Toward Trusted Wireless Sensor Networks

WEN HU
CSIRO ICT Centre
HAILUN TAN
University of New South Wales
PETER CORKE
Queensland University of Technology
WEN CHAN SHIH
CSIRO ICT Centre
and
SANJAY JHA
University of New South Wales

This article presents the design and implementation of a trusted sensor node that provides Internet-grade security at low system cost. We describe trustedFleck, which uses a commodity Trusted Platform Module (TPM) chip to extend the capabilities of a standard wireless sensor node to provide security services such as *message integrity, confidentiality, authenticity*, and *system integrity* based on RSA public-key and XTEA-based symmetric-key cryptography. In addition trustedFleck provides secure storage of private keys and provides platform configuration registers (PCRs) to store system configurations and detect code tampering. We analyze system performance using metrics that are important for WSN applications such as computation time, memory size, energy consumption and cost. Our results show that trustedFleck significantly outperforms previous approaches (e.g., TinyECC) in terms of these metrics while providing stronger security levels. Finally, we describe a number of examples, built on trustedFleck, of symmetric key management, secure RPC, secure software update, and *remote attestation*.

Categories and Subject Descriptors: C.2.0 [**Computer-Communication Networks**]: General—*Security and protection (e.g., fire walls)*

Part of this work was published in *Proceedings of the 6th European Conference on Wireless Sensor Networks (EWSN)* [Hu et al. 2009]. The extension of to the EWSN article is the research on the idea of trusted sensor networks, and the capabilities evaluation of remote attestation. There are significant modifications in Section 3.2, 4.1, 4.2, 4.3.1, 4.3.3, 5.3, 5.4, 5.5, and 6.5.
Authors' addresses: W. Hu and W. C. Shih, CSIRO ICT Centre, 1 Technology Court, Pullenvale, QLD 4069, Australia; email: {wen.hu, teddy.shih}@csiro.au; P. Corke, Queensland University of Technology; email: peter.corke@ieee.org; H. Tan and S. Jha, School of Computer Science and Engineering, The University of New South Wales, Sydney 2032, Australia; email: {thailun, sanjay}@cse.unsw.edu.au.

DOI 10.1145/1806895.1806900 http://doi.acm.org/10.1145/1806895.1806900

General Terms: Security, Design, Performance, Measurement

Additional Key Words and Phrases: Wireless sensor networks, public key (PK), RSA, TPM, remote attestation, trusted computing

ACM Reference Format:

Hu, W., Corke, P., Shih, W. C., Tan, H., and Jha, S. 2010. Toward trusted wireless sensor networks. ACM Trans. Sensor Netw. 7, 1, Article 5 (August 2010), 25 pages.
DOI = 10.1145/1806895.1806900 http://doi.acm.org/10.1145/1806895.1806900

1. INTRODUCTION

Wireless sensor network (WSN) applications [Dinh et al. 2007; Hu et al. 2005; Mainwaring et al. 2002; Szewczyk et al. 2004; Wark et al. 2007] are growing, yet security and privacy remain largely ignored in reported deployments. Algorithms such as those for over-the-air programming, while useful to researchers and network operators, generally leave the door "wide open" for injection of malicious code. While the importance of security and privacy is generally agreed upon, the problem seems to be considered impractical to solve given the limited computational and energy resources available at the node level. For commercial deployment, privacy, authenticity, and confidentiality *will be required* for applications such as monitoring resource utilization (for billing purposes of water or power), as well as for the remote management of WSN and upgrading of software (such as over-the-air reprogramming of application images). The hard-learned lesson from the PC industry is that ignoring security at the outset leads to huge pain when the technology becomes ubiquitous. In this article we present *trustedFleck* (see Figure 1), which increases very strong (internet-grade) security services through the use of a low-cost commodity coprocessor that is practical to add (in terms of physical size, memory usage, and energy consumption) to a sensor node.

Symmetric (shared) key algorithms are tractable on mote-class hardware and can achieve message confidentiality. However, key distribution and management remain significant practical challenges, and these algorithms poorly support message authenticity and integrity. On the Internet, public key cryptography (PKC) is widely used to support symmetric session key management, as well as message authenticity and integrity. Researchers have investigated methods to support PKC technology in WSN [Watro et al. 2004; Liu and Ning 2008]. Such approaches have focused on software-based PKC technologies, such as Rivest-Shamir-Adelman (RSA) and Elliptic Curve Cryptography (ECC) but the performance has been poor given the low clock rate and memory availability. Consequently, a smaller RSA public exponent (e) and a shorter key size are chosen, which seriously compromises the security level of PKC encryption.

Multihop over-the-air programming (MOAP) protocols such as Deluge [Hui and Culler 2004] enable application users to program and reprogram WSNs easily, but it also opens the door for unauthorized users to implant malicious code into the WSN. Current approaches [Dutta et al. 2006] focus on how to enforce program authenticity and integrity by introducing software-based RSA PKC, but again with a small RSA public exponent (e) and a short key size—resulting in some, but weak, security.

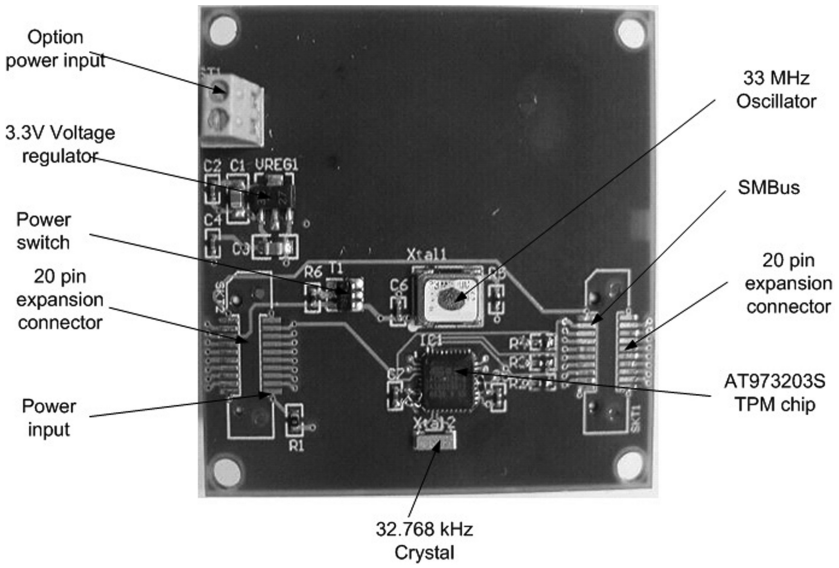

Fig. 1. Prototype trustedFleck TPM module (upper side) implemented as an expansion board for the Fleck WSN node. The board is 50 × 50m but the main component, the TPM chip, is just 6.1 × 9.7mm.

Recently, Francillon and Castelluccia [2008] showed that, when the bootloader (MOAP) is enabled, it is possible to exploit the buffer overflow vulnerabilities in TinyOS [Levis et al. 2005], and it is possible to permanently inject malicious code over the air into the Harvard architecture MicaZ mote[1] program flash. The attacker can then gain control of a large number of nodes and create botnets, which are collections of compromised nodes running software under a common command and control infrastructure. Further, Goodspeed [2007] has demonstrated how to inject malicious code byte-by-byte and execute the code in a Von Neumann architecture (MSP 430) Telosb mote.

As WSNs transition from research tools to production, it is important for the WSN owner to have the capabilities of controling application image reprogramming and verifying the code running in sensor nodes. trustedFleck provides a set of trust primitives based on platform configuration register (PCR) operations on the Trusted Platform Module (TPM), which enables remote attestation in sensor networks. A sink can thus challenge the system configuration of a remote node and find out whether the node has been compromised. The sink can then seek remedies (e.g., reprogram the node) if the node fails the challenge.

In this article, we introduce the design and implementation of trustedFleck, a sensor platform that uses trusted platform module (TPM) hardware to augment the node. Our extensive evaluation shows that trustedFleck provides Internet-level PKC and remote attestation services with reasonable energy consumption and financial overhead. The contributions of this article include the following:

[1]MicaZ motes and Flecks, the sensor nodes used in this article, have similar microcontrollers (Atmega 128; see Section 4.1).

—The design and implementation of the trustedFleck platform, which includes a standard TPM chip and a set of software primitives, to support public key cryptography (PKC), and remote attestations in a WSN. To the best of our knowledge, trustedFleck is the first platform that supports most RSA-based PKC (encryption, decryption, signature, and signature verification) and remote attestation (PCR_extend, PCR_read, PCR_quote, PCR_verifyQuote) primitives in a WSN. RSA-based PKC provides e-commerce-level security on the Internet, and trustedFleck brings this level of security to WSNs.

—Extensive evaluation of the trustedFleck platform in terms of computation time, energy consumption, memory footprint and cost. The results demonstrate the feasibility of the trustedFleck platform, and show that trustedFleck significantly outperforms previous approaches, such as TinyECC, in terms of computational time and memory usage, while providing stronger security levels.

—A demonstration that the trustedFleck platform is easy to use through implementation case studies for remote attestation, key management, secure software update, and secure remote procedure calls (RPC) services.

The rest of this article is organized as follows. We present related work in Section 2. In Section 3, we give a brief overview of the RSA algorithm and the concept of *trusted computing*, which trustedFleck is based on, followed by a detailed description of the software and hardware architecture of trustedFleck (Section 4). We evaluate the performance of trustedFleck in terms of computational time, energy consumptional and financial cost in Section 5. Section 6 describes, by means of a case study, how the trustedFleck primitives can be used to implement state-of-the-art remote attestation, key management, secure RPC, and a secure software update protocol, and improve the protocol's performance. Finally, we conclude the article in Section 7.

2. RELATED WORK

In this section, we provide a brief overview of secure communications most directly relevant to trustedFleck.

RSA is the most widely used PKC on the Internet, and a comprehensive guide to RSA is available in Rivest et al. [1978]. RSA is much slower than the eXtended Tiny Encryption Algorithm (XTEA) [Needham and Wheeler 1997] and other (shared) symmetric cryptography such as TinySec [Karlof et al. 2004].

It is our thesis that most of the secure communications in resource-constrained WSNs will be based on symmetric cryptography. However, due to the vulnerability of nodes, an effective symmetric key establishment and management scheme is of prime importance. RSA and Diffie Hellman key agreement techniques [Goldwasser 1997] are widely used key agreement protocols on the Internet, but have been previously considered infeasible for WSNs because of the resource constraints of sensor devices.

Researchers have proposed a limited version of RSA PKC (TinyPK) that performs encryption operations only and uses smaller RSA parameters such as public exponents and key sizes [Watro et al. 2004]. However, the security

levels of RSA cryptography are severely compromised by using smaller public exponents and key sizes. Recently, the importance of symmetric key cryptography and the critical roles of the key management mechanism in WSNs was observed by Nilsson et al. [2008], who proposed an efficient symmetric key management protocol for WSNs. However, they focused on the protocol design and formal verification, and did not address the resource constraints in implementing the protocol. MiniSec provides confidentiality and authenticity for both unicast and broadcast messages by applying a nonce (a combination of packet number and time information) to the message payload before encrypting them by the Offset Code Book (OCB)[Luk et al. 2007]. However, MiniSec assumes that a share key is kept safe at all nodes. Therefore, MiniSec still needs an effective key establishment and management scheme.

The research community is developing faster and more energy efficient PKC algorithms such as Tiny Elliptic Curve Cryptography TinyECC [Liu and Ning 2008] for resource-impoverished WSNs. While TinyECC shows the most promise to run at usable speeds on WSN nodes [Gurn et al. 2004], there are concerns related to patents, which are one of the main factors limiting the wide acceptance of ECC. In this regard, we note that the RSA and XTEA algorithms used in this work are both in the public domain. Further, there is no remote attestation support in TinyECC.

While multihop over-the-air programming (MOAP) protocols [Hui and Culler 2004] enable application users to program and reprogram WSNs easily, it also opens the door for unauthorized users to implant malicious code into WSNs. Dutta et al. [2006] attempted to secure the MOAP by introducing program authenticity and integrity with software-based RSA PKC similar to TinyPK [Watro et al. 2004]. As in TinyPK, the security levels of RSA cryptography will be compromised by using smaller RSA public exponents and key sizes.

Believing that PKC such as RSA and ECC is too resource-intensive for resource-impoverished WSNs, researchers have investigated alternative methods to ensure program authenticity and integrity by secure hash chains, hash trees, and/or their hybrids [Deng et al. 2006; Tan et al. 2008].

In contrast to the existing alternatives to PKC that typically have limited functions, trustedFleck provides e-commerce-level PKC, which facilitates secure communication services such as confidentiality, authenticity, and integrity with low financial overhead, by exploiting the capability of a commodity Trusted Platform Module (TPM) chip.

In the development of remote attestation for WSNs, software-based attestation mechanisms have gained a lot of attentions in the past few years. Shaneck et al. [2005] argued for obfuscating the attestation routine to maximize the difficulties of attackers against remote attestation protocols [Shaneck et al. 2005]. SWATT [Seshadri et al. 2004] performs the attestation on embedded devices with a completely software-based approach for memory verification. SAKE [Seshadri et al. 2008] employs random memory check based upon ICE (Indisputable Code Execution) [Seshadri et al. 2006]. Most of these existing software-based attestation mechanisms focus on providing a trustworthy routine to guarantee the trustworthiness of the attestation process. It is commonly agreed that software-based solutions are more vulnerable to attacks

than hardware-based solutions. Hence, equipped with TPM on fleck, remote attestation could be more robust against attacks.

3. A BRIEF INTRODUCTION TO THE RSA ALGORITHM AND TPM

In this section, we provide an overview of the RSA algorithm [Rivest et al. 1978] and the TPM, a motivator for our trustedFleck.

3.1 Rivest-Shamir-Adelman (RSA) Algorithm

RSA is an algorithm for Public Key Cryptography (PKC), also called *asymmetric cryptography*, in which the encryption key is different from the decryption key. RSA is the first proposed algorithm suitable for signing and encryption, and is used widely in secure communication protocols, such as Secure Shell (SSH) and Secure Sockets Layer (SSL),[2] on the Internet.

In addition to the RSA algorithm itself, we will also discuss some RSA terms and parameters, such as modulus (n), random numbers (p and q), public exponents (e), and key sizes (k), and their implications for the RSA algorithm computation complexity and security levels.

The RSA algorithm generates a public key and a private key pair simultaneously as follows. First, RSA chooses two *large random prime numbers* p and q. Second, RSA calculates the product (n) of p and q: $n = pq$, where n is used as the modulus for RSA public and private keys. Third, RSA calculates the Euler's totient function of n, given by $\varphi(n) = (p-1)(q-1)$. Fourth, RSA chooses an integer (e, also called the *public exponent*) such that $1 < e < \varphi(n)$, and $gcd(e, \varphi(n)) = 1$, where *gcd* stands for the *greatest common divisor* (GCD). The public exponent (e) and the modulus (n) together comprise the public key. Fifth, RSA calculates the private exponent (d) by $de \equiv \ mod\ \varphi(n)$, where parameters d, p, q are kept secrets.

Since the public key (n, e) of Alice is available to everyone, Bob can then encrypt a plain text message (m) by

$$c = m^e \ mod\ n, \tag{1}$$

where c is the cipher text (cipher) of plain text message m, and $0 \leq c < n$. Only Alice, the owner of kept secrets (d, p, q), can decrypt the cipher (c) and obtain plain text message (m) by

$$m = c^d \ mod\ n. \tag{2}$$

Further, with her private key (d, p, q), Alice can use the RSA algorithm to sign a message by generating a signature (s) by substituting c with a hash value $H(m)$ of m in Equation (2), for example, $s = H^d(m)\ mod\ n$. After receiving (m, s), Bob uses the same hash function (H), together with Alice's public key (n, e), to verify the signature by Equation (1).

Because the sizes of p and q are approximately half the key size (k), the security level of RSA cryptography is a function of e and k. A popular choice (such as in OpenSSL[3]) for the public exponent is $e = 2^{16} + 1 = 65,537$. Using

[2] http://tools.ietf.org/html/rfc5246.

[3] http://www.openssl.org/.

small e values such as 3, 5, or 17 can dramatically reduce computational cost, but lead to greater security risks [Rivest et al. 1978; Hastad 1986]. The default e value in trustedFleck is 65,537. It is commonly believed that a RSA key size of 512 bits is too small to use nowadays. Bernstein [2001] has proposed techniques that simplify brute force attack RSA, and other work based on Bernstein [2001] has suggested that 1024-bit RSA keys can be broken in 1 year by a device that costs $10 million rather than trillions as in previous predictions [Shamir and Tromer 2003]. Using an RSA key of least 2048-bit length is for recommended deployments beyond 2010.[4] Therefore, the default RSA key size in trustedFleck is set to 2048 bits.

3.2 Trusted Platform Module (TPM)

The TPM is a dedicated security chip following the Trust Computing standard specification [TCG 2007].The objective of a TPM is to provide a hardware-based root of trust for a computing system. According to the standard specification in TCG [2007], the following two key components are included in the TPM design that are most relevant to WSNs:

—*Cryptography operation engine*. In TPM, a range of the cryptographic operations are available, including RSA engine for signature generation and message decryption, Secure Hash Algorithm (SHA) Engine, and Random Number Generation (RNG). The security design in WSN with TPM improves over the software-based security design in the following two ways. First, every TPM is programmed with a unique RSA key pair and the private part never leaves the nonvolatile storage area (i.e., protected memory) of TPM. Even in the event of sensor nodes being captured by an adversary, the private part of the RSA key would not be available to the adversary for further attacks. Second, the cryptographic operation engine in TPM is much more efficient than the software-based one [Watro et al. 2004; Liu and Ning 2008] in terms of execution time and power consumption, which is further explored in this article in the context of WSNs.

—*Platform Configuration Register (PCR)*. TPM contains a number (usually 16) of platform configuration registers (PCRs). The content stored in each PCR is a digest of messages in regard to the platform environment (or some other integrity-sensitive messages). PCRs are located in the nonvolatile storage area (shielded internal memory slots) and hence cannot be directly tampered with.

4. THE TRUSTEDFLECK ARCHITECTURE

In this section, we discuss both hardware and software modules of trustedFleck.

4.1 Hardware Module

Fleck [Sikka et al. 2004] (see Figure 2) is built around the Harvard architecture Atmel Atmega 1281 micro-controller, with 8 kBytes of RAM and an 8-MHz

[4]`http://www.rsa.com/rsalabs/node.asp?id=2004.`

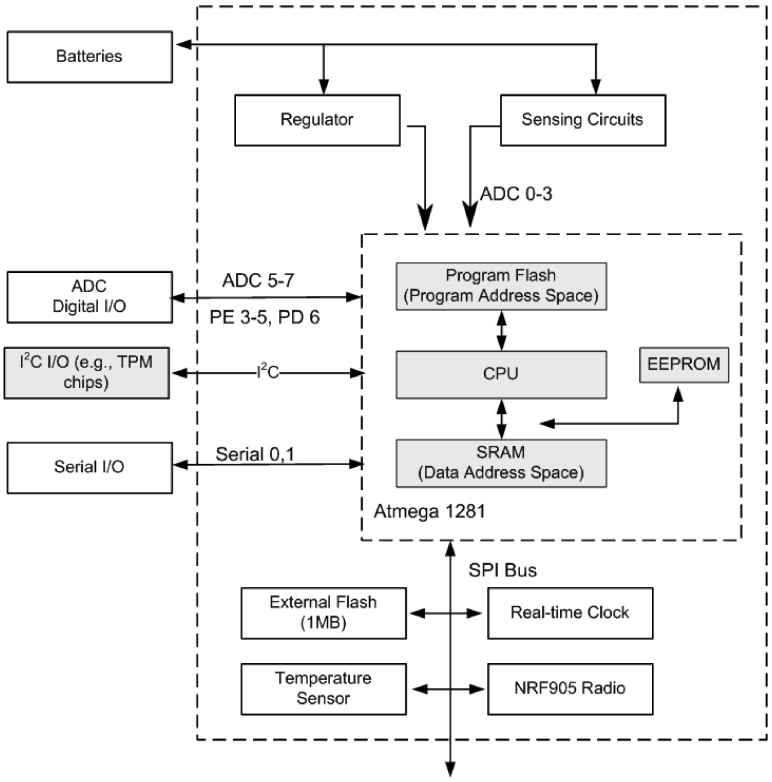

Fig. 2. The Fleck node that features the Harvard architecture Atmega 1281 micro-controller and I^2C bus for embedded devices (both highlighted).

central processing unit(CPU). Unlike the more common Von Neumann architecture, Harvard architecture micro-controllers separate data address space from program address space. This separation makes it harder for malicious users to exploit software vulnerabilities such as stack overflows.

Fleck3 uses the packet-based Nordic NRF905 transceiver for communication, which has a transmission range of up to 1,000 m in outdoor environments. The Fleck also features 1 MByte of flash storage and a real-time clock.

The Fleck hardware architecture relies heavily on the SPI bus. Atmega 1281 acts as the SPI master and can communicate with the radio, the flash memory, the real-time clock, and the temperature sensor over the SPI interface. The real-time clock and the radio can both interrupt Atmega1281 to signal alarms, packet transmission, and packet reception. To facilitate embedded sensor device connections, Fleck3 also features an I^2C interface (highlighted in Figure 2).

The core of trustedFleck is an Atmel AT97SC3203S TPM chip (see Figure 1) mounted on a Fleck expansion board (see Figure 3). The TPM chip follows version 1.2 of the Trusted Computing Group (TCG) specification for TPM.[5] It has

[5]`http://www.atmel.com/dyn/resources/prod_documents/5132s.pdf.`

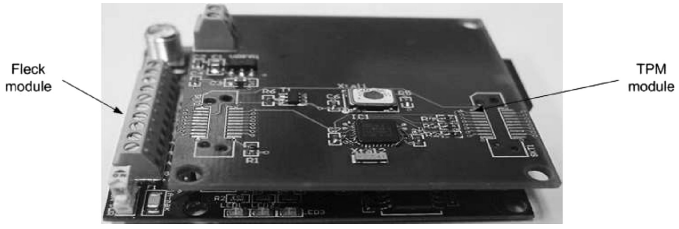

Fig. 3. The trustedFleck (Fleck3 and TPM module).

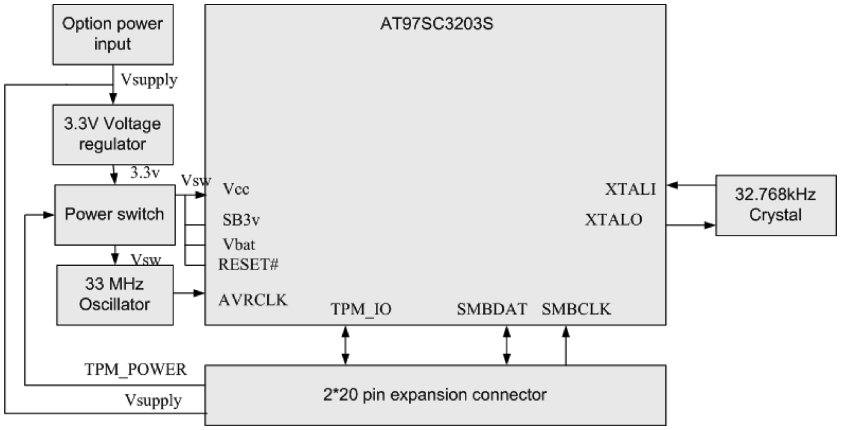

Fig. 4. The trustedFleck TPM module block diagram.

a true random number generator (RNG), which is Federal Information Processing Standards (FIPS) 140-2 compliant. By implementing computationally intensive RSA operations in hardware, the TPM chip performs these operations in an efficient manner. For example, it can compute a 2048-bit RSA signature in 500 ms according to the Atmel data sheet.[5] The TPM module has a 100-kHz SMBus which is electrically equivalent to the I^2C, enabling the TPM to be connected to the Fleck's I^2C interface. We believe that it should not be difficult to interface the TPM component to other microcontrollers such as the MSP 430, which also has an I^2C interface.

Figure 4 shows a block diagram of the TPM module, which includes the bare minimum of components required for its operation: the TPM chip, a crystal, oscillator, a voltage regulator, a power switch, and an expansion connector.

4.2 Software Module

The fleck operating system (FOS) is similar in spirit to MANTIS OS developed at the Department of Computer Science, University of Colorado at Boulder [Bhatti et al. 2005]. FOS provides a priority-based, nonpreemptive (cooperative) threading environment with separate fixed-size stacks for each thread, and a separate interrupt stack. The kernel checks all stacks at system entry and invokes a panic if a stack overflow has occurred. FOS has the advantage of providing a simple concurrent programming model which does not require

semaphores. The scheduler is also responsible for CPU power management and enters the lowest mode consistent with thread resource requirements.

Time-critical operations such as analog data sampling or high-speed timers are handled by interrupt-level callbacks. A virtualized timer based on an event-time queue is provided for non-time-critical delays. FOS currently provides a classic carrier sense multiple access (CSMA) media access control (MAC) with optional acknowledgment and low-power duty-cycling support.

FOS provides uniform access to underlying resources via a POSIX-like API for configuring the hardware and sending/receiving data using the hardware, including the SPI-bus, serial ports, nonvolatile storage, etc. Support is also provided for a large collection of interface boards including GPS and inertial measurement sensors used in the mobile animal nodes. The I/O model is for blocking read and write primitives, with Unix-like semantics, and optional timeout on read.

FOS has been ported to our range of Flecks, all of which use the Atmega 128 and 1281 processors and different radios from the Nordic nRF90x series and solar power. Hardware abstraction is always a difficult balance between generality and efficiency, and our philosophy has been a middle ground where the programmer is expected to be aware of the device limitations. With timers, for example, a consistent and powerful interface to timers is presented; however the number of timers, their individual characteristics, and their effect on power management are issues the programmer must deal with.

Other support tools include a static analyzer for stack sizing, a memory dump analyzer, an RPC mechanism with support for host programs written in Python, and a bootloader for reliable over-the-air programming.

4.3 trustedFleck Primitives

For the convenience of WSN application developers, we have implemented a set of TPM primitives as an FOS library module. We divide these primitives into three categories: general TPM functions, symmetric and asymmetric key cryptography functions, and trust functions.

4.3.1 *General TPM Functions.* This Application Programming Interface (API) is summarized in Figure 5. Previous research [Hill and Culler 2002] has shown that the primitives, which allow an application to turn a system component on or off, are important to conserve system energy consumption in sensor networks. trustedFleck allows applications to duty cycle the TPM component using the primitives `fos_tpm_startup()` and `fos_tpm_turnoff()`. `fos_tpm_startup()` takes a startup mode parameter that is one of TPM_ST_CLEAR (clean start), TPM_ST_SAVE (restores saved configurations such as PCR values), or TPM_ST_DEACTIVE. These duty cycle primitives are very important in trustedFleck because the TPM's current consumption (around 50 mA) is significantly greater than Flecks' average current consumption (around 5 mA).

Symmetric keys are typically generated by a pseudorandom number generator, for example, see Karlof et al. [2004]. If an attacker can extract the initial random symmetric key, then it is possible for the attacker to compute

```
/* Duty cycle TPM chip functions. */
uint8_t fos_tpm_startup(uint8_t mode);
uint8_t fos_tpm_turnoff(void);

/* True random number generator. */
uint8_t fos_tpm_rand(uint32_t *randNumber, uint8_t len);

/* Secure Hash Algorithm (SHA-1) */
uint8_t fos_tpm_SHA1(uint8_t *msg, int msgLen,
                     uint8_t *digest);

/* Keyed-Hash Message Authentication Code (HMAC) */
uint8_t fos_tpm_HMAC(uint8_t *shareSecret, uint16_t secretLen,
                     uint8_t *input, uint16_t inputLen,
                     uint8_t *digest);

/* Save the TPM state to the permanent TPM internal storage */
uint8_t fos_tpm_saveState(void);
```

Fig. 5. The trustedFleck Application Programming Interface (API) for general TPM functions.

all past and future random symmetric keys. Therefore, a high-quality random number generator is very important for the effectiveness of symmetric key operations. trustedFleck provides the `fos_tpm_rand()` primitive, which is based on a true random number generator (RNG), and is Federal Information Processing Standards (FIPS) 140-2 compliant. The `fos_tpm_rand()` primitive returns an unsigned integer value between 0 and $2^{32} - 1 = 4,294,967,295$.

A Hash Algorithm (SHA-1) is one of the most frequently used cryptographic operations for TPM commands as it produces a collision-free 20-byte hashed digest regardless of the length of the input messages. trustedFleck provides the `fos_tpm_SHA1()` primitive to compute the digest of message strings. Given its characteristics, the input message of some TPM commands would be first "hashed" via SHA-1, whose hashed digest could be the actual input for the respective TPM commands. For example, in `fos_tpm_sign()`, the message being signed is a hashed digest of the message-to-be-signed rather than the message-to-be-signed itself since the length of the message-to-be-signed varies. If its length is larger than the length of the signing key (i.e., 256 bytes), `fos_tpm_sign()` could not handle it. However, with the hashed digest, whose length is fixed at 20 bytes, the input message to `fos_tpm_sign()` could never be larger than the signing key. Another reason why SHA-1 is frequently used is that it is a one-way function which is computationally infeasible to invert, and is used to develop the Hashed Message Authentication Code (HMAC).

HMAC is an extension of SHA-1 that includes the shared secret between the HMAC generator and the verifier. Then SHA-1 is applied to the concatenated shared secret and message to generate HMAC. In TPM, HMAC is mainly used to produce an *authorization digest* for the authorized or dual-authorized TPM commands [TCG 2007] by including the antireplay nonce and the message-to-be-authenticated. As the shared secret is not available to third parties an

```
/* trustedFleck public key collector. */
uint8_t fos_tpm_getPubKey(uint8_t *pubKey);

/* Asymmetric key encryption/decryption. */
uint8_t fos_tpm_encryption(uint8_t *msg, uint16_t len,
                           uint8_t *pubKey, uint8_t *cipher);
uint8_t fos_tpm_decryption(uint8_t *cipher, uint8_t *msg,
                           uint16_t *len);

/* Digital signature and verification. */
uint8_t fos_tpm_sign(uint8_t *digest, uint8_t *signature);
uint8_t fos_tpm_verifySign(uint8_t *signature, uint8_t *pubKey,
                      uint16_t *digest);

/* Symmetric session key encryption/decryption. */
uint8_t fos_xtea_encipher(uint8_t *msg, uint8_t *key,
                           uint8_t *cipher, uint8_t nRounds);
uint8_t fos_xtea_decipher(uint8_t *cipher, uint8_t *key,
                           uint8_t *msg, uint8_t nRounds);

/* Symmetric session key retrieve and store*/
fos_xtea_getkey(uint8_t *key, uint8_t location);
fos_xtea_storekey(uint8_t *key, uint8_t location);
```

Fig. 6. The trustedFleck API for public key infrastructure and symmetric session key functions. The public key cryptographic primitive interfaces allow applications to start up and turn off onboard the TPM chip, to read the public key of the TPM chip, to encrypt or to decrypt a message, to sign a message, and to verify a signature. The symmetric key primitive interfaces allow applications to encrypt and decrypt messages using the XTEA algorithm.

adversary cannot replay the intercepted message due to the antireplay nonce or forge a valid HMAC for the tampered message to circumvent the HMAC check.

In addition to the general security functions discussed above, trustedFleck has some other configuration functions such as `fos_tpm_saveState`, which is to preserve the nonvolatile area in each TPM duty cycle [TCG 2007]. This function is important as it can ensure that the platform configuration information such as PCR values can be preserved even after the TPM is turned off.

4.3.2 *Asymmetric Key and Symmetric Key FOS Functions.* This API is summarized in Figure 6. Each TPM has a unique 2048-bit private key established during manufacture which cannot be read. However, an application can acquire the corresponding public key from the TPM which can be shared with other nodes for encryption and signature verification purposes. An application encrypts a message by providing the plain text, the length of the plain text (which has to be less than or equal to the size of the public key), and a public key—the cipher text is returned. Similarly, an application can decrypt cipher text. The trustedFleck encryption and decryption facilitates message confidentiality.

```
/*TPM PCR functions*/
uint8_t fos_tpm_pcrExtend(uint8_t pcrIndex, uint8_t *inputSha1,
                                  uint8_t *extendSha1);

uint8_t fos_tpm_pcrRead(uint8_t pcrIndex, uint8_t *outputSha1);

uint8_t fos_tpm_pcrQuote(uint8_t key_index, uint8_t *signature,
                           uint8_t *digest);

uint8_t fos_tpm_verifyPcrQuote(uint8_t* signature, uint8_t *pubKey,
                     uint8_t *digest);
```

Fig. 7. The trustedFleck API for trust functions.

trustedFleck provides two additional primitives, `fos_tpm_sign()` and `fos_tpm_ verifySign()`, for applications to sign messages or to verify the signatures of messages. To generate a signature of a plain message, an application produces the 20-byte HMAC value (also called `digest`) of the plain message by using the `fos_tpm_hmac()` primitive. Then the application passes the `digest` to the `fos_tpm_sign()`, which returns the signature of the plain message.

A base station typically has more computational power, memory, and energy resources and can be treated as a certification authority (CA). All nodes store the CA's public key in their permanent memories such as electrically erasable programmable read-only memory (EEPROM) before deployment, and the base station has the public keys of all nodes. Multiple base stations and/or dedicated CA nodes with more memory can be used to improve the scalability of this approach. Therefore, message authenticity can be facilitated.

Previous work [Karlof et al. 2004] has shown that symmetric key cryptography is tractable on mote-class hardware and can achieve message confidentiality. Further, symmetric key cryptography is significantly less resource-intensive than asymmetric key cryptography such as RSA. trustedFleck also features a 128-bit symmetric block cipher based on the eXtended Tiny Encryption Algorithm (XTEA) [Needham and Wheeler 1997]. XTEA operates on 64-bit blocks with 32 or 64 rounds. trustedFleck chooses XTEA symmetric key cryptography because of its small random access memory (RAM) footprint, which makes it a good candidate for resource-impoverished sensor devices that typically have less than 10 kB RAM. XTEA can be used in an output feedback mode to encrypt or decrypt variable-length strings.

4.3.3 *Trust Functions.* This API is summarized in Figure 7. trustedFleck is able to attest to its state upon challenge from a remote device such as another trustedFleck or the base station. Such behavior is called *remote attestation*. Namely, a Fleck reports its integrity state, for example, the values inside its PCR, to the remote trusted party (e.g., base station). By doing so, the attestor can ensure that the attested trustedFleck is running the expected program, and thus is trustworthy.

The trustedFleck primitive `fos_tmp_pcrExtend()` takes the secure hash function (SHA-1) of input values together with the current value in specified PCR, and saves the returned hash value to the specified PCR.

Table I. Comparison of RSA Encryption Times

Public Exponent (e)	Software 1024 bits	Software 2048 bits	Hardware 2048 bits
3	0.45 s	65 s	N/A
65,537	4.185 s	450 s	0.055 s

Table II. RSA Computation Time in TrustedFleck for e =65,537 and 2048 Bit Key

Encryption	Decryption	Sign	Verification
55 ms	750 ms	787 ms	59 ms

The trustedFleck primitive `fos_tmp_pcrRead()` returns the hash value stored in the specified PCR while `fos_tmp_pcrQuote()` could return one or multiple PCR values. The difference between `fos_tmp_pcrRead()` and `fos_tmp_pcrQuote()` is that `fos_tmp_pcrRead()` is a *nonauthorized* command (i.e., no authentication is imposed on it); hence, `fos_tmp_pcrRead()` could be used for local attestation but is not suitable for remote attestation. The trustedFleck primitive `fos_tmp_pcrQuote()` is an *authorized* command. The returned PCR values cannot be tampered with without being detected as these values are signed with the private part of a designated key pair stored in the TPM. Each key pair stored in the TPM is assigned a shared secret when it is created [TCG 2007] and this secret is utilized to produce an authorization digest through HMAC, as described in Section 4.3.1.

The trustedFleck primitive `fos_tmp_verifyPcrQuote()` takes the output from `fos_tmp_pcrQuote()` and verifies the respective signature.

5. PERFORMANCE EVALUATION

In this section, we discuss the performance of the trustedFleck platform in terms of computation time, energy consumption, and financial cost.

5.1 Asymmetric Key (RSA) Operations

As part of our benchmarking we also implemented the RSA encryption algorithm, on Fleck, in software for comparison. Table I shows the encryption time for different key sizes and RSA public exponents (e) in both the software and hardware implementations. The results show that the TPM chip can reduce the computation time of RSA encryption by a factor of 8000, when e = 65,537 and the key size is 2048 bits. Table I also shows that the software RSA implementation is thus impractical using embedded microcontrollers such as Atmega 128 when $e > 3$ and for key size larger than 1024 bits. A small e will make RSA less secure, and a key size of 1024 bits will no longer be considered secure in a few years time (see Section 3.1 for a detailed discussion).

We have not implemented the RSA decryption algorithm in software because it is significantly more computationally intensive than the RSA encryption algorithm (see Table II). Table II also shows RSA encryption, decryption, sign, and signature verification computational times in trustedFleck.

Table III. trustedFleck Current Consumption

Module	Current (mA)
Fleck3 (without radio, node idle)	8.0
Fleck3 + Receive	18.4
Fleck3 + Transmit	36.8
Fleck3 + TPM encryption	50.4
Fleck3 + TPM decryption	60.8
Fleck3 + TPM signature	60.8
Fleck3 + TPM signature verification	50.4

Table IV. trustedFleck (RSA and XTEA) Encryption Energy Consumption for 1 *bit* of Data

Platform	Current (mA)	Time (μs)	Energy (μJ)
RSA (software, $e = 65{,}537$, 2048-bit key)	8.0	219,730	7,030.0
RSA (hardware, $e = 65{,}537$, 2048-bit key)	50.4	27	5.4
XTEA (software, 128-bit key)	8.0	18	0.6

Table III shows the current consumption for different trustedFleck operations. It shows that RSA operations consume 37% to 65% more current than transmitting in trustedFleck. Table IV shows the energy consumption of a 2048-bit RSA encryption operation when $e = 65,537$. It shows that the software based approach consumes approximately 1300 times more energy than trustedFleck for an RSA encryption operation. Table IV also shows that the software based approach RSA encryption in WSNs consumes a large amount (approximately 7 mJ/bit) of energy when $e = 65,537$ and the key size is 2048 bits. On the other hand, trustedFleck makes it feasible to support PKC technology in WSNs.

5.2 Symmetric Key (XTEA) Operations

We tested the performance of XTEA cryptography to determine its computational speed on the Fleck platform. trustedFleck can encrypt one block of 64-bit data in approximately 1.15 ms. Therefore, it takes approximately 18 μs (Table IV, Row 3) to encrypt 1-bit data.

Table IV also shows that software symmetric key cryptography is indeed significantly faster than hardware RSA asymmetric key cryptography (18 μs vs. 27 μs per bit). Furthermore, XTEA encryption consumes approximately 10 times less energy compared to hardware RSA encryption, and approximately 12,000 times less energy compared to software RSA encryption. It suggests that, in an energy-impoverished WSN, we should use symmetric cryptography for most secure communications, and should use asymmetric cryptography in critical tasks only (i.e., symmetric key management).

The effective data rate of Fleck transceiver (Nordic nRF905) is 50 kb/s with Manchester encoding. For a 32-byte physical layer payload, there are a 4-byte address and a 2-byte Cyclic Redundancy Check-16 (CRC-16) overheads. Therefore, the available bandwidth for the Media Access Layer (MAC) is $50 \times 32 \div (32+4+2) = 42.11$ kb/s. It takes 23.75 μs for a NRF905 transceiver to transmit 1 bit, which is significantly longer than the encryption time (18 μs).

Table V. trustedFleck PCR Operation Computational Time, Current and Energy Consumption

Module	Computational Time (ms)	Current (mA)	Energy (mJ)
Fleck3 + TPM PCR read	5.8	52.8	0.918
Fleck3 + TPM PCR quote	1400	64	268.8
Fleck3 + TPM PCR verify quote	900	52	140.4
Fleck3 + TPM PCR extend	See Figure 8	51.2	See Figure 8

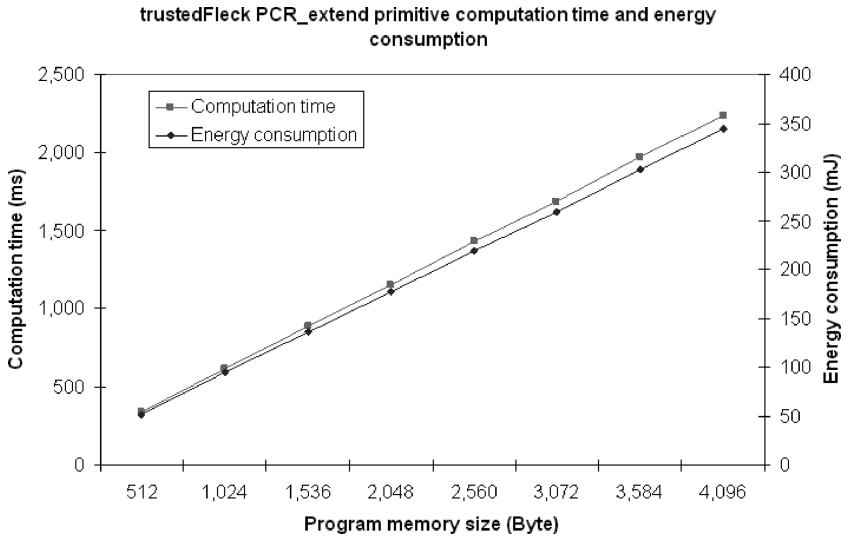

Fig. 8. trustedFleck primitive `fos_tpm_pcrExtend` computation time and energy consumption versus the size of input program memory.

The other key advantage of XTEA is *space efficiency*. The FOS XTEA implementation has less than 100 lines of C codes, and requires 52 bytes of RAM and 1082 bytes of program space only.

5.3 PCR Operations

We tested the performance of PCR operations to determine the computational speed on the Fleck platform. Table V shows that it is very inexpensive (less than 5.8 ms or 1 mJ in energy consumption) to read the value of a PCR. However, the sign operation is significantly more expensive (see Table II)—it takes approximately 1.4 s or costs 268 mJ in energy use for trustedFleck to perform a `fos_tpm_pcrQuote()` operation.

Figure 8 shows the computational time and energy consumption of the `fos_tpm_ pcrExtend()` primitive against the size of the input program memory. It shows that time and energy are approximately linear with memory size. For a 10 kB program image, the energy consumption of the `fos_tpm_pcrExtend()` primitive is approximately 847 mJ.

Table VI. Energy Consumption of trustedFleck (2048 bits) and TinyECC (192 bits)

	Current (mA)	Time (ms)	Energy (mJ)
TinyECC 192k1 (sign)	8	3070	73.7
TinyECC 192k1 (verify)	8	3612	86.7
trustedFleck 2048 (sign)	60.8	787	143.5
trustedFleck 2048 (verify)	50.4	59	8.9

Table VII. Sign and Verify Operation Code Sizes (bytes) of the trustedFleck (2048-bit) and TinyECC (192-bit)

	trustedFleck (2048)	TinyECC (192k1)
ROM	1958	13,488
RAM	108	1712

5.4 TinyECC and trustedFleck

We study the performance of trustedFleck and TinyECC in this section. We reuse the performance results of TinyECC (192k1, window size = 4) in the MicaZ platform, which has a similar microcontroller (Atmega 128) as trustedFleck, as shown in Liu and Ning [2008]. Note that the security level of a 2048-bit RSA is equivalent to that of a 224-bit ECC [Gurn et al. 2004]; therefore, a 2048-bit RSA is stronger than a 192-bit ECC.

Table VI shows the computation time and energy consumption of 192-bit TinyECC and 2048-bit the trustedFleck sign and signature verification operations. The computation time of trustedFleck is significantly less than that of TinyECC; for example, a sign operation of TinyECC is four times longer than that of trustedFleck. Therefore, trustedFleck can reduce end-to-end signature generation and verification processes significantly. While trustedFleck consumes approximately twice the energy of TinyECC for a sign operation, it consumes approximately 12% energy that TinyECC requires for a signature verification operation. Overall, trustedFleck consumes slightly less energy than TinyECC for the sign and a signature verification operations combined. Since a RSA 2048-bit is stronger than a ECC 192-bit, trustedFleck provides a stronger security level than TinyECC with a similar energy consumption. Since a sign operation in trustedFleck consume 15 times more energy than a signature verification operation, we can perhaps design security protocols that require more sign operations in resource-rich nodes like the sinks [Watro et al. 2004].

Table VII shows the program flash (ROM) and RAM sizes of 192-bit TinyECC and 2048-bit the trustedFleck sign and signature verification operations. It shows that TinyECC uses seven times more space than trustedFleck in terms of both ROM and RAM. Space efficiency is very important for resource-impoverished embedded devices such as sensor nodes.

5.5 Node Lifetime Estimation

We assume that the node is powered by 2 AA 2800-mAHr batteries, which is equivalent to $2800 \times 2 \times 1.5 \times 3,600 = 30,240,000$ mJ. Then we can calculate

Table VIII. Estimated Node Lifetime (in years) When a 10% Fraction of Battery Capacity is Available for the trustedFleck Security Operations

	Number of Operations Per Day			
	1	2	12	24
Sign	57.7	28.9	4.8	2.4
Verify	928.7	464.4	77.4	38.7
PCR quote	59.0	29.5	4.9	2.5
PCR quote verify	30.8	15.4	2.6	1.3
All	14.8	7.4	1.2	0.6

the expected lifetime of the node based on the energy consumption figure from Tables II, III, and V when 10% of the battery capacity is available for security operations. Table VIII shows these results. It also shows that the trustedFleck operations are fairly affordable for a typical WSN node setting. For example, if the node performs sign, verify, PCR quote, and PCR quote verify operations twice per day, the expected node lifetime is 7.4 years.

5.6 The Financial Cost of the trustedFleck

An Atmel AT97SC3203S TPM chip costs $4.5 when ordered in large quantities,[6] which is less than 5% of the cost of popular wireless sensor nodes such as Telosb, Iris mote, and Fleck (about $100). The TPM chip is small in size (Figure 1), measuring just 6.1×9.7 mm, which is less than 2% of area of Fleck, and could easily be integrated into a commercial manufacturing environment rather than using the cumbersome expansion board used in this prototype.

6. CASE STUDIES AND DISCUSSION

In this section, we demonstrate the power of our trustedFleck primitives (see Section 4.3) to easily and efficiently realize secure WSN applications. These applications include, but are not limited to, secure over-the-air programming, remote attestation, secure remote procedure calls (RPC), and secure session key management. In addition to the variants of state-of-the-art key management [Nilsson et al. 2008] and secure software update protocols [Hui and Culler 2004], we have also implemented a remote attestation protocol with the trustedFleck primitives, and show how the trustedFleck primitives can improve the protocol's performance.

6.1 Symmetric Session Key Encryption/Decryption

Symmetric key cryptography consumes significantly less energy than RSA asymmetric key cryptography (see Table IV), and we envision that symmetric session key cryptography will be used for most WSN secure communications, and asymmetric cryptography will be used for limited critical tasks. For example, asymmetric cryptography is used to exchange a new symmetric key daily

[6]http://www.atmel.com/dyn/products/view_detail.asp?ref=&FileName=embedded10_18.html&Family_id=620.

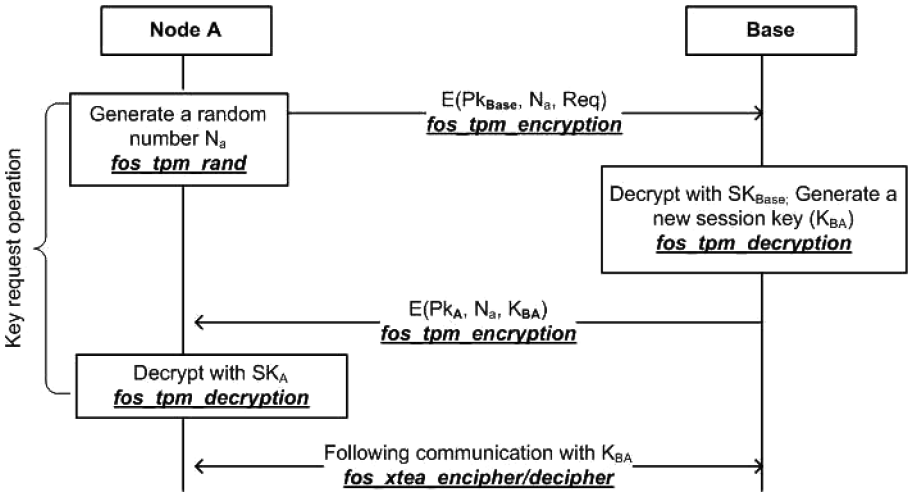

Fig. 9. Symmetric session key request operation with the trustedFleck primitives (underlined). Node A requests a session key from a base station.

or hourly (also called the *rekey process*). We will discuss the rekey process in detail in Section 6.2.

By utilizing just two trustedFleck primitives, we can easily achieve symmetric key cryptography in trustedFleck (see Figure 6). `fos_xtea_getkey()` reads a symmetric key from trustedFleck memory, and `fos_xtea_encipher()` encrypts a plain message (msg), and returns an encrypted message (cipher). Therefore, link-level secure transmissions can be achieved by passing the returned cipher over the radio.

An application can choose to store the session keys in Fleck RAM, EEPROM, or TPM EEPROM. When the keys are stored in Fleck RAM or EEPROM, the key (getting and storing) operations consumes significantly less energy than when the keys are stored in TPM EEPROM. However, storing the keys in Fleck also exposes the keys to more risks. Hartung et al. [2005] demonstrated how to extract the information in a node's EEPROM and RAM within 1 min. Perhaps it is better to store the key in the TPM chip for those infrequent operations (e.g., sending one temperature sample to the base station every 5 min) and store the key in the Fleck memory for high-throughput operations such as secure over-the-air programming.

6.2 Sensor Node Symmetric Session Key Request/Assignment Operation

Figure 9 shows the protocol for a sensor node (Node A) to request a new symmetric key from a base station. Node A initiates this process periodically, for example, hourly or daily. It generates a random number (N_a, also called *nonce*) and encrypts the nonce along with the Request (Req) command using Base's public key (Pk_{Base}) before transmitting it to the base. The purpose of nonce (N_a) is to defend against replay attacks. After receiving the Request message from Node A, the base decrypts the message with its private key (SK_{Base}). The base then responds to the Req command by generating a new symmetric session key

(K_{BA}), and encrypts it together with N_a using Node A's public key (Pk_A) before transmitting it to Node A. Node A decrypts the message from the Base with its private key (SK_A) and obtains the new symmetric key (K_{BA}). Node A and the base can then use K_{BA} for future secure communications. Figure 9 also shows the five trustedFleck primitives associated with each step of the key request protocol.

The session key *assignment* operation is symmetric to the key *request* operations. The key assignment protocol is initiated, for example, in a group key establishment event (see Section 6.3), by the base station instead of a node.

6.3 Group Key Establishment Operation

Group key establishment can be achieved by a combination of sensor node symmetric session key request operations and sensor node symmetric session key assignment operations. For example, if node A wants to communicate with nodes B and C, Node A will request a new group session key from the base station via the session key request operation introduced in Section 6.2. After receiving the key request operation from Node A, the base station generates a new symmetric key (K_{abc}). The base station assigns K_{abc} to Nodes B and C via two session key assignment operations (see Section 6.2) before transmitting K_{abc} to Node A. Then, Nodes A, B, and C begin secure communications using the group session key K_{abc}.

6.4 Secure Software Update Protocol

Multihop over-the-air programming (MOAP) protocols such as Deluge [Hui and Culler 2004] enable users to reprogram/retask the WSN remotely, which is critical to efficient and effective management of large-scale long-duration WSNs. The basic Deluge protocol works as follows. A node (Node A) advertises its program version periodically. One of its neighbors (Node B) will send a request to download a copy of the program from Node A if Node B's version is older than Node A's. Node A begins the download process after receiving the request. To support concurrent data disseminations and reduce network reprograming time, Deluge divides a program into a number of pages.

By using the group key establishment operation introduced in Section 6.3, trustedFleck can provide data confidentiality to Deluge. Furthermore, a base station can achieve integrity and authentication by signing the advertisement message and the program pages of Deluge with its private key (SK_{Base}) before disseminating it to the network. After receiving a program page or an advertisement message, a trustedFleck node can then verify the page or the advertisement message with the public key of the base station (Pk_{Base}). This mechanism ensures that wireless bootstrap can only be initiated by an authorized host, that the code stream is private, and that a page is not committed to flash unless it is from an authorized host.

A trustedFleck node can verify the signature of the a 256-byte page in 59 ms (see Table II), which is more than 4,300 bytes/s. This trustedFleck signature verification rate is approximately 50 times faster than the average 88.4-bytes/s dissemination rate achieved by Deluge in a 75-node network [Hui and Culler 2004].

6.5 Trusted Sensor Nodes

6.5.1 *Adversary Model.* We assume an adversary could have access to a powerful computational resources such as a Notebook computers. He/she could launch an *external attack* or an *insider attack*. In an *external attack*, an adversary could eavesdrop on the information, inject malicious packets into the network, replay previously intercepted packets, or impersonate other nodes. Denial-of-service (DoS) such as jamming (i.e., an adversary keeps sending garbage packets to cause collisions at the MAC layer) or power depletion attacks (i.e., an adversary sends garbage packets to trigger TPM operations repeatedly in its neighbors until the power of these neighbor nodes is depleted) is out of the scope of this article. How to counter DOS attacks will be part of our future work.

The adversary could also compromise some nodes to attack the rest of the nodes in the network [Francillon and Castelluccia 2008; Goodspeed 2007]. Such an attack is called an *insider attack* because the compromised sensor nodes are considered as legitimate nodes in the WSN before they are detected and removed. The adversary could instruct the compromised nodes not to follow the specifications of the security protocols (e.g., to selectively drop the packets) or to exploit the vulnerabilities of the protocols. In this article, the adversary could learn about the content stored in the memory areas of the sensor nodes via node compromises. However, despite nodes being compromised, the cryptographic information internally stored in the TPMs could not be learned[7] because TPMs are tamper-proof hardware. For example, the private keys used for signing messages or message decryption cannot be known to the attacker even though the sensor nodes to which the TPMs are attached to have been compromised.

6.5.2 *Secure Bootloading and Remote Attestations.* For the purpose of secure bootloading, the Atmega 1281 processor is configured via the fuses so that, on reset, control is transferred to the initialization code within the bootloader segment. Then the initialization code performs a TPM clear state (TPM_ST_CLEAR) reset, which clears all TPM configurations including PCR values, followed by PCR extension (`fos_tpm_pcrExtend()`) for all 128 kB of the program flash memory (including the bootloader segment). In addition, the lock bit fuses are set so that the bootloader segment cannot be written from software using the Scratch Pad Memory (SPM) instruction, and this confirms *our assumption that bootloader can be trusted*.

The remote attestation protocol shown in Figure 10 is used to test the system integrity of a node and to defend against *over-the-air malicious code injection attacks* as introduced in Francillon and Castelluccia [2008]. First, the challenger sends out an attestation challenge to an attestator with a tuple (P_i, N_a), where P_i is a specified PCR index and N_a is an antireplay nonce. If the challenger does not hold the public key (Pk_a) of the attestator, it will request the public key from a trusted third party such as a sink. Second, the attestator collects the signature $S(P_{i\_val}, N_a)$ using the trustedFleck primitive `fos_tpm_pcrQuote()` and its

[7]Such information is secured by Storage Root Keys (SRKs), which are directly created when the ownerships of the TPMs are taken.

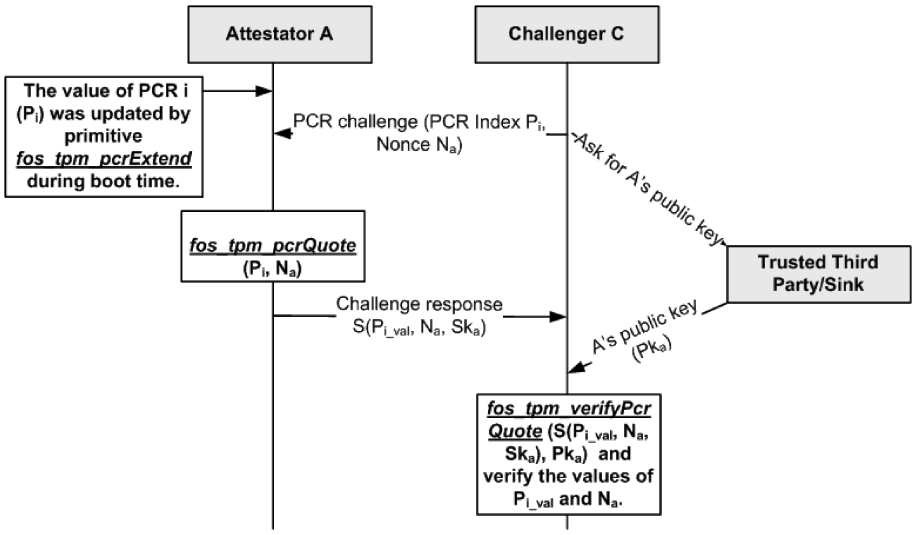

Fig. 10. Remote attestation operation with the trustedFleck primitives (underlined). Node C (Challenger) issues an integrity challenge to Node A (Attestator).

private key (SK_a), where P_{i_val} is the specified PCR value. Third, the attestator sends the signature $S(P_{i\_val}, N_a)$ as the response of attestation challenge to the challenger. Finally, the attestation can verify P_{i_val} and its authenticity by using the trustedFleck primitive `fos_tpm_verifyPcrQuote()`. If both P_{i_val} and N_a are correctly verified, the challenge can then trust the attestator.

6.6 Backward Secrecy and Forward Secrecy

trustedFleck can enhance the security levels of the re-key process by providing both backward and forward secrecy. Backward secrecy means that compromising the current symmetric link key does not allow an attacker to learn previously recorded messages encrypted with the previous key. Forward secrecy means that compromising the symmetric link key does not allow an attacker to learn future communication encrypted with the following key.

A symmetric link key can be found by an attacker by extracting it directly from a captured node via a JTAG or similar device [Hartung et al. 2005] because of the physically exposed and remote nature of nodes in WSN. Furthermore, the attacker can also extract the initial random key used by the software pseudorandom number generator. This key allows the attacker to compute all past and future nonces used in the key updating protocol, which in turn allows the attacker to compute all past and future keys.

Equipped with a FIPS 140-2 compliant true random number generator (RNG), trustedFleck can increase the security level of the protocols. It is very difficult, if not impossible, for an attacker, who has obtained the current symmetric link key, to find out the past or future keys generated by a true RNG.

6.7 Secure Remote Procedure Calls (RPC)

The Fleck Operating System (FOS) uses remote procedure calls (RPCs) to allow application programs to seamlessly access services on one or more sensor nodes.

Table IX. Common Secure FOS RPC Actions

RPC Actions	Description	Cryptography
assign_session_key	assign a new symmetric session key to a node	PKC
request_session_key	request a new symmetric session key from a base	PKC
kernel	get FOS system memory statistics	*share*
read_eeprom	read from EEPROM	*share*
read_ram	read from RAM	*share*
threads	get information about threads, label and stack usage	*share*
write_eeprom	write to EEPROM	*share*
write_ram	write to RAM	*share*
rtc_get	get time from the real-time clock	*share*
rtc_set	set the real-time clock	*share*
txpwr_set	set radio transmit power	*share*
leds	set or toggle LEDs	*share*
power	get battery and solar cell status	*share*

Each node-side service is described by an action file, a C-like function that supports multiple input and output arguments. A code generator, in Python, parses all action files and generates a server function and all the serializing and deserializing code, as well as a Python class to be used by base station applications. All nodes support the common set of actions listed in Table IX, in addition to application-specific actions.

An RPC call message comprises the function arguments, function enumerator, sequence number, and node ID of the caller, and a CRC-32. On receipt of an RPC call message (indicated by the routing header type) the message is decrypted using the session key and the CRC-32 checked. Except for the `assign_session_key` and `request_session_key` RPC messages, all the other RPC messages are encrypted using XTEA with the current session key (see Section 6.1). `assign_session_key` and `request_session_key` RPC messages are encrypted and signed with PKC as introduced in Section 6.2. In a sensor network, it is possible to broadcast the RPC call encrypted by a group symmetric key (see Section 6.3), and have the function executed in parallel on many nodes which all return their results to the caller. In this case the result of an RPC call would be a list of return values rather than just one.

Secure RPC, based on the trustedFleck primitives, provides privacy of commands and return values, and authentication, and also immunity to replay attacks.

7. CONCLUSION

We have shown that E-commerce-strength RSA-based security is feasible on a sensor network device. We have utilized commodity TPM hardware technology to create a trusted node that provides essential security services such as *message integrity, confidentiality, authenticity*, and *system integrity* based on RSA public-key and XTEA-based symmetric-key cryptography. Our evaluation shows that trustedFleck provides these services within the computational, memory and energy limits that apply to WSN nodes. Our results also show that trustedFleck significantly outperforms previous approaches such as TinyECC in terms of computational time and memory usage while providing stronger security levels. Additional advantages of the hardware approach include secure

storage of the private key and support for system configuration checking. An RSA-based security approach also allows for seamless and secure interoperability between WSNs and Internet-based applications. We have shown with examples of remote attestation, symmetric key management, secure RPC, and secure software update, that the trustedFleck platform is easy to use.

ACKNOWLEDGMENTS

The authors thank Mr. Leslie Overs, Dr. John Zic, Dr. Diet Ostry (CSIRO, Australia), Dr. Juanma Gonzalez Nieto (Queensland University of Technology, Australia), and the anonymous *EWSN* and *TOSN* reviewers for their comments and suggestions.

REFERENCES

BERNSTEIN, B. 2001. Circuits for integer factorization: A proposal. Manuscript. `http:cr.yp.to/papers.html`.

BHATTI, S., CARLSON, J., DAI, H., DENG, J., ROSE, J., SHETH, A., SHUCKER, B., GRUENWALD, C., TORGERSON, A., AND HAN, R. 2005. MANTIS OS: An embedded multithreaded operating system for wireless micro sensor platforms. *Mob. Netw. Appl. 10*, 4, 563–579.

DENG, J., HAN, R., AND MISHRA, S. 2006. Secure code distribution in dynamically programmable wireless sensor networks. In *Proceedings of the 5th International Conference on Information Processing in Sensor Networks (IPSN)*. ACM, New York, NY, 292–300.

DINH, T. L., HU, W., SIKKA, P., CORKE, P., OVERS, L., AND BROSNAN, S. Oct. 2007. Design and deployment of a remote robust sensor network: Experiences from an outdoor water quality monitoring network. In *Proceedings of the 2nd IEEE Workshop on Practical Issues in Building Sensor Network Applications (SenseApp)*.

DUTTA, P. K., HUI, J. W., CHU, D. C., AND CULLER, D. E. 2006. Securing the deluge network programming system. In *Proceedings of the 5th International Conference on Information Processing in Sensor Networks (IPSN)*. ACM, New York, NY, 326–333.

FRANCILLON, A. AND CASTELLUCCIA, C. 2008. Code injection attacks on harvard-architecture devices. In *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS)*. ACM, New York, NY, 15–26.

GOLDWASSER, S. 1997. New directions in cryptography: Twenty some years later (or cryptograpy and complexity theory: a match made in heaven). In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society Press, Los Alamitos, CA, 314.

GOODSPEED, T. 2007. Exploiting wireless sensor networks over 802.15.4. In *Proceedings of ToorCon*.

GURN, N., PATEL, A., WANDER, A., EBERLE, H., AND SHANTZ, S. C. 2004. Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUS. Lecture Notes in Computer Science vol. *3* 156. Springer, Berlin, Germany, 119–132.

HARTUNG, C., BALASALLE, J., AND HAN, R. 2005. Node compromise in sensor networks: The need for secure systems. Tech. rep. University of Colorado at Boulder. Boulder, CO.

HASTAD, J. 1986. On using RSA with low exponent in a public key network. In *Proceedings of the Advances in cryptology—(CRYPTO'85)*. Lecture Notes in Computer Science, vol. 218 Springer-Verlag, New York, NY, 403–408.

HILL, J. AND CULLER, D. 2002. Mica: A wireless platform for deeply embedded networks. *IEEE Micro 22*, 6, 12–24.

HU, W., BULUSU, N., CHOU, C. T., JHA, S., TAYLOR, A., AND TRAN, V. N. 2005. A hybrid sensor network for cane-toad monitoring. In *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems (SenSys)*. ACM Press, New York, NY, 305–305.

HU, W., CORKE, P., SHIH, W. C., , AND OVERS, L. 2009. secFleck: A public key technology platform for wireless sensor networks. In *Proceedings of the 6th European Conference on Wireless Sensor Networks (EWSN)*.

HUI, J. W. AND CULLER, D. 2004. The dynamic behavior of a data dissemination protocol for network programming at scale. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor (SenSys)*. ACM Press, New York, NY, 81–94.

KARLOF, C., SASTRY, N., AND WAGNER, D. 2004. TinySec: A link layer security architecture for wireless sensor networks. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys)*. ACM Press, New York, NY, 162–175.

LEVIS, P., MADDEN, S., POLASTRE, J., SZEWCZYK, R., WHITEHOUSE, K., WOO, A., GAY, D., HILL, J., WELSH, M., BREWER, E., AND CULLER, D. 2005. TinyOS: An operating system for sensor networks. *Ambient Intell.*, 115–148.

LIU, A. AND NING, P. 2008. TinyECC: A configurable library for elliptic curve cryptography in wireless sensor networks. In *Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN)*. IEEE Computer Society Press, Los Alamitos, CA, 245–256.

LUK, M., MEZZOUR, G., PERRIG, A., AND GLIGOR, V. 2007. MiniSec: a secure sensor network communication architecture. In *Proceedings of the 6th International Conference on Information Processing in Sensor Networks (IPSN)*. ACM Press, New York, NY, 479–488.

MAINWARING, A., CULLER, D., POLASTRE, J., SZEWCZYK, R., AND ANDERSON, J. 2002. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications (WSNA)*. ACM Press, New York, NY, 88–97.

NEEDHAM, R. AND WHEELER, D. 1997. TEA extensions. Tech. rep., University of Cambridge, Cambridge, UK.

NILSSON, D. K., ROOSTA, T., LINDQVIST, U., AND VALDES, A. 2008. Key management and secure software updates in wireless process control environments. In *Proceedings of the 1st ACM Conference on Wireless Network Security (WiSec)*. ACM Press, New York, NY, 100–108.

RIVEST, R. L., SHAMIR, A., AND ADLEMAN, L. 1978. A method for obtaining digital signatures and public-key cryptosystems. *Comm. ACM 21*, 2, 120–126.

SESHADRI, A., LUK, M., AND PERRIG, A. 2008. SAKE: Software attestation for key establishment in sensor networks. In *Proceedings of the 4th IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS)*. Springer-Verlag, Berlin, Heidelberg, Germany, 372–385.

SESHADRI, A., LUK, M., PERRIG, A., VAN DOORN, L., AND KHOSLA, P. 2006. SCUBA: Secure code update by attestation in sensor networks. In *Proceedings of the 5th ACM Workshop on Wireless Security (WiSec)*. ACM Press, New York, NY, 85–94.

SESHADRI, A., PERRIG, A., VAN DOORN, L., AND KHOSLA, P. 2004. SWATT: Software-based attestation for embedded devices. In *Proceedings of the IEEE Symposium on Security and Privacy*. 272–282.

SHAMIR, A. AND TROMER, E. 2003. On the cost of factoring RSA-1024. *RSA CrytoBytes. 6*, 2.

SHANECK, M., MAHADEVAN, K., KHER, V., AND KIM, Y. 2005. Remote software-based attestation for wireless sensors. In *Proceedings of the ESAS*. 27–41.

SIKKA, P., CORKE, P., AND OVERS, L. 2004. Wireless sensor devices for animal tracking and control. In *Proceedings of the 1st EmNetS*. 446–454.

SZEWCZYK, R., OSTERWEIL, E., POLASTRE, J., HAMILTON, M., MAINWARING, A., AND ESTRIN, D. 2004. Habitat monitoring with sensor networks. *Comm. ACM 47*, 6, 34–40.

TAN, H., JHA, S., OSTRY, D., ZIC, J., AND SIVARAMAN, V. 2008. Secure multi-hop network programming with multiple one-way key chains. In *Proceedings of the first ACM Conference on Wireless Network Security (WiSec)*. ACM Press, New York, NY, 183–193.

TCG. 2007. TCG specification architecture overview. Tech. rep. Trust Computing Group. Trusted Computer Group, Beaventon, OR.

WARK, T., CROSSMAN, C., HU, W., GUO, Y., VALENCIA, P., SIKKA, P., CORKE, P., LEE, C., HENSHALL, J., PRAYAGA, K., O'GRADY, J., REED, M., AND FISHER, A. 2007. The design and evaluation of a mobile Sensor/actuator network for autonomous animal control. In *Proceedings of the 6th International Conference on Information Processing in Sensor Networks (IPSN)*. ACM Press, New York, NY, 206–215.

WATRO, R., KONG, D., FEN CUTI, S., GARDINER, C., LYNN, C., AND KRUUS, P. 2004. TinyPK: Securing Sensor networks with public key technology. In *Proceedings of the 2nd ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN)*. ACM Press, New York, NY, 59–64.

Received January 2009; revised June 2009; accepted December 2009