TinyTO: TWO-WAY AUTHENTICATION FOR CONSTRAINED DEVICES IN THE INTERNET OF THINGS

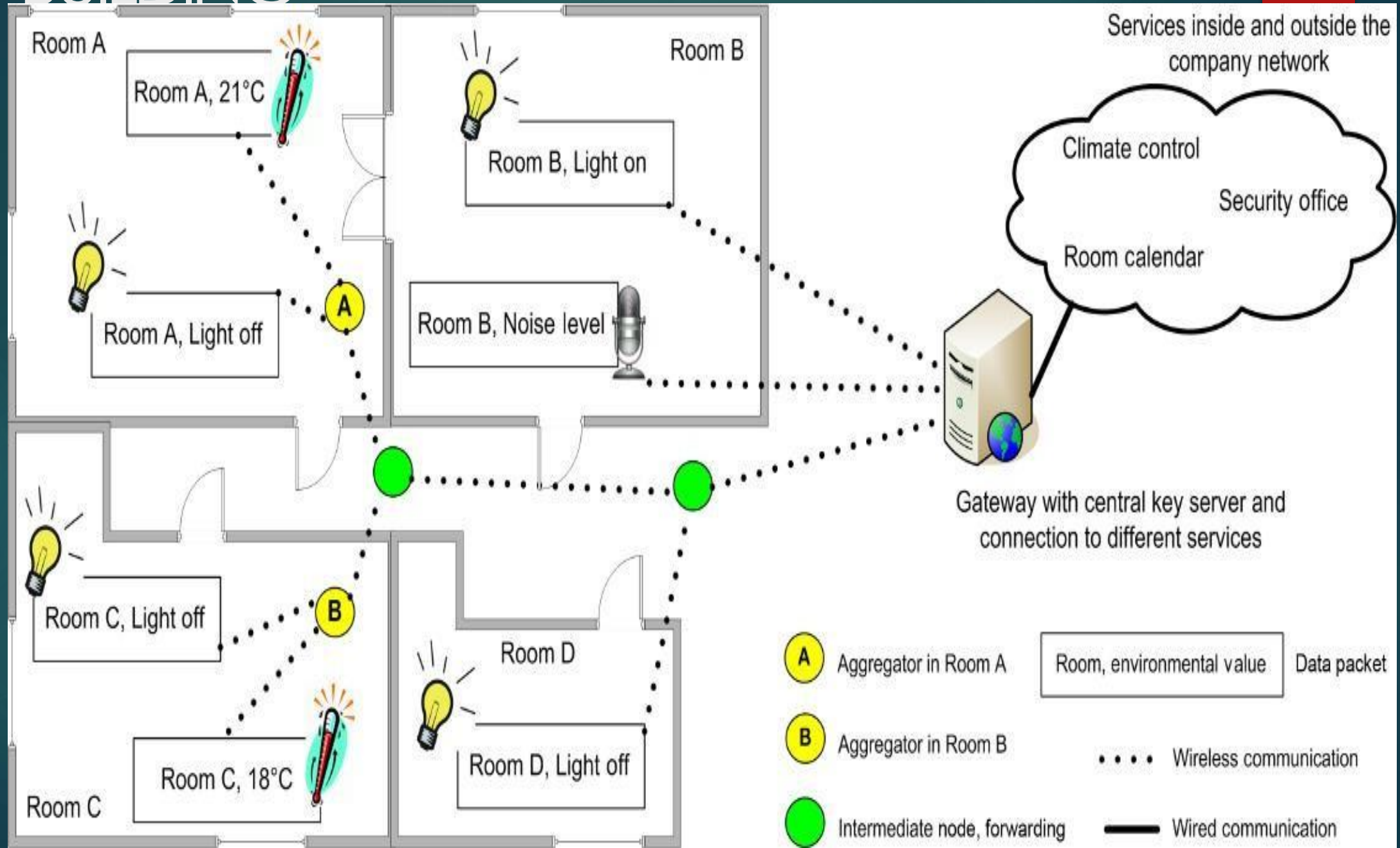
INTRODUCTION

- The Internet of Things (IoT) consists of manifold devices, ranging from IP networks and servers to small devices such as Wireless Sensor Network (WSN) devices (e.g., Radio-Frequency IDentification (RFID) tags or sensor nodes).
- Throughout the years, especially WSN consisting of constrained devices with limited resources in memory, energy, and computational capacity, rapidly gained popularity.
- Thus, the questions raised of how to integrate them into the IoT and what challenges occur when looking at their constrained resources.
- The number of possible deployments of such networks rises, and more applications have a need for confidential and authenticated communication within the network.

INTRODUCTION (CONTINUED)

- This security issue must be addressed, due to the fact that sensitive information (e.g., identity (ID), names, or Global Positioning System (GPS) information) is linked almost everywhere to all kinds of collected data, such as temperature, sound, and brightness.
- Hence, collected data is no longer anonymous and is often desired to be kept confidential.
- The following diagram this case for a building scenario, where environmental data is collected in rooms and transmitted over multiple hops to the gateway in order to make the data available to applications, such as climate control, security office, and room calendar.
- If room information can be retrieved by eavesdropping due to missing security in the communication, then an attacker would be aware of sensitive information and could plan, for example, a burglary.

BUILDING


Room A
Room A, 21°C
Room A, Light off
A
Room B
Room B, Light on
Room B, Noise level
Services inside and outside the company network
Climate control
Security office
Room calendar
Gateway with central key server and connection to different services
Room C, Light off
B
Room C, 18°C
Room C
Room D
Room D, Light off
A Aggregator in Room A
Room, environmental value
Data packet
B Aggregator in Room B
Wireless communication
Intermediate node, forwarding
Wired communication

INTRODUCTION (CONTINUED)

- Therefore, collected data must be transmitted in a secure manner and/or over a secure channel providing end-to-end security, giving only authorized entities (e.g., gateway, security system, or company members) access to this confidential information.
- But how is this supposed to be done? Keeping in mind that WSNs are part of the IoT and consist of constrained devices with limited resources, any security risks are aggravated by WSN design and security requirements of the IoT.
- Ultimately, an end-to-end security solution is required to achieve an adequate level of security.
- Protecting data only after it leaves the scope of the local network (e.g., WSN) is not sufficient.

INTRODUCTION (CONTINUED)

- Using existing technologies (e.g., Secure Sockets Layer (SSL)/Transport Layer Security (TLS) or cryptography) is the easiest way to achieve the goal of secure data- transmission.
- But this becomes increasingly challenging when looking at WSN devices used today (e.g., RFIDs, heart beat monitor, or environmental sensors), as their resources are strictly limited in memory, power, and computational capacity.
- Those WSN devices are divided into constrained classes corresponding to their computational capacity and memory resources as shown in following table.
- Security support is very challenging when assuming they are Class 1 devices (e.g., TelosB), as done for the proposed solution TinyTO, because they offer only about 10 kByte RAM and 100 kByte ROM.

INTRODUCTION (CONTINUED)

Table 13.1 Device Classes (1 KiB = 1024 Byte) [11]

Name	RAM	ROM	IP Stack	Security
Class 0	≪10 KiB	≪100 KiB	NO	NO
Class 1	~10 KiB	~100 KiB	CoAP [15], BLIP [16,17]	YES
Class 2	~50 KiB	~250 KiB	HTTP, TLS	YES

INTRODUCTION (CONTINUED)

- A standard approach for securing communications in the Internet is SSL/TLS, relying on asymmetric cryptography such as RSA, which requires many resources and computational capacity, and, thus, is only feasible for at least Class 2 devices (which have approximately 50 kByte RAM and 250 kByte ROM).
- Additional challenges are the device diversity in today's WSNs, the network size itself, and multiple requirements (e.g., lifetime or security support) due to the target application.
- Developing a proper solution is still a challenge, especially for security issues under consideration of the aforementioned challenges and constraints.
- Depending on the application, it might be prohibited to reuse existing solutions (e.g., military area).

INTRODUCTION (CONTINUED)

- In general, it is preferred to either reuse standards or to develop a generic solution that can be integrated without major modifications and would not require hardware features such as cryptographic coprocessors, certain radio modules, or specific processors.
- On the software side, it would not require a specific protocol stack, but it would rely on the most basic interfaces, and be kept separate from applications in order to allow simple integration into any used protocol-stack with a limited number of connection points (i.e., interfaces).
- Furthermore, all additional features have to avoid affecting excessive performance- and memory-consumption.

INTRODUCTION (CONTINUED)

- Based on the aforementioned hardware and application requirements, the proposed security solution TinyTO, an optimized two-way authentication solution for tiny devices, provides confidential data transfer with an additional integrity protection and data authentication, as well as a two-way authentication between sender and receiver of messages, delivering end-to-end security even for Class 1 devices.
- This is achieved by introducing an efficient handshake with a direct authentication and key exchange between pairs of nodes in the network, thus setting up an encrypted data transfer with an integrated encryption scheme.
- To minimize overall hardware requirements, the Elliptic Curve Cryptography (ECC) is used for key generation, key exchange, encryption, decryption, and signature generation.

INTRODUCTION (CONTINUED)

- Initially, each node is only familiar with the gateway.
- This relationship is authenticated with an individual shared key (in TinyTO of 16 Byte length), which is only known to the gateway and to the node, and is deployed to all nodes during the initial programming routine.
- Individual keys between nodes are either established during the handshake performance or can be requested by a node from the gateway (e.g., in case of communication with the aggregator).
- TinyTO is designed to fit WSN requirements, is application-independent, and allows for an easy integration into existing applications due to its modular nature.
- TinyTO explicitly supports in-network aggregation by enabling a full and secure end-to-end communication without the need for a networkwide shared secret.

SECURITY ASPECTS AND SOLUTIONS

- The necessity of providing an end-to-end security solution in WSNs is not entirely new.
- Over the years, different approaches have emerged that address various security issues.
- Thus, an often-quoted solution is to predistribute symmetric keys.
- However, flexibility of the deployment, connectivity between nodes, and resilience against attackers is limited significantly.
- Instead, a solution that applies public key authentication to smaller-resource-demanding symmetric key operations, where a one-way hash function is used to authenticate public keys.
- The basic idea is to allow for individual nodes to verify that a transmitted public key matches the claimed identity, without relying on a trusted third party (e.g., Certificate Authority (CA)).

SECURITY ASPECTS AND SOLUTIONS (CONTINUED)

- For an exhausting mapping among all keys and identities, a large number of keys and certificates must be stored on every node, which is not feasible.
- Hence, a hash function, mapping from identity to the hash value of the corresponding public key, is preshared.
- Thus, only hash values and identities must be compared, which requires only a fraction of the memory and computational power.
- This can be optimized further by using Merkle Trees, in which nonleaf nodes are labeled with the hash of the labels of its children.

SECURITY ASPECTS AND SOLUTIONS (CONTINUED)

- ECC determines a promising option for WSN security solutions, in particular, for message encryption, because ECC can deliver strong security with only a small amount of resources needed.
- A 192-bit ECC key provides the same level of security as an RSA-key in the range of 1024 bit to 2048 bit.
- ECC is viable for key generation, key exchange, encryption, decryption, and signatures, especially in resource-constrained applications.

SECURITY ASPECTS AND SOLUTIONS (CONTINUED)

- The HIP DEX protocol for hop-by-hop secure connections using a Diffie–Hellman key exchange for public keys and the AES encryption for the session key-exchange.
- Computational requirements are reduced by limiting cryptographic primitives to a minimum (e.g., removing expensive signature algorithms and any form of cryptographic hash functions).
- Cryptographic challenges are included in the first messages of the handshake proposed, in order to avoid flooding attacks.
- Identity authentication is achieved by password verification within the handshake, where nodes need to know their respective passwords in advance.

SECURITY ASPECTS AND SOLUTIONS (CONTINUED)

- The PAuthKey protocol for application-level end-to-end security overcomes the problem of two-way authentication (i.e., mutual authentication) between sensor nodes.
- It provides pervasive lightweight authentication and keying mechanisms, allowing nodes to establish secure and authenticated communication channels with each other.
- PAuthKey employs ECC-based implicit certificates, using a trusted central CA to handle authentication security.
- Thus, it stands in contrast to other authentication approaches, as certificates are generally considered to be resource-challenging for WSNs, and they require additional hosting infrastructure (e.g., CA) or hardware (e.g., Trusted Platform Module (TPM)) that can be integrated on the gateway or as an external network entity.

SECURITY ASPECTS AND SOLUTIONS (CONTINUED)

- The UbiSec&Sens project offered a toolbox of security-aware components.
- The proposed Zero Common Knowledge (ZCK) protocol for authentication can establish well-defined pairwise security associations between entities, even in the absence of a common security infrastructure and pre-shared secrets.
- ZCK authentication is based on re-recognition between entities, allowing entities to authenticate any other entity known from the past.
- This approach does not provide full security, as required, for instance, for financial transactions because the first contact between entities cannot be authenticated.
- However, in a scenario without any form of preestablished knowledge or a trusted third party, ZCK provides the best level of security that can be achieved under those limitations given.

SECURITY ASPECTS AND SOLUTIONS (CONTINUED)

- The ZCK protocol itself does not cater to a key exchange, but can be used in combination with any form of cryptography, such as Diffie–Hellman.
- TinyDTLS—a DTLS-based solution for constrained (tiny) devices—provides end-to-end security, but targets Class 2 devices with additional memory resources.
- In this case the platform used includes a TPM, offering additional dedicated memory and computational power for costly security functions.
- TinyDTLS performs a TLS handshake, using X.509 certificates for authentication and Advanced Encryption Standard (AES) for encryption, but still exceeds most alternatives, due to the high amount of available resources on its target devices.

SECURITY ASPECTS AND SOLUTIONS (CONTINUED)

- An advantage of using this solution is its compatibility with established standard protocols such as SSL/TLS.
- The security aspects addressed by TinyTO are a direct result of the aforementioned existing solutions and of the final design-decisions, especially to counter Unknown Key-Share Attacks (UKSA) and Man-In-The-Middle (MITM) attacks.
- Therefore, TinyTO's goals are summarized as:

SECURITY ASPECTS AND SOLUTIONS (CONTINUED)

1. TinyTO brings end-to-end security to Class 1 devices by providing two-way authentication.

2. The TinyTO's handshake design with two-way authentication adds immensely to the security level without an involvement of certificates and CA in the network's infrastructure, or special hardware components such as TPM on the device.

3. TinyTO is protected against MITM attacks, in contrast to other solutions, such as UbiSec&Sens and ZCK, for Class 1 devices.

SECURITY ASPECTS AND SOLUTIONS (CONTINUED)

4. TinyTO allows for adding devices dynamically to the secure network, in contrast to static Merkle Trees.

5. TinyTO uses the Routing Protocol for Low power and Lossy Networks (RPL), which offers various measurements to improve routing, which, in turn, can be used for an attack detection and defense.

- In order to address these goals, TinyTO requires preprogrammed master keys for authentication between devices and the gateway, RPL routing, and support of an ECC functionality for encryption and signing.

DESIGN DECISIONS

- An ideal solution for the two-way authentication should work generically on WSN nodes of all classes, especially because the trend goes toward heterogeneous WSNs.
- However, because WSN nodes are primarily designed to collect data, they prioritize frugality and longevity over processing-power and memory size.
- Class 1 devices are, by definition in RFC7228, very constrained to run security schemes beyond the very specific implementations.
- Thus, the newly proposed end-to-end security solution targets Class 1 devices as a minimum requirement.

DESIGN DECISIONS (CONTINUED)

- Even though Class 1 devices can connect to the Internet without additional proxies or gateways, they are limited in communication with peers, if those peers have a full protocol-stack employed, which would overwhelm available resources of Class 1 devices.
- Therefore, Class 1 devices require a specifically designed protocol-stack for constrained devices, such as the Constrained Application Protocol (CoAP) over User Datagram Protocol (UDP).
- Consequently, traditional security concepts for wireless networks, such as Wired Equivalent Privacy (WEP) or TLS in their native form, are unsuitable for WSNs.

DESIGN DECISIONS (CONTINUED)

- One approach to adapt the traditional Public-Key Cryptography (PKC) to WSNs is the integration of extra hardware into nodes, for performing security operations and operations that are separate from the main application and the node processor.
- At first glance, Class 2 devices have more resources and can be used for this purpose.
- Among other functionalities, Class 2 devices can deliver Internet-level security by providing confidentiality and message authentication at high speed.
- A TPM chip outperforms most alternative solutions of similar resource levels
- But at second glance, as a drawback, all nodes in a WSN need to be equipped with an appropriate amount of resources (e.g., more RAM/ROM or using a TPM) to apply the security scheme network-wide.

DESIGN DECISIONS (CONTINUED)

- A class 1 device cannot build and maintain an RFC-compliant PKI while executing its main task—data collection and data forwarding—that is already resource-consuming in itself.
- One commonly used OpenSSL X.509 RSA-1024 certificate alone has a size of about 800 Byte, and adding the corresponding RSA key pair to this takes an additional 800 Byte.
- Assuming an aggregation support, n+ 2 certificates and n+ 2 key pairs for a degree of aggregation (DOA) of n must be stored, quickly filling the available memory.
- For example, following those calculations, an aggregator with DOA = 5 needs to store an additional 11.2 kByte of data, only for certificates and corresponding key pairs.

DESIGN DECISIONS (CONTINUED)

- This extreme memory consumption can be avoided by utilizing PKC only between designated node pairs, so that every node (aggregator or collector) only has to store its own key pair and the public key of the given recipient (ie, gateway or next hop).
- The general feasibility of PKC on simple 8-bit processors, as typically found within WSN nodes.
- Therefore, TinyTO's security solution is based on PKC.
- Furthermore, memory and energy-consumption savings are gained by applying ECC instead of RSA (Rivest, Shamir, and Adleman) for key generation, key exchange, signatures, and encryption.

DESIGN DECISIONS (CONTINUED)

- The National Institute of Standards and Technology (NIST) recommends that SP 800-57 explains that an RSA key in the range of 1024–2048 bit delivers the same security level as a 160-bit ECC key, that is, the same amount of resources is required to break them.
- Even more, ECC implementations are faster and require less energy compared to equally secure RSA algorithms.
- In general, standardization bodies and researchers agree on a set of security objectives that are necessary to achieve information security: confidentiality, integrity, authenticity, availability, and accountability of all messages.

DESIGN DECISIONS (CONTINUED)

- Furthermore, a set of requirements that are particular to WSNs and to the development goals for TinyTO must be considered:

(1)End-to-end security to prevent eavesdropping and spoofing attacks, meaning risk for the communication because the underlying network infrastructure is only partially under the user's control and might be compromised. Especially in a WSN, where multi-hop communications are common, authentication and key exchange are essential design goals.

(2)In WSNs, connections are often not lossless. Transmission Control Protocol (TCP) erroneously invokes congestion-control mechanisms to counter the loss of packets, which drastically impact the performance, and results in the UDP to serve as a better choice for WSNs.

DESIGN DECISIONS (CONTINUED)

(3)Two-way authentication denotes two entities authenticating each other at the same time. In the scope of WSNs, it is not sufficient to authenticate only the sender to the receiver, but the sender has to be sure also about the identity and authorization of the potential receiver of confidential information.

(4)ECC is promising to save resources, when performing PKC in TinyTO. For message encryption an Integrated Encryption Scheme (IES) is applied, especially to harness the speed-advantage of symmetric encryption for large amounts of data without the drawback of a repeated key-exchange for every transmission, which otherwise is necessary so that no secret credential would be used more than once.

DESIGN DECISIONS (CONTINUED)

- An authentication protocol should always be linked to the key exchange for later encryption, otherwise an attacker might just wait until the authentication is completed to compromise the established communication channel thereafter.
- The objective of a key-exchange protocol in a very intuitive way: A key-exchange protocol is secure, if it is impossible or at least infeasible for an attacker to distinguish the generated key from a random value.
- The same fundamental concept can be applied to the Authenticated Key Exchange (AKE) protocol.
- But additionally, entity (or party) authentication has to guarantee the identity of communicating parties in the current communication session, and, therefore, has to prevent impersonation.

DESIGN DECISIONS (CONTINUED)

- A good authentication protocol combines several properties, as explained by various researchers, and is relevant to TinyTO's design:

(1)Forward secrecy guarantees, such that, if a generated private key of one or more of the participating entities is compromised, the security of previous communications is not affected.

(2)Asymmetry of messages is required to prevent reflection attacks, where one entity simply replays the same message back to the sender; it is desirable to avoid symmetries. In other words, the authentication responses of two different parties must not be identical.

DESIGN DECISIONS (CONTINUED)

(3)Direct authentication is provided by a protocol if the authentication is completed in a successful handshake, that is, if both parties have proven knowledge of the shared secret.

(4)Timestamps are to be avoided, because not every participating entity can be expected to maintain a reliable local clock, which must be synchronized periodically, too.

TINYTO PROTOCOL

- Due to TinyTO's main goal of supporting an end-to-end security with two-way authentication on Class 1 devices, the authentication protocol has to always include a key exchange, such that several possible handshake candidates can be considered in practice, leading to the final design and implementation of TinyTO.
- First, handshake candidates for TinyTO and their drawbacks are introduced.
- Second, the resulting TinyTO handshake, including two-way authentication purposes and aggregation support, are described.
- Finally, key information on the respective implementation is presented.

TINYTO PROTOCOL - POSSIBLE HANDSHAKE PROTOCOL CANDIDATES

- Handshake protocol candidates considered in this section support a two-way authentication of two independent entities without prior information exchange, which make them highly appropriate for TinyTO.
- From this stage on, the traditional naming pattern of cryptography is applied to protocol descriptions, assuming two communication parties—Alice and Bob—which are instantiated as sensor nodes.
- At first glance the Station-to-Station protocol (STS) seems to be an ideal candidate for TinyTO because STS is based on a Diffie–Hellman's key exchange, followed by an exchange of authentication signatures.

TINYTO PROTOCOL - POSSIBLE HANDSHAKE PROTOCOL CANDIDATES (CONTINUED)

- Both parties, Alice (A) and Bob (B), compute their private key x and a public key X in the beginning.
- Next, Alice sends her public key X_A to Bob.
- Once Bob receives X_A, he can compute shared secret K_{AB} with X_A and x_B, according to the Diffie–Hellman's key-exchange algorithm.
- Bob can now encrypt any message to Alice using K_{AB}.
- For decryption purposes Bob sends X_B back to Alice, so that she can compute the same shared secret K_{AB}.
- Additionally, Bob sends a token consisting of both public keys, signed with his own private key to authenticate himself.
- Alice can use X_B to verify that Bob was indeed the same person who had signed the message and computed the shared secret.

TINYTO PROTOCOL - POSSIBLE HANDSHAKE PROTOCOL CANDIDATES (CONTINUED)

- Bob is now authenticated to Alice.
- As the last step of the two-way authentication, Alice constructs an authentication message and sends it to Bob to authenticate herself to Bob.
- To avoid unnecessary communication overhead, the second key-exchange message is combined with the first authentication message.
- As a result, STS entails the establishment of a shared-secret key between two parties, with mutual entity-authentication and mutual implicit key-authentication.
- The forward secrecy can be provided by deriving a new ephemeral key from the shared secret for the encryption of every message in that exchange.
- The signatures are used to obtain protection against impersonation during the exchange.

TINYTO PROTOCOL - POSSIBLE HANDSHAKE PROTOCOL CANDIDATES (CONTINUED)

- However, there are two main shortcomings:

(1)Although the STSis relatively simple to execute, it does not include any explicit key-confirmation. Neither Bob nor Alice inherently can be sure that the other party has actually computed a shared secret without additional messages.

(2)Furthermore, STS is vulnerable to UKSAs and the MITM attack. To prevent UKSAs and to provide explicit key-authentication, the signatures used can be encrypted additionally with the successfully computed K_{AB}.

- Thus, Bob is assured that he shares K_{AB} only with one single party, namely Alice.
- Because he has created X_B specifically for this handshake and Alice has signed X_B and X_A, her signature is now tied to this particular handshake.

TINYTO PROTOCOL - POSSIBLE HANDSHAKE PROTOCOL CANDIDATES (CONTINUED)

- By encrypting the message with the resulting K_{AB}, Alice assures Bob that she was indeed the entity who had created X_A.
- Similar assumptions can be made from the position of Alice.
- This modification requires more computational capacity, due to parallel execution of signature and symmetric encryption algorithms.
- Hence, for WSN devices below Class 2, it is desirable to avoid this sort of overhead.
- The need for encryption can be resolved by including the identity of both parties in the exchanged signatures, resulting in the adapted STS protocol.
- When combining the adapted STS with identities in signatures it becomes almost functionally identical to the Bellare–Canetti–Krawczyk protocol (BCK).
- The only difference in BCK is the absence of the sending parties' identities.

TINYTO PROTOCOL - POSSIBLE HANDSHAKE PROTOCOL CANDIDATES (CONTINUED)

- According to Basin et al., it is generally desirable to include identities of both parties, to avoid the spoofing of identities.
- But in a bidirectional exchange, as is the case for BCK, it is only required to include the receiver's identity: at least in one direction, the receiving party is presented with an invalid signature that does not contain its own identity, and as a result it immediately aborts the handshake.
- At this point, BCK is computationally relatively inexpensive, but still vulnerable to MITM attacks.
- This weakness boils down to the fact that it is impossible to reliably map a public key to a specific entity, that is, to derive their public key from their identity.

TINYTO PROTOCOL - POSSIBLE HANDSHAKE PROTOCOL CANDIDATES (CONTINUED)

- Any party can claim any public key as its own.
- To counteract, it is essential to strongly couple a public key with the respective identity.
- The prevalent solution for this is to introduce a PKI with certificates and trusted CAs, as proposed for TLS.
- A certificate contains the identity and the corresponding public key.
- Entities can be assured of the correct coupling between key and identity, because trusted CAs had constructed the certificate.
- However, BCK itself does not suit the given requirement of Class 1 devices, but can be used as a baseline.

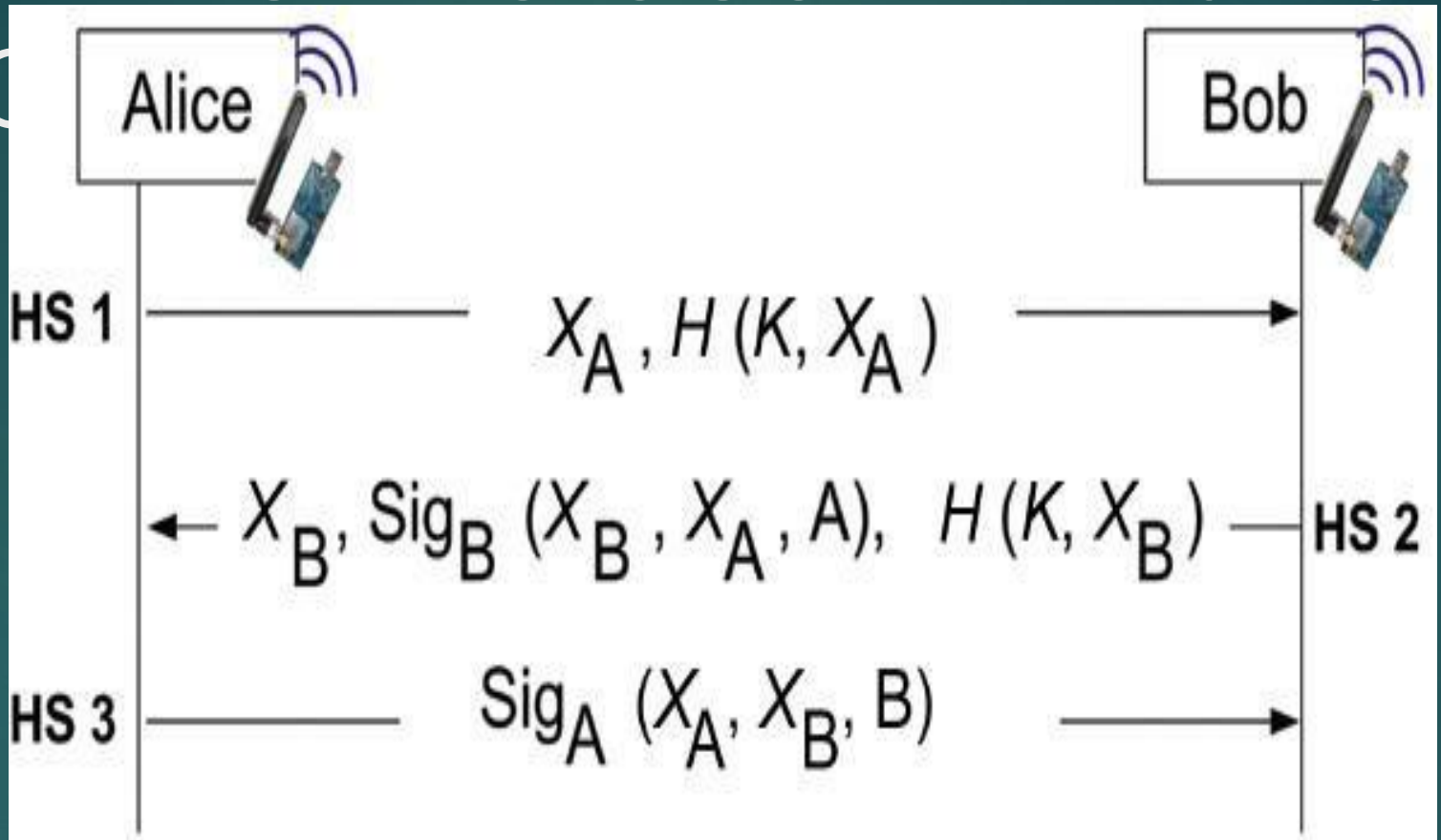
TINYTO PROTOCOL - BCK WITH PRESHARED KEYS FOR TINYTO

- PSK is suitable to provide authentication, while requiring only a small amount of computational power and memory.
- Thus, it is selected for TinyTO to verify the identity of an entity.
- The distribution of PSKs is simple in the context of WSN devices.
- Adding a unique PSK to the programming procedure introduces practically no overhead because nodes need to be programmed before deployment in any case, and the key generation and management can be moved to the software programming the nodes.

TINYTO PROTOCOL - BCK WITH PRESHARED KEYS FOR TINYTO (CONTINUED)

- Compared to approaches where every node is equipped with a set of keys for encryption between peers before deployment, TinyTO assumes that every node has only one PSK, solely for authentication toward the gateway.
- The developed handshake for TinyTO compares to BCK, with preshared keys that form master keys for an initial authentication between the node and the gateway.
- The following diagram illustrates the resulting handshake, where Alice and Bob can represent any one of the following device types in the WSN:

EXTENDED BCK PROTOCOL WITH PSK FOR TINYTO



TINYTO PROTOCOL - BCK WITH PRESHARED KEYS FOR TINYTO (CONTINUED)

- **A collector** is a device-collecting sensor, which forward them directly to the next device within communication range.
- **An aggregator** works with the data received, either as aggregating several messages into one large message, or preprocessing data (e.g., average, max, min calculation of values) before forwarding them to the next device within communication range.
- **The gateway** defines the gate to the world, connecting the WSN to other applications in the IoT domain.

TINYTO PROTOCOL - BCK WITH PRESHARED KEYS FOR TINYTO (CONTINUED)

- Under the assumption that only the two parties under investigation have knowledge of the PSK, each party can be assured that indeed the other communication party uses this PSK.
- It is vital not to transmit the PSK in plaintext during the authentication, in order to keep the PSK a secret between the two parties.
- Otherwise any attacker who picks up that message containing the PSK can use the PSK.

Thus, it must be avoided to send any form of information that can

(1) be used to retrieve the PSK or

(2)be replayed to achieve authentication for any other entity. Traditionally, those two goals are met by transmitting a cryptographic hash digest of the PSK together with a cryptographic nonce

TinyTO Protocol - BCK with Preshared Keys for TinyTO (Continued)

- Including a different nonce in every message makes it impossible to reuse an authentication message (e.g., replay attack).
- In comparison, TinyTO desires to couple a unique public key with the PSK (and, thus, the identity), which may be replayed several times, but never for another public key, which makes it very hard to recalculate the PSK by an attacker.
- Hence, it is possible to use the public key instead of a random nonce and to create a hash from the PSK and from this public key, that is, $H(K, X_A)$.
- This ensures Bob that X_A is indeed Alice's public key.
- A cryptographic hash function is infeasible to be reverted, even with a partially known input (the public key is obviously publicly known).
- But the PSK is not recoverable.

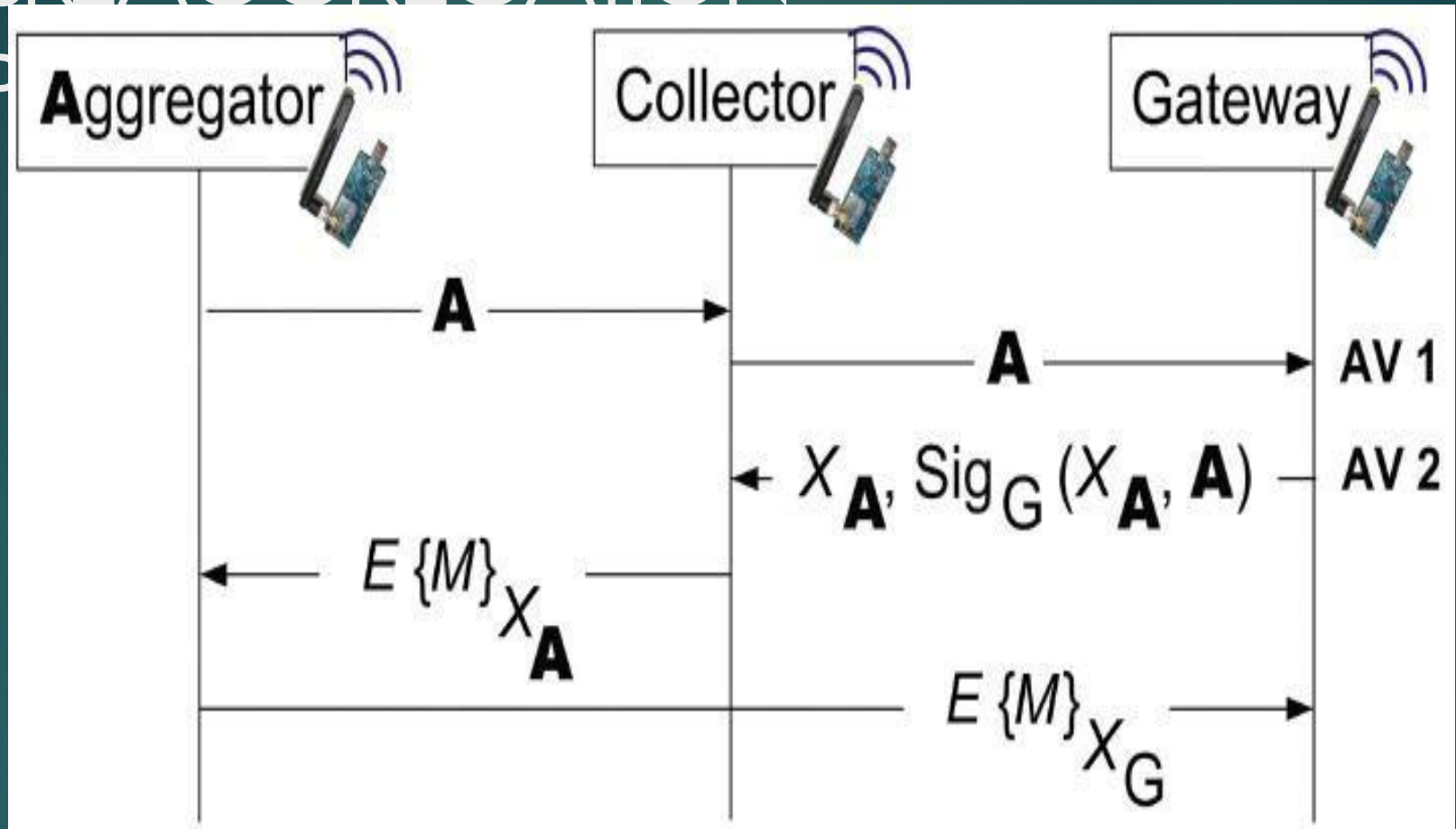
TINYTO PROTOCOL - BCK WITH PRESHARED KEYS FOR TINYTO (CONTINUED)

- At the same time, a spoofed hash digest for a different public key can be produced only with the knowledge of the PSK.
- To provide mutual authentication in the TinyTO protocol, those digests must be computed from both parties, with their respective public keys.
- To avoid transmission overhead, these digests can be included in the first and second handshake messages (HS1 and HS2 in the diagram) in order to avoid any transmission overhead by additional messages.
- In accordance with the requirements for TinyTO, this approach determines the two-way authentication protocol, which includes, as the key agreement, delivering a direct and explicit key authentication.

TINYTO PROTOCOL - BCK WITH PRESHARED KEYS FOR TINYTO (CONTINUED)

- Messages do not include timestamps, they are completely asymmetrical, and they cannot be used for a replay or for reflection attacks.
- Appropriate encryption techniques (e.g., RSA or AES) of subsequent messages are required to guarantee the forward secrecy beyond the handshake.
- Two flexible roles—collector and aggregator—are possible for a node.
- The gateway, in contrast, is unique and static.
- Collectors and aggregators use TinyTO to establish a secure communication channel with the gateway. .
- Aggregators introduce additional performance overhead to TinyTO and the WSN, because the handshake is more complicated.

SECURE AGGREGATION SUPP



X_A : Aggregator's public key

Sig_G : Gateway's signature

A: Identity Aggregator

X_G : Gateway's public key

$E\{M\}_P$: Encryption with P

AV: Aggregator verification message

TINYTO PROTOCOL - BCK WITH PRESHARED KEYS FOR TINYTO (CONTINUED)

- Also, the collectors need to switch the destination of their data stream from the gateway to the aggregator, which, in turn, needs to process the information.
- Therefore, the aggregator sends a presence announcement via a broadcast to collectors that redirect their streams upon receipt.
- Four conceptual steps are required for an aggregator introduction, if no authentication is required.
- The TinyDTLS solution inspired the development of TinyTO.

TINYTO PROTOCOL - BCK WITH PRESHARED KEYS FOR TINYTO (CONTINUED)

- Four steps that must be taken in order to establish a two-way authentication, and those steps must be slightly adapted for the proposed TinyTO solution in the following manner:

(1)Collectors complete their TinyTO handshake with the gateway (Fig. 13.2) and transmit data over a secure channel.

(2) In turn, the aggregator can be activated, contacting the gateway immediately, and executing the TinyTO handshake, resulting in a secure channel.

(3)The aggregator broadcasts its presence to collectors in range that are programmed to wait for such a specific message type (e.g., simple echo request, counter, or nonce). The aggregator's public key is included in the broadcast message to avoid additional message exchanges.

(4)Finally, collectors redirect their streams, encrypted with the aggregator's public key ($E\{M\}_{XA}$) , to the aggregator. The aggregator decrypts incoming streams, processes messages, encrypts the results again, and sends the new message securely to the gateway ($E\{M\}_{XG}$) or to the next hop.

TINYTO PROTOCOL - BCK WITH PRESHARED KEYS FOR TINYTO (CONTINUED)

- Aggregator integration provides an encryption of messages among all parties, and, therefore, a protection against eavesdropping between collectors and the aggregator as well as the aggregator and the gateway, it entails one important drawback.
- Collectors have executed the complete TinyTO handshake with the gateway, resulting in a twoway authentication of both parties.
- However, in the aforementioned steps (3) and (4), collectors sacrifice all assertions about identities, if they blindly react to the aggregator's broadcasts.
- Attackers just need to broadcast an aggregator announcement to reach access to data streams from every collector in range that is conveniently encrypted with the attacker's own public key.

TINYTO PROTOCOL - BCK WITH PRESHARED KEYS FOR TINYTO (CONTINUED)

- Because such a situation breaks the entire security concept, collectors are requested to establish a new secure channel to aggregators, which fulfill TinyTO's design principles without blindly trusting aggregator broadcasts.
- Consequently, the authentication needs to be extended by an authorization: collectors need the confirmation that a broadcasting aggregator is a valid aggregator and not an intruder who is trying to access confidential information.

TINYTO PROTOCOL - BCK WITH PRESHARED KEYS FOR TINYTO (CONTINUED)

- Assuming that collectors or aggregators communicate only with the gateway, the request for secure communication is implicitly covered by the exchange of preprogrammed PSKs.
- Intuitively, it is possible to preprogram aggregators and collectors with pairwise PSKs in the same way, followed by a handshake execution, including the authentication and the key exchange.
- But this workaround does not fulfill the flexibility requirement for TinyTO: in this case, aggregators can only aggregate data streams from predefined collectors and will further need to hold n + 1 PSKs for n collectors.

TINYTO PROTOCOL - BCK WITH PRESHARED KEYS FOR TINYTO (CONTINUED)

- A more flexible and less resource-demanding solution is to lever the already fully authenticated and secured channels between both aggregator and collector, and the gateway.
- Upon receipt of an aggregator announcement, collectors need to check only with the fully authenticated gateway, whether the broadcast sender is an authorized aggregator.
- If so, the gateway can reply with the aggregator's public key, signed by his own trusted private key.
- This is similar to a PKI, where the gateway takes the role of the certificate authority as a trusted third party.
- For TinyTO it is assumed that all parties have completed previously and successfully their handshakes with the gateway.

TINYTO PROTOCOL - BCK WITH PRESHARED KEYS FOR TINYTO (CONTINUED)

- The signature is used in order to verify the mapping between the aggregator's public key and its identity, which makes spoofing attacks on the public key impossible.
- This stands in contrast to the previously exchanged authentication messages, where the identity of the receiving party must be included instead of the key owner's identity because the channel between gateway and collector is already authenticated.

TINYTO PROTOCOL - BCK WITH PRESHARED KEYS FOR TINYTO (CONTINUED)

- The aggregator's identity is not encrypted between the collector and the gateway, allowing for spoofing attacks on the identity because expensive computations would be required for this additional encryption.
- Hence, it is substituted with the identity in the signature from the next message, computed by the more powerful gateway.
- The gateway might reply with the public key for a spoofed identity, but it is detectable by the collector due to the invalid signature, resulting in the process' abortion.

TINYTO PROTOCOL - HANDSHAKE IMPLEMENTATION

- From now on, it is assumed that TinyTO is implemented in TinyOS, where different components are “wired” to each other and use the offered set of functionality.
- Thus, the TinyTO handshake is implemented in the component HandshakeHandler, which is exclusively responsible for handshake-message handling, including message composition and reply handling.
- HandshakeHandler is wired to components called in for cryptographic functions.
- The three TinyTO handshake messages HS1 to HS3 are implemented in a similar manner.

TINYTO PROTOCOL - HANDSHAKE IMPLEMENTATION (CONTINUED)

- Following shows a model of the structure of handshake message HS2, where ***nx_uint8_t*** stands for the network-serializable unsigned integer type.

```
// Handshake Message 2 (HS2):
typedef nx_struct to_hsm2_t{
    nx_uint8_t msgType;
    nx_uint8_t hsType; // 0x02
    nx_uint8_t public_x[24];
    nx_uint8_t public_y[24];
    nx_uint8_t digest[20];
    //variable length, ANS.1 encoded
    nx_uint8_t signature[];
} to_hsm2_t;
```

TINYTO PROTOCOL - HANDSHAKE IMPLEMENTATION (CONTINUED)

- The msgType field is used to distinguish between handshake messages and other types of control messages that are sent via the same port.
- hsType identifies different handshake messages HS1, HS2, and HS3.
- Furthermore, public ECC keys are broken down into x- and y-coordinates for easy handling on the node's side.
- Elliptic Curve Digital Signature Algorithm (ECDSA) signatures are defined as integer key pairs, written as (r, s), and, therefore, difficult to include in a fixed-length packet, because the bit length of the hexadecimal representation of large integers may vary.

TINYTO PROTOCOL - HANDSHAKE IMPLEMENTATION (CONTINUED)

- Actual length detection and signature processing is not complicated, but the TinyECC library is very selective on input parameters and requires accurate length information before a signature validation.
- As a consequence, the signature in HS2 is encoded in the Abstract Syntax Notation One (ANS.1), inherently including length information for r and s.
- The reply can be sent with two plain fixed-length arrays for the signatures, because the powerful gateway can handle the necessary computations to strip any padding and to encode the signature correctly.

TINYTO PROTOCOL - HANDSHAKE IMPLEMENTATION (CONTINUED)

- Thus, the computation time on the node is minimized.
- Similar to the three handshake messages, the two necessary authentication messages of the aggregator to the collectors are implemented.
- The aggregator's verification message 1 (AV1), sent from the collector to the gateway upon receipt of the aggregator announcement, includes msgType, hsType, and the aggregator address.
- Additionally, AV2, sent from the gateway to the collector, includes the public agg x, public agg y, and a signature from the previously authenticated gateway, confirming the aggregator's public key with the given address.

EVALUATION

- TinyTO is evaluated for memory and energy consumption, and its ability to fit those requirements of Class 1 devices.
- Furthermore, the performance is analyzed and the security level is compared to related work.
- In order to show the feasibility of TinyTO for Class 1 devices, TelosB nodes are used exclusively in a testbed for evaluation purposes.
- TinyOS is the operating system chosen for the current implementation.
- A simplified setup for the testbed, and the following situation is assumed: In Rooms A, C, and D the light is turned off and the room temperature is low.
- Sensors in Room B report lights being switched on and the microphone shows a high noise-level.

EVALUATION (CONTINUED)

- All data collected is sent either directly (see Room B—use-case 1) or via multiple hops (see Rooms A, C, and D—use-case 2) to the gateway.
- Software on the gateway can analyze the data collected and sends corresponding information to other systems (e.g., climate control or room-booking system).
- The analysis result for Room B indicates that a conference takes place, and, thus, the climate control is activated and an entry is made in the room's calendar that Room B is currently occupied.

EVALUATION (CONTINUED)

- For Rooms A, C, and D the internal room-lock system is informed that these rooms are empty and shall be locked automatically.
- In this example, the addressing of inconspicuous data can lead to the claim that confidential information collected allows for conclusions about room occupancy.
- Additionally, this introduces security risks in the application, as in the case of room information being retrieved by eavesdropping due to missing security in the communication of those sensors, an attacker can become aware of this situation and could plan for a burglary.

EVALUATION - MEMORY CONSUMPTION (CONTINUED)

- TinyTO's main challenge was to require only a small part of the resources available, allowing applications (e.g., TinyIPFIX) to run, in addition to the security solution.
- The memory consumption of applications can be determined for TinyOS directly from the compiling tool, because resources are already known at the time of compilation.
- Deactivation of components (e.g., RPL or TinyECC optimizations) via the compiling tool or by removing components (e.g., TinyTO component or HandshakeHandler) in the code allows for a recording of the individual memory consumption of TinyTO components, shown in following table.

EVALUATION - MEMORY CONSUMPTION (CONTINUED)

Table 13.2 Memory Consumption of Components [53]

Operation	Aggregator		Collector	
	ROM (Byte)	RAM (Byte)	ROM (Byte)	RAM (Byte)
Handshake	1,636	602	1,138	612
Cryptography	11,406	406	9,378	406
TinyTO total	**13,042**	**1,018**	**10,516**	**1,018**
Data handling [12]	26,904	6,964	31,144	5,478
RPL [17,28]	6,270	498	6,228	1,498
Total	**46,216**	**8,470**	**48,114**	**7,994**

EVALUATION - MEMORY CONSUMPTION (CONTINUED)

- The small memory difference between conceptually identical components (handshake and cryptography) in the collector's and aggregator's implementation originates from marginally different use-cases.
- For example, collectors only need to store one message at a time, but aggregators need additional memory to buffer data before that aggregation can be performed.
- Because the memory is statically reserved, the detailed memory footprint depends on the DOA.
- Furthermore, collectors only need code for an encryption, whereas aggregators need code for both the decryption and the encryption.

EVALUATION - MEMORY CONSUMPTION (CONTINUED)

- In comparison to aggregators, collectors periodically read their sensor values, which requires additional memory.
- Overall, this leads to a ROM consumption of 37,590 Byte purely for the collector and 33,174 Byte for aggregator applications (including data handling and RPL), and yields slightly more than 4 kByte of additional free memory for aggregators, which can be used to enable ECC optimizations.
- The following table shows that optimizations have a direct impact on memory consumption and whether they are used (indicated by X) on TinyTO.

EVALUATION - MEMORY CONSUMPTION (CONTINUED)

Table 13.3 Memory Consumption of TinyECC Optimizations [53]

Operation	ROM (Byte)	RAM (Byte)	Aggregator	Collector
Barrett reduction	780	114	—	—
Hybrid multiplication	12	0	X	—
Hybrid square	114	0	X	X
Secpt optimization	414	0	X	—
Projective coordinates	850	0	X	X
Sliding window	206	2350	—	—

EVALUATION - RUNTIME PERFORMANCE

- In terms of performance the slow microcontroller and limited memory have a high impact on all results.
- A message size of 116 Byte was assumed, because it is typically used for the application TinyIPFIX supported.
- Further collectors do not need to decrypt data.
- For measurements performed, a timer was read before and after an operation was executed.
- The resolution was at 65.53 ms, allowing accurate measurements within the scope of several seconds.
- The following table shows the execution times for various cryptographic operations in TinyTO's aggregators and collectors.

EVALUATION - RUNTIME PERFORMANCE (CONTINUED)

Table 13.4 Execution Times for ECC Operations [53]

ECC Operation	Aggregator (s)	Collector (s)
EC Key Generation	4.77	8.77
SHA-1	≤0.10	≤0.10
ECDSA Sign	5.14	9.28
ECDSA Verify	10.20	18.51
ECIES Encrypt	5.98	9.41
ECIES Decrypt	4.96	—

EVALUATION - RUNTIME PERFORMANCE (CONTINUED)

- Aggregators are generally almost twice as fast as collectors, which is mainly due to more activated ECC optimizations.
- A performance evaluation for TelosB, showing the speed for ECC operations, when all optimizations are used : ECDSA signing takes only about 1.6 s (TinyTO aggregator needs about 5.14 s; TinyTO collector, about 9.28 s), and verification, about 2 s(TinyTO aggregator needs about 10.20 s; TinyTO collector, about 18.51 s).
- This is much faster than TinyTO, and proves the performance limitations due to the restricted memory of chosen hardware (here, TelosB).

EVALUATION - RUNTIME PERFORMANCE (CONTINUED)

- The following table shows the execution times of composite operations, including transmission times in both directions and response-calculation times.
- The fact that the gateway performs faster than nodes has no influence on the overall result.
- The value for a message aggregation with DOA = 2 is calculated based on two decryption operations and one encryption operation on the aggregator.
- Based on those results, it is possible to determine the minimal interval t, where collectors send their encrypted messages and aggregators can still catch up with incoming packets.
- Once the handshake is executed, collectors send data after t = 9.41 s, which is the minimal time needed for encryption, even if data is immediately available.

EVALUATION - RUNTIME PERFORMANCE (CONTINUED)

Table 13.5 Energy Consumption of Composite Operations [53]

Operation	Time (s)	Energy (mJ)
Handshake Aggregator	20.14	85.90
Handshake Collector	36.59	154.99
Aggregator verification	18.52	78.58
Message aggregation (*DOA* = 2)	15.90	68.28

EVALUATION - RUNTIME PERFORMANCE (CONTINUED)

- Assuming sufficient memory on an aggregator's device is available for caching packets of degree DOA—plus an aggregate computation—an aggregator requires t = DOA * 4.96 s+ 5.89 sto decrypt incoming messages of the degree DOA and to encrypt the aggregate.
- For DOA = 2, the aggregator needs t = 15.81 s.
- This equation does hold for a small degree of aggregation and small networks (e.g., 20 nodes).
- The formula for t must be adapted with the required time for ECIES encryption and decryption, if the aggregator: (1) is more powerful, (2) can support a greater degree of aggregation, and (3) can perform faster operations, and if the network becomes larger.

EVALUATION - ENERGY CONSUMPTION

- WSN devices are usually battery-powered and depend on the deployment, which makes an exchange not very easy.
- Hence, TinyTO must be energy efficient to avoid a fast battery depletion.
- TelosB nodes in the testbed are powered by two off-the-shelf batteries, each with a capacity of 2000 mAh and voltage of 1.5 V, in total delivering U = 3.0 V.
- Wireless data transmissions and computations in the microcontroller show the largest impact on energy consumption.
- Auxiliary components, such as LEDs or serial connectors, are not taken into consideration, as they are typically deactivated during a final deployment.

EVALUATION - ENERGY CONSUMPTION (CONTINUED)

- TelosB nodes are equipped with a CC2420 RF chip on the IEEE 802.15.4 2.4 GHz band, which has a maximum output power of −25 to 0 dBm for data transmissions.
- Assuming 0 dBm, the current draw for sending (Tx) is at I_{Tx} = 17.4 mA and at I_{Rx} = 19.7 mA for receiving (Rx).
- The theoretical transmission rate of the CC2420 is at 250 kbps, but some practical measurements are as low as 180 kbps.
- For the purposes of the calculation being as close to reality as possible, it can be assumed that the full transmission rate R is never actually reached, and R = 220 ± 20 kbps.

EVALUATION - ENERGY CONSUMPTION (CONTINUED)

- Knowing the transmission rate and implementation details of messages allows for the calculation of the transmission time for each message and the total energy consumption ERas follows: ERdepends on the message size Sfor a given voltage, a current draw, and a transmission rate, resulting in ER(S)= U* I * S.
- In particular, this concerns the data handling with TinyIPFIX, the three handshake messages (HS1 to HS3), and the aggregator verification (AV1 and AV2).
- In comparison to ECCoperations, TinyIPFIX operations are almost instantaneous and negligible in the light of the overall energy consumption of TinyTO.

EVALUATION - ENERGY CONSUMPTION (CONTINUED)

- Furthermore, messages as outlines in the handshake design consider only the size of individual data fields in an unencrypted message.
- In reality, the packets transmitted are much larger than supported by IEEE 802.15.4 on the MAC layer (102 Byte out of the total frame size of 127 Byte).
- Hence, packet fragmentation support for TinyTO is essential.
- Because 12 Byte are used by TinyOS and the cyclic redundancy check (CRC) for error detection is added, 90 Byte remain for the actual payload in every message on the MAC layer.

EVALUATION - ENERGY CONSUMPTION (CONTINUED)

- TinyTO handshake messages are not transmitted as plaintext, but in larger Elliptic Curve Integrated Encryption Scheme (ECIES) cipher texts, requiring 69 Byte [1 Byte to indicate the point compression type, 24 Byte for each Elliptic Curve (EC) point component, and 20 Byte for the message authentication more than the pure message size].
- For example, an HS1 message has a size of 70 Byte and a size of 139 Byte after encryption with ECIES.
- Given the maximum payload size of 90 Byte, HS1 is fragmented to fit into MAC layer packets.
- Every fragment requires additional headers and other fields (e.g., fragment number and header field indicating fragmentation).

EVALUATION - ENERGY CONSUMPTION (CONTINUED)

- Thus, the effective data size DS, which is transmitted to convey a payload of size ps = 139 Byte, is calculated as DS = ps + ⌈ps/90 Byte⌉ * 37 Byte = 213 Byte.
- The following table shows the results of these considerations for different message types and the energy consumption for their transmissions.
- It can be stated that the energy consumption for HS2 is the highest, with 6.34 mJ, compared to HS1 and HS3, due to its enormous message size of 189 Byte. If a message size is approximately 100 Byte then the energy- consumption levels are around 0.3 mJ.

EVALUATION - ENERGY CONSUMPTION (CONTINUED)

Table 13.6 Energy Consumption of the Radio Transmission [53]

Message	*ps* (Byte)	*DS* (Byte)	Time (ms)	Energy (mJ)
HS 1 (Tx)	139	223	8.11	0.42
HS 2 (Tx)	189	300	10.91	0.64
HS 3 (Tx)	114	203	7.38	0.38
AV 1 (Tx)	82	171	6.22	0.32
AV 2 (Tx)	168	242	8.80	0.52

EVALUATION - ENERGY CONSUMPTION (CONTINUED)

- Similarly to energy calculations for radio transmissions, the energy consumption of the microcontroller (MSP430F1611 16-bit Ultra-Low-Power Micro-Controller Unit (MCU) from Texas Instruments [57]) for different cryptographic operations can be calculated.
- The current draw in active mode (i.e., only MCU and no radio transmissions) is given as 1.8 mA.
- However, experimental measurements show that the relevant difference between idle and a fully utilized MCU is only at $I_{AM} = 1.4$ mA.
- The formula used to calculate the energy consumption E_{MCU} of the MCU, depending on the computation time t subsequently, is $E_{MCU}(t) = U * I_{AM} * t$.

EVALUATION - ENERGY CONSUMPTION (CONTINUED)

- As shown in the following table, the energy consumption differs for the aggregator and the collector, due to different activations of these ECC optimizations.
- Given the cost of a radio transmission and the computation of single cryptographic operations, the energy consumption for the entire handshake, and similarly for more complex sequences of operations, can be calculated.
- According to the design, the handshake requires six operations: ECKey Generation, sending of HS1, reception of HS2, ECDSA signature verification, ECDSA signature signing, and sending of HS3.

EVALUATION - ENERGY CONSUMPTION (CONTINUED)

Table 13.7 Energy Consumption of Cryptographic Operations [53]

Operation	Aggregator		Collector	
	Time (s)	Energy (mJ)	Time (s)	Energy (mJ)
EC Key Generation	4.77	20.03	8.77	36.83
ECDSA Sign	5.14	21.59	9.28	38.98
ECDSA Verify	10.20	42.84	18.51	77.74
ECIES Encrypt	5.98	25.12	9.41	39.52
ECIES Decrypt	4.96	20.83	—	—

EVALUATION - ENERGY CONSUMPTION (CONTINUED)

- Similarly, the verification of an aggregator needs three operations: sending of the aggregator's identity in AV1, the reception of the signed message containing the public key in AV2, and an ECDSA signature verification.
- Aggregation with DOA = 2 requires a combination of the reception of two data packets, including data collected, two times an ECIES decryption, the ECIES encryption, and the sending of one aggregated data packet.
- The battery-powered TelosB requires a minimal voltage of 1.8 V, meaning a battery cannot be depleted to an energy level below 60% of the original charge, otherwise the voltage will drop below that threshold.

EVALUATION - ENERGY CONSUMPTION (CONTINUED)

- Thus, it can be calculated that 12.96 kJ are available in one set of batteries.
- Measurements show that TelosB nodes draw on average of 70.7 mA, while remaining idle when no sleep modes for MCU and radio are activated.
- Thus, the expected runtime without any application is about 12.96 kJ = 61,103 s, or roughly 16 h and 58 min for one 3 V * 70.7 mA set of batteries.
- If collectors are programmed to collect, encrypt, and send data in the format proposed by, then every interval t, the impact on the runtime of collectors, and their aggregators can be calculated accordingly.

EVALUATION - ENERGY CONSUMPTION (CONTINUED)

- Assuming every tenth transmission contains the TinyIPFIX template (only metainformation) instead of a TinyIPFIX record (data values) and an aggregation with DOA = 2 is performed, the same batteries will last for 16 h and 53 min in aggregators (which compares to a reduction of 0.5% or 5 min) and 16 h and 55 min in collectors (reduction of 0.3% or 3 min), given $t = 1\ min^{-1}$.