TLS Solutions for Wireless Sensor Networks

Sebastian Wöhrl
Betreuer: Corinna Schmitt
Seminar Future Internet SS2012
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: sebastian.woehrl@mytum.de

ABSTRACT

Wireless Sensor Networks are an interesting research topic with many possible real world applications. The increasing number of sensor networks and their widespread deployment throughout the world makes them a more and more interesting target for attackers. With the Internet Protocol (IPv6) becoming the standard for communication among these networks it is possible to use established standards for this task. The current standard for secure communication in the internet is Transport Layer Security (TLS) which is used worldwide and easy to implement. This paper discusses several solutions for and enhancements of TLS to use it in IP-based Wireless Sensor Networks to leverage the power of TLS while still keeping in mind the limited resources of sensor networks.

Keywords

TLS, SSL, IPv6, Security, Wireless Sensor Network

1. INTRODUCTION

A Wireless Sensor Network (WSN) is a network of small autonomous sensor nodes (usually with an embedded processor and therefore with low computing power) which are communicating using wireless connections. The application area of such sensor networks is usually monitoring external conditions ranging from physical to environmental values[11]. A relativly new field of application is using the sensors to monitor medical readings of human patients[12].

With the internet and its primary protocol (IP) being the worldwide standard for data communication it is only logical to also build WSNs based on this standard. This became possible with the upcoming of IPv6 and its greatly enlarged address pool making it possible to create IP-based networks for sensor nodes with each sensor having its own IP-Address. This makes it possible to integrate these sensor networks into the internet and using established standards.

Due to the nodes communicating wirelessly special considerations have to be made for securing these links. With TLS/SSL being **the** standard protocol for encrypting and authenticating connections in IP networks it is only logical to also use it for IP-based WSNs. Of course with the sensor nodes being very limited in terms of energy and computing power and considering the special circumstances of the deployment of such networks just using plain TLS will not be very efficient.

This paper will describe and discuss several approaches to this topic and is organized as follows. Section 2 describes "original" TLS based on the official RFC. Sections 3 to 5 will discuss possible approaches for using TLS with more than two entities[1], for using identity-based cryptograhy with TLS[2] and finally Tiny-3-TLS[3]. Finally, the solutions are compared and summarized in Section 6.

2. TLS FOLLOWING RFC 5246

The current version of TLS, 1.2 (or SSL 3.1), is defined in RFC 5246[1] [4]. TLS consists of a series of protocols. The basic protocol is the so called "TLS Record Protocol" which in the ISO/OSI-Layer-Model is put directly above the Transport Layer (c.p. Figure 1).

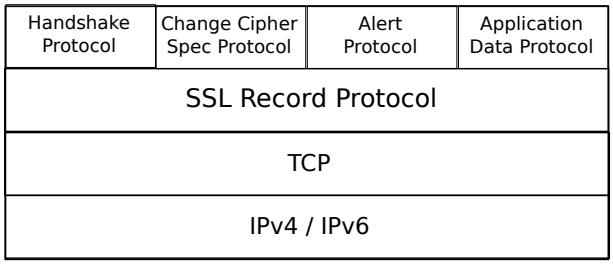

Figure 1: TLS in the ISO/OSI-Model [2]

Above the record protocol are the Handshake Protocol, the Change Cipher Spec Protocol (used to negotiate key and algorithm changes), the Alert Protocol (to signal problems) and the Application Data Protocol (used to transport the user data). The most interesting is the Handshake Procotol because it is used to establish the TLS session and to negotiate the session parameters like encryption keys and algorithms. As such it provides the most starting points to optimize the TLS protocol.

Following is a description of a normal TLS Handshake between a Client and a Server using X.509 Certificates as shown in Figure 2.

The connection starts with the client sending a ClientHello (1) containing a random number (*rnc*) and his supported algorithms (the Cipher Suites). The random number *rnc* is generated and transmitted to protect against replay attacks. The Server responds with a ServerHello (2) also containing

[1]For a more lightweight reading you can refer to [6]

doi: 10.2313/NET-2012-08-1_06

a random number (rns) and the Cipher Suite the server has chosen from the offered ones. The server will also send its certificate (3) and will optionally request a certificate from the client (4) before ending the Hello phase (5). The client checks the transmitted server certificate and transmits its own certificate (6).

With certificate checks done the client generates a new random number (called the pre-master-secret, **PMS**, 7) and sends it to the server (8) encrypted with the servers public key which is contained in the server certificate. Using this pre-master-secret both calculate the master-secret (10, **MS**) using a pseudo-random-function specified in the cipher suite. Earlier versions of TLS used the MD5 and SHA-1 hash functions.

Also the client proves it really is in control of the private key for which he has presented a certificate by sending a CertificateVerify message (9) which is a signature over all previously exchanged messages using the clients private key.

Once the master-secret is calculated both parties send a ChangeCipherSpec message (11, 13) and a Finished message (12, 14) and from there on encrypt all messages using this master secret. The Finished messages (11, 13) are already encrypted and contain a hash and a Message Authentication Code (MAC) over all previously exchanged messages. If the other party cannot decrypt the Finished message or the hash or MAC verification fails then the entire handshake is considered a failure and the connection closed.

Encryption of messages during data transport is usually done using AES. In Wireless Sensor Networks AES-128 is used most often as it provides the best trade-off between security and computational effort. As AES is already quite efficient most TLS optimizations for Wireless Sensor Networks focus on the TLS Handshake.

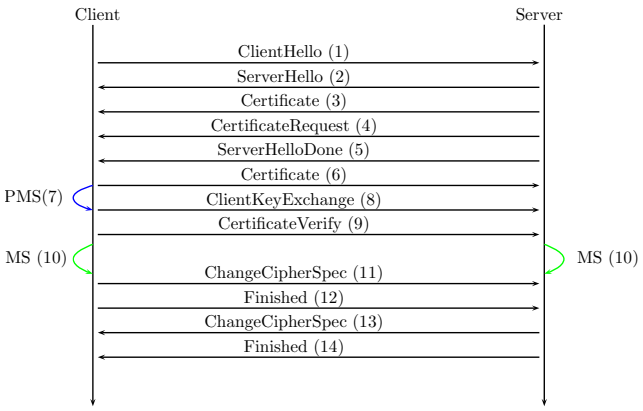

Figure 2: Time-schematic TLS Handshake[6]

3. TLS FOR MORE THAN TWO ENTITIES

Mohamad Badra describes in [1] a way to expand the original TLS protocol to use it for more than two entities and therefore to no longer be bound by the client/server-architecture of the original design.

3.1 Why more than two?

Most of the communication in the internet is done between two entities, either in a client/server-mode or as P2P (peer-to-peer). As client/server is the standard, TLS was designed for that use case. Wireless Sensor Networks are often built with several layers. The sensor nodes communicate with each other and over possibly several hops with a router node. These in turn communicate with a gateway node which connects the WSN to the internet and allows clients in the internet - e.g. a lab computer controlling the nodes and checking the sensor readings - to communicate with the sensor nodes (for details see [9] and [11]).

The gateway node can act as a caching proxy to reduce the number of messages to the sensor nodes if many clients want to pull the same readings or to log the readings for futher study. But with an encrypted connection between the sensor and the client the proxy has no way to read the content of the messages. Therefore a way must be found to allow a secure connection between three entities. This can be generalized to N entitites, one client, one server and N-2 intermediaries.

3.2 Expanding TLS

A naive approach at solving this problem would be to have each of the entities maintain N-1 seperate connections and send each message to all the other nodes. It is quite clear that this is not the best approach. Mohamad Badra describes "an enhanced way to establish a TLS session between N entities. To simplify [he] describe[s] [his] solution with N=3"[1]. The following section is based on his paper (Section IV).

The basic extension is that all intermediate entities are told the pre-master-secret so they can compute the master-secret and therefore decrypt all the messages sent. Following is a detailed description of a handshake for $N = 3$ entites, Client (C), Intermediary (E) and Server (S), also shown in Figure 3:

The client sends a ClientHello to the Intermediary (containing a random number rn_C and suggested cipher suites), the Intermediary selects one or more of the cipher suites and sends an own ClientHello to the Server containing the same random number rn_C. The Server generates its own random number rn_S and sends a ServerHello to the Intermediary. Then these two do a normal TLS Handshake Hello as described in section 2 (including sending and validating the server certificate). Once this is done the Intermediary passes on the ServerHello and the certificate from the Server but also includes his own certificate. The client verifies all certificates and sends his own client certificate to the Intermediary which relays it to the Server. After this the client computes a normal pre-master-secret but instead of just sending it to the server (encrypted with the server's public key) he also sends it encrypted to the Intermediary (whose public key he has from the transmitted intermediary certificate). As all involved parties now have the pre-master-secret they can each calculate the master secret and do a ChangeCipherSpec. Following that step client and server have a TLS-secured encrypted connection which all intermediaries can legally eavesdrop on.

The extension to the original TLS Handshake protocol are

doi: 10.2313/NET-2012-08-1_06

several additional messages used to exchange information with the intermediaries, client and server still do a normal handshake.

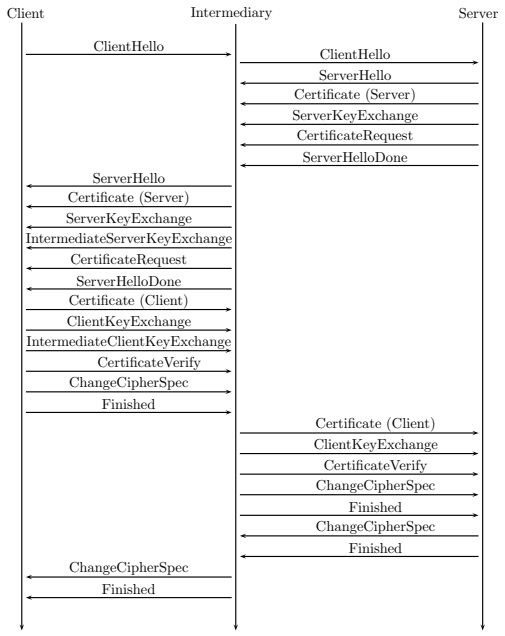

Figure 3: Time-schematic TLS Handshake for more than two entites [1]

3.3 Discussion

The described approach is superior to the mentioned naive approach in all respects. Doing the intermediary-handshake is cheaper than doing N-1 seperate handshakes but still more expensive than just doing one client/server-handshake. Also during the session each message only needs to be encrypted with one key (the common master secret) regardless of the number of N. This makes it interesting for Wireless Sensor Networks which often need to communicate with several entities in the loop. But doing all these crypto operations for the establishment of the connection is not ideal for sensor nodes. In terms of energy and computing needs RSA (which is normally used in TLS with certificates) is quite expensive and therefore can be too costly for the embedded processors used in sensor nodes. This leads to the next section which describes solutions for cheapening TLS.

4. TLS WITH IDENTITY-BASED CRYPTOGRAPHY FOR IP-BASED WSNS

In [2] the authors describe two ways to do a TLS Handshake without using RSA and X.509 Certificates: One is done using Identity-based Cryptography and Eliptic Curve Diffie-Helmann, the other is done using Eliptic Curves and Bilinear Pairing.

4.1 Some Explanations

First some introductions and explanations of the techniques and algorithms used in these approaches.

4.1.1 Identity-Based Cryptography

Identity-based Cryptography (IBC) avoids using certificates which reduces the number of messages to be exchanged during a key exchange as certificates are usually quite large. Instead in IBC there is a Private Key Generator (PKG) which is the replacement for the Certification Authority (CA) used in Certificate-based cryptography. It generates a secret key for each node based on its unique identity (which for ip-based sensor nodes is usually the IPv6 address). The nodes must be preloaded with this secret key prior to their deployment. Due to this role the PKG is a trusted party and must not be compromised.

This even provides security against IP address spoofing. An attacker could try to use the address of a legit node. But since he does not have the private key corresponding to the IP address and - without compromising the PKG and getting its secure parameters - has no way of generating it.

4.1.2 Bilinear Pairing

In short bilinear Pairing allows two parties to agree on a shared key (e.g. as a session key) without exchanging any messages. In more mathematical terms:

"Let G_1 denote a cyclic additive group of some large prime order q and G_2 a cyclic multiplacative group of the same order. A pairing is a map $e : G_1 \times G_1 \rightarrow G_2$ and has an important property that is bilinearity; if $P, Q, R \in G_1$ and $a \in Z_q^*$ [...] $e(aP, Q) = e(P, aQ) = e(P, Q)^a$."[2]

4.1.3 Eliptic Curves

Eliptic Curves (EC) [10] are an algebraic concept which are plane curves described by the equation

$$y^2 = x^3 + ax + b \tag{1}$$

and the points on that curve.

Suffice it to say that as long as the discrete logarithm problem is still expensive and difficult to solve, eliptic curve cryptography can be considered secure. Their main advantage over RSA is that the same level of security can be achieved with much shorter keys (usually around 128 bit EC is considered the same as 1024 bit RSA). This point makes them interesting to use for limited devices such as sensor nodes or smartcards as shorter keys imply cheaper operations.

4.1.4 Eliptic Curve Diffie-Hellman

The Diffie-Hellman key exchange is a protocol that allows two parties to agree on a shared key over an unsecure communications channel - this works without sending the key itsself over the wire. It is defined in RFC 2631 [7]. Eliptic Curve Diffie-Hellman (ECDH) is a variation of the protocol that uses eliptic curve public/private-key-pairs. The algorithm was introduced in [5], a longer description can be found there. Basically the exchange between Alice (A) and Bob (B) goes as following:

Both must have the same eliptic curve (or more concretly a generator P which is a point on the curve). Also each one needs a public/private-key pair (denoted as d_A and d_B for the private part and Q_A and Q_B for the public part). d is a randomly selected value and Q is calculated as $Q = d * P$. After Alice and Bob transmit their respective public keys

doi: 10.2313/NET-2012-08-1_06

(Q) to each other they can both calculate the shared key x as $x_A = d_A * Q_B$ respectivly $x_B = d_B * Q_A$. It holds $x = x_A = x_B$ because $d_A * Q_B = d_A * d_B * P = d_B * d_A * P = d_B * Q_A$. Normally some form of hash function is used on x to get the shared key.

4.2 TLS Handshake with IBC and ECDH

With this approach of using TLS with IBC and ECDH [2] the authors have tried to optimize the TLS handshake protocol for use with low-power devices such as sensor nodes. Prior to their deployment inside a network all the nodes need to be equipped with a private key and an identity (IPv6 address). As mentioned above a PKG is needed as a trusted party to generate private keys based on the IPv6 address, it also initializes some parameters for the eliptic curves (namely the generator point P of the eliptic curve).

The start of the TLS Handshake is the same as the original handshake with transmission of the ClientHello and the ServerHello (c.p. Figure 4). Also in accordance to the original specification both nodes generate random keys (let them be rn_C for the client and rn_S for the server). But instead of sending over his certificate the server calculates $rn_S * P$, signs it using his private key and sends it to the client with a ServerKeyExchange message. The client does likewise, computes $rn_C * P$, signs it and sends it to the server using a ClientKeyExchange message. The parties then exchange the normal ChangeCipherSpec and Finished messages to end the handshake. The client now knows $rns * P$ and rnc and therefore can calculate $rn_C * P * rn_S$ which is used as pre-master-secret. Likewise for the server who knows $rn_C * P$ and rn_S and can als calculate the pre-master-secret. Then they both can derive the master secret using the pre-master-secret in the normal way.

The step of exchanging certificates (which is part of the original TLS specification) can be omitted because in this incarnation the IPv6 addresses act as certificates and are already known to the communication partner due to the communication being ip-based. This saves two rather costly messages. The other point to note is that in the original TLS specification the pre-master-secret (a random number calculated by the client) is sent to the server encrypted using the servers public key. With this implementation (using IBC and ECDH) this is not necessary, the parts of the pre-master-secret ($rn_C * P$ and $rn_S * P$) are sent in plaintext because as mentioned above it is for an attacker not feasable to calculate the parts P and rn_C or rn_S of the value sent over the wire.

4.3 TLS Handshake with ECC and bilinear pairing

The authors also propose a second adaption of the TLS handshake using ECC and bilinear pairing which further reduces the number of messages sent.

Prior to deployment of the nodes there is again a PKG needed to choose/compute some parameters for the bilinear pairing. These are a random number $S \in Z_q^*$, the groups G_1 and G_2 of the same prime order q, a point $P \in G_1$, the bilinear map e and a hash function H returning points on an eliptic curve for identities (IPv6 addresses, named ID).

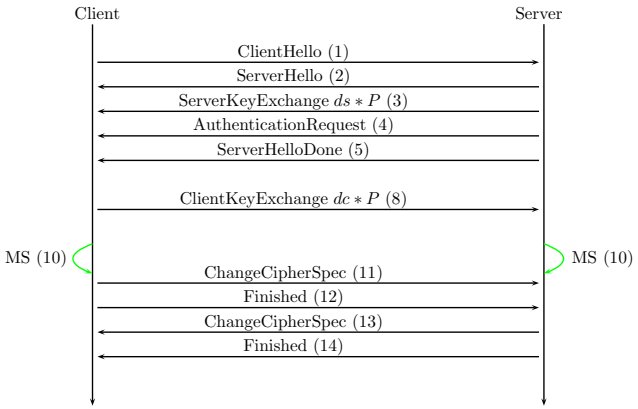

Figure 4: Time-schematic TLS Handshake with IBC and ECDH [2]

For each node j the PKG computes $Q_j = S \times H(ID_j)$ which is the private key.

The TLS handshake again starts with the exchange of ClientHello and ServerHello (c.p. Figure 5). After ending the hello phase with a ServerHelloDone both client and server can calculate the pre-master-secret as follows: Let IDc be the identity of the client (=IPv6 address) and IDs the identity of the server. S is the random number chosen by the PKG but not directly known by neither the server nor the client. The client computes the pre-master-secret as $e(Qc, H(IDs))$ where Qc is the private key of the client. The server computes it as $e(H(IDc), Qs)$. These two are identical because $Qc = S \times H(IDc)$ and $Qs = S \times H(IDs)$, so the client computes $e(S \times H(IDc), H(IDs))$ and the server computes $e(H(IDc), S \times H(IDs))$ which are equal according to the definition of bilinear pairing above with S as a, $H(IDc)$ as P and $H(IDs)$ as Q.

Using this common pre-master-secret both the client and the server can calculate the master-secret, and end the handshake after doing a ChangeCipherSpec.

In comparison to the above described solution with IBC and ECDH another two messages can be saved (namely the ServerKeyExchange and the ClientKeyExchange).

But the security of the whole process relies on the S being kept secret so that only the PKG nows its value. If the value would become public a eavesdropper could very easily compute the pre-master-secret and therefore also the master-secret because the identites (IDc and IDs) are public.

4.4 Usefullness to WSNs

As already mentioned the sensor nodes of Wireless Sensor Networks usually use embedded processors with a low computing and power capacity. This makes RSA an undesireable element of the TLS Handshake as it is very expensive to compute and the X.509 certificates accompanying the handshake are relativly big and therefore expensive to transmit.

doi: 10.2313/NET-2012-08-1_06

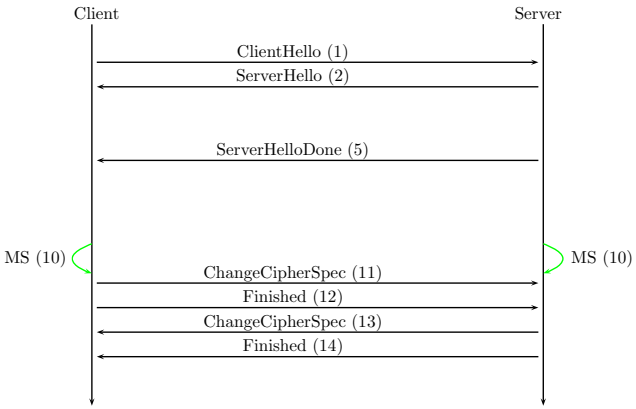

Figure 5: Time-schematic TLS Handshake with ECC and Bilinear pairing [2]

Both approaches deal with this by throwing out RSA and certificates and introducing Eliptic Curves and Identity-based Cryptography as their replacements. Thus reducing the needed computing power and the number of messages which have to be sent (which for wireless connections is extremely power-consuming) for a successfull TLS Handshake.

TLS with IBC and ECDH still stays close to the original standard by still doing a key exchange (computing and transmitting random encrypted numbers) while TLS with ECC and bilinear pairing fully dispenses with exchanging keys. So both approaches make beneficial changes to the original TLS standard for use with Wireless Sensor Networks.

But for all the advantages there is also a disadvantage which comes with the design of the Private Key Generator. It is a critical part of the infrastructure as it is responsible for generating the private keys for all nodes. If the PKG were to be compromised an attacker had all the information he had for eavesdroping on the secured connections between the nodes. This is not unlike the Certification Authority used in certificate-based cryptography which also needs to be maintained and kept secure. Which is not so easy as hacks in recent times have shown [8].

5. TINY-3-TLS

As Wireless Sensor Networks usually use some form of gateway node anyway to connect to the outside world (i.e. the Internet) it would make sense to use this gateway node as a helper for establishing secure communication between a sensor node and an outside client seeing as such a gateway node normally has stronger hardware and can therefore shoulder the complex computations for cryptography more easily. One such approach is Tiny-3-TLS whose "goal [...] is to provide an end-to-end secure communication between a remote device and a wireless sensor network"[3].

5.1 Basics

The paper differentiates between a partially trusted gateway, which means the gateway helps in establishing the connection but should not be able to eavesdrop on the end-to-end secure channel, or a fully-trusted gateway which will possess the shared secret key and therefore be able to listen in on the secure channel.

As a possible use case the authors mention the MAGNET.Care-Project[12]. The scenario is that a patient of a hospital carries medical sensors organized as a wireless sensor network and a physican at the hospital wants to connect to the sensors to get current readings using a security gateway. From a patients viewpoint the security gateway at the hospital ist not fully trusted and should therefore not be able to read the transferred medical data. Whereas if the patient is at home and his home router acts as security gateway it is fully trusted and allowed to read the sensitive data.

As already previously mentioned traditional asymmetric cryptography like RSA is relativly expensive in terms of computational needs so Tiny-3-TLS again substituts RSA with Eliptic Curve Cryptography (ECC). As a means of agreeing on a shared secret key the protocol uses Eliptic Curve Diffie-Hellman (ECDH) which was already explained in an earlier section of this paper. The ECDH public values mentioned below refer to the public key part of the ECC public/private-key pair, above denoted as Q_x.

One basic assumption is made for both approaches: Between the sensor node and the security gateway there is a shared secret key, denoted as K.

5.2 Partially Trusted Gateway

In this scenario the gateway (GW) assists in establishing the secure connection between a remote terminal acting as a client and a sensor node acting as a server but does not possess the TLS session key at the end.

The TLS handshake (as shown in Figure 6) starts with a ClientHello containing the usual CipherSuite offers, the client identity (ID_c) and a nonce (N_c) from the client to the GW which encrypts the entire packet using the shared symmetric key K and sends it on to the server. In response the server sends a ServerHello message (encrypted wth K) to the GW additionally containing the server identity (ID_s), a nonce (N_s) and its ECDH public values. The GW does not pass the message on to the client but instead composes his own ServerHello which does not contain the servers ECDH public values but instead contains the GW certificate and a request for a client certificate. To this the client responds with his own certificate, ECDH public values and a second nounce (N_g), called gateway authentication nonce. The entire message is encrypted asymmetricly using the GWs public key found in the certificate. Upon receipt the GW proves its ownership of the private key mentioned in the GW certificate by sending the gateway authentication nonce back to the client, it also includes the ECDH public values of the server which the GW removed from the first ServerHello (all encrypted with the clients public key). Also the GW transmits the ECDH public values received from the client to the server (again encrypted with K).

As both the server and the client now have the ECDH public values of the other partner they can now calculate the pre-

doi: 10.2313/NET-2012-08-1_06

master-secret according to the ECDH algorithm and using the exchanged nonces (N_c and N_s) and identities (ID_c and ID_s) calculate the master key. With this both can encrypt the Finished messages and start using their secure end-to-end channel.

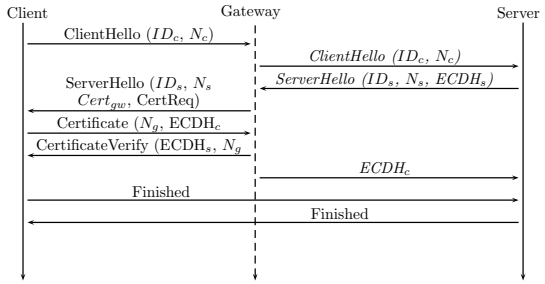

Figure 6: Time-schematic Tiny-3-TLS with partially trusted GW [3]

5.3 Fully Trusted Gateway

5.3.1 TLS Handshake

The procedure for doing the TLS handshake using the fully-trusted gateway (as shown in Figure 7) is at the beginning very similar to the procedure for the partially-trusted gateway. The client sends its ClientHello, the GW passes it on encrypted with K to the server which responds with a ServerHello but this time without ECDH public values. The GW again passes the message along but includes his certificate and a certificate request. The client responds to that request by transmitting his own certificate to the GW. Then - as in standard TLS - it generates a pre-master-secret (usually just a random number), encrypts it with the public key of the GW and sends it on to the GW.

The GW generates a client-read-key and a client-write-key and sends them along with a random number encrypted (using K) to the server. Once the server confirmed it has received and decrypted the keys (by sending back the random number) the GW ends the TLS handshake with the client by sending a Finished message. With that the secure channel is established.

Communication between the client and the GW is done using the master secret derived from the pre-master-secret sent by the client. Between the GW and the server messages are encrypted using the client-read-key and the client-write-key generated by the GW.

5.3.2 Comparison to TLS for more than two entities

Tiny-3-TLS with a fully trusted gateway is very similar to the TLS enhancement for more than two entites from [1] explained in section 3. The main difference to the outcome is that in [2] at the end all entities have and use the same master secret whereas in Tiny-3-TLS a real TLS session is only established between the client and the GW which relays the messages to the server using a special key not known to the client. This means the GW has to do additional cryptography computations for reencrypting all messages passed around. But this can be an advantage. Client and GW can

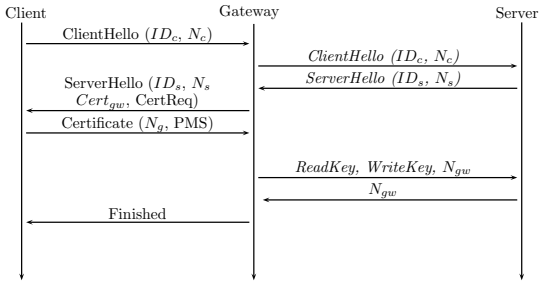

Figure 7: Time-schematic Tiny-3-TLS with fully trusted GW [3]

use strong and complicated encryption algorithms to secure the messages while travelling through the internet or another unsecure network. Whereas between server and GW a cheaper algorithm can be used which is more suited for the low-power processors used in sensor nodes.

6. SUMMARY

In this paper three different extensions/enhancements to the Transport Layer Security Protocol were described which all have their merits and faults (a tabular comparison is shown in Table 1). TLS for more than two entities is the ideal solution for interconnecting multiple entities with one shared secured channel. But it is not very usefull for applications in Wireless Sensor Networks as it still uses the computationally heavy standard TLS asymmetric cryptopgrahy protocols such as RSA with X.509 certificates. But it provides an interesting basis for connections between multiple nodes which in sensor networks is more common than in the classic internet.

Tiny-3-TLS remedied that weakness by introducing eliptic curves and a gateway node for handling the intensive cryptography computations. It can be seen as an enhanced way for the approach in section 3 - specifically tailored for $N = 3$.

The introduction of Identity-based Crytography is a different approach also making use of eliptic curve cryptography and other concepts to reduce the number of messages and computations that have to be done by the nodes but keeps the involved number of entities fixed at $N = 2$.

All three approaches enhance TLS for specific applications and should not be considered as a general improvement of Transport Layer Security.

7. REFERENCES

[1] M. Badra: *Securing Communications between Multiple Entities Using a single TLS Session*, IEEE 2011

[2] R. Mzid, M. Boujelben, H. Youssef, M. Abid: *Adapting TLS Handshake Protocol for Heterogenous IP-Based WSN using Identity Based Cryptography*, In Proceedings of the International Conference on Wireless and Ubiquitous Systems, 8-10 October 2010, Sousse, TUNISIA

[3] S. Fouladgar, B. Mainaud, K. Masmoudi, H. Afifi: *Tiny 3-TLS: A Trust Delegation Protocol for Wireless*

doi: 10.2313/NET-2012-08-1_06

Table 1: Advantages & Disadvantages of the different approaches

Approach	Advantages	Disadvantages
TLS $N > 2$	• little additional effort required • compatible to original TLS	• uses X.509 Certificates • uses RSA (computationally expensive)
IBC, EC, BP	• Eliptic Curves are better than RSA • Bilinear Pairing optimal for number of messages sent	• Private Key Generator is needed • Not conformant to TLS standard
Tiny-3-TLS	• Uses gateway node which is needed anyway • Less number of messages	• Gateway needs to be trusted • K must be securely shared

Sensor Networks, ESAS 2006, LNCS 4357, pp. 32–42, 2006

[4] T. Dierks, E. Rescorla: RFC 5246 - The Transport Layer Security (TLS) Protocol, Version 1.2, `http://tools.ietf.org/html/rfc5246`

[5] L. Law, A. Menezes, M. Qu, J. Solinas. S. Vanstone: *An Efficient Protocol for Authenticated Key Agreement*, Technical Report CORR 98-05, Dept. of C&O, University of Waterloo, Canada, March 1998

[6] S. A. Thomas: *SSL and TLS Essentials - Securing the Web*, Wiley Computer Publishing, USA 2000

[7] E. Rescorla: RFC 2631 - Diffie-Hellman Key Agreement Method, June 1999, `http://tools.ietf.org/html/rfc2631`

[8] *What You Need to Know About the DigiNotar Hack*, `http://threatpost.com/en_us/blogs/what-you-need-know-about-diginotar-hack-090211`

[9] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci: *Wireless sensor networks: a survey*, Computer Networks 38 (2002) p. 393-422

[10] C, Eckert: *IT-Sicherheit: Konzepte, Verfahren, Protokolle*, Oldenbourg Wissenschaftsverlag, München 2008

[11] H. Karl, A. Willig: *Protocols and Architectures for Wireless Sensor Systems*, Wiley 2005

[12] *IST-MAGNET*, `http://www.ist-magnet.org`

doi: 10.2313/NET-2012-08-1_06