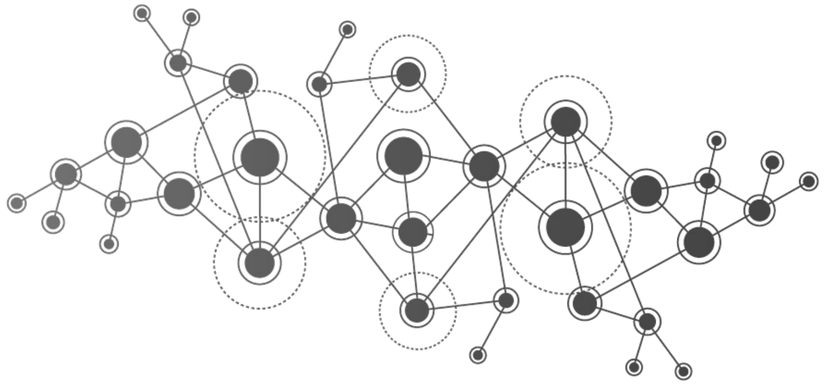
UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA



Intelligent Internet of Things

IoT & Security

Prof. Marco Picone

A.A 2023/2024

IoT & Security

- Introduction
- IoT & Security Threats
- Security Services
- IPSec
- TLS
- DTLS
- Security & Energy Consumption
- Security & MQTT

2023 - Marco Picone, Ph.D. (marco.picone@unimore.it)

Introduction

- Security is a very important aspect in current Internet and, probably more, in IoT scenarios
- Securing IoT communications is particularly complicated by the nature of IoT Smart Objects and Devices:
  - limited computational power
  - limited memory capabilities
  - limited computation resources
  - often battery-powered
  - often without a user interface

Threats

- Threats related to the physical nature of smart objects:
  - **Cloning of smart things by untrusted manufacturers and/or Malicious Substitution of things during the installation**

    e.g., with lower-quality things

    untrusted manufacturer may change functionality of the cloned thing, resulting in degraded functionality
  - **Firmware Replacement attack**

    e.g. when a thing is in operation or maintenance phase

    its firmware or software may be updated to allow new functionalities or features
  - **Extraction of security parameters from physically unprotected devices**

    if a group key is used and compromised, the entire network may be compromised

Threats

- Some threats that could compromise an individual thing, or network as a whole:
  - **Eavesdropping attack**

    if the communication channel is not adequately protected

    e.g. if operational keying materials, security parameters, or configuration settings, are exchanged in clear using a wireless medium, or in case of session key compromise due to a long period of usage without key renewal
  - **Man-in-the-middle attacks**

    particularly during key establishment and/or device authentication procedure

    device authentication/authorization may be not automated and need support of a human decision process, since things usually do not have a priori knowledge about each other

Threats

- and:
  - **Routing attack**

    routing information in IoT can be spoofed, altered, or replayed, in order to create routing loops, attract/repel network traffic, extend/ shorten source routes, etc.

    relevant routing attacks may also include:

    - Sinkhole attack (or blackhole attack): an attacker declares himself to have a high-quality route/path to the base station, thus allowing him to do anything to all packets passing through it
    - Selective forwarding: an attacker may selectively forward pkts or simply drop a packet
    - Sybil attack: an attacker presents multiple identities to other things

Threats

- and:
  - **Privacy threat**

    an attacker can gather and track thing's information, deducing behavioural patterns

    such information can subsequently be sold to interested parties
  - **Denial-of-Service attack**

    typically, things have tight memory and limited computation, they are thus vulnerable to resource exhaustion attack

    attackers can continuously send requests to be processed by specific things so as to deplete their resources

    this is especially dangerous in the IoT since an attacker might be located in the backend and target resource-constrained devices in an LLN

    additionally, DoS attack can be launched by physically jamming the communication channel, thus breaking down the device to device communication channel

    network availability can also be disrupted by flooding the network with a large number of packets

Countermeasures

- Countermeasures against threats related to the physical nature of smart objects:
  - safe supplying and installation measures, such as avoiding untrusted manufacturers and installers
  - protect smart objects in safe places
- Countermeasures against threats related to constrained networked scenario:
  - secure communication protocols and cryptographic algorithms

Security Services

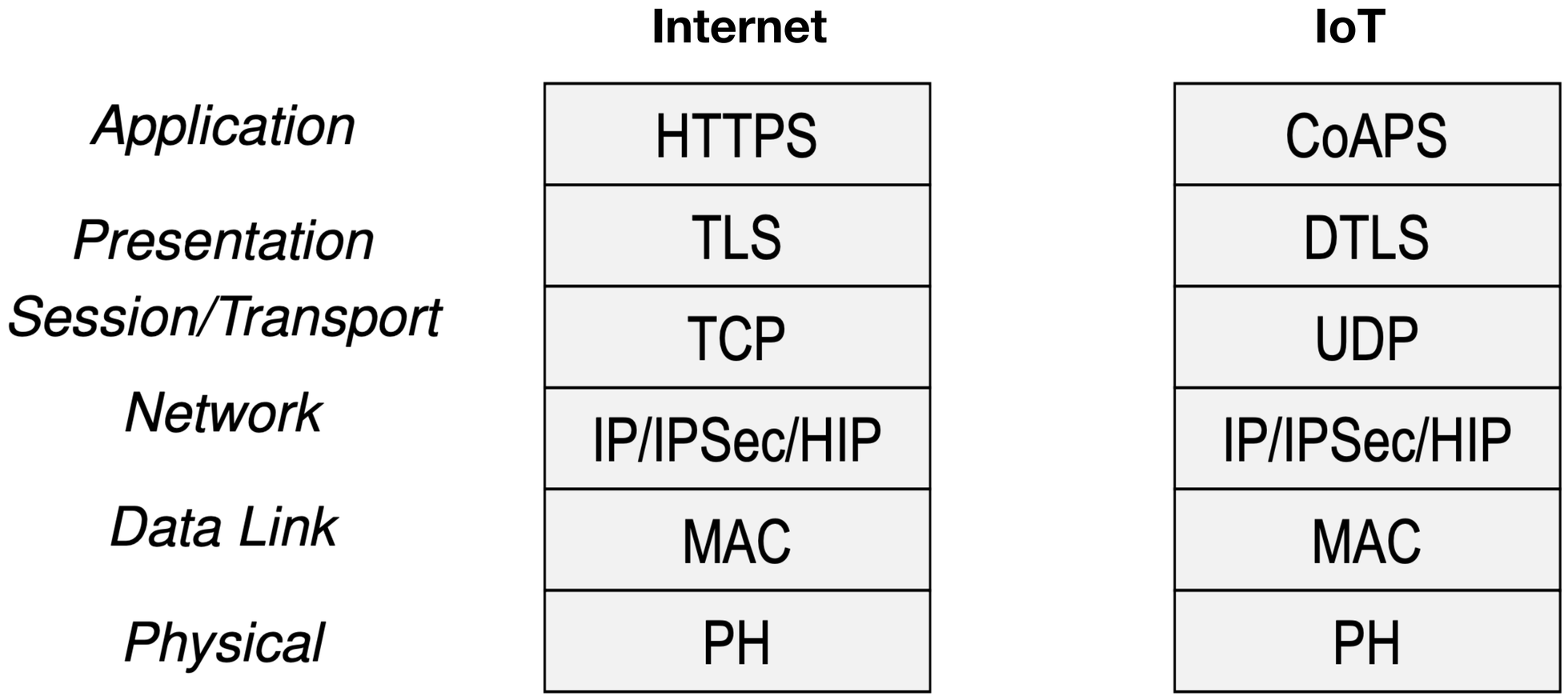
- While it is possible to cope with physical nature-related issues by adopting safe supplying and installation measures
- The other security threats can be tackled by adopting means, such as secure communication protocols and cryptographic algorithms, in order to enforce the following basic security services:
  - peer authentication/authorization
  - data authentication/integrity
  - confidentiality
  - non-repudiation
  - availability
  - anonymity, and/or pseudonymity

Security Services

- These security services can be implemented by a combination of:
  - cryptographic mechanisms:
    - symmetric block ciphers
    - hash functions
    - asymmetric cryptographic
    - signature algorithms
  - non-cryptographic mechanisms, which implement authorization and other security policy enforcement aspects
- For each of the cryptographic mechanisms, **a solid key management infrastructure** is fundamental to handling the required cryptographic keys

IP-based IoT stack with security protocols

- Example of IP-based IoT protocol stack with security enforcement vs. classical Internet protocol stack



IP-based IoT stack

- Although data authentication, integrity, and confidentiality can be provided at lower layers, such as PH or MAC (e.g., in IEEE 802.15.4 systems), no end-to-end security can be guaranteed without a high level of trust on intermediate nodes
  - however, due to the highly dynamic nature of wireless multi-hop communications expected to be used to form the routing path between remote end nodes, this kind of security (hop-by-hop) is not, in general, sufficient
  - for such reason, security mechanisms at network, transport, or application level should be considered instead of (or in addition to) PHY and MAC level mechanisms

Selecting security protocols

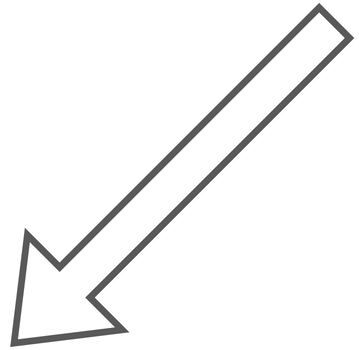
- **Avoiding security service duplication**
  - it has impact on the computation and transmission performance
  - e.g., end-to-end security can be guaranteed through IPSec at the network layer or TLS/DTLS at the transport layer
- **Interoperability**
  - many of the Internet security protocols are independent by the used security algorithms – used algorithms and parameters sometime can be negotiated between the communication parties
  - in order to guarantee full interoperability, and to reduce the number of implemented cipher suites, it is important to define a set of mandatory algorithms (targeted for constrained environments) that all objects must implement for minimal support
  - however it is not easy to agree on a set of common protocols and algorithms that can be

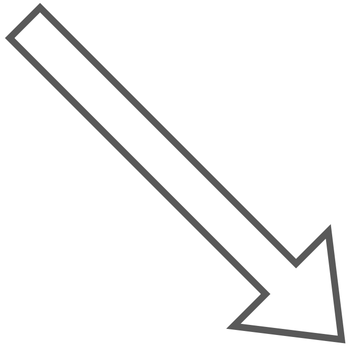
Internet Protocol Security (IPSec)

- At network layer, an IoT node can secure data exchange in a standard way by using the Internet Protocol Security (IPSec)
- IPSec (RFC 4301) is the Security Architecture for the IP protocol
  - It was originally developed for IPv6 but found widespread deployment, first, as an extension in IPv4, into which it was back- engineered
  - was an integral part of the base IPv6 protocol suite, but has been made optional
- IPSec can provide
  - confidentiality
  - integrity
  - data-origin authentication
  - protection against replay attacks

IPSec Security Services

Such security services are implemented by two IPSec security protocols





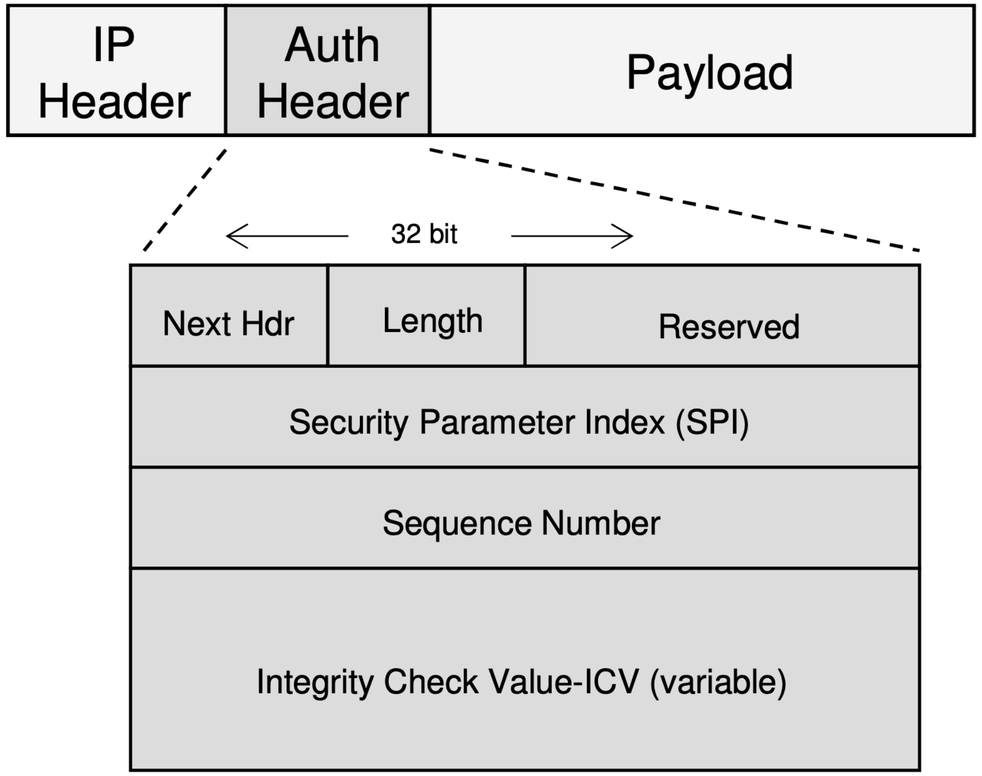
Authentication Header (AH):

provides data origin authentication, integrity, and optionally anti-replay.

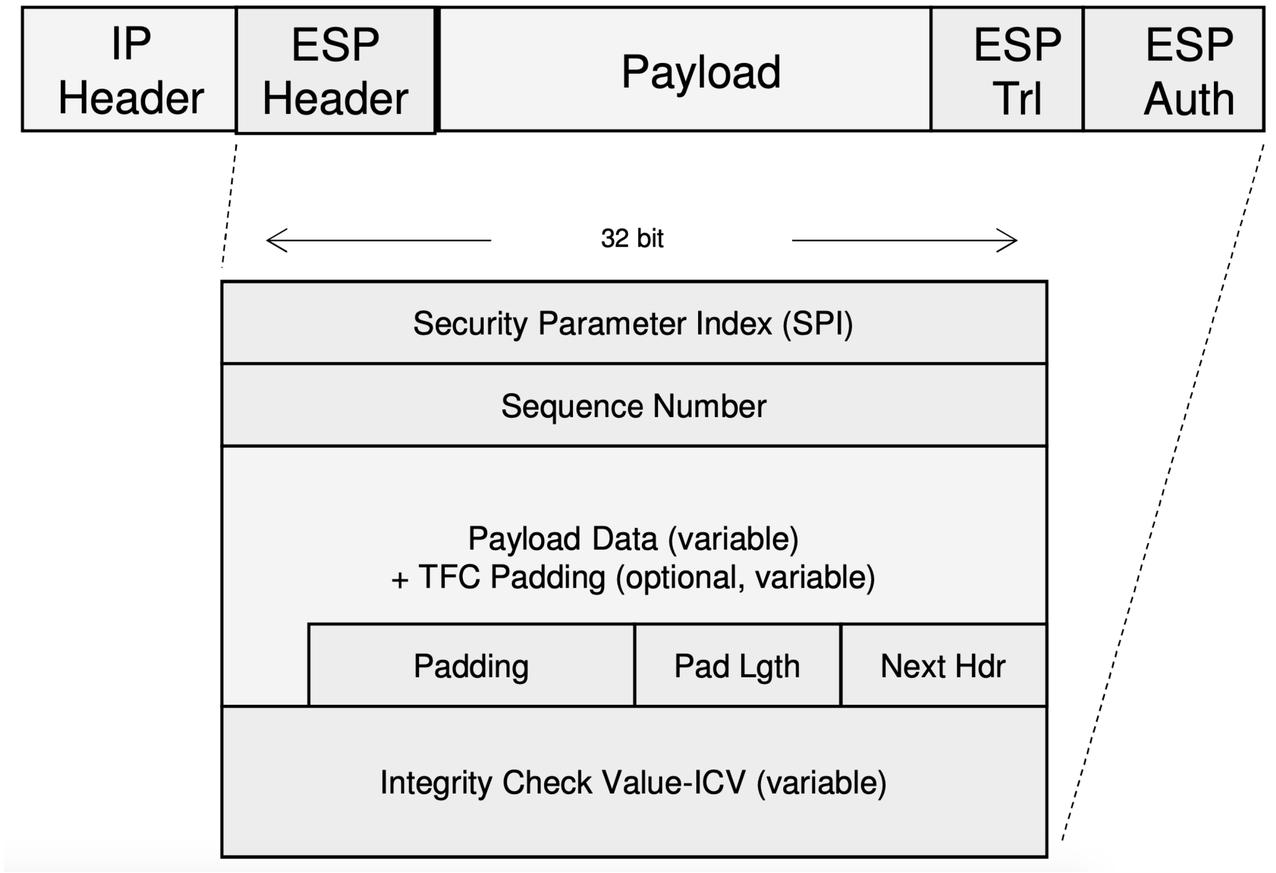
Encapsulated Security Payload (ESP):

provides confidentiality, data origin authentication, integrity, and anti-replay capabilities

IPSec AH and ESP packet formats



IPSec AH packet format



IPSec ESP packet format

IPSec

- IPSec AH and ESP define only the way IPSec control information are encapsulated
  - the effective security algorithms are specified separately algorithms can be selected amongst a set of available cipher suites
  - this makes IPSec usable also in the presence of very resource- constrained devices, if a proper algorithm that guarantees both usability and sufficient security level is selected
- The keying material and the selected cryptographic algorithms used by IPSec for securing a communication (Security Association)
  - **can be pre-configured** (specifying a priori a pre-shared key, hash function and encryption algorithm)
  - **can be dynamically negotiated** by the **IKE (Internet Key Exchange)** protocol. Unfortunately, as the IKE protocol uses asymmetric cryptography, which is **computationally heavy** for very small devices and other extensions should be considered using lighter solution targeting the IoT ecosystem

IPSec - Problems

- Some problems related to the implementation of IPSec in constrained IoT nodes are:
  - **data overhead**

    introduced by the extra header encapsulation of IPSec AH and/or ESP

    this can be limited by:

    - using only one protocol between AH and ESP
    - selecting the “transport mode” encapsulation when possible
    - implementing header compression techniques, similarly to what is done in 6LoWPAN for the IP header
  - **configuration, and practical implementation aspects**

    IPSec works at network layer is often designed for VPNs, thus making it difficult for them to be dynamically configurable and controlled by an application

Transport Layer Security

- In the standard Internet, data exchange between applications can be secured at transport layer through:
  - **Transport Layer Security (TLS)**

    widest used secure protocol, running on top of the TCP providing to the application layer the same connection and stream-oriented interface of TCP
  - **Datagram Transport Layer Security (DTLS)**

    introduced more recently in order to provide a security service similar to TLS on top of UDP

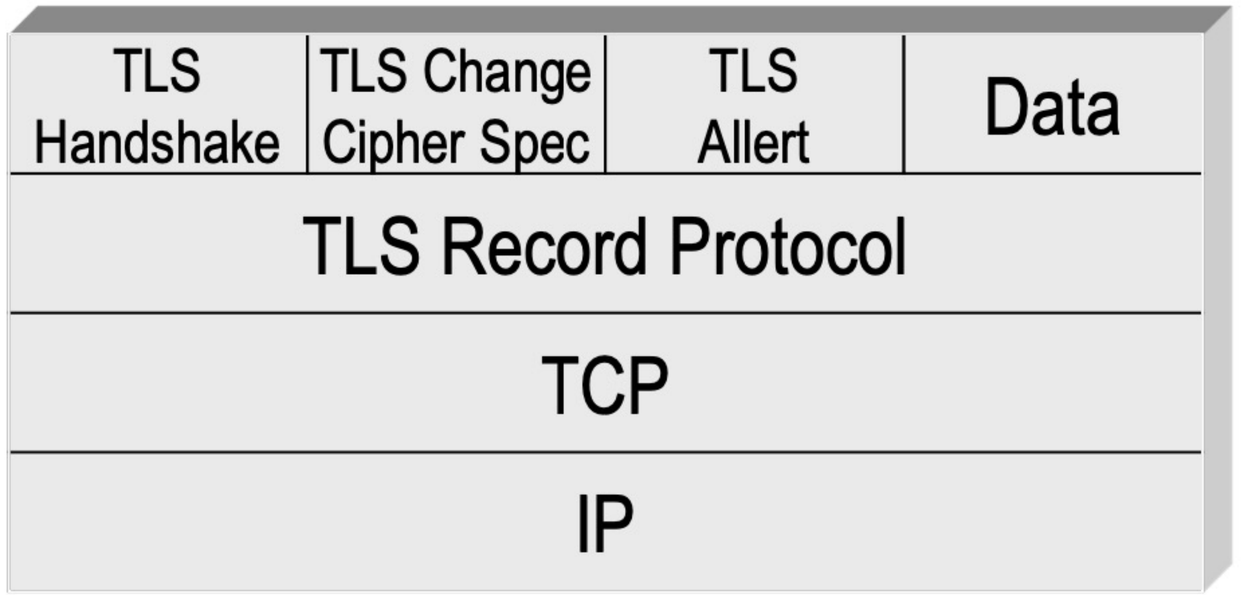
    it can be used to secure datagram traffic for client/server applications

Transport Layer Security (TLS)

- Transport Layer Security (TLS) protocol (RFC 5246)
- Widest used secure protocol, running on top of the TCP
- Same connection and stream-oriented interface of TCP
- Use X.509 certificates
- Provides privacy and data integrity between two communicating client/server applications with the following characteristics:
  - **peer's identity authentication:** using asymmetric cryptography (e.g., RSA, DSA, etc.)
  - **negotiation of encryption algorithm and key exchange:** protected against eavesdropping, modifying and replay attacks
  - **confidentiality:** symmetric cryptography is used
  - **data authentication and integrity:** message transport includes a message integrity check using a keyed MAC (e.g. based on SHA-1 hash)

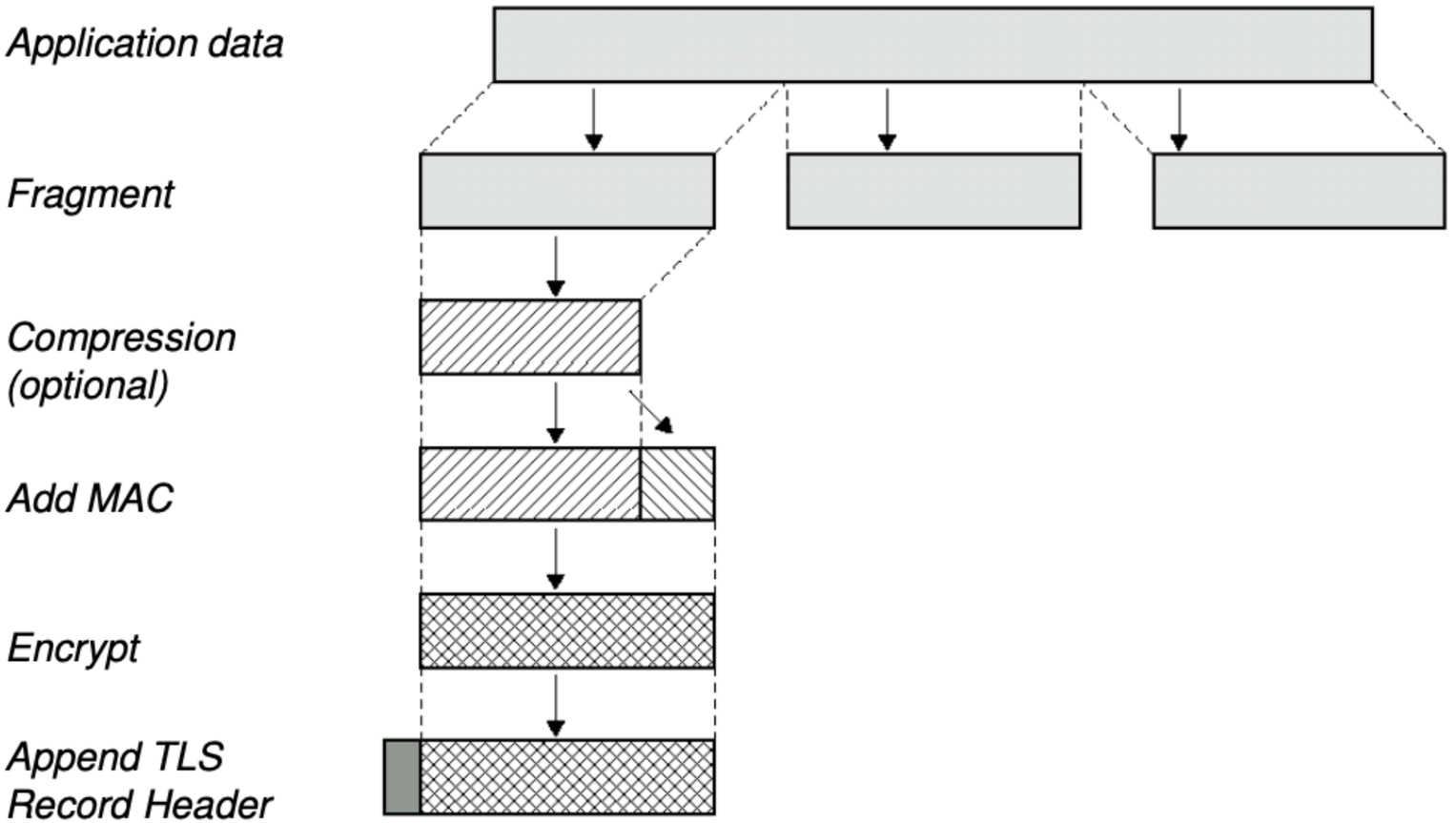
TLS Protocol Stack

- Provides privacy and data integrity between two communicating client/server applications with the following characteristics:
  - **TLS Handshake Protocol**
  - **TLC Record Protocol**



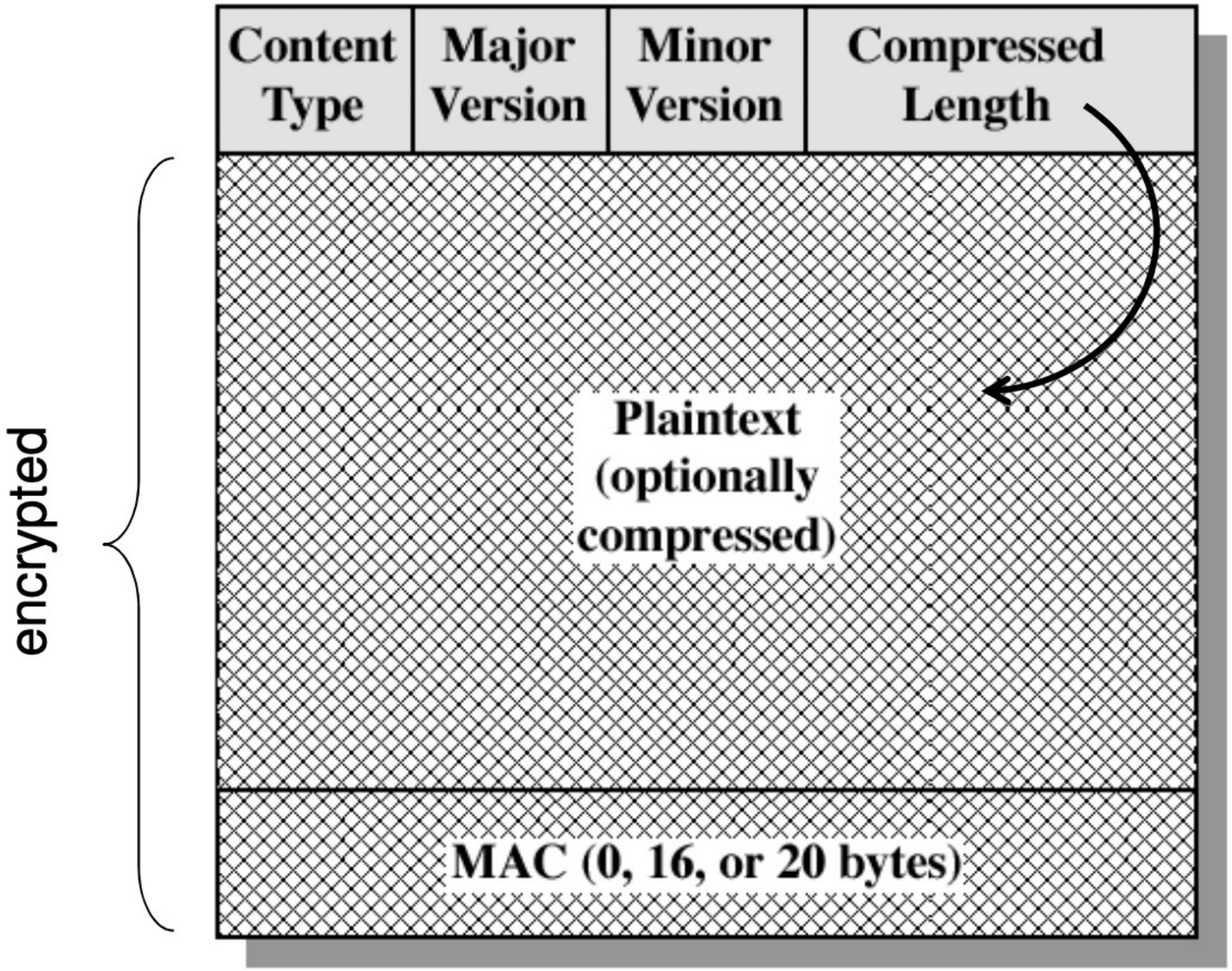
TLS Record Protocol

- TLS Record Protocol operation can be schematically represented by the following schema:



TLS Record Protocol

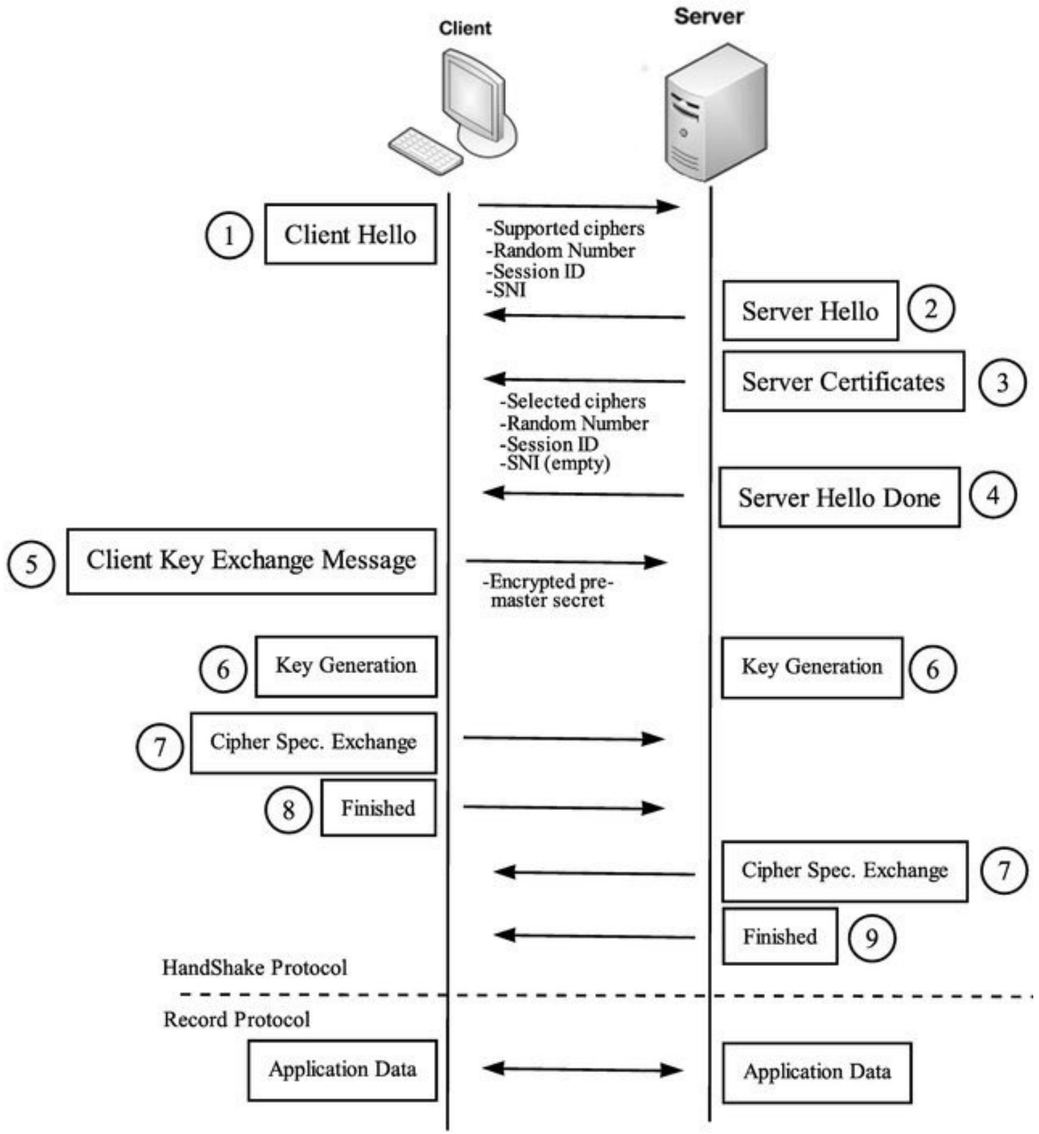
- TLS Record Protocol format is the following:



TLS Handshake Protocol

- Peer-entity authentication and key exchange is provided by the TLS Handshake phase, performed at the beginning of the communication
- TLS Handshake Protocol allows the server and client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before the application protocol transmits or receives its first byte of data
  - the peer's identity can be authenticated using asymmetric, or public key, cryptography (e.g., RSA, DSA, etc.)
  - secure negotiation of a shared secret (unavailable to eavesdroppers, or to attackers who can place themselves in the middle of the connection)

TLS Handshake Protocol



Phase A

Establish security capabilities, including protocol version, session ID, cipher suite, compression method and initial random numbers

Phase B

Server may send certificate, key exchange, and request certificate. Server signales end of hello message phase.

Phase C

Client sends certificate if requested. Client sends key exchange. Client may send certificate validation

Phase D

Change cipher suite and finish handshake protocol

TLS and DTLS

- TLS cannot be directly used with UDP, since packets may be lost or reordered (unreliability)
- TLS does not allow independent decryption of individual records
  - because the integrity check depends on the sequence number, if record N is not received, then the integrity check on record N+1 will be based on the wrong sequence number and thus will fail
  - prior to TLS 1.1 also the decryption would fail
- Datagram Transport Layer Security (DTLS) Version 1.2
  - TLS over datagram transport (UDP)
  - DTLS makes only the minimal changes to TLS required to fix the problem

Datagram Transport Layer Security (DTLS)

- Recently introduced in order to provide a security service similar to TLS on top of UDP
- Although it is still poorly supported in standard Internet nodes, it is currently the reference security protocol for IoT systems
  - It uses UDP as transport
  - It does not suffer from the problems originated by the use of TCP in network-constrained scenarios due to the extremely variable transmission delay and lossy links

Datagram Transport Layer Security (DTLS)

- Loss-Intensive Messaging
- In TLS Record Layer records are not independent
  - Cryptographic context of stream cipher key stream is retained between records
  - Anti-replay and message reordering protection are provided by a MAC that includes a sequence number, but the sequence numbers are implicit in the records
- DTLS solves the first problem by banning stream ciphers
- DLTS solves the second problem by adding explicit sequence numbers

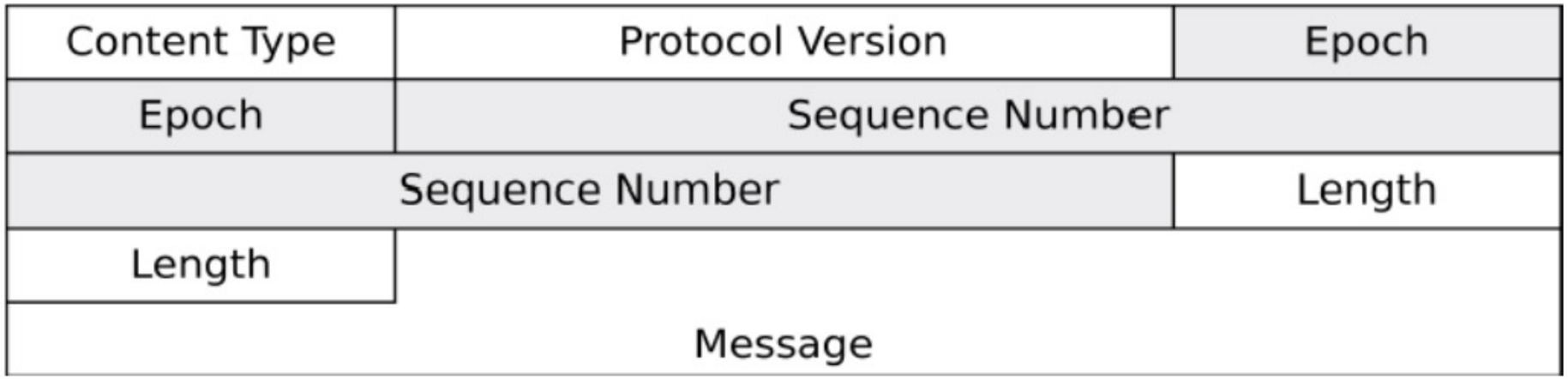
Datagram Transport Layer Security (DTLS)

- Providing reliability for Handshake
  - the TLS handshake is a lockstep cryptographic handshake since messages must be transmitted and received in defined order and any other order is an error (it is not compatible with reordering and message loss)
  - **DTLS uses a simple retransmission timer to handle packet loss**
  - **In DTLS, each handshake message is assigned to a specific sequence number within that handshake**. When a peer receives and handshake message, it can quickly determine whether that message is the next message it expects

Datagram Transport Layer Security (DTLS)

- TLS handshake messages are potentially larger than any given datagram, thus creating the problem of IP fragmentation
- **In order to compensate for this limitation, each DTLS handshake message may be fragmented over several DTLS records, each of which is intended to fit in a single IP datagram**
- each DTLS handshake message contains both a fragment offset and a fragment length

DTLS Record Protocol Format



- **Epoch**: a counter value that is incremented on every cipher state change
- **Sequence Number**: the sequence number of this record
- **Message Sequence Number**: The first message each side transmits in each handshake always has MsgSeqNum = 0. Whenever each new message is generated, the MsgSeqNum value is incremented by one.
- **Handshake Message Fragmentation and Reassembly**: When transmitting the handshake message, the sender divides the message into a series of N messages, all with the same MsgSeqNum value as the original handshake message

DTLS vs IPSec

- Both IPSec and DTLS provide the same security features with their own mechanisms at different stack layers. Furthermore, also the IPSec IKE key agreement reflects almost the same DTLS Handshake function
- The main advantage of securing communications at transport layer with DTLS consists in allowing applications to directly and easily select which, if any, security service has to be set up.
- The adoption of DTLS can allow for the reuse of the large experience and implementations that came with TLS
- **DTLS has recently received significant attention for constrained networked applications. it has been standardized as the security protocol for CoAP associated to “coaps” URIs**

Transport vs Application Level Security

- In case of intermediate nodes, end-to-end security can be still provided at transport or IP level in presence of very trusted intermediate systems. The overall security is complicated by the handling of such hop-by- hop trust management
- Providing security at IP layer (through IPSec) or transport layer (through TLS or DTLS) has some advantages
  - the same standard mechanism and the same implementation can be shared by all applications, resulting in code reuse and reduced code size
  - programmers do not have to deal with the implementation of any security mechanism

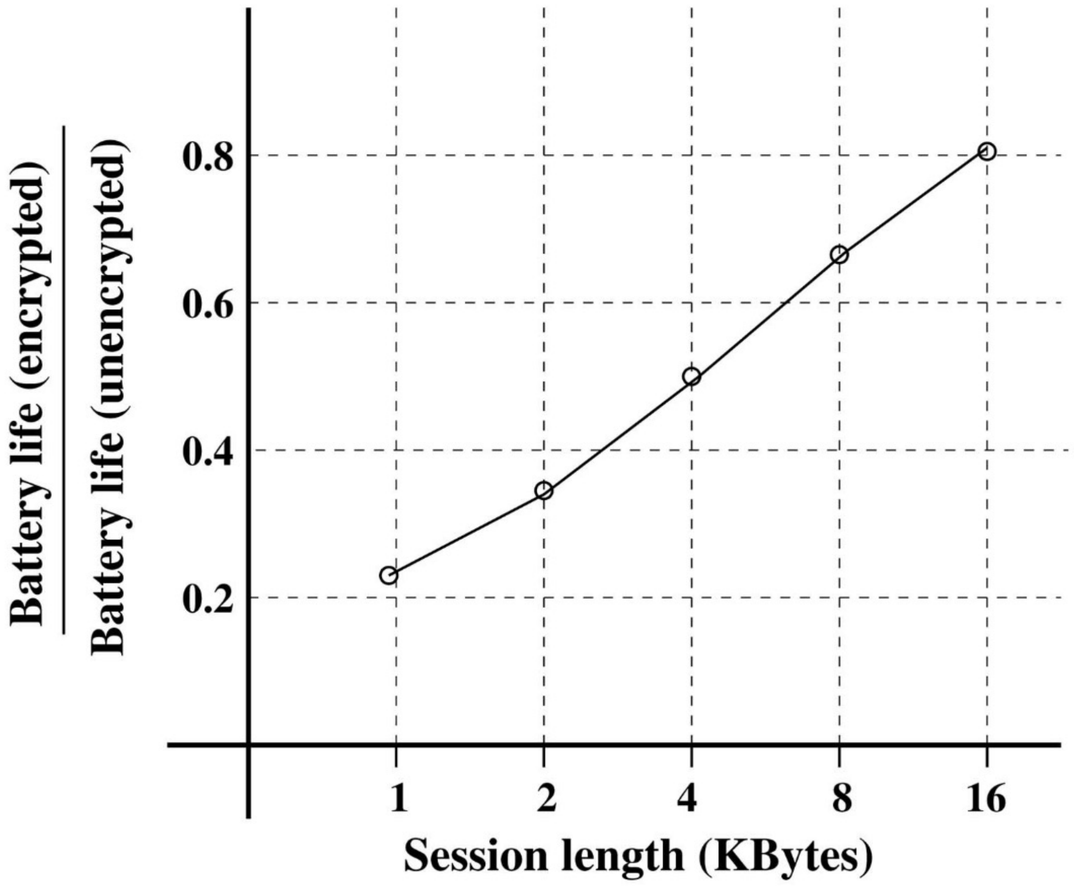
Application Level Security

- A different approach for providing complete end-to-end security is to enforce security directly at application level in order to:
  - simplifies the requirements for underlying layers, and probably reduces the cost, in term of packet size and data processing. Only application data have to be secured and per-data and not per-packet overhead is introduced
  - multicast communications, and in-network data aggregation is easier to be implemented at application level
- The main disadvantages are:
  - the complications introduced for application development
  - the overall code size due to poor reuse of software codes
- This is mainly due to the lack of well defined and adopted secure protocols at application level. Examples of standards that can be used for this purpose are S/MIME and SRTP

Energy Consumption

- For battery-powered embedded systems, one of the foremost challenges is the mismatch between:
  - the energy and performance requirements of security processing
  - the available battery and processor capabilities
- Example of impact of encryption on battery life: Motorola MC68328 processor, data rate of 10Kbps,

  with a battery capacity of 26KJ



N. R. Potlapally, S. Ravi, A. Raghunathan, and N. K. Jha, "A Study of the Energy Consumption Characteristics of Cryptographic Algorithms and Security Protocols", IEEE Transactions on Mobile Computing.

Energy Consumption

Energy Cost of Digital Signature Algorithms

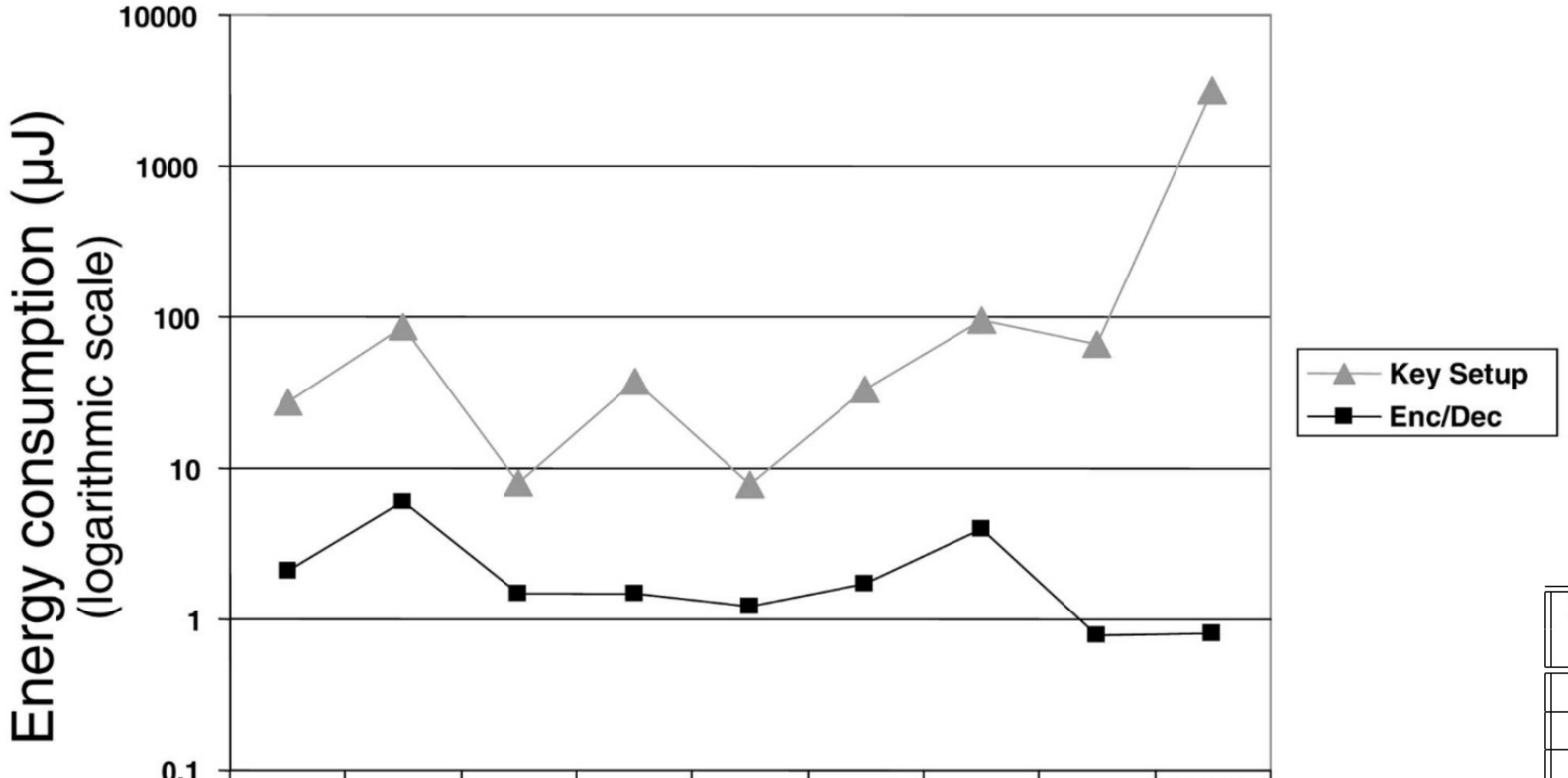
Algorithm	Key size ($bits$)	Key generation (mJ)	Sign (mJ)	Verify (mJ)
RSA	1,024	270.13	546.50	15.97
DSA	1,024	293.20	313.60	338.02
ECDSA	163	226.65	134.20	196.23
ECDSA	193	281.65	166.75	243.84
ECDSA	233	323.30	191.37	279.82
ECDSA	283	504.96	298.86	437.00
ECDSA	409	1034.92	611.40	895.98

Energy Cost of Key Exchange Algorithms

Algorithm	Key size ($bits$)	Key generation (mJ)	Key exchange (mJ)
DH	1,024	875.96	1,046.5
ECDH	163	276.70	163.5
DH	512	202.56	159.6

N. R. Potlapally, S. Ravi, A. Raghunathan, and N. K. Jha, "A Study of the Energy Consumption Characteristics of Cryptographic Algorithms and Security Protocols", IEEE Transactions on Mobile Computing.

TLS- Energy Consumption



	DES	3DES	IDEA	CAST	AES	RC2	RC4	RC5	BLOW FISH	
Key Setup	27.53	87.04	7.96	37.63	7.87	32.94	95.97	66.54	3166.3	(µJ)
Enc/Dec	2.08	6.04	1.47	1.47	1.21	1.73	3.93	0.79	0.81	(µJ/byte)

Energy Costs of AES Variants

Key size (bits)	Key setup (μJ)	ECB (μJ/B)	CBC (μJ/B)	CFB (μJ/B)	OFB (μJ/B)
128	7.83	1.21	1.62	1.91	1.62
192	7.87	1.42	2.08	2.30	1.83
256	9.92	1.64	2.29	2.31	2.05

N. R. Potlapally, S. Ravi, A. Raghunathan, and N. K. Jha, "A Study of the Energy Consumption Characteristics of Cryptographic Algorithms and Security Protocols", IEEE Transactions on Mobile Computing.

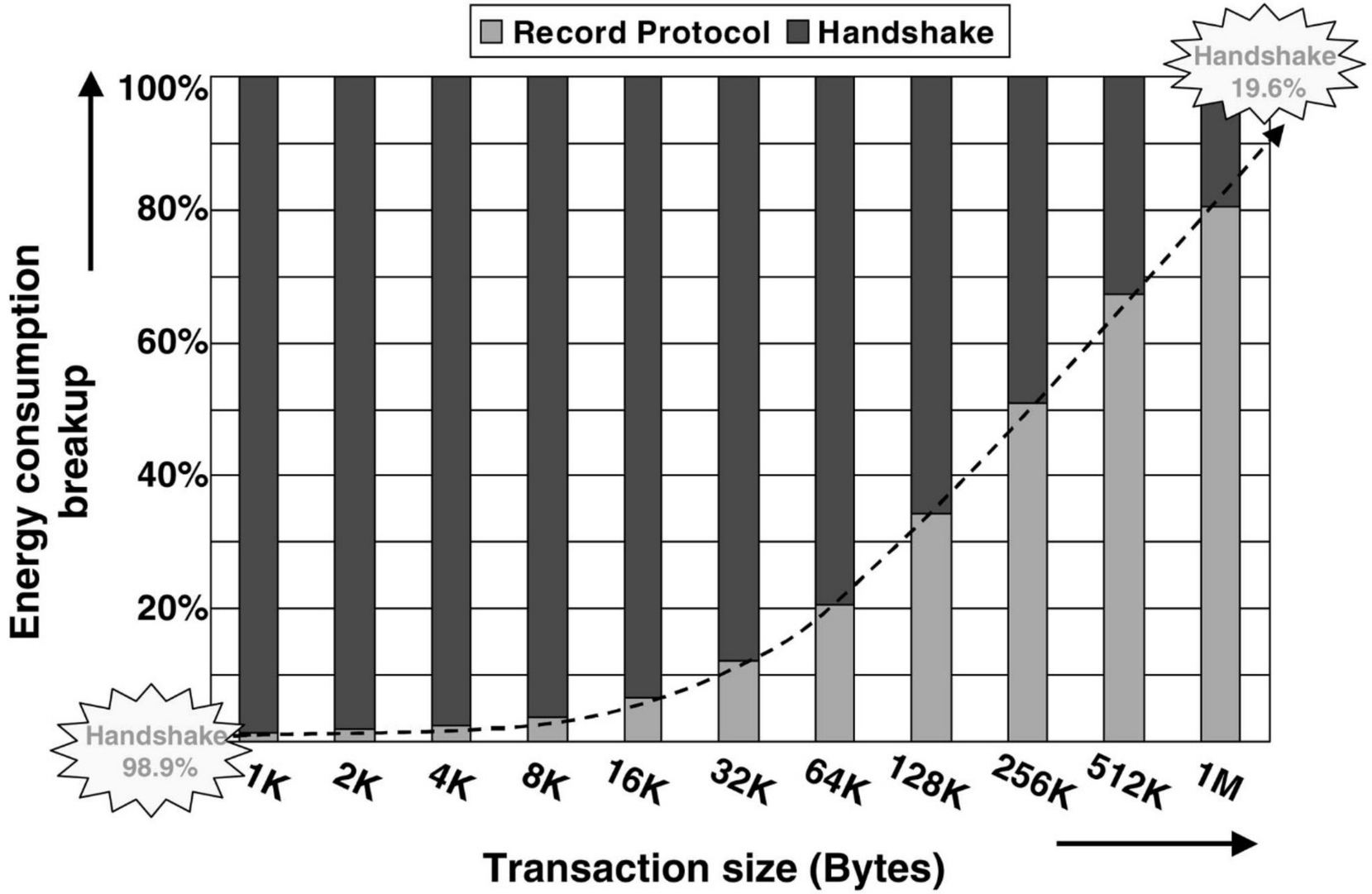
TLS - Energy Consumption

Communication Energy Overhead in the Client Due to SSL Security Processing

Transaction size ($bytes$)	Energy needed without SSL overhead (uJ)	Action	Data exchanged when using SSL ($bytes$)	Energy spent with SSL overhead (uJ)	Energy overhead of SSL usage (uJ)
1K	2,703.36	Tx Rx	236 2,748	8,047.68	5,344.32
10K	27,033.6	Tx Rx	236 12,092	32,715.84	5,682.24
100K	270,336.00	Tx Rx	236 105,660	279,735.36	9,399.36
1M	2,768,240.64	Tx Rx	236 1,066,620	2,816,669.76	48,429.12

N. R. Potlapally, S. Ravi, A. Raghunathan, and N. K. Jha, "A Study of the Energy Consumption Characteristics of Cryptographic Algorithms and Security Protocols", IEEE Transactions on Mobile Computing.

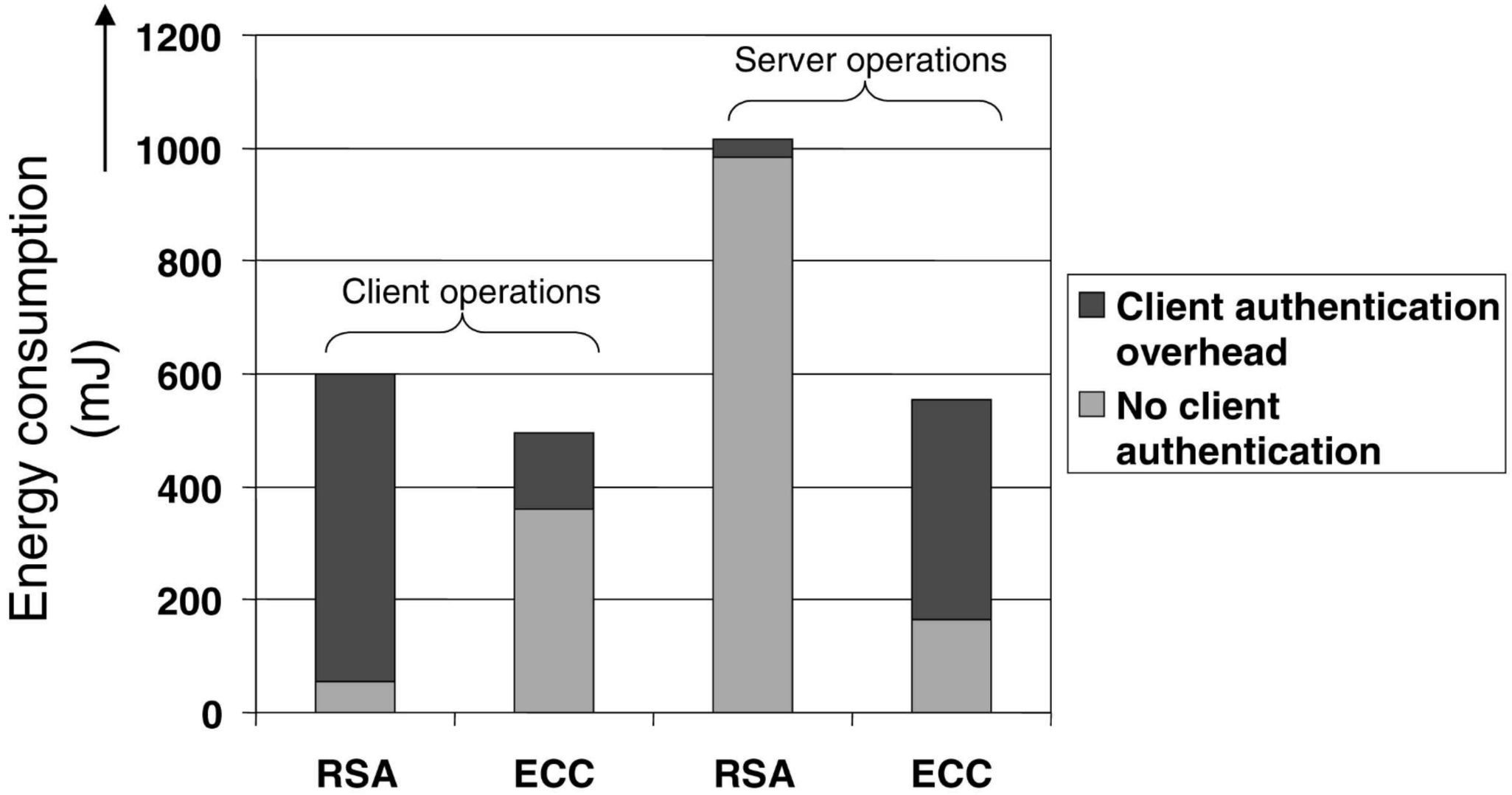
TLS - Energy Consumption



Variation of energy consumption contributions from the SSL handshake and record stages with increasing transaction sizes.

N. R. Potlapally, S. Ravi, A. Raghunathan, and N. K. Jha, "A Study of the Energy Consumption Characteristics of Cryptographic Algorithms and Security Protocols", IEEE Transactions on Mobile Computing.

TLS - Energy Consumption



Energy consumption for client and server operations in SSL handshake under the presence or absence of client authentication

N. R. Potlapally, S. Ravi, A. Raghunathan, and N. K. Jha, "A Study of the Energy Consumption Characteristics of Cryptographic Algorithms and Security Protocols", IEEE Transactions on Mobile Computing.

Security & MQTT

- When it comes to MQTT IoT network security there are three basic concepts to keep in mind: **identity**, **authentication** and **authorization**
- In each MQTT scenario, there is at least a client (acting as a publisher or a subscriber) and a broker
- A broker receives all messages and coordinates the publishing of messages to clients that are subscribed
- The **broker** is **responsible** for persisting connections, as well as **identifying** and **authorizing the transfer of data to MQTT clients**. The MQTT connection is only between one client and a broker

Security & MQTT - Identity

- In order for a client to make a connection with a broker, it must initiate a request to connect
- The MQTT protocol specifies that a client must report a client id when requesting a connection
- Ideally, every client has a unique client identifier; most devices come equipped with a universal unique identifier (UUID) or a MAC address of the network device used to connect the client
- Once a broker receives a command from a client to connect, it determines if the client is eligible to connect if the message received contains a valid **client id**, **username** and **password**
- Username and password are optional when requesting a connection to a broker but can be used and associated to the target client id

Security & MQTT - Authentication with Username/Password

- The MQTT protocol provides username and password fields in the CONNECT message for authentication. The client has the option to send a username and password when it connects to an MQTT broker
- The username is an UTF-8 encoded string. The password is binary data with a maximum of 65535 bytes (MQTT version 3.1.1 also removes the previous recommendation for 12 character passwords)
- When the username and password are set on the client, **the information is sent to the broker in plain text**. This text is vulnerable to eavesdropping and provides an easy way for attackers to obtain the credentials. Secure transmission of usernames and passwords requires transport encryption.
- The broker evaluates credentials based on the authentication mechanism that is implemented (more on that in the next post) and returns one of the following return codes:

Return Code	Return Code Response
0	Connection Accepted
4	Connection Refused, bad user name or password
5	Connection Refused, not authorized

Security & MQTT - Authentication with X.509

- In addition to authentication with username and password, the MQTT protocol allows a device to authenticate with a X.509 certificate
- X.509 is a digital certificate that uses a public key infrastructure to verify that a public key belongs to a client.
- The X.509 authentication is associated to TLS as its encryption method on top of the TCP/IP layer and to a certificate authority responsible to verify the identity of a client
- During the connection handshaking process:
  - the client presents the broker with its certificate containing information such as its identity and public key
  - the broker then relays this certificate to the certificate authority for verification. Once verified, the broker can ensure that the client certificate is genuine and gain trust in the binding with the client name and public key
- X.509 certificates provide the added benefit of verifying the identity of MQTT clients as well as providing authentication at the transport level.

Security & MQTT - TLS Best Practices

- **Always use TLS if you can:** if you can afford the additional bandwidth and your clients have enough computing power and memory for TLS
- **Use the highest available TLS version possible:** While TLS 1.0, TLS 1.1 and TLS 1.2 are not broken in practice (at least until now), you should always use the highest TLS version that is feasible for your MQTT clients
- **Always validate the X509 certificate chain:** To prevent Man-In-The-Middle attacks, have your MQTT clients validate the X509 certificate from the MQTT broker

Security & MQTT - TLS Best Practices

- **Use certificates from trusted CAs:** Certificates from trusted CAs are worth the extra money they cost. Self-signed certificates don’t play well with many TLS implementations.
- **Use other security mechanisms:** In addition to TLS, it is always a good idea to use further security mechanisms such as **payload encryption**, **payload signature verifications**, and authorization mechanisms
- **Only use secure cipher suites:** There are many insecure cipher suites available for TLS. Make sure to only allow secure cipher suites for your communication. Many cipher suites are known to be broken and don’t a false sense of security.

Security & MQTT - Authorization

- When a client establishes a connection with the broker, it can perform two activities: publish and subscribe to topics
- **Topics are the main resource available to clients and require authorization/protection for a secure system – otherwise, without authorization, any client would have the ability to subscribe and publish to any topic available on the broker**
- The most common types of authorization used are:
  - **Role Based Access Controls (RBAC):** a role provides a level of abstraction between a client and the main resource, i.e. topics in this case. Permissions are always associated with a certain role, allowing the broker to authorize the ability for a client to publish or subscribe to a certain topic
  - **Access Control List (ACL):** associates certain clients with a list of permissions. These permissions provide policies on what topics a client can subscribe/publish to
- Using ACL or RBAC a broker can be configured with topic permissions. During the run-time, the broker can determine allowed topics, allowed operations, and allowed quality of service
- If a client attempts to perform an unauthorized operation, the broker can perform actions such as disconnecting from a client or acknowledging the client but preventing it from the publishing data to other clients that have subscribed to the same topic.

Security & MQTT - Authorization with Access Tokens

- Access tokens provide an additional mechanism for scoping permissions for clients. By scoping the permissions of a client, you can prevent unauthorized access to read or write data that may have adverse effects on other client devices connected to your IoT infrastructure.
- When publishing or subscribing, the broker needs to authorize the client. Token authorization allows for a client to claim what scope and/or permissions
- In order to connect to a broker with an access token, the client must send its access token with a connect message using the password field. In this case, additional identification can be provided with the username field as well. Before requesting a connection, the client must be provided with an access token and scope.
- There are a variety of token services available. The most commonly used is OAuth 2.0. The client credential flow is modeled above where a client requests a token using credentials and is granted an access token.

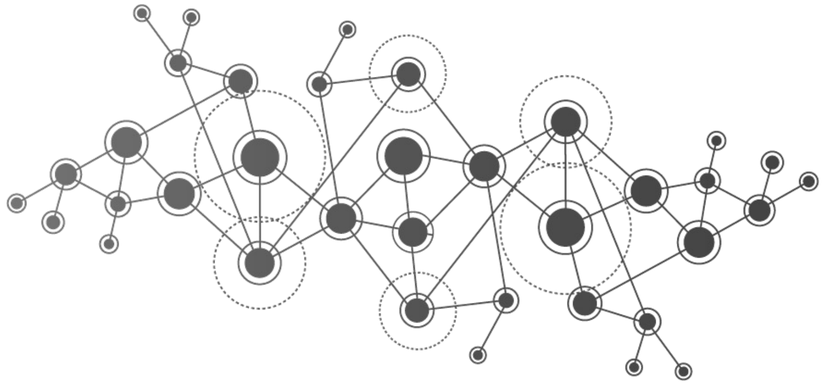
Security & MQTT - Authorization

- Without proper authorization, each authenticated client can publish and subscribe to all available topics. This can be desirable in a closed system. However, for most use cases, fine-grained restrictions make a lot of sense and should be enforced
- To restrict a client to publishing or subscribing to only authorized topics, topic permissions must be implemented on the broker side. These permissions need to be configurable and adjustable during the run time of the broker. Here is a possible topic permission:
  - Allowed topic (exact topic or wild card topic)
  - Allowed operation (publish, subscribe, both)
  - Allowed quality of service level (0, 1, 2, all)

Security & MQTT - Authorization - Best Practice

- A common best practice is to include the client ID of the publishing client in the permission.
- The client is restricted to publish only to topics that are prefixed with its own client ID
- For example:
  - client123/temperature or client123/#
- The same solution can be used for subscribing
- This is a good pattern for topics that are only concerned with one client
- Of course, this is often not the only permission needed. Frequently, a client has permissions to subscribe to more general topics. For example, clients/status or clients/command. Use of this pattern depends highly on your individual use case.

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA



Intelligent Internet of Things

IoT & Security

Prof. Marco Picone

A.A 2023/2024