RollBack- A New Time-Agnostic Replay Attack Against the Automotive Remote Keyless Entry Systems

Levente Csikor*[1,2], Hoon Wei Lim[2], Jun Wen Wong†[2,3], Soundarya Ramesh[4], Rohini Poolat Parameswarath[4], and Mun Choon Chan[4]

[1]*Institute for Infocomm Research (I²R), A*STAR, Singapore*
[2]*NCS Group, Singapore*
[3]*DSBJ Pte. Ltd., Singapore*
[4]*National University of Singapore*

Abstract

Automotive Remote Keyless Entry (RKE) systems implement disposable rolling codes, making every key fob button press unique, effectively preventing simple replay attacks. However, RollJam was proven to break all rolling code-based systems in general. By a careful sequence of signal jamming, capturing, and replaying, an attacker can become aware of the subsequent valid unlock signal[1] that has not been used yet. RollJam, however, requires continuous deployment indefinitely until it is exploited. Otherwise, the captured signals become invalid if the key fob is used again without RollJam in place.

We introduce RollBack, a new replay-and-resynchronize attack against most of today's RKE systems. In particular, we show that even though the one-time code becomes invalid in rolling code systems, replaying a few previously captured signals consecutively can trigger a rollback-like mechanism in the RKE system. Put differently, the rolling codes become resynchronized back to a previous code used in the past from where all subsequent yet already used signals work again. Moreover, the victim can still use the key fob without noticing any difference before and after the attack.

Unlike RollJam, RollBack does not necessitate jamming at all. Furthermore, it requires signal capturing only once and can be exploited any time in the future as many times as desired. This time-agnostic property is particularly attractive to attackers, especially in car-sharing/renting scenarios where accessing the key fob is straightforward. However, while RollJam defeats virtually any rolling code-based system, vehicles might have additional anti-theft measures against malfunctioning key fobs, hence against RollBack. Our ongoing analysis (covering Asian vehicle manufacturers for the time being) against different vehicle makes and models using RKE implementations from NXP revealed that more than 80% of them are vulnerable to RollBack.

Keywords: remote keyless entry, rolling code, vulnerability, replay attack, RollJam, `RollBack`, resynchronization

*Levente Csikor was with NCS Group when this research work started.

†Jun Wen Wong was with NCS Group during this work.

[1] In this paper, the terms *code* and *signal* are used interchangeably.

1 Introduction

The automotive industry has undergone a tremendous evolution since the first car was made more than a century ago. While the efficiency and versatility have been continuously evolving, since the early 1980s, manufacturers have constantly been squeezing more and more embedded computers, known as Electronic Control Units (ECUs), into our cars to enhance safety [1], stability [2], diagnostics [3], and comfort [4, 5], to name a few [6]. On the one hand, this paradigm shift from the traditional mechanical mechanisms to an all-digital control has been clearly proven beneficial. On the other hand, computerized vehicles open up a broad set of new attack surfaces [7–14].

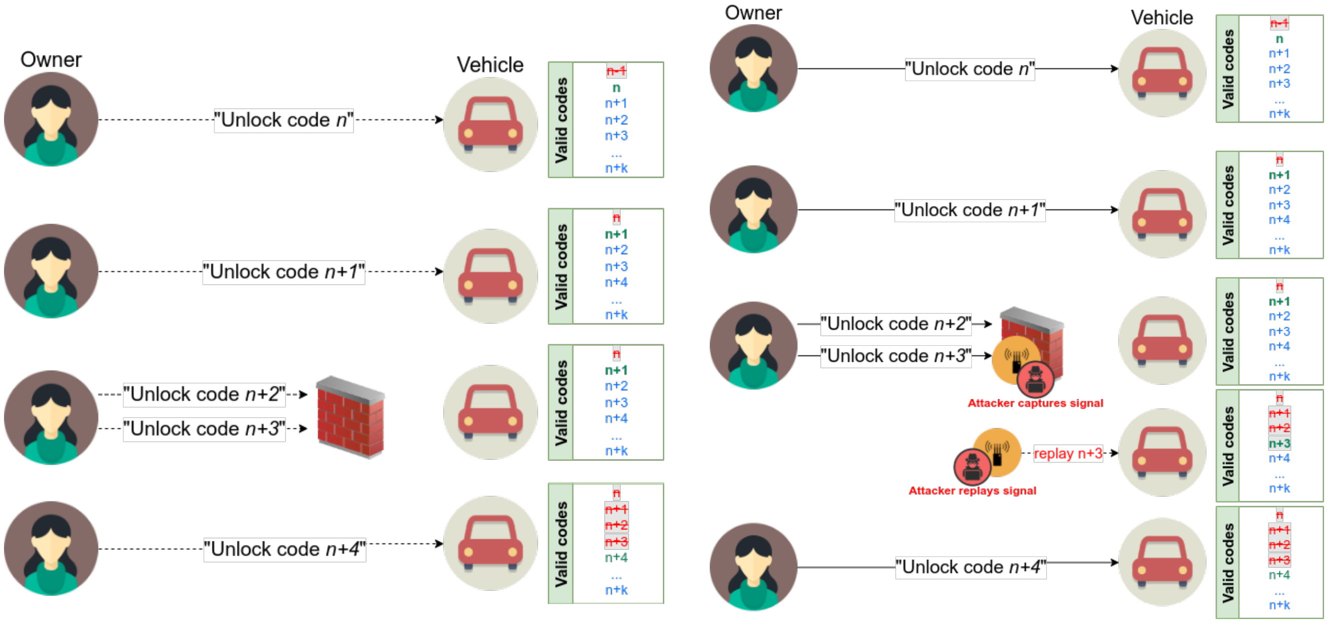
One of the earliest comfort-enhancing inventions is the *Remote Keyless Entry (RKE)* system that eliminates the need for physical keys and allows one to remotely lock and unlock the vehicle[2] merely by using a key fob. Since RKE is already present in commercial vehicles from the early 1980s [5], it has been (and still is) one of the main targets of the attackers [10, 11, 14–16]. RKE systems use wireless radio signals, and due to the limited number of required commands (e.g., lock, unlock) and, most importantly, the power and resource constraints of the small battery-operated key fobs, the communication between the key fob and the vehicle is designed to be simple. Some deployments may use encryption to avoid eavesdropping (i.e., capture and decode signals) or tampering attacks (i.e., "flipping" lock signals to unlocks); however, replaying signals, even if they are encrypted, is straightforward. Today, many RKE systems still implement static codes to control the vehicle from the key fob. Therefore, capturing an encrypted "unlock" signal allows an attacker to replay it and access the vehicle anytime afterward.

To cope with these simple replay attacks, *rolling codes*, i.e., code hopping [17], have been introduced wherein a particular code (e.g., an "unlock" code) is considered disposable, i.e., it is only used once. In a nutshell, every button click on the key fob triggers a counter in the key fob and in the vehicle upon reception to roll, making it valid for subsequent use in the future. Put differently, sent codes that are *used once* are invalidated by the next code, effectively preventing replay attacks (cf. Fig. 1a).

Note that a sent code can also be considered *unused* when the key fob has emitted the signal, but the vehicle did not receive it. For instance, when the unlock button was accidentally pressed (i.e., in our pocket, or when our toddler plays with the key fob) outside of the vehicle's vicinity (depicted by "unlock code n+2" and "n+3" in Fig. 1a). To avoid getting out-of-sync and hence locking ourselves out of our vehicle in such cases, rolling code-based systems provide a safety feature that allows the key fob's counter to be be steps ahead compared to the vehicle's counter. This is achieved by having *not one but a set of valid "future codes"* maintained at the vehicle. If the received code from the key fob matches any of these future codes, the vehicle resynchronizes to the code in the last key fob signal, and invalidates all previous (but unused) ones from this set (refer to "Unlock code n+4" in Fig. 1a). Clearly, if an attacker could obtain one of these unused future codes (i.e., capture the signals of the accidental button presses outside of the vicinity of the car), and she can replay it before the owner uses the key fob again, the attacker can get access to the vehicle (cf. Fig. 1b). However, obtaining these future codes are extremely difficult in practice, especially if an attacker wants to target a random victim. That is the reason why this safety provisioning is considered a handy feature that makes the key fob use seamless and less troublesome.

In 2015, a sophisticated attack technique called RollJam [16] has proven the rolling code-based key fob systems to be breakable. In a nutshell, by using a careful sequence of signal jamming, capturing, and replaying, RollJam can effectively convert this safety provisioning feature into an exploit.

[2] In newer models, a key fob can also be used to turn on and off the anti-theft alarms, or even start and stop the engine.



(a) Essence of rolling codes: every signal is unique and gets invalidated by the next one.

(b) "Straightforward exploit" of the safety feature in rolling code-based systems

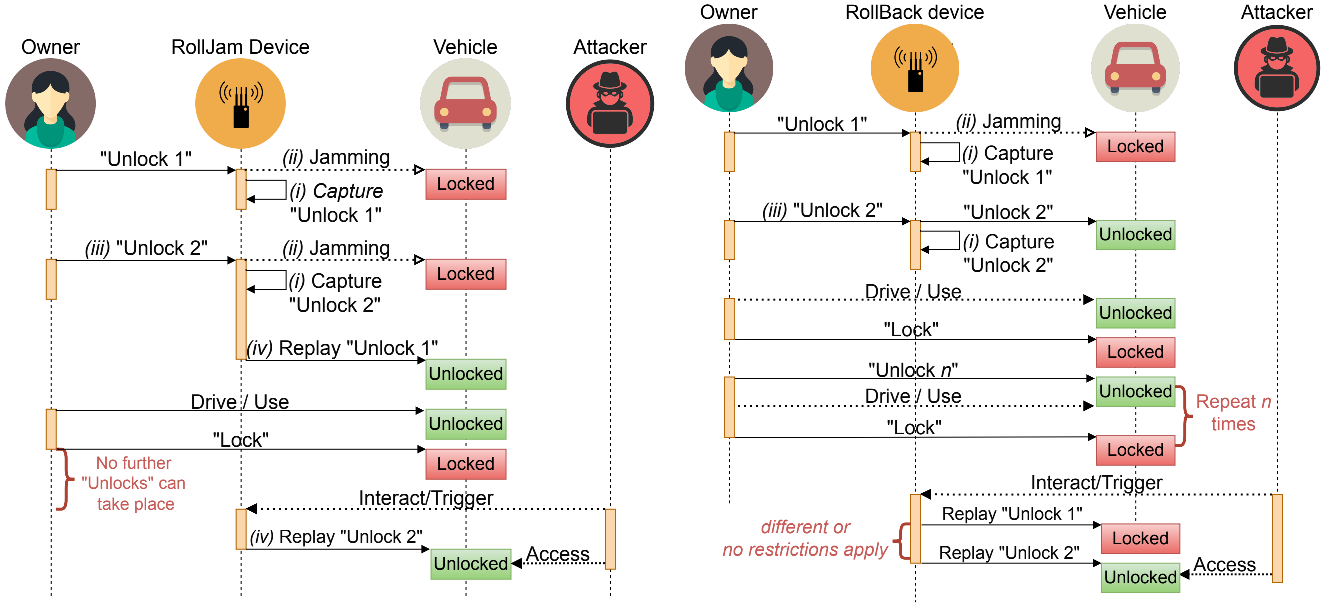
Fig. 1: Rolling code technology in a nutshell, and its safety feature exploited.

RollJam is based on four main "principles", *(i)* capturing unlock signals, *(ii)* jamming the frequency band towards the vehicle at the same time to hinder correct signal reception, *(iii)* the owner's second trial as a fail-over mechanism, and most importantly, *(iv)* timely replay of previously captured signals. To this end, a special-purpose device (hereafter, rolljam device) is used as a man-in-the-middle proxy and a signal jammer between the key fob and the vehicle (cf. Fig. 2a). Briefly, the victim is lured to *(iii)* press the unlock button in a key fob twice by *(ii)* jamming the first unlock signal. At the same time, both first and second unlock signals are *(i)* captured; however, when the second signal is jammed, the rolljam device quickly *(iv)* replays the one captured the first time. As a result, the vehicle acts as intended, i.e., unlocks, and the victim assumes that the signal reception was lousy on the first try. On the other hand, the attacker (i.e., by the rolljam device) becomes aware of the following valid unlock signal (see more details in §2.3). Therefore, once the owner stops using the vehicle and leaves it unattended, the attacker can replay this signal to access the vehicle.

RollJam, however, has two main drawbacks. First, suppose the owner unlocks the vehicle again *without* the rolljam device in action. In this case, the rolling code in the RKE system advances, invalidating all previous codes, including the one possessed by the attacker. Consequently, properly suffixing the rolljam device at a hidden spot of the vehicle and replaying the *valid* unlock signal in a timely manner, i.e., step *(iv)*, are the keys to the success of RollJam. Second, similarly to the above, if the attacker succeeds in using the captured valid yet unused signal, she cannot use it again; to repeat unlocking the same vehicle in the future, the whole attack must be redone from scratch.

In this paper, we present ***RollBack***, a new time-agnostic replay-and-resynchronize attack. Even though a one-time code becomes invalid in rolling code-based systems, replaying a few previously captured (consecutive) signals can trigger a rollback-like mechanism in most RKE systems, making all former captured (unlock) signals valid again; hence the name `RollBack`[3]. At

[3] Rollback is a process in database management that involves canceling a (set of) transaction(s) to bring the database to its previous state before those particular transactions would have been performed.



(a) RollJam is particularly sensitive to timing; it has to be aware of the next valid unused code.

(b) A `RollBack` variant using only two captured signals at any time.

Fig. 2: Differences between RollJam and `RollBack`.

the same time, the rollback-like mechanism involves the execution of the instruction encoded in the signals, e.g., unlocking the vehicle.

Consequently, unlike RollJam, `RollBack` does not have to keep track of the latest valid yet unused code continuously. In other words, we do not need the long step-sequence $(i) \rightarrow (ii) \rightarrow (iii) \rightarrow (i) \rightarrow (ii) \rightarrow (iv)$ to be repeated, and additionally (iv), every time to eventually access the vehicle (cf. Fig. 2). In general, `RollBack` does not need step *(iv)* at all, and only requires steps $(i) \rightarrow (ii) \rightarrow (iii) \rightarrow (i)$ once; then, replaying the captured signals can unlock the victim's vehicle *any time in the future* and *as many times as desired*. This makes `RollBack` more flexible and time-agnostic, significantly reducing the complexity and the efforts needed by an attacker.

In fact, even jamming the first signal *(ii)* is only required by `RollBack` to obtain the signals in a relatively short time frame. Put differently, due to the time-agnostic feature of `RollBack`, it does not matter whether the captured signals are received by the vehicle (see details in §3).

During our analysis[4], we found that not all vulnerable vehicles and RKE systems are equally susceptible to `RollBack`. Therefore, we derive *four* different variants of `RollBack` w.r.t. a small set of properties (e.g., number of previously captured signals, sequence of the signals, time frame and pace of replay) required for the successful replay attack. We found that vehicles and RKE systems being the most vulnerable to `RollBack` can be unlocked with only *two signals captured any time in the past*. Moreover, these two signals do not even have to be strictly consecutive (see explicit definitions later), i.e., the victim can still use the key fob between the times the attacker manages to capture those two signals. This makes `RollBack` particularly alarming as, in addition to the aforementioned appealing properties, it further minimizes the required efforts of the attacker.

Last but not least, to make `RollBack` even more dangerous, we will show that `RollBack` is *instruction-agnostic*. This means that it does not matter whether the captured signals belong to lock or unlock instructions, making the capturing process even more simpler (see more details in §5.2). Only the last captured and replayed signal has to contain the desired instruction, i.e.,

[4] Our analysis is still ongoing, and, at the time of writing, we have already tested around ~20 different vehicle makes, models, and RKE systems.

unlock to get access to the car.

Similar to RollJam and other RKE attacks, permanent mitigation might be cumbersome if RKE ECU firmware cannot be upgraded over-the-air, requiring calling back whole fleets of vehicles to the factory or dealerships. Some precautionary measures can be applied against signal jamming-based attacks, like RollJam, by assuring proper signal reception by being close to the vehicle, pressing the lock button for the second try if the first unlock signal is not received. In certain scenarios, e.g., car-sharing use cases, risks can be minimized by disabling RKE system until the vehicle is unlocked through the car-sharing app (see details in §8). Nevertheless, since `RollBack`, in essence, is a passive listener in the signal capturing phase without the need of signal jamming, none of the previously-mentioned approaches are applicable to `RollBack`.

Our main contributions are summarized below:

- After revisiting keyless entry systems and RollJam in more details (cf. §2), we propose `RollBack` (cf. §3) that, in contrast to RollJam, can unlock a vehicle *indefinitely* at *any time in the future* and *as many times as desired* by merely replaying previously captured (unlock) signals being already invalid. Hence, `RollBack` is more effective.

- We delineate a (hidden) property of today's RKE systems that mimics the *modus operandi* of `RollBack`, hence being the most relevant candidate to be the root cause of the vulnerability (cf. §7.2). However, for the time being, we could not ascertain whether our attack exploits an implementation bug or a limitation inherited from the design of the key fob re-synchronization or learning feature.

- Through a currently limited yet ongoing real-world experiment, we scrutinize the effectiveness of `RollBack` on a variety of popular vehicles[5], and show that most of them use RKE implementations that are vulnerable to `RollBack` (cf. §4).

- We propose four different variants of `RollBack` based on the requirements, e.g., number of different signals to capture and replay, the time frame and pace of replay, and the consecutiveness of the signals.

- We also discuss that due to the re-synchronization and instruction-agnostic property of `RollBack` and the typical human behavior, astute attackers can rely on capturing lock signals to either fasten the signal capturing process (without signal jamming) or to cover the tracks by locking the vehicle again (cf. §5).

- While the root cause of the attack is unknown mostly due to the lack of documentation, access to resources and knowledge, we delineate a key fob learning process, as a potential root cause, that mimics the behavior or `RollBack`.

- Finally, we discuss possible mitigation strategies; some are precautionary measures the vehicle owner can take when `RollBack` requires signal jamming, and advises to car-sharing services that are particularly vulnerable to `RollBack` (cf. §8). We also discuss possible practical mitigation, e.g., using timestamps**ToDo: *Rohini's work?***.

2 Background and related work

Next, we briefly discuss the evolution of the keyless entry systems. Then, we present the main types of attacks that emerged against this fundamental feature of today's vehicles.

[5] We used our and our friends' and family members' vehicles with their consent due to responsibility.

2.1 The evolution of keys and entry systems

2.1.1 Physical keys

For several decades after the very first car was made in 1886, vehicles had no key at all [18]. The first key was introduced in 1949 by Chrysler Corporation for ignition and starting the engine [19]. It also acted as a safety precaution to prevent children from accidentally starting and moving the car if left in gear.

2.1.2 Immobilizer

To deter vehicle theft, Honda has made the first keys enhanced with a so-called immobilizer. The immobilizer is a passive device that uses RFID technology to communicate with the transponder near the keyhole and verifies the legitimacy of the key fob before starting the engine. Without the correct transponder, the keyhole is either mechanically blocked, avoiding illegitimate keys to turn, or ECUs will not let the fuel flow and start the ignition. Research conducted in Australia and EU have shown that car thefts have been significantly reduced after making immobilizers mandatory [20, 21].

2.1.3 Remote Keyless Entry (RKE)

RKE is an uni-directional authentication system. In RKE, besides advanced features recently became available (e.g., start, stop, panic), user unlocks or locks the vehicle by pressing the corresponding button on the key fob. When a button is pressed, Radio Frequency (RF) signals are emitted towards the car in the frequency bands of 315 MHz, 433 MHz, or 868 MHz depending on the geographic location. The receiver located in the vehicle receives the RF signals (from even up to hundreds of meters) and carries out the intended action (e.g., lock, unlock).

2.1.4 Passive Keyless Entry System (PKES)

Unlike RKE, the Passive Keyless Entry System (PKES) operates automatically when the user, i.e., the key fob, is near the vehicle. Also, PKES uses bi-directional challenge-response communication for appropriate authentication. PKES allows the owner with the correct key fob to unlock and automatically lock the car by pulling the door handle and when the owner walks away, respectively. PKES key fobs are also integrated with RKE, i.e., it still has buttons as a fail-safe/secondary mechanism or feature for drivers in favor of the "old-fashioned" button-based operation.

While PKES also uses rolling codes, due to the owner's proximity and the fact that an attacker does not know when the unlock signals are emitted, they are significantly less vulnerable to typical replay attacks that affect RKE systems.. However, they are susceptible to relay attacks [22].

In this paper, we focus on the RKE systems exclusively.

2.2 Rolling codes

Next, we briefly discuss the evolution of rolling codes used in RKE systems and define some notations used later in the document. The history of RKE systems history reaches back to the 1970s [23] where early motorized garage openers used static codes sent in "plain text" over the air to carry out the intended action (e.g., open, close). However, by merely sniffing and replaying captured signals, attackers were able to easily unlock garage doors. To overcome this issue, rolling codes [17] were introduced, and they have been widely used due to its increased protection (compared to static codes) yet with less computation complexity (compared to the increased protection). The latter property is particularly important as it results in small and simple key fobs with an average battery life of up to four years [24].

There are a few well-known manufacturers providing rolling code-based RKE systems for the automotive industry. For instance, Microchip Technology provides `Classic`, `Advanced`, and `Ultimate` KEELOQ with publicly available documentation and data sheets. On the other hand, semiconductor companies like NXP [25], Omron, and Texas Instruments also provide proprietary solutions for vehicle manufacturers. For the technical explanations below, we focus on RKE systems using the `Classic` and `Advanced` KEELOQ technology since their documentations are publicly available. Note, however, in essence, all rolling code-based technologies are conceptually similar.

Applying the rolling code technology means that every key fob signal transmission is unique, i.e., it changes with every individual button press. Uniqueness is achieved by incrementing a 16-bit wide *counter*[6] in the key fob (and in the vehicle upon reception) with each button press. A button press is valid if the counters at each side are in sync. Then, each of the parties increments its counter[7] to be in sync for the following button press. Accordingly, if an attacker captures a valid signal sent from the key fob and received by the vehicle with counter $C_k = n$ and replays it, it will be discarded by the receiver in the vehicle as its counter $C_v > C_k$, i.e., $C_v = (n+k) : k > 0$.

On the other hand, provision is made for cases in which a button is pressed on the key fob while it is out of range of the vehicle, i.e., when using the key fob to lock/unlock the car and $C_k > C_v$. These cases are further divided into two different *operation windows* [26, 27].

2.2.1 Single window

If $C_{diff} = C_k - C_v$ is small[8], e.g., $C_{diff} < 16$, counter synchronization takes places immediately at the first button press without the need of any additional steps. Counter synchronization means that the receiver unit in the vehicle invalidates all non-received codes before the one present in the last key fob signal.

2.2.2 Resync/double window

If $16 < C_{diff} < 2^{15}$, the receiver temporarily stores the counter $C_k = l$ and waits for a subsequent transmission, i.e., the same button has to be pressed once more. If the subsequent transmission has counter $C_k = l + 1$, the receiver resynchronizes on the last transmission received. Observe, the synchronization requires two button presses, and the vehicle acts only upon the reception of the second one when synchronization finishes.

If any of the above fails[9], the key fob signal received by the vehicle is discarded. Note, furthermore that due to the underlying encryption mechanisms (e.g., in [26]), the change of even one bit of information (e.g., counter increment) results in significant change in the final transmitted signal. Hence, it is computationally infeasible for an attacker to infer the next valid, say, unlock signal by capturing the previous one.

[6] Recent advanced implementations, e.g., `Ultimate` KEELOQ, also maintain timestamps to improve security [26], however it is not confirmed whether RKE manufacturers already adopted them.

[7] For simplicity, here, we suppose an integer increment of 1, however, in the reality the next valid counter is generated via cryptographic hash functions.

[8] Note, different manufacturers use different thresholds.

[9] This window is termed as *blocked window* [27].

2.3 Related work: different attacks against RKE systems

In essence, the design of the rolling code scheme should provide a sufficient level of security, however, the earliest deployments have been proven to be breakable. For instance, `Classic KeeLoq` technology primarily used by garage doors only nowadays, was broken by cryptoanalysis [28, 29] and side-channel attacks on the key derivation scheme used by the receiver [30, 31]. Subsequently, enhanced `KeeLoq` implementations, i.e., `Advanced KeeLoq` and `Ultimate KeeLoq`, have addressed these issues by using stronger encryption algorithms and longer keys [26].

Another simple yet efficient method criminals use against rolling code-based key fobs is jamming the signals when victims press the lock button to hinder the vehicle from receiving it correctly. If it happens without the victim's notice, the car is left unlocked. A more sophisticated variant of this attack is "selective jamming and replaying", where besides the previously-mentioned jamming, the attackers also capture the lock signal. Consequently, if this happens again without the victim's notice, the criminals can lock the vehicle after stealing all belongings to make a false feeling of having the car left adequately locked. Note, once a signal is captured, without additional knowledge (e.g., encryption keys, command code table), it is impossible to convert it into another signal, i.e., flipping a lock signal to an unlock is infeasible.

Hitag2 from NXP, another widely used RKE scheme using rolling codes, has been used by many car manufacturers worldwide (e.g., Renault, Ford, Chevrolet, Lancia, Opel). Recently, researchers have demonstrated a correlation-based attack allowing the recovery of the cryptographic key and thus cloning the key fob with capturing only four to eight rolling codes [32]. Furthermore, the research also revealed that most VW Group vehicles (e.g., VW, Seat, Audi, Porsche) manufactured since 1995 rely on a few master keys. By recovering these keys from the ECUs, an attacker can effortlessly clone the key fob of any such vehicle by only capturing one unlock signal.

In 2015, Samy Kamkar with his RollJam [16] attack has proven all rolling code-based schemes to be breakable. RollJam does neither rely on any cryptoanalysis nor side-channel attacks; it converts a safety feature into an exploit. In essence, RollJam is an advanced "selective jamming and replaying" method; with a careful sequence of jamming, capturing, and replaying signals, it allows an attacker to capture an unused signal from the key fob that can be replayed later to unlock the target vehicle without the victim's notice. As briefly discussed in §1, RollJam is based on four principles, *(i)* capturing unlock signals, *(ii)* jamming the frequency band towards the vehicle at the same time to force the owner *(iii)* to retry, and *(iv)* timely replaying of previously captured signals.

The operation of RollJam is summarized in Fig. 2a. When the unlock button is pressed on the key fob, the rolljam device hidden on or near the target vehicle *(i)* captures the signal and, at the same time, *(ii)* jams the frequency band towards the vehicle to hinder correct signal reception. Since the vehicle does not respond, *(iii)* the owner presses the same button again assuming a lousy signal reception. This time, however, the rolljam device repeats not only step *(i)*-*(ii)*, but also quickly *(iv)* replays the previously captured signal towards the vehicle (without jamming). As a result, the vehicle acts as intended, i.e., unlocks the doors. Besides, the rolljam device becomes aware of the next valid code for the same action, i.e., it knows what signal to send to unlock the car again in the future. However, if the owner uses the key fob to unlock the car again without the rolljam device in action, the signal the attacker possesses will be invalidated forcing her to redo the whole process. While RollJam, in general, is effective against all rolling code-based RKE systems, it requires careful and continuous attention due to *(iv)*.

Recently[10], an attack called Rolling-PWN [33] saw the light of day and hit the headlines of several online news sites, e.g., New York Post [34], The Drive [35], Security Affairs [36]. The authors of Rolling-PWN found that Honda vehicles manufactured between 2012 and 2022, im-

[10] Around a month before the Black Hat debut of `RollBack`, i.e., in the beginning of July 2022.

plementing rolling code-based RKE systems, are vulnerable to replay attacks. In particular, the authors found a somewhat similar behavior to `RollBack`[11]; sending the unlock commands in a consecutive sequence to the Honda vehicles will resynchronize the counter. However, it has not yet been publicly disseminated, what is the required sequence of codes, exactly how many codes need to be captured and replayed, or any other relevant (hardware-specific) details.

3 `RollBack`: a new time-agnostic replay attack

Next, we propose `RollBack`, a new time-agnostic replay attack, which by exploiting a hidden property in the RKE systems, overcomes the limitation of Rolljam. In particular, `RollBack` can unlock a vehicle by simply capturing and replaying a few, already invalidated unlock signals at *any time in the future* and *as many times as desired* without the need of recapturing any further signals later on[12]. In what follows, we describe the threat model of `RollBack` by using same setting as shown for RollJam (i.e., by applying signal jamming) to ease the comparison. However, while jamming can fasten the attack process, unlike RollJam, `RollBack` *does not necessitate signal jamming* at all.

3.1 Threat model and the operation of `RollBack`

The primary goal of the attack is to unlock a vehicle without the victim's authorization (and potentially, its notice). Like in all RKE attacks, the vehicle becomes unlocked the same way as using the original key fob, leaving the car intact.

In our threat model, the attacker has a device that can capture, jam, and replay signals in the frequency band used by the target vehicle. For simplicity, let us call this device `RollBack`-device. In particular, let $\mathcal{S}_I^i$ denote a key fob signal sent towards the vehicle with a rolling code counter $i \in \{1, 2, ..., 2^{15}\}$ and an instruction $I := \{unlock, lock\}$. For instance, $\mathcal{S}_{unlock}^{534}$ marks an *unlock* signal with rolling code counter $i = 534$. Furthermore, let $Capture_A(\mathcal{S}_I^i)$ and $Jam_A(\mathcal{S}_I^i)$ denote that an attacker A captures the key fob signal $\mathcal{S}_I^i$ and jams the frequency band toward the vehicle, respectively, at the same time, i.e., when $\mathcal{S}_I^i$ was sent by the victim. Finally, let $Send_V(\mathcal{S}_I^i)$ and $Send_A(\mathcal{S}_I^i)$ mark when the victim (V) and the attacker (A) send $\mathcal{S}_I^i$ using the original key fob and using a special-purpose device intended to replay captured signals, respectively.

The operation of `RollBack` can be divided into two phases (cf. Fig. 2b).

3.1.1 Reconnaissance phase

The attacker places the `RollBack`-device near the car that is locked and left in public (e.g., in a parking lot). When the victim comes back to his/her car and tries to unlock it via the key fob, i.e., when the victim runs $Send_V(\mathcal{S}_{unlock}^i)$, the `RollBack`-device *(i)* captures the signal ($Capture_A(\mathcal{S}_{unlock}^i)$), and *(ii)* jams the frequency band ($Jam_A(\mathcal{S}_{unlock}^i)$) to hinder the vehicle from receiving it correctly. As a result, the victim assumes a lousy reception and *(iii)* presses the same unlock button again, i.e., s/he runs $Send_V(\mathcal{S}_{unlock}^{i+1})$. This time, the `RollBack`-device captures the second consecutive unlock signal (i.e., it runs $Capture_A(\mathcal{S}_{unlock}^{i+1})$), *however*, unlike RollJam, it also lets the car receive it, i.e., the attacker *does not* run ($Jam_A(\mathcal{S}_{unlock}^{i+1})$). Accordingly, the vehicle unlocks, and the victim drives away, assuming that no harm has been done. Note, since `RollBack` does not have to keep track of the next valid unlock signal, it is unnecessary to suffix the `RollBack` device to (a hidden spot of) the vehicle. Practically speaking, due to the size of the inexpensive elements needed (see later in §3.2), such a special-purpose wallet-size [37] `RollBack`-device can be

[11] https://twitter.com/Kevin2600/status/1545593961313472512

[12] See RollBack in action at https://www.youtube.com/playlist?list=PLYodcy84oQL1gxwiuRm13xRXxTQL9cO5t

simply thrown below the vehicle. At the end of the reconnaissance phase, the attacker becomes aware of two consecutive correct unlock signals. Recall, by the rolling code design, both captured signals are *not valid* anymore.

3.1.2 Exploitation phase

Unlike RollJam, this phase does not have to follow the first phase directly. In other words, the victim can continue to lock, unlock, and use her/his car as usual as *many times* s/he wants (cf. Fig. 2b). Nevertheless, at any given latter time, once the vehicle is locked, the attacker can unlock the vehicle (without the victim's authorization) by replaying the previously captured two consecutive unlock signals, i.e, by running $Send_A(\mathcal{S}^{(i)}_{unlock})$ and $Send_A(\mathcal{S}^{(i+1)}_{unlock})$.

For brevity, our threat model does *not* cover further intentions of the attacker after unlocking the vehicle. The attacker might steal belongings left inside the car, or use other attack methods (if necessary) to steal the vehicle itself.

3.2 Essential hardware

For our comprehensive analysis, we use Software Defined Radio (SDR) devices. In essence, these devices have wireless receivers (and transmitters) that can be fine-tuned via software, for instance, in which frequency domain they should listen to signals. One of the most well-known and commodity-of-the-shelf (COTS) devices is HackRF One [38], which is capable of both transmitting and receiving signals, and costs $\sim 300-400$ USD at the time of writing. The COTS software, called `gqrx` can be used to easily identify the exact frequency used by the key fob to transmit the signals. On the other hand, since all key fobs operate in the licensed spectrum, they all (must) have a unique registered identifier with Federal Communications Commission (FCC). Therefore, one can lookup the publicly available details of a key fob by keying in its FCC ID at `https://fccid.io/`. Once the correct frequency is identified, the other COTS software, called Universal Radio Hacker (URH, [39][13]), can be used to control SDR devices, i.e., to practically capture and replay (the unlock) signals. To jam the frequency using the SDR device, an attacker has a large variety of options, and it is completely up to her appetite and knowledge. For instance, she might use inexpensive programmable development boards and radio transmitters, such as Arduino-based deployments, or even a Raspberry Pi with a full-fledged operating system and RTL-SDR dongles [40] for reception and/or CC1101 transceivers for jamming [41]. Note that, essentially, `RollBack` relies on the exact hardware requirements as RollJam. Moreover, since jamming is not necessarily needed (cf. §3) for the success of `RollBack`, a `RollBack`-device has even less requirements. Therefore, it would cost no more than a couple of tens of US dollars [42].

3.3 Different variants of `RollBack`

When we first discovered the vulnerability, we have tested a pretty outdated vehicle, a Nissan Latio from 2009 (see details in §4.1). In this case, `RollBack` had the following properties.

Naturally, first, we identified how many signals do we need to replay. In case of the Nissan Latio, this number turned out to be only *two*; however, as we will show, other vulnerable systems might require more than that. Accordingly, the first (and most important) property of `RollBack` is the number of signals (i.e., `#SIGNALS`) an attacker has to capture (and replay).

The second observation we had is that the attacker strictly has to run $Capture_A(\mathcal{S}^{i}_{unlock})$ and $Capture_A(\mathcal{S}^{i+1}_{unlock})$ and replay them in the same sequence. Put differently, capturing and replaying,

[13] There are several other publicly available free and/or open-source software, e.g., GNURadio, OpenSDR, that can be used for the same purpose.

Variant	#SIGNALS	SEQUENCE	TIMEFRAME
$\texttt{RollBack}^{\texttt{Loose}}_{\otimes}(2)$	2	Loose	$\otimes$
$\texttt{RollBack}^{\texttt{Strict}}_{N}(2)$	2	Strict	N sec
$\texttt{RollBack}^{\texttt{Strict}}_{\otimes}(3)$	3	Strict	$\otimes$
$\texttt{RollBack}^{\texttt{Strict}}_{\otimes}(5)$	5	Strict	$\otimes$

Tab. 1: Different variants of RollBack derived from our analysis. Each variant encodes all properties as $\texttt{RollBack}^{\texttt{SEQUENCE}}_{\texttt{TIMEFRAME}}(\texttt{\#SIGNALS})$.

for instance, $\mathcal{S}^{i}_{unlock}$ and $\mathcal{S}^{i+k}_{unlock} : k > 1$ does not trigger the expected rollback-like mechanism. Hence, we call the second property SEQUENCE and it can be Strict (like in case of the Nissan Latio mentioned before), or Loose if it is not required, i.e., when replaying signals in the capturing (i.e., ascending) order is sufficient but there could be further valid and forfeited signals in between.

Furthermore, in the case of the Nissan Latio, we observed that the two consecutive unlock signals have to be replayed *within five seconds*; otherwise, the RollBack is unsuccessful. We termed the third property TIMEFRAME and it indicates the maximum number of seconds that can elapse between two signals when replayed. We indicate TIMEFRAME as $\otimes$ when there is *no limit* on the maximum number of seconds. When TIMEFRAME $\neq \otimes$, we confirmed the value of TIMEFRAME, by carefully trimming gaps between the captured signals to exactly $N = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ seconds. Then, we saved the signals, replayed them and observed whether RollBack succeeds. Note, once the signals are captured, TIMEFRAME can be easily adjusted via the SDR software by cutting or copy-pasting the breaks/noises between the signals.

During our analysis (detailed later in §4), we derived *four* different versions of RollBack regarding the properties mentioned above. The different combinations are summarized in Table 1.

4 Evaluation

Next, we evaluate RollBack and discuss which vehicles are vulnerable.

Disclaimer

For our experiments, we *did not* carry out any attempts with RollBack in the wild. All tests were executed in an isolated environment, where no other vehicles and/or key fobs were in close vicinity. All of the captured signals (for the tests) had been stored temporarily only; after capturing the signals and replaying them, the data had been removed permanently immediately. We have stored two key fob signals for a longer period, i.e., ~ 100 days to validate RollBack's time-agnostic feature. Afterward, those stored signals were also removed permanently. Note, furthermore, replaying key fob signals do not cause any harm to the vehicle, the key fob, and the whole electronic ecosystem irrespectively of being vulnerable to RollBack. Thus, the tested vehicles continue to work and behave as usual.

This paper is the first publicly disseminated, detailed written information about our findings and about RollBack in general. We used its shorter and more condensed preliminary versions of this document during our attempts in initiating disclosure processes with RKE chip manufacturers and AUTO-ISAC members. See more details about the disclosure processes and findings in §7.1.

4.1 Vehicles Evaluated

As mentioned in §4, we could examine a limited number of vehicles. In particular, for the time being, we could examine several popular Asian vehicle makes and models available in Singapore. The vehicles examined and their relevant data are detailed in Table 2. Model date means the time frame the actual model was in production, while the Mfg. date denotes the actual manufacturing date of the vehicle we tested. Such information were obtained by using the vehicles' identifier, i.e., their VIN numbers, and publicly available services[14].

Car Make	Model	Model date	Mfg. date	RKE manufacturer	RollBack (variant)
Honda	Fit (hybrid)	2016-2018	2016	NXP F2951X	$\texttt{RollBack}_{\otimes}^{\texttt{Strict}}(5)$
	Fit	2018	2018	NXP 61X0915	$\texttt{RollBack}_{\otimes}^{\texttt{Strict}}(5)$
	City	2017	2017	NXP F2951X	$\texttt{RollBack}_{\otimes}^{\texttt{Strict}}(5)$
	Vezel	2016-2022	2017	NXP F2951X	$\texttt{RollBack}_{\otimes}^{\texttt{Strict}}(5)$
Hyundai	Elantra	2013-2015	2015	Omron MD-015	$\texttt{RollBack}_{\otimes}^{\texttt{Loose}}(2)$
	Elantra	2012	2012	NXP 32182C[15]	NO
	Avante	2018-2020	2020	NXP F7936[16]	NO
Kia	Cerato/Forte K3	2016-2018	2017	Omron MD-011	$\texttt{RollBack}_{\otimes}^{\texttt{Loose}}(2)$
	Cerato/Forte K3	2012-2018	2015	Omron MD-011	$\texttt{RollBack}_{\otimes}^{\texttt{Loose}}(2)$
Mazda	3	2018	2018	NXP A2V25	$\texttt{RollBack}_{\otimes}^{\texttt{Strict}}(3)$
	2 Sedan	2018	2018	NXP F7953	$\texttt{RollBack}_{\otimes}^{\texttt{Strict}}(3)$
	2 HB (facelift)	2020	2020	NXP A2V25	$\texttt{RollBack}_{\otimes}^{\texttt{Strict}}(3)$
	Cx-3	2019	2019	NXP A2V25	$\texttt{RollBack}_{\otimes}^{\texttt{Strict}}(3)$
	Cx-5	2018	2018	NXP F7953	$\texttt{RollBack}_{\otimes}^{\texttt{Strict}}(3)$
Nissan	Teana	2014	2014	NXP 063168C	NO
	Latio	2007-2012	2009	Microchip	$\texttt{RollBack}_{5}^{\texttt{Strict}}(2)$
	Sylphy	2012-2019		NXP F7952	$\texttt{RollBack}_{8}^{\texttt{Strict}}(2)$
Toyota	Wish	2009-2017			NO
	Corolla Axio	2015-2017		TI 37143ADN	NO
	Altis	2005		TI 37200A	NO
	Prius (hybrid)	2020	2020	TI	NO

Tab. 2: Vehicles' details used for our in-house experiments. For the vehicles where the release date and manufacturing date are the same, only the manufacturing date was available by using the vehicle's identifier (VIN). For the Toyota vehicles, the VIN numbers were not available, hence we left those cells intentionally blank. Moreover, for some vehicles, we could also not identify the RKE system manufacturer; hence, corresponding cell was also left intentionally blank.

Different vehicles and their key fobs use different frequencies, however, since the used frequency did not have an impact on whether the vehicle is vulnerable to `RollBack`, we omit the exact frequency bands. Furthermore, we could also obtain the exact RKE manufacturer and chip version and serial number most of the times by manually disassembling the key fobs[17]. When disassembling the key fob was either infeasible or the the chip(s) on the PCB were obscured (e.g., via black paint), we tried to gather manufacturer information by keying in its FCC ID at `https://fccid.io/` or looking for spare key fobs on different retailers' sites. The found chips are

[14] One can rely on https://vindecoderz.com to check all publicly available basic servicing information about a vehicle by using its VIN number

[15] Inferred from https://bit.ly/3POlZaz.

[16] Inferred from https://bit.ly/3OrwbEV.

[17] In some cases, the key fob's printed circuit board had an extra plastic cover, which could not be removed without making permanent damage.

detailed in the penultimate column of Table 2. If we could not obtain the RKE manufacturer by any of the above-mentioned ways, we left the corresponding cells in Table 2 intentionally blank.

Finally, the last column indicates whether the vehicle, or more precisely, the RKE system is vulnerable to `RollBack` (indicated by the actual `RollBack` variant that works).

From our experiment (cf. Table 2), which we continuously update[18], we can conclude the following. First, more than $\sim 70\%$ of the examined vehicles found vulnerable to a `RollBack` variant. Furthermore, the vulnerability is not specific to any sole vehicle, car make, or model.

While the age (i.e., model and manufacturing date) does not seem to be a deciding factor, the used RKE system's manufacturers *might be* a telltale sign. In particular, RKE systems from Omron found in most Korean vehicles (e.g., Kia, Hyundai) are the most vulnerable requiring only two unlock signals that could even be captured independently in the past (i.e., `SEQUENCE=Loose`). On the other hand, by having an RKE system from NXP does not necessarily indicate whether our vehicle is vulnerable (to any `RollBack` variant) as some of the evaluated vehicles with NXP transponders in their key fobs turned out to be safe. Furthermore, we observe that all three tested Toyota vehicles turn out to be immune to `RollBack`. From an RKE manufacturer aspect, even though the case of Toyota Wish where we could not identify the RKE system used, we observe that the RKE systems of the Toyota vehicles rely on Texas Instruments transponder chips in their key fobs and, as mentioned above, none of them is susceptible to `RollBack` at all. Last but not least, Microchip RKE systems were probably more ubiquitous in the past, however, their rolling code-based solution can still be found in today's vehicles and they might all be vulnerable to `RollBack`.

Note, however, that not the key fob (as it only sends the signals) but its counter-part (i.e., the receiving unit in the car *per se*) seems to be vulnerable. Moreover, the key fob manufacturer usually produces key fobs (i.e., the transponders) *only*, and the receiving units are produced by different OEMs. Yet, our results indicate a strong relationship between the key fob manufacturer and the receiving unit as we have not found any two RKE systems that use the same transponder chip in their key fobs but react differently to `RollBack`.

5 Further appealing features of `RollBack`

This section discusses how easily *attackers might hide their tracks* after accessing a vehicle, and shows that `RollBack`, in certain cases, can be successfully launched with even less effort, i.e., *without the need for signal jamming.*

5.1 Re-locking the vehicle after access

Recall that due to the counter re-synchronization, if subsequent signals are captured and replayed, they also work as expected straight away afterward. Using the notations defined in §3.1, assume the attacker not only captures consecutive unlock signals (e.g., $Capture_A(\mathcal{S}^{i}_{unlock})$, $Capture_A(\mathcal{S}^{i+1}_{unlock})$ in case of $\mathtt{RollBack}^{\mathtt{Loose}}_{\otimes}(2)$), but also captures a following lock signal $\mathcal{S}^{i+2}_{lock}$ (i.e., $Capture_A(\mathcal{S}^{i+2}_{lock})$). In this case, irrespectively of whether the victim continues to use the key fob as normal (i.e., whether the last signal received by the car is $\mathcal{S}^{i+2}_{lock}$ or $\mathcal{S}^{i+j}_{(un)lock} : j > 2$), after $Send_A(\mathcal{S}^{i}_{unlock})$ and $Send_A(\mathcal{S}^{i+1}_{unlock})$ (in case of $\mathtt{RollBack}^{\mathtt{Loose}}_{\otimes}(2)$), the vehicle unlocks and also resynchronizes to the counter $(i+1)$. Accordingly, after the attacker accessed the vehicle, when running $Send_A(\mathcal{S}^{i+2}_{lock})$, the car will lock, making a false feeling for the owner of having the vehicle left adequately locked.

[18] Please see an online crowdsourced version of this table here: https://docs.google.com/spreadsheets/d/1cj5jK_7_Ibb7q6H9yvGbxAg8rQMZm4M2wgL0GyLUxHE/edit?usp=sharing

5.2 `RollBack` is instruction-agnostic

To achieve the re-synchronization via `RollBack`, the instructions embedded in the signals do not matter. For instance, in case of $\texttt{RollBack}^{\texttt{Loose}}_{\otimes}(2)$, capturing and replaying one lock signal and then an unlock signal is sufficient to unlock the target vehicle. Suppose now that the attacker captures the lock signals emitted when the victim left the vehicle in a parking lot (i.e., $Capture_A(\mathcal{S}^{i}_{lock})$). Then, the attacker wait for the victim to come back and unlock the vehicle; this time the attacker runs $Capture_A(\mathcal{S}^{i+1}_{unlock})$. Recall, in case of $\texttt{RollBack}^{\texttt{Loose}}_{\otimes}(2)$, the second signal does not even have to be strictly consecutive, i.e., the attacker can simply capture any following unlock signal (e.g., $Capture_A(\mathcal{S}^{i+k}_{unlock} : k > 1)$) to unlock the vehicle. After replaying these two signals in sequence, the vehicle will be locked and resynchronized to the counter $(i + 1)$, and the vehicle will react according the instruction in the last signal, i.e., it unlocks.

This makes `RollBack` particularly alarming as this signal sequence can be easily captured at once without applying any signal jammer. Moreover, even if the vehicle is susceptible to a `RollBack`-variant that requires more signals, they can be also be captured without jamming due to the following typical human behavior and the vehicles' safety features. For instance, when we leave something worthy unattended (e.g., the vehicle in the parking lot, the main entry door to our home), we usually confirm whether locking was done adequately. For this reason, most of us still push (down the handle on) the door of our home after locking to double-check whether the lock itself is not malfunctioning. Similarly, it is always worth pressing the lock button on the key fob once more when we leave our vehicle behind since it confirms adequate locking by flashing the emergency signals and/or honking.

Pressing the lock button again (for third or even more time) afterward thereby making the vehicle honk can also become handy afterward. People tend to use this feature in huge parking lots to locate the vehicle *per se.*

On the other hand, vehicles usually implement a safety feature when unlocking the car via the key fob. This feature allows the owner to only unlock the driver's door upon pressing the unlock button for the first time. However, if one does not driving alone, giving access to the other co-riders (e.g., family members), we have to press the unlock button twice to unlock all doors.

These features and usual human factors enable all `RollBack`-variants to be successfully launched without the need of any signal jammer.

6 Car-sharing Services: The Most Attractive Targets of `RollBack`

Car sharing has recently been viral, especially in countries where the cost of ownership for a vehicle is extremely high compared to the average. Car sharing, in essence, makes classic car renting much more accessible, more convenient, and much cheaper too. Instead of renting a vehicle for at least a day, doing a lot of paperwork in-person, get lost among the different insurance policies and waivers, car-sharing costs are significantly lower due to the non-necessity of staff, an hour or minute-based conditions, and the convenience of using a mobile application to access and lock the vehicle in the beginning and at the end of the rental, respectively.

The typical steps of car-sharing are as follows. Users (already registered for the service) can use the mobile app to book a car (for a certain period). Once the booking timeslot starts, the user can unlock the vehicle by instructing the mobile application to do so. In the background, the car-sharing company's service remotely unlocks the vehicle utilizing additional ECUs added to the car for this specific reason. Once the vehicle unlocks, the user will find the original key fob at a hidden spot in the car (usually in the glove compartment), then s/he can start driving. Note, typically, there are further different steps the car-sharing company might require (e.g., photo-taking, damage and petrol level checking); however, from our attacking point of view, they are not relevant. After the user returns the vehicle to a designated parking lot, s/he has to put back the key fob to the hidden spot it was found in the beginning. To finish the renting, the user has to get out of the vehicle, close all doors, carry out any aforementioned additional steps required by the car-sharing company, and use the application to lock the vehicle[19].

An attacker can easily use the key fob to capture the required number of unlocking signals during the renting phase. Since the attacker temporarily owns the vehicle, she might even carry out further tests (e.g., checking which RollBack-variant works and how many signals are required accordingly). Once she returns the vehicle, the rental process officially ends, and during that period, the attacker took care of the vehicle well, and no harm was done. Later, other users will use the car. An attacker, most of the time, does not even need any effort (e.g., physically following the car, installing a GPS tracker) to keep track of the vehicle. The car-sharing service gives all the necessary information to the attacker. In particular, in point-A-to-A car-sharing, where each vehicle has a single dedicated lot it has to be returned to to finish its rental, the given vehicle's status and booking schedules are usually available upfront. In the case of point-A-to-B car-sharing, i.e., where vehicles can be picked up and returned to different places, individual booking schedules might not be available. However, information required for a seamless booking experience (e.g., license plate numbers of nearby vehicles, only showing currently available vehicles) is available through the application. This means that attackers can easily implement crawling scripts to obtain the necessary location information about the target vehicle.

Utilizing such information, the attacker can significantly reduce suspiciousness by waiting for the vehicle to be booked (and used) by several other users. Once there is a time-slot when the vehicle is available, the attacker can launch RollBack to access and steal the vehicle (since the key fob is inside the car). Note, since car-sharing companies usually install GPS trackers to keep track of their fleet, stealing the vehicle might be less appealing. Yet, using the same availability information, the attacker can check when a particular vehicle will be booked in the future. Then, she can approach the vehicle before the scheduled booking starts, wait for the victim to rent the vehicle, and follow him/her until the vehicle is temporarily left, i.e., when it is locked but not returned, for instance, during shopping. The attacker can then use RollBack to unlock the vehicle and steal the belongings left behind.

While one can quickly come up with countless different ways how and when to exploit RollBack and what an attacker might do afterward, due to the simplicity and the little effort needed, RollBack is particularly alarming for car-sharing (and classic car-renting) companies as attackers can do much harm to the rental companies' user bases; eventually to their reputation.

7 Responsible disclosure process to identify the root cause

In this section, we describe our responsible disclosure process, particularly, how we started, what obstacles we bumped into, and eventually, what take-aways we received. The steps we describe are not necessarily the recommended steps to take in such situation. Due to the limited time frame we were given, we made the following steps, and we share below our experience that could potentially help us (and others) in future responsible disclosure processes.

[19] Some advanced car sharing companies have already gone completely keyless, i.e., there is no key in the vehicle at all, and even temporarily locking the vehicle in a parking lot (without returning the car) is done through the mobile app.

7.1 Responsible disclosure process(es)

Step 1: Who should we contact and who responded?

While this question seems to be the easiest, it was not in our case due to one main reason: we could only carry out our experiments in a limited yet diverse set of vehicles. Accordingly, after finding one car make and model vulnerable, should we contact the car manufacturer, e.g., Hyundai, straight away? They would probably ask first: which specific *vehicles* are vulnerable? Are they the newest models, or older ones? Is there any other model from the same make that was found vulnerable? Are all Hyundai vehicles vulnerable? We would have not been able to answer (m)any of these questions due to our limited experiment.

Therefore, we kept experimenting with different vehicles we could have access to until we reached a certain point when 2-3 RKE systems using different key fob transponder chips from the same key fob vendor were found vulnerable, irrespective to the vehicle itself.

This led us to two key fob manufacturers, namely NXP and Omron (cf. Table 2). While Omron did not have a specific website for reporting vulnerabilities, we have tried to reach out to them through their contact forms found on their international [20] and local[21] (i.e., Singapore) sites. However, we did not receive any response.

NXP, on the other hand, takes vulnerability disclosure processes very seriously. Vulnerabilities can be reported to their PSIRT (Product Security Incident Response Team) for which all necessary information is provided on their website[22].

Step 2: First meeting with NXP

We have scheduled a virtual session with NXP in March 2022. By definition, before that, many legal processes had to be done, including Non-Disclosure Agreement (NDA) between the companies. While the content of our disclosure process is protected by NDA, we share below the conclusions we could draw. The vulnerability we found is indeed a vulnerability and there is no such feature that exactly works the same way as `RollBack`. However, the vulnerability is in the receiver side of the RKE system, which manages the rolling codes, and verifies the validity of each code received; the key fob only sends the signals expected by the vehicle.

On the other hand, it is somewhat known that vendors producing key fobs *only produce* the transponders, and car manufacturers obtain the receiving parts from other OEMs. Accordingly, it is very likely that vehicles using key fobs from other vendors might have the same type of vulnerability due to supply chain for the receiving units.

Step 3: Meeting with Auto-ISAC members

The key fob manufacturers are (likely) not responsible for the receiving unit, which seems to be the component vulnerable to `RollBack`. However, for us, researchers, it is particularly difficult to reach out to those manufacturers as we do not even know who they are. The reason why we could end up with NXP is that we could disassemble the key fob, where the chip manufacturer can be identified. However, we neither had the appropriate knowledge nor the approval from the car owners to tear down the vehicle, identify the corresponding ECU, disassemble it, and get to know the manufacturer by reading any of the chips on the PCB. Luckily, we were able to get engaged with the Automotive Information Sharing and Analysis Center (Auto-ISAC[23]). Auto-ISAC is a

[20] https://bit.ly/3yXGElG
[21] https://bit.ly/3ooVr42
[22] https://bit.ly/3BeMLF1
[23] Their website can be found at https://automotiveisac.com/.

US-based industry-driven community, which shares and analyzes intelligence about emerging cybersecurity risks to the vehicle, and collectively enhances vehicle cybersecurity capabilities across the global automotive industry. They are partnered with several car and OEM manufacturers, and thanks to Ricky Brooks, Auto-ISAC members could set up a virtual meeting with us (in May 2022) where many (10+) representatives from the industry were present. Although, we did not have time and/or intention to sign an NDA with each manufacturer, otherwise our sharing session would have still not been scheduled. Consequently, the Auto-ISAC members could only agree to join the session, listen to our findings, but they were not allowed to get engaged more with us. This means, that besides general questions they could ask, we could not advance towards identifying the root cause of the vulnerability and any possible mitigation.

Step 4: Take-aways from the disclosure process

We ended up having two main take-aways from the disclosure process. First, the members acknowledged the vulnerability as well as our intention to present our findings (with or without limitations on the context) at Black Hat USA 2022. Second, from the nature of some more generic questions and reactions, we could draw a conclusion. Note, this conclusion is utterly our opinion on the subject and it does not reflect any statements from any car manufacturers. Since our attack targets one specific vehicle (not a fleet of vehicles in general) and has to redone from scratch for other vehicles (even from the same make/model), it might not be particularly alarming for the car manufacturers. Roughly speaking, there is not much difference between breaking the windows/lock-picking the doors of the target vehicle to steal belongings, and doing a more sophisticated and unnoticeable attack like `RollBack` to achieve the same. Both approaches always need to pick the target, find the right timing, and carry out the attack. Furthermore, `RollBack` on its own does not allow an attacker to steal the vehicle itself.

We found that to the vulnerability revealed recently (Rolling-PWN [33]), the reaction of Honda [43] has somewhat underpins our above-mentioned conclusions drawn.

7.2 Towards Finding the Root Cause

According to the normal operation (discussed in §2.2.1 and §2.2.2), since the counter value C_k of the key fob signals replayed by `RollBack` is smaller than C_v, they should be discarded. Thus, when we first discovered this vulnerability, we immediately thought that the phenomenon belongs to some sort of key fob re-synchronization, which is required when a new transmitter (i.e., a key fob) is learned to the receiver (i.e., the vehicle's RKE system) or when the battery is replaced in the key fob and it might lose its last counter values[24]. *However, currently, we cannot confirm the root cause of this vulnerability* for several reasons. First, datasheets with explanation on how the system architecture works (including the described learning process) is only available for Microchip offerings [26, 27]. Therefore, we discuss the key fob learning process in Microchip KeeLoq systems in detail, and also point out the critical steps that are not (completely) in line with the operation of `RollBack`.

In the KeeLoq system [27], the typical learning process is as follows (cf. Fig. 3). After entering into the learning mode, when a button on the new key fob is pressed, the first signal is sent to the vehicle. The signal has an unencrypted part containing the key fob's serial number and an encrypted part containing the rest of the data, e.g., rolling code counter, discrimination bits, button pressed[25]. Using the master key added during manufacturing, the receiver in the vehicle

[24] Note, one can easily find a third-party tutorial (video) on how to learn a new key fob to a certain vehicle make and model, however, these tutorials neither reveal which manufacturer's RKE system they configure nor why the learning process works in that way.

[25] For more details about the basic packet formats, refer to [27].

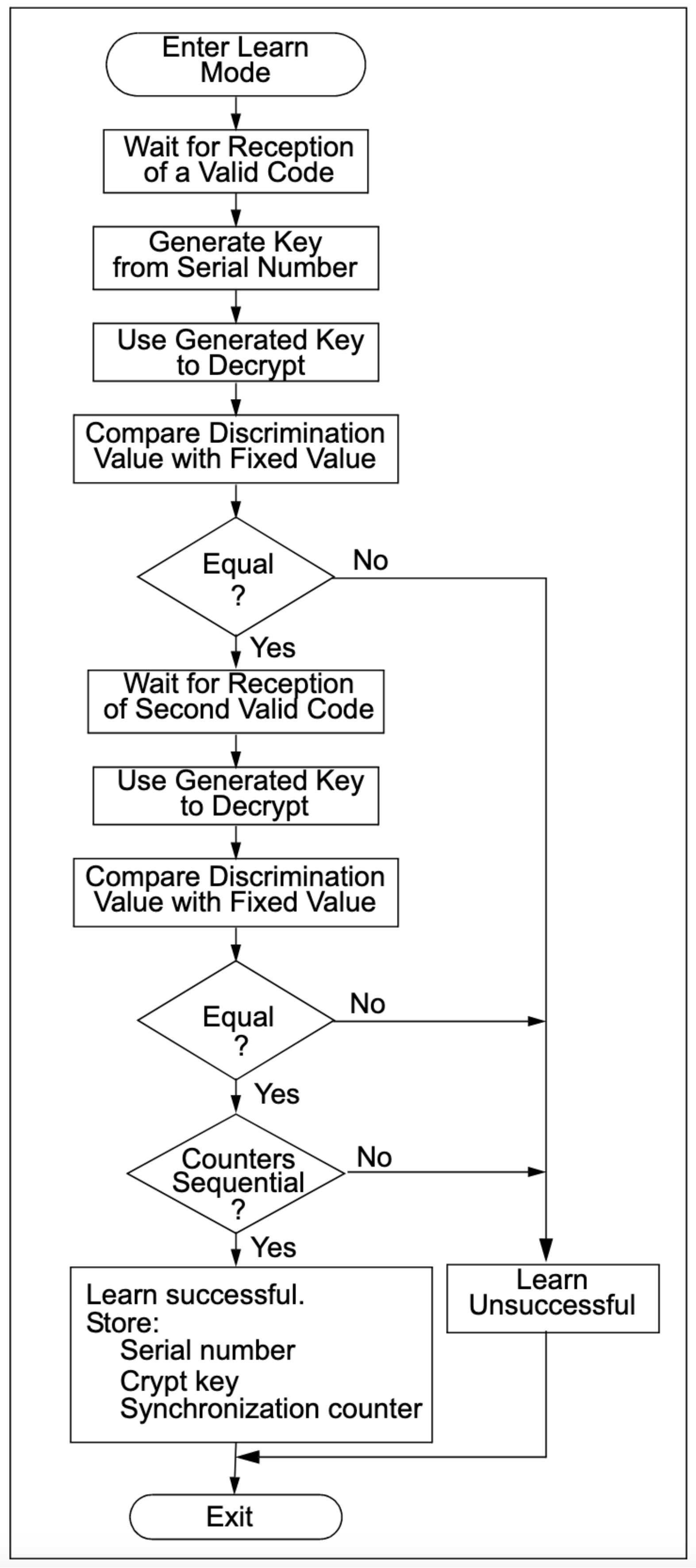

Enter Learn Mode
Wait for Reception of a Valid Code
Generate Key from Serial Number
Use Generated Key to Decrypt
Compare Discrimination Value with Fixed Value
Equal ?
No
Yes
Wait for Reception of Second Valid Code
Use Generated Key to Decrypt
Compare Discrimination Value with Fixed Value
Equal ?
No
Yes
Counters Sequential ?
No
Yes
Learn successful.
Store:
Serial number
Crypt key
Synchronization counter
Learn Unsuccessful
Exit

Fig. 3: Typical learning sequence in KeeLoq HCS200 / HCS300 RKE systems [27].

generates the correct encryption/decryption key[26] for the key fob using its serial number emitted unencrypted in the first signal. Then, after decrypting the packet using the freshly generated key, the receiver authenticates the signal. Briefly, authentication involves validating the correct key use via the discrimination bits and buffering the counter value $C_k = n$. Afterward, the receiver waits for the second signal, i.e., for the second button press on the key fob. When the second signal is received (and authenticated), the receiver checks whether the transmission is indeed the second one, i.e., whether the second counter $C_k = n + 1$. The receiver stores the key fob's serial number, current synchronization counter, and appropriate decryption key upon successful completion of this process. Finally, the system exits from the learning mode. After this point, whenever the freshly added key fob is used in the future, this decryption key is retrieved from the memory along with the stored synchronization counter.

Clearly, the operation of the above-mentioned learning process mimics the operation of `RollBack`. However, there are *five* key observations we have to consider as they are not elaborated sufficiently and they might undermine such a claim accordingly.

7.2.1 Learn mode

Observe that the learning sequence starts with a step `Enter Learn Mode`. Depending on the make, model, and build-year, different vehicles implement different yet intricate approaches to put the receiver in the car into learn mode. In other words, to avoid accidentally entering into learn mode, the vehicle (i.e., the RKE system) requires very uncommon sequence of actions that would not be carried out during normal use. For instance, some Toyota vehicles require the key to be turned in the ignition from OFF to ON and repeat within five seconds [44][27]. However, `RollBack` does not require entering into this mode explicitly.

On the other hand, upon a successful learning process, the system should exit from this mode by default (cf. `Exit` step in Fig. 3). This means that the vehicles found vulnerable to `RollBack` (see details in §4) are either always in a learn mode (i.e., do not exit) or do not have this initial step at all, i.e., synchronizing a new key fob to the vehicle is over-simplified.

7.2.2 Timeframe

As discussed in §3.3, some RKE implementations require the captured signals to be replayed within a certain time frame (e.g., `RollBack`$_N^{\texttt{Strict}}$(2)), while others have no such requirement. This property is not defined in the available documentation, e.g., in [26, 27]. However, even [27] claims that the method describes a typical implementation, real-world deployments might be altered to fit other needs.

7.2.3 Number of signals and their sequence

While the learning process require the key fob to be pressed two times in a sequence, several `RollBack`-variants we derived work differently. For instance, `RollBack`$_\otimes^{\texttt{Loose}}$(2) does not requires strictly consecutive signals, while other variants, e.g., `RollBack`$_\otimes^{\texttt{Strict}}$(5), need more than two signals. Recall that the learning process described in Fig. 3 applies to Microchip's solutions; however, the previously mentioned `RollBack`-variants work against other RKE manufacturers (see details in §4).

[26] The KeeLoq algorithm uses a symmetrical block cipher; hence the encryption and decryption keys are identical.

[27] One can easily find several tutorial videos online on how to learn a new key fob to a vehicle.

7.2.4 Vehicle's reaction

Another missing piece from the puzzle is to describe which *(i)* actual button (and its instructed action) should be pressed, and *(ii)* whether the same button has to be pressed for the second time. However, since only the key fob's serial number and the discrimination bits matter during the learning process, pressing two different buttons and sending two different signals *(i)*-*(ii)* accordingly should have no impact on the learning process. Put differently, sending a lock signal and an unlock signal should be sufficient to learn a new key fob to the vehicle.

Nevertheless, at the end of the learning process (cf. `Learn Successful` in Fig. 3), there is no indication of whether the vehicle should react to the second button press with the intended action (e.g., lock the doors if lock button was pressed). However, in the case of `RollBack`, the intended action in the last signal (e.g., unlock) is always materialized.

7.2.5 Re-learning the same old key fob

We can observe that there is no information available about what happens if an already learned key fob (e.g., the original key fob) is being re-added to the system. One of the vital steps in the learning process is to save the serial number of the key fob and the accompanying crypt key in memory. Thus, the vehicle can have this information straight away from memory in the future, when the the new key fob is used. During the learning process, however, there is no step involved in checking whether the serial number of the key fob is already known (before adding it to the memory). Due to this missing check and §7.2.4, it is unclear whether re-adding an already known key fob is silently ignored (i.e., leaving the system still in learning mode waiting for a new key fob to be added) or re-added as new.

7.2.6 Out-of-sync counters

Finally, observe that during the learning process, the counters of the key fob are buffered for the first signal and only stored upon success. However, the counter's value C_k is not checked (against the counter at the vehicle C_v. This, on the other hand, is somewhat expected; normally, a new key fob cannot be in sync with the vehicle, hence the learning process. Furthermore, synchronizing the new key fob's counters to the counters of the actual key fob we use everyday would make no sense at all either. The different key fobs are always going to be out of sync due to using one of them at a time; hence, the vehicle's receiver stores a separate synchronization counter for all key fobs learned. This can be the case why consecutive but out-of-sync old counters are always accepted without further validations.

While the learning process is the only action we identified in the RKE system that somewhat mimics the operation of `RollBack`, according to our arguments above, we cannot state with confidence whether `RollBack` indeed exploits this feature. Nevertheless, if the found exploit is in the learning process, then the vulnerable vehicles are *probably* unintentionally left in a *"forever" learn mode* (§7.2.1), which allows re-adding an already learned key fob (§7.2.5) by simply replaying old consecutive signals (§7.2.6), and the vehicle will react accordingly (§7.2.4).

8 Mitigation

To identify and propose proper mitigation strategies or patches, the root cause of the vulnerability must be identified first. However, as mentioned in §7.2, for the time being, we were not able to pin-point the root cause with confidence. Accordingly, in this section, we devise different types of mitigation strategies; general advises for an owner to be vigilant and avoid being targeted of

RKE attacks mostly relying on jamming (e.g., RollJam), for the case of astute attackers (cf. §5), and the car-sharing/renting scenarios.

8.1 General advices

Since RollBack, just like other replay-based attack techniques (e.g., RollJam [16]), can utilize jamming to speed up the whole process, a user can be vigilant to realize a possible exposure to signal jamming. The most important thing is always to be close enough to the vehicle to avoid lousy signal reception. Thus, if the first button press was not realized by the vehicle (but the second was[28]), then there is a high chance of the first signal being jammed (and captured). In such circumstances, the owner may press the lock and unlock buttons interchangeably until *(i)* both two last button presses were correctly received, and *(ii)* the vehicle acts as intended. If only *(i)* holds, the owner might still be exposed to continuous attacks such as RollJam, which jams the latest signal and replays a previously captured one. However, with *(ii)*, the owner can definitely rule out the possibility of such attacks taking place.

Additionally, advanced rolling code implementations having precise timestamps besides the counters (e.g., in Ultimate KEELOQ [26]) avoid any practical replay attacks because of the time difference between the vehicle and the key fob's signal.

Note, RollBack does not require jamming at all. Accordingly, since in essence it works as a passive listener during the reconnaissance phase (§3.1.1, there is no way to realize whether one is a victim of RollBack.

8.2 The problem of instruction-agnosticism

While having one rolling code per each learned key fob simplifies the design and reduces the resource requirements, implementing different rolling codes for each instruction will easily evade the problem discussed in §5. In particular, by replaying lock signals and hence re-synchronizing its counters, only the further yet invalid *lock* signals would work. On the other hand, the rolling codes of the unlock instructions would remain intact, still preventing the replay of a single unlock signal to open the vehicle (after re-synchronizing the lock instruction's counter). This would significantly reduce the easiness of RollBack, requiring signal jamming in almost all cases. As mentioned above (cf. §8.1), once signal jamming is taking place, a vigilant user can identify it.

8.3 Car-sharing Scenarios

Car-sharing companies require additional ECUs to enable their users to unlock and lock their vehicles using the mobile application. There are several options to implement such behavior (e.g., using internet and API calls, mobile SMS); however, most of the time, that function works independently of the other ECUs in the vehicle. This means that even if the vehicle is locked through this ECU (i.e., via the mobile app), the original RKE system can still be used to unlock the vehicle, hence it is still vulnerable to RollBack. Therefore, for car-sharing companies, it would be worth "connecting" this additional mobile app-related ECU to the rest of the system and enabling the RKE system only if the vehicle is unlocked through the app (and disabling otherwise). However, this only protects the vehicle after it is returned. When someone, who is renting the vehicle, leaves it temporary in a parking lot adequately locked via the key fob but the rental is still ongoing, RollBack can still be launched.

[28] This can also justify that the battery has sufficient charge in the key fob.

9 Conclusion

Remote Keyless Entry (RKE) systems have been the target of attackers for a long time. Attacks such as jamming, tampering, and replaying captured key fob signals, have been quite common. Thus, since the late 1990s, deployments have implemented rolling code technology that, by invalidating all previous codes every time a button is pressed on the key fob, renders the attackers' job much more difficult. However, in 2015, RollJam was proven to break, in general, all rolling code-based systems. By carefully jamming, capturing, and replaying key fob signals, RollJam can always be one step ahead of the original key fob, letting an attacker unlock any vehicle. However, if the owner uses the key fob without the RollJam device being in operation (which requires careful placement to hidden spots on the vehicle, continuous control, etc.), the next (unlock) code the attacker possesses becomes invalidated thanks to the rolling codes.

Here, we developed `RollBack`, a new time-agnostic replay-and-resynchronize attack against today's most RKE systems. We showed that even though the one-time code becomes invalid in rolling code systems, replaying a few previously captured signals consecutively can trigger a rollback-like mechanism in the RKE system. `RollBack` is instruction-agnostic, meaning that any captured signals (irrespective of belonging to an unlock or lock instruction) can trigger the same behavior. Therefore, in a typical use case, `RollBack` does not require signal jamming at all. Furthermore, it is time-agnostic; signals have to be captured only once and can be replayed any time in the future as many times as desired.

We derived *four* different variants of `RollBack` w.r.t. the required number of signals to be captured, sequence, and time frame of the replay. Our limited yet ongoing analysis revealed that $\sim 70\%$ of the vehicles are vulnerable to a variant of `RollBack`. While most of the vehicles found vulnerable until this point are from Asian manufacturers, the impact is likely to be bigger worldwide.

As a countermeasure, we proposed several general advises for the vehicle owners on how they possibly avoid all types of signal jamming-based RKE attacks in different scenarios, including car-sharing use cases that are the most attractive targets to `RollBack`. However, since `RollBack` does not necessitate jamming and the root cause of the vulnerability is yet to be identified, adequate countermeasures and patches could not be proposed for the time being.

Acknowledgements

This research was supported by the National University of Singapore, NCS Group, and I^{2}R, A*STAR, Singapore. The authors would like to thank Xu Jia for his comments.

References

[1] A. Paul, R. Chauhan, R. Srivastava, and M. Baruah, "Advanced Driver Assistance Systems," SAE Technical Paper 2016-28-0223, https://bit.ly/3aJUEUz, Feb 2016 [Accessed: Jul 2022].

[2] Bosch, "Electronic Power Steering (EPS)," Online, https://bit.ly/2ZJNI7k, [Accessed: Jul 2022].

[3] CSS Electronics, "OBD2 Explained - A Simple Intro (2021)," Online, https://bit.ly/3pZeyn6, [Accessed: Jul 2022].

[4] J. Wang, Y. Shao, Y. Ge, and R. Yu, "A survey of vehicle to everything (v2x) testing," *Sensors*, vol. 19, no. 2, 2019. [Online]. Available: https://bit.ly/3vOtw0b

[5] M. Lake, "HOW IT WORKS; Remote Keyless Entry: Staying a Step Ahead of Car Thieves," New York Time post, https://nyti.ms/3DLnSyS, Jun 2001 [Accessed: Jul 2022].

[6] Embitel, “Electronic Control Unit is at the Core of All Automotive Innovations: Know How the Story Unfolded,” Blog post, https://bit.ly/3DNRCei, Jul 2017 [Accessed: Jul 2022].

[7] A. Greenberg, “Hackers Remotely Kill a Jeep on the Highway—With Me in It,” WIRED article, https://bit.ly/3AJLhjn, 2015 [Accessed: Jul 2022].

[8] Jmaxxz, “You Car is My Car,” Presentation at DEFCON 27, https://bit.ly/3peIzPu, Aug 2019.

[9] “FREE-FALL: HACKING TESLA FROM WIRELESS TO CAN BUS,” Presentation at BlackHat, https://bit.ly/3FZyRGx, 2017.

[10] D. F. Oswald, “Wireless attacks on automotive remote keyless entry systems,” in *Proceedings of the 6th International Workshop on Trustworthy Embedded Devices*, ser. TrustED ’16. New York, NY, USA: Association for Computing Machinery, 2016, p. 43–44. [Online]. Available: https://bit.ly/3pzxRmN

[11] K. Karnik, Manandeep, S. Kale, and A. Medhekar, “On vehicular security for rke and cryptographic algorithms: A survey,” *International Journal of Engineering Research and*, vol. 9, 2020.

[12] R. Verdult, F. D. Garcia, and J. Balasch, “Gone in 360 seconds: Hijacking with hitag2,” in *21st USENIX Security Symposium (USENIX Security 12).* Bellevue, WA: USENIX Association, Aug. 2012, pp. 237–252. [Online]. Available: https://bit.ly/3maFaPO

[13] R. Verdult, F. D. Garcia, and B. Ege, “Dismantling megamos crypto: Wirelessly lockpicking a vehicle immobilizer,” in *24th USENIX Security Symposium (USENIX Security 15).* Washington, D.C.: USENIX Association, Aug. 2015. [Online]. Available: https://bit.ly/3m6Elrb

[14] T. Eisenbarth, T. Kasper, A. Moradi, C. Paar, M. Salmasizadeh, and M. Manzuri, “Physical cryptanalysis of keeloq code hopping applications.” *IACR Cryptology ePrint Archive*, vol. 2008, p. 58, 01 2008.

[15] F. D. Garcia, D. Oswald, T. Kasper, and P. Pavlidès, “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: USENIX Association, Aug. 2016. [Online]. Available: https://bit.ly/3pwZKvV

[16] S. Kamkar, “Drive It Like You Hacked It: New Attacks and Tools to Wirelessly Steal Cars,” Presentation at DEFCON 23, https://bit.ly/3j0NZKc, Aug 2015.

[17] K. Marneweck, “An introduction to KeeLoq™ code hopping,” Microchip App Notes, https://bit.ly/3BVV5qs, 1996 [Accessed: Jul 2022].

[18] “The history of car technology,” https://bit.ly/3lFTHCK, accessed: 2021-08-06.

[19] L. Herbert, “90 firsts in american automotive history,” in *Popular Science.* Bonnier Corporation, 1964, pp. 81–83. [Online]. Available: https://bit.ly/3BIxee4

[20] R. Potter and P. Thomas, “Engine immobilisers: How effective are they?” https://bit.ly/3aC75l4, 2001, accessed: 2021-08-10.

[21] J. C. van Ours and B. Vollaard, “The engine immobiliser: A non-starter for car thieves,” *The Economic Journal*, vol. 126, no. 593, pp. 1264–1291, 2016. [Online]. Available: https://bit.ly/3pw0MIB

[22] A. Francillon, B. Danev, and S. Capkun, “Relay attacks on passive keyless entry and start systems in modern cars.” *IACR Cryptology ePrint Archive*, vol. 2010, p. 332, 01 2010.

[23] B. Santo, “The Consumer Electronics Hall of Fame: LiftMaster Garage Door Opener,” IEEE Spectrum, https://bit.ly/3BJ2Z6t, Oct 2019 [Accessed: Jul 2022].

[24] YourMechanic, “How Long Does a Key Fob Battery Last?” AutoBlog post, https://bit.ly/2T4oSw2, 2016 [Accessed: Jul 2022].

[25] NXP, “Advancing keyless entry/go,” NXP solutions brochure, https://bit.ly/3LGJyjB, 2013 [Accessed: Jul 2022].

[26] C. Toma, “Introduction to Ultimate KeeLoq™ Technology,” Microchip App Notes, https://bit.ly/3jjz79W, 2014 [Accessed: Jul 2022].

[27] Microchip, “KeeLoq™ Code Hopping Encoder,” Microchip HCS200, https://bit.ly/3GqCl5c, 2011 [Accessed: Jul 2022].

[28] A. Bogdanov, “Attacks on the keeloq block cipher and authentication systems,” in *In RFIDSec*, 2007.

[29] W. Aerts, E. Biham, D. De Moitié, E. De Mulder, O. Dunkelman, S. Indesteege, N. Keller, B. Preneel, G. A. E. Vandenbosch, and I. Verbauwhede, “A practical attack on keeloq,” *J. Cryptol.*, vol. 25, no. 1, p. 136–157, Jan. 2012. [Online]. Available: https://bit.ly/2Zj91Nv

[30] T. Eisenbarth, T. Kasper, A. Moradi, C. Paar, M. Salmasizadeh, and M. T. M. Shalmani, “On the power of power analysis in the real world: A complete break of the keeloqcode hopping scheme,” in *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008*, ser. Lecture Notes in Computer Science, vol. 5157. Springer, 2008, pp. 203–220. [Online]. Available: https://bit.ly/3jyIHG0

[31] M. Kasper, T. Kasper, A. Moradi, and C. Paar, “Breaking keeloq in a flash: On extracting keys at lightning speed,” in *Proceedings of the 2nd International Conference on Cryptology in Africa: Progress in Cryptology*, ser. AFRICACRYPT ’09. Berlin, Heidelberg: Springer-Verlag, 2009, p. 403–420. [Online]. Available: https://bit.ly/3Bec4Dg

[32] F. D. Garcia, D. Oswald, T. Kasper, and P. Pavlidès, “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: USENIX Association, Aug. 2016. [Online]. Available: https://bit.ly/3b4tU18

[33] Kevin2600 and W. Li, “Rolling Pwn Attack,” [Online], https://bit.ly/3czwTCw, July 2022 [Accessed: Jul 2022].

[34] T. Barrabi, “Honda key fob hack could leave all vehicle models since 2012 vulnerable: reports,” New York Post [Online], https://bit.ly/3b4364x, July 2022 [Accessed: Jul 2022].

[35] R. Stumpf, “I Tried the Honda Key Fob Hack on My Own Car. It Totally Worked,” security affairs [Online], https://bit.ly/3Os6dRt, July 2022 [Accessed: Jul 2022].

[36] P. Paganini, “Experts demonstrate how to unlock several Honda models via Rolling-PWN attack,” security affairs [Online], https://bit.ly/3RVusdZ, July 2022 [Accessed: Jul 2022].

[37] D. Goodin, “Meet RollJam, the $30 device that jimmies car and garage doors,” Blog post, https://bit.ly/2YKvmD8, 2015 [Accessed: Jul 2022].

[38] S. Gadgets, “HackRF One,” Online, https://bit.ly/3DCNmy6, [Accessed: Jul 2022].

[39] J. Pohl and A. Noack, “Universal radio hacker: A suite for analyzing and attacking stateful wireless protocols,” in *12th USENIX Workshop on Offensive Technologies (WOOT 18)*. Baltimore, MD: USENIX Association, 2018. [Online]. Available: https://bit.ly/3p56MYG

[40] RTL-SDR.com, “Quick Start Guide,” Online, https://bit.ly/3vJu1Zk, [Accessed: Jul 2022].

[41] Texas Instruments, “CC1101 - Low-Power Sub-1 GHz RF Transceiver,” Datasheet, https://bit.ly/3H7dYK7, 2021 [Accessed: Jul 2022].

[42] A. Greenberg, “This Hacker’s Tiny Device Unlocks Cars And Opens Garages,” WIRED article, https://bit.ly/3EedD6d, June 2015 [Accessed: Jul 2022].

[43] B. Toulas, “Hackers can unlock Honda cars remotely in Rolling-PWN attacks,” BleepingComputer News, https://bit.ly/3otJ8U4, Jul 2022 [Accessed: Jul 2022].

[44] Oak Lawn Toyota, “How to Program a Toyota Key Fob,” Online, https://bit.ly/3mxmnhH, [Accessed: Jul 2022].