µDTNSec: A security layer with lightweight certificates for Disruption-Tolerant Networks on microcontrollers

Dominik Schürmann · Georg von Zengen · Marvin Priedigkeit · Sebastian Willenborg · Lars Wolf

Preprint submitted to Annals of Telecommunications on October 2017, revised March 2018 and accepted June 2018

Abstract In Delay/Disruption-Tolerant Networks, Man-in-the-Middle attacks are easy: Due to the Store-Carry-Forward principle, an attacker can simply place itself on the route between source and destination to eavesdrop or alter bundles. This weakness is aggravated in networks where devices are energy-constrained but the attacker is not.

To protect against these attacks, we design and implement µDTNSec, a security layer for Delay/Disruption-Tolerant Networks on microcontrollers. Our design establishes a Public Key Infrastructure with lightweight certificates as an extension to the Bundle Protocol. It has been fully implemented as an addition to µDTN on Contiki OS and uses Elliptic Curve Cryptography and hardware-backed symmetric encryption. In this enhanced version of µDTNSec, public key-identity bindings are validated by exchanging certificates using Neighbor Discovery. µDTNSec provides a signature mode for authenticity and a Sign-then-Encrypt mode for added confidentiality. Our performance evaluation shows that the choice of the curve dominates the influence of the payload size. We also provide energy measurements for all operations to show the feasibility of our security layer on energy-constrained devices. Because a high quality source of randomness is required, we evaluated the Random Number Generators by the AT86RF231 radio, its successor AT86RF233, and one based on the noise of the A/D converter. We found that only AT86RF233 provides the required quality.

This paper is an extended version of "D. Schürmann, G. von Zengen, M. Priedigkeit, L. Wolf: µDTNSec: A Security Layer for Disruption-Tolerant Networks on Microcontrollers" [1].

D. Schürmann · G. von Zengen · M. Priedigkeit · S. Willenborg · L. Wolf
Institute of Operating Systems and Computer Networks, TU Braunschweig, Germany

Keywords Disruption-Tolerant Networking · DTN · Microcontroller · Security · PKI

1 Introduction

In recent years Internet of Things (IoT) technology emerged into more and more fields where sensitive data is handled. Namely industrial applications like process monitoring where production data needs to be protected from being accessed unauthorized. Another situation is in the field of intelligent transportation systems, where control data must not be manipulated by an attacker [2]. The most challenging fields are healthcare applications. IoT technology has to face several challenges at once: The first challenge is the mobility of nodes. Devices are mounted to patients, therefore they have to cope with mobility and thus disruption of connections. To ensure a curtain level of reliability in terms of data delivery, these networks have to cope with the mobility of their nodes. Also, the security layer of such a network needs to be tolerant against network disruption. Considering frequent connection losses and short connection periods, key exchange mechanisms like Diffie Hellman might not be able to exchange a key prior every connection. The second challenge is the variety of communication partners and their different access rights to the data. For example, an X-ray unit might only need an identifier of the patient but a doctor needs the full history of the patient.

To keep the costs of these systems low, direct connections as well as multihop connections are needed. Introducing multihop and different access rights in one network makes traditional IoT security concepts not applicable. For instance in a network using IEEE 802.15.4, a key can be specified that is used by the MAC sublayer

for full link layer AES encryption. Thus, every device in the network needs to have the same key to be able to read routing information inside the packets. This means, every device is able to read all data it forwards. Another problem with this encryption is that if one device is hijacked and an attacker was able to read out its key, the attacker is able to read all data transferred in the whole network. This is a severe issue when taking into account that the devices will be handed out to patients for several days or weeks.

In this paper we present a system that overcomes these challenges. To overcome the challenge of disruptions in the connections, we propose to use µDTN [3], a lightweight implementation of the bundle protocol [4]. By utilizing public key cryptography we solve the security part of the disruption challenge. Another advantage of public key cryptography is the ability to ensure authentication. The most profound advantage of public key cryptography is the robustness against Man-in-the-Middle attacks. By using a lightweight design of the bundle security protocol we enable IoT devices to use all these advantages.

This paper presents an extended version of µDTNSec. The main addition to the original version [1] is the introduction of certificates. Prior to this extension, µDTNSec required that each node's public key must be pre-deployed on all other nodes in the network. Thus, to add a new node, all existing nodes must be re-programmed. In this extended version, we allow dynamic addition of new nodes by introducing a method to exchange certificates by piggybacking them on DTN IP Neighbor Discovery (IPND) [5] packets. Now, the certificates defines the affiliation to the network. Thus re-programmming is no longer needed and the nodes exchange keys automatically via IPND. The performance and energy overhead by certificates is evaluated in Section 4.2. For the security of µDTNSec's session keys, initialization vectors and nonces a cryptographically secure Random Number Generator (RNG) is required. This is especially difficult on microcontrollers due to their limited sources of randomness. Thus, we evaluated the different RNGs available on our sensor nodes to rate their quality in Section 4.4.

2 Related Work

A possible protocol for Delay/Disruption-Tolerant Networks (DTNs) has been standardized as the Bundle Protocol (BP) in RFC 5050 [4]. The BP is designed for establishing a Store-Carry-Forward overlay network on top of convergence layers, such as TCP/UDP, or directly on top of the data link layer. It defines addresses with Endpoint Identifiers (EIDs), overlay routing algorithms, and a data unit format called bundles. Each bundle consists of several blocks, such as the payload block. The protocol can be extended by defining new block types.

RFC 6257 [6] specifies an extension with security block types, security processing rules, and ciphersuites. Ciphersuites are based upon the cryptographic primitives of RSA with SHA-256 for signatures, Keyed-Hash Message Authentication Code (HMAC) for data authenticity, RSA for encrypting symmetric session keys, and Advanced Encryption Standard (AES) in Galois/Counter Mode (GCM) for authenticated data encryption. While most desktop implementations, such as IBR-DTN [7] and DTN2 [8], support this extension, implementations for embedded devices lack security features. There are currently three actively maintained implementations of DTN for microcontrollers:

µDTN[1] [3] is our implementation of the BP. It was implemented to meet the hardware and energy constrains of wireless sensor nodes. To meet these we use the compressed bundle header encoding [9]. Originally, µDTN was developed for 8-Bit micro controllers running Contiki OS[2]. By transmitting bundles directly in IEEE 802.15.4 frames we reduced the overhead to a minimum, as shown by Pöttner et al. [10]. To the best of our knowledge it was the first RFC 5050-compatible DTN implementation for wireless sensor nodes. Recently, it was ported to run with FreeRTOS[3] on STM32F4 microcontrollers under the name miniDTN [11]. Both, µDTN and miniDTN, share the same architecture and most of the code. Therefore, in the remainder we handle them as one.

µPCN[4] [12] is a microcontroller-compatible BP implementation that focuses on CubeSat applications. Therefore, the authors consider a routing scenario with a single hop from the satellite to a ground station and multiple nodes behind the ground station. In satellite applications, not every node is reachable from all ground stations and malicious ground station might announce neighbours they can not reach. To tackle these special challenges, µPCN's decision whether to transmit a bundle to the ground station is based on two factors: the reliability and the trustworthiness. Reliability is the number of successful contacts minus the number of unsuccessful ones. Trustworthiness is the probability that a transmitted bundle reaches its destination. With these factors µPCN implements routing security but other security and authentication aspects are left to future work.

[1] https://www.ibr.cs.tu-bs.de/projects/mudtn/
[2] http://contiki-os.org
[3] http://freertos.org
[4] https://upcn.eu

DTNLite [13] is an implementation of the concepts of DTN for TinyOS[5]. Instead of implementing the BP, it relies on existing routing protocols and therefore forms an overlay network. The typical DTN Store-Carry-Forward mechanism is implemented by storing data in non-volatile memory on nodes and transporting it towards a single sink over multiple hops. The authors do not consider encryption or authentication in their work.

While these DTNs do not provide confidentiality and authenticity, security on microcontrollers has been studied in other contexts. In general, the energy limitations on microcontollers lead to difficult design constraints as discussed by Trappe et al. [14]. Because asymmetric RSA operations do not scale linearly with their key size [15] and secure keys must be at least 3072 bit [16], RSA is not practical for embedded systems. Thus, for resource constrained networks, encryption is often implemented by pre-deploying symmetric keys. To automate this process, plenty of key pre-distribution protocols have been designed. However, symmetric key distribution does not scale for huge networks as envisioned for the IoT [17]. Thus, more performant asymmetric operations, such as Elliptic Curve Cryptography (ECC), have been evaluated for ATmega128 and CC1010 microcontrollers by Gura et al. [15]. With NanoECC [18], a fast implementation for TinyOS has been written in C/nesC with assembly optimizations.

Another approach has been developed by Oliveira et al. [19,20]. Their security layer is based on Identity Based Cryptography (IBC) to encrypt messages by deriving public keys from the nodes' IDs. Their last implementation TinyPBC is based on cryptographic pairings [20]. For this, the authors developed a cryptographic library for microcontrollers called RELIC [21]. It has been shown that RELIC provides the fastest operations but with higher RAM usage in comparison to TinyECC and Wiselib [22]. Later, several improvements have been made to RELIC, e.g., performance improvements on ARM Cortex-M0+ [23]. Due to the costs of cryptographic co-processors, asymmetric operations are still implemented in software. In contrast, most modern radio chips support symmetric algorithms, such as the AES in hardware. In this paper, we leverage the low-level operations of AES provided by the AT86RF23X chip [24]. Its high-level functionality for Contiki OS has been implemented previously during the design of RAIM [25].

Besides cryptographic primitives, full security layers have been proposed and implemented for TinyOS and Contiki. TinySec [26] provides modes for authentication-only (TinySec-Auth) and fully authenticated encryption (TinySec-AE). Here, the Skipjack cipher is used in Cipher Block Chaining (CBC) mode. TinyKey [27] provides key management for TinySec, but introduces no algorithm changes. A successor of TinySec called MiniSec [28] still uses Skipjack, but in Offset Codebook (OCB) mode. All these use Skipjack instead of a more widely used and standardized block cipher, such as AES. Asymmetric ECC operations have been introduced by Liu et al. for key exchange and digital signatures [29].

For Contiki, ContikiSec [30] has been designed and implemented. It provides modes for encryption-only (ContikiSec-Enc), authentication-only with CMAC (ContikiSec-Auth), and a full authenticated encryption mode by using AES in OCB mode (ContikiSec-AE). All modes require a single network-wide key that is pre-deployed on all devices. Optionally, by using other key exchange methods, a session key can be plugged in. ContikiSec uses standardized and well researched cryptographic primitives.

More recently, Datagram Transport Layer Security (DTLS) gained a lot popularity in the Wireless Sensor Networks (WSN) and IoT communities. An optimized DTLS implementation for the IoT that implements an optimized ECC key exchange is provided by Capossele et al. [31]. To better adapt DTLS to constrained hardware, Pittoli et al. propose to reduce the payload overhead and number of handshake messages [32]. A full mobility enabled system for healthcare applications is presented by Moosavi et al. using DTLS for end-to-end security [33]. Their system consists of three layers: the sensor node layer, the *smart gateway* layer that interconnects the sensor node layer with the third layer, the cloud layer where data processing is done. By having all *smart gateways* connected to each other, DTLS is capable of handling the mobility of the sensor nodes. While DTLS can be appropriately applied in their work, in DTNs, there is no permanent connection between gateways.

To the best of our knowledge, there is no implementation of a DTN for microcontrollers that provides end-to-end security. In contrast to existing work, we provide a fully integrated solution for resource-constrained DTNs with software-based ECC using the NanoECC [18] and AVR-Crypto-Lib [34] libraries and hardware-based AES using the AT86RF23X radio.

3 µDTNSec

µDTNSec provides end-to-end security between source and destination. It has been designed as a security layer for µDTN and provides two main modes of operation: Signature-only mode for authenticity and Sign-then-Encrypt mode for combined authenticity and confidentiality. A short reference specification is given in Table 1.

[5] https://github.com/tinyos

In the following, we provide a threat model and discuss µDTNSec's design and modes in detail.

3.1 Threat Model

Before discussing the design of µDTNSec, we first specify a clear threat model. This allows us to derive a set of security features.

Eavesdropping: The typical threat for all network communications is passive eavesdropping. In a wireless network without payload encryption an attacker can eavesdrop on the wireless channel and read transmitted data.

Man-in-the-Middle: In networks with peer-to-peer encryption, but no end-to-end encryption, an attacker could place herself into the route between source and destination to read bundles intended for the victim. In comparison to passive eavesdropping, a Man-in-the-Middle attack is active and requires a more sophisticated attacker with network knowledge.

Data Modification: In a variation of the Man-in-the-Middle attack, the attacker not just reads but modifies bundles in transit. Every node participating in Store-Carry-Forward routing can do this.

Impersonation: For this attack, the attacker impersonates the victim to receive its bundles or to transmit malicious bundles in the victim's name.

3.2 Security Features

Our threat model covers the most important threats posed by *outside attackers*. µDTNSec's full Sign-then-Encrypt mode provides confidentiality, node authenticity/non-reputability, and data integrity. These features protect against eavesdropping, Man-in-the-Middle attacks, impersonation, and data modification between both endpoints. We introduced certificates to manage network access and provide key authenticity.

Inside attackers, i.e., nodes that are already part of the network, are considered out of scope for µDTNSec. Currently, revocation of certificates is not implemented, but will be part of a future version. For the typical use cases of sensor networks, anonymity and metadata protection mechanisms are not relevant and are contrasting node authenticity. While other BP implementations, such as IBR-DTN, need to take extra care to secure the lower network layer, e.g., TCP or UDP, µDTN works directly above the data link layer. Thus, only Denial-of-Service jamming attacks against the physical layer must be considered additionally to µDTNSec.

Finally, attacks against the hardware, which require physical access to a placed node, are considered out of scope. This includes tampering with the hardware and side channel and fault attacks, such as key extraction by power measurements.

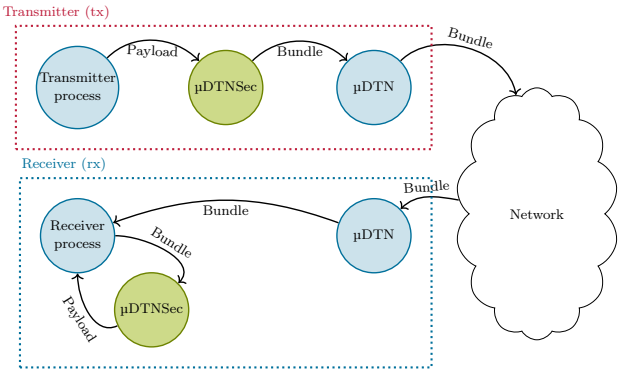


Fig. 1: Interaction between µDTN and µDTNSec

3.3 Trade-off Between Security and Performance

In this version of µDTNSec, certificates are used to allow dynamic addition of new nodes to the network, and providing an automatic key exchange mechanism over IPND. However, when the Sign-then-Encrypt mode is used, signatures are only verified by the final receiver after bundle decryption. This is a deliberate decision to have no computational overhead on forwarding nodes. The Signature-only mode or an additional Sign-Encrypt-Sign mode could be used for bundle verification on forwarding nodes to protect against routing attacks. Still, we advice against doing this, as the protection against these attacks does not justify the performance penalty on forwarding nodes. Conclusively, µDTNSec currently provides end-to-end security, while hop-by-hop security is considered out of scope.

3.4 System Design

µDTNSec has been developed for the INGA sensor node platform [36]. INGA is equipped with a AT86RF23X radio chip that already provides a hardware implementation of AES-128 in Electronic Code Book (ECB) and CBC mode. Interfaces to its AES methods and RNG have already been implemented for Contiki during the design of RAIM [25]. ECC is implemented in software using the NanoECC [18] and AVR-Crypto-Lib [34] libraries.

Previously, in the µDTN program flow, Contiki processes were able to call a µDTN function to transmit a payload together with the destination (receiver) IDs. µDTN encapsulated the given data into a bundle according to RFC 5050 for Store-Carry-Forward routing. We

Table 1: µDTNSec Specification: Modes with their corresponding BP block, BP ciphersuite and details

Mode	Block	Ciphersuite	Details
Signature	PIB	PIB-ECDSA-SHA256	Signatures are generated and verified with ECDSA and SHA-256. Public keys from validated certificates are used for verification.
Sign-then-Encrypt	PIB +PCB	PCB-ECDH-SHA256-AES128-PIB-ECDSA-SHA256	After signature generation, a shared secret is calculated by ECDH on valid public keys. A session key is derived using the ANSI-X9.63-KDF and used for hardware-backed AES-128 encryption in CBC mode. The payload is encrypted in-place while the PIB is encrypted separately.

Supported SECG [35] curves: secp128r1, secp192r1, secp256r1

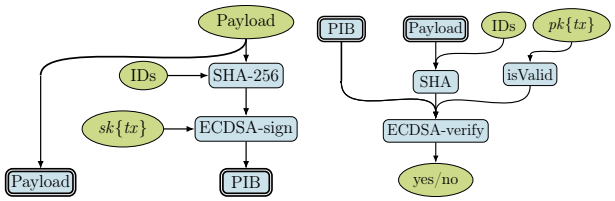
integrate µDTNSec in this flow as depicted in Figure 1. On the transmitting side, Contiki processes should now call µDTNSec's function instead of µDTN. This allows µDTNSec to apply the configured security protection before handing the encapsulated bundle to µDTN. At the final destination (receiver), µDTN processes the bundle and hands it to the Contiki receiver process. If the bundle includes security protection, the process calls µDTNSec for decryption/verification.

µDTNSec is based on ECC as specified by the Standards for Efficient Cryptography Group (SECG) [37, 35]. It includes the algorithms Elliptic Curve Digital Signature Algorithm (ECDSA) for signatures and Elliptic curve Diffie–Hellman (ECDH) for encryption. For each node, a secret/public key pair sk, pk is generated. Our implementation supports the secp128r1, secp192r1, secp256r1 curves for different security levels [35]. The implementation follows parts of RFC 6257 [6] recommendations, but does not provide full compatibility.

3.5 Signature Mode

µDTNSec's signature mode provides end-to-end authenticity/non-reputability, which automatically includes integrity protection. We implemented signature generation and verification based on ECDSA with SHA-256. Following the flow chart in Figure 2a, first a SHA-256 hash is calculated over the payload and the non-mutable fields of the bundle. Non-mutable fields are all fields that are not changed by nodes forwarding the bundle on the way to its final receiver. In µDTNSec, the non-mutable fields include the IDs of the transmitter, the transmitter process, the receiver, and the receiver process. This is designed similar to the *Mutable Canonicalization* in the Bundle Security Protocol RFC 6257 [6]. No timestamps are included by µDTN.

$$h = SHA{-}256(payload \parallel non{-}mutable{-}fields)$$



(a) Transmitter: Generation (b) Receiver: Verification

Fig. 2: Flow charts of µDTNSec's signature mode: Generation and verification of signatures encapsulated in the PIB

The resulting hash h is finally signed with ECDSA [38] using the secret key $sk\{tx\}$ to produce a signature.

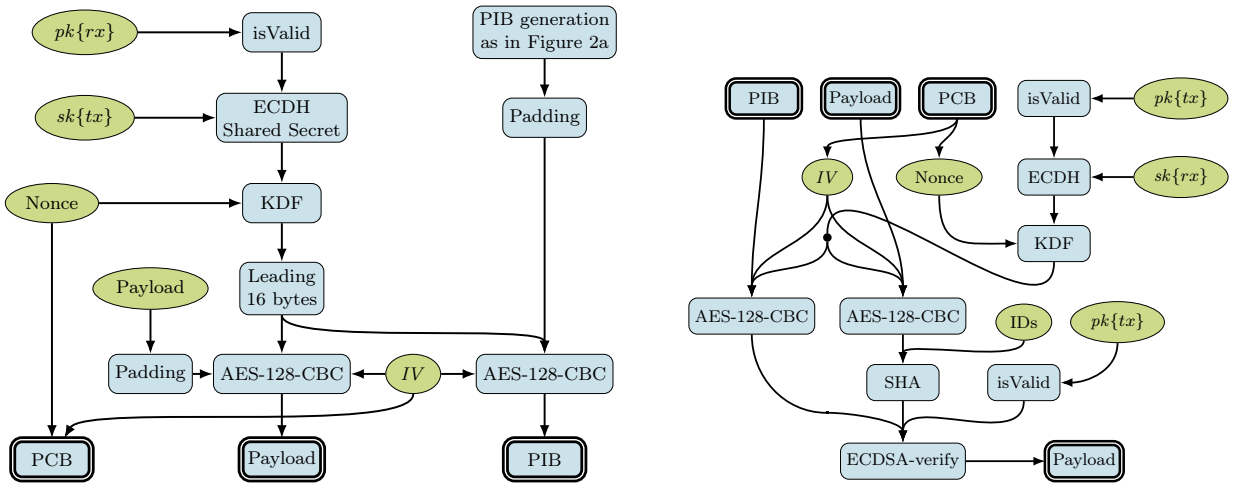
$$\sigma = ECDSA{-}sign(sk\{tx\}, h)$$

The Payload Integrity Block (PIB) consists of the default fields of a BP block (size in bytes), namely Type (1), Flags (1), Length (dynamic), and encapsulates the signature σ ($(192 \cdot 2)/8 = 48$ for secp192k1). The resulting bundle is handed back to µDTN. The verification of bundles including a PIB works analogously. Following Figure 2b, a hash is calculated over the payload and non-mutable fields. To protect against related key attacks, the public key of the transmitter $pk\{tx\}$ is validated first, i.e., it is checked if the key is based on the correct curve. For verifying σ, ECDSA verification is executed.

$$ECDSA{-}verify(pk\{tx\}, h, \sigma)$$

3.6 Sign-then-Encrypt Mode

µDTNSec's Sign-then-Encrypt mode provides end-to-end confidentiality in addition to the security features provided by its signature mode. Besides ECDSA signatures, this mode uses hardware-backed AES encryption, while its session key is asymmetrically secured with ECDH. Following the flow chart in Figure 3a, a



(a) Transmitter: Encryption and generation of signature

(b) Receiver: Decryption and verification of signature

Fig. 3: Flow charts of µDTNSec's Sign-then-Encrypt mode: AES encryption/decryption is done in-place inside the payload block/PIB. The PCB contains a nonce and the IV. Signatures are encapsulated in the encrypted PIB

shared secret is derived using ECDH. This requires the receiver's public key $pk\{rx\}$, which has been validated to be based on the correct curve, and the transmitter's secret key $sk\{tx\}$. Because this shared secret would be the same for each communication, an additional nonce, generated from a secure Random Number Generator, is taken into account. A unique session key is derived using the ANSI-X9.63-KDF [37].

$$s = KDF(ECDH(pk\{rx\}, sk\{tx\}), nonce))$$

A PIB is generated as described in µDTNSec's signature mode. The payload and PIB are symmetrically encrypted with the session key s using AES in CBC mode. It is important to note that symmetric encryption is done in-place without adding new blocks. Here, the low-level operations of AES are provided by the AT86RF23X chip [24]. The utilized CBC mode requires a public initialization vector IV, which should be unique for each bundle. The Payload Confidentiality Block (PCB) consists of the default fields of a BP block (size in bytes), namely: Type (1), Flags (1), Length (dynamic), and additionally contains the Nonce (1), IV (16), the encrypted payload (dynamic) and padding (0-15). The resulting bundle is handed back to µDTN.

After µDTNSec extracts the three blocks, the decryption and verification of bundles works as follows. Following Figure 3b, $pk\{tx\}$ is validated and used together with $sk\{rx\}$ to derive a shared secret. Using the public nonce, the same session key s is derived. Together with the IV, it is used to decrypt the payload and PIB. The signature encapsulated in PIB is verified as described before. Finally, the payload is only returned to the calling process if the verification succeeded.

3.7 Lightweight Certificates

We implemented lightweight certificates exchanged between nodes via IPND. This allows for key distribution, provides a closed network and most importantly key authenticity, i.e., a way to verify that a given public key corresponds to the alleged node. First, a secret/public key pair $sk\{ca\}, pk\{ca\}$ is generated for the Certificate Authority (CA) operating the network. All network nodes come with $pk\{ca\}$ pre-deployed. A certificate is issued before deployment by signing the nodes's EID, Certificate Flags (CertFlags), and public key $pk\{tx\}$:

$$\begin{aligned}\Sigma\{tx\} = &ECDSA\text{–}sign(sk\{ca\},\\ &SHA\text{–}256(EID \parallel CertFlags \parallel pk\{tx\}))\end{aligned}$$

$\Sigma\{tx\}$ is piggybacked on IPND and thus exchanged with encountered nodes. Receiving nodes can verify the key authenticity to this specific EID.

$$\begin{aligned}&ECDSA\text{–}verify(pk\{ca\},\\ &\quad SHA\text{–}256(EID \parallel CertFlags \parallel pk\{tx\}),\\ &\quad \Sigma\{tx\})\end{aligned}$$

It is extremely important to also validate that the certificate's EID matches the one announced by the node. Certificates are stored when receiving the IPND packet, but validated only when receiving the first bundle of the corresponding EID. Consecutive bundles from the same EID require no additional certificate validation. Besides the default fields of an IPND packet (size in bytes), namely Version (1), Flags (1), and SequenceNumber (2), we added EID (4), CertFlags (1), $pk\{tx\}$ $(24 \cdot 2)$ $\Sigma\{tx\}$ $(24 \cdot 2)$ for our certificate exchange. CertFlags could

potentially be used for revocation or other additional features in a future version of µDTNSec.

While the BP proposes the Bundle Authentication Block (BAB) for controlling network access [6], we argue that in our lightweight design and threat model this block is not required: A closed network has been created by issuing certificates to verified nodes only.

4 Evaluation

Our evaluation focuses on the energy and time overhead introduced by µDTNSec's security modes and its certificate exchange. In addition, a quality estimation of the RNGs by the AT86RF231 radio, its successor AT86RF233, and one based on the noise of the A/D converter is provided.

4.1 Energy and Performance of Security Modes

In our evaluation setup we connected two INGA sensor nodes to a precise energy measurement device named Potatoscope [39]. By utilizing this device we were able to trigger the measurement exactly at the time µDTNSec started processing a bundle and also to stop the measurement when µDTNSec finished.

In the first part of the evaluation, we performed a measurement just for the PIB with the secp192r1 curve, in the second part we compare the PIB+PCB processing with three curves namely: secp128r2, secp192r1, and secp256r1. By comparing the two parts we can point out the overhead introduced by the encryption or decryption. The ciphersuites used are the ones mentioned in Table 1. To investigate the influence the payload size, we evaluated six different byte sizes from 1 B to 250 B.

Figure 4 shows the mean time needed to verify the authenticity of a bundle (blue circles). As the values are neither linear rising nor falling and have a standard deviation higher than the span of the of the mean values, we can assume the influence of the payload size for the verification is negligible. The red triangles represent the time needed to process the PIB at the transmitter of the bundle. With up to 7.95 s the transmitter needs significantly less time than the receiver with up to 9.51 s. These results match with the ones by Jansma et al. [40]. Similar to the receiver, the payload does not have a significant influence.

The comparison between the time in Figure 4 and the consumed energy plotted in Figure 5 shows that both correlate almost perfectly. At the transmitter the energy ranges from 0.1258 mWh to 0.1265 mWh. As for the time, the receiver's effort in terms of energy is higher with 0.1484 mWh to 0.1521 mWh. Thus, the

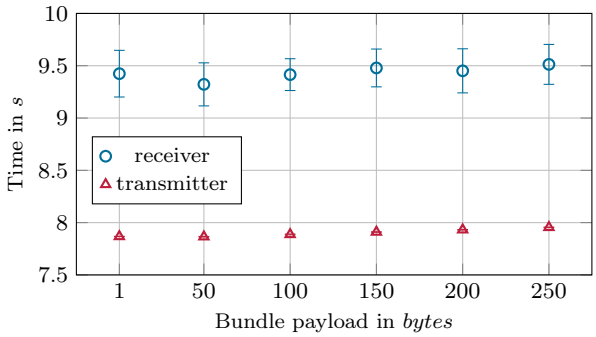


Fig. 4: Signature mode: The red triangles show the time need to generate a PIB with secp192r1 at the transmitter, the blue circles indicate the time used to verify its authenticity at the receiver

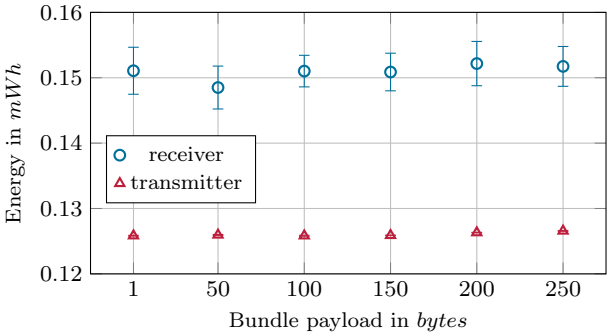
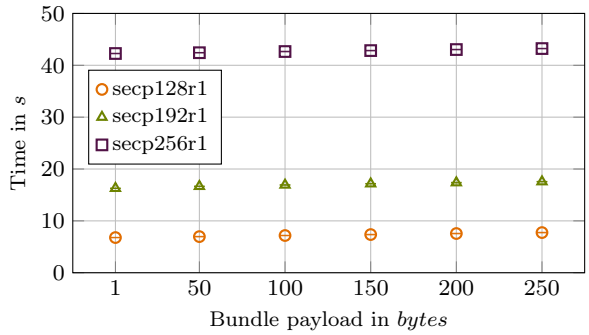


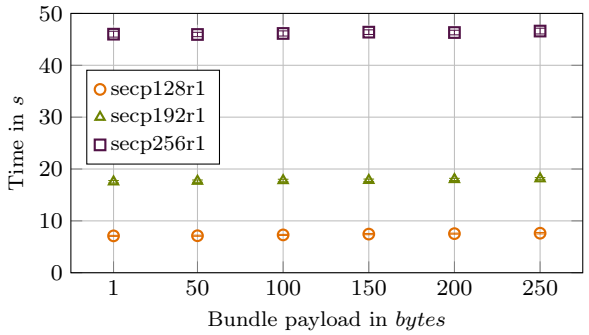
Fig. 5: Signature mode: The red triangles show the energy need to generate a PIB with secp192r1 at the transmitter, the blue circles indicate the energy used to verify its authenticity at the receiver

receiver needs approximately 20 % more energy than the transmitter to process the PIB.

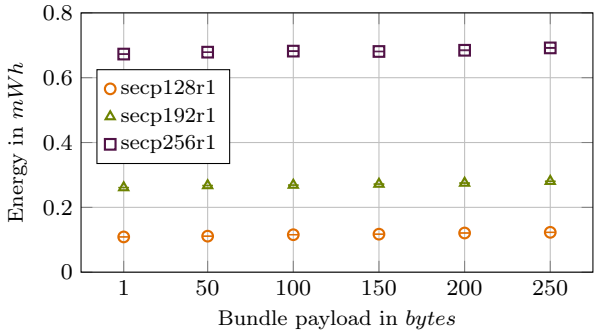
In comparison to the PIB, the processing of PIB+PCB doubles the amount of time and energy needed, as the green triangles in Figures 6a to 6d show. This is due to the additional encryption operation of the payload. Another observation from the figures is that the influence of the chosen curve clearly dominates the influence of the payload size. While the payload's influence is hardly noticeable, secp256r1 doubles the energy and time needed to process a bundle compared to secp192r1 and this doubles it from secp128r1. For secp128r1 the receiver needs up to 0.121 mWh to decrypt and verify the payload of Bundle. It lasts 7.621 s to perform this operations. For secp256r1 0.741 mWh and 46.6 s are needed. The values for the transmitter of a bundle are similar: 0.122 mWh and 7.742 s for secp128r2. Utilizing secp256r1 at the transmitter the consumption is raised to 0.691 mWh and 43.227 s.



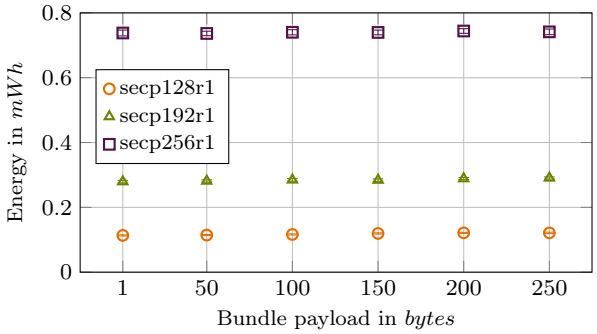
(a) Sign-then-Encrypt mode: Time needed to encrypt and sign a bundle's content at the transmitter for the three evaluated curves



(b) Sign-then-Encrypt mode: Time need to decrypt and verify a bundle's content at the receiver for the three evaluated curves



(c) Sign-then-Encrypt mode: Energy consumed to encrypt and sign a bundle's content at the transmitter for the three evaluated curves



(d) Sign-then-Encrypt mode: Energy consumed to decrypt and verify a bundle's content at the receiver for the three evaluated curves

Fig. 6: Comparison of the influence of payload size and ECC curve choice on energy consumption and performance

The results show that the curve should be chosen carefully to the needs of the application. It needs to be considered whether the additional security that secp256r1 offers compensates the five times higher energy and time consumption compared to secp128r1. Further it need to be considered whether an application needs encryption. If data authenticity is sufficient only the PIB should be used, this reduces the energy and time consumption by half.

4.2 Energy and Performance of Certificates

To evaluate the performance of the certificate validation we transmitted eight consecutive bundles to one destination and measured the time and energy used to process every bundle.

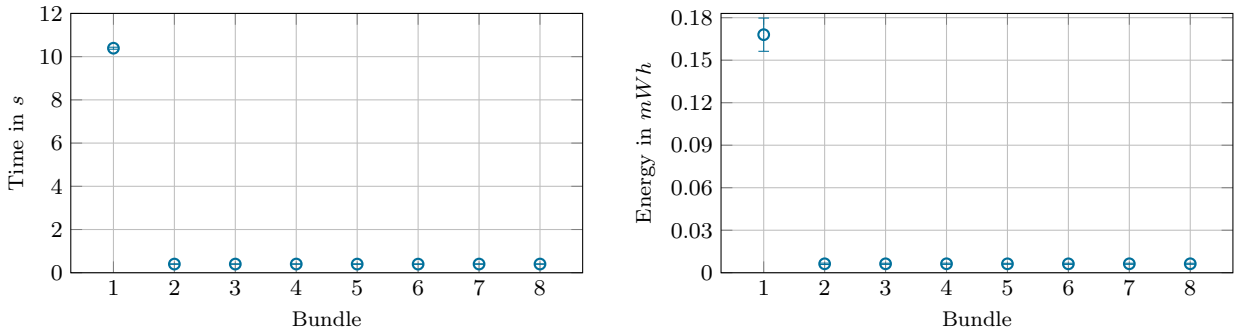
Figure 7a depicts the results for the timing measurements. It shows that the certificate is only validated once when the first bundle arrives at the receiver. The consecutive bundles 2 to 8 need only around 400 ms while the first bundle needs 10.39 s on average. Figure 7b shows the same effect: the first received bundle needs 0.168 mWh while the following bundles need 0.0063 mWh on average. Therefore, our certificate exchange only adds 10 s or 0.162 mWh each time a newly discovered neighbor sends a bundle.

Table 2: Performance of µDTN in comparison to µDTNSec (curve secp192r1, mean time)

Payload	µDTN	µDTNSec	
		Signature	Sign-then-Encrypt
1 B	0.007 s	17.299 s	33.833 s
50 B	0.020 s	17.207 s	34.343 s
100 B	0.035 s	17.337 s	34.743 s
150 B	0.041 s	17.430 s	35.025 s
200 B	0.051 s	17.434 s	35.408 s
250 B	0.073 s	17.540 s	35.774 s

4.3 Performance Penalty of µDTNSec

To evaluate the true performance penalty introduced by µDTNSec, we evaluated the time of the full process of creating, sending, receiving, and processing bundles between two devices with a direct connection. Ta-



(a) Time needed to process the single bundles in a stream of consecutive bundles

(b) Energy needed to process the single bundles in a stream of consecutive bundles

Fig. 7: Timing and energy evaluation of lightweight certificate validation

ble 2 provides the comparative measurements between µDTN and µDTNSec's modes. The comparison shows the huge performance penalty introduced by cryptography on extremely resource-constrained microcontrollers, i. e., our INGA node with its ATmega1284p MCU and AT86RF23X radio.

Thus, we decided to design µDTNSec with end-to-end security in mind and leave hop-by-hop security out of scope, i. e., the current version does not verify signatures on forwarding nodes, as described in Section 3.3.

Another suggestion we deduce from the results is to gather measurement data as long as possible and therefore transmit bundles as big as possible. How long measurement data can be gathered locally depends on the capabilities of the sensor node and the requirements of the application. In longterm patient monitoring for example, larger samples of sensor data can be gathered before forwarding it protected by µDTNSec.

Altogether, the evaluation showed that µDTNSec needs be configured for its application to perform best. Thus, µDTNSec provides no appropriate solution for real-time communications. However, its configurability makes it suitable to a wide range of delay-tolerant applications.

4.4 Quality of RNGs

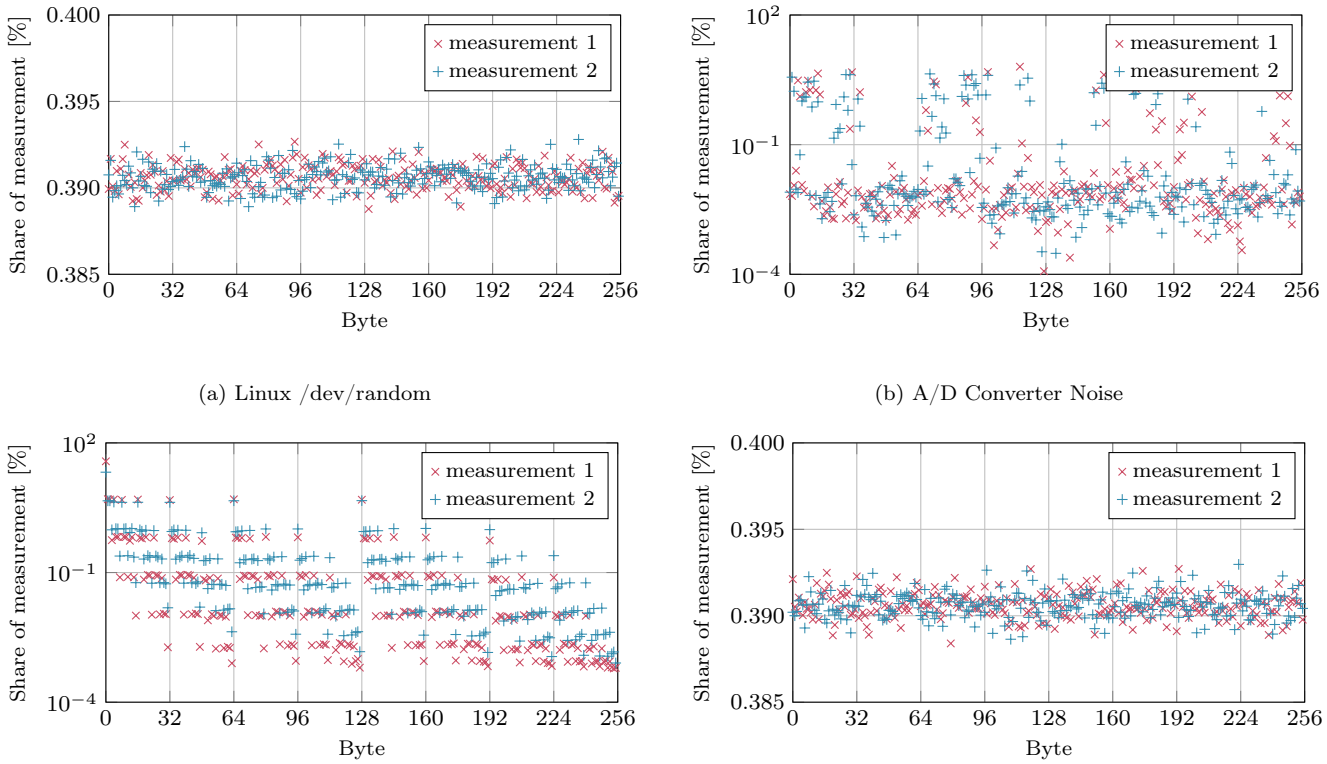
For cryptographic applications a cryptographically secure RNG is required. In case of µDTNSec, the security of the generated nonce, IV, and the session key depends on the randomness of the RNG. We run the Contiki [41] operating system, which provides two sources of randomness, the noise of the least significant bits of the A/D converter and the RNG of the attached radio chip. On our platform we used the AT86RF231 and its successor AT86RF233. Thus, we compare the quality of these three RNGs with the default Linux /dev/random RNG.

To evaluate and compare these four RNGs, we plotted graphs showing the probability distribution of generated values. Furthermore, the statistical analysis program Dieharder [42] has been used. While it is incapable of proofing randomness of RNG output, it can still be used to evaluate the statistical properties of the values' distribution, e.g., by detecting patterns. It is thus helpful to compare RNGs with each other. For each RNG evaluation, we collected two times 64 MiB data. Probability distributions have been plotted in Figures 8a to 8d show.

4.4.1 Linux /dev/random

Before testing the actual targeted RNGs, we ran the test suite on a well-tested RNG as a ground truth for comparison. The data has been generated using /dev/random on a current GNU/Linux system (Ubuntu 14.04.2 LTS, GNU/Linux 3.16.0-30 x86_64).

Looking at the output of the analysis by Dieharder and at the probability distribution of the different byte values (cf. Figure 8a) in the generated 64 MiB, these results look exactly as expected from a good RNG. The byte values appear in nearly the same frequency. A perfect uniform distribution would be expected at 0.3906 %. The Dieharder evaluation results in 13/15 'passed' tests. The tests which are 'failed' or 'weak' are probably due to the comparatively small amount of analyzed data. This also results in the repeated usage of the same random data for some tests. To have enough data for theses tests, presumably approx. 1 GiB random data would be required. However, we assume that for a basic quality rating of a RNGs randomness 64 MiB are enough.



(a) Linux /dev/random

(b) A/D Converter Noise

(c) AT86RF231 RNG

(d) AT86RF233 RNG

Fig. 8: Comparison between RNGs: Probability distribution of generated values

4.4.2 A/D Converter Noise

Contiki ships with a RNG that uses the noise of the two least significant bits of the A/D converter to generate random numbers. Due to the relatively long A/D conversion, this RNG can only reach a data rate of 0.2 KiB/s for the generation of random numbers.

All 15 Dieharder tests 'failed', which means — with a high probability — that the RNG based on A/D converter noise does not produce cryptographically secure random numbers. This outcome is confirmed when looking at the byte values in Figure 8b. It can be seen that some byte ranges appear with a much higher probability, i.e., up to 10 %. In reverse, this means that some byte ranges appear with a much lower probability, i.e., down to 0.01 %. Conclusively, random numbers generated based on noise from the A/D converter do not appear to be uniformly distributed and are, thus, not suitable for cryptographic applications.

4.4.3 AT86RF231 RNG

A faster and more trustworthy source of randomness is provided on the INGA platform as a two bit RNG provided by the built-in radio. According to the data sheet, AT86RF231 provides a "True Random Number Generation for Security Application" [43]. This RNG provides a two bit random value every 1 µs. This corresponds to a theoretical maximum data rate of approx. 244.1 KiB/s. In our test setup using the Contiki operating system, a maximum data rate of 10.1 KiB/s has been reached. Because the SPI bus, which is connected to the sensor node, runs with a maximum of 4 MHz[6], the maximum possible data rate is already limited to approx. 61.0 KiB/s. Taking into account the additional overhead of the Contiki operating system, this explains the real-world data rate.

Looking at Figure 8c and the results from Dieharder, this RNG has serious problems. All 15 tests of Dieharder 'fail'. The probability distribution shows that values which contain many '0's in their binary representation, have a much higher probability of occurrence. For example, all values containing no or only one '1' in their binary representation (0, 1, 2, 4, 8, 16, 32, 128) can be seen easily by looking at Figure 8c. We were not able to find the definitive cause of the bad result.

[6] The ATmega1284P [44] of the INGA platform runs with 8 MHz and supports a maximum SPI data rate of $f/2$.

4.4.4 AT86RF233 RNG

Using the same firmware on the current version of the INGA platform, which is based on a new revision of the radio, the AT86RF233 [45], does not exhibit this problem.

The results from Dieharder as well as the probability distribution shown in Figure 8d demonstrate a nearly similar quality as /dev/random. 13/15 (Measurement 1) and 12/15 (Measurement 2) 'passed' Dieharder tests and only one 'weak' test result, show — with high probability — that this RNG can be used for cryptographic applications with good confidence.

4.4.5 Summary

Based on the discussed results of three different RNGs available on the INGA platform running the Contiki operating system, only AT86RF233 provides random numbers with a quality, which can be used for cryptographic applications. The other two RNGs, the AT86RF231 RNG and the one based on the noise of the A/D converter, must not be used for cryptography because their output exhibit patterns. Thus, parts of their output could be guessed with certain probability, which would break the usage of cryptographic algorithms.

5 Conclusion

We presented µDTNSec, an extension to the BP implementation µDTN for Contiki OS. We provide a Public Key Infrastructure with lightweight certificates that are exchanged via IPND. Due to the performance and energy impact of ECC, the required security level must be chosen carefully. For example, if only authenticity is required, µDTNSec's signature mode should be used without encryption. We evaluated the Random Number Generators for our cryptographic operations provided by the AT86RF231 radio, its successor AT86RF233, and one based on the noise of the A/D converter. We found several issues making only AT86RF233 suitable for cryptographic applications.

µDTNSec is licensed as open-source and can be downloaded at https://www.ibr.cs.tu-bs.de/projects/mudtn/.

References

1. Dominik Schürmann, Georg von Zengen, Marvin Priedigkeit, and Lars Wolf. uDTNSec: A Security Layer for Disruption-Tolerant Networks on Microcontrollers. In *Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net)*, pages 1–7, June 2017.
2. S. Chen, H. Xu, D. Liu, B. Hu, and H. Wang. A Vision of IoT: Applications, Challenges, and Opportunities With China Perspective. *IEEE Internet of Things Journal*, 1(4):349–359, August 2014.
3. Georg von Zengen, Felix Büsching, Wolf-Bastian Pöttner, and Lars Wolf. An Overview of μDTN: Unifying DTNs and WSNs. In *Proceedings of the 11th GI/ITG KuVS Fachgespräch Drahtlose Sensornetze (FGSN)*, Darmstadt, Germany, September 2012.
4. S. Burleigh K. Scott. Bundle protocol specification. RFC 5050, December 2007.
5. D. Ellard, R. Altman, A. Gladd, D. Brown, and R. in't Velt. DTN IP Neighbor Discovery (IPND). draft-irtf-dtnrg-ipnd-03, November 2015.
6. S. Symington, S. Farrell, H. Weiss, and P. Lovell. Bundle security protocol specification. RFC 6257, May 2011.
7. Sebastian Schildt, Johannes Morgenroth, Wolf-Bastian Pöttner, and Lars Wolf. IBR-DTN: A lightweight, modular and highly portable bundle protocol implementation. *Electronic Communications of the EASST*, 37:1–11, January 2011.
8. DTN2 Reference Implementation.
9. S. Burleigh. Compressed Bundle Header Encoding (CBHE). RFC 6260, May 2011.
10. Wolf-Bastian Pöttner, Felix Büsching, Georg von Zengen, and Lars Wolf. Data elevators: Applying the bundle protocol in delay tolerant wireless sensor networks. In *The Ninth IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS)*, Las Vegas, Nevada, USA, October 2012.
11. Stephan Rottmann, Robert Hartung, Jan Käberich, and Lars C Wolf. Amphisbaena: A Two-Platform DTN node. In *The 13th International Conference on Mobile Ad-hoc and Sensor Systems (MASS 2016)*, Brasilia, Brazil, October 2016.
12. M. Feldmann and F. Walter. μPCN — A bundle protocol implementation for microcontrollers. In *2015 International Conference on Wireless Communications Signal Processing (WCSP)*, October 2015.
13. Sergiu Nedevschi Rabin Patra. DTNLite: A Reliable Data Transfer Architecture for Sensor Networks. *CS294-1: Deeply Embedded Networks (Lecture)*, 2003.
14. W. Trappe, R. Howard, and R. S. Moore. Low-energy security: Limits and opportunities in the internet of things. *IEEE Security Privacy*, 13(1):14–21, January 2015.
15. Nils Gura, Arun Patel, Arvinderpal Wander, Hans Eberle, and Sheueling Chang Shantz. Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs. In *Cryptographic Hardware and Embedded Systems (CHES)*, pages 119–132. Springer, 2004.
16. NIST. Recommendation for Key Management. *Special Publication 800-57 Part 1 Rev. 4*, 2016.
17. Yang Xiao, Venkata Krishna Rayi, Bo Sun, Xiaojiang Du, Fei Hu, and Michael Galloway. A survey of key management schemes in wireless sensor networks. *Computer Communications*, 30(11-12):2314–2341, 2007.
18. Piotr Szczechowiak, Leonardo B. Oliveira, Michael Scott, Martin Collier, and Ricardo Dahab. NanoECC: Testing the limits of Elliptic Curve Cryptography in sensor networks. In Roberto Verdone, editor, *Wireless Sensor Networks*, volume 4913 of *Lecture Notes in Computer Science*, pages 305–320. Springer, 2008.
19. Leonardo B Oliveira and Ricardo Dahab. Pairing-based cryptography for sensor networks. In *5th IEEE International Symposium on Network Computing and Applications, Cambridge*, 2006.

20. Leonardo B. Oliveira, Diego F. Aranha, Conrado P.L. Gouvêa, Michael Scott, Danilo F. Câmara, Julio López, and Ricardo Dahab. TinyPBC: Pairings for authenticated identity-based non-interactive key distribution in sensor networks. *Computer Communications*, 34(3):485–493, 2011. Special Issue of Computer Communications on Information and Future Communication Security.
21. D. F. Aranha and C. P. L. Gouvêa. RELIC is an Efficient LIbrary for Cryptography. `https://github.com/relic-toolkit/relic`.
22. M. Sethi, J. Arkko, and A. Keranen. End-to-end security for sleepy smart object networks. In *IEEE 37th Conference on Local Computer Networks Workshops (LCN Workshops)*, pages 964–972, October 2012.
23. Ruan de Clercq, Leif Uhsadel, Anthony Van Herrewege, and Ingrid Verbauwhede. Ultra low-power implementation of ECC on the ARM Cortex-M0+. In *Proceedings of the 51st Annual Design Automation Conference (DAC)*, pages 112:1–112:6, New York, NY, USA, 2014. ACM.
24. Atmel Corporation. AT86RF231/ZU/ZF datasheet.
25. Dominik Schürmann, Felix Büsching, Sebastian Willenborg, and Lars C Wolf. RAIM: redundant array of independent motes. In *Conference on Networked Systems (NetSys'17)*, Göttingen, Germany, March 2017.
26. Chris Karlof, Naveen Sastry, and David Wagner. TinySec: a link layer security architecture for wireless sensor networks. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys '04)*, pages 162–175, New York, NY, USA, 2004. ACM.
27. R. Doriguzzi Corin, G. Russello, and E. Salvadori. TinyKey: A light-weight architecture for Wireless Sensor Networks securing real-world applications. In *Eighth International Conference on Wireless On-Demand Network Systems and Services (WONS)*, pages 68–75, January 2011.
28. M. Luk, G. Mezzour, A. Perrig, and V. Gligor. MiniSec: a secure sensor network communication architecture. In *6th International Symposium on Information Processing in Sensor Networks (IPSN)*, pages 479–488. IEEE, 2007.
29. An Liu and Peng Ning. TinyECC: A configurable library for elliptic curve cryptography in wireless sensor networks. In *International Conference on Information Processing in Sensor Networks (IPSN'08)*, pages 245–256. IEEE, 2008.
30. L. Casado and P. Tsigas. ContikiSec: A secure network layer for wireless sensor networks under the contiki operating system. *Identity and Privacy in the Internet Age*, pages 133–147, 2009.
31. A. Capossele, V. Cervo, G. De Cicco, and C. Petrioli. Security as a CoAP resource: An optimized DTLS implementation for the IoT. In *2015 IEEE International Conference on Communications (ICC)*, pages 549–554, June 2015.
32. Philippe Pittoli, Pierre David, and Thomas Noël. DTLS improvements for fast handshake and bigger payload in constrained environments. In Nathalie Mitton, Valeria Loscri, and Alexandre Mouradian, editors, *Ad-hoc, Mobile, and Wireless Networks*, pages 251–262, Cham, 2016. Springer International Publishing.
33. Sanaz Rahimi Moosavi, Tuan Nguyen Gia, Ethiopia Nigussie, Amir M. Rahmani, Seppo Virtanen, Hannu Tenhunen, and Jouni Isoaho. End-to-end security scheme for mobility enabled healthcare internet of things. *Future Generation Computer Systems*, 64:108 – 124, 2016.
34. 'Bg'. AVR–Crypto–Lib.
35. SECG. SEC 2: Recommended elliptic curve domain parameters. *Standards for Efficient Cryptography Group, Certicom Corp*, January 2010.
36. Felix Büsching, Ulf Kulau, and Lars Wolf. Architecture and Evaluation of INGA - An Inexpensive Node for General Applications. In *IEEE Sensors*, pages 842–845, Taipei, Taiwan, October 2012. IEEE.
37. SECG. SEC 1: Elliptic curve cryptography. *Standards for Efficient Cryptography Group, Certicom Corp*, May 2009.
38. Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ECDSA). *International Journal of Information Security*, 1(1):36–63, 2001.
39. Robert Hartung, Ulf Kulau, and Lars C Wolf. Demo: PotatoScope - scalable and dependable distributed energy measurement for WSNs. In *IEEE SECON 2016 Conference Proceedings*, London, UK, June 2016.
40. Nicholas Jansma and Brandon Arrendondo. Performance comparison of elliptic curve and RSA digital signatures. Technical report, University of Michigan, College of Engineering, April 2004.
41. Adam Dunkels, Bjorn Gronvall, and Thiemo 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*, LCN '04, pages 455–462, Washington, DC, USA, 2004. IEEE Computer Society.
42. Robert G. Brown. Dieharder: A Random Number Test Suite v3.31.1, June 2011.
43. Atmel Corporation. Low Power 2.4 GHz Transceiver for ZigBee, IEEE 802.15.4, 6LoWPAN, RF4CE, SP100, WirelessHART, and ISM Applications - AT86RF231, September 2009.
44. Atmel Corporation. 8-bit AVR Microcontroller with 128K Bytes In-System Programmable Flash - ATmega1284P, November 2009.
45. Atmel Corporation. Low Power 2.4 GHz Transceiver for ZigBee, IEEE 802.15.4, 6LoWPAN, RF4CE, SP100, WirelessHART, and ISM Applications - AT86RF233, July 2014.