Bidirectional TLS Handshake Caching for Constrained Industrial IoT Scenarios

Jörn Bodenhausen*, Simon Mangel*, Thomas Vogt*, Martin Henze*§
*_Security and Privacy in Industrial Cooperation_, RWTH Aachen University, Germany
§_Cyber Analysis & Defense_, Fraunhofer FKIE, Germany
{bodenhausen, vogt, henze}@spice.rwth-aachen.de · simon.mangel@rwth-aachen.de

***Abstract*—While TLS has become the de-facto standard for end-to-end security, its use to secure critical communication in evolving industrial IoT scenarios is severely limited by prevalent resource constraints of devices and networks. Most notably, the TLS handshake to establish secure connections incurs significant bandwidth and processing overhead that often cannot be handled in constrained environments. To alleviate this situation, we present BiTHaC which realizes bidirectional TLS handshake caching by exploiting that significant parts of repeated TLS handshakes, especially certificates, are static. Thus, redundant information neither needs to be transmitted nor corresponding computations performed, saving valuable bandwidth and processing resources. By implementing BiTHaC for wolfSSL, we show that we can reduce the bandwidth consumption of TLS handshakes by up to 61.1% and the computational overhead by up to 8.5%, while incurring only well-manageable memory overhead and preserving the strict security guarantees of TLS.**

***Index Terms*—Transport Layer Security, Industrial IoT, Handshake Caching, Session Resumption, Constrained Environments**

I. Introduction

With the omnipresence of Internet technology in modern communication, its use in industrial settings such as smart cities [1], vehicular communication [2], sensor networks [3], or industrial control systems [4] becomes increasingly relevant. Due to the connection to critical infrastructure and the resulting significance of operational safety [5], end-to-end (E2E) security of industrial communication is crucially important [6].

To ensure interoperability, Transport Layer Security (TLS), the de-facto standard protocol for E2E security on the Internet, is a key candidate for securing IIoT communication [7]. Often, the use of TLS in industrial communication is even required by laws and standards [8], [9]. However, as any security solution in constrained IIoT scenarios, TLS has to adhere to often strict constraints regarding bandwidth, processing, and memory prevalent across industrial devices and networks [10], [11], especially when considering the push towards deeply embedded devices relying on wireless communication [12], [13]. Contrary to these strict resource bounds, especially the use of public-key cryptography during TLS handshakes incurs significant bandwidth and processing overheads. This makes the use of TLS in constrained IIoT scenarios challenging [8], [14], an issue that will further exaggerate with the ongoing transition towards post-quantum cryptography [15], [16].

To account for the restrictions imposed by constrained IIoT scenarios and still allow for the use of TLS, related work has proposed optimizations for TLS: (i) use of profiles [17], [18] and optimized encoding [19], [20], (ii) out-of-band transmission [21], [22] and handshake delegation [23], [24], as well as (iii) caching of sessions [25], [26] and handshakes [27], [28].

Especially caching allows for substantial resource savings. However, *session* caching [25], [26] requires frequent full handshakes to cryptographically decouple connections [29], making it unsuited for industrial scenarios [8]. Contrary, *handshake* caching [27], [28] fully preserves E2E security guarantees and thus is an extremely promising candidate to optimize TLS for constrained IIoT scenarios. Still, existing approaches mainly target to reduce the bandwidth overhead imposed by server certificates in Internet communication and thus do not consider the specifics of the constrained IIoT.

To fill this gap, we present BiTHaC, our approach to realize bidirectional TLS handshake caching for constrained IIoT scenarios. BiTHaC exploits that significant parts of the TLS handshake are static, and thus cacheable. Consequently, after an initial full handshake, those parts neither need to be transmitted (saving bandwidth) nor corresponding computations performed (reducing processing overhead and latency) again. To this end, BiTHaC adapts the idea of caching server certificates on the client [28] and adds a corresponding counterpart to cache client certificates on the server. Furthermore, BiTHaC specifically accounts for constrained devices through memory-optimized caching and skipping of redundant computations. As such, BiTHaC substantially decreases the overhead of TLS handshakes without impacting E2E security guarantees.

Contributions. We address the need to reduce bandwidth consumption and computational overhead of the TLS handshake in constrained scenarios with the following contributions:

1) We tailor the idea of caching static parts of server messages in TLS handshakes to constrained IIoT scenarios and provide a mechanism to also cache static parts of client messages without compromising security.
2) We propose a novel paradigm to handling client-side caches such that redundant yet resource-intensive computations can be skipped with minimal memory impact.
3) Our evaluation for TLS 1.2 and 1.3 shows that BiTHaC successfully reduces the bandwidth consumption by up to 61.1%. Likewise, BiTHaC reduces the computational overhead of TLS handshakes by up to 8.5%.

Availability Statement. Our implementation of BiTHaC is available at https://github.com/RWTH-SPICe/BiTHaC

Author's version of a paper accepted for publication in Proceedings of the 2025 IEEE 50th Conference on Local Computer Networks (LCN).

II. E2E Security in Constrained Environments

To better understand the root causes for the challenges of using E2E security and specifically TLS in resource-constrained environments [6], [8], [12], we first introduce TLS and then discuss resulting implications for constrained environments.

A. Transport Layer Security (TLS)

TLS is an application layer cryptographic protocol to realize communication security in an E2E manner, i.e., between client and server. Over the years, multiple updates and extensions of TLS have been developed, with TLS 1.2 [30] and 1.3 [31] being the currently most relevant ones [7].

TLS realizes its functionality through various sub-protocols. Both for security and resource-consumption, the *handshake protocol* for establishing a secure connection is particularly relevant. It is used to negotiate the required parameters and authenticate one or both peers. Negotiation allows the protocol to be versatile and support a wide range of cryptographic algorithms. Moreover, an extension mechanism can be used to realize additional functionality. Using this mechanism, the client can offer additional functionality which the server can choose to accept. The extension mechanism is designed rather lightweight and consist of a header with a two-byte identifier, a two-byte length field, followed by extension specific data.

While previous version updates kept the handshake largely unchanged, TLS 1.3 introduces notable changes to achieve a more secure and faster connection establishment. The main differences are illustrated in Fig. 1 alongside the size of each message based on an exemplary connection establishment. In TLS 1.2, connection establishment requires four flights and starts with the *Hello* messages, in which fundamental parameters and the extensions are negotiated. Subsequently, key-exchange and authentication steps are performed through the exchange of *Certificate* and *Key Exchange* messages. Finally, the connection establishment is concluded through *Finished* messages. Connection establishment in TLS 1.3 differs substantially and requires only three flights. To achieve this, dedicated *Key Exchange* messages are omitted and their functionality moved to extensions. Moreover, all messages after the *Hello* messages are encrypted, including a newly added *Encrypted Extensions* message, for additional privacy.

The message sizes in Fig. 1 show that a single handshake adds up to a bandwidth overhead in the order of KBs, even for single self-signed certificates. In reality, chains of multiple certificates further increase the bandwidth overhead [32]. Nevertheless, even in this simple case, the two *Certificate* messages make up 61.5% of the TLS 1.2 handshake messages and 57.9% for TLS 1.3. Moreover, when considering the upcoming use of post-quantum algorithms, this will further increase, as the utilized public keys are several KBs in size [32] and expected to be utilized in a hybrid manner [33].

B. Constrained Industrial IoT Scenarios

Devices and networks in Industrial IoT (IIoT) environments impose various resource constraints [10], [12], [34]. Most notably, industrial devices and networks are severely constrained w.r.t. bandwidth, processing, memory, or energy [10], [35]–[37], especially for wireless communication [12], [38], [39].

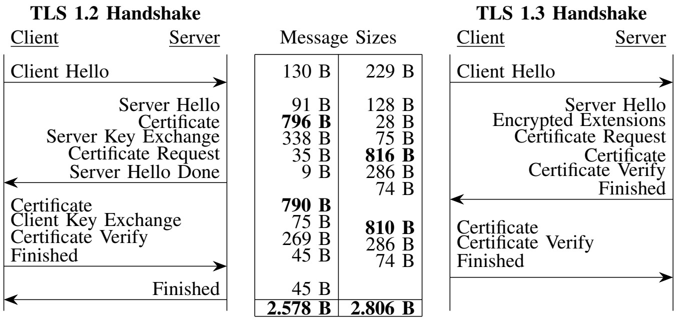


Fig. 1. Handshake comparison between TLS 1.2 and 1.3 alongside measured message sizes. Both handshakes use mutual authentication with single self-signed 2048-bit RSA certificates. A single cipher is offered by the client, resulting in an ECDHE key exchange and use of AES 256 GCM SHA384.

IIoT *devices* are often limited w.r.t. processing power, memory, and energy [10]. Using TLS to secure communication affects all three dimensions. Energy consumption is affected, as the transmission of larger messages over wireless networks consumes considerable power. Moreover, asymmetric cryptography utilized in TLS is particularly resource intensive [40], leading to substantial computational and indirectly power consumption overhead. Lastly, the memory overhead of TLS, especially the impact of asymmetric cryptography [40], is challenging for devices with only a few KB of memory [10].

Considering IIoT *networks*, constraints resulting from the prevalent use of wireless network technology limit the bandwidth available to each device and thus its ability to establish and maintain a TLS connection [8]. Moreover, wireless network technologies commonly impose restrictions on individual message sizes [41]. For instance, LoRaWAN has a maximum message size of only 256 bytes [12], which would already cause the *Certificate* messages from our example with a single certificate (cf. Fig. 1) to be fragmented, thus invoking additional overhead on lower network layers and delays.

Overall, the use of certificate chains and associated computations for mutual authentication are integral for security in TLS. At the same time, they heavily influence the overall resource overhead and particularly bandwidth consumption, especially in constrained IIoT environments, often rendering the application of secure communication infeasible.

III. Related Work on Optimizing TLS Overhead

Various work motivate the use of TLS in the IoT [42]–[44] without detailing applicability and resulting overheads. Other proposals suggest to realize E2E secure communication other than TLS, e.g., based on HIP [45], [46] or custom solutions [47], [48]. As these jeopardize the interoperability of E2E security and often do not meet the requirements of industrial contexts [8], [9], our discussion focuses on approaches to optimize the overhead imposed by TLS.

Profiles & Optimized Encoding aim to reduce overhead through enhanced message formats or protocol restrictions, leaving protocol features and security guarantees largely unchanged. For example, header compression improves the overhead of Datagram TLS (DTLS) [19], [49], [50]. Moreover, Compact TLS 1.3 reduces the amount of transmitted information through re-design and a templating mechanism [20]. However, such savings usually come at the cost of interoperability. Improvements on the typically used X.509 [51] certificates encoding are explored through compression [52], [53] and more efficient encoding [54]–[56]. Still, these improved certificates still need to be transmitted for every connection. Conversely, different approaches reduce the overhead of TLS through restriction to a subset of available features based on profiles, e.g., exclusively using ECC [17], [54], [57]. Furthermore, an official set of profiles for IoT environments [18] is being updated for TLS 1.3 [58]. As such configuration optimizations are relevant in any particular scenario, they are orthogonal to more fundamental improvements.

Out-of-Band Transmission improves the overhead of TLS by utilizing an additional, typically unconstrained, communication channel. A prominent example is on-demand retrieval, as utilized by client certificate URLs [59]. However, this mechanism merely shifts overhead from client to server and comes with various security considerations. As an alternative, pre-provisioning of such information allows to entirely leave out information in the actual connection establishment. Examples are the templating mechanism of Compact TLS 1.3 [20], the direct use of raw public keys [22], or pre-sharing of symmetric keys [21]. However, as such information needs to be exchanged in advance, potential application scenarios and scalability are limited. Lastly, handshake delegation can be utilized for devices that are too constrained to establish a connection on their own by (partly) offloading connection establishment to another device [23], [24], [60], [61]. Such mechanisms, however, require the presence of a trustworthy powerful device [62], which often is an unrealistic assumption. Finally, the less intrusive approach of certificate pre-validation on gateways [63] avoids the unnecessary validation of invalid certificates, but still causes the full overhead for valid certificates (which is the majority of cases).

Caching omits static parts of the connection establishment across repeated connections. As the most drastic form of caching, session caching or resumption caches the entire established connection. Up to TLS 1.2, the used ID-based mechanism [17], [30], [64], shows significant security implications, especially a lack of perfect forward secrecy (PFS). In contrast, session ticket-based resumption [25] allows for PFS [31] and proven advantageous for IoT environments [18], [63]. Further extensions have been proposed, e.g., handshake delegation [23], [65] and rTLS [26]. However, session resumption generally cryptographically links subsequent connections to the original connection, which limits their lifetime and leads to the need for frequent full handshakes [29].

Handshake caching constitutes a promising alternative (or complement) to session resumption by more granularly retaining aspects of the connection establishment rather than the entire connection. Various approaches for handshake caching were proposed [27], [54], [66], [67], leading to the standardization of the cached information extension (RFC 7924) [28]. Unlike other alternatives, this extension does not break the protocol and can be flexibly negotiated, positioning it as a promising approach to improve resource efficiency of the TLS handshake. However, RFC 7924 [28] still has significant drawbacks for constrained IIoT environments: As caching only happens on the client-side, only half of the optimization potential is leveraged in IIoT scenarios where mutual authentication is imperative [68]. Furthermore, device constraints are not considered, resulting in optimization potential w.r.t. the implementation of the cache. On the technical side, RFC 7924 does not work with TLS 1.3 [15] due to ephemeral content in *Certificate* and *Certificate Request* messages.

IV. Bidirectional TLS Handshake Caching

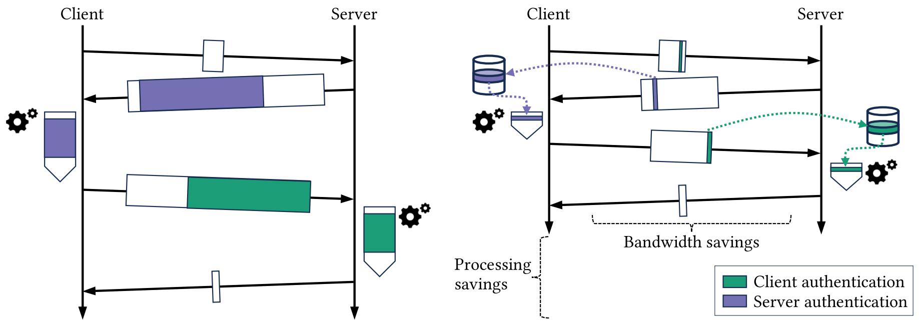
Existing approaches to optimize E2E security do not consider the specifics of IIoT scenarios (cf. Sec. III), most importantly strict resource constraints on the client and the mandatory use of mutual authentication [68]. To fill this gap, we present BiTHaC to optimize TLS for IIoT scenarios without compromising security. As illustrated in Fig. 2, BiTHaC leverages the substantial static parts of TLS handshakes (Fig. 2a) to cache their transmission and corresponding processing (Fig. 2b) without impacting E2E security.

To reduce the bandwidth overhead of TLS, we first leverage an existing caching mechanism to avoid the unnecessary transmission of static information sent by the server (Sec. IV-A), before we introduce a novel caching scheme for static information sent by the client (Sec. IV-B). Based on this, we propose a mechanism to substantially reduce processing overhead with cached information (Sec. IV-C). Finally, we present required adaptations for TLS 1.3 (Sec. IV-D).

A. Caching of Static Server Information (RFC 7924)

To cache static server information on the client-side, we leverage the caching mechanism of RFC 7924 [28] (illustrated in Fig. 3 in purple) that allows for caching of the server's *Certificate* and *Certificate Request* messages. After an initial full handshake, the client caches these messages (Fig. 3-i). In subsequent handshakes, it provides the server with options from its cache by including a list of object type and fingerprint (i.e., hash of the cached message) in the *Client Hello* (Fig. 3-ii). The server can then reuse cached elements by indicating its selection in the *Server Hello* (Fig. 3-iii). Subsequently, the respective messages are replaced with the fingerprint and thus substantial bandwidth savings realized.

As RFC 7924 [28] only defines the changes to the TLS handshake protocol, to actually use it in BiTHaC, we derive a cache structure with two distinct lookup mechanism specifically tailored to constrained IIoT devices. Here, the primary lookup utilizes the type and fingerprint value to retrieve a cached object and can be used to access the required cached information selected by the server. Moreover, we utilize a secondary lookup via a peer index to scope the selection of cached elements included in the *Client Hello* to each server.



(a) original TLS handshake (b) adapted BiTHaC handshake

Fig. 2. Significant parts of bandwidth- and computationally-intensive TLS handshakes are static (Fig. 2a). BiTHaC caches these parts to significantly reduce bandwidth consumption and processing overhead (Fig. 2b).

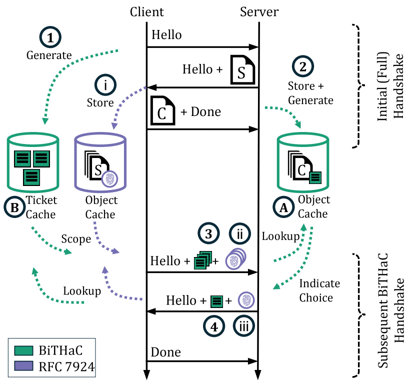


Fig. 3. BiTHaC introduces a ticket-based signaling flow to realize server-side caching.

Otherwise, the client would need to send *all* cached elements for each new connection, leading to unnecessary bandwidth overhead and a potential risk of tracking. To uniquely identify servers, we rely on the commonly used server name indication (SNI) extension or the IP address, if SNI is not used.

With these adaptations, we realize the first half of bandwidth savings while catering for the specific requirements of the IIoT.

B. A Novel Caching Scheme for Static Client Information

While caching of static server information already saves some bandwidth (by caching the servers' *Certificate* and *Certificate Request* messages), the prevalent use of mutual authentication in constrained IIoT scenarios [68] coupled with tight bandwidth restrictions demand for also caching static client information on the server-side. To this end, we propose a ticket-based mechanism for caching static client information, i.e., the *Certificate* message (while extensible to other objects, we only consider the *Certificate* message here). Our design specifically focuses on bandwidth efficiency and client privacy, i.e., tracking prevention. In the following, we provide an overview of our caching scheme, detail cache structure, extension format, ticket generation, and adapted behavior during the TLS handshake alongside Fig. 3 (in green).

Overview of Caching Scheme. Our caching scheme for static client information relies on tickets to negotiate the use of cached information. More specifically, the client generates a ticket which is used to identify an object cached at the server after a successful handshake (Fig. 3-1). Analogously, the server stores the respective object and generates the associated ticket (Fig. 3-2). In subsequent handshakes, the client includes suitable tickets in its *Hello* message (Fig. 3-3). If the server has a matching entry in its cache, it informs the client which cached information it selected for use (Fig. 3-4). Subsequently, the client omits the selected cached information from sent messages. Unlike for caching static server information, we thus move the choice which cached objects will be used to the entity storing the cached information (i.e., the server). This is necessary, as servers usually cannot identify clients at the beginning of connection establishment (while clients can easily identify servers, e.g., based on their SNI). Consequently, if the client were to make the selection of which cached information to use (similar to RFC 7924), the server would need to send information on its *complete* cache to the client, which is not feasible and would raise privacy issues.

Cache Structure. Using tickets requires to not only maintain an object cache at the server (for static client information – Fig. 3-A), but also a cache of tickets at the side (Fig. 3-B). For each ticket, this cache retains the type of the cached object, the associated server's identity (i.e., SNI) for scoping (to prevent tracking), and a reference to the matching object, as we support caching multiple objects (e.g., certificate chains). On the server side, an object cache is required to retain static client information and link it to tickets. As no scoping is required here, a single lookup mechanism via the ticket suffices.

Extension Format. For the exchange of information between client and server, we create a dedicated TLS extension, as this allows for minimal modification and ensures interoperability. More specifically, we define extensions for the respective *Hello* messages of client and server (cf. Sec. II). On both sides, we utilize a single list, such that the extension consists of the four-byte header, a two byte length field, and the (potentially empty) list. The *Client Hello* message then contains a list of entries with the form object type (1 B), ticket length (1 B) and ticket (see below), where object types are chosen according to RFC 7924 [28]. Similarly, the *Server Hello* message contains a list of elements with the form object type (2 B) and ticket index (2 B), indicating the server's selection of cached objects. In case the server does not cache any of the objects signaled by the client, it responds with an empty list. During the initial handshake, client and server signal support for BiTHaC using a zero-length extension.

Ticket Generation. As we use tickets to identify cache objects, client and server need to obtain the same ticket when caching static client information. Typically, e.g., in session resumption [25], one party generates a ticket and explicitly transmits this to the other party. However, this results in unnecessary bandwidth overhead. Instead, BiTHaC relies on implicit ticket generation based on the TLS key derivation function [30] to derive eight-byte tickets from the *MasterSecret*: `PRF(MasterSecret, "ssc_ticket_label",` `object type || object hash)`. As ownership of the actual cached object is still proven in the handshake, no substantial bit security needs to be provided by the ticket's length. Furthermore, our mechanism generates fresh tickets for each connection, thus cryptographically decoupling connections.

Adapted Behavior. To realize our caching scheme for static client information, we adjust different steps of the TLS handshake at the client and server. First, when building the *Client Hello* message, the client checks its cache for available tickets scoped to the server and includes them into the extension (if no tickets are available, it sends an empty extension to signal support for BiTHaC). Upon receiving tickets from the client, the server checks its cache for corresponding entries and responds with the cache entry it selected (or an empty list if no matching cache entry exists). If the client only sends an empty extension (to signal support for BiTHaC), the server likewise signals its support. Should either client or server not support BiTHaC, the handshake proceeds as normal, thus ensuring backwards compatibility. Upon receiving the *Server Hello* message, the client keeps track of server support for BiTHaC and parses the server's selection of cached objects. If a cached certificate should be used, the client replaces the content of the *Certificate* message with a zero-length message. The server expects such an abbreviated message and uses the selected cached certificate chain instead. Finally, after exchanging the *Finished* messages, both peers generate (new) tickets (as described above) and update their caches.

By caching static client information, we can omit redundant content of messages sent by the client, most notably the client certificate, and thus substantially reduce bandwidth usage.

C. Minimizing Processing with Cached Static Information

Besides cutting bandwidth consumption, BiTHaC also leverages cached information to substantially reduce the processing overhead for validating certificate chains. To this end, BiTHaC validates the certificate chain only once during the initial handshake and subsequently only re-validates time-dependent operations, e.g., certificate revocation checks. Thus, BiTHaC not only substantially reduces processing overhead but also obviates the need to cache the complete certificate chain and thus allows to save valuable memory on constrained devices. In an essence, only the public key of the leaf certificate required during every handshake as well as information required to perform time dependent validation steps need to be cached.

These steps validate certificate lifetimes, for which it suffices to cache the two most restrictive timestamps (i.e., the maximum of "not before" and minimum of "not after" values across the certificate chain). Furthermore, if optional checks for revocation using Certificate Revocation Lists (CRL) [51] or Online Certificate Status Protocol (OCSP) [69] are used, also the hash of the issuer name and public key, the serial number, and potentially the CRL distribution points are cached.

Thus, by substantially reducing the number of signature verifications for repeated handshakes, BiTHaC speeds up handshakes and cuts energy consumption. Likewise, as cache entries are reduced to one public key and two timestamps (if no optional revocation checks are used) instead of storing complete certificate chains, the memory overhead of BiTHaC is well-manageable even for memory constrained IIoT devices.

D. Interoperability with TLS 1.3

While TLS 1.2 still is the prevalent choice in IIoT scenarios [7], future deployments will likely only support TLS 1.3 and should equally benefit from the tremendous improvements of BiTHaC. As TLS 1.3 introduces significant changes to connection establishment (cf. Sec. II), we briefly discuss how these changes impact BiTHaC. The most notable change in TLS 1.3 is that all messages after the *Hello* messages are encrypted and an additional *Encrypted Extension* message is introduced. We leverage this to move the server-side signaling to this message for additional tracking protection.

The changes to the *Certificate* message require further consideration. Here, TLS 1.3 adds a request context and the option to add extensions to each certificate in the chain. Although providing useful functionality, this might lead to non-static content, complicating caching. To address this, we leverage a conversion function that (after verification of the certificate chain) transforms a TLS 1.3 message into its TLS 1.2 equivalent by removing certificate extensions and request context, which can then be handled by BiTHaC as for TLS 1.2. An abbreviated *Certificate* message is then achieved by utilizing zero-length certificate fields followed by their non-static extensions rather than a zero-length *Certificate* message.

With these minor adaptations, the advantages of BiTHaC also apply to constrained IIoT devices using TLS 1.3.

V. Evaluation

To quantify the improvements of BiTHaC and thus the potential to widely deploy TLS in constrained IIoT environments, we implement BiTHaC on top of wolfSSL version 5.6.0 [70]. We implement RFC 7924 [28], our caching scheme for static client information, and our approach to reduce processing overhead based on caching for TLS 1.2 and 1.3.

A. Bandwidth Improvements

To demonstrate the effectiveness of BiTHaC in reducing the bandwidth overhead of TLS handshakes by omitting redundant transmissions of certificate chains, we perform measurements of bandwidth usage across various parameters.

Methodology. We evaluate BiTHaC in a Docker [71] environment by deploying both a client and server utilizing our adapted wolfSSL library in a docker container based on Alpine Linux version 3.18.3. A third docker container runs `tcpdump` [72] to capture communication over a virtual Ethernet network. As wireless networks used in constrained IIoT environments impose tight restrictions on message sizes, TLS messages often get fragmented [8], [12]. Thus, we adapt the virtual network to study different Maximum Transmission Units (MTUs): 127 B (MTU in IEEE 802.15.4 wireless networks [73]), 576 B (minimum receivable datagram size IPv4 devices must support [74]), and 1500 B (typical MTU in Ethernet [75]).

To put the bandwidth improvements of BiTHaC into perspective, we compare (i) an unmodified version of wolfSSL without caching support (*Vanilla*), (ii) our extended version of wolfSSL with support for RFC 7924 (*RFC7924*), (iii) our implementation of BiTHaC (*BiTHaC*), and (iv) an unmodified version of wolfSSL with ID-based (TLS 1.2) or ticket-based (TLS 1.3) session resumption (*Sess.Res.*).

For each measurement, we establish a connection, send one message per direction and then close the connection. Unless stated otherwise, both peers use a chain of three 2048-bit RSA certificates for authentication. Root certificates are not transmitted, as allowed by the specification [30]. Connections relying on caching or resumption are preceded by an initial full handshake to populate the cache resp. generate a session. We repeat each measurement ten times, report the arithmetic mean over these measurements, and indicate the minimum and maximum observed values through error bars.

Different MTUs. In Fig. 4, we report on the resulting bandwidth usage of the four TLS configurations for TLS 1.2 and varying MTUs, divided by the different protocol layers. As expected, the unmodified TLS configuration (Vanilla) consumes the most bandwidth with 6 to 11 kB (slight deviations denoted by the error bars result from race conditions leading to different numbers of acknowledgments). Already by deploying RFC 7924 to cache static server information, bandwidth usage can be reduced by 31.67% to 27.58% (depending on the MTU). When using BiTHaC to additionally cache static client information, this can be further decreased to between 43.25% and 38.86%. While session resumption is even more efficient and decreases the required bandwidth by up to 80%, this comes at the cost of cryptographically linking *all* resumed sessions to the initial session, severely limiting the allowed lifetime of cached sessions (< 24 h for TLS 1.2 sessions [30]), thus requiring to carry out full handshakes frequently [8]. Still, BiTHaC and session resumption are not mutually exclusive and can be used in tandem to combine their strengths.

Considering the influence of varying MTUs, we observe that the different network layers contribute differently. While the bandwidth required for TLS remains unchanged, a decrease in the MTU leads to a disproportionate increase in bandwidth overhead on the lower layers. This particularly highlights the importance of optimizing bandwidth consumption for severely constrained wireless networks such as IEEE 802.15.4.

Savings of BiTHaC. Further investigating the bandwidth reductions realized by BiTHaC, we observe that the server's *Certificate* message is reduced to a constant size of 42 B with caching. Likewise, the variable size of the client's *Certificate* message is reduced to a constant size of only 9 B, due to BiTHaC's more efficient caching scheme for static client information. Furthermore, the *CertificateRequest* message is replaced with the selected fingerprint, which results in a marginal reduction of the already constant size of 45 B to 42 B. To realize these savings, BiTHaC introduces a reasonable overhead of combined 111 B in the *Hello* messages of client and server. For initial handshakes, BiTHaC's signaling mechanism (cf. Sec. IV-B) causes merely 8 B of overhead.

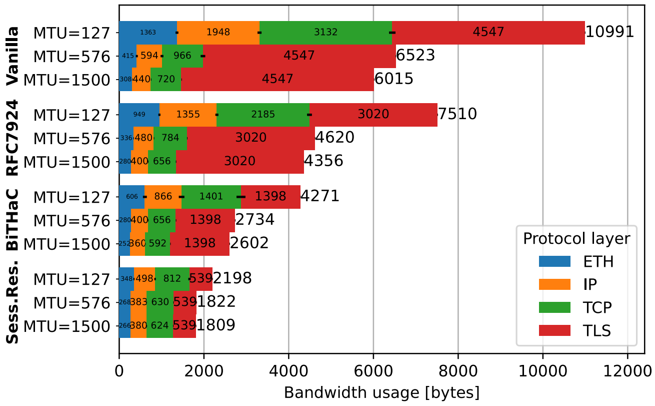


Fig. 4. BiTHaC substantially reduces bandwidth usage compared to unmodified TLS 1.2 as well as straightforward caching based on RFC 7924. While session resumption provides even more savings, those weaken security.

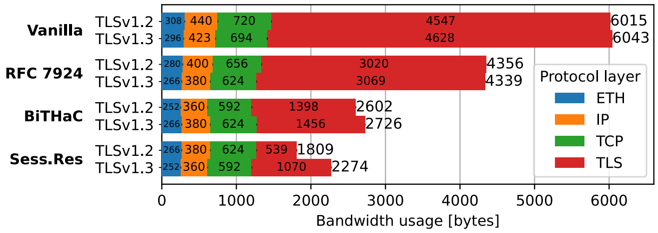


Fig. 5. When switching from TLS 1.2 to 1.3, BiTHaC is still able to achieve substantial bandwidth savings. In contrast, session resumption cannot uphold the same level of bandwidth savings due to the switch to session tickets.

Notably, BiTHaC replaces variable-sized static information with a fixed, small placeholder. Consequently, for larger certificates (or chains), bandwidth savings become even more pronounced. E.g., when switching to a 4096-bit RSA certificate chain for the server, BiTHaC reduces the TLS portion of the bandwidth usage from 5667 B to only 1494 B. Thus, with the ongoing switch to post-quantum cryptography, BiTHaC's optimizations become even more relevant.

TLS 1.2 vs. 1.3. In Fig. 5, we compare TLS 1.2 and 1.3 for an MTU of 1500 B. Overall, the results are mostly comparable, with BiTHaC realizing a bandwidth reduction of more than 3.3 kB (54.89%) for TLS 1.3. Particularly noteworthy is the relative improvement of BiTHaC compared to session resumption, which only realizes additional bandwidth savings of 16.58% (452 B) for TLS 1.3, with still weaker security guarantees. Session resumption's increase in bandwidth mainly stems from the switch to session tickets in TLS 1.3 to provide PFS [31]. Consequently, with the ongoing shift towards TLS 1.3 in IIoT scenarios [7], the added security of BiTHaC over session resumption becomes even more attractive.

B. Processing Improvements

Besides substantially reducing bandwidth overhead, BiTHaC also reduces processing overhead using cached static information to avoid redundant computations (cf. Sec. IV-C). Most notably, for cached certificate chains, this promises to save several costly signature verifications.

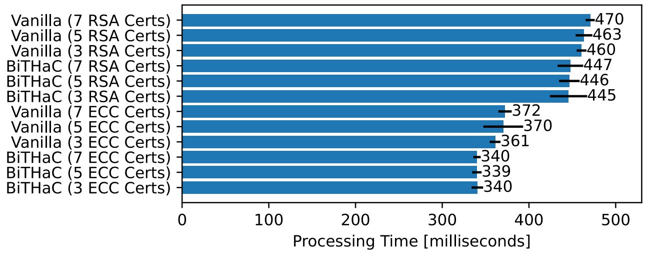


Fig. 6. BiTHaC reduces the variable processing overhead for validating the entire certificate chain to a constant overhead that only depends on the public key in the leaf certificate, achieving a reduction of up to 8.6% (32 ms).

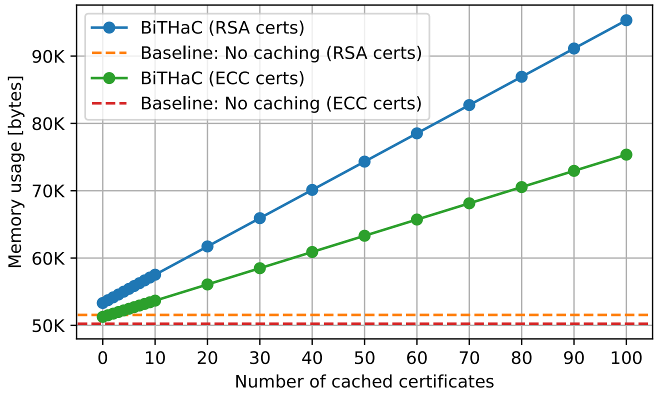


Fig. 7. The memory usage of BiTHaC scales linearly in the amount of cached certificates and is well-manageable for memory-constrained IIoT devices.

Methodology. To evaluate the processing advantages of BiTHaC, we implement the corresponding optimizations on the client side for wolfSSL [70] version 5.7.0 and perform a series of measurements on a Raspberry Pi Zero W equipped with a 1GHz single-core CPU. In our measurement setup, client and server communicate directly via the local interface. We record the total E2E duration of handshakes, repeat measurements 100 times, and report the arithmetic mean as well as the standard deviation over these repetitions.

Savings of BiTHaC. In Fig. 6, we report on the processing improvements of BiTHaC compared to unmodified TLS. As expected, connection establishments with RSA certificates invoke a larger processing overhead than ECC certificates and longer certificate chains cause additional overhead. For BiTHaC, however, we measure a constant processing overhead for both certificate types. For RSA, this leads to a reduction of 3.3% (15 ms) to 4.9% (23 ms), while BiTHaC achieves an even larger relative reduction for the already more efficient case of ECC certificate chains, were the overhead is reduced by 5.5% (21 ms) to 8.6% (32 ms). Thus, BiTHaC not only realizes substantial bandwidth improvements, but additionally speeds-up handshakes by tens of ms, which constitutes a significant improvement for latency-critical IIoT applications [76], [77].

C. Memory Costs

While BiTHaC realizes substantial bandwidth and processing savings, these require to cache additional information. To assess whether this is feasible for constrained devices, we evaluate the memory overhead of BiTHaC.

Methodology. The evaluation setup is similar to the bandwidth evaluation (cf. Sec. V-A). However, instead of capturing network traffic, we utilize the memory profiler Bytehound [78]. Furthermore, we use the *lowresource* compilation option of wolfSSL to obtain a realistic baseline for constrained devices. With this setup, we record the maximum measured memory overhead for various events of the connection setup.

Peak Memory Usage. First, we observe that the memory overhead usually peaks while sending the *ClientKeyExchange* message. Moreover, the overhead for a single cached certificate, i.e., a client connecting only to one particular server, is almost negligible small. Specifically, the relative memory overhead amounts to 3.44% on the initial handshake and to 1.22% for subsequent handshakes. Thus, the memory overhead of BiTHaC when connecting to only one server is negligible and well-manageable even for tightly-constrained devices.

Memory Usage for Larger Caches. To assess the memory usage based on the number and type of cached certificates, we examine the memory overhead for different types (2048-bit RSA keys and 256-bit ECC keys) and numbers of cached certificates, again for chains of length 3. As shown in Fig. 7, memory usage scales linearly. More precisely, each cache entry amounts to an overhead of 420 B for RSA certificates and 241 B for ECC certificates. Considering that constrained IIoT devices connect to a small number of servers, the substantial bandwidth and processing savings clearly outweigh the resulting minor memory overhead. Furthermore, if cache size should become a limiting factor, old entries can be pruned.

Overall, BiTHaC reduces both bandwidth consumption and computational overhead of TLS handshakes, while incurring only minor costs in terms of a memory overhead.

VI. Discussion

BiTHaC substantially reduces bandwidth *and* computational overhead without impacting E2E security. By refraining from protocol-breaking changes, we ensure interoperability with legacy deployments. As both peers are authenticated and no trust in a third party is needed, strong security is assured.

Does BiTHaC weaken security? In contrast to other approaches such as offloading [23], [24], [60], [61], which require a trusted third party, or improvements at the cost of security, e.g., by foregoing perfect forward secrecy in ID-based session resumption [17], [30], [64], BiTHaC shrinks TLS handshakes while preserving security guarantees. Most importantly, BiTHaC does not change the semantics of the key establishment of the handshake. Merely the explicit transmission of bandwidth-heavy parts, particularly the certificate chain, is omitted. Still, those remain cryptographically linked to the handshake through the included tickets or fingerprints. Furthermore, both peers still prove possession of the associated private keys. Our optimized certificate cache warrants additional consideration, as the certificate chain is stored only in significantly abbreviated form (cf. Sec. IV-C). To rule out potential attack vectors, only public keys included in a verified certificate chain are added to the cache after an initial successful handshake. Thus, attack potential only arises from an attacker using a certificate chain that was compromised after it was added to the cache or through direct cache modification. As BiTHaC upholds support for revocation checks, the risks arising from compromised certificate chains are identical to TLS without caching. Likewise, direct manipulation of the cache requires an attacker to modify application memory, which also contains further security-critical information such as the session key, resulting in comparable risk to plain TLS.

Is there an increased risk for device tracking? Since BiTHaC caches certificates, which hold identifiable information, the question of privacy risks arises. However, these risks are not larger than those inherent to client certificates [79]. For our discussion, we differentiate between TLS 1.2 [30], where certificates are transmitted in clear, thus allowing for passive tracking, and TLS 1.3 [31], where additional encryption and changes in the handshake prevent passive tracking (cf. Sec. II-A). Due to these stronger privacy guarantees, we focus our discussion on TLS 1.2. To prevent tracking, BiTHaC scopes the cached client-side information (cf. Sec. IV), i.e., the client onlys transmit fingerprints and tickets associated with the current peer. Without proper scoping, e.g., naïvely sending all fingerprints in the cache to every server, clients would become sufficiently identifiable for tracking. For server-side caching, tickets are freshly generated after each connection and implicitly derived. Hence, passive correlation of TLS connections is prevented even though tickets are transmitted in clear. As this does impact the additional privacy guarantees of TLS 1.3., BiTHaC does not add any new risks over those deemed an acceptable trade-off for the use of TLS anyways.

What about compatibility with legacy devices? In contrast to other approaches to reduce the overhead of TLS which rely on protocol breaking changes (cf. Sec. III), BiTHaC ensures compatibility with legacy devices by using the built-in extension mechanism of TLS [30], [31]. Both peers signal support for BiTHaC by including a corresponding extension in their *Hello* message (cf. Sec. IV-B). Including arbitrary (new) extensions is fully protocol compliant and a peer will simply ignore unsupported extensions [30], [31]. Thus, if one peer does not signal support for BiTHaC, the connection establishment continues with a default TLS handshake.

How is BiTHaC different from RFC 7924? BiTHaC builds upon the cached information extension (RFC 7924) [28]. However, RFC 7924 merely targets to reduce the bandwidth overhead resulting from the use of *server* certificates and *CertificateRequest* messages and leaves cache structure and content to individual implementations. In contrast, BiTHaC not only adds functionality to significantly reduce the bandwidth overhead resulting from *client* certificates (whose use is often required in IIoT scenarios [8], [9]), but additionally substantially reduces the computational overhead resulting from the verification of TLS certificates on constrained IIoT devices. As such, BiTHaC even provides performance improvements over RFC 7924 when only using server authentication.

Why not use session resumption instead? The advantages of the full authentication provided by BiTHaC over ID-based session resumption used up to TLS 1.2 [30], where keys for resumed sessions are directly derived from previous secrets, are striking due to the resulting lack of PFS and limited session lifetime. For TLS 1.3, which provides PFS, BiTHaC's advantages require further consideration. Most notably, BiTHaC performs a full handshake for each connection and thus provides identical security as TLS without session resumption. Hence, while session resumption has a bandwidth advantage over BiTHaC (cf. Sec. V-A), these savings are limited to rather short session ticket lifetimes. While TLS 1.3 generally allows for a ticket lifetime of up to 7 days [31], these are significantly restricted by policies or regulations [8], [9], down to the upper limit of 24 hours for TLS 1.2 [30]. As a cache entry in BiTHaC can remain valid for the complete validity period of cached certificates, its lifetime greatly exceeds that of sessions tickets. Notably, session resumption and BiTHaC are not mutually exclusive and can thus be used together to combine short-term (session resumption) *and* long-term (BiTHaC) savings.

VII. Conclusion

By bi-directionally caching static parts of recurring TLS handshakes, BiTHaC addresses the substantial bandwidth and processing overhead of TLS, which often prevents its use in constrained IIoT environments. When leveraging BiTHaC, redundant information, e.g., certificates, neither have to be transmitted nor corresponding costly computations, such as the validation of certificates, performed repeatedly. Notably, BiTHaC is fully compatible with legacy implementations through the use of TLS extensions, allowing for incremental deployability. Finally, through our approach of extending caching to redundant computations, we particularly improve upon the computational overhead of asymmetric cryptography, while keeping the memory overhead, i.e., the inherent trade-off of any caching mechanism, minimal. Our evaluation of BiTHaC shows bandwidth savings of up to 61.1% and processing reductions of up to 8.5%, while the memory overhead remains well-manageable. Most notably and unlike other approaches such as session resumption, BiTHaC fully upholds the strong security notions of E2E security. With the ongoing deployment of post-quantum cryptography in TLS and resulting further increases in bandwidth demand, the savings offered by BiTHaC will become even more relevant.

Acknowledgments

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC-2023 Internet of Production – 390621612, the German Federal Office for Information Security (BSI) under project funding reference number 01MO23003D (PlusMoSmart), and the German Federal Ministry of the Interior represented by the German Federal Agency for Public Safety Digital Radio (BDBOS) under project funding reference number 16BEC0049 (MissionXconnect). The responsibility for the content of this publication lies with the authors.

References

[1] T.-h. Kim, C. Ramos, and S. Mohammed, "Smart City and IoT," *Future Generation Computer Systems*, vol. 76, 2017.

[2] H. Zhang and X. Lu, "Vehicle communication network in intelligent transportation system based on Internet of Things," *Computer Communications*, vol. 160, 2020.

[3] J. Ko, J. Eriksson, N. Tsiftes, S. Dawson-Haggerty, J.-P. Vasseur, M. Durvy, A. Terzis, A. Dunkels, and D. Culler, "Industry: Beyond interoperability: Pushing the performance of sensor network IP stacks," in *SenSys*, 2011.

[4] S. Lenz, D. Schachtschneider, S. Jonas, L. Tirpitz, S. Geisler, and M. Henze, "CoFacS – Simulating a Complete Factory to Study the Security of Interconnected Production," in *LCN*, 2025.

[5] L. P. Rondon, L. Babun, A. Aris, K. Akkaya, and A. S. Uluagac, "PoisonIvy: (In) secure Practices of Enterprise IoT Systems in Smart Buildings," in *BuildSys*, 2020.

[6] E. Wagner, D. Heye, M. Serror, I. Kunze, K. Wehrle, and M. Henze, "Madtls: Fine-grained Middlebox-aware End-to-end Security for Industrial Communication," in *ACM ASIA CCS*, 2024.

[7] M. Dahlmanns, J. Lohmöller, J. Pennekamp, J. Bodenhausen, K. Wehrle, and M. Henze, "Missed Opportunities: Measuring the Untapped TLS Support in the Industrial Internet of Things," in *ACM ASIA CCS*, 2022.

[8] M. Rademacher, H. Linka, J. Konrad, T. Horstmann, and K. Jonas, "Bounds for the Scalability of TLS over LoRaWAN," in *ITG MKT*, 2022.

[9] F. Heimgaertner and M. Menth, "Distributed controller communication in virtual power plants using smart meter gateways," in *ICE IEEE/ITMC*, 2018.

[10] C. Bormann, M. Ersue, and A. Keranen, "Terminology for Constrained-Node Networks," RFC Editor, RFC 7228, 2014.

[11] M. Henze, J. Hiller, S. Schmerling, J. H. Ziegeldorf, and K. Wehrle, "CPPL: Compact Privacy Policy Language," in *WPES*, 2016.

[12] J. Bodenhausen, C. Sorgatz, T. Vogt, K. Grafflage, S. Rötzel, M. Rademacher, and M. Henze, "Securing Wireless Communication in Critical Infrastructure: Challenges and Opportunities," *MobiQuitous*, 2023.

[13] M. Rademacher, H. Linka, T. Horstmann, and M. Henze, "Path Loss in Urban LoRa Networks: A Large-Scale Measurement Study," in *VTC Fall*, 2021.

[14] A. L. M. Neto, A. L. Souza, I. Cunha, M. Nogueira, I. O. Nunes, L. Cotta, N. Gentille, A. A. Loureiro, D. F. Aranha, H. K. Patil *et al.*, "AoT: Authentication and Access Control for the Entire IoT Device Life-Cycle," in *SenSys*, 2016.

[15] P. Schwabe, D. Stebila, and T. Wiggers, "More Efficient Post-quantum KEMTLS with Pre-distributed Public Keys," in *ESORICS*, 2021.

[16] A. O. Bang, U. P. Rao, A. Visconti, A. Brighente, and M. Conti, "An IoT Inventory Before Deployment: A Survey on IoT Protocols, Communication Technologies, Vulnerabilities, Attacks, and Future Research Directions," *Computers & Security*, vol. 123, 2022.

[17] V. Gupta, M. Wurm, Y. Zhu, M. Millard, S. Fung, N. Gura, H. Eberle, and S. C. Shantz, "Sizzle: A standards-based end-to-end security architecture for the embedded Internet," *Pervasive and Mobile Computing*, 2005.

[18] H. Tschofenig and T. Fossati, "Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things," RFC 7925, 2016.

[19] S. Raza, D. Trabalza, and T. Voigt, "6LoWPAN Compressed DTLS for CoAP," in *DCOSS*, 2012.

[20] E. Rescorla, R. Barnes, H. Tschofenig, and B. M. Schwartz, "Compact TLS 1.3," Internet-Draft, 2023, work in Progress.

[21] T. Kothmayr, C. Schmitt, W. Hu, M. Brünig, and G. Carle, "A DTLS based end-to-end security architecture for the Internet of Things with two-way authentication," in *LCNW - Workshops*, 2012.

[22] P. Wouters, H. Tschofenig, J. Gilmore, S. Weiler, and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)," RFC 7250, 2014.

[23] R. Hummen, H. Shafagh, S. Raza, T. Voig, and K. Wehrle, "Delegation-based authentication and authorization for the IP-based Internet of Things," in *SECON*, 2014.

[24] S. Raza, L. Seitz, D. Sitenkov, and G. Selander, "S3K: Scalable Security With Symmetric Keys—DTLS Key Establishment for the Internet of Things," *TASE*, 2016.

[25] P. Eronen, H. Tschofenig, H. Zhou, and J. A. Salowey, "Transport Layer Security (TLS) Session Resumption without Server-Side State," RFC 5077, 2008.

[26] K. Tange, D. Howard, T. Shanahan, S. Pepe, X. Fafoutis, and N. Dragoni, "rTLS: Lightweight TLS Session Resumption for Constrained IoT Devices," in *Information and Communications Security*, 2020.

[27] G. Apostolopoulos, V. Peris, and D. Saha, "Transport layer security: how much does it really cost?" in *IEEE INFOCOM '99.*, 1999.

[28] S. Santesson and H. Tschofenig, "Transport Layer Security (TLS) Cached Information Extension," RFC 7924, 2016.

[29] S. Hebrok, S. Nachtigall, M. Maehren, N. Erinola, R. Merget, J. Somorovsky, and J. Schwenk, "We Really Need to Talk About Session Tickets: A Large-Scale Analysis of Cryptographic Dangers with TLS Session Tickets," in *USENIX Security 23*, 2023.

[30] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," RFC Editor, RFC 5246, 2008.

[31] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," RFC Editor, RFC 8446, 2018.

[32] P. Kampanakis and M. Kallitsis, "Faster post-quantum TLS handshakes without intermediate CA certificates," in *CSCML*, 2022.

[33] D. Stebila, S. Fluhrer, and S. Gueron, "Hybrid key exchange in TLS 1.3," IETF, Internet-Draft draft-ietf-tls-hybrid-design-10, Apr. 2024, work in Progress.

[34] M. Eggert, R. Häußling, M. Henze, L. Hermerschmidt, R. Hummen, D. Kerpen, A. Navarro Pérez, B. Rumpe, D. Thißen, and K. Wehrle, "SensorCloud: Towards the Interdisciplinary Development of a Trustworthy Platform for Globally Interconnected Sensors and Actuators," in *Trusted Cloud Computing*, 2014.

[35] M. Serror, S. Hack, M. Henze, M. Schuba, and K. Wehrle, "Challenges and Opportunities in Securing the Industrial Internet of Things," *IEEE TII*, vol. 17, no. 5, 2021.

[36] C. Karlof, N. Sastry, and D. Wagner, "TinySec: A Link Layer Security Architecture for Wireless Sensor Networks," in *SenSys*, 2004.

[37] M. Henze, R. Hummen, R. Matzutt, D. Catrein, and K. Wehrle, "Maintaining User Control While Storing and Processing Sensor Data in the Cloud," *IJGHPC*, 2013.

[38] S. Michaelides, S. Lenz, T. Vogt, and M. Henze, "Secure Integration of 5G in Industrial Networks: State of the Art, Challenges and Opportunities," *Future Generation Computer Systems*, vol. 166, 2025.

[39] A. Brighente, J. Mohammadi, P. Baracca, S. Mandelli, and S. Tomasin, "Interference Prediction for Low-Complexity Link Adaptation in Beyond 5G Ultra-Reliable Low-Latency Communications," *IEEE Transactions on Wireless Communications*, vol. 21, no. 10, 2022.

[40] G. Restuccia, H. Tschofenig, and E. Baccelli, "Low-Power IoT Communication Security: On the Performance of DTLS and TLS 1.3," in *IFIP PEMWN*, 2020.

[41] J. Bodenhausen, L. Grote, M. Rademacher, and M. Henze, "Adaptive Optimization of TLS Overhead for Wireless Communication in Critical Infrastructure," *CSNet*, 2024.

[42] T. Kothmayr, W. Hu, C. Schmitt, M. Bruenig, and G. Carle, "Poster: Securing the Internet of Things with DTLS," in *SenSys*, 2011.

[43] M. Brachmann, S. L. Keoh, O. G. Morchon, and S. S. Kumar, "End-to-end transport security in the IP-based internet of things," in *ICCCN*, 2012.

[44] R. Behrens and A. Ahmed, "Internet of Things: An end-to-end security layer," in *ICIN*, 2017.

[45] S. Sahraoui and A. Bilami, "Compressed and distributed host identity protocol for end-to-end security in the IoT," in *NGNS*, 2014.

[46] ——, "Efficient HIP-based approach to ensure lightweight end-to-end security in the internet of things," *Computer Networks*, vol. 91, 2015.

[47] J. Granjal and E. Monteiro, "Adaptable End-To-End Security For Mobile IoT Sensing Applications," in *SafeThings*, 2017.

[48] J. Plusquellic, E. E. Tsiropoulou, and C. Minwalla, "Privacy-Preserving Authentication Protocols for IoT Devices Using the SiRF PUF," *IEEE Transactions on Emerging Topics in Computing*, vol. 11, no. 4, 2023.

[49] S. Raza, H. Shafagh, K. Hewage, R. Hummen, and T. Voigt, "Lithe: Lightweight Secure CoAP for the Internet of Things," *IEEE Sensors Journal*, 2013.

[50] U. Banerjee, C. Juvekar, S. H. Fuller, and A. P. Chandrakasan, "eeDTLS: Energy-Efficient Datagram Transport Layer Security for the Internet of Things," in *GLOBECOM*, 2017.

[51] S. Boeyen, S. Santesson, T. Polk, R. Housley, S. Farrell, and D. Cooper, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," RFC 5280, 2008.

[52] D. McGrew and M. Pritikin, “The Compressed X.509 Certificate Format,” Internet-Draft, 2010, expired.

[53] A. Ghedini and V. Vasiliev, “TLS Certificate Compression,” RFC 8879, 2020.

[54] D. A. Ortiz-Yepes, “Optimizing TLS for Low Bandwidth Environments,” in *Foundations and Practice of Security*, 2015.

[55] J. Höglund, S. Lindemer, M. Furuhed, and S. Raza, “PKI4IoT: Towards public key infrastructure for the Internet of Things,” *Computers & Security*, 2020.

[56] J. P. Mattsson, G. Selander, S. Raza, J. Höglund, and M. Furuhed, “CBOR Encoded X.509 Certificates (C509 Certificates),” Internet-Draft, 2024, work in Progress.

[57] W. Jung, S. Hong, M. Ha, Y.-J. Kim, and D. Kim, “SSL-Based Lightweight Security of IP-Based Wireless Sensor Networks,” in *AINA Workshops*, 2009.

[58] H. Tschofenig, T. Fossati, and M. Richardson, “TLS/DTLS 1.3 Profiles for the Internet of Things,” Internet-Draft, 2023, work in Progress.

[59] D. E. Eastlake 3rd, “Transport Layer Security (TLS) Extensions: Extension Definitions,” RFC 6066, 2011.

[60] S. Fouladgar, B. Mainaud, K. Masmoudi, and H. Afifi, “Tiny 3-TLS: A Trust Delegation Protocol for Wireless Sensor Networks,” in *ESAS*, 2006.

[61] T. Polk, D. Cooper, R. Housley, A. N. Malpani, and T. Freeman, “Server-Based Certificate Validation Protocol (SCVP),” RFC 5055, 2007.

[62] M. Henze, R. Hummen, R. Matzutt, and K. Wehrle, “A Trust Point-based Security Architecture for Sensor Data in the Cloud,” in *Trusted Cloud Computing*. Springer, 2014.

[63] R. Hummen, J. H. Ziegeldorf, H. Shafagh, S. Raza, and K. Wehrle, “Towards viable certificate-based authentication for the internet of things,” ser. HotWiSec ’13, 2013.

[64] T. S. Sobh, A. Elgohary, and M. Zaki, “Performance improvements on the network security protocols,” *International Journal of Network Security*, 2008.

[65] R. Hummen, J. Gilger, and H. Shafagh, “Extended DTLS Session Resumption for Constrained Network Environments,” Internet-Draft, 2013, expired.

[66] H. Shacham, D. Boneh, and E. Rescorla, “Client-Side Caching for TLS,” *ACM TISSEC*, 2004.

[67] A. Langley, “Transport Layer Security (TLS) Snap Start,” Internet-Draft, 2010, expired.

[68] M. Schukat, “Securing Critical Infrastructure,” in *DT*, 2014.

[69] S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, and D. C. Adams, “X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP,” RFC 6960, 2013.

[70] “wolfSSL: Embedded TLS Library for Applications, Devices, IoT, and the Cloud,” https://github.com/wolfSSL/wolfssl.

[71] “Docker: Accelerated Container Application Development,” Available online: https://www.docker.com/.

[72] “tcpdump: Command-line packet analyzer,” Available online: https://www.tcpdump.org/.

[73] C. Gomez, J. Crowcroft, and M. Scharf, “TCP Usage Guidance in the Internet of Things (IoT),” RFC 9006, Mar. 2021.

[74] Information Sciences Institute, University of Southern California, “Internet Protocol,” RFC 791, Sep. 1981.

[75] “IEEE Standard for Ethernet,” *IEEE Std 802.3-2022 (Revision of IEEE Std 802.3-2018)*, vol. IEEE 802.3-2022, pp. 1–7025, 2022.

[76] J. Hiller, M. Henze, M. Serror, E. Wagner, J. N. Richter, and K. Wehrle, “Secure Low Latency Communication for Constrained Industrial IoT Scenarios,” in *IEEE LCN*, 2018.

[77] S. Michaelides, J. Mucke, and M. Henze, “Assessing the Latency of Network Layer Security in 5G Networks,” in *WiSec*, 2025.

[78] “Bytehound - a memory profiler for Linux,” Available online: https://github.com/koute/bytehound.

[79] L. Foppe, J. Martin, T. Mayberry, E. C. Rye, and L. Brown, “Exploiting TLS Client Authentication for Widespread User Tracking,” *PETS*, 2018.