Combined Secure Storage and Communication for the Internet of Things

Ibrahim Ethem Bagci[+], Shahid Raza[*], Tony Chung[+],
Utz Roedig[+], Thiemo Voigt[*,†]
[+]School of Computing and Communications, Lancaster University, Lancaster, UK
{i.bagci, u.roedig}@lancaster.ac.uk, a.chung@theiet.org
[*]SICS Swedish ICT, Kista, Sweden
{shahid, thiemo}@sics.se
[†]Uppsala University, Sweden

***Abstract*—The future Internet of Things (IoT) may be based on the existing and established Internet Protocol (IP). Many IoT application scenarios will handle sensitive data. However, as security requirements for storage and communication are addressed separately, work such as key management or cryptographic processing is duplicated. In this paper we present a framework that allows us to combine secure storage and secure communication in the IP-based IoT. We show how data can be stored securely such that it can be delivered securely upon request without further cryptographic processing. Our prototype implementation shows that combined secure storage and communication can reduce the security-related processing on nodes by up to 71% and energy consumption by up to 32.1%.**

I. Introduction

The Internet of Things (IoT) is becoming a reality and vast numbers of smart objects are interconnected via the Internet Protocol (IP). A number of applications in this context handle sensitive information. For example, smart objects may be used for patient monitoring in hospitals, implementations of security systems in airports or to monitor crucial business processes in factories. Thus, security mechanisms are required to ensure confidentiality, integrity and authenticity of the collected information.

Due to resource limitations of smart objects it is not feasible to use the existing IP protocol throughout the entire IoT. IP header compression, as defined in the 6LoWPAN [1] framework, is used in wireless IEEE 802.15.4 networks which smart objects generally use for interconnectivity. 6LoWPAN header compression and decompression is carried out by gateway nodes when relaying packets between IEEE 802.15.4 networks and the existing IP network infrastructure.

As the IoT relies on the established and tested IP protocol it is reasonable to also use security mechanisms defined in this context. The IPsec [2] framework defines security mechanisms for IP networks and it is supported by nearly all hosts currently in use. A definition of IPsec 6LoWPAN extensions [3] exists which allows smart objects to participate in IPsec secured communication. Thus, secure communication in the IoT using standardised mechanisms is feasible.

Smart objects now provide vast amounts of storage space due to the recent advances in flash memory technology. IoT applications rely on this storage space in order to improve system performance [4]. It is therefore becoming more important to not only secure communication but to also protect sensitive data while it is stored on smart objects. Various secure storage solutions exist that can be used to protect data on nodes. For example, Codo [5] is an extension of the Contiki [6] CFS [7] filesystem that provides security services.

The previously outlined secure communication and storage solutions have been developed individually. It is not taken into account that tasks such as key exchange or cryptographic processing are executed for both system components. Thus, in many situations cryptographic work performed by smart objects is unnecessarily carried out twice or more. Given that smart objects are very resource limited devices it is desirable to prevent such process duplication. Freed resources may be used to reduce hardware complexity, improve energy consumption or to add additional application features.

We address the previously outlined shortcoming of existing solutions and provide a design of a combined secure storage and communication framework that allows us to reduce security related processing on smart objects (see Fig 1). In particular we consider the IP, 6LoWPAN and IPsec standards as the base for our work. We believe that a standard compliant solution is more desirable than a proprietary system. Furthermore, it is safer to build on tested and trusted security mechanisms rather than designing an entirely novel mechanisms. Data is stored securely on the flash file system such that it can be directly used for secure transmission. This is not a trivial task as packet header content of future transmissions must be considered when securing data for storage. We show in this paper that an IP based combined secure storage and communication solution is possible and that this can save up to 71% of a node's security related processing effort. A cost in regards to additional storage space is incurred as a result of the secure storage; however, given that smart objects can now provide ample amounts of storage space we do not see this as limiting factor. The specific contributions of this paper are:

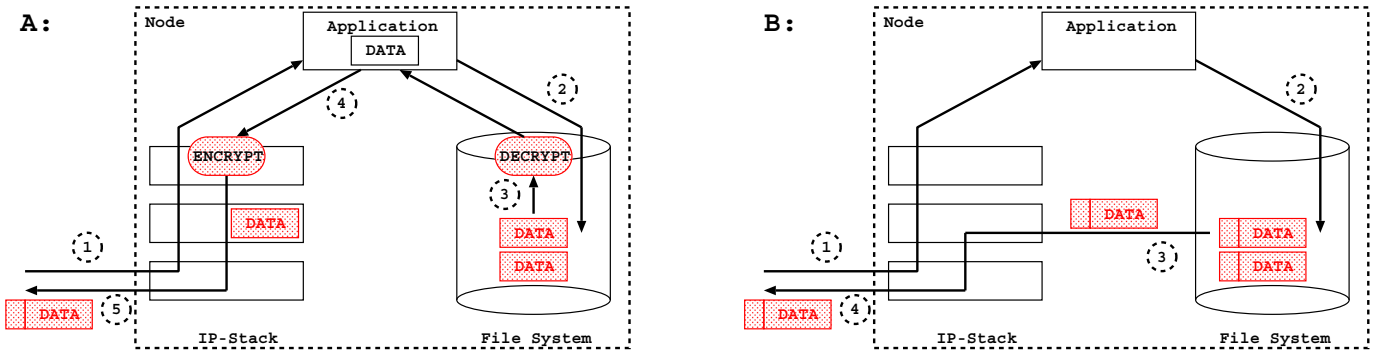


Figure 1: *A: Traditional Operation:* 1 - Data is requested from the node. 2 - The application forwards the request to the filesystem. 3 - The data is decrypted and passed to the application. 4 - The application sends data for transmission to the IP stack which secures the data. 5 - The data is transmitted.
B: Combined Secure Storage and Communication: 1 - Data is requested from the node. 2 - The application forwards the request to the filesystem. 3 - The secured data is directly passed into the IP stack. 4 - Data is transmitted without cryptographic processing.

- The definition of a framework for combined secure storage and communication for IP/6LoWPAN networks.
- An implementation of the framework for the Contiki operating system.
- A detailed evaluation of the performance gains of the framework.

The next section of the paper discusses related work in the area of secure communication and storage for smart objects. Section III describes the proposed combined secure storage and communication framework and its implementation for the Contiki OS, which is then evaluated in detail in Section IV. Section V discusses our findings and concludes the paper.

II. Related Work

Security in the IoT is a research topic that has attracted a lot of interest. Work has been carried out to improve efficiency of cryptographic algorithms [8], to provide specialised hardware support [9], to organize key distribution [10], to define secure communication protocols [3] and to organise secure data storage [11]. Solutions for secure communication and secure storage of data in the IP based IoT exist, but these functions are generally designed and operated independently of each other. To the best of our knowledge, this is the first work which aims to combine both aspects. Thus, the following shall discuss these IoT security aspects separately.

Secure Communication for the IoT: : Communication in the IoT can be secured on different layers. The IoT uses the IEEE 802.15.4 [12] link-layer. IEEE 802.15.4 link-layer security is the current state-of-the-art security solution for the IP-connected IoT; it defines data encryption and integrity verification.

IEEE 802.15.4 security does not provide end-to-end security when connecting a IEEE 802.15.4 network via a gateway router to the existing Internet. Thus, additional solutions exist which protect data traveling from Internet hosts to the border router. For example, ArchRock PhyNET [13] applies IPsec in tunnel mode between the gateway router and Internet hosts.

To achieve true end-to-end security between Internet hosts and smart objects an IPsec extension for 6LoWPAN has been proposed [3]. Unmodified Internet hosts can communicate directly with smart objects. The border router applies 6LoWPAN header compression in order to enable efficient transport of IPsec packets in IEEE 802.15.4 networks. We use this mechanism for our framework.

End-to-end security can be provided by using Transport Layer Security (TLS) or its predecessor Secure Sockets Layer (SSL). SSL has been proposed as security mechanism for the IoT by Hong et al. [14]. Foulagar et al. propose a TLS implementation for smart objects [15].

Secure Storage in the IoT: There are a number of secure storage solutions available [5], [11], [16] and [17]. Codo [5] is a security extension for the Coffee [7] filesystem in the Contiki [6] OS. Codo optimises performance of security operations by enabling caching of data for bulk encryption and decryption. We use Codo as a base for the work presented in this paper.

III. The Secure Storage and Communication Framework

Our proposed secure storage and communication framework is based on the established IPv6/6LoWPAN protocols. IPv6/6LoWPAN defines IPsec/ESP (Encapsulating Security Payload) that provides encryption and authentication of transmitted data packets. We use the same cryptographic methods and data formats defined by ESP for data processing before storage. This requires us to store not only data but also all header information that is involved in the cryptographic processing. Encrypted data must be stored in ESP compatible form such that requested data can be transmitted over the network without further cryptographic processing. This requires us to anticipate content of communication protocol header fields such as IP destination addresses, sequence numbers and checksums at storage time. As IPsec is the base for communication and storage, the existing key exchange mechanisms defined for IPsec can be reused for the storage element of the framework.

The next subsection describes IPsec/ESP usage in 6LoWPAN networks. This represents the communication element of our framework. Thereafter follows a description of the storage

element of the framework. We then briefly discuss application layer protocols that may be used with the framework and describe our Contiki based implementation. Finally we discuss expected performance gains and cost in terms of storage overhead and provide a security analysis.

A. Communication Component

IPv6 uses IPsec [2] to secure IP communication between two end points. IPsec is a collection of protocols that include Authentication Header (AH), which provides authentication services, and Encapsulating Security Payload (ESP), which provides both authentication and privacy services. A suite of encryption and authentication algorithms are also defined. A node keeps track of security associations (SA) that specify how IP flows are treated in terms of security.

In an ESP [18] packet data (for example, a UDP packet), padding, pad length and next header information are encrypted. All header information may be authenticated using the optional Integrity Check Value (ICV). In an 802.15.4 network an ESP header will not be transmitted directly. Its compressed form as defined by 6LoWPAN is used instead to reduce header overheads.

6LoWPAN defines header compression mechanisms. LOWPAN_IPHC is used for IP header compression and LOWPAN_NHC for the next header compression. The NH field in LOWPAN_IPHC when set to 1 indicates that the next header following the compressed IPv6 header is encoded with LOWPAN_NHC. LOWPAN_NHC has a length of 1 or more octets, where the first variable length bits identify the next header type and the remaining bits are used to encode header information. Currently, 6LoWPAN defines LOWPAN_NHC for the IP extension header (LOWPAN_NHC_EH) and the UDP header (LOWPAN_NHC_UDP). A definition for ESP encoding (LOWPAN_NHC_ESP) is provided in [3] and its fields are defined as:

- The first four bits in the LOWPAN_NHC_ESP represent the NHC ID defined for ESP. These are set to 1110.
- If $SPI = 00$: the default SPI for the 802.15.4 network is used and the SPI field is omitted. We set the default SPI value to 1. This does not mean that all nodes use the same security association (SA), but that every node has a single preferred SA, identified by SPI 1.
  If $SPI = 01$: First 8 bits of the SPI are carried inline; the remaining 24 bits are elided.
  If $SPI = 10$: First 16 bits of the SPI are carried inline; the remaining 16 bits are elided.
  If $SPI = 11$: All 32 bits of the SPI are carried inline.
- If $SN = 0$: The first 16 bits of sequence number are used. The remaining 16 bits are assumed to be zero.
  If $SN = 1$: All 32 bits of the sequence number are carried inline.
- If $NH = 0$: The next header field in ESP will be used to specify the next header and it is carried inline.
  If $NH = 1$: The next header will be encoded using LOWPAN_NHC. In case of ESP this would require the end systems to perform 6LoWPAN compression/decompression and encryption/decryption jointly.

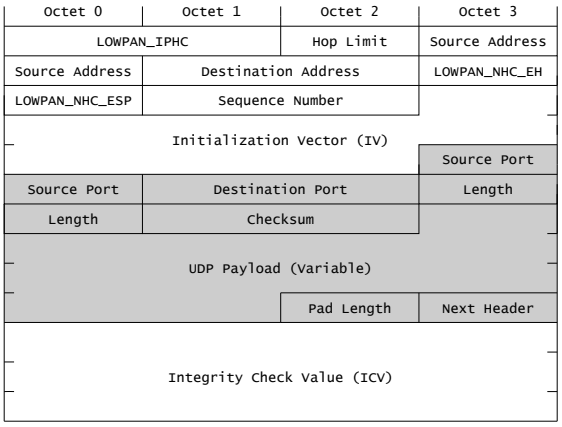


Figure 2: A compressed and ESP secured IPv6/UDP packet.

Figure 2 shows a UDP/IP packet secured with compressed ESP. An initialisation vector (IV) may be carried in the ESP packet if the selected encryption algorithm requires transmission of this information with every packet. The shaded portion represents encrypted data. Authentication can be provided using the ICV.

B. Storage Component

Data is stored securely such that it can be transmitted as ESP compliant packets on request without additional cryptographic processing. This requires storage of all cryptographically processed elements of the ESP packet within the file system. ESP header elements that are not cryptographically processed and can be constructed with little effort when data is requested and therefore do not have to be stored.

Data is stored as blocks representing the shaded part (and the ICV if authentication is required) shown in Figure 2. If data stored within a block is requested the block is read from the file system and the full packet, as shown in Figure 2, is assembled and transmitted. The receiver may only be interested in part of the received data and some undesirable transmission overhead may occur. However, typical applications will require bulk data transfer (large parts of a file) in which case such overheads do not occur. For example, for further data analysis an application may request recorded sensor samples within a particular time frame or, for performance debugging purposes, recorded link quality metrics over a longer time period may be requested.

Some stored information is dependant on the communication relationship. At the time of storage, assumptions regarding the forthcoming communication relationship must be made in order to enable cryptographic processing. Elements to be considered are:

- *UDP Header:* A UDP header is stored within the encrypted ESP payload. Assumptions regarding destination and source UDP port must be made at time of storage. The destination IP address of packets is used within the IPv6 UDP checksum calculation. Thus, IP source and destination address assumptions must be made as well.

- *Initialisation Vector IV:* The IV (if required) is used for ESP encryption. Most protocols allow a counter mode where the IV for each packet is constructed by adding a transmission sequence number to an initial IV.
- *Sequence Number:* The ESP header includes a sequence number. This sequence number is not encrypted but it is included in the ICV calculation. If ESP authentication is used a sequence number must be selected at time of storage in order to generate a ICV for storage alongside the data.

UDP Header Construction: A UDP header has to be prepared at time of data storage. The header consists of 4 fields of $2byte$ length: Source Port, Destination Port, Length and Checksum. The Length field is defined by the amount of data contained in the UDP packet. To reduce packet overheads the amount of data contained in each UDP packet is selected such that the maximum 802.15.4 frame size of $127byte$ is utilised.
The selection of a Source and Destination Port is not problematical. It can be assumed that well known ports can be used for data retreival.
The calculation of the Checksum field is challenging. The checksum is mandatory in IPv6 and is calculated using a pseudo header. This pseudo header contains the IP Source Address, the IP Destination Address, UDP Length and IP Next Header field. As the IP Destination Address is included assumptions regarding the IP address requesting stored information must be made. The checksum is calculated as the 16-bit one's complement of the one's complement sum of the pseudo header, the UDP header, and the data.
It is a reasonable assumption that a particular host is used most of the time to request information from nodes (e.g. the sink). The IP address of this node may be used for storage preparation.
In some cases data may be requested from a different node and the IP destination address used for checksum calculation does not match the destination of the data requestor. In this situation it is possible to correct the checksum in a way that does not require the decryption and encryption of all of the data again. Thus, performance is reduced as part of the stored data must be cryptographically processed before transmission but it is still beneficial in comparison with a system that does not combine secure storage and transmission (See Evaluation). ESP can use encryption algorithms which operate on blocks (e.g. AES using $16byte$ blocks). It is possible to decrypt only the first block of a larger stored ESP packet which will contain the UDP header and its checksum. Since the UDP checksum algorithm is a simple summation checksum re-calculation is trivial. By substituting the old destination address for the new destination address, a new checksum can be calculated. Now the first block of the ESP packet can be encrypted and it is ready for transmission to an alternative destination.
IV Construction: The IV does not have to be stored in the file system together with the encrypted ESP fields. An initial IV can be used and the storage block number is added to construct the IV.
Sequence Number Construction: If authentication is required it is possible to also store the ICV. As the ESP header includes a sequence number which is included in the ICV calculation, it is necessary to predict at storage time what sequence numbers will be used during communication.
Data belonging to a file is stored as sequence of ESP encrypted blocks and we can use the block number as ESP sequence number. IPsec allows us to reset the sequence number counters at the start of a communication relationship by establishing a new SA. Thereafter, data from the file can be delivered sequentially. In this setting we ensure that the communication uses sequence numbers that were selected at time of data storage.

C. Framework Usage

Application Layer Protocol: Nodes store data securely which may be requested by Internet hosts. Stored data has an application specific semantic. For example, sensor values may be stored as a $4byte$ sensor value together with a $4byte$ time stamp and $2byte$ sequence number. Nodes execute a storage application that is able to respond to queries such as "send sensor samples recorded between 12:00:00 and 13:00:00". A host executes a storage application that is able to send these requests and to process arriving data. Host and node storage applications use UDP for communication. Similar to the well known FTP protocol, separate flows are used for command and data transfer which makes different IPsec security settings (including keys and security mechanisms) for both channels possible.
Security Configuration: The IPsec Security Association (SA) defines how data flows are protected. The SA holds secret keys, encryption algorithm descriptions and IP addresses to identify flows. Each SA holds a security parameters index (SPI), which is a 32-bit value used by a receiver to identify the correct SA.
If each file should be encrypted with a different key it is necessary to specify distinct SAs that each use a unique SPI. The SPI is transmitted in the 6LoWPAN header in compressed form (See Section III-A). However, compression is only possible when the default SPI value is used; otherwise SPI information must be carried within the packet. Thus, the most frequently used file should use the default SPI in order to improve efficiency. In a practical setting this is not an issue as most nodes are using a single large file for storage of sensor data.

D. Implementation

We implemented the outlined framework for the Contiki [6] operating system. The implementation uses Contiki's μIP stack with 6LoWPAN/IPsec extensions as defined in [3] as the communication component. The storage component uses Contiki's Coffee filesystem (CFS) [7] with Codo [5] to provide filesystem security extensions. The μIP stack was modified in order to enable direct passing of ESP encrypted packets from the filesystem to the communication stack. On the host side we used a standard Ubuntu Linux host.

For encryption/decryption we used AES in counter mode (CTR), with a $128bit$ key, in either hardware (e.g. via the CC2420 radio chip present on many sensor node platforms) or the MIRACL [19] library if hardware support is not available. If authentication is required, AES-XCBC-MAC-96 is used to calculate the necessary ICV (Provided via cryptographic processor or the MIRACL library).
The maximum 802.15.4 payload is 127byte and the available MAC layer payload size is 102byte. As seen in Figure 2, 7byte are required for the compressed 6LoWPAN header, 12byte are required for the compressed ESP header fields, 2byte are required for the ESP trailer fields, 12byte are required for the ICV if it is used and 8byte are required for the UDP header. This leaves a maximum payload of 61byte. The AES algorithm requires a minimum block size of 16byte. Thus, the maximum feasible amount of data that can be stored per block before fragmentation must occur is 54byte. Storage blocks contain 64byte of encrypted data (8byte encrypted elements of the UDP header, 2byte encrypted ESP trailer and 54byte payload). Other feasible payload sizes are 6, 22 and 38. To avoid padding an application should align write operations with these payload sizes.
At this point in time, our Contiki IPsec implementation does not support key exchange mechanisms such as the Internet Key Exchange (IKE) protocol. Keys are set manually before deployment. However, for most application scenarios this would not be an issue limiting the frameworks usability.

E. Security Discussions

In this section we briefly discuss the security of the combined storage and communication system. We consider key management, cryptographic algorithms, message encryption and message authentication. In particular, we determine if the combination of secure storage and secure communication provides weaker security than systems treating both subsystems individually.
Confidentiality - Communication is secured using IPsec's ESP procedures. The solution does not deviate from procedures defined in the IPsec framework. An attacker with access to the communication channel has access to the same information as an attacker on any other ESP secured communication. If we consider IPsec a secure solution the provided solution can be considered secure as well.
Our implementation uses AES in counter mode (CTR) with $128bit$ keys. The best known AES attack for this key length is four times better than exhaustive search [20], and does not adversely affect its security.
Integrity and Authentication - When authentication is required, the ICV is calculated and appended to the ESP. Here we have to balance security and performance needs. Storing the ICV along with the ESP will ensure data integrity and authentication for storage and communication. However, when storing ICVs along with the encrypted payload it is necessary to select sequence numbers at the time data is stored. Hence, sequence numbers are predictable and will repeat when stored file content is transmitted repeatedly. Thus, protection against replay attacks in the communication channel is weakened. On the other hand, we will have performance gains as the ICV does not have to be computed at transmission time. If we decide to calculate the ICV before each transmission the replay protection is strongly enforced while the performance gains are reduced and stored data is missing integrity and authentication data.
To provide both, strong data integrity and relatively weaker anti-replay, functionalies when the ESP authentication field (ICV) is also stored in flash memory, sequence numbers should be in order before calculating ICVs for all the stored packets in a file, and the sender's and receiver's counter should be reset (by establishing a new SA) prior to the transmission of a file.
Storage - Data is stored in the same format as it is later transmitted. An attacker with access to the file system has the same information available as an attacker with access to the communication channel. If transmitted information secured using ESP is considered to be secure then information stored in the file system must be considered secure as well.
Key Management - Data in flash memory is secured using the same key that is later used for communication. Hence, transmission of the same stored data requires usage of the same key on the communication link. It is not possible to negotiate a fresh key for each communication relationship compared to when IPsec is used on its own. However, many practical IPsec deployments use pre-shared fixed keys so we consider this a secure option.
Similarly, if multiple nodes have to be able to access the same stored information they will also have to use the same key for communication. This is similar to practical situations where IPsec is used with a single pre-shared key.
The proposed system has difficulties with revocation of keys; if a new key is selected data already stored in the flash file system must be re-encrypted, which is costly on resource constrained systems.

IV. Evaluation

In this section we first discuss the costs in terms of storage overhead that are associated with the proposed scheme of combined storage and communication. Thereafter we analyse the processing performance and energy consumption gains associated with our scheme. We use our Contiki implementation for the Telos B platform.

A. Storage Overheads

Storing encrypted data together with the ESP fields that demand cryptographic processing requires additional storage space compared to a solution which would only store encrypted data.
If only encryption is used an extra $10byte$ per stored data block is required. Thus, it is better to store large blocks (large ESP packets) as this reduces the overhead. Figure 3 shows the overhead in dependency of the payload size. We show overheads for payload sizes which align with the AES

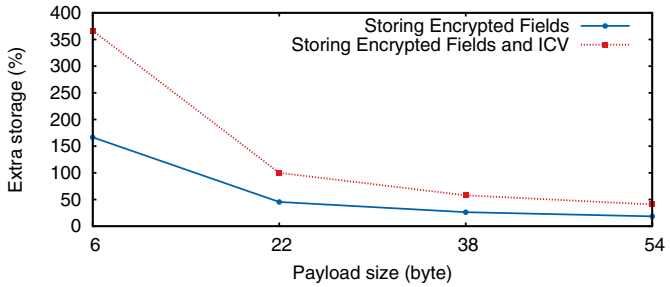


Figure 3: Storage overheads for different payload sizes.

encryption block size of $16byte$ and that do not require fragmentation when transmitted.

If authentication is also required then overheads increase as the ICV data of $12byte$ has to be stored alongside the other encrypted information.

The results show that the proposed framework reduces the effective storage size of the available flash storage space on nodes by 40.7% when using a payload size of $54byte$ and use of ICV. However, if we consider a common flash memory size of $16GB$ in which an $10byte$ sensor reading is recorded every minute the time until the storage capacity is exceeded is reduced from $3266years$ to $1329years$. Both values are acceptable in any deployment context and it can be concluded that the necessary storage overhead is not a limiting factor of the proposed framework.

B. Performance Gains

The combined storage and communication framework provides performance improvements. To analyse performance benefits in detail we use 4 different experiments. In all 4 experiments, ESP encryption and authentication is provided. The different experiments are used to show increasing performance benefits with increasing integration of storage and communication. *Experiment A* uses a system without the combining storage and communication. In these experiments (*Experiment B, C, D*) the framework is used in different configurations.

Experiment A is the baseline experiment where data is read from flash memory, decrypted and re-encrypted for IPsec conform transmission. In addition, authentication is provided and an ESP ICV is constructed. In *Experiment B*, ESP encrypted fields are stored in the flash memory and can be directly transmitted upon request. The ICV for authentication is still constructed at transmission time. *Experiment C* differs from *Experiment B* in terms of UDP checksum calculation. A non-matching IP address was used at storage time and a re-calculated before transmission is necessary. In *Experiment D*, ESP encrypted fields and the ESP authentication field (ICV) are stored in flash memory and can be transmitted directly upon request. All of the experiments are carried out with both software and hardware encryption. Table I summarises the experiment settings.

	Encryption	Authentication	UDP checksum re-calculation
Experiment A	individual	individual	-
Experiment B	combined	individual	not required
Experiment C	combined	individual	required
Experiment D	combined	combined	not required

Table I: Experiment setup details used for evaluation. All experiments use ESP encryption and authentication. The combined storage and communication framework is used for different aspects. UDP checksum re-calculation is assumed in some settings.

Experiment A: Baseline experiment

In this experiment, data is read from a file and sent using conventional methods. ESP conform AES encryption is used for the storage component and communication component to allow for comparison with the other experiments. Payload data is read in blocks of 6, 22, 38 and $54bytes$. The payload is decrypted and then re-encrypted for IPsec transmission. ICV authentication data is constructed before transmission. Decryption is carried out within the Contiki CFS; encryption and ICV calculation is carried out within the Contiki μIP stack. Figure 4 shows the time that is necessary on a node to process one payload. The processing time is measured from the start of the file system read operation to the completion of the packet transmission. The total processing time is broken down to show the contribution of significant individual operations:

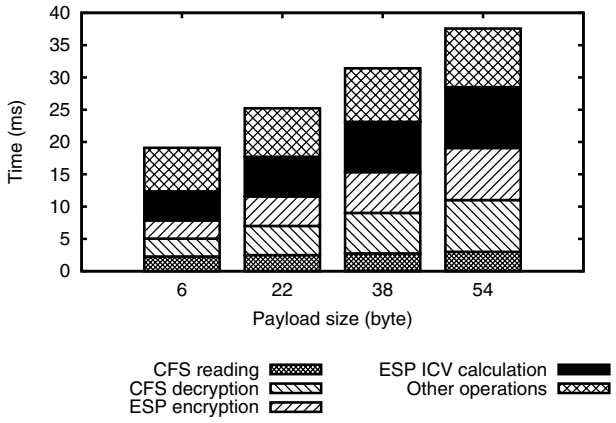
- *CFS reading* is the time required to read data from the file system.
- *CFS decryption* represents the time necessary to perform data decryption.
- *ESP encryption* represents the time necessary to encrypt the ESP payload.
- *ESP ICV calculation* is the time required to produce authentication data.
- *Other operations* summarises the duration of all other operations.

Figure 4a shows the processing duration breakdown when cryptographic processing is carried out in software. The total time to prepare a single packet is $19.1ms$, $25.2ms$, $31.4ms$ and $37.6ms$ for $6byte$, $22byte$, $38byte$ and $54byte$ payload data, respectively. CFS reading time is 8%, CFS decryption time is 21.3%, ESP encryption time is 21.5% and ESP ICV calculation time is 25.1% of the overall processing time for $54byte$ payload.

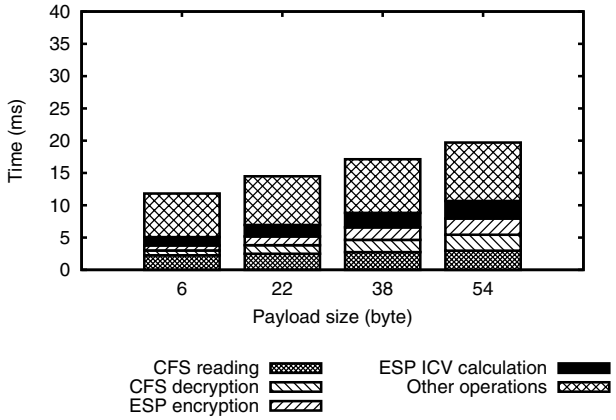
Figure 4b shows the duration of operations when using hardware supported cryptographic processing. Total times for preparing a single packet are $11.8ms$, $14.5ms$, $17.1ms$ and $19.7ms$ for the different payload sizes. CFS reading time is 15.1%, CFS decryption time is 12.5%, ESP encryption time is 12.7% and ESP ICV calculation time is 13.8% of the overall time when preparing $54byte$ of data.

Enabling hardware support improves performance by 38.1%, 42.6%, 45.5% and 47.5% for $6byte$, $22byte$, $38byte$ and $54byte$ payloads, respectively.

The experiments show that when a $54byte$ payload is transmitted processing the node spends 67.9% of the preparation time on cryptographic processing (software supported). This

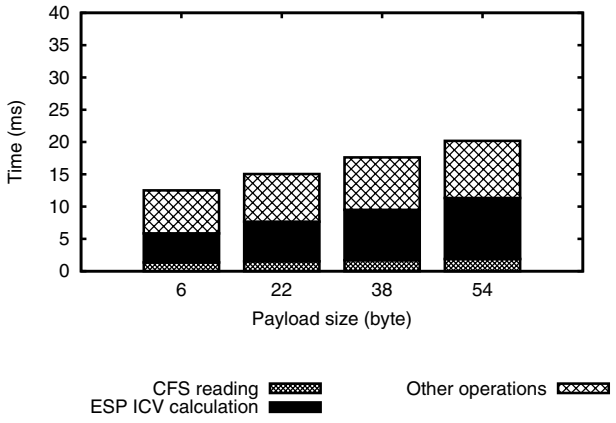


(a) With software encryption.

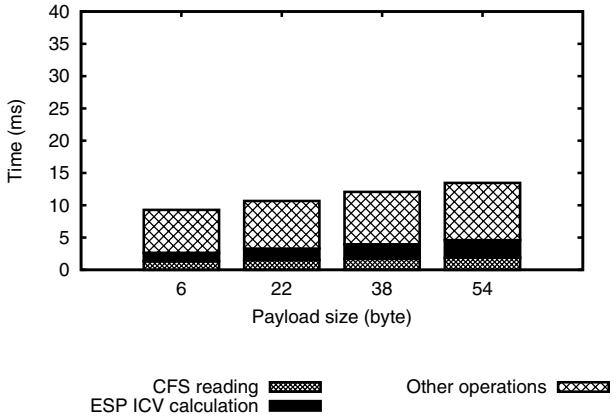


(b) With hardware encryption.

Figure 4: Duration of different operations involved in preparing single packet for transmission with software and hardware encryption.



(a) With software encryption.



(b) With hardware encryption.

Figure 5: Duration of different operations involved in preparing single packet for transmission with software and hardware encryption when storing ESP encrypted fields.

cryptographic processing time can be avoided by the proposed framework as we show in the following experiments.

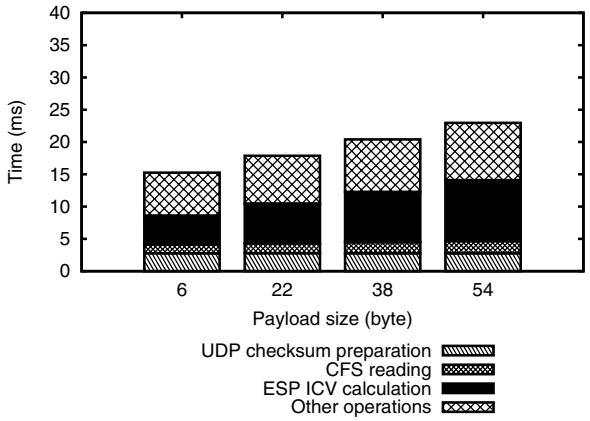
Experiment B: Storing ESP fields

In this experiment, ESP encrypted fields are stored in flash memory with the payload data. 6, 22, 38 and $54byte$ payloads are used. As additional stored ESP fields are $10byte$ length $16byte$, $32byte$, $48byte$ and $64byte$ must be read from the file system. Compared to *Experiment A* CFS decryption and ESP encryption is now not necessary, so processing time is saved. Figure 5a shows the duration for different operations when preparing a single packet for transmission with software encryption. Total times for preparing a single packet are $12.5ms$, $15ms$, $17.6ms$ and $20.2ms$; and improvements in system performance when it is compared to the baseline experiment with software encryption are 34.6%, 40.4%, 43.9% and 46.3%. Figure 5b shows processing times when using hardware encryption. Total times for preparing a single packet are $9.3ms$, $10.7ms$, $12.1ms$ and $13.5ms$; and improvements in system performance when it is compared to the baseline experiment with software encryption are 51.3%, 57.7%, 61.6% and 64.1%. It is notable that the CFS read time is less than in *Experiment A* even though more data has to be read from the file system (e.g. instead of $54byte$, $64byte$ are read as ESP information is included). This is due to the fact that elements of the CFS can be bypassed when directly reading encrypted data for transmission.

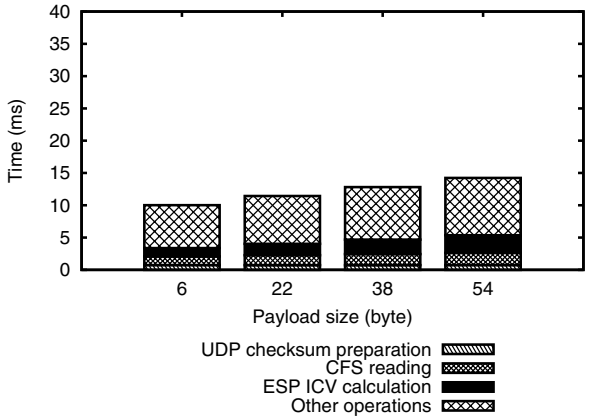
Experiment C: Using a non-matching IP address

This experiment is similar to *Experiment B*. The difference is the IP address of the destination when carrying out ESP encryption for storage. The UDP checksum enclosed in ESP packets must be corrected before transmission. The time necessary to perform decryption of the $16byte$ block containing the checksum, its correction and encryption of the $16byte$ block containing the corrected checksum is referred to as *UDP checksum preparation*.

Figure 6a shows results using software encryption. The total time for preparing a single packet is $15.3ms$, $17.9ms$, $20.4ms$ and $23ms$; and improvements in system performance when it is compared to the baseline experiment with software encryption are 20.1%, 29.1%, 35.1% and 38.9% for the different payload sizes.

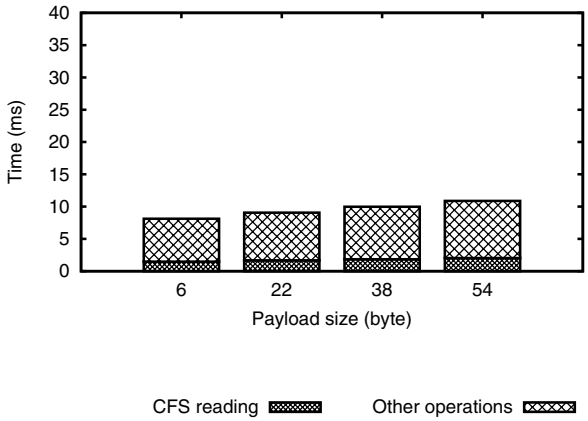


(a) With software encryption.

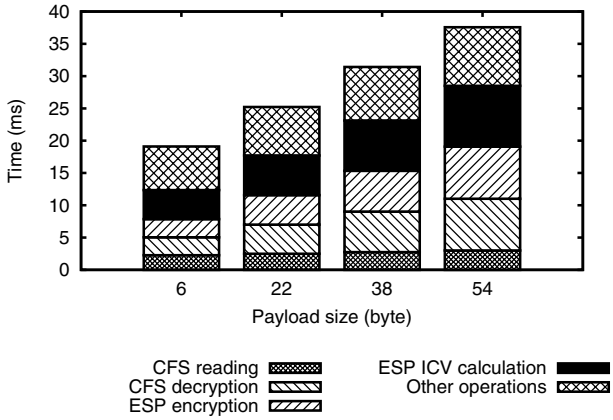


(b) With hardware encryption.

Figure 6: Duration of different operations involved in preparing single packet for transmission with software and hardware encryption when using a non-matching IP address.



(a) Combined storage and communication



(b) Individual security processing

Figure 7: Duration of different operations necessary to prepare a single packet for transmission when using the combined storage and communication framework and when using individual storage and communication security solutions.

Figure 6b shows results when using cryptographic hardware support. Total times are $10ms$, $11.4ms$, $12.8ms$ and $14.2ms$; and improvements in system performance when it is compared to the baseline experiment with software encryption are 47.6%, 54.7%, 59.2% and 62.1% for 6, 22, 38 and $54byte$ data.

The correction of the UDP checksum, which may be necessary in cases we cannot anticipate the endpoint to which stored data must be delivered, is not very costly. For a $54byte$ payload using hardware support the performance gain is only reduced from 64.1% to 62.1%.

Experiment D: Storing ESP and ICV fields

In this experiment all options of the proposed framework are in use. Data is stored in ESP compatible form alongside the ICV authentication data. In this case no encryption processing is required when data is requested, and thus processing times are independent from cryptographic algorithm implementations (hardware or software). The results are shown in Figure 7a. For direct comparison we again show the results of *Experiment A* (software encryption) in Figure 7b.

In this experiment, ESP encrypted fields and ESP authentication field (ICV) are stored in flash memory together with the payload data. As encrypted fields have a length of $10bytes$ and the authentication field (ICV) is $12bytes$ long, blocks of $28byte$, $44byte$, $60byte$ and $76byte$ have to be read for different payload sizes.

Compared to the previous two experiments, CFS read times increase as the additional ICV has to be read. Total time for preparing a single packet is $8.1ms$, $9.1ms$, $10ms$ and $10.9ms$. Improvements in system performance when compared with the baseline experiment with software encryption are 57.5%, 64.1%, 68.3% and 71% for 6, 22, 38 and $54byte$ payload data. These results show that the proposed framework of combined storage and communication can achieve significant performance gains. When the framework is used, requested data can be delivered approximately 3 times faster as cryptographic processing is not required at the time when data is prepared for delivery.

C. Energy Consumption

We have shown that combining secure storage and communication reduces processing time on sensor nodes. However, it is not immediatly clear if savings in processing time translate

to energy savings as the proposed mechanism changes usage patterns of hardware components such as flash memory and hardware encryption.

We therefore compare energy consumption of the conventional storage method with our combined storage and communication method. We use the setups previously described as *Experiment A* and *Experiment B*. We use energy consumption values for CC2420 radio operations and ST M25P80 flash operations from the Tmote Sky datasheet [21], and for CC2420 hardware cryptographic support from [22].

If $54byte$ data is required to be stored and transmitted later using the conventional method, $54byte$ data has to be encrypted and written to the flash memory for storage; $54byte$ data has to be read from the flash memory and has to be decrypted; $64byte$ data has to be encrypted in IPsec; $80byte$ data has to be authenticated in IPsec and the packet has to be transmitted, respectively. In case that $54byte$ of data is required to be stored and transmitted using combined storage and communication method, $64byte$ data has to be encrypted and written to flash memory for storage; $64byte$ data has to be read from the flash memory; $80byte$ of data has to be authenticated and the packet has to be transmitted.

The system is able to skip two cryptographic operations when using combined secure storage and communication. Therefore, the energy consumption decreases by 32.1%, even when additional $10byte$ have to be written to and read from flash memory. Due to space restrictions we do not detail energy savings for all other experiment combinations as discussed in the previous section. However, in all cases our proposed method leads to energy savings. In the worst-case, energy consumption decreases by 18.7% (in *Experiment D* with $6byte$ data size).

V. Conclusion

We have shown that combined secure storage and communication can reduce security related real-time processing on nodes dramatically (up to 71% reduction). As shown, this can be achieved while decreasing as well a nodes power consumption (up to 32.1%). Furthermore, we have shown that this is possible within the context of the IP protocol family which we believe will be used in the future IoT. The described solution requires additional storage space on nodes. However, we believe that currently available flash memory sizes can absorb these overheads.

Data on nodes must be secured when stored *and* transported in order to implement a comprehensive security solution. As resource-constrained embedded systems are limited in resources it is necessary to find efficient solutions. As shown, the proposed framework combining security aspects of storage and communication can help to achieve this goal.

VI. Acknowledgements

This work has been partially supported by SSF.

References

[1] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks," RFC 4944, Internet Engineering Task Force, 2007.

[2] S. Kent and K. Seo, "Security Architecture for the Internet Protocol," RFC 4301, Internet Engineering Task Force, 2005.

[3] S. Raza, S. Duquennoy, J. Höglund, U. Roedig, and T. Voigt, "Secure communication for the internet of things - a comparison of link-layer security and ipsec for 6lowpan," *Security and Communication Networks*, 2012. [Online]. Available: http://dx.doi.org/10.1002/sec.406

[4] N. Tsiftes and A. Dunkels, "A database in every sensor," in *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, 2011, pp. 316–332.

[5] I. E. Bagci, M. R. Pourmirza, S. Raza, U. Roedig, and T. Voigt, "Codo: Confidential data storage for wireless sensor networks," in *8th IEEE International Workshop on Wireless and Sensor Networks Security (WSNS 2012)*, October 2012.

[6] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, 2004, pp. 455–462.

[7] N. Tsiftes, A. Dunkels, H. Zhitao, and T. Voigt, "Enabling large-scale storage in sensor networks with the coffee file system," in *Proceedings of the 2009 International Conference on Information Processing in Sensor Networks*. IEEE Computer Society, 2009, pp. 349–360.

[8] P. Szczechowiak, L. B. Oliveira, M. Scott, M. Collier, and R. Dahab, "Nanoecc: testing the limits of elliptic curve cryptography in sensor networks," in *Proceedings of the 5th European conference on Wireless sensor networks*, 2008, pp. 305–320.

[9] W. Hu, P. Corke, W. C. Shih, and L. Overs, "secfleck: A public key technology platform for wireless sensor networks," in *Proceedings of the 6th European Conference on Wireless Sensor Networks*. Springer-Verlag, 2009, pp. 296–311.

[10] D. Liu and P. Ning, "Establishing pairwise keys in distributed sensor networks," in *Proceedings of the 10th ACM conference on Computer and communications security*, ser. CCS '03. ACM, 2003, pp. 52–61.

[11] N. Bhatnagar and E. L. Miller, "Designing a secure reliable file system for sensor networks," in *Proceedings of the 2007 ACM workshop on Storage security and survivability*, 2007, pp. 19–24.

[12] IEEE std. 802.15.4 - 2003, "Wireless medium access control (mac) and physical layer (phy) specifications for low rate wireless personal area networks (lr-wpans)." *IEEE*, 2003.

[13] ArchRock Corporation, "Phynet n4x series," 2008.

[14] S. Hong, D. Kim, M. Ha, S. Bae, S. J. Park, W. Jung, and J.-E. Kim, "Snail: an ip-based wireless sensor network approach to the internet of things," *Wireless Communications, IEEE*, vol. 17, no. 6, pp. 34 –42, december 2010.

[15] S. Fouladgar, B. Mainaud, K. Masmoudi, and H. Afifi, "Tiny 3-tls: a trust delegation protocol for wireless sensor networks," in *Proceedings of the Third European conference on Security and Privacy in Ad-Hoc and Sensor Networks*, 2006, pp. 32–42.

[16] J. Girao, D. Westhoff, E. Mykletun, and T. Araki, "Tinypeds: Tiny persistent encrypted data storage in asynchronous wireless sensor networks," *Ad Hoc Netw.*, vol. 5, pp. 1073–1089, September 2007.

[17] W. Ren, Y. Ren, and H. Zhang, "Hybrids: A scheme for secure distributed data storage in wsns," in *Proceedings of the 2008 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing - Volume 02*, 2008, pp. 318–323.

[18] S. Kent, "IP Encapsulating Security Payload (ESP)," RFC 4303, Internet Engineering Task Force, 2005.

[19] CertiVox, "MIRACL - Multiprecision Integer and Rational Arithmetic C/C++ Library." [Online]. Available: http://certivox.jira.com/wiki/display/MIRACLPUBLIC/Home

[20] A. Bogdanov, D. Khovratovich, and C. Rechberger, "Biclique cryptanalysis of the full aes," in *Proceedings of the 17th international conference on The Theory and Application of Cryptology and Information Security*, 2011, pp. 344–371.

[21] "Tmote Sky datasheet," 2006, http://www.eecs.harvard.edu/~konrad/projects/shimmer/references/tmote-sky-datasheet.pdf.

[22] F. Zhang, R. Dojen, and T. Coffey, "Comparative performance and energy consumption analysis of different aes implementations on a wireless sensor network node," *International Journal of Sensor Networks*, vol. 10, no. 4, pp. 192–201, 2011.