Performance Evaluation of a Secure Low Level Reader Protocol (LLRP) Connection

Sana Qadir[†] and Mohammad Umar Siddiqi[††],

International Islamic University Malaysia, Kuala Lumpur, Malaysia

Summary
The recently ratified Low Level Reader Protocol (LLRP) specifies the interaction between a RFID Reader and Client. It has been of much interest in the RFID community but adoption is being stalled by its lack of formal scrutiny especially with regard to its security. This paper surveys the work that has been undertaken on this protocol, assesses its security vulnerabilities and examines possible security solutions. It then presents the design and implementation of LLRP endpoints that use Transport Layer Security (TLS) to setup a secure LLRP connection. Based on previous performance studies, appropriate metrics are selected to indicate the performance of the resulting TLS-LLRP connection. Specifically, the overhead of securing a LLRP connection using TLS is quantified and a detailed analysis of the results is undertaken to determine the TLS cipher suites and parameters that provide the best compromise between the level of security and performance.

Key words:
LLRP, TLS, Performance, cipher suite.

1. Introduction

EPCglobal Inc. is the company in charge of the development of standards to govern the deployment of RFID technology for supply chain management. These standards are intended to lead to the development of the *EPCglobal Network*. Basically, this is the effect achieved when companies employ EPCglobal standards and core services to manage their supply chains and for interacting with business partners and with EPCglobal [1]. The advantage of the creation of EPCglobal Network is to automate supply chain management and eventually to enable enterprises to seamlessly and securely exchange relevant data on a global scale.

Several EPCglobal Standards are still under development but those that can be deployed within an enterprise have been specified. An example of an intra-enterprise RFID setup that shows two very important EPCglobal Standards is given in Fig. 1.

The Electronic Product Code (EPC) is a 96-bit code stored on a RFID tag. The product, to which this tag is attached, is identified by reading the EPC using the air interface protocol called UHF Class 1 Gen 2. The RFID Reader passes on this data along with other information to a Client using the Low Level Reader Protocol (LLRP). It is this latter standard that is the focus of this paper.

An important condition for fully benefiting from any RFID setup is the security of the communication channels. Security is especially important for the communication channel that connects a RFID Reader to a company's intranet. Progress on this front has been impeded in the past due to the resource constraints of RFID devices and to the unacceptable delay in performance resulting from the inclusion of security mechanisms. Improvements in technology are however diminishing the impact of these limitations. This paper will first assess the security requirements of a LLRP connection and then select a security protocol to secure the connection. Finally, a performance evaluation of the secure LLRP connection is carried out. This is intended to provide relevant input to security experts, protocol designers and system implementers of how a security protocol affects the performance of a system.

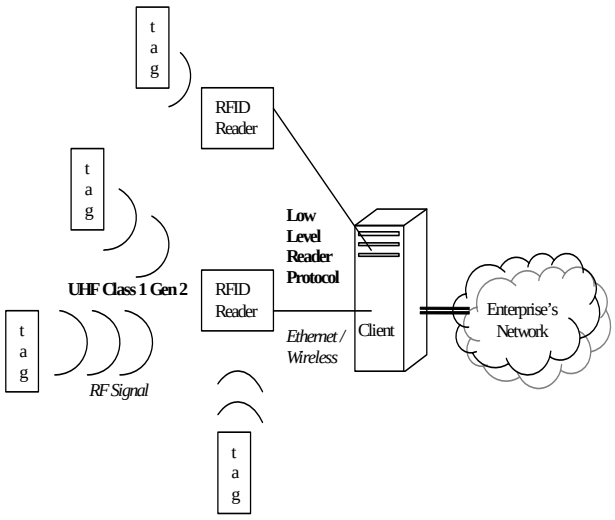

Fig. 1: Example of an intra-enterprise setup of EPCglobal Network

Manuscript received March 5, 2009
Manuscript revised March 20, 2009

2. Introduction to LLRP and Related Work

LLRP is one of the latest EPCglobal standard and 'is specifically concerned with providing the formats and procedures of communication between a Client and a Reader' [2]. Clients use the protocol to get and set Reader configuration, discover capabilities of a Reader and to manage a Reader's access and inventory operations [3].
Readers use the protocol to report their status, the result of RF surveys, access and inventory operations [3]. Communication via LLRP is in the form of protocol data units called *messages* that are sent to and from the Reader and Client. Each message is encoded in binary before being sent over TCP [4].
The first major work on LLRP was the development of a class library for the specification by Joe Hoag of the University of Arkansas. The library, written in Java, helps to encode/decode LLRP messages to and from their binary representation. This library was used by Pramari, a company specializing in the development of virtual RFID Readers. They utilized it to enable a LLRP virtual Reader to be simulated by their Pramari Virtual Reader software [3]. Their objective is to enable researchers to simulate a particular Reader before it is available in the market [3]. The library was also later used to develop a LLRP Reader agent called `LLRPReaderAgent` [3]. This agent is capable of talking to any LLRP Reader and was integrated into `TagCentric` (an open source agent-based RFID middleware application).
The LLRP has been more warmly received by the vendor community than its predecessor, the Reader Protocol. This is because the Reader Protocol provided a more 'abstract, event-driven approach' to controlling the operation of a Reader while the LLRP is more directly involved with a Reader's operation [5]. In particular, it is aware of the specific air protocol used to communicate with tags and therefore provides air 'protocol-specific parameters and control' [5]. It also allows control of other hardware aspects of a Reader. Another advantage of LLRP is that it provides more support for vendor extensions e.g. in the form of custom messages. It is for these reasons that the RFID company Impinj Inc. has recently released a LLRPv1.0.1 compliant Reader (Speedway® Reader). They also claim that using LLRP improves the performance of a RFID system [6]. Vendors are likely to follow Impinj and more releases of LLRP compliant Readers are expected. Although Impinj's implementation of LLRP is propriety, the company is also involved in developing an open source programming toolkit for LLRP. The purpose of this project is to provide libraries in many languages 'for the development of LLRP-based applications and smart readers' [7]. The main development languages being used are Java, C++ and Perl [7]. These libraries are to facilitate the building and parsing of LLRP messages and will be handy to Reader and software vendors [7].

3. Security Assessment of LLRP

Security and privacy are services that EPCglobal aims to provide its users. To achieve this, their specifications either include security features or recommend the best security practice [1]. However, any supply chain management system built using current RFID components is vulnerable to several security threats. This is because recommended security features have simply not been adopted. The main reason for this is that such features are regarded as features that can be added on later or ignored because they can seriously degrade system performance. As a result, most RFID systems have no security mechanisms implemented at all and data is transmitted without being checked or encrypted. [8] proposes 'a secure web service framework applying to an industry-wide EPCglobal Network utilizing Web Services Security (WSS) technology'. They use Web services because their focus is on the security of information exchanged between trading partners, i.e. an inter-enterprise setup. This is in contrast to an intra-enterprise setup where LLRP operates.
Reference [9] undertook to assess the security of the components that comprise the EPCglobal Network. They found that the most significant threats in an intra-enterprise scenario are posed by:

- malicious or compromised Readers, and
- eavesdropping, spoofing, man-in-the-middle and replay attacks on the communication channel between Reader and Client.

The security services that should be adopted are [9] [2]:

- Reader-Client mutual authentication and authorization, and
- privacy protection and integrity protection of LLRP messages

The LLRP standard itself merely states that a LLRP Client and Reader MAY implement Transport Layer Security (TLS) [2]. In other words, including TLS is permissible within the scope of the standard but *no* recommendation is made. Even [9] suggests TLS as a solution for the security requirements of a Reader-Client Interface.
Some Readers nowadays (e.g. handheld models), include a wireless interface (802.11b/g) to communicate with a Client. For usage over a wireless connection, TLS was fine-tuned to a new 'lighter' protocol called Wireless Transport Layer Security (WTLS). Despite its existence, WTLS was not recommended by EPCglobal for use with LLRP. It was considered more important to remain interoperable with wired systems that use the ubiquitous

TCP/IP stack and public key infrastructure (including X.509 certificates). Moreover, WTLS is part of the Wireless Application Protocol (WAP) suite and that requires the application protocol to be HTTP. In our case, LLRP not HTTP, is the application protocol being considered.

4. Securing LLRP Connection and TLS

To the authors' knowledge no other work has been done on securing LLRP connections. TLS has many advantages because of which it is the primary choice for securing a LLRP connection. In addition to providing the security services listed above, it is light weight and has been proved to be reliable and secure by millions of users [2].

The TLS protocol v1.0 is the result of IETF's effort to standardize Netscape's Secure Socket Layer (SSL) protocol v3.1. It comprises two main sub-protocols called the Handshake Protocol and Record Protocol as shown in Fig. 2 [10]. Their functions are summarized in Table 1 [10]. The Record Layer is a layered protocol in which data is fragmented, compressed (optional), used for the generation of a message authentication code and finally encrypted. The security parameters needed by the Record Layer are negotiated during the TLS Handshake. The TLS Handshake protocol is mainly responsible for authenticating the identities of the communicating endpoints but it also negotiates other cryptographic parameters needed by a TLS session [10].

A full handshake is carried out only when a new session is requested and an abbreviated handshake is used when a previous session is being resumed. In the latter type of handshake, no public key operations are used so we consider only those cases where a full handshake takes place. Even in a full handshake, the *key exchange method* and the endpoints being authenticated determine the exact message flow [10].

We focus on cases where the identity of both endpoints' is being verified and when both endpoints employ identical type of certificates.

In general, full handshakes are based on either Rivest-

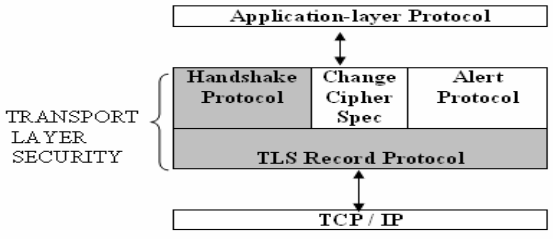

Fig.:2: Sub-protocols of TLS

Table 1: Functions of main TLS sub-protocols

Protocol	Responsibility	Mechanism:
Record Protocol	Privacy	Encryption (via symmetric cryptography)
	Message Integrity	Keyed message authentication digest
Handshake Protocol	Authentication of communicating peers	Public-key cryptography
	Setting the cryptographic parameters of the session state (i.e. negotiate encryption algorithm and cryptographic session keys)	Public-key cryptography

Shamir-Adleman (RSA) or Elliptic Curve Cryptography (ECC) as shown in Fig. 3 and Fig. 4 respectively [10][11].

In the `ClientHello` message, a list of preferred cipher suites is sent from the TLS Client to the TLS Server. Each cipher suite is a specification of the key exchange algorithm, symmetric cipher and MAC algorithm that can be used on the TLS connection. The Server is required to select one cipher suite from this list and send it to the TLS Client in the `ServerHello` message. If a cipher suite is not agreed upon, the handshake fails and the connection is terminated. A list of predefined cipher suites is included for use in the TLS standard. The strength of TLS is flexible and can easily be changed by the communicating endpoints negotiating a different cipher suite during a TLS handshake [2].

5. Review of Related Performance Studies

It is acknowledged that security protocols introduce overhead into the performance of a system. This was the case with TLS when it was being adopted for securing Web servers. To quantify the delay incurred because of TLS become urgent because the slow response time of the secure Web Servers was translating into loss of revenue for e-commerce sites.

There are also a number of trends that make evaluation of performance imperative and the continued use of RSA difficult to justify. These trends include the need to secure simpler, more constrained devices, the larger number of applications requiring security and the demands for higher level of security [12].

In line with this, a succession of performance evaluation

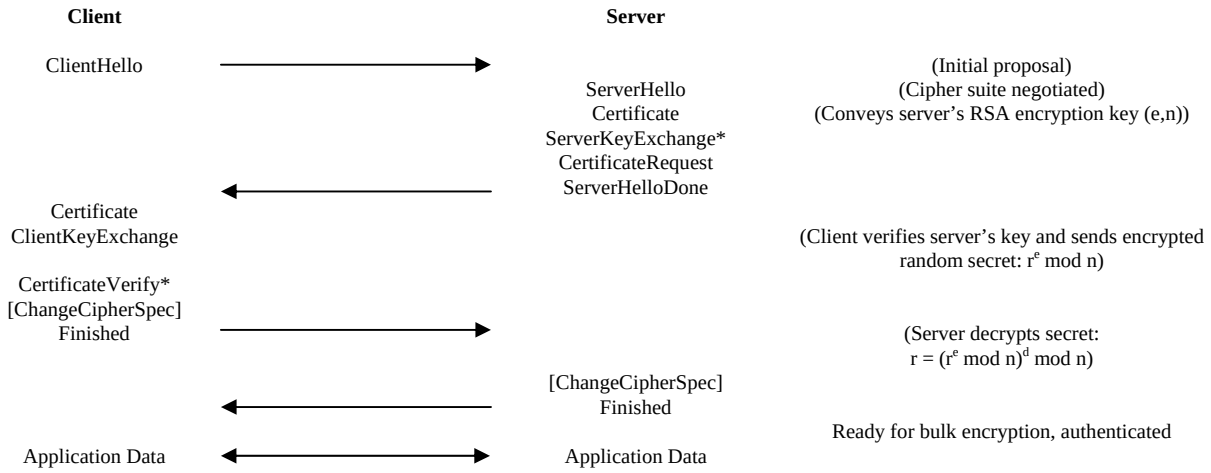

Fig. 3: RSA-based Full Handshake

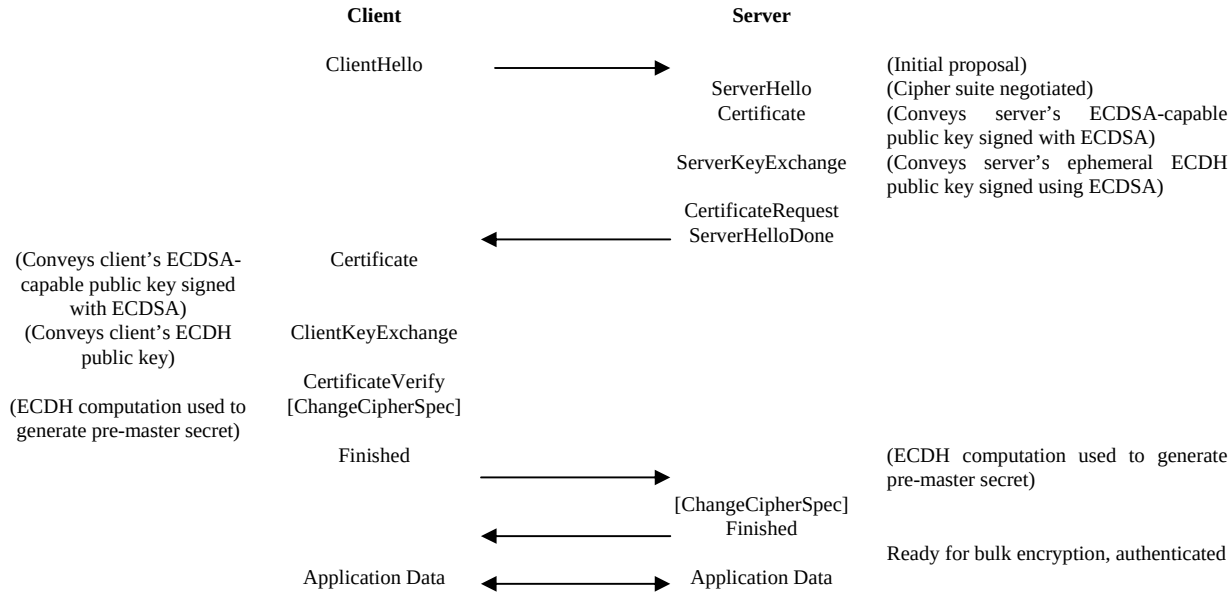

Fig. 4: ECC-based Full Handshake

studies have been carried out and in majority of them TLS is used to secure HTTP transactions. Other application protocols, like LLRP, differ from HTTP in terms of the nature of the messages, usage profile and security requirements. Nonetheless, several useful points can be noted from these studies. Firstly, the public key operations in the TLS Handshake Protocol are responsible for majority of the overhead of securing HTTP transactions [13] [14]. Secondly, the overhead due to the cryptographic operations performed by the Record Layer is relatively small and only significant for larger transactions [13]. [13] and [14] are not only dated but their focus on server authentication only and on RSA is a serious limitation. Nonetheless, like in their studies, we use the duration of handshake as a very important metric of performance. We also use the metric Propagation Time of TLS-LLRP message to indicate overhead of the Record Protocol.

With the increasing popularity of ECC, [15] and [16] compared the improvements that can be expected if RSA is replaced with ECC. Specifically, [15] compared the time spent during a handshake on RSA operations with time spent on ECDH_ECDSA operations if they are used

instead. They concluded that ECC cipher suites results in better performance of both the SSL client and server and that the performance advantage of ECC increases as higher levels of security are used. [16] reached a similar conclusion (the cipher suite TLS_ECDH_ECDSA_WITH_RC4_128_SHA performs better than TSL_RSA_WITH_RC4_128_SHA) although different performance metrics were used. To be more specific, these metrics accurately reflected performance from the user's viewpoint and recorded the rate at which the secure Web server was able to fulfill web page requests. The authors of [16] highlight the importance of their results, by stating that public key operations are the most computationally expensive part of SSL and finding techniques that reduce this computational overhead should be a high priority. Their study is however, restricted to the specific Web browser and server used as well as by their custom implementation of ECC. The latest study to be carried out [12] was in line with the trend of requiring or recommending the inclusion of ECC in security mechanisms. It finds that 'replacing RSA with ECC reduces that server's processing time for new SSL connections across the entire range of page sizes from 10KB to 70KB' [12]. Although [12] is the most recent study, it does not include RSA key sizes larger than 2048 bits, nor does it deal with the impact of using AES or with mutual authentication.

The performance evaluation of public-key cryptosystems in WTLS is carried out in [17]. Their metrics 'client processing time of public-key operations', 'server processing time of public-key operations' and 'the amount of data exchanged between the client and server' all show that in WTLS, ECC curves outperform RSA cryptosystems [17].

6. Design and Implementation

All design and implementation decisions were made by the authors since EPCglobal does not currently provide any specifications for the implementation of security features [8].

TLS always lies above TCP and consequently will lie below LLRP. This means that the initiation of a TLS-LLRP connection comprises of the following sequence: a TCP handshake, a TLS handshake and a LLRP connection initiation sequence. Likewise the termination of a TLS-LLRP connection comprises of the sequence: a LLRP connection termination, a TLS connection termination and a TCP connection termination.

The steps that have to be taken when sending and receiving

LLRP messages over a TLS-LLRP connection are shown in Fig. 5.

Development can commence only after open source implementations of LLRP Reader and LLRP Client have been selected. The Accada EPC Network Prototyping platform (developed using Java) is at the forefront of enabling a quick-setup of EPC-compliant systems [18]. However, its main Reader Core module is an implementation of the Reader Protocol and not LLRP. Furthermore, since LLRP is seen as a replacement of the Reader protocol, it does not make sense to use Accada.

The only open source LLRP Reader in the market is the `Rifidi Emulator v1.5` [19]. It is written in Java. Although still a prototype, `Rifidi`'s virtual LLRP Reader

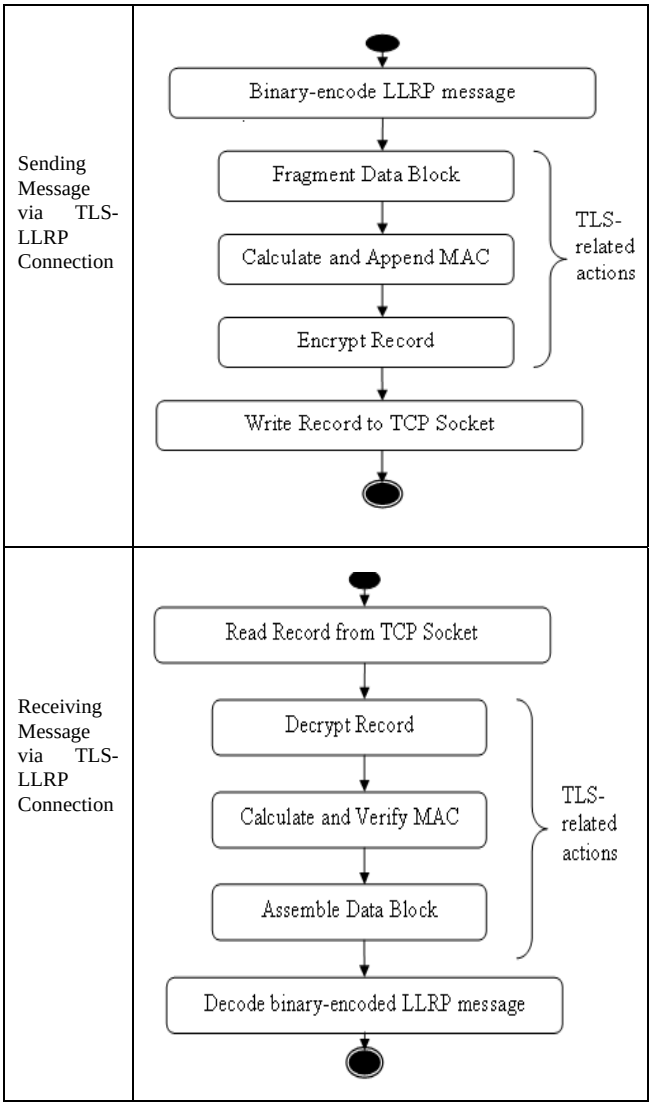

Fig. 5: Messages sent and received over TLS-LLRP

functions like a real Reader down to the packet level and can emulate TCP connections. It can therefore be extended by adding TLS so that instead of operating like a TCP server, it operates like a TLS server. The default port for accepting TLS-LLRP connections is also set to 5085 as specified in [2].

To establish a TLS-LLRP connection, a TLS-LLRP Client is also required. To get an accurate assessment of performance, it is important that the TLS-LLRP Client estimates workload properly. In the case of LLRP, not only are there no benchmarks available but also no sample usage data. Therefore, as a starting point, we select an open source Java program called `LLRPHelloWorldClient` that is based on the typical LLRP timeline given in the LLRP standard [2: p21] [20]. This program is extended to include the function of establishing a TLS-LLRP connection with the TLS-LLRP Reader described in the preceding paragraph.

The TLS implementation to be used to extend each LLRP endpoint had to be carefully selected. In the end, the Java Secure Socket Extension (JSSE) implementation of TLSv1.0 was chosen. Although written in Java and probably not as fast as C/C++ implementations of TLS like `cryptlib` and `OpenSSL`, JSSE is chosen for this work because it is the best implementation of TLS in Java. A Java implementation is compulsory. This is because it has to be used to extend the Java programs `Rifidi Emulatorv1.5` and `LLRPHelloWorldClient`. When configured to work with Network Security Services (NSS) libraries, JSSE provides support for 34 cipher suites. It should be noted that despite the advantages of Java like portability and simplicity, support for the language is only just beginning to be included in RFID Readers (e.g. Intermec's IF 5 Fixed Reader, Symbol's XR 440 Fixed Reader and INfinity 510 by Sirit Inc). Such basic support is however not enough for a Java TLS-LLRP implementation (like the one developed in this work), to be practical on current RFID Readers. A Java TLS-LLRP implementation is only practical on RFID Readers that use Java to operate and it is not unreasonable that given the current trend in the market, such Readers will be released in the near future.

7. Performance Metrics and Experimental Procedure

The experiments require two identical machines (one for each program) linked via a wireless 11 Mbps connection. Public keys, private keys and certificates for both TLS-LLRP endpoints are generated using `keytool`.

Evaluation of performance requires suitable choice of metrics, i.e. metrics that accurately estimate 'perceived' performance [15]. We chose two main metrics to indicate performance. The experimental procedure as well as the TLS-LLRP Client used differs for each of the two main metrics. Details of the two main metrics are given below:

i. *Handshake Duration*: to estimate the performance of the Handshake Protocol
   12 of the most important cipher suites and other options provided by TLS were arranged into 26 different *combinations* (see column one and two of Table 3). Each *run* is defined by launching the TLS-LLRP Reader on one machine and running the TLS-LLRP Client to completion on the other machine.10 runs are carried out for each of the 26 different TLS combinations.
   The TLS-LLRP Client used in these runs establishes a TLS-LLRP connection with the TLS-LLRP Reader, exchanges 34 LLRP messages with it and then terminates the TLS-LLRP connection (see Fig. 6). The network protocol analyzer, `Wireshark`, is used to confirm the exchange of TLS-LLRP packets between the two machines.
   Timestamps were generated using the Java function `System.nanoTime()`.

ii. *Propagation Time of TLS-LLRP Message*: to estimate the performance of the Record Protocol.
   A different TLS-LLRP Client was written to record this metric accurately. This was because measuring propagation time using the TLS-LLRP Client shown in Fig. 6 revealed very high variability. The only difference between this new

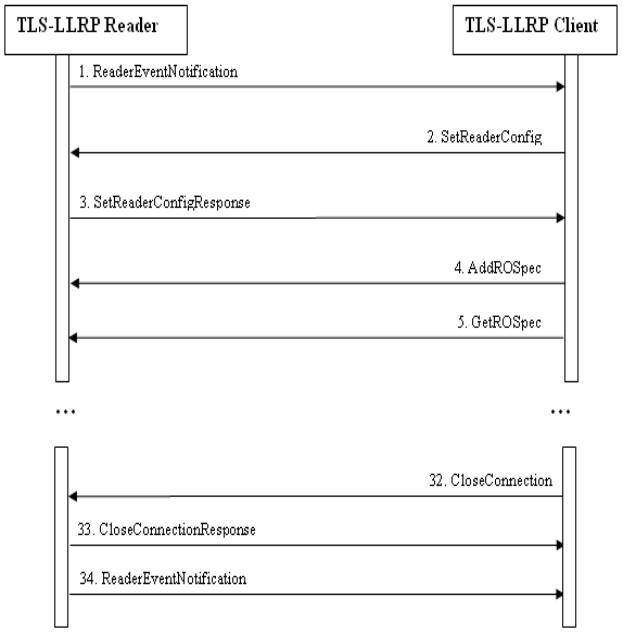

Fig. 6: Diagram of TLS-LLRP Client based on typical LLRP timeline

TLS-LLRP Client and the one used in i is that this TLS-LLRP Client sends one LLRP message multiple times to the TLS-LLRP Reader before terminating (See Fig. 7).

Generating timestamps for this metric was a very difficult task. Firstly, it required the use Network Time Protocol (NTP) to synchronize the clocks on the two machines. Secondly, a Java Native Interface (JNI) for C functions that return the precise time under Windows XP had to be written so that these function could be called from the Java TLS-LLRP programs.

Two other metrics that were used to indicate performance from a user's viewpoint are:

iii. The performance of the TLSL-LLRP Client – indicated by recording the time it took to initialize and interact with the TLS-LLRP Reader.

iv. The performance of the TLS-LLRP Reader – indicated by recording the time it took to serve the TLS-LLRP Client.

8. Performance Evaluation

In a business context, performance is paramount and large delays in these metrics because of the inclusion of TLS are unacceptable.

Without TLS, the time taken by metric i, iii and iv is given in the second column of Table 2.

The minimum overhead of adding TLS on Handshake Duration is 419.8 ms (TLS_RSA_WITH_AES_128_CBC_SHA, 1024 bit RSA keys) while the maximum is 1026.0 ms (TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA, 3072 bit RSA keys).

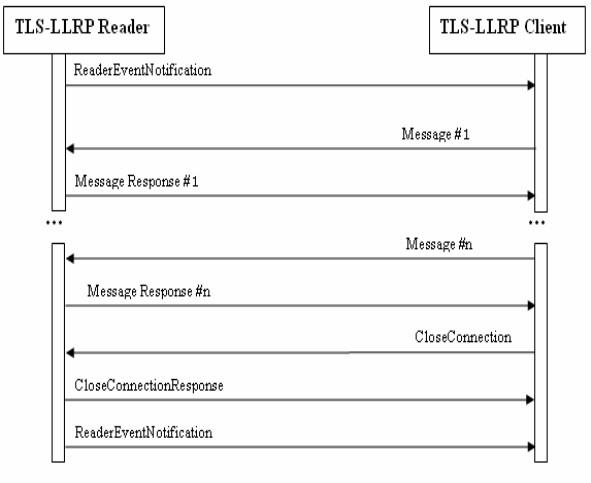

Fig. 7: Diagram of TLS-LLRP Client to investigate the Propagation Time of TLS-LLRP messages

The mean overhead on metrics i, iii and iv when different combinations of TLS are used is given in Table 3. It is easy to note that the largest overhead in all three metrics is observed when 3072 bit RSA keys are used and thus these combinations of TLS should be avoided.

Fig. 8 shows a plot of the five TLS combinations that result in minimum overhead on these three metrics. TLS_RSA_WITH_AES_128_CBC_SHA results in minimum overhead and this is probably the reason why it is the mandatory cipher suite should a Reader Protocol implementation support HTTPS [21].

RSA 1024 keys are to be replaced in the near future and larger RSA keys should be avoided as was mentioned above. It is thus imperative that a cipher suite should be identified that provides a higher level of security (than TLS_RSA_WITH_AES_128_CBC_SHA) but simultaneously has the smallest possible impact on performance. Since it is acknowledged that public key operations are most computationally expensive part of a TLS handshake, we use hypothesis testing ($\alpha = 0.1$) to arrange the key exchange methods shown in Table 3 in order of causing significant increase to the mean value of the performance metrics:

- ECDH_ECDSA / ECDHE_ECDSA
- RSA
- ECDHE_RSA
- DHE_RSA

We are thus able to recommend TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA (with ECC 224 bit keys and certificates signed using SHA1withECDSA) as the next best cipher suite when using TLS to secure a LLRP connection.

The above recommendation is based on the mean handshake duration but it remains to analyze the performance of TLS-LLRP connection as measured by the Propagation Time of TLS-LLRP messages. Essentially, the

Table 2: Mean values of metrics i,, iii and iv

Metric (ms)	Mean value (without TLS)	Minimal overhead of adding TLS	Maximum overhead of adding TLS
i	0	419.8	1026.0
iii	16206.9	895.6	1521.3
iv	16141.1	385	992.3

Table 3: Overhead of TLS combinations on metrics i, iii and iv (ms)

Cipher Suite	Alg / Key Size / Sign Alg.	i	iii	iv
TLS_RSA_WITH_AES_128_CBC_SHA *	***RSA / 1024 bits / SHA512withRSA***	***419.8***	***895.6***	***385***
TLS_RSA_WITH_AES_256_CBC_SHA		457.2	942.2	422.4
TLS_DHE_RSA_WITH_AES_256_CBC_SHA		507.5	986.7	473.1
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA		483.5	962	448.1
TLS_RSA_WITH_AES_128_CBC_SHA	RSA / 2048 bits / SHA512withRSA	460.1	947.9	425.1
TLS_RSA_WITH_AES_256_CBC_SHA		429.7	914.6	394.2
TLS_DHE_RSA_WITH_AES_256_CBC_SHA		585.8	1073.9	551.3
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA		684.2	1175.9	652.1
TLS_RSA_WITH_AES_256_CBC_SHA	RSA / 3072 bits / SHA512withRSA	678.6	1172	642.4
TLS_DHE_RSA_WITH_AES_256_CBC_SHA		935.6	1432.2	902.4
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA		991.8	1482.7	956.1
TLS_DHE_RSA_WITH_AES_256_CBC_SHA	RSA / 1024 bits / SHA1withRSA	494.7	968	461.3
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA		472.9	946	438.6
TLS_DHE_DSS_WITH_AES_256_CBC_SHA	DSA / 1024 bits / SHA1withDSA	561.6	1035.2	527
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	RSA / 2048 bits / SHA1withRSA	703.1	1185.4	668.6
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	RSA / 3072 bits / SHA1withRSA	1011.2	1495.7	976.5
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA	EC / 224 bits / SHA1withECDSA	480.2	964.6	445.5
TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA		427.4	905.5	391.9
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA	EC / 256 bits / SHA1withECDSA	464.7	951.1	429.1
TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA		440.4	931.2	405.7
TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA	EC / 256 bits / SHA1withECDSA	454.7	933.7	419.2
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA		488.8	975.6	453.9
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA *	***RSA / 3072 bits / SHA1withRSA***	***1026.0***	***1521.3***	***992.3***
TLS_DHE_RSA_WITH_AES_128_CBC_SHA		967.5	1456.8	934.2
TLS_DHE_DSS_WITH_AES_128_CBC_SHA	DSA / 1024 bits / SHA1withDSA	539.4	1022.1	505.5
TLS_RSA_WITH_AES_256_CBC_SHA	RSA / 2048 bits / SHA1withRSA	685.3	1168.6	645.3

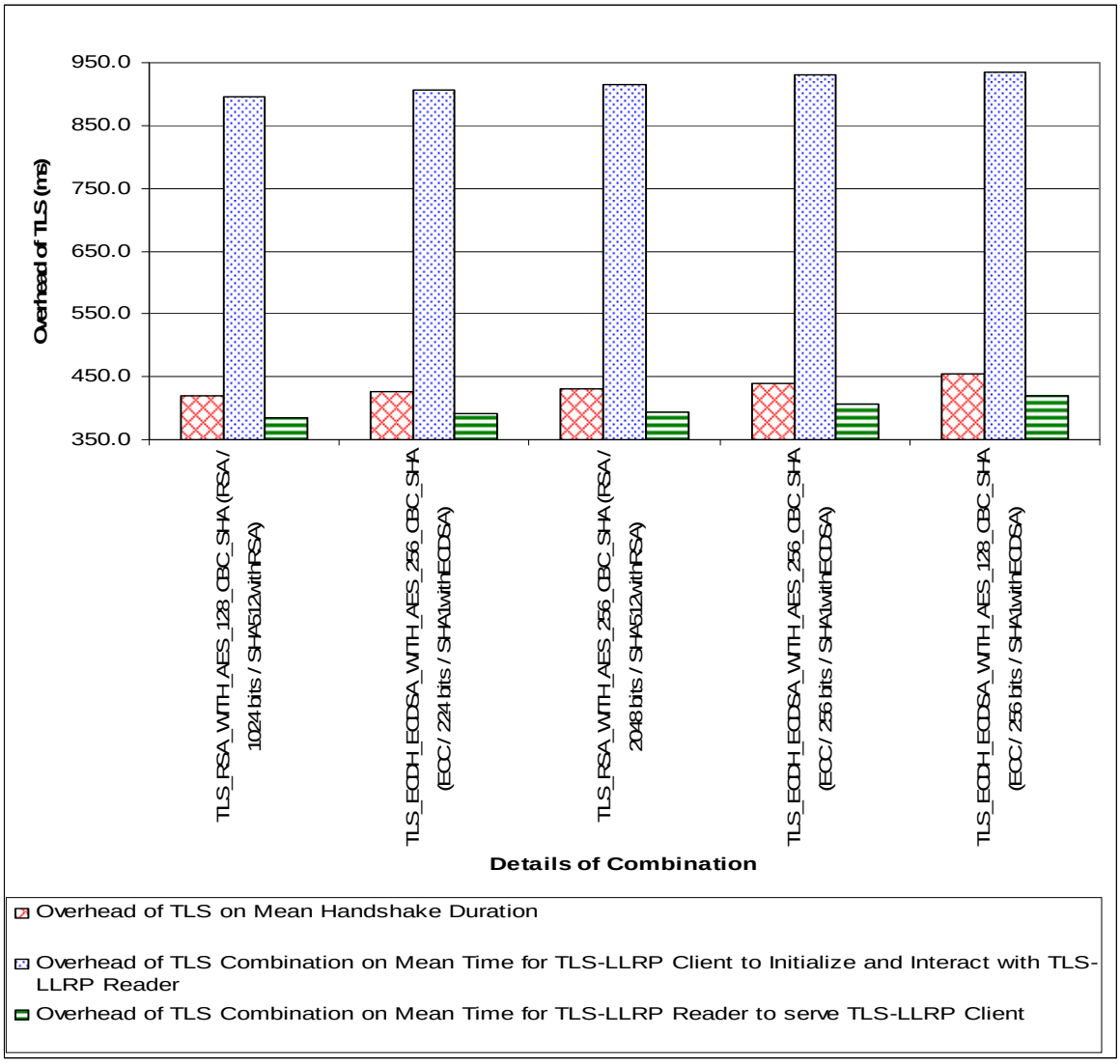

Fig. 8: Five TLS combinations that cause minimum overhead on metrics i, iii and iv

performance of the Record Protocol is affected by the use of AES-128 or AES-256. If we analyze the mean propagation time data for LLRP messages `SetReaderConfig` and `GetReaderCapabilities` (as shown in Table 4), we find:

- Using TLS (i.e. AES-128) increases mean propagation time by almost 3 times.
- Hypothesis testing (α = 0.05) reveals that the mean propagation time of `SetReaderConfig` and `GetReaderCapabilities` is statistically larger when AES-256 is used compared to when AES-128 is used.

Table 4: Analysis of Mean Propagation Time of TLS-LLRP message (ms)

LLRP Message	**No TLS**	TLS_RSA _WITH_ **AES_128_** CBC_SHA	TLS_RSA _WITH_ **AES_256_** CBC_SHA
`SetReader Config`	2.2	6.6	8.5
`GetReader Capabilities`	1.1	3.1	4.8

9. Conclusion

This work is an initial but important input into the security

of the new LLRP standard. Security requirements and possible solutions are discussed and TLS is selected for the securing a LLRP connection. A description of the development of the first TLS-LLRP endpoints is then given. This work also details the metrics selected to indicate performance and outlines the experimental procedures accordingly. The data collected revealed that adding TLS introduces an overhead of at least 419 ms due to its handshake protocol and almost triples the propagation time of LLRP messages. Further analysis showed that if the level of security is to be raised (i.e. beyond RSA 1024 bits), then TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA (with 224 bit ECC keys and certificates signed using SHA1withECDSA) should be adopted because of its minimal impact on performance.

Future work could include carrying out performance studies using C/C++ implementations of TLS. This will be possible only when open source C/C++ implementations of LLRP endpoints become available. Likewise, performance evaluation can be fine-tuned once a more accurate model of the workload of a LLRP session has been developed.

References

[1] *The EPCglobal Architecture Framework* [Online]. EPCglobal Standard. Available: http://www.epcglobalinc.org/standards/architecture/architecture_1_2-framework-20070910.pdf, 2007.

[2] *Low Level Reader Protocol (LLRP)* [Online]. EPCglobal Standard. Available: http://www.epcglobalinc.org/standards/llrp/llrp_1_0_1-standard-20070813.pdf, 2007.

[3] S. Samanta, "RFID reader agent based on low level reader protocol (LLRP) standard," M.S. thesis, Dept. Comp. Sci. & Comp. Eng., Univ. of Arkansas, United States, 2007.

[4] Rifidi. (2007). "LLRP HelloWorld Quick Start Guide" [Online]. Available: http://www.rifidi.org/llrp/LLRPQuickStartGuide.pdf.

[5] D. M. Dobkin. (2007, September). "A Brief Introduction to Low Level Reader Protocol (LLRP)" [Online]. Available: http://www.rfidtribe.com/home/index.php?option=com_content&task=view&id=138.

[6] Impinj Inc. (2007). "LLRP-Reader Control Simplified" [Online]. Available: http://www.impinj.com/WorkArea/downloadasset.aspx?id=2481.

[7] llrp.org. (2007). "llrp.org" [Online]. Available: http://www.llrp.org.

[8] D-H. Shih and P-L. Sun. (2005), Securing industry-wide EPCglobal Network with WS-security. *Industrial Management & Data Systems. 105(7)*, pp. 972-996.

[9] D.M. Konidala, W-S Kim, and K. Kim. (2006). "Security assessment of EPCglobal architecture framework" [Online]. Available: http://www.autoidlabs.org/uploads/media/AUTOIDLABS-WP-SWNET-017.pdf.

[10] T. Dierks and E. Rescorla, *The Transport Layer Security (TLS) protocol version 1.1* [Online]. Available: http://www.ietf.org/rfc/rfc4346.txt. 2006.

[11] S. Blake-Wilson, N. Bolyard, V. Gupta, C. Hawk and B. Moeller. *Elliptic Curve Cryptography (ECC) cipher Suites for Transport Layer Security (TLS)* [Online]. Available: http://www.ietf.org/rfc/rfc4492.txt. 2006.

[12] V. Gupta, D. Stebila and S.C. Shantz. (2004). "Integrating elliptic curve cryptography into the Web's security infrastructure" [Online]. Available: http://research.sun.com/projects/crypto/p915-gupta-final.pdf.

[13] G. Apostolopoulos, V. Peris and D.Saha, "Transport layer security: how much does it really cost?" in *Proc. IEEE INFOCOM*, March 1999, pp.717-725.

[14] G. Apostolopoulos, V. Peris, P. Pradhan and D. Saha, (2000, July / August). Securing electronic commerce: reducing the SSL overhead, *IEEE Network*. pp. 8-16.

[15] V. Gupta, S. Gupta and S. Chang, "Performance analysis of elliptic curve cryptography for SSL," presented at WiSE'02 [Online]. Atlanta, Georgia, USA, 2002. Available: http://research.sun.com/projects/crypto/performance.pdf.

[16] V. Gupta, D. Stebila, S. Fung, S. C. Shantz, N. Gura, and H. Eberle, (2004) "Speeding up secure web transactions using elliptic curve cryptography" [Online]. Available: http://research.sun.com/projects/crypto/ecc-ssl-ndss2004.pdf.

[17] A, Levi and E. Savas, "Performance evaluation of public-key cryptosystems in WTLS protocol," presented at the 8th IEEE Int. Symp. on Computers and Communication (ISCC 2003), Kemer-Antalya, Turkey., June-July, 2003.

[18] - (2007). "Accada EPC Network Prototyping Platform" [Online]. Available: http://www.accada.org. 2007.

[19] Rifidi. (2008), "Rifidi" [Computer Program] [Online]. Available: http://wiki.rifidi.org/index.php/Main_Page.

[20] Rifidi. (2007). "LLRP HelloWorld Sample Application" [Computer Program] [Online]. Available: http://www.rifidi.org/llrp/LLRPHelloWorldClient.zip.

[21] *EPCglobal Reader Protocol Standard* [Online]. EPCglobal Standard. Available: http://www.epcglobalinc.org/standards/rp/rp_1_1-standard-20060621.pdf, 2006.

[22] D. Hook, *Beginning Cryptography with Java*. Indianapolis: Wiley Publishing, 2005.