encrypting the new key is the so-called Default Secret Key which is factory set [6] .

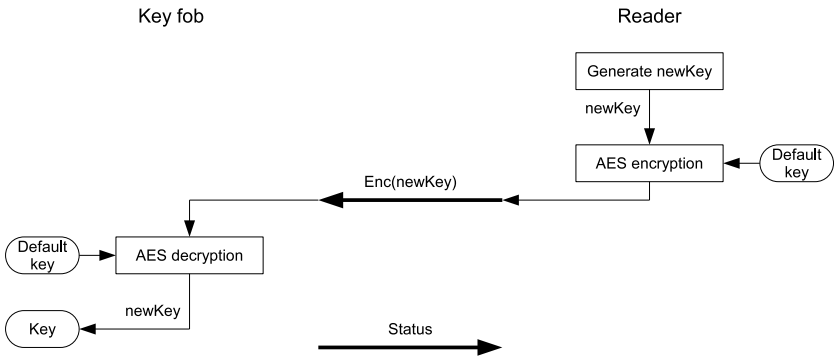

Fig. 4. "LearnSecretKey" command in secure mode.

Overwriting keys in open mode is trivial, as the malicious reader only has to send the according command to set the keys to those of her choice. However, even in secure mode it is possible to overwrite keys though the value of the new keys stays hidden to the attacker. This is possible without knowledge of the Default Secret Key because the secure key learn command only uses the encrypted key but no integrity check for it. Therefore, an attacker can send a random value as encrypted key and the key fob will set the decrypted value as new key.

Thus, in both open and secure mode, keys can be overwritten without the need of knowing a shared secret. Once this has been done, the key fob will no longer work with the car. If the key fob is queried in intervals while the car is in motion, it might even be possible to force the immobilizer to stop the car by overwriting the keys.

The open mode is vulnerable against this attack per design. To defend against the attack in secure mode, a message authentication code (MAC) should be included with the encrypted key and the key should only be overwritten when the MAC is verified successfully. This entails communication overhead for transmission of the MAC from the reader to the key fob and computational overhead of MAC generation in the reader and MAC verification in the key fob.

[6] The protocol stack specification does not indicate whether this Default Secret Key is device specific or shared by several key fobs. If this key is shared, eavesdropping on a "LearnSecretKey" command could enable an attacker to clone a key fob by replaying the recorded command to a blank key fob.

3.4 Replay Attack on Authentication

A unique property of the "bilateral authentication" protocol in the immobilizer stack is that the key fob is not required to generate nonces. Instead, the encrypted nonce from the reader is "reused" as the challenge from the key fob. While this makes the structure of the key fob simpler, it also means the commands from the reader can be recorded and replayed at a later time to achieve authentication. The attack is depicted in Figure 5. Thus, the "bilateral authentication" protocol does not provide mutual entity authentication but only entity authentication of the reader to the key fob and data origin authentication of the key fob's response to the car.

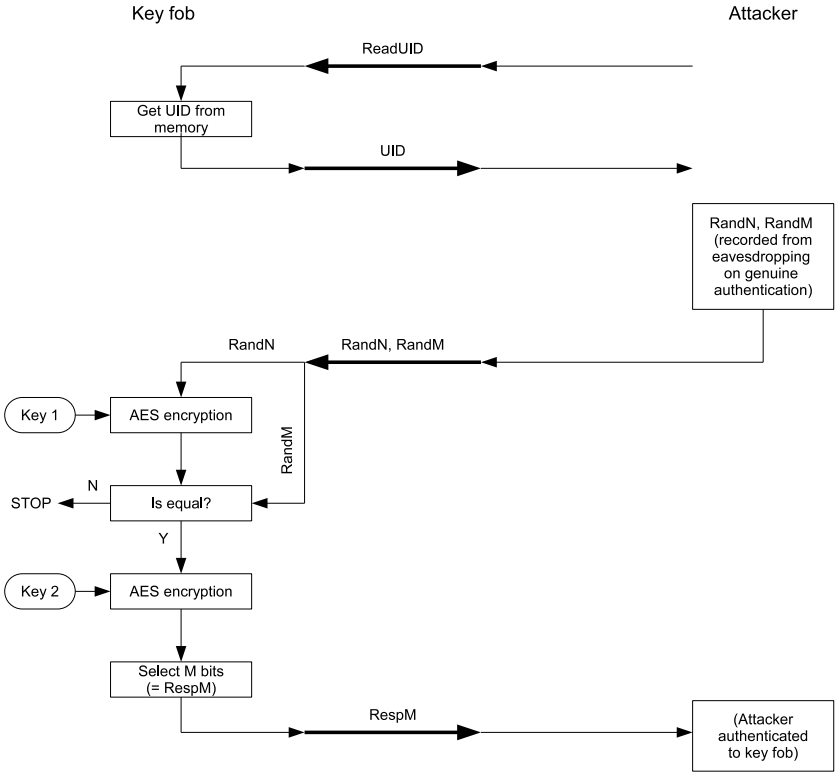

Fig. 5. Replay attack on the key fob.

In order to be able to record the appropriate reader command, the attacker can record it during a run of authentication between the genuine key fob and the car. Alternatively, the car can be prompted to issue a valid authentication challenge without the presence of the genuine key fob, if the attacker is able to send the correct UID. For example, the attacker could query the UID from the genuine key fob via the "Read UID" command (similar as in the tracking

scenario in Section 3.2). Thus an attacker can pretend to be an authenticated reader, which gives her access to advanced commands like "Read User Memory" and "Write User Memory". Note that the specification [1] does not mention the scenario of replay attacks, so it remains unclear whether the protocol designers consider protection against it as out of scope of "bilateral authentication".

A defense against this attack is to have the key fob generate the challenges for the reader. Without a challenge from the key fob, the replay of the reader command will lead to a successful authentication of the reader.

Alternatively, the reader could include an authenticated timestamp in its "Start Authentication" command to the key fob. A possible solution would be to replace the car's challenge RandN (cf. Figure 1) with the timestamp. The car then authenticates itself by providing (part of) the encryption of the timestamp (in the place of RandM). The key fob must store the timestamp of the last successful authentication and check it against the timestamp of a new authentication. If the new timestamp is not more recent than the stored one or the encryption of the timestamp is incorrect, the authentication is aborted. On successful authentication, the key fob updates the stored timestamp. This measure prevents an attacker from replaying any previously recorded challenges from the reader as none of their timestamps can be more recent than that of the last successful authentication.

3.5 Spoofing Attack on Memory Access Protection

The protocol stack allows the reader to lock the EEPROM sections AP1 to AP3 via a "Write Memory Access Protection" command. This command is accepted by the key fob without prior authentication and could be issued by any attacker with a standard reader close to it. Depending on the actual use of these EEPROM sections, an attacker could impair the functionality of the key fob by locking them with a spoofed command.

By requiring prior authentication for the "Write Memory Access Protection" command this attack can be prevented.

3.6 Hijacking Communication Sessions

Privileged commands (e.g. "Write User Memory") can only be executed if there has been a successful authentication previously (using the "Start Authentication" command). However, if a malicious reader is present during the communication session between car and key fob, it could be possible for it to remain dormant until after the successful authentication and then to "hijack" the session by overshadowing the car's communication with its own. In that way an attacker could gain access to privileged commands similar as in the replay attack described in Section 3.4. In particular, session hijacking could still occur even if the protocol was secured against the replay attack.

A possible way to prevent session hijacking would be to enhance the authentication to an authenticated key agreement, in which a session key is generated. This session key would then be used to authenticate subsequent commands (and

possibly also responses), e.g. by including a MAC over each message. An attacker without knowledge of the session key would then be unable to take over the session and could only disrupt it by jamming the communication channel. Note that such a non-persistent denial-of-service-attack[7] is a general threat which cannot be solved solely on the logical protocol layers but would have to be addressed at the physical layer.

Under the assumption that privileged commands are used infrequently, another defense against hijacking could be to authenticate each privileged command individually. A possible way to achieve this is to modify the "Start Authentication" command so that the nonce RandN is replaced by (timestamp — privileged command). While the timestamp would prevent replay attacks as described in Section 3.4, a privileged command code would also be conveyed by the reader. Upon checking the authenticity of the encrypted (timestamp — privileged command) value, the key fob could execute the privileged command and return its result alongside the usual authentication response (RespM).

4 Conclusions

In this paper we have identified a number of potential security vulnerabilities in an open car immobilizer stack. The vulnerabilities include tracking of key fobs, denial-of-service attacks to render key fobs useless, achieving key fob authentication despite absence of the key fob (relay attack), achieving reader authentication via a replay attack, and a spoof attack to lock out EEPROM sections of the key fob. For each of the identified vulnerabilities we propose countermeasures. This proves the great value of the openness of the protocol stack to public review. Some of our proposed countermeasures can be implemented rather easily while others require enhanced functionalities from the reader and/or the key fob.

Future work involves implementation of the attacks with the help of a prototyping system and formal verification of the proposed countermeasures to ensure their correctness.

Acknowledgements. The authors would like to acknowledge the help of Loïc Duflot, Shujun Li, Simon Hoerder, and the anonymous reviewers, whose comments helped to improve this paper.

The research described in this paper has been supported by EPSRC grant EP/H001689/1. The information in this document reflects only the author's views, is provided as is, and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

[7] We denote jamming of the communication channel as a non-persistent attack as it requires the presence of a disruptive device and will disappear as soon as said device is no longer close to the authentic communication parties.

References

1. Atmel. Open Source Immobilizer Protocol Stack. Available online at `http://www.atmel.com/tools/opensourceimmobilizerprotocolstack.aspx` (registration required), 2010.
2. J. H. Conway. *On Numbers and Games.* Academic Press, 1976.
3. Daily RFID Co., limited. LF RFID Reader-03. `http://www.rfid-in-china.com/2008-09-06/products_detail_2140.html`.
4. S. Drimer and S. J. Murdoch. Keep Your Enemies Close: Distance Bounding Against Smartcard Relay Attacks. In *Proceedings of the 16th USENIX Security Symposium*, pages 87–102, 2007.
5. A. Francillon, B. Danev, and S. Capkun. Relay Attacks on Passive Keyless Entry and Start Systems in Modern Cars. In *Proceedings of the 18th Annual Network & Distributed System Security Symposium (NDSS 2011)*. ISOC, 2011.
6. GAO RFID Inc. 125 kHz Long Range Reader. `http://www.gaorfid.com/index.php?main_page=product_info&products_id=363`.
7. F. D. Garcia, G. de Koning Gans, R. Muijrers, P. van Rossum, R. Verdult, R. W. Schreur, and B. Jacobs. Dismantling MIFARE Classic. In S. Jajodia and J. Lopez, editors, *13th European Symposium on Research in Computer Security (ESORICS 2008), Malaga, Spain, 6-8 October, 2008, Proceedings (to appear)*, Lecture Notes in Computer Science. Springer Verlag, 2008.
8. S. Indesteege, N. Keller, O. Dunkelman, E. Biham, and B. Preneel. A Practical Attack on KeeLoq. In N. Smart, editor, *Advances in Cryptology - EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2008.
9. P. Lepek. Configurable, Secure, Open Immobilizer Implementation. In *Proceedings of the 8th Embedded Security in Cars (ESCAR) Conference.*
10. M. Liskov, R. L. Rivest, and D. Wagner. Tweakable Block Ciphers. In *Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology*, pages 31–46. Springer, 2002.
11. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography.* Series on Discrete Mathematics and its Applications. CRC Press, 1997. ISBN 0-8493-8523-7, Available online at `http://www.cacr.math.uwaterloo.ca/hac/`.
12. National Institute of Standards and Technology. AES Competition Website (archived). `http://csrc.nist.gov/archive/aes/index.html`.
13. National Institute of Standards and Technology. SHA-3 Competition Website. `http://csrc.nist.gov/groups/ST/hash/sha-3/index.html`.
14. S. Tillich and M. Wójcik. Security Analysis of an Open Car Immobilizer Protocol Stack. In *4th International Conference on Trusted Systems (INTRUST 2012).*
15. S. Tillich and M. Wójcik. Security Analysis of an Open Car Immobilizer Protocol Stack. In *7th Workshop on Embedded Systems Security (WESS 2012).*
16. S. Tillich and M. Wójcik. Security Analysis of an Open Car Immobilizer Protocol Stack. In *10th escar Embedded Security in Cars Conference.*
17. S. Tillich and M. Wójcik. Security Analysis of an Open Car Immobilizer Protocol Stack. Presented at the industry track of the 10th International Conference on Applied Cryptograpy and Network Security (ACNS'12), June 2012.