EPOS' Secure Communication Support

Davi Resner
Software/Hardware Integration Lab
Federal University of Santa Catarina
davir@lisha.ufsc.br

May 30, 2014

Abstract

Communication security is an important concern in Wireless Sensor Networks. In many scenarios, there is the need to make sure that exchanged messages have the properties of authentication, confidentiality and integrity. This report shows how to use EPOS' secure communication classes to enable sensors to establish a symmetric key with a central server and then use this key to communicate with the aforementioned principles granted transparently. Each parameter comes followed by the filename and line number in which it is set. This document is meant to go along with the secure_nic_master.cc and secure_nic_slave.cc applications.

Keywords: Internet of Things; Wireless Sensor Networks; Trustfulness; Security; Key Establishment; Confidentiality; Authentication; Integrity

1 Overview

EPOS' Secure Key Bootstrapping and Authentication protocol is based on the Elliptic Curve version of Diffie-Hellman. It enables sensors to establish a symmetric key with a trusted server and used it to encrypt messages and communicate securely. The protocol relies on unique sensor IDs known beforehand only by the server and the corresponding sensor.

AES is used as an encryption engine; PTP [1] is used to synchronize clocks and thus be able to use timestamps to prevent replay attacks; Poly1305-AES [2] is the One-Time-Password generation engine.

To use the protocol, a number of parameters need to be set the same on each node, which are explained throughout this document.

For a good yet not dense explanation of the Diffie-Hellman protocol and Elliptic Curves, refer to [3]. The implementation of the required arithmetic was greatly based on [4] [5]. EPOS' PTP implementation is explained in [1]. A little information about the protocol explained here can be found at [6].

2 Machine-specific parameters

You can tweak these two parameters to optimize the implementation to a particular machine. If your machine has a 32-bit architecture, you shouldn't need to change these.

Bignum base (or digit size): include/traits.h:230
The base in which the bignums are represented. For best performance, this should be set to the native processor's word size (usually unsigned int). The digit size is defined as the size of the type of the base (e.g. sizeof unsigned int). Always use an unsigned type.

Double-digit: include/traits.h:231
A type of size two times larger than the one defined as digit. Should also be unsigned.

3 Network Parameters

These parameters are public and define the elliptic curve to be used by the Diffie-Hellman algorithm for this specific network. These values should be set equally in every node to enter the network. You may use the values provided as comments in the model applications without a problem.

Modulo (or Prime):
app/secure_nic_master.cc:30
app/secure_nic_slave.cc:31
This should be a big prime number. It is written in little-endian, digit-by-digit format. In general, the bigger this number is, the more secure the network becomes, but the complexity of the Bignum operations grow accordingly. You can use SECV [7] recommendations of elliptic curve parameters.

Bignum size: include/traits.h:232
The size in digits of each bignum. This should be the same size as the defined modulo. The bigger this value is, the more complex each bignum operation gets, but the network security increases accordingly. NIST suggests [8] that, to provide the same security as a 128-bit symmetric key, a 256-bit prime should be used. You can adjust this value to your processing constraints.

Barrett μ:
app/secure_nic_master.cc:31
app/secure_nic_slave.cc:32
The modular reduction after multiplication is implemented with the Barrett Reduction [4] algorithm. This is a constant used by the method to accelerate reduction, and is defined as:

$$\mu = \lfloor b^{2*k}/p \rfloor \tag{1}$$

Where b is the base (usually 2^{32}), k is the size in digits of the prime, and p is the prime (defined in **Modulo**). μ will always be one digit larger than the prime. It is written in little-endian, digit-by-digit format.

Base point:
app/secure_nic_master.cc:28,29
app/secure_nic_slave.cc:29,30

A little-endian, byte-by-byte representation of the elliptic curve point used as Base Point by Diffie-Hellman. You can use SECV [7] recommendations of elliptic curve parameters.

Time window: include/traits.h:256
Every time a time stamp is used by the security module, it is rounded up to a multiple of this value. This is effectively the time window in which any given secure message will be considered valid. Replay attacks are possible within the same time window, so you should not set this value too high. A value too low might make it impossible to send valid messages due to duty cycling in the network, so you should take that into account.

4 User's Guide

This guide is divided in two: one for the server and one for the sensor nodes. Each of these is subdivided in three phases, that must be followed in order: 1) Initialization; 2) Key Establishment and Authentication; 3) Secure Communication.

4.1 Deploying a Server

4.1.1 Initialization

These values characterize this particular node and prepare for execution. For the server, the following parameters need to be set:

NIC Address: app/secure_nic_master.cc:158
The MAC address of the server. This value should be known by every node.

PTP role: app/secure_nic_master.cc:117-119
Tells the PTP object that this is a master node, which will be used as a reference clock by the other nodes, and will issue synchronization messages.

Random seed: app/secure_nic_master.cc:123
It is important to make sure that each node has its own seed, because Diffie-Hellman private parameters are chosen at random, and two nodes ideally shouldn't have the same private keys.

PTP: app/secure_nic_master.cc:120,145
Line 120 tells the PTP object to initialize its internal parameters. Line 145 sets up a periodic thread which will issue a PTP synchronization message every configured time period.

Secure_NIC: app/secure_nic_master.cc:125
The Secure_NIC constructor will calculate this node's Diffie-Hellman keypair. This is a costly process and might take a few seconds, depending on the prime size in use. It takes as parameters a boolean which is true *iff* this is a server, the cipher to be used, the Poly1305 implementation and the NIC used

as the lower layer.

Trusted IDs: app/secure_nic_master.cc:128-136
The ID of every node to be trusted by this server should be informed via the insert_trusted_id method. Any node with an ID not registered will *not* be able to authenticate.

4.1.2 Key Establishment and Authentication

The server just needs to start accepting authentication requests. The authentication process is then carried out automatically when requests arrive.

Start accepting requests: app/secure_nic_master.cc:139
The server will *not* try to authenticate any node unless this value is set. You can write to this variable anytime to start/stop accepting connections. It starts as false by default.

4.1.3 Secure Communication

Nodes that are authenticated with the server can send and receive secure messages to/from it. If a destination node is *not* authenticated with this server, calling the send method will result in no message being sent.

Receiving secure messages: app/secure_nic_master.cc:69,74,76,142
You can receive secure messages by registering an observer to the secure nic. Every time a secure message arrives, the update method will be called. Decryption is handled transparently, and the message is delivered already decrypted.

Sending secure messages: app/secure_nic_master.cc:81
You can send secure messages by just calling the send method. If the destination node is authenticated, the message will be encrypted and sent.

4.2 Deploying a Sensor

4.2.1 Initialization

These values characterize this particular node and prepare for execution. For the sensors, the following parameters need to be set:

Server's NIC Address: app/secure_nic_slave.cc:19
The MAC address of the server. This value should be known by every node.

NIC Address: app/secure_nic_slave.cc:20,161
The MAC address of this sensor. Every node in the network should have a unique MAC address. If this is not the case, the authentication process may go wrong.

PTP role: app/secure_nic_slave.cc:97,98
Tells the PTP object that this is a slave node, which will respond to synchronization messages and adjust its clock according to the server's.

Random seed: app/secure_nic_slave.cc:106
It is important to make sure that each node has its own seed, because Diffie-Hellman private parameters are chosen at random, and two nodes ideally shouldn't have the same private keys.

PTP: app/secure_nic_slave.cc:100
Tell the PTP object to initialize its internal parameters.

Secure_NIC: app/secure_nic_slave.cc:111
The Secure_NIC constructor will calculate this node's Diffie-Hellman keypair. This is a costly process and might take a few seconds, depending on the prime size in use. It takes as parameters a boolean which is true *iff* this is a server, the cipher to be used, the Poly1305 implementation and the NIC used as the lower layer.

IDs app/secure_nic_slave.cc:112
Every trusted node must have an unique ID, known to the gateway. Use the set_id method to set the ID of this node in the Secure_NIC object.

4.2.2 Key Establishment and Authentication

The sensor nodes need to ask the server for authentication.

Request authentication from server: app/secure_nic_slave.cc:121
Use this method to send the first message in the authentication protocol. The rest of the messages will be handled automatically.

Poll for authentication: app/secure_nic_slave.cc:124
The authenticated() method will return true only when this node finishes the protocol correctly, thus being authenticated to the server and sharing a symmetric key with it.

4.2.3 Secure Communication

Nodes that are authenticated with the server can send and receive secure messages to/from it. If the server is *not* authenticated with this node, calling the send method will result in no message being sent.

Receiving secure messages: app/secure_nic_slave.cc:75-80,134
You can receive secure messages by registering an observer to the secure nic. Every time a secure message arrives, the update method will be called. Decryption is handled transparently, and the message is delivered already decrypted.

Sending secure messages: app/secure_nic_slave.cc:141
You can send secure messages by just calling the send method. If the destination node is authenticated, the message will be encrypted and sent.

5 Using Bignums with different moduli

If you need to operate on several finite fields in the same application, this is possible. If the fields use moduli of the same size, you can just use the set_mod method in Bignum to set a different modulo per-object. Keep in mind that *you* must handle allocation of the modulo data (the data is <u>not</u> copied).

If you need to use moduli of different sizes, you can create a class that extends Bignum and overwrite the word and sz_word parameters, as well as the modulo and Barrett μ. This is exactly what is done by Poly1305-AES, so check out the files include/poly1305.h and src/abstraction/poly1305.cc.

6 Questions?

Feel free to contact me via email: davir@lisha.ufsc.br.

References

[1] P. Oliveira, A. M. Okazaki, and A. A. Fröhlich, "Sincroniza cão de tempo a nível de so utilizando o protocolo ieee1588," in *Simpósio Brasileiro de Engenharia de Sistemas Computacionais*, Natal, Brazil, November 2012.

[2] D. J. Bernstein, "The poly1305-aes message-authentication code," in *Proceedings of Fast Software Encryption*, Paris, France, February 2005, pp. 32–49.

[3] W. Stallings, *Cryptography And Network Security*, 5th ed. Pearson, 2011.

[4] M. Brown, D. Hankerson, J. López, and A. Menezes, "Software implementation of the nist elliptic curves over prime fields," in *Topics in Cryptology - CT-RSA 2001*, ser. Lecture Notes in Computer Science, D. Naccache, Ed. Springer Berlin Heidelberg, 2001, vol. 2020, pp. 250–265.

[5] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography.* CRC Press, 1996.

[6] A. A. Fröhlich, A. M. Okazaki, R. V. Steiner, P. Oliveira, and J. E. Martina, "A cross-layer approach to trustfulness in the internet of things," in *9th Workshop on Software Technologies for Embedded and Ubiquitous Systems (SEUS)*, Paderborn, Germany, June 2013.

[7] *Standards for Efficient Cryptography (SEC) SEC 2: Recommended Elliptic Curve Domain Parameters*, Certicom Research, September 2000.

[8] N. S. Agency. (2014, May) The case for elliptic curve cryptography. [Online]. Available: http://www.nsa.gov/business/programs/elliptic_curve.shtml