secFleck: A Public Key Technology Platform for Wireless Sensor Networks

Wen Hu, Peter Corke, Wen Chan Shih, and Leslie Overs

Autonomous Systems Laboratory, CSIRO ICT Centre, Australia
{wen.hu, peter.corke,teddy.wen-chan, leslie.overs}@csiro.au,
WWW home page: http://www.sensornets.csiro.au

Abstract. We describe the design and implementation of a public-key platform, secFleck, based on a commodity Trusted Platform Module (TPM) chip that extends the capability of a standard node. Unlike previous software public-key implementations this approach provides E-Commerce grade security; is computationally fast, energy efficient; and has low financial cost — all essential attributes for secure large-scale sensor networks. We describe the secFleck message security services such as confidentiality, authenticity and integrity, and present performance results including computation time, energy consumption and cost. This is followed by examples, built on secFleck, of symmetric key management, secure RPC and secure software update.

1 Introduction

Wireless sensor network (WSN) applications [6, 12, 1, 2, 21] are growing. While the importance of security and privacy is generally agreed, it is still largely ignored since the problem is considered impractical to solve given the limited computation and energy resources available at node level. In the future, privacy, authenticity and security will be required for WSN gathered resource utilization (for billing purposes), and authenticity and security of WSN management commands and program downloads. The lesson from the PC industry is that ignoring security at the outset leads to huge pain when the technology becomes ubiquitous.

Symmetric (shared) key algorithms are tractable on mote-class hardware and can achieve message confidentiality. However, key distribution and management remains a significant practical challenge, and these algorithms poorly support message authenticity and integrity. On the Internet, Public Key Cryptography (PKC) is widely used to support symmetric key management, as well as message authenticity and integrity. Researchers have investigated methods to support PK technology in WSN [22, 15]. Such approaches have focused on software-based PK 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 compromises the security level of asymmetric encryption.

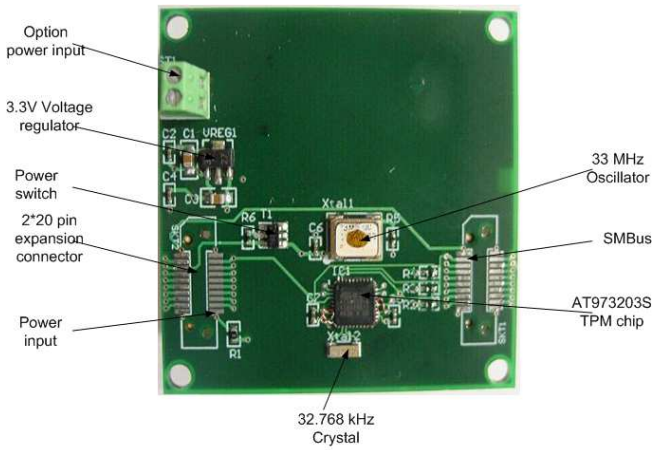

Fig. 1. secFleck TPM module (upper side) — an expansion board for the Fleck WSN node.

In this paper, we introduce the design and implementation of secFleck, a PK platform that uses Trusted Platform Module (TPM) hardware to augment the node. Our evaluation shows that the secFleck provides Internet-level PK services with reasonable energy consumption and financial overhead. The contributions of this paper include:

- The design and implementation of the secFleck platform, which includes a standard TPM chip and a set of software primitives, to support Public Key Cryptography (PKC) in a WSN. To the best of our knowledge, the secFleck is the first platform that supports most RSA-based PKC functions (encryption, decryption, signature, and signature verification) in WSN. RSA is the most widely used PKC in the traditional networks such as the Internet.
- Extensive evaluation of the secFleck platform in terms of computation time, energy consumption, memory footprint and cost. The results demonstrate the feasibility of the secFleck platform.
- The demonstration that the secFleck platform is easy-to-use through case studies of how to implement key management, secure software update, and secure Remote Procedure Calls (RPC) services using the secFleck primitives.

The rest of this paper is organized as follows. In section 2, we give a brief overview of the RSA algorithm, which secFleck is based on, followed by a detailed description of the software and hardware architecture of secFleck (Section 3). We evaluate the performance of secFleck in terms of computation time, energy consumption and financial cost in Section 4. Section 5 describes, by means of a case study, how secFleck primitives can be used to implement state-of-the-art key management and secure software update protocol, and improve the proto-

col's performance. We present related work in Section 6. Finally, we finish with conclusions and future work in Section 7.

2 A brief introduction to the RSA algorithm

In this section, we provide an overview of the Rivest Shamir Adelman (RSA) algorithm [18], which secFleck is motivated by and built upon. 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 to the RSA algorithm computation complexity and security levels.

RSA is an algorithm for public key cryptography (PKC), also called asymmetric cryptography, in which the encryption key is different to the decryption key. RSA is the first algorithm that is suitable for signing and encryption, and is used widely in secure communication protocols, such as Secure Shell (SSH) and Secure Sockets Layer (SSL), in the Internet.

The RSA algorithm generates a public key and a private key simultaneously as follows. First, RSA chooses two large random 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 public exponent) such that:

$$1 < e < \varphi(n), \tag{1}$$

and

$$gcd(e, \varphi(n)) = 1, \tag{2}$$

where gcd stands for the Greatest Common Divisor (GCD). Public exponent (e) and modulus (n) together comprise the public key.

Fifth, RSA calculates private exponent (d) by

$$de \equiv \ mod\ \varphi(n), \tag{3}$$

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

$$c = m^e\ mod\ n, \tag{4}$$

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

$$m = c^d\ mod\ n. \tag{5}$$

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 Eq. (5). After receiving ($H(m)$, s), Bob uses the same hash

function, together with Alice's public key (n, e), to verify the signature by Eq. (4).

Because the sizes of p and q are approximately half of the size of the key size (k), the security level of RSA cryptography is a function of e and k. A popular choice for the public exponent is $e = 2^{16} + 1 = 65,537$. Using small e values such as 3, 5 or 17 can dramatically reducing computational cost, but will lead to greater security risks [18]. The default e value in secFleck is 65,537. It is common to believe that a RSA key size of 512 bits is too small to use nowadays. Bernstein has proposed techniques that simplify brute-forcing RSA [3], and other work based on [3] suggests that 1024-bit RSA keys can be broken in one year by a device that costs $10 million rather than trillions as in previous predictions [19]. It is currently recommended to use an RSA key at least 2048 bit long. Therefore, the default RSA key size in secFleck is 2048 bits.

3 Platform Architecture

In this section, we discuss both hardware and software modules in secFleck.

3.1 Hardware module

The core of secFleck is an Atmel AT97SC3203S TPM chip (see Fig. 1) mounted on a Fleck expansion board (see Fig. 2). The TPM chip follows version 1.2 of Trusted Computing Group (TCG) specification for TPM. It has 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 500ms according to the Atmel data sheet.

Fig. 1 is a picture of TPM board. The TPM board, connected to a Fleck, can be enabled to meet WSN application requirements. The TPM board and the Fleck is shown in Fig. 2. The Fleck is a wireless sensor node that features an Atmega 1281 micro controller (8 MHz clock rate and 8 KB memory) and a Nordic nRF905 radio [4]. The TPM module has a 100 kHz SMBus which is similar to the I2C and the TPM is connected to the Fleck's I2C interface. The SMBus makes the TPM chip easily integrated in embedded systems.

Fig. 3 shows a block diagram of the TPM module, which includes the bare minimum of components required for operation: TPM chip, crystal, oscillator, voltage regulator, power switch and an expansion connector.

3.2 Software module

For the ease of WSN application developers, we have implemented a set of RSA public key cryptographic primitives as a Fleck OS (FOS) [4] module, which include encryption, decryption, signing, and signature verification etc., as well as XTEA symmetric cryptographic primitives (see Fig. 4). FOS is a C-based cooperative multi threaded operating system for WSN.

Asymmetric key (RSA) FOS functions Previous research [11] shows 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. secFleck allows applications to duty cycle TPM component by calling primitives `fos_tpm_startup()` and `fos_tpm_turnoff()`. These duty-cycle primitives are more important in secFleck because its current consumption (around 50 mA) is significantly more than Fleck's average current consumption (around 5 mA).

Symmetric keys are typically generated by a pseudo-random number generator in previous work [14]. If an attacker can extract the initial random symmetric key, then it is possible for the attacker to compute all past and future keys. Therefore, a high quality random number generator is very important for the effectiveness of symmetric key operations. secFleck provides a `fos_tpm_rand()` primitive, which is based on a true Random Number Generator (RNG), and is Federal Information Processing Standards (FIPS) 140-2 compliant.

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, and a public key — the cipher text is returned. Similarly, an application can decrypt cipher text. The secFleck encryption and decryption facilitates message confidentiality.

secFleck provides two additional primitives, i.e., `fos_tpm_sign()` and `fos_tpm_verifySign()`, for applications to sign messages or to verify the signatures of messages. The `digest` parameter in these two primitives are generated by the Secure Hash Algorithm (SHA-1) of plain messages. FOS also provides a function for computing SHA-1.

A base station typically has more computation, memory and energy resources and can be treated as a Certificate Authority (CA). All the nodes store the CA's public key in their permanent memories such as 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.

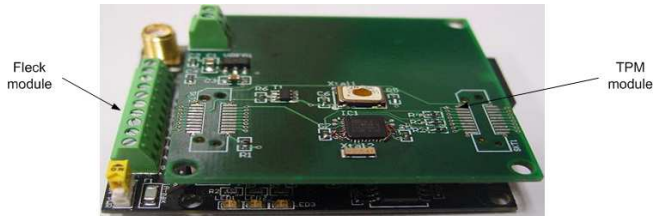

Fig. 2. secFleck (Fleck3 and TPM module).

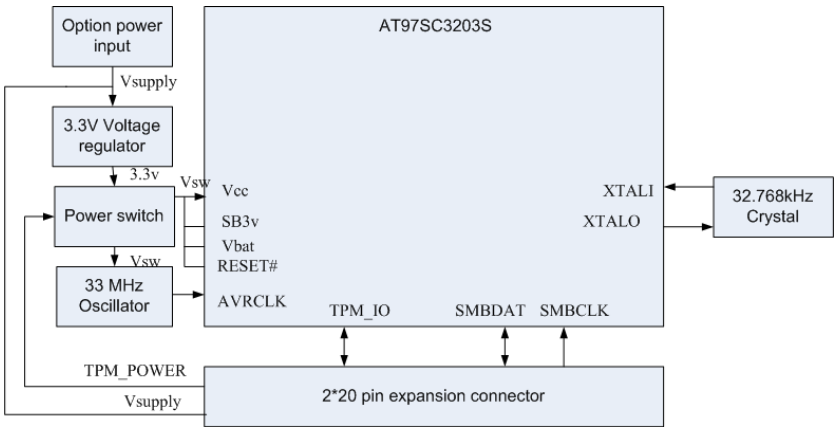

Fig. 3. secFleck TPM module block diagram.

Symmetric key (XTEA) FOS functions Previous work [14] show 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. secFleck also features a 128-bit symmetric block cipher based on eXtended Tiny Encryption Algorithm (XTEA) [16]. XTEA operates on 64-bit blocks with 32 or 64 rounds. secFleck chooses XTEA symmetric key cryptography because of its small Random Access Memory (RAM) footprint, which makes it a good candidate for tiny 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 Performance Evaluation

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

4.1 Asymmetric key (RSA) operations

Table 1. Comparison of RSA encryption times.

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

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

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

/* secFleck 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);
```

Fig. 4. secFleck application interface for public key infrastructure and symmetric session key functions. The public key cryptographic primitive interfaces allow applications to start up and turn off on-board 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 by XTEA algorithm.

As part of our benchmarking we also implemented the RSA encryption algorithm in software for comparison. Table 1 shows the encryption time for different key sizes and RSA public exponents (e) in both software and hardware implementation. The results show that the TPM chip can reduce the computation time of RSA encryption by a factor of $8,000$, when $e = 65{,}537$ and key size is 2048 bits. Table 1 also shows that software RSA implementation is impractical using embedded micro controllers 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.

We have not implemented the RSA decryption algorithm in software because it is significantly more computationally intensive than the RSA encryption al-

Table 2. RSA computation time in secFleck for $e = 65,537$ and 2048 bit key.

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

gorithm (see Table 2). Table 2 also shows RSA encryption, decryption, sign and signature verification computation time in secFleck.

Table 3. secFleck 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 3 shows the current consumption for different secFleck operations. It shows that RSA operations consume 37% to 65% more current than transmitting in secFleck. Table 4 shows the energy consumption of 2048-bit RSA encryption operation when $e = 65,537$. It shows that the software-based approach consumes around 1,300 times more energy compared to secFleck for an RSA encryption operation. Table 4 also shows that the software-based approach RSA encryption in WSN is indeed impractical in terms of both computation time and energy consumption for reasonable RSA exponent and key size. On the other hand, secFleck makes it feasible to support PK technology for WSN.

Table 4. secFleck (RSA and XTEA) encryption energy consumption for *one 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

4.2 Symmetric key (XTEA) operations

We tested the performance of XTEA cryptography to determine its computation speed on the Fleck platform. secFleck can encrypt one block of 64-bit data in approximately 1.15 ms. Therefore, it takes approximately 18 μs (Table 4, Row 3) to encrypt one bit data. Furthermore, 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 four-byte address and a two-byte Cyclic Redundancy Check-16 (CRC-16) overheads. Therefore, the available bandwidth for 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 one bit, which is significantly longer than the encryption time (17.97 μs).

Table 4 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 ten times less energy compared to hardware RSA encryption, and approximately 12,000 times less energy compared to software RSA encryption. It suggests that, in energy-impoverished WSN, we should use symmetric cryptography for most secure communications, and should use asymmetric cryptography in critical tasks only (i.e., the symmetric key management).

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 1,082 bytes of program space only.

4.3 The financial cost of secFleck

An Atmel AT97SC3203S TPM chip costs $4.5 when ordered in quantities[1], which is less than 5% of the cost of popular sensor devices such as Telosb, Iris mote, and Fleck (about $100). The TPM chip is small in size (Figure 1) measuring just 6.1 $\times$ 9.7 mm and is less than 2% of the area of the Fleck and could be integrated onto a future version rather than the cumbersome expansion board used in this prototype.

5 Case studies

In this section, we demonstrate the power of our secFleck primitives (shown in Fig. 4) to easily and efficiently realize secure WSN applications. These applications include, but are not limited to, secure over-the-air programming, secure Remote Procedure Calls (RPC), and secure session key management. We have chosen to implement variants of state-of-the-art key management [17] and secure software update protocol [13] with secFleck primitives, and show how secFleck primitives can improve the protocol's performance.

[1] `http://www.atmel.com/dyn/products/view_detail.asp?ref=&FileName=embedded10_18.html&Family_id=620` (accessed on 18th June, 2008).

5.1 Symmetric session key encryption/decryption

Symmetric key cryptography consumes significantly less energy than RSA asymmetric key cryptography (see Table 4), as 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 or hourly (also called the rekey process). We will discuss the rekey process in detail later.

By utilizing two secFleck primitives, it is easy to achieve symmetric key cryptography in secFleck (see Fig. 5). `fos_xtea_getkey()` (Line 4) reads a symmetric key from secFleck 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.

```
1 #DEFINE NROUNDS 64
2
3 /* XTEA encryption in secFleck. */
4 fos_xtea_getkey(key, location);
5 fos_xtea_storekey(key, location);
6 fos_xtea_encipher(msg, key, cipher, NROUNDS);
```

Fig. 5. XTEA encryption with secFleck primitives.

An application can choose to store the session keys in Fleck RAM, EEPROM, or the TPM EEPROM. When the keys are stored in the Fleck RAM or EEPROM, the key (getting and storing) operations consumes significantly less energy than when the keys are stored in the TPM EEPROM. However, storing the keys in the Fleck also exposes the keys to more risks. Hartung et al. demonstrated how to extract the information in a node's EEPROM and RAM within one minute in [10]. 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 five minutes); store the key in the Fleck memory for those high-throughput operations (e.g., secure over-the-air-programming).

5.2 Sensor node symmetric session key request/assignment operation

Fig. 6 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, e.g., hourly or daily, by generating a random number (N_a) and encrypting N_a with the Request (Req) command using Base's public key (Pk_{Base}) before transmitting it to the base. 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. Fig. 6 also shows the five secFleck 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, e.g., in an group key establishment event (see Section 5.3), by the base station instead of a node.

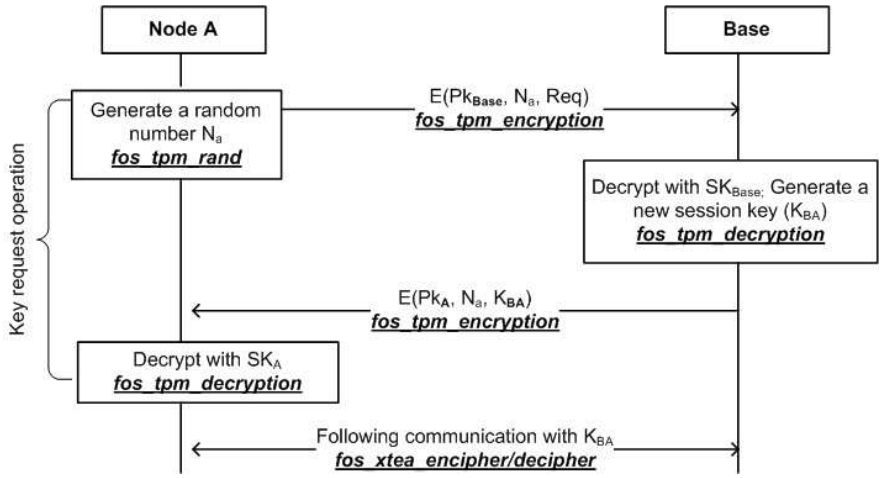

Fig. 6. Symmetric session key request operation with secFleck primitives (underline). Node A request a session key from a base station.

5.3 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 node 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 5.2. 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 Node B and C via two session key assignment operations (see Section 5.2) before transmitting K_{abc} to Node A. Then, Node A, B, and C begin secure communications using group session key K_{abc}.

5.4 Secure software update protocol

Multihop Over the Air Programming (MOAP) protocols such as Deluge [13] 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 5.3, secFleck 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 secFleck 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 secFleck node can verify the signature of the a 256 byte page in 59 ms (see Table 2), which is more than 4,300 bytes/second. This secFleck signature verification rate is approximately 50 times faster than the average 88.4 bytes/second dissemination rate achieved by Deluge in a 75 node network [13].

5.5 Backward secrecy and forward secrecy

secFleck can enhance the security levels of the rekey process by providing backward secrecy 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 [10] because of the exposed nature of nodes in WSN. Furthermore, the attacker can also extract the initial random key used by the software pseudo-random 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), secFleck can increase the security level of the protocols. It is very difficult, if not impossible, for the attacker, who has obtained the current symmetric link key, to find out the past or future keys generated by a true RNG. An application can obtain a true random number by calling `fos_tpm_rand()` primitive (Line 6, Fig. 4).

5.6 Secure Remote Procedure Calls

The Fleck Operating System (FOS) uses Remote Procedure Calls (RPC) to allow application programs to seamlessly access services on one or more sensor nodes.

RPC actions	Description	Cryptography
assign_session_key	assign a new symmetric session key to a node	PK
request_session_key	request a new symmetric session key from a base	PK
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

Table 5. Common secure FOS RPC actions.

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 5, in addition to application specific actions.

An RPC call message comprises the function arguments, the function enumerator, sequence number, node id of the caller and a CRC-32. 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 5.1). `assign_session_key` and `request_session_key` RPC messages are encrypted and signed with PKC introduced in Section 5.2. 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.

In a sensor network, it is possible to broadcast the RPC call encrypted by a group symmetric key (see Section 5.3), 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 in secFleck primitives, provides privacy of commands and return values, authentication and immunity to replay attacks.

6 Related work

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

Rivest Shamir Adelman (RSA) is the most widely used Public Key Cryptography (PKC) in the Internet, and a comprehensive guide to RSA is available in [18]. RSA is much slower than Xtended Tiny Encryption Algorithm (XTEA) [16] and other (shared) symmetric cryptography such as TinySec [14].

It is our thesis that most of the secure communications in resource-constrained WSN will be based on symmetric cryptography. A symmetric key can be discovered by an attacker by extracting it directly from a captured node via a JTAG or similar device [10] because of the distributed and embedded nature of nodes in WSN. Therefore, an effective symmetric key establishment and management scheme is of prime importance. RSA and Diffie Hellman key agreement techniques [8] are widely used key agreement protocols in 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 [22]. However, the security levels of RSA cryptography is severely compromised by using smaller public exponents and key sizes. Recently, the importance of symmetric key cryptography and the critical roles of key management mechanism in WSN was observed by Nilsson et al. [17] who proposed an efficient symmetric key management protocol for the WSN. However, they have focused on the protocol design and formal verification, and have not addressed the resource constraint problems in implementing the protocol.

The research community is developing faster and more energy efficient PKC algorithms such as Tiny Elliptic Curve Cryptography TinyECC [15] for the resource-impoverished WSN. While TinyECC shows the most promise to run at usable speeds on WSN nodes [9], there are concerns related to patents, which are one of the main factors limiting the widely acceptance of ECC. In this regard we note that the RSA and XTEA algorithms used in this work is in the public domain.

While Multihop Over the Air Programming (MOAP) protocols [13] enable application users to program and reprogram WSNs easily, it also opens the door for unauthorized users to implant malicious code into the WSN. Dutta et al. attempt to secure the MOAP [7] by introducing program authenticity and integrity with a cut-down version of software-based RSA PKC similar to TinyPK [22]. 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 the resource-impoverished WSN, researchers have investigated alternative methods to ensure program authenticity and integrity by secure hash chain, hash tree and/or their hybrid [5, 20].

In contrast to the existing alternatives of PKC that typically have limited functions, secFleck 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.

7 Conclusion and future work

We have presented secFleck, a TPM-based PK platform for sensor networks that facilitates message security services such as confidentiality, authenticity and integrity. Our evaluation shows that secFleck provides Internet-level public-key services quickly, with low energy consumption and at low cost in terms of parts and board size. This is followed by examples, built on secFleck, of symmetric key management, secure RPC and secure software update, which demonstrates that the secFleck platform is easy-to-use.

Our next step is to investigate other features of TPM module such as secure storages and remote attestations.

8 Acknowledgments

The authors thank Hailun Tan (Univeristy of New South Wales, Australia), Dr. Juanma Gonzalez Nieto (Queensland University of Technology, Australia) and the anonymous reviewers for their comments and suggestions.

References

1. Habitat monitoring on great duck island. http://www.greatduckisland.net/index.php.
2. Habitat monitoring on james reserve. http://www.jamesreserve.edu/.
3. B. Bernstein. Circuits for integer factorization: A proposal. *Manuscript, http://cr.yp.to/papers.html*., 2001.
4. P. Corke and P. Sikka. Demo abstract: FOS — a new operating system for sensor networks. In *Fifth European conference on wireless sensor networks (EWSN 2008)*, Bologna, Italy, Jan, 2008.
5. J. Deng, R. Han, and S. Mishra. Secure code distribution in dynamically programmable wireless sensor networks. In *IPSN '06: Proceedings of the fifth international conference on Information processing in sensor networks*, pages 292–300, New York, NY, USA, 2006. ACM.
6. T. L. Dinh, W. Hu, P. Sikka, P. Corke, L. Overs, and S. Brosnan. Design and deployment of a remote robust sensor network: Experiences from an outdoor water quality monitoring network. In *Second IEEE Workshop on Practical Issues in Building Sensor Network Applications (SenseApp 2007)*, Dublin, Ireland, Oct. 2007.
7. P. K. Dutta, J. W. Hui, D. C. Chu, and D. E. Culler. Securing the deluge network programming system. In *IPSN '06: Proceedings of the fifth international conference on Information processing in sensor networks*, pages 326–333, New York, NY, USA, 2006. ACM.

8. S. Goldwasser. New directions in cryptography: twenty some years later (or cryptograpy and complexity theory: a match made in heaven). In *FOCS '97: Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS '97)*, page 314, Washington, DC, USA, 1997. IEEE Computer Society.
9. N. Gurn, A. Patel, A. Wander, H. Eberle, and S. C. Shantz. Comparing elliptic curve cryptography and rsa on 8-bit cpus. *Lecture Notes in Computer Science*, 3156:119–132, 2004.
10. C. Hartung, J. Balasalle, and R. Han. Node compromise in sensor networks: The need for secure systems. Technical report, University of Colorado at Boulder, January 2005.
11. J. Hill and D. Culler. Mica: a wireless platform for deeply embedded networks. *IEEE Micro*, 22(6):12–24, Nov. 2002.
12. W. Hu, V. N. Tran, N. Bulusu, C. T. Chou, S. Jha, and A. Taylor. The design and evaluation of a hybrid sensor network for cane-toad monitoring. In *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*, page 71, Piscataway, NJ, USA, 2005. IEEE Press.
13. J. W. Hui and D. Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 81–94, New York, NY, USA, 2004. ACM.
14. C. Karlof, N. Sastry, and D. Wagner. Tinysec: a link layer security architecture for wireless sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 162–175, New York, NY, USA, 2004. ACM.
15. A. Liu and P. Ning. Tinyecc: A configurable library for elliptic curve cryptography in wireless sensor networks. In *IPSN '08: Proceedings of the 2008 International Conference on Information Processing in Sensor Networks (ipsn 2008)*, pages 245–256, Washington, DC, USA, 2008. IEEE Computer Society.
16. R. Needham and D. Wheeler. Tea extensions. Technical report, University of Cambridge, October 1997.
17. D. K. Nilsson, T. Roosta, U. Lindqvist, and A. Valdes. Key management and secure software updates in wireless process control environments. In *WiSec '08: Proceedings of the first ACM conference on Wireless network security*, pages 100–108, New York, NY, USA, 2008. ACM.
18. R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.
19. A. Shamir and E. Tromer. On the cost of factoring rsa-1024. *RSA CrytoBytes.*, 6(2), 2003.
20. H. Tan, S. Jha, D. Ostry, J. Zic, and V. Sivaraman. Secure multi-hop network programming with multiple one-way key chains. In *WiSec '08: Proceedings of the first ACM conference on Wireless network security*, pages 183–193, New York, NY, USA, 2008. ACM.
21. T. Wark, C. Crossman, W. Hu, Y. Guo, P. Valencia, P. Sikka, P. Corke, C. Lee, J. Henshall, K. Prayaga, J. O'Grady, M. Reed, and A. Fisher. The design and evaluation of a mobile sensor/actuator network for autonomous animal control. In *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*, pages 206–215, New York, NY, USA, 2007. ACM Press.
22. R. Watro, D. Kong, S. fen Cuti, C. Gardiner, C. Lynn, and P. Kruus. Tinypk: securing sensor networks with public key technology. In *SASN '04: Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks*, pages 59–64, New York, NY, USA, 2004. ACM.