TLS solutions for WSNs

Philipp Lowack
Betreuer: Corinna Schmitt
Seminar Sensorknoten: Betrieb, Netze und Anwendungen SS2012
Lehrstuhl Netzarchitekturen und Netzdienste, Lehrstuhl Betriebssysteme und Systemarchitekturen
Fakultät für Informatik, Technische Universität München
Email: lowack@in.tum.de

ABSTRACT

As most nodes of wireless sensor networks are not very powerful in regard to computational operations and additionally very limited in power supply, it is not a trivial task to deploy security features. As more applications for wireless sensor networks are developed, it becomes more and more important and necessary to also provide secure communication between nodes and to external terminals. In this paper, three already existing approaches for using the Transport Layer Security protocol in combination with wireless sensor networks are presented. The first uses identity based cryptography and bilinear pairing based on elliptic curve cryptography instead of RSA. The second solution modifies the TLS handshake in order to establish a single TLS session between multiple entities. In the third solution expensive cryptographic operations are outsourced to a more powerful gateway node. For each solution the principles of operation are explained and afterwards the presented approaches are compared in regard to applicability and energy consumption.

Keywords

Wireless Sensor Networks, secure communication, Transport Layer Security, bilinear pairing, Elliptic Curve Cryptography, Tiny 3-TLS

1. INTRODUCTION

The `Transport Layer Security (TLS)` protocol is a widespread and well accepted cryptographic protocol for encrypting and integrity-protecting the payload of network data streams. Because of its transparency in regard to the outer and inner protocols, it is nowadays widely used in many different applications. The perhaps most well-known protocol which uses TLS is the `Secure HyperText Transport Protocol (HTTPS)` which is used by everyone while surfing in the Internet. But TLS can be used with almost any kind of protocol and offers sophisticated security for simple private websites as well as for large commercial banking systems.

The question for researchers is: is this protocol also suitable for embedded applications with very limited computing power and where the power consumption of algorithms is an essential criterion?

In this paper three separate already existing approaches are analyzed and compared in regard to their suitability for embedded platforms. To accomplish this task, at first in Section 1.1 a short motivation is given why it is not trivial to find a solution. Then an introduction to TLS is given in Section 1.2, explaining the basic working principles. Then the three papers are presented in Section 2 and each one is explained. The solution presented in Section 2.1 uses identity based cryptography and bilinear pairing to reduce the number of packets for the TLS handshake. The second solution, presented in Section 2.2, adapts the TLS handshake protocol to be able to establish a session between multiple entities. The last solution, presented in Section 2.3, makes the use of TLS feasible on embedded platforms by offloading as much cryptographic operations as possible to a more powerful gateway node without interrupting the security model. The last Section 3 of this paper compares the previously presented approaches in regard to applicability and energy consumption, followed by a conclusion.

1.1 Motivation

`Wireless Sensor Networks (WSN)` are networks of embedded systems which communicate via a wireless network. These embedded systems, called `nodes`, are usually limited in computing power as well as in power supply because only microprocessors with battery supply are used. The nodes are deployed in an area to collect sensor measurements which are then transfered to a central entity. The data is routed through the network via multiple nodes and finally arrives at the central entity, called gateway node, which translates between the wireless network and an external network, for example the Internet, in order to send the data to its destination or allow an operator to manage the WSN. The gateway node is usually not as limited as the sensor nodes and may even have a static power supply.

If sensitive sensor data is collected, for example health-data of patients in a hospital or security information of an alarm system, it is desirable to protect this data against unwanted disclosure and unauthorized modifications. In order to achieve this goal encryption algorithms and message authentication codes can be used.

Of course, there are many protocols available for this purpose. Renowned protocols are `IPsec`[9] and `Transport Layer Security (TLS)`[5]. The disadvantage with `IPsec` is that it encrypts the network layer payload and therefore also non-critical data like for example TCP handshakes and adds more extra size to the original packet than TLS [1]. As the computing power of embedded devices is strictly limited, this additional overhead is undesirable. TLS however works between the transport and the application layer and therefore reduces said overhead. It offers privacy and data integrity and, by using asymmetric key cryptography, also can authenticate the sender of a message. Additionally, TLS is transparent to the operating system and the network and can be used to encapsulate any other protocol.

When implementing and deploying encryption in a WSN the two key points to consider are the computational impact and the implicated power consumption of encryption algorithms. Therefore not all available algorithms are applicable. But this does not only concern the encryption of the payload data, but most importantly the costly operations to do public-key cryptography when initializing a session. Another very important aspect to consider is the number of packets to send. Transmitting data via a radio link consumes more energy the stronger the signal is. If the distribution of the sensor nodes is very sparse and therefore the signal has to be rather strong, transmitting more data strongly affects the power consumption. But also if the nodes are distributed rather dense and therefore the signal does not have to be that strong, it is still desirable to limit the amount of data to send.

1.2 Introduction to TLS

The goal of TLS is to provide a secure communication channel between two applications. The established channel ensures confidentiality and data integrity and also offers replay protection.

The `Secure Socket Layer (SSL)` was originally developed by Netscape and the first draft was released to the public in 1995 [8]. In 1999, the `Transport Layer Security (TLS)` protocol was released as an upgrade to SSL 3.0, but because of "significant changes" was not interoperable with SSL [4]. The latest version of TLS is 1.2 and has been released in 2008 [5].

TLS itself is two-layered and consists of multiple subprotocols. The `TLS Record Protocol` builds the lower layer of TLS and is responsible for transporting any data, may it be of other TLS sub-protocols or of higher level protocols. This layer is also responsible for encrypting and integrity protecting the data. As asymmetric cryptographic operations are very expensive, the TLS Record Protocol uses a symmetric algorithm for the bulk encryption of the data.

The most important sub-protocol of the upper TLS layer is the `TLS Handshake Protocol`. It is responsible for establishing a session by negotiating security parameters and can also be used to authenticate the two endpoints. Other protocols of the upper layer are the `Change Cipher Spec Protocol` (switches the currently used algorithms and keys), the `Alert Protocol` (communicates alerts to the remote endpoint) and the `Application Data Protocol` (encapsulates higher level data).

Because of the huge number of available encryption and hash algorithms, TLS uses so called `cipher suites` to organize them. A cipher suite is a set of four algorithms, namely a key-exchange algorithm, a bulk-encryption algorithm, a message authentication code and a pseudorandom function. The key-exchange algorithm determines how the server and the client authenticate each other during the session initialization. The bulk-encryption algorithm is used by the `TLS Record Protocol` to encrypt the data. The message authentication code algorithm is also used by the `TLS Record Protocol` to protect the data's integrity. The pseudorandom function is a cryptographically secure pseudorandom number generator used for generating keying material.

TLS is mostly used along with asymmetric cryptography and certificates. When a `Public Key Infrastructure (PKI)` is not available or authentication is not necessary the Diffie-Hellman key exchange algorithm (or one of its variants) can be used instead of public and private keys [5]. But as this technique is vulnerable to man-in-the-middle attacks it is barely ever used.

The following explanation of the Handshake Protocol is based on the use of public key cryptography with certificates. In order to prevent the explanation from getting to complex, it is limited to essential elements which are necessary to understand TLS itself and the improvements that will be presented. Therefore some messages are omitted. In order to provide a consistent naming scheme, Table 1 defines the used symbols and their meaning.

Symbol	Description
{x}_K	encryption of message x with key K
a|b	concatenation of messages a and b
Pub-x, Pub_x	the public key of entity x
Cert-x, $Cert_x$	the certificate of entity x (includes Pub_x)
N-x, N_x	nonce choosen by entity x
ECDH-x, $ECDH_x$	the public ECDH values of entity x
E-q, E_q	an elliptic Curve
P	a point on E_q

Table 1: Variables used in this paper

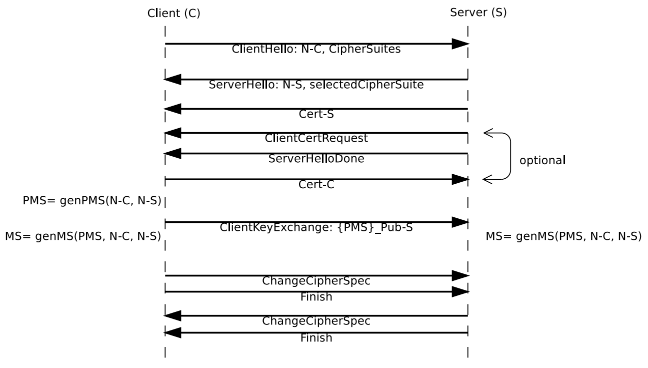

Figure 1: Standard TLS handshake using asymmetric cryptography. [5]

The basic message flow for establishing a TLS session is depicted in Figure 1. As TLS works with exactly two endpoints, the one establishing the connection is named `client`, the other one is named `server`. The client initiates the hello phase of the `TLS Handshake Protocol` by sending an (unencrypted) `ClientHello` message to the server. This message contains mainly a random number (named `nonce`, number only used once) and all supported cipher suites of the client.

The server also generates a nonce, chooses an acceptable cipher suite out of the received ones and sends these values in a `ServerHello` message to the client. The server also sends his certificate, which contains his private key, to the client. If the client needs to be authenticated, the server also sends a requests for the client's certificate. The `ServerHelloDone` message indicates the end of the hello phase. If the client's certificate has been requested by the server, the client will send it to the server upon reception of the `ServerHelloDone` message.

Now begins the authentication phase. In order to ensure authenticity, the received certificates are now verified. If the certificates can be verified, the client generates a so called `pre-master secret` from both random numbers, encrypts it with the servers public key and sends it in a `ClientKeyExchange` message to the server. The server can decrypt the pre-master secret with its private-key. As both parties now are in the possession of the pre-master secret and both random numbers, they both can generate the `master secret` from these values. The master secret is used for generating the session keys for the encryption of all further data.

In the finish phase, the protocol switches to the new cipher suite and session keys. The client sends a `change cipher spec` message to signal that the negotiated parameters are valid and should be used now. The server also answers with a `change cipher spec` message and now expects encrypted messages. In order to verify that the encryption works correctly, both endpoints send a `finish` message to each other and verify that they are able to decrypt the received message.

In regard to WSN, there are also some disadvantages with TLS. One problem is that TLS oftentimes uses RSA for asymmetric cryptography as this algorithm is considered very secure. But the computational impact of the RSA algorithm is very high, resulting in high energy consumption when encrypting or decrypting data. Though with the recent progress on `Elliptic Curve Cryptography (ECC)` this problem has become less severe. A 224-bit key used with ECC is considered as secure as a 2048-bit key used with RSA and at the same time ECC is as fast as the RSA public-key operation and even outperforms the RSA private-key operation by one order of magnitude [7]. With longer keys this results in an even greater advantage for ECC. Another problem with TLS is the number of packets in the handshake. The standard handshake with mutual authentication needs 13 packets. As the radio module needs a relatively large amount of energy when transmitting a signal, even saving only a few packets per handshake results in a significant energy saving, as the number of performed handshakes can be very large. Designing power-aware protocols is therefore very important for WSNs [2]. All these problems combined make it non-trivial to implement TLS-based security on such limited hardware as it is used in WSNs.

2. TLS SOLUTIONS FOR WSN

In the following part three already existing solutions for securing communication in a WSN using TLS are presented. The first one uses identity based cryptography and bilinear pairing in order to reduce the number of sent messages. It is followed by an approach which enables TLS to establish a shared session between multiple parties. The third solution, called Tiny 3-TLS, offloads expensive cryptographic operations from the sensor node to the gateway node in order to allow for very performance limited sensor nodes. All solutions will be compared in Section 3.

2.1 Adapting TLS for Identity Based Cryptography

When using public-key cryptography while also allowing peer to peer communication, a certificate for each node is necessary. Additionally the whole certificate path to the root certificate authority needs to be known in order to verify a certificate. All these certificates need to be stored on each node and occupy precious storage.

Mzid et al. describe in [12] how to modify the TLS handshake protocol to support identity based cryptography in an IP-based WSN in order to save this space.

`Identity Based Cryptography (IBC)` is based on ECC and eliminates the need for certificates [13]. In the context of WSNs the (public) IP address of a node can be used as its public key. Instead of a PKI only a `Private Key Generator (PKG)` is needed, which takes the role of the Trusted Third Party. For each node, its private key is generated by the PKG and then, bundled with some further parameters like the elliptic curve itself as well as a point P on that curve, transferred to and stored on the node before deployment. The key must only be known to the node and the PKG. All other parameters may be publicly known and the elliptic curve even has to be the same for all nodes. Using IBC together with elliptic curve based Diffie-Hellman key exchange (`ECDH` [11]) as shown in Figure 2, it is now possible to reduce the number of necessary handshake protocol packets from 13 to 10.

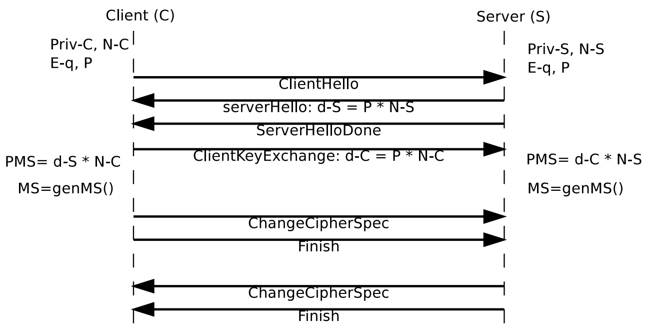

Figure 2: TLS handshake using IBC and ECDH. [12]

The second proposed improvement to the handshake protocol is the use of bilinear pairing, which is the basis for a different key exchange algorithm. Let G_1 denote a cyclic additive group of some large prime order $q = 2^m$ and G_2 denotes a cyclic multiplicative group of the same prime order. Then a map $e : G_1 \times G_1 \mapsto G_1$ is called a pairing. This map has the very special property of bilinearity: if $P, Q \in G_1$ and $a \in Z_q^*$ then $e(aP, Q) = e(P, aQ) = e(P, Q)^a$. More details can be found in reference [10]. This mathematical construct enables two parties to generate a shared secret without any interaction between them.

In order to use bilinear pairing, two separated phases are considered. In the pre-deployment phase, the PKG chooses several parameters:

- an elliptic curve E_q,
- $G_1 \subseteq E_q$ and G_2,
- a bilinear map e as described above,
- a hash function H which returns points of the elliptic curve, and
- a random number $S \in Z_q^*$.

Except S all values can be publicly known. The PKG generates the private key of a node calculating $Priv_i = S * H(IP_i)$ where IP_i is the IP address of node i. To finish the pre-deployment phase, the tuple $< G_1, G_2, e, q, Priv_i, H >$ is transfered to and stored on the nodes.

In the post-deployment phase, two nodes can generate the pre-master secret as soon as they know the address of the remote end by calculating:

- on the server: $e(H(IP_{client}), Priv_{server})$
- on the client: $e(Priv_{client}, H(IP_{server}))$

As both parties are in possession the remote IP address after the ServerHello message[1], the number of necessary packets can be reduced from 13 to only 7. Figure 3 illustrates the improved handshake.

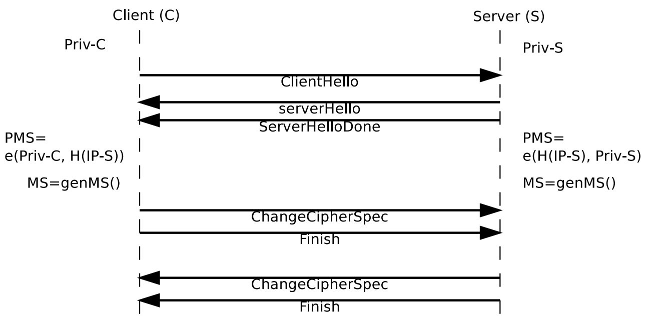

Figure 3: TLS handshake using bilinear pairing. [12]

Mzid et al. not only propose the improvements but also evaluates them in regard of storage costs, handshake duration and energy consumption [12]. The storage costs are significantly reduced by the use of IBC as no certificates need to be stored. Each node only has to store its private key as the public key can be derived from an identity. The duration of the handshake is especially important for real-time applications but it is not only then desirable to reduce this delay. Both improvements reduce the duration of the handshake. And when using both server and client authentication, the improvement is almost by an order of magnitude as two certificate verification operations are needless. Also both improvements reduce the energy consumption of the handshake by an order of magnitude.

[1]Client and server have first to agree on a cipher suite that uses bilinear pairing.

2.2 Using a single TLS session for multiple entities

Badra proposes a solution to establish a TLS session between more than two entities [3]. The new behavior is implemented by a new set of cipher suites which reuse existing RSA based cipher suites. Solely the TLS handshake protocol is modified to establish a session between multiple entities. As all entities then have the session keys, they can freely communicate with each other.

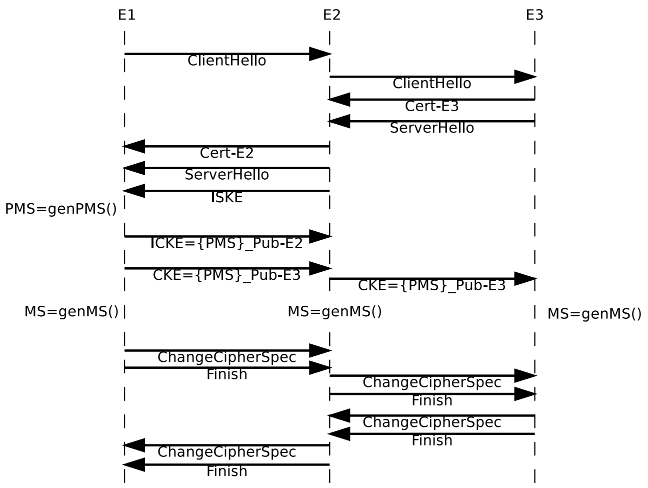

Figure 4: Handshake between three entities. [3]

For the following part, three entities are considered, but the protocol can be easily adapted to any number of entities. The basic message flow is depicted in Figure 4. The modified handshake happens as follows: The first entity (E_1) connects to the second entity (E_2) and sends a ClientHello message which includes one or more of the new cipher suites in the list of supported cipher suites and a random number. If E_2 supports the new cipher suites, then one of them is selected. E_2 now sends a ClientHello message to the third entity (E_3) which includes the selected cipher suite and the same random number as received by E_1.

If the cipher suite sent by E_2 is accepted by E_3, it sends a ServerHello message containing a session identifier, a new random value and the selected cipher suite to E_2. E_2 and E_3 now finish the hello phase and E_3's ServerHello message as well as its certificate are forwarded by E_2 to E_1. In addition, the new `IntermediateServerKeyExchange (ISKE)` message is also sent to E_1. It contains a list of all host names involved in the session and E_2's certificate.

In the authentication phase, E_1 generates a pre-master secret. The pre-master secret is then encrypted once with E_3's public key to get the `EncryptedPreMasterSecret` and once with E_2's public key to get the `InterEncryptedPre-MasterSecret`. The InterEncryptedPreMasterSecret is sent via the new `IntermediateClientKeyExchange (ICKE)` message to E_2 and the EncryptedPreMasterSecret is sent to E_2 via the `ClientKeyExchange (CKE)` message, which also includes the previously received list of involved host names. Also, E_1 immediately sends its finish message.

E_2 can now compute the master-key from the pre-master secret included in the IntermediateClientKeyExchange message and therefore verify E_1's certificate, signature and finish message. All messages except the IntermediateClientKeyExchange message are forwarded to E_3. E_3 can now decrypt the ClientKeyExchange message, compute the master key and also verify E_1's certificate, signature and finish message. If these operations are successful, a finish message is sent to E_2 which forwards it to E_1. After E_1, and optionally also E_2, verified E_3's finish message, the TLS handshake is complete and all involved entities are in possession of the master key. If one of the entities now wants to communicate with any other involved entity, it just sends a ClientHello message containing the previously stored session identifier to the other entity[2]and is able to securely communicate with the other entity without performing a full handshake.

This design is fully backwards compatible with standard TLS and even allows to communicate with non-modified clients. Furthermore this approach theoretically performs better than establishing separate TLS sessions, as for N entities at best $N-2$ signature operations, $N-2$ certificate verification operations and $N-2$ signature verification operations are saved.

2.3 Tiny 3-TLS

The goal of "Tiny 3-TLS" as presented in [6] is to offload expensive cryptographic operations to the more powerful gateway node in order to relieve the sensor nodes without putting the security at risk. The network model of the paper is as follows. A WSN consisting of multiple sensor nodes is connected to an external network via a gateway acting as reverse proxy. A so called "remote monitor" R wants to establish a secure connection to one of the sensor nodes S. Each sensor node has a pre-shared key K which is known to the gateway G. In order to ensure confidentiality, each sensor node should have a different pre-shared key.

Two distinct modes of operation are presented which differ in the trust-level of the gateway. If the gateway is fully trusted, it can handle all of the cryptographic operations of the TLS handshake and then provide the sensor node with the actual session keys. That way, the gateway is also in possession of the secrets. If the gateway is only partially trusted, then the gateway does handle the asymmetric encryption and decryption of the TLS-handshake packets but the generation of the session keys happens at the remote monitor and the sensor node via ECDH. This way, the gateway does not know the session keys. In both cases, all three parties are mutually authenticated. The gateway and the sensor nodes of the WSN authenticate each other by proving the knowledge of the pre-shared key K. The gateway and the remote endpoint authenticate each other via certificates. The trust between the remote endpoint and the sensor node is then transitively achieved. If on the other hand the gateway is not trusted at all, for example because it is either unknown or publicly available, this solution is not applicable. An untrusted or even malicious gateway could easily perform a man-in-the-middle (MITM) attack and successfully capture or manipulate any data.

[2]TLS calls this a resumed handshake.

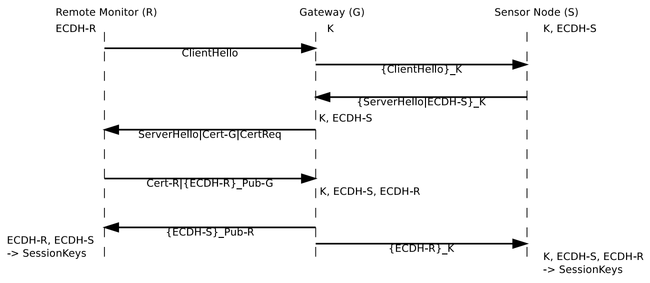

Figure 5: Tiny3-TLS with a partially trusted gateway. [6]

The message flow in case of a partially trusted gateway is depicted in Figure 5. When the remote monitor wants to communicate with a sensor node it sends a ClientHello message, which is encrypted with the appropriate pre-shared key by the gateway and relayed to the sensor node. The node answers with a ServerHello message encrypted by K and also its public ECDH values. The gateway decrypts the message, keeps the ECDH values $ECDH_S$ and forwards the ServerHello message to the remote monitor. It also adds its own certificate and a request for the certificate of the remote monitor. The remote monitor validates the gateway's certificate and, if valid, answers with his own certificate and his public ECDH values $ECDH_R$, both encrypted with the gateway's public key. If the gateway can validate the remote monitor's certificate it sends $ECDH_R$, encrypted with K, to the sensor node and $ECDH_S$, encrypted by the remote monitor's public key, to the remote monitor. Now the sensor node and the remote monitor can generate the session keys from $ECDH_R$ and $ECDH_S$ and can encrypt all further direct communication.

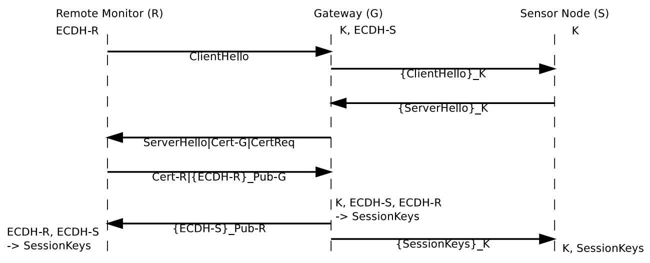

Figure 6: Tiny 3-TLS with a fully trusted gateway. [6]

In case of a fully trusted gateway, the handshake happens as before, but this time $ECDH_S$ is not selected by the sensor node but directly by the gateway. The message flow is depicted in Figure 6. That way the gateway itself can generate the session keys without any interaction of the sensor node. Once the session keys are generated, the gateway encrypts them with the pre-shared key and sends them to the sensor node.

This approach can, if applicable, reduce the load on the sensor node drastically. Instead of expensive asymmetric cryptographic operations like signature verifications only rather inexpensive symmetric encryption and decryption operations are necessary. But this comes at the cost that the gateway

	Protocols			
	TLS	Single-TLS-Session	Tiny-3TLS	IBC
Uses certificates	x	x	x	-
Based on	RSA	RSA	ECC	ECC
usable for P2P	x	x	-	x
Ext. $\rightarrow$ Node	x	x	x	x
Performance	13 messages	saves: $N-2$ signature operations $N-2$ signature verifications $N-2$ certificate verifications	GW: 7 message Node: 3 messages	7 messages

Table 2: Comparison of the presented solutions

node has a much higher load.

3. CONCLUSION

All three solutions presented in this paper help to secure communication in a WSN environment. The improvements presented in Section 2.1 reduce the number of necessary packets for a TLS handshake and also eliminate the need for certificates and a PKI. This also significantly improves the energy consumption of the nodes, enabling them to operate for a longer period of time, or even making it possible to lower the hardware requirements. Establishing a single TLS session for multiple nodes as described 2.2 also saves packets and therefore energy. But as that paper did not have embedded systems in mind, power consumption was not a key point of the improvement. This approach is also only applicable when it is desirable to share a single set of session keys for multiple entities. But in an environment where data streams of different nodes must be protected from each other, for example sensor nodes collecting health-data from patients in a hospital, it is not desirable that all nodes share the same keys. Finally, Tiny 3-TLS, presented in Section 2.3, only secures the communication with an external remote terminal, but does not provide secure point to point communication within the WSN.

Tiny 3-TLS and the paper about establishing a single TLS session for multiple entities have each a special use-case in mind for which they try to find an optimal solution. Therefore they do not directly compete with each other as different environments and use-cases require different solutions. As for IBC and bilinear pairing, these approaches can help the former solutions to save more energy and other resources and therefore further boost them.

References

[1] A. Alshamsi and T. Saito. A technical comparison of ipsec and ssl. In *19th International Conference on Advanced Information Networking and Applications, 2005. AINA 2005.*, volume 2, pages 395–398. IEEE, 2005.

[2] S. Aslam, F. Farooq, and S. Sarwar. Power consumption in wireless sensor networks. In *Proceedings of the 7th International Conference on Frontiers of Information Technology*, page 14. ACM, 2009.

[3] M. Badra. Securing communications between multiple entities using a single TLS session. In *New Technologies, Mobility and Security (NTMS), 2011 4th IFIP International Conference on*, pages 1–4. IEEE, 2011.

[4] T. Dierks and C. Allen. The TLS protocol. IETF RFC 2246, January 1999.

[5] T. Dierks and E. Rescorla. The transport layer security (TLS) protocol, version 1.2. IETF RFC 5246, August 2008.

[6] S. Fouladgar, B. Mainaud, K. Masmoudi, and H. Afifi. Tiny 3-TLS: A trust delegation protocol for wireless sensor networks. In L. Buttyán, V. Gligor, and D. Westhoff, editors, *Security and Privacy in Ad-Hoc and Sensor Networks*, volume 4357 of *Lecture Notes in Computer Science*, pages 32–42. Springer Berlin / Heidelberg, 2006. ISBN 978-3-540-69172-3.

[7] N. Gura, A. Patel, A. W, H. Eberle, and S. C. Shantz. Comparing elliptic curve cryptography and RSA on 8-bit CPUs. 2004.

[8] K. E. Hickman. The SSL protocol. Netscape Communications Corp., Feb 1995.

[9] S. Kent and K. Seo. Security architecture for the internet protocol. IETF RFC 4301, December 2005.

[10] D. Meffert. Bilinear pairings in cryptography. Master's thesis, Radboud Universiteit Nijmegen, 2009.

[11] V. Miller. Use of elliptic curves in cryptography. In H. Williams, editor, *Advances in Cryptology — CRYPTO '85 Proceedings*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426. Springer Berlin / Heidelberg, 1986. ISBN 978-3-540-16463-0.

[12] R. Mzid, M. Boujelben, H. Youssef, and M. Abid. Adapting TLS handshake protocol for heterogenous IP-based WSN using identity based cryptography. In *International Conference on Communication in Wireless Environments and Ubiquitous Systems: New Challenges (ICWUS), 2010*, pages 1–8. IEEE, 2010.

[13] A. Shamir. Identity-based cryptosystems and signature schemes. In *Advances in cryptology*, pages 47–53. Springer, 1985.