Lightweight Secure Communication Protocols for In-Vehicle Sensor Networks

Miao Xu
Dept. of CSE
Univ. of South Carolina
Columbia, SC 29208
xum@email.sc.edu

Wenyuan Xu*
Dept. of CSE
Univ. of South Carolina
Columbia, SC 29208
College of EE
Zhejiang University
Hangzhou 310027, China
wyxu@cse.sc.edu

Jesse Walker
Intel Labs
Hillsboro, OR 97124
jesse.walker@intel.com

Benjamin Moore
Dept. of CSE
Univ. of South Carolina
moorebw@email.sc.edu

ABSTRACT

Wireless sensor networks are increasingly being integrated into the modern automobile with the intention of improving safety and reducing costs. However, it has been shown that the first widely-deployed and mandatory in-car sensor networks –Tire Pressure Monitoring Systems (TPMSs) – employ no cryptographic algorithms for protecting their wireless communication, incurring serious security and privacy risks for consumers. In this paper, we design and evaluate cost-effective secure communication protocols that can resolve the privacy and security vulnerabilities of existing TPMSs as well as other forthcoming in-car wireless sensor networks. Our implementation on Arduino platforms shows that the proposed protocol incurs modest overhead and is applicable to resource-constrained embedded systems.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*

Keywords

TPMS; tracking; spoofing; secure protocols

1. INTRODUCTION

The quest for increased safety and efficiency in automotive transportation systems are leading car makers to integrate wireless communication systems into automobiles. Mandated by laws in the United States [30] and European Union [7], the first widely-deployed wireless networks installed in every new vehicle are in-vehicle sensor networks: Tire Pressure Monitoring Systems (TPMSs). Although TPMSs were intended to increase overall road safety and fuel economy, recent research [20] shows that the wireless communication protocols of TPMSs introduce security and privacy risks, making it possible to track drivers, to trigger false low tire pressure warnings, and even to damage the automobile hardware. To cope with the risks, we design secure communication protocols that suit the energy and computational budget of tire pressure sensors.

TPMSs continuously measure air pressure inside all tires of passenger cars, trucks, and multipurpose passenger vehicles, and alert drivers if any tire is significantly underinflated or overinflated. Most automobiles are equipped with *direct TPMSs*. These systems rely on battery-powered pressure sensors inside each tire to measure tire pressure and communicate their data via a radio frequency (RF) transmitter. The receiving tire pressure control unit, in turn, analyzes the data and can send results or commands to the central car computer over the Controller Area Network (CAN), e.g., to trigger a warning message on the vehicle dashboard. In this paper, we focus on direct TPMSs because of their popularity and known-vulnerabilities.

Essentially, the vulnerabilities of direct TPMSs stem from the lack of cryptographic mechanisms. TPMS communications are based on standard modulation schemes and simple protocols. Hence, they can be reverse-engineered, making eavesdropping feasible. With a 32-bit immutable identifier of each tire sensor being transmitted in every message, TPMSs render sensor IDs sufficiently unique to track cars. Moreover, the implementation of the in-car system appears to fully trust all received messages, and shows no evidence of basic security practices, such as input validation. Therefore, spoofing attacks are made possible and can cause TPMSs to malfunction.

Given the constrained resources of sensors and the need of being user-friendly, it is impractical to design a bullet-proof protocol to secure TPMS communications. For instance, tires along with the in-tire sensors are expected to be replaced by mechanics with little computer knowledge. Any sophisticated initial key establishment protocols are unlikely to be operable. Asymmetric encryption algorithms and even some symmetric encryption algorithms are beyond the computational capabilities of sensors. Thus, we propose a secure communication protocol that uses a lightweight hardware

*Corresponding author.

CyCAR'13, November 4, 2013, Berlin, Germany.

ACM 978-1-4503-2487-8/13/11 ...$15.00.
http://dx.doi.org/10.1145/2517968.2517973.

block cipher –Katan32– as the basic building block. We use it to build a hash function, eliminating the need to add extra hardware of a hash function. To balance the trade-off between security and sensor constraints, we divide the TPMS data reporting into sessions, and assign a key to each session to protect TPMS packets. Before the number of encryptions under the same session key reaches an alarming level, the assigned session key will be updated. Our secure communication protocol consists of three phases: 1.) master key distribution, 2.) session key update, and 3.) secure TPMS packet reporting. Accordingly, two types of TPMS messages are used in the secure protocol: 1.) TPMS key update messages which establish session keys between the sensors and the ECU, and 2.) TPMS data messages which report the tire status such as its temperature and pressure.

The rest of the paper is organized as below. We begin in Section 2 by presenting the architecture and communication schemes of existing TPMSs. In Section 3, we describe our threat models, design requirements, and hardware assumptions for a lightweight secure TPMS protocol. We present the secure TPMS protocol that contains three phases in Section 4. The Arduino-based implementation details and performance evaluation results are provided in Section 5. Finally, we end this paper with related work in Section 6 and concluding remarks in Section 7.

2. THE TPMS BACKGROUND

Our goal is to design a TPMS secure communication protocol that requires few changes to the existing one. Thus, we first introduce the architecture and communication protocols of existing direct TPMSs in this section.

The TPMS architecture. A typical direct TPMS contains the following components: TPMS sensors (fitted into the back of the valve stem of each tire), a TPMS electronic control unit (ECU), a receiving unit (either integrated with the ECU or stand-alone), a dashboard TPMS warning light, and one or four antennas (connected to the receiving unit). Figure 1 depicts a four-antenna configuration. The TPMS sensors periodically broadcast pressure and temperature measurements together with their identifiers (IDs). The TPMS ECU/receiver receives the packets and performs a few simple operations (e.g., ID-based filtering), and then illuminates the low-tire-pressure warning light if the reported tire pressure is abnormal. An ECU is typically better equipped than sensors, as it is powered by car batteries and could contain a more powerful microprocessor.

Communication protocols. The communication protocols used between sensors and an ECU are proprietary. From supplier websites and marketing materials, however, we learned that TPMS communications are bidirectional with each direction using different RF techniques: the periodic data packets sent by sensors, and the activation signals transmitted by an ECU to sensors.

Sensors commonly transmit at the 315 MHz or 433 MHz bands (UHF) and employ ASK (Amplitude Shift Keying) or FSK (Frequency Shift Keying) modulation. Prior work discovered that two popular brands of tire pressure sensors employ the differential Manchester encoding scheme [20]. Each tire pressure sensor carries an identifier (ID), which has to be entered to the ECU either manually or automatically during tire installation. Afterwards, the sensor ID becomes the key

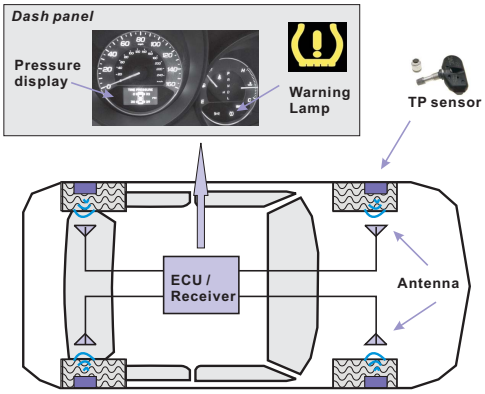

Figure 1: The TPMS architecture with four antennas.

information to determine the origin of the data packet and to filter out packets transmitted by other vehicles.

To prolong the battery life, tire pressure sensors are designed to sleep most of the time, but can be waken up in response to an RF activation signal. Activation signals are designed for diagnosis scenarios and the initial sensor ID binding phase, when sensors are required to transmit their IDs or other measurements. The RF activation signals operate at 125 kHz in the low frequency (LF) radio frequency band and can only wake sensors within a short range, due to the generally poor characteristics of RF antennas at that low frequency. According to manuals from different tire sensor manufacturers, the activation signal can be either a tone or a modulated signal. In either case, the LF receiver on the tire sensor filters the incoming activation signal and wakes up the sensor only when a matching signal is recognized.

3. SYSTEM ASSUMPTIONS

In this section, we articulate the capabilities of adversaries, and outline the design requirements of TPMS communication protocols.

3.1 Threat Models

We consider an *outsider* TPMS adversary, who has no physical access to the inside of automobiles and can only launch an attack from distance. Therefore, we consider the ECU and the tire pressure sensor hardware to be secure against physical attacks. In addition, we assume adversaries have enough communication knowledge [20] to reverse engineer RF signals, so that they can learn the modulation and encoding schemes. In cases that TPMS messages are transmitted in plaintext, adversaries can capture the messages and retrieve all information in them, such as sensor IDs and tire pressure readings. They can also craft and transmit arbitrary messages (including false information). With the capabilities of eavesdropping and spoofing TPMS messages, adversaries could perform the following attacks.

Tracking. Since automobiles have become essential elements of our social fabric — they allow us to commute to and from work; they help us take care of errands like shopping and taking our children to day care — tracking automobiles presents substantial risks to location privacy. Adversaries may be highly interested in wireless tracking of cars, at least for traffic monitoring purposes. Several entities are using mobile toll tag readers [3] to monitor traffic flows. Tracking through the

TPMSs offers greater 'benefit' to adversaries because the use of TPMSs is not voluntary and they are hard to deactivate.

Spoofing Attacks. The openness of wireless communication makes it easy to launch message spoofing attacks because no physical connection is required. The spoofing attacks can result in various dangerous consequences. For example, spoofing a low tire pressure readings can lead to a dashboard warning and will likely cause the driver to pull over and inspect the tire. It has been reported that highway robbers make drivers pull over by puncturing the car tires [21] or by simply signaling a driver that a tire problem exists. With the help of existing TPMSs, the robbery may stop a car without performing physical activities.

Replay Attacks. Similar to spoofing attacks, wireless communication makes replay attacks easier than its wired counterpart. An adversary could record an earlier TPMS message and transmit it later. These attacks require few communication knowledge. A simple RF recording and transmitting unit (e.g., software defined radio) suffices the attack. Replay attacks could be used to trigger the low-pressure warning light, or to cause a system to malfunction.

Battery-Drain Attacks. Tire pressure sensors are expected to last longer than the lifespan of a tire. An adversary could continuously transmit activation signals, which wake sensors and trigger them to respond for several times. As a result, the lifetime of sensors are reduced.

We assume that an adversary does not intend to launch a targeted attack against a specific automobile, e.g., following one car for an extended period while eavesdropping the wireless communication. Instead, we assume that an adversary launches an attack in public locations (e.g., in parking lots, on the road) with the goal of eavesdropping all passing-by cars. In addition, the adversary cannot access private places (e.g., automobile factories, car dealerships), since private locations can be protected physically (e.g., restricting an area from public access). When a car is left in the parking lot unattended, an adversary could have abundant time to launch their attacks. While traveling on the road, an adversary may only have a brief access to the cars, making the attack window small. Regardless of the attacking scenarios, an adversary could conduct the aforementioned attacks. Granted that there are other attacks that can harm the reliability of TPMSs, e.g., jamming attacks, coping with jamming attacks is out of the scope of this paper, and we refer readers to known strategies [25, 38] to defend against jamming attacks.

3.2 Requirements for Secure TPMS Protocols

The secure TPMS communication protocol has to satisfy the following requirements.

Low Cost. One of the key factors that determines the success of the secure communication protocols is the amount of required extra hardware for supporting the secure protocol. Any communication protocols that impose excessive manufacturing cost increase or battery life decrease are unlikely to be accepted by the profit-driven automobile industry. Therefore, the secure solution should consider the manufacturing, computation and communication costs.

Notation	Explanation
id_s	The original identifier of a sensor
id_n	The pseudo identifier of a sensor
id_x	The index of a sensor (e.g. 1 for left front sensor)
K_m	The master key
K_s	The session key
K_a	The authentication key

Table 1: Summary of notations.

User-friendly. In order to achieve the security goals, the secure communication protocols will require key establishment every time a new tire pressure sensor is installed. This could happen when a tire is replaced by a mechanic with little security knowledge, and thus the key establishment should require little human intervention to avoid complicated tire maintenance.

Secure while unattended. While it is economic infeasible to design a bullet proof system, we aim to address random mischief in the public, especially public parking lots. While left in a parking lot unattended, the ECU may be powered off but the tire pressure sensors, operating on batteries, can be awoken. Thus, the design has to ensure that no adversary can tamper with the security of the system while an ECU is off.

3.3 Hardware Assumptions

To support the secure communication protocols, we consider that both the tire pressure sensors and the TPMS units have the following hardware components.

1. **Non-volatile memory.** A sensor is equipped with non-volatile memory for storing information that will be used for communications, such as a master symmetric key, the sensor ID, and the status counters (e.g., the latest observed sequence number).
2. **A hardware-oriented block cipher.** We assume a hardware-based block cipher is available as the building block for the cryptographic communication protocol. A block cipher takes a fixed length of message and maps it to a cipher according to a symmetric key. A good candidate of hardware-oriented block cipher is the KATAN family, which has been widely used in resource-constrained devices such as RFID tags [12].
3. **A random number generator (RNG).** We assume that both the sensors and the ECU contain a hardware based random number generator (RNG). The RNG can harvest randomness hidden in the inter-arrival time of messages from the radio channel, the tire pressure, the RSSI over time, the acceleration etc. After a RNG has taken a few measurements and accumulated enough entropy, it generates the random number. We note it is also possible to create a pseudo RNG using the hardware-oriented block cipher. In particular, a secret key can be used as a seed of the RNG, and encrypting a counter with the secret key based on `Katan32` creates a random number.

Note that a hash function is a standard function required in many cryptographic protocols and so does our protocol. However, we didn't mandate a hardware-oriented hash function, considering that a hash function typically requires more

complicated hardware than a block cipher. Instead, we create a hash function by using the block cipher (details in Section 4.1). In summary, except for the non-volatile memory which is available in existing TPMSs, the secure communication protocol requires a block cipher and a random number generator as its building blocks.

4. PROTOCOL DESCRIPTION

4.1 Overview

The goal of the lightweight secure communication protocol for TPMSs is to protect the communication between TPMS sensors and an ECU from tracking risks, spoofing attacks, replay attacks, and battery-drain attacks at modest costs. To cope with all threats, we propose to use symmetric cryptographic algorithms. Particularly, to balance the trade-off between security and costs of the cryptographic protocols, we let each sensor and the ECU share a master key (K_m), and divide the entire lifetime of a TPMS into sessions. At the beginning of each session, a new session key (K_s) is derived to protect the TPMS communication onwards. Thus, the lightweight secure communication protocol operates in three phases: master key distribution, session key update, and secure TPMS packet reporting, as illustrated in Figure 2. Since the TPMS sensors are designed to report data when the vehicles travel at a speed higher than 40km/h [20], we define each session as a driving trip, e.g., the session starts when a vehicle is ignited and ends when the engine shuts down. The proposed secure scheme utilizes a hardware-oriented block cipher, pseudo IDs, message authentication code (MAC), and linear feedback shift register (LFSR).

Block cipher. We use `Katan32` [12] as our block cipher. KATAN is a family of hardware-oriented lightweight block ciphers with optional block sizes of 32, 48, and 64 bits and 80-bit keys. `Katan32` can be implemented in about 1000GE[1]. The KATAN family is primarily designed for resource constrained platform such as RFID devices and other embedded systems. Even though it has a small block size, `Katan32` can provide a high security level for applications that generate only small amounts of data. The KATAN family has been the subject of extensive cryptanalysis, and is known to have high immunity to differential, linear, and algebraic attacks; no other known attacks are more effective than the brute force attacks, which is commonly considered as the most inefficient attacks. In summary, the 32-bit KATAN variant minimizes the hardware size and power requirements without compromising security.

Pseudo ID. Essentially, the ease of vehicle tracking is because TPMS packets contain a fixed sensor ID and are transmitted in plaintext, making the linkage between vehicles and their sensor IDs trivial to obtain. The sensor IDs cannot be simply eliminated because each message must convey a sensor ID that allows the ECU to identify the message source. To cope with the tracking risk, a naive approach is to encrypt the entire TPMS message. However, using `Katan32`, which encrypts a 32-bit block, to encrypt an entire message requires to execute `Katan32` at least three times (the packet size is more than 64 bits). To reduce the computation cost,

[1]A gate equivalent (GE) is a unit of measure that specifies a digital circuit area. In terms of a hardware-based block cipher, GE describes the complexity of the block cipher [33].

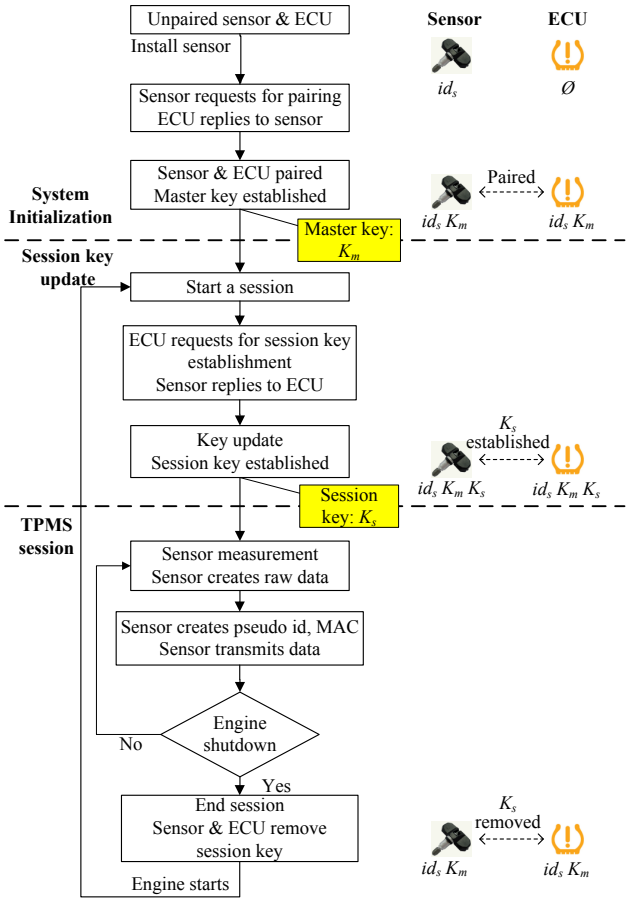

Figure 2: Overall flowchart of the lightweight secure TPMS communication protocol.

we use a fresh pseudo ID in each message from which the ECU can easily identify the right sensor. The pseudo ID is computed by hashing a well-known IV (initialization vector) under the sensor id and a 14-bit counter value that we name `seq`, which will be discussed in the next bullet.

To avoid the cost of a hardware-oriented hash function, we use the block cipher (e.g., `Katan32`) to perform the hash function. `Katan32` takes an 80-bit key and a 32-bit IV and outputs the 32-bit cipher. It serves the purpose well because it satisfies the requirements of hash functions, i.e., one way and collision resistant. One way means that it is computationally infeasible to find `x` given `hash(x)`, and collision resistant means that given `hash(x)` it is computationally infeasible to find `x'` such that `hash(x') = hash(x)`. The pseudo ID can be calculated using Algorithm 1, i.e.,

$$id_n := \texttt{pseudo-id}(id_s, seq),$$

where seq is a sequence number calculated using an LFSR. In Algorithm 1, the symbol '||' denotes concatenation, and we use Merkle-Damgard construction [6] for padding. The Merkle-Damgard-construction-based padding starts with a '1' bit followed by several '0' bits, and ends with the bit length of the original message [27]. For example, in our case, the padding is a '1' bit followed by 25 zeros (denoted by $0b10^{25}$) and concatenated by 0x2E (i.e., the bit length of 32-bit sensor ID plus 14-bit LFSR). IV should be a well-known

Name	Parameters	Explanation
`Katan32`(key, m)	key: an 80-bit key m: a 32-bit message block	A 32-bit version block cipher from the KATAN family.
`K-MAC`(key, m)	key: an 80-bit key m: a message whose length is padded to be multiples of 32	A function that returns a 32-bit CBC-MAC calculated by the `Katan32`-based block cipher.
`lfsr`(seq)	seq: a 14-bit sequence number	A function of linear feedback shift register
`pseudo-id`(id_s, seq)	id_s: an original sensor id seq: a 14-bit sequence number	A function that returns the pseudo identifier for a sensor id_s.
`gen-key`(K_m, t, app)	K_m: master key. t: flag, where AU for creating authentication key K_a and SE for session key K_s. app: a string for generating keys.	A function that generates an 80-bit key, either a session key K_s or an authentication key K_a.
`rand`()	None	A random number generator which creates a 64-bit pseudo-random number.
`trunc`(m, n)	m: a message whose length is larger than n n: the desired message length	An auxiliary function that returns the first n-bit of the message m.

Table 2: Summary of function notations used in this paper.

initialization vector, preferably with a Hamming weight of a half of the block cipher's block size[2].

LFSR as sequence numbers. To prevent replay attacks, we propose to use a linear feedback shift register (LFSR) to create a sequence number that appears to be random yet does not require excessive bookkeeping. Monotonically increased sequence numbers are commonly used to prevent replay attacks because only the largest sequence number needs to be remembered. However, such a sequence number is predictable (i.e., increased by 1 each time) and could be used to track devices [26]. Completely random sequence numbers require storing all observed sequence numbers, too costly to the sensors. An LFSR is a register initialized to a non-zero value and its output is a linear function of its previous state. For an n-bit LFSR that is initialized to a non-zero value, it will have a period of $2^n - 1$ and appear to be random (uniformly distributed). Thus, it is predicable by insiders yet appears to be random to outsiders, which is ideal for TPMS communications.

An LFSR can be implemented efficiently. Particularly, it is a register incremented via a clocking routine. The clocking routine computes a value of one bit b by XORing together selected bits, shifting all of the bits in the register one bit to the right, and inserting the newly computed bit b into the hole left by the shift. The bits are selected according to the non-zero coefficients of a primitive polynomial that has a degree equal to the bit length of the register. For example, to create a 14-bit sequence number, a 14-bit clock function can be used for LFSR, and an example primitive polynomial is $X^{14} + X^{10} + X^6 + X + 1$, which has a period of $2^{14} - 1 = 16383$.

MAC. To cope with spoofing attacks, a message authentication code (MAC) will be added to each TPMS packet. In particular, we adopt a cipher block chaining message authentication code (CBC-MAC) using `Katan32`. CBC-MAC requires that all messages MACed under the same key have exactly the same length. Otherwise there is a fairly easy forgery attack called a length extension attack [5]. Since there are two types of TPMS messages (i.e., key update messages and data messages, as illustrated in Figure 3.) need to be protected by CBC-MAC and they have different length, two keys are used: a session key (K_s) for data messages and an authentication key (K_a) for key update messages. For the rest of the paper, we denote the `Katan32`-based CBC-MAC as `K-MAC`.

We summarize the frequently-used notations in Table 1 and functions in Table 2. We discuss protocols for each phase in the following sections.

[2]In our case, we choose IV to be 0xC5365C6A, whose Hamming weight is sixteen, i.e., a half of the block size of `Katan32`.

4.2 Session Key Update

Every time a vehicle is ignited, the ECU will initiate a session key update. The session key (K_s) will be used to generate a MAC for each TPMS data message for the entire session. In a rare case when the number of messages encrypted using the same session key is so large that it jeopardizes the security of the block cipher, the ECU will trigger another session key update.

4.2.1 Protocol Description

The complete session key update procedure consists of a three-way handshake, and it is initiated by the ECU. We summarize the protocol in Table 3 and depict the message format in Figure 3(a). Each message contains a `Cmd` field to indicate its format, a `Tire-idx` field to indicate the sensor index id_x, and a `Pseudo-id` field to represent the pseudo identifier id_n. Note that id_x is a 3-bit number that is used to identify the location of the sensors For instance, id_x of the left front tire sensor is 1, id_x of the right front one is 2, etc.

Before establishing any session key, both the ECU and the TPMS sensors have agreed and stored a master key K_m in their non-volatile memory, which is discussed in Section 4.4, and the ECU is aware of the sensor IDs.

Algorithm 1 `pseudo-id`(id_s, seq): generating a pseudo id.

Input:
id_s: the original sensor ID
seq: a 14-bit sequence number
Output:
id_n: the pseudo id
Procedure:
1: padding := $0\text{b}10^{25}$ || 0x2E
2: $key := id_s$||seq||padding
3: $id_n :=$ `Katan32`$(key, \text{IV}) \oplus \text{IV}$
4: **return** id_n

Session Key Update				
1.	ECU	:	$seq \leftarrow \texttt{lfsr}(seq)$;	ECU generates pseudo id, random number etc., and concatenates them to form a request message
			$id_n \leftarrow \texttt{pseudo-id}(id_s, seq)$;	
			$R \leftarrow \texttt{rand}()$	
			$M_1 \leftarrow \texttt{REQ}\|\|id_x\|\|id_n\|\|seq\|\|R$	
2.	$ECU \rightarrow N$:		M_1	ECU transmits request message to sensor N
3.	N	:	if (id_n!=$\texttt{pseudo-id}(id_s, seq)$) quit;	Sensor verifies pseudo id, and quits if failed.
			$S \leftarrow \texttt{rand}()$	Otherwise generates a random number and MAC to form
			$mac \leftarrow \texttt{K-MAC}(K_a, seq\|\|S\|\|R)$	a reply message
			$M_2 \leftarrow \texttt{REP}\|\|id_x\|\|id_n\|\|seq\|\|S\|\|R\|\|mac$	
4.	$N \rightarrow ECU$:		M_2	Sensor replies to ECU's request with a message M_2
5.	ECU	:	if ($\texttt{isDifferernt}(id_x, id_n, seq, R)$) quit;	ECU verifies the consistency of the received R, seq, id_x
			if ($mac! = \texttt{K-MAC}(K_a, seq\|\|S\|\|R)$ quit;	and id_n, and verifies the MAC.
			$mac \leftarrow \texttt{K-MAC}(K_a, seq\|\|R\|\|S)$	If all are successful, ECU reorders information and creates M_3
			$M_3 \leftarrow \texttt{CON}\|\|id_x\|\|id_n\|\|seq\|\|R\|\|S\|\|mac$	
6.	$ECU \rightarrow N$:		M_3	ECU transmits message M_3
7.	N	:	if ($\texttt{isDifferernt}(id_x, id_n, seq, R, S)$) quit;	Sensor verifies the consistency of the received S, R, seq, id_n, and id_x, and verifies mac. If any of them fails, sensor ignores M_3 and quits.
			if $mac! = \texttt{K-MAC}(K_a, seq\|\|R\|\|s)$ quit;	
Session Key Generation				
8.	ECU	:	$K_s \leftarrow \texttt{gen-key}(K_m, SE, R\|\|S)$	ECU generates session key
	N	:	$K_s \leftarrow \texttt{gen-key}(K_m, SE, R\|\|S)$	N generates session key

Table 3: Session key establishment between an ECU and a sensor N. Both the ECU and the sensor have agreed on the master key K_m before the first session key update.

- **The ECU request:** Once a vehicle is ignited, the ECU will initiate a session key update. First, it derives the authentication key (K_a) based on the master key (K_m)(see Algorithm 2):

  $$K_a := \texttt{gen-key}(K_m, AU, null),$$

  where AU is a flag that indicates the generation of an authentication key. The ECU calculates a new sequence number using the LFSR and generates a 32-bit pseudo identifier id_n (see Algorithm 1):

  $$seq := \texttt{lfsr}(seq)$$
  $$id_n := \texttt{pseudo-id}(id_s, seq)$$

  Then, the ECU generates a 64-bit random number R, and creates a message:

  $$M_1 := \texttt{REQ}||id_x||id_n||seq||R,$$

  where REQ is a flag indicating that it is the first request message for updating the session key, id_x is a 3-bit number indexing the sensors belonging to this vehicle. For instance, id_x of the left front tire sensor is 1, id_x of the right front one is 2, etc.

  ECU $\rightarrow$ Sensor:

  $$M_1$$

- **The sensor reply:** The sensor checks whether id_x is intended to itself and whether seq is a subsequent value of what it has observed. If either of the conditions does not hold, it discards M_1 and quits session key update. Otherwise, the sensor verifies id_n using seq contained in the message, and discards it if the verification fails. If the verification succeeds, the sensor generates a random number S and a message:

  $$M_2^{'} := \texttt{REP}||id_x||id_n||seq||S||R,$$

  where REP indicates this is a reply from the sensor. Finally, the sensor derives K_a if it is removed from its memory and calculates MAC using K_a:

  $$mac := \texttt{K-MAC}(K_a, seq||S||R)$$

  Sensor $\rightarrow$ ECU:

  $$M_2 := M_2^{'}||mac$$

- **The ECU confirmation:** The ECU examines id_x, id_n, R and seq in message M_2. If any of the fields is different from its previous message, it discards the message and quits. Otherwise, the ECU verifies whether the received mac equals $\texttt{K-MAC}(K_a, seq||S||R)$. If the verification succeeds, the ECU generates a confirmation message:

  $$M_3^{'} := \texttt{CON}||id_x||id_n||seq||R||S$$

  and a MAC

  $$mac := \texttt{K-MAC}(K_a, seq||R||S)$$

  ECU $\rightarrow$ Sensor:

  $$M_3 := M_3^{'}||mac$$

  Finally the ECU generates the new session key using Algorithm 2:

  $$K_s := \texttt{gen-key}(K_m, SE, R||S),$$

  where SE is a flag that indicates the generation of a session key.

- The sensor verifies id_x, id_n, seq, R, and S in the message M_3, and examines whether the received mac equals $\texttt{K-MAC}(K_a, seq||R||S)$. If the verification succeeds, the

sensor derives the new session key in the same way as the ECU.

$$K_s := \texttt{gen-key}(K_m, SE, R||S)$$

4.2.2 Security Analysis

The three-way session key update protocol can prevent the risks of tracking, replay attacks, spoofing attacks, and greatly reduces the risks of battery drain attacks, given that adversaries are unaware of the master key, the sensor IDs, or the polynomial of the LFSR. First, the use of pseudo identifiers and LFSR sequence numbers makes it difficult to track a vehicle based on TPMS communication messages. Second, replay attacks have little impact because of the sequence number. The sensors will ignore the session key update request, if its sequence number has been used before. Third, the construction of the request message reduces the risk of battery drain attacks, because a sensor will transmit a reply only if the sequence number is fresh and the pseudo identifier is correct. Note that in some sense, the pseudo identifier in the first request message serves as a MAC to prevent spoofing attacks.

4.3 Protecting TPMS Data Messages

In each TPMS session, tire pressure sensors report their measurements periodically, e.g., once every 60 seconds. The current packet format in a TPMS includes a sensor ID field, fields for temperature and tire pressure, fields for various warning flags, and a CRC checksum. Transmitted in plaintext, the current TPMS data packets are vulnerable to tracking-related privacy breach, spoofing attacks, and replay attacks. To prevent these attacks, we enhance the TPMS data packet with a pseudo ID and a sequence number that appears to be random. When a sensor plans to transmit a message p that contains tire pressure, temperature, and flags, it first calculates a new sequence number using the LFSR, and derives a new pseudo ID:

$$seq := \texttt{lfsr}(seq) \quad (1)$$

$$id_n := \texttt{pseudo-id}(id_s, seq) \quad (2)$$

After calculating the MAC:

$$mac := \texttt{K-MAC}(K_s, seq||p),$$

where p is the data payload, the sensor constructs a TPMS data message, as illustrated in Figure 3(b). Similar to the format of TPMS key update messages, the first flag `cmd` is used to indicate whether this is a data message or a session key reply message. The 3-bit `Tire-idx` is used to indicate the location of the sensor. The CRC checksum is preserved to detect bit errors.

Algorithm 2 `gen-key`(K_m, t, app): derive an 80-bit key.

Input:
K_m: master key
t: a 32-bit flag, AU for K_a and SE for K_s
app: an appendix string. It could be $null$, i.e., an empty string

Output:
key: an 80-bit key

Procedure:
1: $ctr := 1$ % a 32-bit integer
2: $k_0 := 0$ % a 32-bit integer
3: $k_1 := \texttt{K-MAC}(K_m, t||ctr||k_0||app)$
4: $k_2 := \texttt{K-MAC}(K_m, t||ctr+1||k_1||app)$
5: $k_3 := \texttt{K-MAC}(K_m, t||ctr+2||k_2||app)$
6: $key := \texttt{trunc}(k_1||k_2||k_3, 80)$
7: **return** key

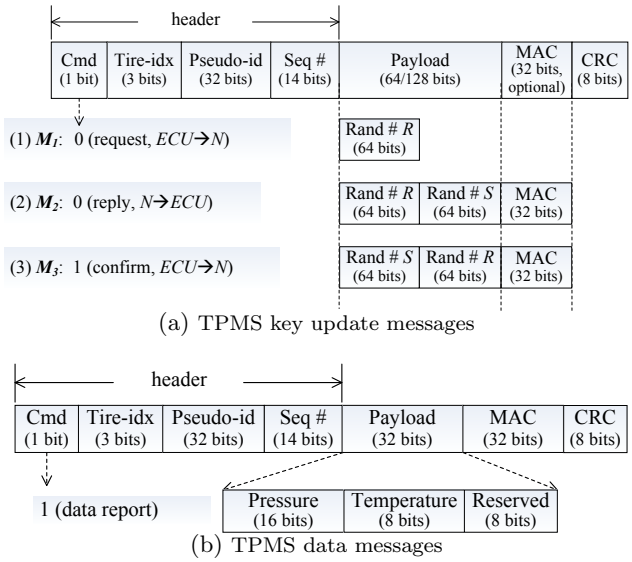

Figure 3: Packet format of two types of TPMS messages: (a) TPMS key update messages for session key establishment, and (b) TPMS data messages for tire status report. Note that the size is not proportional to real packet fields.

Overhead. Overhead is a major concern for sensors. Each message requires one encryption to derive a pseudo ID, and a few encryptions to calculate the MAC. To reduce the overhead, the MAC is calculated on the sequence number and data payload, excluding pseudo ID. The protocol is secure, because the pseudo ID is bounded with the sequence number (Algorithm 1), and containing the sequence number in the MAC is equivalent to including pseudo IDs. As a result, for a 32-bit payload and a 14-bit sequence number, only two encryptions are needed to calculate the MAC using a 32-bit block cipher. In total, only three encryption are enough for each TPMS data message.

Why 14-bit sequence number? Most block ciphers are subject to birthday attacks, leading to compromised keys, when $2^{k/2}$ encryptions are performed under the same key for a block cipher with block size k [8]. `Katan32`, as a 32-bit block cipher, can have up to 2^{16} messages before rendering the current session key insecure. Hence, we select the sequence number to be 14 bits. Once the sequence number is exhausted, the session key has to be updated. This way, we limit the maximum allowed number of messages that can be sent under the same session key K_s to be 2^{14}. Considering that a sensor generates a TPMS data message once every 60 second, it will take $2^{14}/(60 \times 24) = 11.4$ days to exhaust the sequence numbers, much larger than a typical driving duration.

4.4 Master Key Establishment

The goal of master key (K_m) establishment is to distribute it securely and correctly between a sensor and an ECU. The

K_m-establishment should require little human intervention since an ECU may need to be paired with different sensors multiple times as a consequence of tire replacement. When designing the protocol, we ask the following questions: (1) Who should initiate the K_m-establishment? (2) How should K_m be distributed? (3) How to verify that K_m is correctly distributed?

Who? The K_m establishment should be initiated by a sensor not an ECU. Vehicles are frequently left in public parking lots unattended, whereby sensors can receive messages while an ECU is powered off. If sensors are allowed to passively accept K_m establishment request, then adversaries can impersonate ECUs and distribute a fake K_m when a vehicle is parked publicly. Thus, we envision that sensors will initiate the K_m establishment procedure when they are installed. For example, when a tire is first inflated, the associated sensor can start the K_m establishment. Similarly, an ECU cannot just passively accept the K_m establishment, but it is only allowed to enter K_m establishment mode after being triggered by events that are less likely to happen. For instance, an event could be pressing breaks while switching on and off the left-turn and right-turn lights multiple times. Alternatively, an ECU could be equipped with a hidden infrared receiver that can accept commands sent from a device inside the vehicles.

How? We envision that K_m is exchanged in a factory or at a dealership during servicing. Both factories and dealerships can be considered as trustworthy sites, because those private sites can enforce physical protection against adversaries. Although a determined adversary may bypass physical protection of the sites, the paper focuses on casual adversaries that are unwilling to endeavor to launch attacks.

There are a few ways to generate and distribute K_m. For example, K_m can be generated and stored in the tire pressure sensor during manufacturing, and a scannable bar code that maps to K_m is attached to the tire pressure sensor. When installing a tire in a dealership, a mechanic will scan the bar code, then deliver K_m to the ECU through the OBD port using a handheld device. Such a device is already available for entering the sensor IDs for existing TPMS. Alternatively, after a sensor initiates the K_m-establishment request, an ECU can generate a K_m and deliver it over the 125kHz channel. Such a channel is already available in existing TPMS and exhibits a small communication range due to the mismatched antenna.

Key Verifying. It is essential to ensure that both a sensor and an ECU have the correct K_m. Thus, the K_m-establishment is finalized with a key verification procedure. The sensor and the TPMS ECU will derive their authentication key,

$$K_a := \texttt{gen-key}(K_m, AU, null).$$

Then, they will perform a session key update. After a successful three-way message exchange, both the sensor and the ECU verify the K_m and the sensor ID (id_s). In case K_m is not exchanged correctly, the system will display error messages. The key exchange process will repeat until K_m is correctly verified.

Discussion. In general, the security level is the most important metric that should be considered for a block cipher. But when it comes to a resource-constrained platform such as TPMS sensors, the performance is extremely important and determines the viability of the protocol. In this paper, we propose to use `Katan32` because it has the advantage of both high security level and compact hardware size. However, we point out that our secure TPMS communication protocol is independent of the choice of block ciphers. For those TPMS sensors that have already been in the market, software-oriented block ciphers such as KLEIN [15] can be installed by upgrading the firmware without changing the hardware. Therefore, the proposed secure TPMS protocol can be deployed on both existing sensors and future sensors.

Figure 4: Equipment used for experiments: [left] customized TPMS platform (Arduino + RFM22), [right] ATEQ VT55 TPMS handheld device.

5. IMPLEMENTATION AND EVALUATION

5.1 Implementation

Due to the lack of programmable TPMS sensors at the time of writing this paper, we implement the block cipher `Katan32` and session key update procedure on our customized TPMS platform, shown as in Figure 4. The customized platform employs an Arduino Uno [1] board and an RFM22 [4] radio transceiver. Arduino is an open-source electronic prototyping board and serves as the computation component in our customized platform. RFM22 is a low cost radio transceiver that supports transmission frequency at 315MHz or 433MHz. In addition, we acquired a handheld device (ATEQ VT55 [9]), which can decode information from a variety of tire sensor implementations. The tool is commonly used by car technicians and mechanics for troubleshooting. For our experiments, we used it to verify our customized Arduino-based TPMS platform.

5.1.1 Transmission of TPMS Packets

TPMS sensors transmit signals at 4.9kbps and utilize the differential Manchester encoding [20]. However, our RFM22 transceiver does not support this encoding scheme, and only supports basic Manchester encoding or Non-Return Zero (NRZ) encoding schemes. To overcome the challenge and make the physical layer parameters of RFM22 compatible with a TPMS sensor, we configured the RFM22 to operate in NRZ encoding scheme, and added a middle layer called *N2M overlay* (NRZ-to-Differential Manchester overlay, see Figure 5) between the MAC layer and the application layer to convert the NRZ encoding into differential Manchester encoding. The NRZ encoding uses a high level to code a bit '1' and a low level to code a bit '0'. In comparison, Manchester encoding uses a transmission to represent a bit:

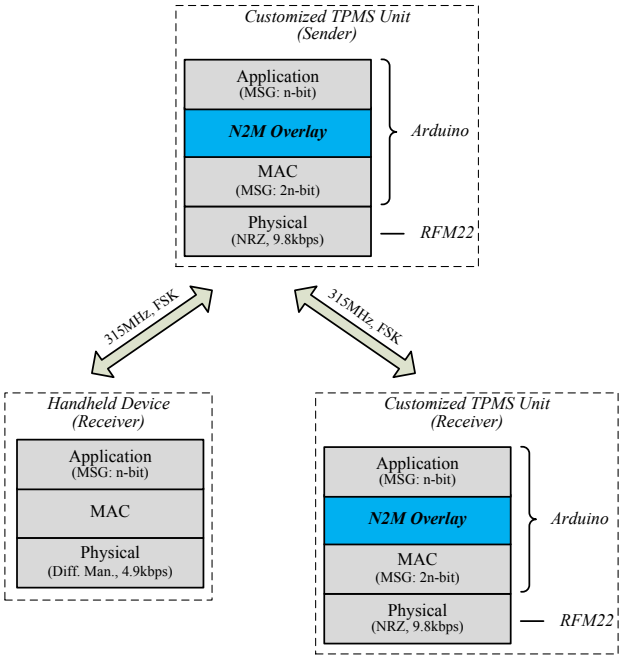

Figure 5: The protocol stack of customized TPMS unit. An N2M overlay is added between the application layer and the MAC layer on the Arduino-based TPMS platform.

e.g., a raising transmission represents a bit '1'. Thus, two bits ('01') encoded with NRZ emulate a raising transition in Manchester encoding (i.e. one bit). For instance, in order to emulate a TPMS message of '1010', the middle layer will create a message of '10100101' for the RFM22 radio to transmit under the NRZ encoding. Since Manchester encoding needs twice the clock speed of NRZ's to achieve the same bit rate, we set the data rate of RFM22 twice of the sensor's, i.e., at a transmission rate of 9.8kbps. Finally, we disable the build-in CRC generation on the RFM22 transceiver and use the middle layer to create an intermediate bit strings. Figure 6 summarizes the conversion of TPMS message transmission between our Arduino-based platform and an off-the-shelf handheld device. In summary, we implemented a middle layer to pre-process the bit stream before pushing it to the transceiver.

We selected a brand of TPMS sensors that use the same modulation scheme as the RFM22 transceiver, i.e., frequency shift keying (FSK), and emulated the TPMS sensors by constructing a packet following the same message format. Our experiments showed that the handheld device can correctly decode the packets sent by the Arduino-based TPMS platform and display the tire pressure reading and the sensor ID, as shown in Figure 7 (a). This confirms the compatibility of our customized TPMS platform with a real TPMS.

5.1.2 *Block Cipher* – `Katan32`

We use `Katan32` as the block cipher. `Katan32` is a family of hardware-oriented lightweight block ciphers and can be implemented using programmable hardware, such as field-programmable gate array (FPGA). Using Arduino platform, we are unable to implement `Katan32` in hardware but software. Figure 7 (b) depicts the effectiveness of the block

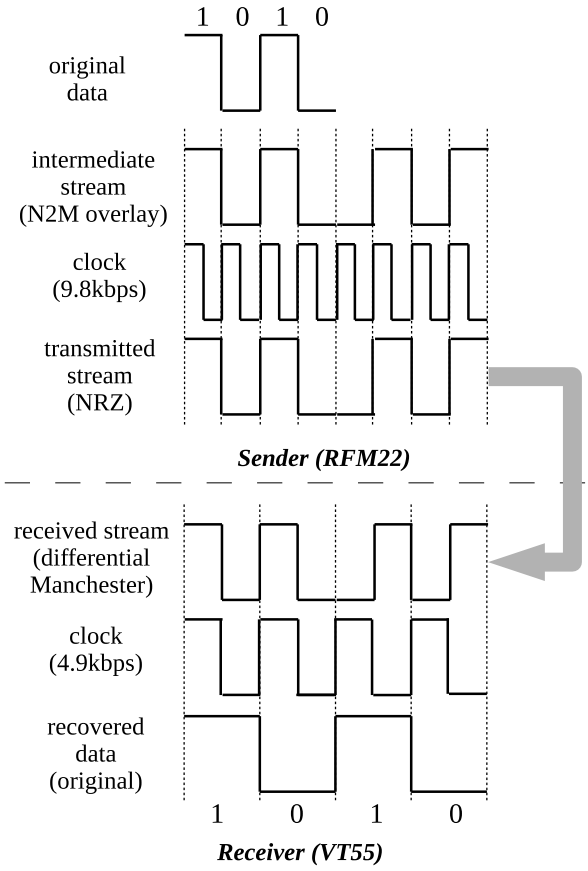

Figure 6: Illustration of TPMS message conversion between the Arduino-based platform and off-the-shelf handheld devices.

cipher. After employing the security-enhanced TPMS messages by applying the block cipher, the handheld device can no longer identify the sensor ID.

5.1.3 *Session Key Update*

The session key update process is launched every time a car is ignited. The ECU sends a request to each tire pressure sensor and waits for their replies. If one or more sensors do not reply to the request before timeout, the ECU will re-transmit the request for several times before concluding that sensors are missed. In this case, the ECU will notify the driver with the situation (e.g., missing sensor(s)). For those sensors that reply to the request, the ECU will continue with the session key update, as discussed in Section 4.2.

5.2 Performance Evaluation

We quantified the performance of the lightweight secure TPMS communication protocol. In particular, we focused on the performance of two phases: TPMS data session and the session key update. For each phase, we evaluated the delay[3] and the power consumption. The results are summarized in Table 4 and discussed below.

[3]Since the RFM22 radio automatically generates a predefined 9-byte preamble that cannot be disabled, all packet transmission in our experiments contains the extra preambles and the measured delay also contains the preambles.

(a) (b)

Figure 7: Illustration of the effectiveness of the secure communication protocol. (a) The existing TPMS: both Arduino-based platform and ATEQ VT55 can read the TPMS message, indicating the vulnerabilities of eavesdropping; (b) the secure TPMS: the Arduino-based platform can correctly identify the sensor ID, and accept the message, but VT55 failed to identify the sensor ID, and showed "WAITING FOR RF", indicating the secure protocol can prevent casual eavesdropping.

TPMS Data Session. To evaluate the processing delay introduced by the secure TPMS protocol on TPMS data transmission, we first conducted an experiment as follows: we run the cryptograpic algorithm (i.e., `Katan32`) 1000 times over a 32-bit message on our customized TPMS platform. The result shows that for one block message, the encryption takes an average of 3.3ms and the decryption takes an average of 4.0ms. We then conducted another experiment to compare the difference of energy consumption and transmission time between existing TPMS messages and secure TPMS messages, which is depicted in Figure 8. The result shows that in both scenarios, the radio transmission consumes more energy than the data processing, and thus increasing the message size imposes more energy consumption than adding a block cipher. The performance comparison between existing TPMS data messages and secure TPMS data messages, in terms of the delay and the energy consumption, is summarized in Table 4. The total delay is 23.5ms and 36.6ms respectively in existing TPMS and secure TPMS. Similarly, the energy consumption is 9826uJ and 15231uJ respectively. This indicates that the secure TPMS protocol incurs a modest overhead.

Session Key Update. Each time a car is ignited, a session key update procedure will be performed. Therefore, it is important to know the performance of the session key update. In particular, we are interested in the delay and energy consumption at the sensors' side. To verify this, we conducted the experiment as follows: the sensor recorded a timestamp when it started to receive the request message M_1, and recorded another timestamp once K_s was derived. Our result shows that the interval between two timestamps is 156.9ms. We also measure the power during the session key update, which is plotted in Figure 9 and summarized in Table 4. The power consumption for session key update is 61722uJ, which is similar as transmitting four data messages.

In summary, the secure TPMS protocol introduces modest overhead yet effectively prevents the vehicle tracking, message spoofing attacks, and replay attacks.

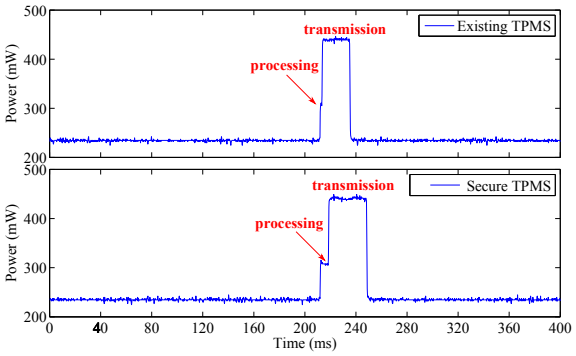

Figure 8: Comparison between different TPMS message transmissions: the top plot shows the transmission of existing TPMS messages (plaintext), and the bottom one shows the transmission of secured TPMS messages (protected by pseudo IDs and MAC.

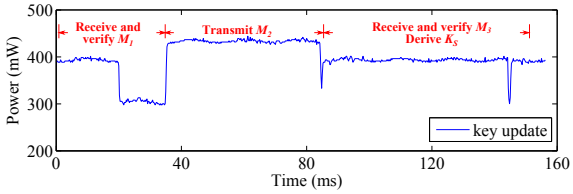

Figure 9: Performance of session key update.

6. RELATED WORK

Wireless devices have become an inseparable part of our social fabric. As such, much effort has been dedicated to analyze their privacy and security issues. Devices being studied include RFID systems [23,28,36], mass-market UbiComp devices [34], household robots [13], implantable medical devices [19], and in-vehicle sensor networks [20]. Instead of analyzing the security and privacy of existing systems, we focus on designing a secure communication protocol that can address the vulnerabilities of the existing systems.

Securing wireless sensor networks has been studied extensively. Perrig *et al.* [32] have proposed a suite of security protocols to provide data confidentiality and authentication for resource-constrained sensors. In terms of key distribution, Eschenauer *et al.* [14] proposed a random key predistribution scheme where each sensor node randomly stores a subset of keys out of the entire possible key space. With a high probability, any two neighboring nodes will share at least one key, and the shared key is used to establish pairwise keys between sensors on demand. Chan *et al.* [11] further proposed a q-composite random key pre-distribution scheme

where any two neighboring nodes share q common keys instead of just one. TESLA [31] is a well-known authentication protocol for verifying a broadcast sender in wireless sensor networks. Authenticating a broadcast sender typically requires to use asymmetric cryptographic algorithms (e.g., RSA), which is not suitable to resource-constrained sensor networks. TESLA utilizes symmetric cryptographic algorithm, yet achieves asymmetric properties by leveraging temporal difference. Those key management schemes cannot work well with TPMSs, since sensor networks are concerned with establishing keys among a large number of sensors while TPMSs focus on establishing keys between four sensors and the ECU only. Thus, our secure communication protocol requires a less amount of resources (e.g., storage or calculating power).

Location privacy in wireless networks has attracted much attention, since wireless devices are known to present tracking risks through explicit identifiers in protocols or identifiable patterns in waveforms. In the area of WLAN, Gruteser *et al.* [18] demonstrated that one can identify a user's location through link- and application-layer information. A common countermeasure against breaching location privacy is to frequently dispose user identity. For instance, Jiang *et al.* [22] proposed a pseudonym scheme where users change MAC addresses each session. Similarly, Greenstein *et al.* [16] have suggested an identifier-free mechanism to protect user identities, whereby users can change addresses for each packet. In cellular systems, Lee *et al.* have shown that the location information of roaming users can be released to third parties [24], and proposed using the temporary mobile subscriber identifier to cope with the location privacy concern. IPv6 also has privacy concerns caused by the fixed portion of the address [29], and thus the use of periodically varying pseudo-random addresses has been recommended. Due to the communication overhead and bandwidth requirement, these schemes are not efficient enough to be applied to in-vehicle sensor networks. To prevent tracking the RFID tags via their IDs, Weis *et al.* [37] proposed to create a randomized pseudo identifier using a one-way hash function when the tags respond to each query. In comparison, our scheme has a lower requirement, since it re-uses a hardware block cipher to calculate pseudo IDs. The block cipher can be implemented with less amount of resources than a hash function.

Several pairing schemes have been proposed to associate devices and setup a master key between them. Most pairing schemes utilize a channel with a limited eavesdropping range to exchange paring messages. For instance, Bluetooth [2] uses out-of-band channels such as inputs from a keypad or Near Field Communication (NFC) for device pairing. Stajano *et al.* [35] proposed a solution, called imprinting, that pairs two devices by exchanging a shared secret via a physically-linked electrical contact. Several schemes [10, 17] utilize movement patterns to establish secrets between devices. *Noisy tags* achieve a secure pairing by using a special tag. This special tag generates noises while the to-be-paired tag is transmitting the secret. Since the RFID reader can remove the noise generated by the special tag, it can receive the secret while an eavesdropper cannot decode the secret. Similar to these protocols, our secure communication protocol also exchange a master key over a channel that is difficult to eavesdrop on. In addition, our protocol has to ensure that unauthorized pairing cannot be performed when a vehicle is left in a public parking lot.

	TPMS message trans.		Session key update
	Existing	Secured	
Delay	23.5ms	36.6ms	156.9ms
Power	9826uJ	15231uJ	61722uJ

Table 4: Summary of performance evaluation: TPMS message transmission and session key update.

7. CONCLUSION

Tire Pressure Monitoring Systems (TPMSs) are the first in-car wireless network integrated into all new cars in the US and in the EU. Prior work shows security and privacy vulnerabilities of existing TPMSs. To cope with the vulnerabilities, this paper presents a lightweight secure communication protocol. The protocol only requires adding a hardware-oriented block cipher and a random number generator. The secure protocol utilizes pseudo IDs, a linear feedback shift register, `Katan32`, and a message authentication code to effectively cope with the risks of car tracking, spoofing attacks, replay attacks and battery drain attacks. We have implemented the secure communication protocol on the Arduino platform, and our performance evaluation of the protocol shows that the protocol is light enough to be implemented on sensors.

8. REFERENCES

[1] Arduino. http://www.arduino.cc.

[2] Bluetooth. https://www.bluetooth.org/en-us/specification/.

[3] Portable, solar-powered tag readers could improve traffic management. Available at http://news.rpi.edu/update.do?artcenterkey=1828.

[4] RFM22. http://www.hoperf.com.

[5] Stop using unsafe keyed hashes, use HMAC. http://rdist.root.org/2009/10/29/stop-using-unsafe-keyed-hashes-use-hmac/.

[6] *Introduction to Modern Cryptography*. Chapman and Hall/CRC Press, 2007.

[7] Improving the safety and environmental performance of vehicles. *EUROPA-Press Releases* (23rd May 2008).

[8] Abdalla, M., and Bellare, M. Increasing the lifetime of a key: A comparative analysis of the security of re-keying techniques. In *Proceedings of the 6th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology* (2000), pp. 546–559.

[9] ATEQ VT55. http://www.tpms-tool.com/tpms-tool-ateqvt55.php.

[10] Castelluccia, C., and Mutaf, P. Shake them up!: a movement-based pairing protocol for cpu-constrained devices. In *Proceedings of the 3rd international conference on Mobile systems, applications, and services* (2005), ACM, pp. 51–64.

[11] Chan, H., Perrig, A., and Song, D. Random key predistribution schemes for sensor networks. In *SP '03: Proceedings of the 2003 IEEE Symposium on Security and Privacy* (2003), IEEE Computer Society, p. 197.

[12] De Canniere, C., Dunkelman, O., and Knežević, M. Katan and ktantan - a family of small and efficient hardware-oriented block ciphers. In *Cryptographic Hardware and Embedded Systems-CHES 2009*. Springer, 2009, pp. 272–288.

[13] Denning, T., Matuszek, C., Koscher, K., Smith, J. R., and Kohno, T. A spotlight on security and privacy risks with future household robots: attacks and lessons. In *Ubicomp '09: Proceedings of the 11th international conference on Ubiquitous computing* (2009), pp. 105–114.

[14] Eschenauer, L., and Gligor, V. D. A key-management scheme for distributed sensor networks. In *Proceedings of the 9th ACM conference on Computer and communications security* (2002), ACM, pp. 41–47.

[15] Gong, Z., Nikova, S., and Law, Y. W. Klein: a new family of lightweight block ciphers. In *RFID. Security and Privacy*. Springer, 2012, pp. 1–18.

[16] Greenstein, B., McCoy, D., Pang, J., Kohno, T., Seshan, S., and Wetherall, D. Improving wireless privacy with an identifier-free link layer protocol. In *Proceeding of Mobile systems, applications, and services (MobiSys)* (2008), ACM, pp. 40–53.

[17] Groza, B., and Mayrhofer, R. Saphe: simple accelerometer based wireless pairing with heuristic trees. In *Proceedings of the 10th International Conference on Advances in Mobile Computing & Multimedia* (2012), MoMM '12, pp. 161–168.

[18] Gruteser, M., and Grunwald, D. A methodological assessment of location privacy risks in wireless hotspot networks. In *Security in Pervasive Computing, First International Conference* (2003), pp. 10–24.

[19] Halperin, D., Heydt-Benjamin, T. S., Ransford, B., Clark, S. S., Defend, B., Morgan, W., Fu, K., Kohno, T., and Maisel, W. H. Pacemakers and implantable cardiac defibrillators: Software radio attacks and zero-power defenses. In *Proceedings of IEEE Symposium on Security and Privacy* (2008), IEEE Computer Society, pp. 129–142.

[20] Ishtiaq Roufa, R. M., Mustafaa, H., Travis Taylora, S. O., Xua, W., Gruteserb, M., Trappeb, W., and Seskarb, I. Security and privacy vulnerabilities of in-car wireless networks: A tire pressure monitoring system case study. In *19th USENIX Security Symposium, Washington DC* (2010).

[21] Italy. *http://aglobalworld.com/international-countries/Europe/Italy.php*.

[22] Jiang, T., Wang, H. J., and Hu, Y.-C. Preserving location privacy in wireless lans. In *MobiSys '07: Proceedings of the 5th international conference on Mobile systems, applications and services* (2007), ACM, pp. 246–257.

[23] Koscher, K., Juels, A., Brajkovic, V., and Kohno, T. EPC RFID tag security weaknesses and defenses: passport cards, enhanced drivers licenses, and beyond. In *Proceedings of the 16th ACM conference on Computer and communications security* (2009), pp. 33–42.

[24] Lee, C.-H., Hwang, M.-S., and Yang, W.-P. Enhanced privacy and authentication for the global system for mobile communications. *Wireless Networks 5*, 4 (1999), 231–243.

[25] Li, M., Koutsopoulos, I., and Poovendran, R. Optimal jamming attacks and network defense policies in wireless sensor networks. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE* (2007), IEEE, pp. 1307–1315.

[26] Li, Q., and Trappe, W. Light-weight detection of spoofing attacks in wireless networks. *IEEE International Conference on Mobile Adhoc and Sensor Systems Conference* (2006), 845–851.

[27] Merkle, R. C. Secrecy, authentication, and public key systems.

[28] Molnar, D., and Wagner, D. Privacy and security in library RFID: issues, practices, and architectures. In *Proceedings of Computer and communications security* (2004), ACM Press, pp. 210–219.

[29] Narten, T., Draves, R., and Krishnan, S. RFC 4941 - privacy extensions for stateless address autoconfiguration in IPv6, Sept 2007.

[30] of Transportation National Highway, D., and Administration, T. S. 49 cfr parts 571 and 585 federal motor vehicle safety standards; tire pressure monitoring systems; controls and displays; final rule. *http://www.tireindustry.org/pdf/TPMS_FinalRule_v3.pdf*.

[31] Perrig, A., Canetti, R., Tygar, J. D., and Song, D. The tesla broadcast authentication protocol.

[32] Perrig, A., Szewczyk, R., Wen, V., Culler, D., and Tygar, J. D. Spins: security protocols for sensor networks. In *MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking* (2001), ACM, pp. 189–199.

[33] Rolfes, C., Poschmann, A., Leander, G., and Paar, C. Ultra-lightweight implementations for smart devices–security for 1000 gate equivalents. In *Smart Card Research and Advanced Applications*. Springer, 2008, pp. 89–103.

[34] Saponas, T. S., Lester, J., Hartung, C., Agarwal, S., and Kohno, T. Devices that tell on you: privacy trends in consumer ubiquitous computing. In *Proceedings of USENIX Security Symposium* (2007), USENIX Association, pp. 1–16.

[35] Stajano, F., and Anderson, R. The resurrecting duckling: security issues for ubiquitous computing. *Computer 35*, 4 (2002), 22–26.

[36] Weis, S. A., Sarma, S. E., Rivest, R. L., and Engels, D. W. Security and privacy aspects of low-cost radio frequency identification systems. In *Security in Pervasive Computing* (2004), vol. 2802 of *Lecture Notes in Computer Science*, pp. 201–212.

[37] Weis, S. A., Sarma, S. E., Rivest, R. L., and Engels, D. W. Security and privacy aspects of low-cost radio frequency identification systems. In *Security in pervasive computing*. Springer, 2004, pp. 201–212.

[38] Xu, W., Ma, K., Trappe, W., and Zhang, Y. Jamming sensor networks: attack and defense strategies. *Network, IEEE 20*, 3 (2006).