Design and Implementation of a remote keyless entry system using state of the art bidirectional communication and authentication mechanisms

Johannes Kinzig
Spektrum Ingenieurgesellschaft mbH
johannes@spektrum-engineering.de

Abstract

Research has shown that Remote Keyless Entry systems (RKEs) often lack the required security features. RKEs from popular and widely known car manufacturers were successfully broken by attackers. The same also applies for non-automotive applications such as the Mifare Classic card which was used for electronic locking systems. This work proposes an alternative design and architecture for a remote keyless entry system solving well-known security issues like wiretapping the communication between the transceivers and extracting the cryptographic key from one of the peer's memory. The communication between the peers (key fob and lock controller) is performed bidirectional, the authentication is ensured by a challenge response mechanism. An appropriate hardware and necessary components are selected with regard to power consumption and timing constraints. Evaluating both parameters gives important information about further engineering for the prototype. Depending on the chosen authentication algorithm, availability of libraries are checked and evaluated whether they can be used in this context. Another task is the definition of a communication protocol which transfers the payload between the peers and supports the chosen authentication mechanism. A prototype is build which allows to evaluate the above mentioned parameters such as timing constraints and power consumption.

1 Introduction

Nowadays nearly every sold car comes with a RKE. This system consists of two devices, the electronic circuitry inside the car key (transponder) and the receiver inside the car. The user can lock or unlock the car remotely, from a distance up to 50 meters far away. It just takes the press of a button on the transponder and, when in reach, the car locks or unlocks the doors. This is known as an active RKE. The same mechanism exists as a passive RKE. The difference is that the user does not need to press a button on the transponder to invoke locking or unlocking. When in reach, the car locks or unlocks automatically in case a door handle is pulled. The signal, which is transmitted between the transponder and the receiver is promised to be secure against manipulation, but depending on the system, vulnerabilities and exploits exist to overcome the integrated security mechanisms to allow a severe manipulation, such as unlocking and starting the car without the proper car key. Attacks against these systems were successfully carried out by other researchers and include methods like cryptanalysis or reverse engineering. While cryptanalysis and reverse engineering also have the aim to determine the working principles of the underlying algorithms, attacks like bridging, jamming or grabbing can be carried out without the detailed knowledge of the underlying technology or cryptographic schemes.

The aim of this work is to propose a new design for RKEs using a different architecture than the systems being currently in use. The architecture will focus on the communication protocol, the physical data transmission scheme and therefore to improve the overall security of the system.

2 Related Works

Several researchers have shown that proprietary ciphers and algorithms which are used in current RKEs were successfully broken. In [6] Garcia et al. describe their reverse engineering process of the four *VW schemes* which were used in RKEs from year 2000 until 2016. All four schemes were demonstrated to be insecure and cryptanalysis revealed that common principles were violated, such as omitting proper authentication and ignoring the need for encryption when using rolling-code schemes. Instead of data encryption, signal obfuscation was used to mislead potential attackers.

Other cryptographic schemes which are used in cur-

rent RKEs are named *KEELOQ*, *Hitag2* and *Megamos*. These three schemes can also be considered broken, because researchers have developed different methods to recover the secret key or to clone the complete transponder. In [5] Eisenbarth et al. demonstrate a side channel attack to clone a transponder which uses the *KEELOQ* hopping code scheme. Another attack to recover the secret key from a *KEELOQ* transponder was performed by [4] using a slide-algebraic attack.

Hitag2 was broken by a brute-force attack by Stembera and Novotny. They describe their brute-force attack using the COPACOBANA[1] in [12]. Brute forcing became possible because the *Hitag2* algorithm allows wiretapping the communication between the transponder and the receiver which reveals the transponder serial number, initialisation vector and authenticator. With the knowledge of these parameters, the COPACOBANA can be used to generate all possible cryptographic keys and to calculate (using the initialisation vector and serial number) the corresponding authenticator. Every calculated authenticator is compared to the sniffed one, if two are equal, then the secret key is known.

Megamos was broken by performing a key-update attack described in [13]. Verdult et al. describe the cryptanalysis of the *Megamos* scheme by wiretapping the communication between the transponder and the receiver. Once the authenticator and the nonce was sniffed successfully, the transponder was manipulated by updating the registers which stored the secret key. The sniffed authenticator and nonce is then replayed to the transponder which only replies with its own authenticator if the secret key fits the replayed authenticator. The secret key gets updated inside the transponder after every unsuccessful attempt and once the transponder replies with its own authenticator, the secret key is found.

These works show that proprietary RKEs often lack necessary security features and that the architecture which is currently in use can be improved to meet higher security standards. Improving the current architecture regarding the security is the aim of this work.

3 RKEs currently in use

RKEs perform the data transmission between the peers by using radio frequency (RF) signals with a frequency of 433 MHz or 868 MHz (in Europe), some manufacturers used infrared (IR) signals instead. Cryptographic schemes were omitted, the protocol mainly consisted of sending commands to the receiver inside the car. This technique is known as ”fixed-code” transmission, because for every requested action the same command (message) is send. This makes systems vulnerable to replay attacks. Nowadays ”rolling-code” or ”hopping-code” schemes are used which overcome the ”replay-attack” vulnerabilities. [1]

The ”fixed-code” and ”rolling-code” schemes are both unidirectional which makes a receiver inside the transponder needless. Rolling-code schemes are mainly used with small, economical and low powered remote controls. The scheme is also found in garage door openers or building access control systems [5]. The plain rolling-code technique consists of a quite simple algorithm which is executed on the transponder and receiver side, it is designed to work with unidirectional data flow. This scheme is not vulnerable to replay attacks, because with every transmission a new code is sent. This is done by a sequence counter which is present in the sender and the receiver, both peers maintain their counter values independently. Additionally, both share a secret key which is used to encrypt the message at the sender side and decrypt it at the receiver side. After each transmission the sender updates its sequence counter (e.g. incrementing), once a message is send, the receiver decrypts the message and validates the received counter value against its own sequence counter. If the value is within a predefined range, the receiver accepts the message, updates its own sequence counter and performs the desired action. If the counter value is out of range, the internal counter is not updated and the action is not performed. [1]

Using unidirectional communication is still the standard for current RKEs, nevertheless some luxury cars nowadays use a bidirectional communication scheme which is technically known as a passive RKE. Transponders being part of a passive RKE additionally require a low frequency receiver which is triggered as soon as the transponder is nearby the car. A low RF signal is send by the car which is received by the transponder and deactivates the sleep mode of the microcontroller. Then a message is send out by the transponder which shows the car that the user is in range. In common scenarios this will unlock the doors. This feature is mainly introduced to save energy because letting the RKE send out connection requests and wait for a response would drain lots of energy and therefore decrease the battery live of the car key. [1]

Passive RKEs such as ”Keyless-Go” systems are vulnerable to bridging attacks. It is called *bridging* because the distance between the key and the car – which is normally limited to just a few meters – is extended by converting the signal and transmitting it over a larger distance. The adversaries just need access to the passive car key and the other adversary needs to be near the car. Getting access to the car key is quite simple, because the driver is most likely having the car key in his pocket or bag and the communication – including transmission of the wake-up signal – is carried out wirelessly. Special devices can emulate the wake-up signal and thereby force the key to initiate a communication. Both, the car key

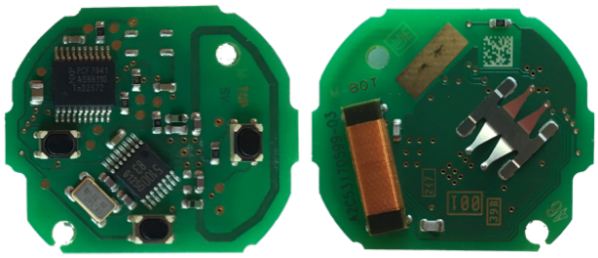

Figure 1: Transponder inside a car key, belongs to a Smart 451-2

and the receiver have no possibility to recognize that the communication is extended by a bridge and that the receiver is farer away than just a few meters.

Performing a jamming attack is effective when trying to overcome active RKEs. This is done by just interrupting the transponder's RF signals in such a way that the receiver is not able to receive the original message (for instance sending an arbitrary message on the same frequency with a higher power rating). In RKEs which implement the rolling-code scheme, this offence can be paired with a grabbing attack. In addition to jamming, the transponder's signal is recorded and later replayed by the adversary. This has the effect that the owner will not recognize a third party having access to the vehicle because unlocking will be carried out as usual. [1]

The immobilizer, which is typically placed inside a car key as well, will not be discussed separately because the security of RKEs and immobilizers rely on the same principle, the authentication mechanism. Once, two peers are mutually authenticated, the concrete use case is not critical any more, because the communication is resistant against manipulation.

Figure 1 shows a car key of a modern car, a Smart ForTwo (451-2), build in 2013. The car key features a mechanical key blade and the electronic circuitry for an active RKE (buttons for user interaction are visible). When searching for the two microchips which are soldered on the PCB it turns out that the *PCF 7941* is a Hitag2 controller[2] (manufactured by NXP) and the other one (labelled as *5100 B3 61250*) is manufactured by Infineon and is a wireless signal transmitter sold as *TDA5100*[3], operating at 868/433 MHz. This leads to the assumption that also this RKE is vulnerable to the attacks introduced in the previous sections.

4 Contribution

Literature research shows (section 2, section 3) that RKEs currently in use lack the necessary security features. Therefore, the author's contribution is the proposition of a new design and alternative architecture for a secure and reliable RKE, focussing the automotive sector and their needs, but allowing an easy adoption of the approach for other domains and industries. The focus lies on open source authentication and cryptographic algorithms, secure key storage and authenticated, bidirectional communication between the two peers. While current systems are focused on low energy consumption and therefore accept lower cryptographic schemes, the approach of this work is to use existing authentication and encryption mechanisms to increase security. It is most likely that these mechanisms demand a higher CPU performance to carry out the operations in the same short time as current RKE systems. In addition to the more complex cryptographic mechanisms, the transmission between the key fob and the receiver will also be implemented and evaluated using non-RKE-standard solutions such as Bluetooth Low Energy (BLE), Near Field Communication (NFC) or Wireless Local Area Network (WLAN) approaches. Higher CPU performance and new transmission schemes may lead to higher power consumption which could require a complete redesign of the electrical specifications and primarily the power supply. At the first glance this seems to be an disadvantage but on the other hand new transmission schemes may allow to simplify the complete system, for instance by introducing smartphones as transponders.

5 Prototype Development

This section describes the details about the prototype development including the defined requirements, proposed hardware and software architecture and explains basic thoughts about the chosen prototyping hardware. The development is lead by the aim to define system characteristics which support increased security compared to the current RKEs, low power consumption and cryptographic algorithms which have low execution times on the chosen hardware. The software implementation will be introduced shortly and will be further discussed in section 6.

5.1 Requirements

The requirements were designed by keeping functionality for the automotive industry in mind, because RKE systems are one of the most important security features to prevent unauthorised access to vehicles. Generally the proposed requirements can also be applied to RKE systems used in other contexts like building access control or similar use cases. The security and communication outside of RKEs is not part of the requirements and the corresponding implementation. Introducing vulnerabilities to a system by connecting it to other less secured

components, should be clear and will therefore not be discussed in this context.

In section 3 several scenarios were described which lead to unauthorised access to a car without requiring knowledge of the underlying algorithms and protocols. These attacks were described as *sniffing, jamming, grabbing* and *bridging*.
Sniffing a communication cannot be prevented, the only possibility is to perform a communication which leaves the adversary with data which cannot be interpreted or evaluated without the knowledge of the decryption key. This can be realised using authentication and encryption. *Jamming* and *grabbing* is also technically hard to prevent because radio signals may interfere with each other when using the same frequency. Nevertheless, the effect of jamming can be attenuated by notifying the user when his request could not be fulfilled at the receiver side. This can be performed by implementing bidirectional communication and an appropriate authentication mechanism.
Grabbing is mainly used for passive RKEs (Keyless-Go systems) which activate the transponder automatically when it is nearby the vehicle. This attack can easily be prevented by establishing a communication solely when the user physically interacts with the car key.
These requirements demand mutual authentication between both peers to increases integrity of the whole system and therefore ask for an appropriate algorithm. Challenge response authentication seems to be a reliable and fast forward way to ensure secure communication. Optionally, after authenticating a message, it can be encrypted. This is not always necessary in this case because a message which is signed by the sender will be rejected by the receiver when the message was modified during transmission. Another indispensable requirement when cryptographic schemes are used, is a secure key storage mechanism for embedded devices. Solving this requirement is also demanded by other applications (like mobile computing) and can therefore be seen as an already resolved task. Nevertheless, mechanisms will shortly be introduced.

The proposed ways of communication, authentication and key management are assumed to be computational more complex than the classical mechanisms described in section 3. Therefore a CPU with sufficient performance is necessary. Taking the possible execution time for data authentication into account leads to the requirement to focus on the timing constraints. Letting the user wait for more than one second until his ”request” is fulfilled seems inadequate regarding the comfort. Beside the timing constraints, the power consumption of the CPU is an important factor. Increased complexity regarding computation and data transmission automatically leads to a higher current drain by the CPU. This needs to be monitored to ensure an optimised alternative for current RKEs. Providing a higher level of security but disregarding comfort by reducing the transponder battery life seems inadequate. Therefore the project has to cover the following technical requirements:

1. Bidirectional communication to allow user notifications and
2. mutual authentication using a challenge response principle
3. Authenticated and (optionally) encrypted communication
4. Secure key storage
5. CPU & algorithm which support timing constraints (performance)
6. CPU, algorithms and RF components which consume low power

5.2 Proposed Architecture

The distinctiveness when engineering and designing such a RKE is the fact, that physical, electrical and computer science principles have to be used in combination to secure access to a physical resource. It is not sufficient to solely secure the resource physically (e.g. bulletproof doors and glass), the information security between the data exchange of the remote control and the local control circuits – which operate the door latch – have the same demands for security. Additionally, these techniques need to be combined into one system, which also requires to provide a secure interface between these two. For instance, having a super secure communication scheme between the transponder and the base station and having super tight doors, door latches and unbreakable glass is not providing any security when the interfaces between the electrical and mechanical parts are physically accessible and an attacker just needs to short circuit the coil which operates the latch. Nevertheless, for this work the author assumes that the physical measures were taken to properly protect the resource and that the interface between the electrical and physical system also complies to sufficient security principles.

Authentication and Cryptography. An embedded system, which performs security related tasks, should stick to the IT-security principles: *confidentiality, integrity, authentication and authorization*. These principles can be met by using cryptographic algorithms to encrypt or decrypt data, generate communication and authentication challenges or requests. The cryptographic components include functions such as symmetrical encryption/decryption, hashing, signing and random number generation.

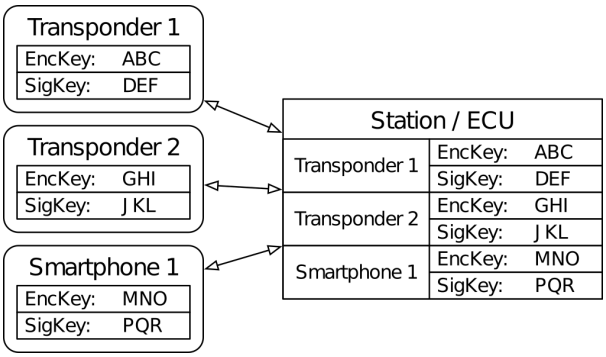

Figure 2: Transponders having own, unique signing and encryption keys

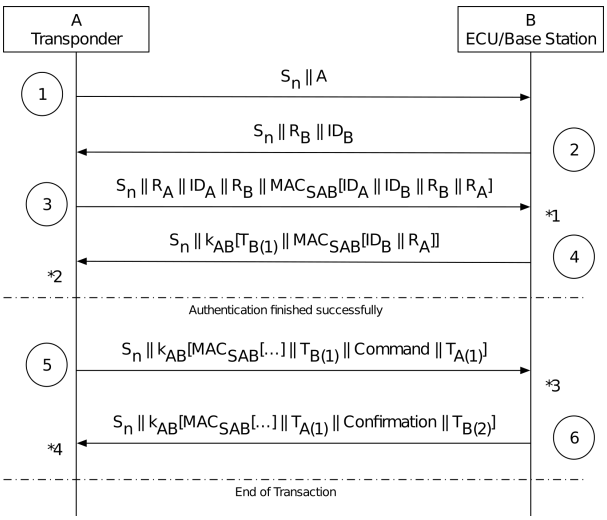

Figure 3: Authentication scheme according to EAP-PSK

The overall basis for this RKE architecture is the authentication scheme. It has to ensure a proper identification of the peers involved in data exchange. Several methods like X.509 certification and public key infrastructure (PKI) exist, but authentication in an itself closed system (closed meaning the ability to communicate to a limited number of partners only) demands other requirements. Implementing a PKI architecture is always more complex and more error prone than implementing a symmetrical cryptographic scheme. Additionally, the features a PKI architecture offers (e.g. verifying certificate validity) seems immoderate for this system and does not provide any benefit regarding security or simplicity. It is rather the opposite case, using a symmetric cryptographic scheme requires less CPU performance compared to PKI schemes. Therefore, using a challenge response authentication mechanism (CRAM) seems appropriate for this project and leads to a pre-shared key (PSK) architecture, such as EAP-PSK [2].

To ensure the above mentioned principles it is necessary to perform mutual authentication and (optional) encryption. Mutual authentication will be performed by unambiguously identifying both peers using EAP-PSK and by authenticating every message. EAP-PSK was chosen because it fulfills the above mentioned security principles by relying on a single, symmetric cryptographic algorithm, CMAC (cipher-based message authentication code) and is not limited to a specific data transmission mechanism. It can be used to transmit data securely over an insecure channel, a protocol which tracks the state of the connection between peers – like TCP/IP – is not necessary. EAP-PSK delivers its own mechanism to keep track of the ongoing connection. The communicating peers identify each other by a fixed pre-shared ID and the shared secret. Figure 2 shows the shared keys between the transponders and the station. It becomes clear that every transponder has its own, unique signing and encryption key (signing key $\neq$ encryption key), which are shared with the base station – these are the shared-secrets. Depending on the transponder's identifier, the base station knows which shared key to use for the authentication and authenticated message exchange. This requirement is not only claimed by the EAP-PSK standard, it additionally allows to easily introduce smartphones to act as transponders.

EAP-PSK was implemented for the prototype and slightly modified to be compliant with the use case, the field of RKE applications and to be used with the available data transmission schemes. Figure 3 shows the sequence diagram where the authentication and communication between both peers is shown, the sequence complies to the EAP-PSK standard (cf. [2, Fig. 9]). According to EAP-PSK, the server is the initiator of the authentication. To comply with this specification, step (1) was introduced to tell the base station (B) to initiate the authentication process. S_n is the sequence number to identify the datagrams, A is just the indicator to tell the base station to start the authentication. These fields then become concatenated which is denoted by $\|$. In (2), the base station (B) starts the authentication by sending S_n, together with the random number challenge R_B and the identity indicator ID_B. The transponder (A) then authenticates itself to B in (3) by demonstrating the ability to compute a message authentication code (denoted as $\mathrm{MAC}_{\mathrm{SAB}}[\ldots]$) over the transponder identifier ID_A, ID_B, R_B and its self generated challenge R_A using the given shared secret SAB. The MAC is sent back to B together with the generated challenge R_A, ID_A and R_B. B then validates the received MAC by calculating the same MAC over the received data and the previously defined challenge. If both MACs are identical, assumed this is the case in (*1), then A is properly authenticated to B, be-

cause it was able to prove its possession of the shared key *SAB* by responding correctly to the challenge. In (4) B now performs the same, it responds to the previously received challenge by computing a MAC over ID_B and R_A using the shared secret *SAB*. The MAC is then sent as a reply to A, together with the token $T_{B(1)}$, where A validates the received MAC by computing the MAC over its local copies of ID_B and R_A. The validation succeeds if both MACs are identical. $T_{B(1)}$ is used as a session identifier to keep the state of the authentication (comparable to RAND_S, message 3 in [2, Ch. 4.1, Fig. 9]). To additionally prove the successful authentication, B can encrypt (denoted as $k_{AB}[\dots]$) the message consisting of $T_{B(1)}$ and the computed MAC using a second shared secret, the encryption key *AB*. This can be compared to the secure channel, denoted as PCHANNEL_S_0 by the EAP-PSK standard. In (*2) B is also authenticated to A, because B could prove the possession of the shared secret *SAB* by computing the MAC over the given challenge R_A.

The authentication is finished and the transponder can continue with the payload preparation. When the user pushes a specific button on the transponder A, then a corresponding, predefined pattern is written to the *Command* buffer which is later inserted into the message in (5). A is sending the *Command* by concatenating the received $T_{B(1)}$, *Command* and a self generated token $T_{A(1)}$. $T_{B(1)}$ is part of the message to prove the state of authentication to the base station, and $T_{A(1)}$ is part of the message to allow B to prove the state of authentication to A with the following message. The transponder expects to receive $T_{A(1)}$ with the next response from B. The message, which is about to be sent in (5), is authenticated by computing a MAC over $[T_{B(1)}\|Command\|T_{A(1)}]$ and the shared key *SAB*. This MAC is placed in front of the payload and is sent to B as the datagram with the format $[MAC\|T_{B(1)}\|Command\|T_{A(1)}]$. This datagram can optionally be encrypted, in case confidential information is contained.

In (*3), B validates the MAC and validates the state of authentication by checking whether $T_{B(1)}$-received is the same as $T_{B(1)}$-send in (4). Then the *Command* (e.g. locking or unlocking the door) is carried out and a confirmation message is prepared.

In (6), a new authentication indicator $T_{B(2)}$ is generated and is send back to A together with the received $T_{A(1)}$, *Confirmation* and a MAC over $[T_{A(1)}\|Confirmation\|T_{B(2)}]$ using the shared secret *SAB*. The datagram has the format $[MAC\|T_{A(1)}\|Confirmation\|T_{B(2)}]$ and can optionally be encrypted before transmission.

In (*4) A validates the MAC and validates $T_{A(1)}$ by comparing the received $T_{A(1)}$ with the previously sent $T_{A(1)}$.

The sequence number (S_n) was chosen to be of the datatype byte, because it is just used to number the messages (same as *Flags* is doing according to EAP-PSK). A maximum of six messages is exchanged during one transaction (authentication, transmitting the command and receiving the confirmation), which suffices to use the datatype *byte* for S_n. The peer's IDs (ID_A, ID_B) were chosen to have a size of 32 bytes, the same applies for the random challenges (R_A, R_B). The benefit of using a large random number space for the challenge is, that dictionary attacks become unlikely, because the amount of messages an attacker will have to sniff is quite large ($2^{32*8} = 2^{256}$). The same applies for the size of the tokens T_x, which prove the state of the authentication and prevent replay protection. The tokens T_x are generated from the same RNG source as the random challenges R_A and R_B. Also, *Command* and *Confirmation* were chosen to have a length of 32 bytes, which allows to transmit additional information together with the actual command.

Table 1: EAP-PSK protocol and datagram sizes

Datagram	**1**	**2**	**3**	**4**	**5**	**6**	**Total**
Size (byte)	2	65	129	65	129	129	519
Sender	A	B	A	B	A	B	X
Receiver	B	A	B	A	B	A	X

The MAC requires a key of 32 bytes and is always producing an output having the size of 32 bytes. In (5) and (6) the MAC was put as the first part of the message (except for S_n) to start the cipher text with a not previously transmitted bitstring – in case encryption is used. Table 1 shows the datagram sizes which are transmitted between the transponder and the base station as demonstrated in figure 3. This becomes important when choosing the data transmission mechanism.

The peer authentication and the message authentication was performed by using the HMAC-SHA256. CMAC and HMAC (keyed-hash message authentication code) both fulfil the similar aim, authenticating a message using a symmetric cryptographic key. The difference is, that the HMAC uses a cryptographic hash function, while the CMAC uses a symmetric key block cipher [11]. Both are specified in a RFC and according to [11, Ch. 1], CMAC should be favoured on systems "in which AES is more readily available than a hash function". For the prototype, this is not the case, it is rather the opposite case – encryption was considered to be optional. This is the reasons why HMAC was favoured over CMAC for this particular case.

The circumstance, which might has caused confusion in figure 3 is that the encryption was put on top of the authenticated plaintext, even though cryptographic working modes, which ensure authenticated-encryption – such as AES-GCM or AES-EAX – exist. This was

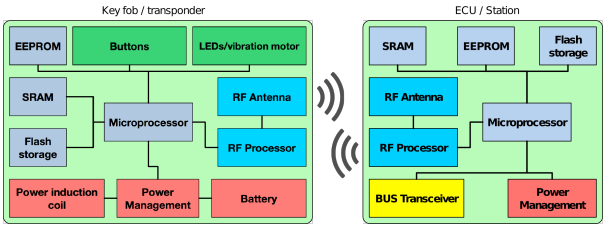

Figure 4: Proposed hardware architecture for prototype

done in view of the forthcoming timing measurements to provide a separation of authenticating a plaintext and encrypting a message. Additionally, encryption was decided to be optional, which led to the decision that just implementing an authenticated-encryption algorithm seems inappropriate, regarding the fact that ”authenticate-then-encrypt” seems to be the current practice. By first authenticating a plaintext and then encrypting the plaintext including the MAC, provides the longest running time and therefore considers the worst case scenario, which is important to judge the results correctly. Nevertheless, the author is aware of the fact, that when authentication in combination with encryption is a definite must, then choosing an authenticated-encryption working mode seems to be a better solution.

Hardware Architecture. Figure 4 shows the proposed hardware architecture of the RKE consisting of the transponder and base station. Both systems are powered by a microprocessor and have the necessary peripherals for operation such as SRAM, EEPROM, and flash storage. A power management module is taking care for powering the devices. The base station additionally has a bus transceiver which is connected to the other parts of the locking system, for instance the electronic door latch, door state sensor (open or closed) or additional devices. Both devices have wireless data transmission components such as the RF processor and the RF antenna. The RF processor is taking care for the transmission, it implements the transport layer. Tasks are data serialization, data modulation, electrical transmission depending on the chosen standard and error detection. Communication is bidirectional and the RF components are not taking care for any data securing mechanism, the input to the RF processor is a bitstream or data packets which will be send as they are. Cryptographic related tasks will be performed on the microprocessor and their related peripherals only. The SRAM is a volatile storage used for basic operations on registers. In this particular use case it can also be used for key generation using physical unclonable functions (PUF). The buttons and LEDs or the vibration motor integrated into the transponder will be used for user interaction. By pressing a button the user requests a corresponding action like locking or unlocking and the vibration motor or LEDs will confirm the requested action.

The transponder is battery powered but in case of an empty battery it is necessary to power it externally. Therefore the induction coil is included and the power management module decides which source has priority for powering the transponder.

As it became already visible in figure 4, the proposed architecture was designed to work as an ”offline architecture” meaning that the authentication and data exchange solely relies on the transponder and the base station. This architecture was chosen intentionally because connecting the base station and transponders to the internet and performing the authentication decentralised may not only introduce new attack vectors, but also makes the architecture more complex regarding overall system availability.

Wireless Data Transmission Architecture. Data transmission and retrieval will be performed using Bluetooth Low Energy (BLE), because it seems promising regarding the energy consumption, communication distance (up to 50 meters, dependent on the specific module) and availability in smartphones. This allows to introduce the smartphone as a transponder easily.

Other standards, such as WLAN (IEEE 802.11x) or ZigBee (IEEE 802.15.4) could also be used, but BLE seems more promising regarding component distribution, device availability, energy consumption and extensibility. BLE is used in smartphones and other portable devices, whereas ZigBee is only available in dedicated devices. Also, discussions about using BLE in industrial and safety-critical applications – such as vehicles – were already made by [7]. They conclude that BLE is a good choice for inter-vehicular data exchange between sensors/actuators regarding energy consumption and data rate. Lin et al. claim in [7] a current drain of 17.9 mA for their evaluated BLE device. They show that BLE outperforms ZigBee regarding the power consumption. Also, they state the maximum data rate of BLE as 1 Mbps, whereas ZigBee is limited to 250 kbps and therefore is also outperformed. According to [10], WiFi (IEEE 802.11) is more power efficient when regarding the energy per bit, but the current drain peaks (WiFi: 116 mA, BLE: 12.5 mA) prevent the compatibility with several battery types, such as CR2032. This limits the field of use drastically and is therefore not a candidate for this proposed RKE architecture.

Nevertheless, the proposed authentication architecture does not rely on BLE being used for wireless data transmission, in this particular use case it seems appropriate.

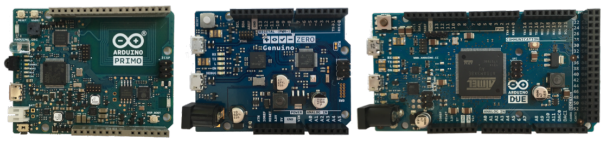

Figure 5: Arduino prototyping platform: Primo, Zero, Due

Secure Key Storage. As already described before, mechanisms to securely store keys on embedded systems already exist. This is mainly performed by cryptographic co-processors which can be integrated in the embedded system or were already integrated in the microchip by the manufacturers to increase security. These systems are used – for instance – in smartphones to store the users credentials or cryptographic keys and work reliably. Another option to perform a secure key storage without additional co-processors are called PUFs. These functions rely on deviations inside a microchip which are caused by the production process. Böhm et al. introduce in [3] a PUF based on the startup values of SRAM cells in a microcontroller. When a microcontroller is starting up the SRAM cells have either a *1* or *0* state. These startup states of every SRAM cell is majoritarian the same throughout one device but majoritarian different compared to other devices. This characteristic can be used to generate or store cryptographic keys securely on a microcontroller or an embedded system. Basically this SRAM PUF is not storing a cryptographic key, it is generating always the same key, but a unique one per device.

5.3 Implementation

The implementation is performed on three different prototyping boards with different CPUs, as seen in figure 5 and the specification seen in table 2. This gives important information about the algorithm behaviour on different platforms. The cryptographic library, which was chosen is named *mbedtls*[4]. *Mbedtls* provides functions to access symmetric and asymmetric cryptography algorithms, it is dual-licensed under GPLv2 and Apache License 2.0 and is maintained by ARM mbed[5]. The library does not have any external dependencies, the compiled binary has a size of 60 KB and requires only 64 KB RAM when executed. This makes it an ideal solution to run on a bare-metal embedded system, such as the Arduino prototyping platform.

Mbedtls provides the required algorithms such as HMAC-SHA256 and AES256-CBC which are used for message authentication and message encryption. Other encryption algorithms such as *Blowfish* can also be used, but *AES* has a good energy efficiency (cf. [8, Ch. 5.5, fig. 11]) and was proposed by Verdult and Garcia in [13, Ch. 9]. For such a block cipher an appropriate working mode is necessary to prevent statistical analysis of the cipher text. An appropriate working mode is cipher-block-chaining (CBC) and is chosen for the implementation.

In section 5.2 the EAP-PSK algorithm was introduced. To properly generate the necessary random challenges, a RNG is required. Random number generation is often an issue on bare-metal systems, because of a missing entropy source. This is why the Cortex-M series CPUs often have an on-chip RNG. This is also the case for the Arduino Primo, which has it integrated on its nRF52832 microcontroller. Additional benefit of this particular RNG is the suitability in cryptographic applications. The hardware RNG inside the nRF52832 uses internal thermal noise to generate true non-deterministic random numbers. [9, Ch. 26, p. 255].

Data transmission was realised using BLE. The nRF52832 microcontroller features an integrated BLE module which was used for this purpose. The software libraries are part of the *Arduino IDE*[6] and are used accordingly. Before transmitting a datagram over BLE, it becomes base64 encoded, which is performed using the above mentioned *mbedtls* library.

6 Prototype Evaluation

In this section the evaluation of the timing constraints and the transponder power consumption will be described, the results will be shown. As described in section 5.3, the timing constraints of the cryptographic components will be evaluated separately on different CPU architectures, which are shown in table 2. Afterwards the evaluation of the cryptographic components in combination with the EAP-PSK protocol (section 5.2) will be carried out on the prototype. This provides valuable information about the prototype's performance in contrast to the other CPUs.

The second part of the evaluation is to measure the power consumption of the transponder to evaluate the feasibility and deployment in a RKE system. This will give necessary information about the required battery capacity when used inside a car key.

6.1 Timing constraint evaluation

To determine the execution time of the software components, each component was evaluated independently from the others: *HMAC, Encryption & Decryption, base64 encoding & decoding*. The size of the input buffer for the cryptographic software components was

Table 2: Arduino prototyping platform hardware specifications

Arduino	Processor	CPU Arch.	Clock Freq.	CPU Manufacturer	Memory	SRAM
Primo	nRF52832	ARM Cortex-M4F	64 MHz	Nordic Semiconductor	512 KB	64 KB
Zero	ATSAMD21G18	ARM Cortex-M0+	48 MHz	Atmel	256 KB	32 KB
Due	AT91SAM3X8E	ARM Cortex-M3	84 Mhz	Atmel	512 KB	96 KB

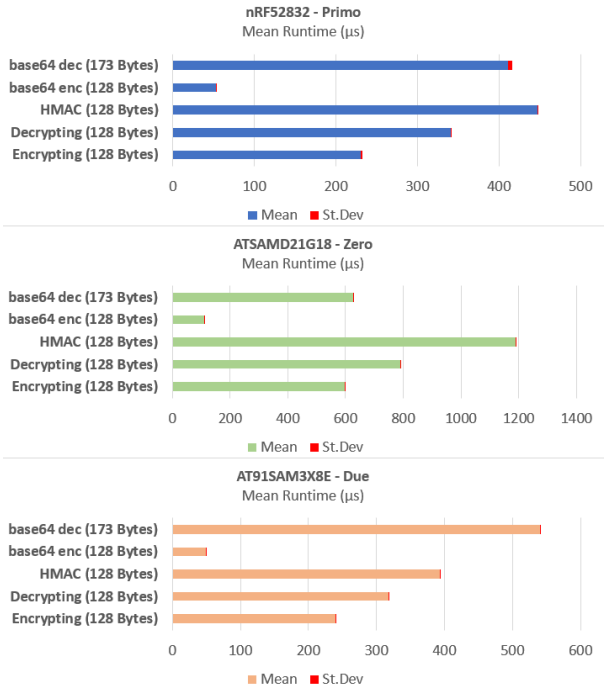

Figure 6: Timing measurements of cryptographic software components on the prototyping boards

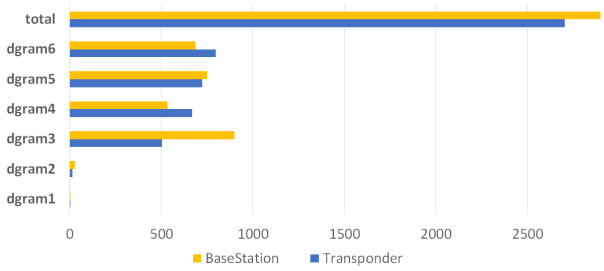

Figure 7: Prototype timing constraints by datagram in µs

determined according to the protocol described in section 5.2 and was therefore set to 128 bytes, because that is the size of the largest datagram (compare table 1). The execution time of the RNG was not measured, because it is implemented in hardware on the *Arduino Primo* and is not a software component.

In figure 6 the execution times of the components were clearly arranged and at the first glance it becomes clear that the *AT91SAM3X8E (Arduino Due)* is the fastest CPU, directly followed by the *nRF52832 (Arduino Primo)*. The operation with the shortest execution time on all CPUs is *base64 encoding*, while the longest execution time is the HMAC, but only on the *Arduino Primo* and *Arduino Zero (ATSAMD21G18)*. On the *Due*, *base64 decoding* has the longest execution time. The *Due* has the CPU with the shortest execution time of all operations, except for encryption and base64 decoding, there the *Primo* is about 9 µs and 129 µs faster. Comparing the values of the *Primo* and *Due* is quite interesting, because the CPU frequency of the *Due* is 20 MHz higher and therefore assuming the *Due* being generally faster seems logical. Therefore the difference in performance will most likely be caused by the architectural difference. When taking a closer look at the specification of the *ARM-Cortex-M4* (architecture of *nRF52832/Primo*), it turns out that this architecture is equipped with a so called DPS[7] extension, short for *digital signal processing*. According to ARM, this extension speeds up mathematical operations such as vector-dot-product, vector multiplication, etc and allows to perform signal processing directly on the microcontroller. In how far the implemented algorithms and libraries have a benefit from the DSP extension is unclear and will not further be analyzed, because it is out of scope at this state of the project. The DSP could be a reason for the similar performance of both CPUs with different clock frequencies, but is nothing more than an assumption.

After measuring the timing constraints of the single software components, the timing constraints of the prototype were analyzed. This measurement is only performed on the *Arduino Primo*, because it requires BLE communication ability and the other prototyping boards do not have this. Figure 7 shows the mean (standard deviation of 14 µs for the transponder and 52 µs for the base station) of 20 measurements for every datagram, which is transmitted between the transponder and the base station (cf. figure 3).

The execution time required by the transponder and the base station are quite similar for the corresponding datagrams, because both peers mainly perform the same operations to mutual authenticate. Datagram 1 takes equally long on both sides, because no critical operations are performed. The base station just gets an authentication request and responds to this with datagram 2.

Table 3: Bluetooth LE specification according to [10] and [9]

Data rate (theoretically)	1 Mbit/s (125 kbytes/s)
Data rate (Application)	305 kbit/s (38.13 kbytes/s)
Power drain (RX)	5.4 mA
Power drain (TX)	5.3 mA
Transmission distance	50 m

For datagram 3, the base station needs almost twice the time to validate the content than the transponder needs to prepare the content. This is caused by the implementation on the base station side. Datagram 3 is validated by checking the integrity of the message, meaning the HMAC is generated over the received data, when this check succeeds, then the actual authentication is performed, meaning the HMAC is validated using the locally stored data (R_B). This shows that the HMAC is performed twice for one datagram, which explains the longer runtime. Datagram 4 takes longer to be validated by the transponder than the base station needs to prepare it. This is caused by the fact, that the encryption is faster than the decryption. Both peers mainly behave similar for datagram 5 and datagram 6, again, here the timing difference between encryption and decryption becomes noticeable.

In total, both peers have about the same timing constraints, the base station is approximately 200 µs slower, the reasons are mainly datagram 3, with a difference of 400 µs and datagram 4, where the base station is faster and therefore compensates datagram 3.

Nevertheless, the measurement and analysis show, that the timing constraints of the software components and the chosen protocol, EAP-PSK is indeed suited for the use in an RKE system. Summing up the total time which is consumed by the base station and the transponder (transmission and datagram evaluation is of course consecutive) gives a total time of $2705\,\mu s + 2898\,\mu s = 5603\,\mu s$. This shows, that the calculations, which are performed locally on the peers to mutual authenticate just consume 5.603 ms. Again, in section 5.1 the maximum runtime for a transmission cycle (mutual authentication, payload transmission) was defined as one second. As the evaluation shows, the cryptographic operations just consume 5‰ of this defined one second. Therefore the proposed cryptographic library and protocol can be used for the architecture and implementation as intended before.

During the implementation and evaluation it turned out, that the data exchange between the transponder and the base station takes longer than demanded by the requirements in section 5.1. This can be seen in figure 10, where the timings of the transmission (channel 03) and receipt (channel 02) are shown. When the channel

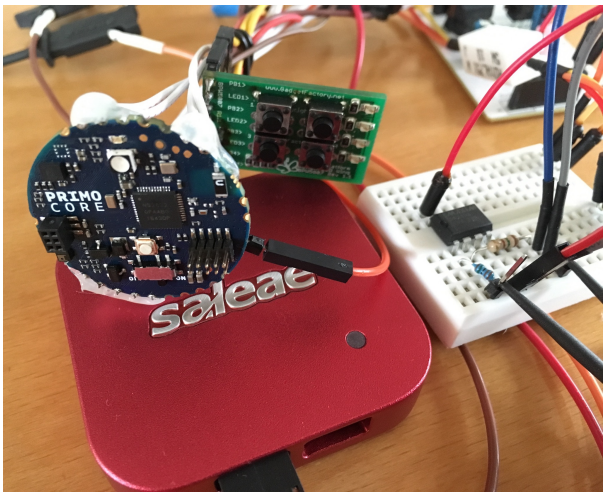

Figure 8: Experimental setup: Primo Core with keypad, amplifying circuit and logic analyzer

is pulled high, the process of transmitting or receiving is active, when the channel is pulled low, the corresponding process is inactive. From the button press on the transponder until the base station receives the *Command* (datagram 5, compare figure 3) and performs the corresponding action, it takes about 14 seconds. The transponder is transmitting and receiving 3 packets, the duration is directly dependent on the message size (compare table 1). Interestingly, the transmission from the transponder to the base station takes longer than the transmission in the opposite direction, as depicted in figure 10. Table 1 shows that both datagrams have equal sizes, but figure 10 shows that the duration of receiving datagram 6 is shorter than the the duration of transmitting the equally sized datagram 5. The reason seems to be the bluetooth library provided with the Arduino package as described in section 5.3. The slow data transmission and the asymmetric data rates indicate, that the provided bluetooth library cannot be used for data transmission.

Nevertheless, this does not mean that BLE is generally unsuited, table 3 shows, that the theoretical data throughput is about 1 Mbit/s while the application throughput is about 38 kbytes/s (compare [10, section: "Throughput"]). The data which is sent between both peers (compare table 1) is 519 bytes in total. According to this specification, the theoretical time which is consumed to transmit the 519 bytes is $\frac{519 bytes}{38000 bytes/s} = 0.0136 s$. This shows that it is possible, at least regarding the specification, to perform the authentication, send the command and receive the confirmation in far less than a second.

6.2 Power consumption evaluation

It is important to measure the transponder's power drain to judge the usability in mobile applications. The power consumption varies depending on the CPU's load when executing a binary. Therefore the power consumption evaluation will help to decide whether the implemented authentication and transmission algorithm is energy-efficient enough to be used in the proposed mobile application. Assuming this is the case, this evaluation will also help choosing a proper battery type with an appropriate capacity. To perform the measurement, the introduced *Arduino Primo* cannot be used, because it features additional components, such as a WLAN module, a power regulator and an on-board debugger, which would have distorted the power consumption and measurement. Therefore a device with less components, but the same CPU was required. A related device is the *Arduino Primo Core*, which also features the nRF52832 microcontroller but without the other components. To get ready for the measurement, the *Arduino Primo Core* was prepared to allow access to some external I/O pins, to connect a keypad (emulating the button press on the transponder) and to connect the voltage measurement equipment (depicted in figure 8). Measuring the power consumption of the base station will not be performed, because it is not a mobile device and will therefore most likely not be battery powered. Additionally, the operations performed on the base station are mainly the same as on the transponder, just the order is different. Therefore the power consumption of the base station can be supposed to be identical with the transponder.

The *Arduino Primo's* on-board microntroller (nRF52832) is sold as a "low power device". According to the key features of the nRF52832 demonstrated in the product specification [9], the electric current when running BLE applications is about 5 mA to 6 mA, when BLE is disabled, the current is even less, about 55 µA. To determine the current, it is possible to measure the potential difference over a shunt, which is connected in series to the microcontroller development board. Figure 9 shows the circuit diagram of a low-side DC current measurement circuit with a shunt, a non-inverting amplifier and the prototyping board, which is the "load". The potential difference measured over the shunt is proportional to the current which is consumed by the load. By using a 1 Ω shunt the current drain of the load can be determined by $I = \frac{U}{1}$. The non-inverting amplifier is required to amplify the voltage gathered over the shunt, because the raw voltage is too low for being measured reliably with standard tools. According to the amplification circuit in figure 9, the voltage is multiplied by factor 11, then measured by an analog-digital converter (ADC). This is important for the analysis, because

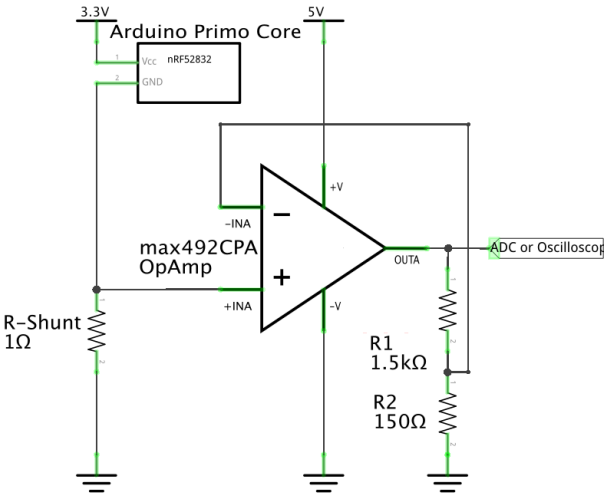

Figure 9: Low-side DC current sensing

the determined current drain has to be divided by 11 to gather the correct values.

The power drain of several transmission cycles is then recorded by a logic analyzer. An appropriate device is the *Saleae Logic 8*[8], which can record analog and digital signals up to 50 MHz. The analog input of the logic analyzer was connected to the amplifier's output (as seen in figure 9), three digital inputs of the logic analyzer were connected to digital outputs of the *Arduino Primo Core* development board. The digital channels are used to determine whether the board is transmitting or receiving data via BLE and to determine the duration of one complete transmission cycle. The sampling rate of the analog measurement was set to 125kS/s. According to Nyquist-Shannon's sampling theorem, the sampling rate has to be $2 * f_{\max}$ to record it correctly. The corresponding sampling rate was checked before using an oscilloscope and determined as $f_{\max} = 5$ kHz, consequently the sampling rate has to be greater 10 kHz. The nearest, larger value which can be set in the sampling software is 125kS/s.

Figure 10 shows the analog outputs of the measurement (channel 00), together with the transmission (channel 03) and receipt (channel 02). Obviously, the power consumption is not changing with the state of the transmission, it mainly stays constant. It does not matter whether the system is transmitting, receiving or performing a cryptographic operation (according to the protocol and the implementation, the system is performing local cryptographic tasks, when not transmitting and not receiving), the power consumption stays the same.

To give further statements about an appropriate battery type the measured current drain was integrated over the run time. The run time is given with a total of 16.89 seconds for a complete transmission cycle (compare figure 10). The integral has the size 2.11514 (product of cur-

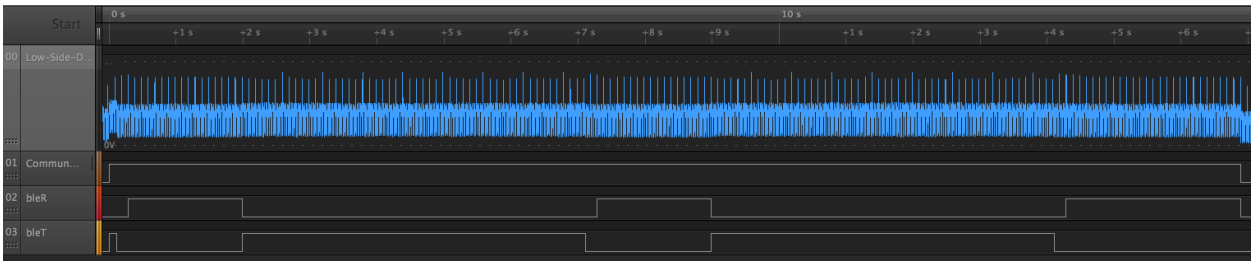

Figure 10: nRF52832: Current drain and transmission states recorded by *Logic 8*

rent drain and time consumed by one transmission cycle). According to the amplification factor of 11, the real current drain is given by $\frac{2.11514}{11} = 0.1923$. The proceeding to decide about a proper battery type is to determine, how often a user can press the button and invoke the transmission cycle until the battery is drained. To do this, the current drain of the transponder will be converted to mA: $0.1923\,\mathrm{A\,s} \widehat{=} 192.3\,\mathrm{mA\,s} \Rightarrow \frac{192.3\,\mathrm{mA\,s}}{16.89\,\mathrm{s}} = 11.39\,\mathrm{mA}$ to get the power drain of the transponder for a single transmission cycle.

Assuming a standard button-cell of type CR2032 having a capacity of 250 mA h as a reference battery, because it is often found in small remote controls or car keys. The overall capacity suffices for $\frac{250\,\mathrm{mA\,h}}{11.39\,\mathrm{mA}} = 21.95\,\mathrm{h}$, meaning the transponder with the current algorithm and BLE library implementation can be powered about 21 h in total. One transmission cycle takes 16.89 s, meaning it can be invoked about $\frac{21*60*60\,\mathrm{s}}{16.89\,\mathrm{s}} \approx 4{,}476$ times. Assuming a usage of 6 times per day (assuming 3 times opening and closing a resource) leads to 6*365 = 2,190 times per year the transponder becomes invoked by the user. According to the battery's capacity, which can invoke this transponder about 4,476 times in total, the battery would last for about $\frac{4{,}476}{2{,}190} \approx 2$ years (self-discharge and quality of battery was neglected). This is similar to current car keys and transponders.

7 Results

The evaluation shows that the proposed and implemented architecture can indeed be used as a RKE system. The evaluation of the timing constraints prove that the introduced authentication algorithm and the proposed cryptographic libraries are fast enough to deliver the required outputs to authenticate the user and transmit the necessary payload. These steps take 5.6 ms in total. The analysis also shows that the only bottleneck is the data transmission, which relies on the BLE library provided with the *Arduino Primo* package. The data transmission takes in total about 17 s, which is far to much. As already described, according to the BLE specification, the transmission of the required 519 bytes is possible in less than a second.

The power efficiency of the potential transponder then was analyzed, to determine which kind of battery should be used. The current drain of the nRF52832 CPU is about 11.39 mA and therefore a standard *CR 2032* battery lasts up to two years.

The timing constraint comparison with related CPUs demonstrates the general availability of microcontrollers suitable for the application in RKE systems. According to their physical footprint (which is definitely larger than the nRF52832's footprint, cf. figure 5) these CPU's may not be used in a transponder, but regarding their timing constraints, they could act as the base station.

The EAP-PSK protocol guarantees reliable, mutual authentication and therefore eliminates vulnerabilities which are found in current RKE systems, such as jamming, grabbing or bridging. *Jamming* will be recognized because the system notices when the authentication process becomes aborted. The *grabbing* vulnerability is also eliminated, because it works with unidirectional communication schemes only. Additional, a *bridging* attack will not be effective, because the authentication challenges and the token (proving the authentication state) are just valid for a short period of time. Enlarging the communication distance will lead to a longer signal travel time and renders the challenge and token invalid when received later than expected.

8 Conclusion

To sum it up, this work demonstrates the ability to build securer and more flexible RKE systems than currently done by the automotive industry. Authentication algorithms exist and were implemented, compatible microcontrollers were introduced and data transmission mechanisms were discussed and evaluated.

The evaluation shows, that basically all requirements were met, or, regarding BLE at least can be met according to their specification. Defining a concrete BLE implementation which meets the required timing constraints is necessary for future works. The implemented authentication mechanism should be validated further

and reviewed according to the security considerations. Additionally, extended literature research about alternative authentication mechanisms should be performed to allow a comparison between authentication algorithms.

Afterwards, a concrete application for this RKE architecture can be defined. As stated before, the use within the automotive industry and therefore the implementation in vehicles seems a good choice but the architecture can also be used in any other field where reliable and wireless authentication is an important requirement.

References

[1] ALRABADY, A. I., AND MAHMUD, S. M. Analysis of attacks against the security of keyless-entry systems for vehicles and suggestions for improved designs. *IEEE Transactions on Vehicular Technology 54* (January 2005), 41–50.

[2] BERSANI, F., AND TSCHOFENIG, H. The eap-psk protocol: A pre-shared key extensible authentication protocol (eap) method. RFC 4764, France Telecom R&D and Siemens Networks GmbH & Co KG, 2007.

[3] BÖHM, C., HOFER, M., AND PRIBYL, W. A microcontroller sram-puf. In *2011 5th International Conference on Network and System Security* (Sept 2011), pp. 269–273.

[4] COURTOIS, N. T., BARD, G. V., AND WAGNER, D. *Algebraic and Slide Attacks on KeeLoq*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 97–115.

[5] EISENBARTH, T., KASPER, T., MORADI, A., PAAR, C., SALMASIZADEH, M., AND SHALMANI, M. T. M. On the power of power analysis in the real world: A complete break of the keeloq code hopping scheme. In *In CRYPTO 2008, volume 5157 of LNCS* (2008), Springer, pp. 203–220.

[6] GARCIA, F. D., OSWALD, D., KASPER, T., AND PAVLIDÈS, P. Lock it and still lose it—on the (in)security of automotive remote keyless entry systems. *25th USENIX Security Symposium 25*, 978-1-931971-32-4 (August 2016).

[7] LIN, J.-R., TALTY, T., AND TONGUZ, O. K. On the potential of bluetooth low energy technology for vehicular applications. *IEEE Communications Magazine 53*, 1 (January 2015), 267–275.

[8] RAVI, S., RAGHUNATHAN, A., KOCHER, P., AND HATTANGADY, S. Security in embedded systems: Design challenges. *ACM Trans. Embed. Comput. Syst. 3*, 3 (Aug. 2004), 461–491.

[9] SEMICONDUCTOR, N. nrf52832 product specification, 02 2017. Available at http://infocenter.nordicsemi.com/pdf/nRF52832_PS_v1.3.pdf.

[10] SMITH, P. Comparing low-power wireless technologies. Technical report, Convergence Promotions LLC, 08 2011. Available at https://www.digikey.com/en/articles/techzone/2011/aug/comparing-low-power-wireless-technologies, last checked at 10.2017.

[11] SONG, J., POOVENDRAN, R., LEE, J., AND IWATA, T. The aes-cmac algorithm. RFC 4493, University of Washington and Samsung Electronics and Nagoya University, June 2006.

[12] TEMBERA, P., AND NOVOTNY, M. Breaking hitag2 with reconfigurable hardware. In *2011 14th Euromicro Conference on Digital System Design* (Aug 2011), pp. 558–563.

[13] VERDULT, R., GARCIA, F. D., AND EGE, B. Dismantling megamos crypto: Wirelessly lockpicking a vehicle immobilizer. *22nd USENIX Security Symposium 22* (August 2013).

Notes

[1] COPACOBANA website: http://www.copacobana.org

[2] Datasheet PCF7941: https://media.digikey.com/pdf/Data%20Sheets/NXP%20PDFs/PCF7x41ATJ.pdf

[3] Datasheet TDA5100: http://media.digikey.com/PDF/Data%20Sheets/Infineon%20PDFs/TDA5100.pdf

[4] Mbedtls website: https://tls.mbed.org

[5] ARM mbed website: https://www.mbed.com/

[6] Arduino IDE website: https://www.arduino.cc/en/Main/Software

[7] ARM DSP extension: https://developer.arm.com/technologies/dsp/dsp-for-cortex-m

[8] Logic 8 data sheet: http://downloads.saleae.com/specs/Logic+8+Data+Sheet.pdf