Abstract—Proper security protocols should be considered as a critical requirement in today’s digital world to prevent adversaries from being able to gain access to valuable assets. Although numerous preventative mechanisms have been implemented, attackers adapt and develop new attack techniques to infiltrate existing technologies. The results of infiltration experiments revealed vulnerabilities in the radio frequency (RF) communication of cars and garages in remote keyless entry (RKE) systems as well. In this work, we present a timestamp-based solution to enhance the security of existing rolling-code based RKE system and demonstrate it through a prototype implementation.

Index Terms—remote keyless entry (RKE), replay attack, timestamp-based defense

I. INTRODUCTION

Remote Keyless Entry (RKE) system allows a user to gain entry to buildings and vehicles via wireless remotes. There are different approaches to implement RKE systems such as fixed-code, rolling-code, and passive keyless. Many of these implementations are vulnerable to attacks, however, they are still in use. The replay attack is easy-to-launch and the more popular attacks on these systems [1]. It involves capturing radio frequency (RF) signals using software-defined radio (SDR) and then replaying them to elicit a response such as unlocking a car or opening a garage door. It has been realized by car safety and security research groups that most car entry systems can be bypassed [2]. Therefore, it is critical to implement strong security protocols in RKE systems to protect precious items.

With time, manufacturers have enhanced the security of their RKE systems against replay attacks. For example, a rolling-code based RKE system offers better security than fixed-code based RKE [3]. It uses a random number generator to generate new codes in both transmitter and receiver, which are synchronized. These codes initially match, however, if the transmitter is pressed more than once the system will no longer be synchronized. The receiver has the ability to recognize future codes and re-synchronize again to overcome the situation. However, this method of re-synchronization has a security flaw [4]. Attackers use jamming and then replaying attack to bypass the security of rolling-code based RKE system.

In this paper, we propose a timestamp-based defense mechanism integrated with rolling-code to enhance RKE security. The timestamp will be a second-factor authentication added to the randomly generated code.

II. TIMESTAMP-BASED DEFENSE MECHANISM

In our defense mechanism, we have considered rolling code used in the RKE system, command for lock and unlock mechanism, and clock time. We have assumed that transmitter and receiver clocks are synchronized. We append rolling code, command, and clock time together, and the appended output is encrypted using Advanced Encryption Standard (AES) [5] with a 16-byte key length. The encrypted code is produced via a sequence of repeated steps. Each step is comprised of four functions: byte substitution, word rotation, mix columns, and round key generation. After each function, the output becomes the new state that the next function will operate on. Byte substitution replaces a byte value with a corresponding value from a look-up table. Word rotation is performed by arranging the state in a square matrix with column vectors corresponding to every four bytes in the state. Then, the rows of this matrix are rotated byte-wise for a specified amount for the round. Mixing the columns requires that the state is multiplied by a multiplication matrix over a Galois field. Finally, a round key is produced by XOR-ing the state with a section of an expanded 176-byte key, generated from the initial 16-byte key. The key is sent to the receiver as a code.

In our system, AES algorithm is implemented using a custom C++ class library. It contains a single AES class that has public functions to encrypt and decrypt data along with private data members and helper functions related to AES encryption and decryption process. The AES class is used to encrypt 16 bytes of data partitioned as such: 4 bytes for rolling code, 8 bytes for a command code to either lock or unlock mechanism, and 4 bytes for the timestamp. In our implementation, the rolling code is a numerical value that increments by one every time the transmitting circuit sends a signal. When evaluating the signal after decryption, the receiving circuit compares the rolling code to a variable containing the expected value of the rolling code. If the received code is less than the expected code, the system rejects the signal. If the received code is greater than the expected code but the difference between them is not greater than 256, the system will continue to evaluate the timestamp. This window of error is designed so that if a user mistakenly sends a signal from the transmitter while out of range of the receiver, the system will not lock the user out in subsequent attempts. The timestamp is recorded by the transmitter when sending
Fig. 1. Flowchart for timestamp-based defense in rolling-code based RKE system

- **Receive new_code**
  - value = R | C | T
  - T-100 ≤ T_{ref} ≤ T+100
  - No
  - Valid

- **AES_Decrypt(new_code)**
  - Command = C_{ref}
  - Yes
  - Desired Output

- **Transmit new_code**
  - C = Command
  - T = Current time
  - R = Rolling code
  - value = R | C | T
  - new_code = AES_Encrypt(value)

**a) Transmitter side logic**

**b) Receiver side logic**

Fig. 2. State transition diagrams for the Transmitter and Receiver

- **Transmit**
  - Sync
  - Sleep
  - Encrypt
  - Handshake

- **Receive**
  - Sync
  - Sleep
  - Decrypt
  - Handshake

**Transmitter FSM**

**Receiver FSM**

a signal using the on-board crystal oscillator. It is included in the data to be encrypted as discussed above. Similarly, the receiving circuit records a timestamp when receiving a signal, and after decryption, the system compares the two timestamps. If they are within a window of 100 seconds of each other, then the system accepts the signal and reads the command code to either lock or unlock the controlled mechanism. We have used a window of timestamp validity for each code sent from the transmitter to handle the clock drift between the transmitter and receiver. The receiver will invalidate any code that is received outside this time frame window. In the current implementation, we have considered a time frame of 100 seconds as the window of validity. The window could be adjusted depending on the use-case. Fig. 1 shows the flow charts of our implemented algorithms on the transmitter and receiver sides.

The timestamp integrated with rolling code enhance the security of RKE system as the system has two varying parameters: rolling code and clock time that make the system hard to be attacked.

III. BASIC PROTOTYPE IMPLEMENTATION

A. Prototyping the security enhanced RKE system

We have used two Arduino Uno [6] to implement a prototype of the RKE system. One is integrated with an FS1000A transmitter operating at RF 433 MHz to act as a transmitter. The second uses an FS1000A receiver operating at RF 433 MHz to act as a receiver. The purpose of transmitting signals between two Arduino systems was to create a RKE similar to a car locking system that uses rolling-code. A code that will change on rolling is sent from the wireless key fob to the receiver located at the car. Similarly, the receiver circuit built using the Arduino expects a set of codes to lock/unlock itself. The locking/unlocking will only occur when the code sent from the transmitter circuit matches one of the expected codes stored within the receiver circuit. Fig. 2 shows the state transition diagrams for the transmitter and receiver implemented in Arduino.

B. Building a prototype circuitry for car door locking system

The circuit in Fig. 3 is created to demonstrate the timestamp-based defense mechanism for rolling-code based RKE system. Fig. 3(a) shows the car locking system that we built. Fig. 3(b) shows the circuitry for the junction box.

The junction box circuitry can be separated into two sections. The left section represents a car locking system without the defense mechanism. The right section represents a car locking system with the defensive code implemented. The wires exiting the bottom of the junction box are connected to a 12-volt car battery that provides power to all the circuitry. The wires exiting the top of box are the key fob transmitter/receiver antenna, and the wires connected to the car door, respectively. The switches, on either side of the junction box, isolate the two systems to demonstrate the level of encryption each side uses. In the center of the left section is an aftermarket RKE kit. The key fob receiver receives the signal to lock, unlock, or start the car which is sent from the key fob. The received signal is wired to control the relays located in the top left. The right section is the prototype RKE system. This system includes the developed defensive code using an Arduino UnoTM, Buckboost power supply and four relays.

C. Capturing and Replaying the code

We need the SDR hardware and GNU Radio [7] to capture and replay the transmitted signal (code) from the transmitter to receiver. We have used LimeSDR [8] as the SDR in our work and interfaced it with GNU Radio software. Using Osomocom Source and File Sink blocks in the GNU Radio, the transmitted signal between the transmitter and receiver can be intercepted and stored in a file. Once the signal is captured, it can be replayed at any time to the receiver to unlock the car door using File Source and Osmocom Sink blocks. Fig. 4 shows the GNU Radio block diagrams for capturing and replaying the transmitted signals.
D. Evaluating the Defense Mechanism

As discussed our car door junction box circuitry consists of an aftermarket RKE and our secure RKE. We exclusively enable only one RKE system in the junction box. For the aftermarket RKE system, we captured the transmitted code and replayed it in a later time. We found that the replayed code was able to lock/unlock the car door. We repeated the same process of capturing and replaying the code with our secure RKE that integrates rolling-code with timestamp and encrypts them with AES. In this case, we observed that the replayed code was unable to lock/unlock the door. The receiver was expecting the code sent in the current time, but the replayed code was captured in time that does not fall in the valid time frame window. The timestamp mechanism on the transmitted signal successfully protected the code from being replayed. The prototype defended against a replay attack just as intended.

E. Power consumption and optimization

A major factor in determining the viability of our key fob prototype is its power consumption. It will determine the transmitter’s battery life. For the prototype to be competitive with current key fob technologies, the battery would need to last more than a year to reduce how often the consumer would need to change the battery for the transmitter. In the current implementation, we use Arduino Unos that draw a minimum of 15mA. However, the current draw can be reduced to 6.5mA and 4.5\(\mu\)A with 3.3V supply in awake and sleep modes, respectively with a few hardware and software tricks discussed in [9]. For our transmitter key fob, we are using a 2000mAh Lithium Ion battery. It will be the case that the transmitter will be mostly in sleep mode.

Assume that the transmitter will be 99% time in sleep mode and 1% time in awake mode. Then, the average current drawn: 
\[
I_{\text{avg}} = (0.99 \times 0.0045m\text{A}) + (0.01 \times 6.5m\text{A}) = 0.0694m\text{A}
\]

Battery Life = 2000mAh/I_{\text{avg}} \approx 28,818h \approx 3.28 \text{ years}

IV. Conclusion and Future Works

In this work, we reviewed and performed the replay attack in existing RKE systems. Using LimeSDR and GNU Radio, we were able to lock/unlock car door and garage systems with captured and replayed codes. To improve the security of RKE systems, we integrated a timestamp-based defense mechanism with the rolling-code and evaluated security. To demonstrate our prototype, we built a car door circuitry as well.

In future, we have plan to provide formal proofs for the robustness of our defense mechanism. We will build a portable
transmitter that involves packaging of the completed prototype into a form similar to that seen in the current car or garage keys. The transmitter will be housed inside a custom-designed 3D printed model. Further investigation of potential issues with the implementation of timestamp code like clock drifting will be researched as well.

REFERENCES


