Breaking (and Fixing) a Widely Used Continuous Glucose Monitoring System

Luca Reverberi
School of Computer Science, University of Birmingham, UK.
luca.reverberi@socketreve.net

David Oswald
School of Computer Science, University of Birmingham, UK.
d.f.oswald@bham.ac.uk

Abstract

A Continuous Glucose Monitoring System is a medical device that continuously monitors a patient's blood glucose concentration, which is essential in the treatment of diabetes. Although such devices are increasingly used, their security has not been thoroughly studied. In this paper, we analyze a widely used wireless blood glucose monitor, the Dexcom G4. We practically demonstrate a series of security issues in this device that enable, amongst others, the tracking of a user and the forging of incorrect sensor readings. The attacks can be carried out at minimal cost using software-defined radio and low-cost RF chipsets. Finally, we devise and practically implement an efficient protocol based on best practices and well-known crypto algorithms to mitigate the weaknesses we discovered.

1 Introduction

According to statistics from the World Health Organization, approximately 430 million people worldwide are affected by diabetes [22]. This number is increasing every year, leading to serious consequences on both health and public expenditures. To keep physical wellness, a diabetic person has to constantly measure their glycemia and take appropriate countermeasures such as insulin injections. To measure the blood glucose, in the past, most diabetic patients used to prick a finger in order to take a blood sample to be used in a special device called blood glucometer, which uses an electrode containing the enzyme glucose oxidase so that glucose reacts and generates a small amount of electricity that can be read by the glucometer.

Today, technology in this area has significantly improved. Continuous Glucose Monitoring Systems (CGMs) are widely used to wirelessly measure the blood glucose and improve the therapy of diabetes. A CGM consists of a replaceable sensor placed under the skin, a transmitter connected to the sensor, and a receiver that receives and displays the measurements. The sensor can be used for several days before it needs to be replaced. During that time, it provides periodic real-time measurements, which avoids the need for fingerprick-testing glucose levels. The sensor under the skin uses the interstitial fluid to infer the glycemia and passes the value to the transmitter, which sends it to the receiver.

For transmission, CGM solutions rely on standard frequencies, for example the Industrial, Scientific, and Medical (ISM) band around 2.4 GHz. As we show in this paper, wide-spread systems do not adopt proper cryptographic measures to protect the exchanged data. In the following, we describe the results of our analysis of Dexcom G4, a well-known and widely used CGM in many western countries.

1.1 Related Work

Over the past few years, researchers have been studying the security issues of medical devices and, in particular, diabetic therapy systems. Several successful attacks on the (insufficient) security mechanisms of remotely controlled insulin pumps have been reported [1, 10, 17, 13]. The authors of [13] suggest countermeasures against the found attacks, giving implementation results on the low-cost Microcontroller (μC) MSP430 (implemented on a Spartan-6 FPGA).

Similar attacks have been reported for other medical devices, e.g. pacemakers and implantable cardiac defibrillators [11, 12, 9]. In 2016, a security analysis by the company MedSec led to significant stock market consequences [15], along with discussion about the disclosure practices adopted by MedSec and Muddy Waters.

In the context of CGM, several people have tried to develop open-source libraries for the proprietary protocol used by Dexcom G4 in order to read blood glucose values with other devices. However, just one implementation [7] documents the basics of the data exchange in

Dexcom G4. We are not aware of a systematic security analysis of the Dexcom G4 system; nor of countermeasures suitable for this use case where monitoring data flows uni-directionally from the sensor to the receiver, similar to a rolling code system in Remote Keyless Entry (RKE) [6].

1.2 Outline

The paper is structured as follows: in Section 2, we describe the process and outcomes of our analysis of the Dexcom G4, documenting the complete protocol. Then, in Section 3, we describe several attacks on the system that enable, amongst others, tracking of a diabetic subject and forging of glucose level readings. A novel protocol that protects against these threats is described in Section 4, along with practical implementation results. In Section 5 we conclude, summarising the main findings and suggesting directions for future work.

2 Analysis of the Dexcom G4 System

In this section, we describe the analysis of the Dexcom G4 and the details of the employed communication protocol. We had full access to a working set of sensor, transmitter and receiver, as well as various expired transmitters. Note that since the device only passively reports sensor readings, studying such a "live" system is less problematic than an active medical device (e.g. insulin pumps or pacemakers).

2.1 Hardware Analysis

The Dexcom G4 sensor is implanted into the patient's skin. Via exposed pads, it then connects to a transmitter that has a five-character identification code engraved on the back (see Figure 1). This code has to be used every time the transmitter is replaced when the battery is depleted. In this case, the user has to read the (new) code from the replacement transmitter and input it in the interface of the receiver. The receiver is a larger, rechargeable handheld device that displays the current glucose level (received from the transmitter) and provides a log of past readings. It can also alarm the user when the glucose level is outside a configurable range.

We mechanically removed the plastic packaging covering an old, expired transmitter. The main System on Chip (SoC) and the antenna are shown in Fig. 2. The transmitter uses a Texas Instruments CC2510 [20], which integrates an 8051 µC and a 2.4 GHz ISM transceiver. When connecting a suitable programmer to the debug pins of the µC, we found that the read-out protection bit was set. Therefore, we decided to analyze the CGM system without access to a firmware binary.

Figure 1: Back of a Dexcom G4 transmitter, with transmitter code "66DRS"

Figure 2: Dexcom G4 transmitter, with coating partially removed through mechanical grinding

Because the Dexcom transmitter is a wireless device sold in the US, it has to be tested for Federal Communications Commission (FCC) compliance, which implies, in turn, the obligation to publish specifications online [5, 3]. Furthermore, the transmitter is also covered by a patent that is publicly available [2]. Additional information is available in [7]. From these public sources, we derived details on the Radio Frequency (RF) operation of the device. The Dexcom G4 transmitter redundantly transmits on four frequencies (approx. 2.425 GHz, 2.45 GHz, 2.475 GHz, and 2.477 GHz), uses Minimum Shift Keying (MSK) modulation, features a packet length of 224 bit (including preamble), employs 286.4 kHz channel spacing, and uses a data rate of 49.987 $^{\text{kBit}}/_{\text{s}}$.

2.2 RF Eavesdropping and Transmission

For an initial analysis of the employed protocol, we decided to use a Software-Defined Radio (SDR) setup

based on the HackRF one [8] and GNU Radio [21]. Using the HackRF, we recorded the IQ data at a frequency of approx. 2.425 GHz (lowest frequency used by the device), initially with the maximum possible sample rate of 20 MHz. We found that the transmitter sends a packet periodically, with an interval of 5 min. between two packets. We also confirmed that the transmissions on the other frequencies (2.45 GHz, 2.475 GHz, and 2.477 GHz) are identical repetitions of the packet sent on the lowest frequency.

We then used the Gaussian Minimum Shift Keying (GMSK) demodulation block (which also supports plain MSK) of GNU Radio to convert the raw IQ samples to binary data. The necessary `samples_per_symbol` parameter for the GMSK demodulator can be found as `samples_per_symbol` = `samples_rate/symbol_rate`, whereas `symbol_rate` = 49.987 $^{\text{kBit}}/_{\text{s}}$, since MSK uses one symbol per bit. With these settings, we obtained the bitstream representation of the data exchanged via the RF protocol. Note that GNU Radio packs each bit into the Least Significant Bit (LSB) of a separate byte.

Subsequently, we implemented the transmission on the Wixel [16], a low-cost module (Figure 3) that is based on the Texas Instrument CC2511 (identical to the CC2510, apart from USB functionality). Using the same SoC as the Dexcom G4 transmitter helped to reduce the requirements for our attack, and also allowed for the realistic evaluation of potential countermeasures. It also facilitated the development of a mobile attack setup, cf. Section 3.6.

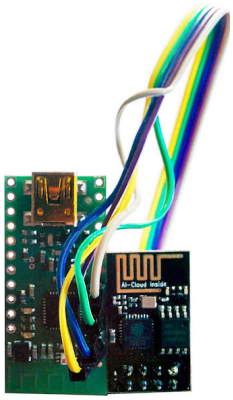


Figure 3: Wixel module with USB interface and CC2511 SoC connected to ESP8266

2.3 Dexcom G4 Protocol

The protocol of the Dexcom G4, as originally described also by [7], is unidirectional i.e. the transmitter sends measurement values to the receiver, without receiving any confirmation or control commands. In that sense, the Dexcom system is similar to a typical RKE system based on rolling codes and very different from other medical devices, where commands are usually sent *to* the implanted device (e.g. a pacemaker) to control its operation. In contrast to RKE systems, the Dexcom G4 operates autonomously, sending a packet containing the measurement values every five minutes without user interaction, resulting in a total of 288 glucose readings per day. As mentioned, the system uses multiple frequencies for redundancy, with a hopping time of 500 ms. When a new transmitter is connected to the receiver, the receiver waits for the first packets from the given transmitter, and then aligns its five-minute receiving window to this initial packet. Any other packet (even with a valid source address) during the time between two transmissions is ignored by the receiver.

SimpliciTI Although Texas Instruments does not force the users to use a specific protocol with the CC2510, they created a lightweight general-purpose protocol called *SimpliciTI* [19]. This protocol was adopted by Dexcom and used to wrap their proprietary data format into SimpliciTI packets. As shown in Figure 4, a SimpliciTI contains several fields in addition to the pure data payload. These include the destination and source addresses `dest_addr` and `src_addr`, as well as `port` and `info` to extend the capabilities of the protocol when it comes to reliability and multiplexing. `txID` is a packet counter, being incremented with each transmission, while `len` indicates the overall length of the packet in byte.

The Received Signal Strength Indicator (RSSI) and Link Quality Indicator (LQI) are not actually transmitted over the RF link, but generated during the receiving phase and provided to the program. The CRC-16 is sent by the transmitter and checked by the receiver transparently without involving user code.

The Dexcom G4 transmitter encapsulates its data payload into a SimpliciTI frame and sets the 32-bit source address in order to identify itself through the five code characters on the back side of the transmitter. Each character in the code is represented as a 5-bit value, which can represent the numbers 0–9 and the subset "ABCDEFGHJKLMNPQRSTUWXY" of the alphabet.

Overall, the five-character code hence consumes 25 bit, with the remaining seven bit of `src_addr` set to zero. Figure 5 shows the visual representation of the 25 bits mapped into characters for a transmitter with code "66DRS". While the source address is specific to a particular transmitter, the destination address has a fixed value of `0xFFFF`, which, according to the SimpliciTI specification, is the broadcast address.

Dexcom Payload The Dexcom payload, contained in the payload field of a SimpliciTI packet, has five fields

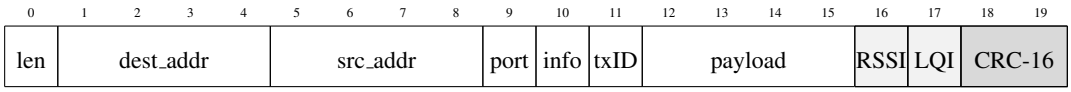


Figure 4: Packet structure of the SimpliciTI protocol, with field lengths given in byte. Preamble not shown. RSSI and LQI are implicitly added by the receiver, while the CRC-16 is checked and stripped by the RF hardware on receiving

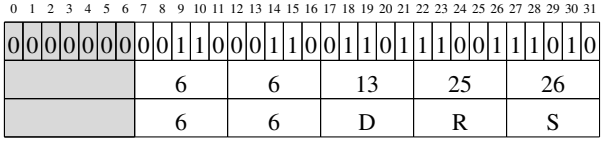


Figure 5: Mapping of the transmitter code to a SimpliciTI source address

as shown in Figure 6: two 2-byte fields (`raw` and `filtered`) for the glucose measurement value, a one-byte field `battery` indicating the charge of the transmitter's battery, a one-byte zero padding, and a one-byte checksum (see Section 2.3 for details) over the previous fields.

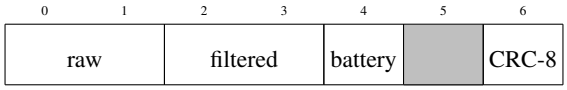


Figure 6: Payload packet of the Dexcom protocol

The `raw` field contains the actual measurement of the glucose level in the body; whereas `filtered` holds the average of multiple readings within the five minutes between each transmission. These two are floating point values, with the first 13 bit for the mantissa and the remaining three bits for the exponent (Figure 7).

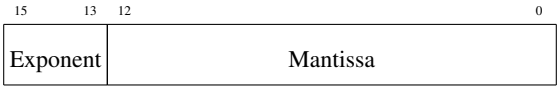


Figure 7: Bit representation of the `raw` and `filtered` 16-bit floating point fields

Recovery of the Dexcom Checksum Algorithm Since the Dexcom checksum is, in contrast to the SimpliciTI CRC-16, not documented, we attempted to recover the underlying function. As we did not have access to the transmitter's firmware, we started with the assumption that a one-byte CRC is used. We collected a number of packets subsequently used for brute-forcing all possible polynomials of a CRC-8. Additionally, we implemented a brute force algorithm considering potential full payload reflection (in which all the bits are reversed), byte-wise reflection (when the inversion is done on each byte), and differences in endianness.

Through brute-force, we confirmed the use of a standard CRC-8, using no reflection and the following polynomial:

$$x^8 + x^5 + x^3 + x^2 + x^1 + x^0$$

This value can be verified using the following two example packets that we captured. The Dexcom CRC is highlighted in green, while the part over which the CRC is computed is blue:

```
12FFFFFFFF 3A376300 3F 03 ED 38DD CA39 D5 00 4D
12FFFFFFFF 3A376300 3F 03 D3 0F1D 2199 D5 00 98
```

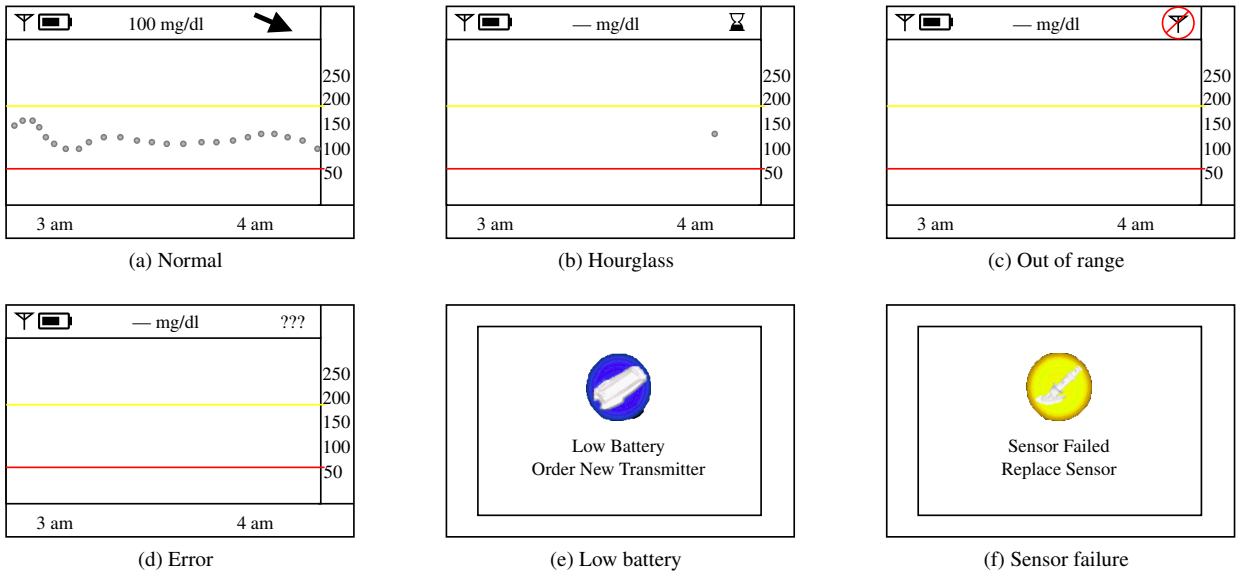
The validity and details of the computation of the CRC (for the second example packet) can be verified online at `https://www.ghsi.de/CRC/indexDetails.php?Polynom=100101111&Message=0F1D+2199+D5+00` .

Receiver Behaviour To understand the different states of the receiver, we also collected all payloads that produce a message or a change in the receiver's behaviour. The results are depicted in Figure 8.

The standard behaviour of measuring and showing the blood glucose value is shown in Figure 8a. When the transmitter is detached from the sensor or the sensor has lost its capacity to read the glucose value, the payload contains a series of zeros, and an hourglass appears on the top right corner of the display (Figure 8b). In this state the receiver forces the user to change the sensor (Figure 8f) after multiple frames with this pattern. Figure 8c represents the scenario in which the transmitter and the receiver are too far apart for communication. When the values from the transmitters appear to be random or without a clear trend (for instance, a value of 300 $^{mg}/_{dL}$ directly following 60 $^{mg}/_{dL}$), the receiver presents a triple question mark (Figure 8d). Also, in this case, after a series of packets without a stable value, the receiver forces the user to change the transmitter (Figure 8f). A value below the threshold of `0xCF` of the `battery` field produces the message in Figure 8e.

3 Attack Scenarios

As evident from the analysis in Section 2, the Dexcom G4 protocol does not employ cryptographic algorithms and therefore neither provides authenticity nor confidentiality. With the knowledge of the CRC algorithm used on the Dexcom protocol level, an adversary can easily



(a) Normal (b) Hourglass (c) Out of range

(d) Error (e) Low battery (f) Sensor failure

Figure 8: Possible states of the Dexcom receiver

generate a valid packet with freely chosen contents[1]. In this section, we describe the resulting attack vectors, together with other threats related to availability and privacy. Note that we practically performed all mentioned attacks and developed a Proof-of-Concept (PoC) for each technique.

3.1 Jamming

The first and perhaps most obvious attack involves jamming the four transmission frequencies. To carry out this attack, it is possible to use the HackRF or directly the Wixel (transmitting a continuous stream of random data). This attack does not necessarily require alignment to the five-minute transmission interval between transmitter and receiver. However, our implementation synchronizes the time window and performs frequency-hopping to jam each of the four frequencies only when necessary, making it harder to detect the jammer. It is important that the jamming is frequency-aligned with the original carrier and that the channel spacing is properly set. Due to the low transmit power of the Wixel, our implementation relies on both the HackRF and three Wixel modules. It was necessary to bypass the normal packet handling behaviour of the CC2511 and use Direct Memory Access (DMA) to generate the desired continuous data stream. Figure 9 depicts the resulting spectrum during jamming. On the side of the receiver, jamming has the effect depicted in Figure 8c. The Dexcom G4 receiver

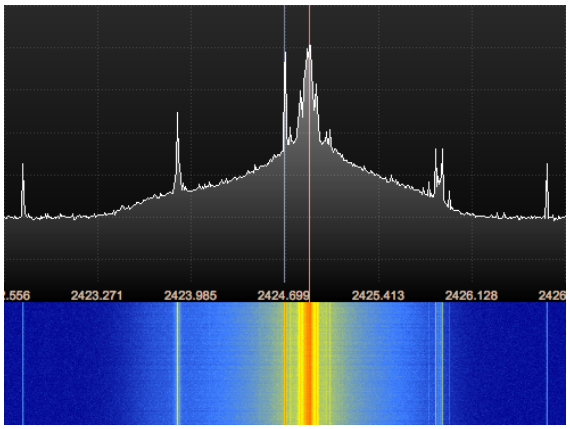


Figure 9: Spectrum during jamming of Dexcom G4

cannot distinguish whether the channels are disturbed or whether the transmitter is out of range.

3.2 Replay and Forging Incorrect Sensor Readings

Replaying old packets (and thus old sensor readings) to the receiver is not trivial because transmitter and receiver work in a synchronized time window with aligned transmission counter `txID`. Therefore, it is essential to send a correctly formed packet exactly before the original transmitter. Note that, however, adapting `txID` is possible even without knowledge of the Dexcom-specific checksum (Section 2.3), since the value is part of the SimpliciTI frame, which is only covered by the CRC-16. To

[1]Note that online brute-forcing the 8-bit checksum value would be also feasible, but rather time-consuming due to the five-minute transmission interval

align the transmission window, our PoC implementation on the Wixel receives the payload, synchronizes its internal timer to align to the window, increments the `txID`, and transmits multiple copies of the previously caught packet on the first frequency a few milliseconds before the Dexcom transmitter sends its next packet. It is sufficient to transmit only on the first frequency because, if the packet is well constructed, the receiver accepts it and just discards the other frequencies.

In a similar way, we could also generate completely new packets accepted by the receiver: after creating the desired, freely chosen contents, the Dexcom CRC is computed, and the frame is wrapped in a SimpliciTI packet with correct source address and `txID`. This packet is then transmitted aligned with the receiving window as described for the replay attack.

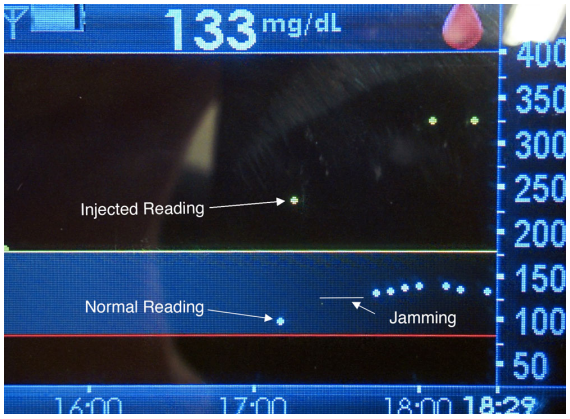


Figure 10: Forged readings on Dexcom G4

3.3 Forcing Transmitter and Sensor Replacement

Being able to generate packets from scratch means that it is feasible to force the receiver to mistakenly believe the transmitter has low battery or that the sensor does not work anymore. To achieve the former effect, it is sufficient to send a correctly formed payload with the battery field set to a value lower than `0xCF`. The latter attack requires multiple frames with inconsistent readings to convince the receiver that the sensor has failed (due to the receiver's tolerance for wrong readings). Considering the market price of approximately USD $90 for the sensor and USD $200 for the transmitter, both attacks incur in a considerable expenditure for the user.

3.4 Receiver Denial-of-Service

In addition to temporarily jamming the RF channel, it is also feasible to perform a semi-permanent Denial-of-Service (DoS) attack on the receiver. This technique is based on (slowly) misaligning the receiving time window. Since the receiver has to account for clock drift and inaccuracies in the transmitter's and its own oscillators, it has to continuously calibrate the receive window based on the prior transmissions. Injecting multiple valid packets slightly before the original transmitter causes the receiver to align its receive window with the adversary's out-of-sync packets. This procedure is illustrated in Figure 11. After a while, this leads to a complete loss of the original synchronization. In our experiments, the only way to recover from this situation was to reboot the receiver.

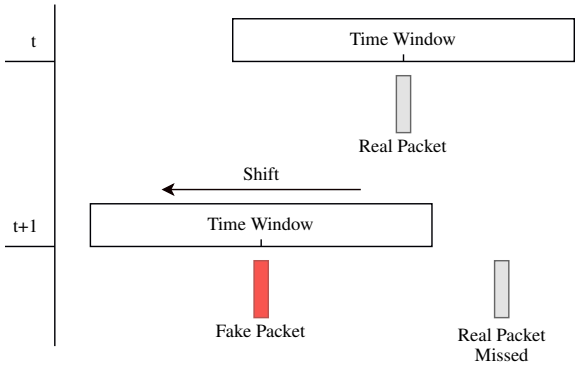


Figure 11: Misaligning the receiver's window

3.5 Tracking and Localization

Since the transmitter periodically broadcasts packets including a unique identifier without user interaction, it can be potentially used to track a specific diabetic subject. A low-cost 2.4 GHz module like the Wixel is sufficient to receive the packets from several meters. It is conceivable that higher ranges can be achieved with better RF equipment. Furthermore, by using multiple receivers, it is possible to roughly estimate the user's location based on the RSSI values. To that end, we first measured the RSSI value received by a Wixel for different distances between transmitter and Wixel, cf. Table 1. Due to the variance of the RSSI, especially when obstacles prevent the signal to pass, our model averages multiple measurements, considering the sensor on the diabetic subject frontally facing the Wixel's antenna (direct exposure for the receiver) or facing backwards (in which the body weakens the signal).

Based on the data from Table 1, we then fitted an RSSI path loss model as shown in Figure 12. This model can be used to estimate the distance between Wixel and the transmitter, based on the received RSSI value. We set up three Wixel modules at equal distance of 6 m between each other, covering the area where a target is to be localized as shown in Figure 13. The Wixels are

RSSI			
Distance (m)	Front	Back	**Average**
0.2	-37	-40	**-38.5**
0.5	-40	-38	**-44**
1	-58	-72	**-65**
2	-63.5	-69.5	**-66.5**
3	-66	-72.5	**-69.3**
4	-68	-81	**-74.5**
5	-72	-82	**-77**

Table 1: RSSI measurements for different distances and orientations

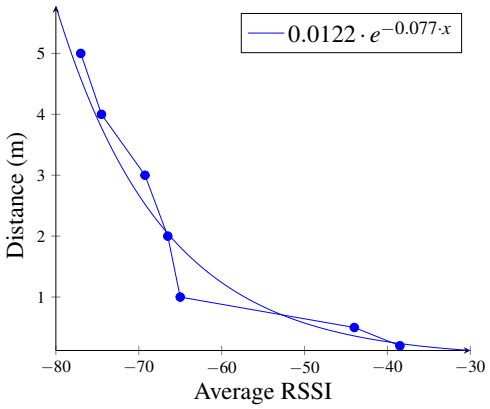


Figure 12: Fitted RSSI path loss model based on Table 1

configured to connect to a WiFi network using the well-known ESP8266 SoC [4] to configure the devices and receive RSSI measurements. A central controlling computer then aggregates the RSSI measurements and performs a trilateration algorithm (based on the min-max algorithm [18]) to obtain the position of the target. In the given (small) setting, the error in localizing the user was below 1 m. However, improvements are feasible and left for future work.

3.6 (Mobile) Hijacker and Locator

PC application We developed a PC application (using the NodeJS Electron framework) that provides a unified interface to the PoCs for the above attacks. A sketch of the application's interface is shown in Figure 14.

On the top left, the tool displays a history of readings for a selected transmitter (which is also shown in a graph on the bottom right), and allows to carry out the described attacks. The software connects to the Wixel via WiFi (or alternatively USB) and to the HackRF via USB. Amongst others, it performs the localization (Section 3.5), aggregating the RSSI values from all sources and displaying the result on the top right.

For fake packet injection, the application either supports the constant replay of an old value or the transmission of arbitrary values. For the former attack, the software simply replays an already received packet, with proper adjustment of the *txID*. In the latter attack, an arbitrary value (or trend, see Figure 15) is sent via the closest antenna (determined by the RSSI).

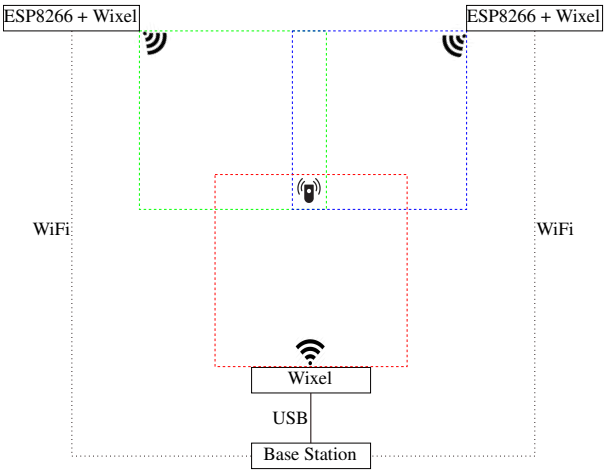


Figure 13: Localising a Dexcom G4 transmitter with three Wixel modules connected by WiFi

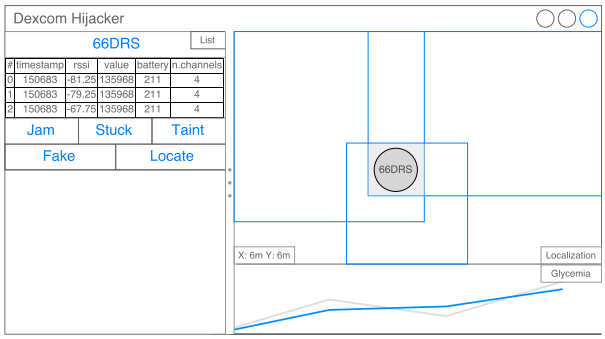


Figure 14: PC tool main screen

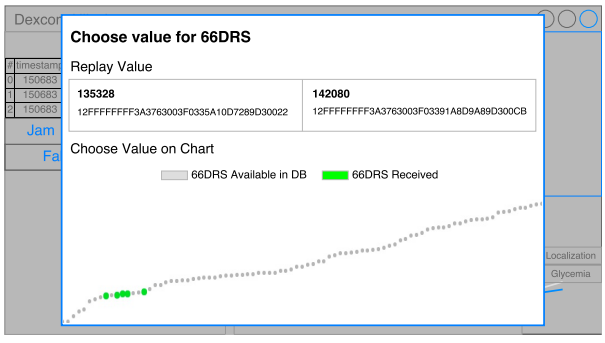


Figure 15: PC tool sending forged packets with a trend

Smartphone application In order to use the features of the PC software without the need of a computer, we developed a similar application for an Android smartphone, using the USB OTG functionality to connect a Wixel to the smartphone. Our app can record Dexcom G4 transmitter identifiers along with the geolocation. This would allow to carry out a form of "wardriving" to collect information about diabetics in a certain area. Figure 16 shows the interface of the app, with the transmitter "66DRS" detected, and a distance estimate being computed based on the RSSI path loss model from Section 3.5.

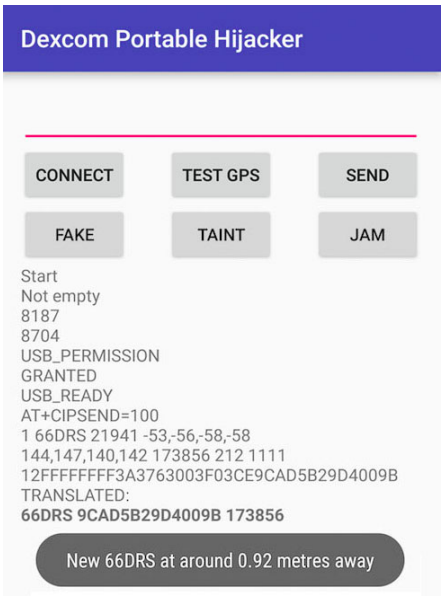


Figure 16: Smartphone app, with transmitter "66DRS" detected and a distance estimate computed

4 Improved Protocol and Countermeasures

In the literature, various countermeasures for similar systems have been discussed, see for example [13, 14]. However, these proposals usually focus on authentication (e.g. preventing replay and packet forgery) rather than also considering confidentiality (to prevent tracking). Besides, countermeasures against jamming are usually not taken into account. In this section, we briefly describe potential modifications to the Dexcom G4 protocol that protect against all these threats.

Approach To this end, we make use of the hardware AES engine provided by the CC2510/11 SoC (used in the transmitter), which executes a single-block AES-128 in 40 clock cycles (780 µs at the default clock of 26 MHz). By removing parts of the SimpliciTI frame, a packet of our protocol still fits into the 18 byte used in the Dexcom protocol. Our protocol furthermore provides:

1. encryption of privacy-relevant data (`src_addr`, glucose values) using AES-128 in Counter (CTR) mode,
2. authentication of the full frame with AES-128 as CMAC,
3. unique, random keys per transmitter, and
4. a random choice of frequencies to mitigate narrowband jamming (to an extent).

To raise the bar for jamming, the transmitter selects for frequencies at random. This prevents simple narrowband jamming without some initial profiling, since the adversary first has to determine the frequencies of interest. Obviously, it is still possible to use high-power wide-band jammers, or implement forms of more intelligent jamming, which is a fundamental problem of RF systems and therefore very hard to counter.

As mentioned, we replaced the SimpliciTI protocol with an ad-hoc protocol in order to preserve the packet length of 18 byte. Our packet structure is shown in Figure 17. The light-grey fields `src_addr`, `raw`, `filtered`, and `battery` are encrypted, while `dest_addr` and `port` are sent in clear. The final 4-byte Message Authentication Code (MAC) is computed over all previous fields, authenticating the packet in a encrypt-then-MAC configuration. Due to the fixed packet length, length extension attacks are avoided and padding oracles are of no concern (since no padding is necessary). Note that the counter (in our case 11-byte) used to prevent replay attacks is never exchanged during normal operation, but implicitly tracked by both sides and used in the CTR mode encryption. The protocol steps are described in more detail next.

Key generation All secrets used in the protocol are generated by the transmitter on its first power-on at random. Alternatively, to avoid the need for random number generations, all values could be generated offline during manufacturing, and simply be programmed into the SoC in a secure environment. The following values are generated during this phase:

$$
\begin{aligned}
K_T &= hash(ID, K_{manu}) \\
ctr &= GenerateRandom() \\
ctr_p &= GenerateRandom() \\
K_E &= GenerateRandom() \\
K_A &= GenerateRandom() \\
freq &= GenerateRandom()
\end{aligned}
$$

where ID is the code of the transmitter, K_{manu} is a manufacturer-wide key, K_T is a transport key (used during pairing), ctr and ctr_p are counters for AES-CTR, K_E is the encryption key, K_A is the MAC key, and $freq$ is

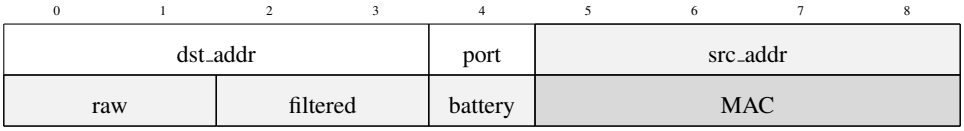


Figure 17: Encrypted and authenticated frame

a set of frequencies. The transmitter-specific transport key K_T is derived from the manufacturer key and the transmitter identifier using a hash function, e.g. SHA-256. Note that K_T is only used during setup, so even if the attacker obtains K_{manu}, he has to be present during the pairing phase (next paragraph) to obtain K_E and K_A. In the following, $E_{K,ctr}(\cdot)$ refers to AES-128 CTR encryption under key K and counter ctr, and $CMAC_K(\cdot)$ to CMAC computation under key K.

Pairing During pairing, the transmitter sends the secrets necessary for the receiver to decrypt and authenticate packets during normal operation. The transmission of these values (ctr, K_A, K_E, $freq$) is secured using the transport key K_T, cf. Figure 18. The transmitter enters pairing mode when first connected to a sensor (i.e. used by the patient) and then periodically broadcasts the encrypted data and the counter used for encryption (overall 68 byte):

$$c = E_{K_T,ctr_p}(ctr, K_A, K_E, freq)$$
$$Transmit(c, ctr_p)$$

Because the communication is one-way, the transmitter has no knowledge when the receiver has correctly received the parameters. For this reason, this phase remains active for a certain amount of time. Afterwards, the transmitter enters normal operation. On the receiver side, the user enters the identifier ID (from the back of the transmitter) and the receiver performs the key derivation to obtain K_T. Then, upon receiving a pairing packet, the keys are unwrapped and normal operation is entered.

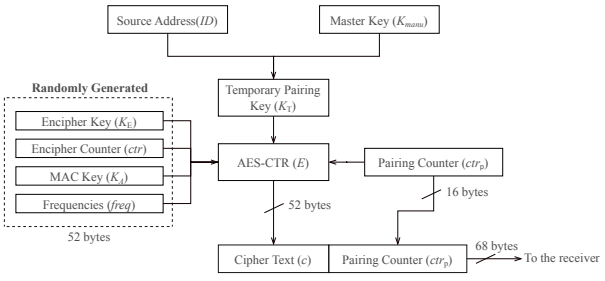


Figure 18: Overview of cryptography during pairing

Normal encryption and authentication During normal operation, the transmitter periodically sends encrypted and authenticated packets, as mentioned before:

$$c = E_{K_E,ctr}(ID, raw, filtered, battery)$$
$$MAC = CMAC_{K_A}(ctr, c, dst\_addr, port)$$
$$Transmit(c, MAC)$$

The overall design of the protocol is depicted in Figure 19. The freshness of the packets is implicitly checked by the counter used for AES-CTR encryption. For each transmitted/received packet, both transmitter and receiver increment the counter. Even if the receiver has not received a valid packet during its time window, it still increments the counter, assuming the same has happened on the transmitter's side. Hence, the protocol can cope with unreliable communication without the need for explicit re-synchronization. Furthermore, the receiver may also accept a certain range of counter values in the future.

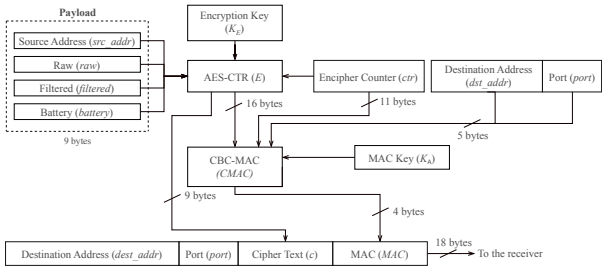


Figure 19: Encryption and authentication of normal packets

5 Summary and Future Work

In this paper, we present the results of the analysis of the Dexcom G4, a widely used CGM. We report the full inner workings of the utilized transmission protocol, including the Dexcom checksum algorithm. Based on this analysis, we demonstrate a series of attacks, allowing for forging of glucose readings, jamming of the communication channels, as well as tracking and localization of a Dexcom G4 user. Finally, we propose an improved (cryptographically secured) protocol that mitigates these threats. The improved protocol can be implemented on the CC2510 µC used in the Dexcom G4 with minimal runtime overhead, preserving the currently used packet size of 18 byte during normal operation.

This paper offers various opportunities for future research. First, our proposed protocol should be thoroughly scrutinized, possibly with a formal proof. Secondly, it would be interesting to further explore the ranges achievable for tracking and localizing user with specialized RF hardware (e.g. directional antennas, better receivers). Finally, other CGMs and similar sensing medical devices could be studied.

Responsible Disclosure

Before the publication we contacted Dexcom in September 2016 and discussed our results with them. They are exploring ways to mitigate the security issues. We have provided this paper to Dexcom. The following statement summarizes the reaction of the vendor:

"At Dexcom, our mission is to empower people to take control of their diabetes and we put our customers safety and security before anything else. More than 200,000 people with diabetes have successfully used the Dexcom CGM systems to improve their lives without a single reported incident involving cybersecurity.

Our development teams have thoroughly reviewed and discussed this paper with the authors to examine the cybersecurity vulnerabilities of the Dexcom G4 PLATINUM CGM (G4) system. We have completed an assessment and determined that the cybersecurity risk to users is extremely low.

We also note that in 2015, the next generation Dexcom G5 Mobile CGM (G5) system was introduced to the market with further cybersecurity enhancements. We continue to invest in people, technologies and systems to ensure that the security of our current and future products is continually improved.

We appreciate the authors for their dedication to science and people with diabetes and for sharing in our goal to make Dexcom CGM systems the securest available.

For inquiries, please contact Steven Pacelli, Executive Vice President, Strategy and Corporate Development, Dexcom, at 858-200-0200."

References

[1] BARNABY JACK. Insulin pump hack delivers fatal dosage over the air, 2011. https://www.theregister.co.uk/2011/10/27/fatal_insulin_pump_attack/.

[2] DEXCOM, INC. Analyte sensor, US Patent 8231531 B2, 2012. http://www.google.co.uk/patents/US8231531.

[3] DEXCOM, INC. Dexcom G4 Platinum Continous Glucose Monitoring System – User's Guide, 2015. available at http://provider.dexcom.com/sites/dexcom.com/files/cgm-education/files/LBL-012206_Rev04-Users-Guide-G4-PLATINUM-Pro-US.pdf.

[4] ESPRESSIF. ESP8266, 2017. https://espressif.com/en/products/hardware/esp8266ex/overview.

[5] FEDERAL COMMUNICATIONS COMMISSION. FCC ID PH29433 – Dexcom Inc. Glucose Monitor, 2014. https://fccid.io/PH29433.

[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. In *25th USENIX Security Symposium (USENIX Security 16)* (Austin, TX, Aug. 2016), USENIX Association.

[7] GITHUB. Wixel-xdrip, 2014. available at https://github.com/StephenBlackWasAlreadyTaken/wixel-xDrip.

[8] GREAT SCOTT GADGETS. HackRF One — an open source SDR platform. https://greatscottgadgets.com/hackrf/.

[9] HALPERIN, D., HEYDT-BENJAMIN, T. S., RANSFORD, B., CLARK, S. S., DEFEND, B., MORGAN, W., FU, K., KOHNO, T., AND MAISEL, W. H. Pacemakers and implantable cardiac defibrillators: Software radio attacks and zero-power defenses. In *Proceedings of the 29th Annual IEEE Symposium on Security and Privacy* (May 2008), pp. 129–142.

[10] LI, C., RAGHUNATHAN, A., AND JHA, N. K. Hijacking an insulin pump: Security attacks and defenses for a diabetes therapy system. In *e-Health Networking Applications and Services (Healthcom)* (June 2011), pp. 150–156.

[11] MARIE MOE. Marie moe on medical device security, 2017. https://threatpost.com/marie-moe-on-medical-device-security/123049/.

[12] MARIN, E., SINGELÉE, D., GARCIA, F. D., CHOTHIA, T., WILLEMS, R., AND PRENEEL, B. On the (in) security of the latest generation implantable cardiac defibrillators and how to secure them. In *Proceedings of the 32nd Annual Conference on Computer Security Applications* (2016), ACM, pp. 226–236.

[13] MARIN, E., SINGELÉE, D., YANG, B., VERBAUWHEDE, I., AND PRENEEL, B. On the Feasibility of Cryptography for a Wireless Insulin Pump System. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy* (New York, NY, USA, 2016), CODASPY '16, ACM, pp. 113–120.

[14] MORADI, A., AND KASPER, T. A new remote keyless entry system resistant to power analysis attacks. In *ICICS '09* (Dec 2009), pp. 1–6.

[15] MUDDY WATERS CAPITAL. Muddy Waters Capital is short St. Jude Medical, Inc. (STJ US), 2016. http://d.muddywatersresearch.com/.../MW_STJ_08252016_2.pdf.

[16] POLOLU CORPORATION. Wixel Programmable USB Wireless Module, 2017. https://www.pololu.com/product/1337.

[17] RADCLIFFE, J. Hacking medical devices for fun and insulin: Breaking the human SCADA system. In *BlackHat '11* (2011).

[18] SAVVIDES, A., PARK, H., AND SRIVASTAVA, M. B. The Bits and Flops of the N-hop Multilateration Primitive for Node Localization Problems. In *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications* (New York, NY, USA, 2002), WSNA '02, ACM, pp. 112–121.

[19] TEXAS INSTRUMENTS. Introduction to SimpliciTI, 2008. http://www.ti.com/lit/ml/swru130b/swru130b.pdf.

[20] TEXAS INSTRUMENTS. CC2510/2511 Datasheet, 2012. available at http://www.ti.com/lit/ds/symlink/cc2511.pdf.

[21] THE GNU RADIO FOUNDATION, INC. GNU Radio — The free & open software radio ecosystem, 2017. https://www.gnuradio.org/.

[22] WORLD HEALTH ORGANISATION. Global report on diabetes, 2016. http://apps.who.int/iris/bitstream/10665/204874/1/WHO_NMH_NVI_16.3_eng.pdf?ua=1.