The Journal of CPRI,
Vol. 13, No. 3, September 2017 pp. 423-432

Hardening of the modbus protocol

Varun M Rao*, Rajesh Kalluri** and Ganga Prasad G L**

A large section of industrial control where network security is of paramount importance and where glitches can cause massive disruptions in societies has mostly been overlooked and ignored in recent times. Modern critical infrastructure assets (e.g., power plants, refineries and water supply systems) use ICT systems to provide reliable services and offer new features. Many maintenance and management operations at these installations involve the use of SCADA systems are controlled remotely using public networks, mostly over the Internet. While the automation and inter connectivity contribute to increased efficiency and reduced costs, they expose critical installations to new threats. Thus, issues relevant to the securing of this information when it's being transmitted via unsafe channels and unsecured protocols were chosen to be addressed. Various protocols that are used have either no provisions for secure transmission of its information or have outdated security structures. Our focus was on the Modbus protocol because of its wide application and lack of security features in the protocol structure. The objective was to establish a novel approach to the transmission via the Modbus protocol preserving the lower level attributes of transmission and at the same time adding a layer of security without adding significant delay.

Keywords: *Modbus, security, SCADA, ICT, encryption, key exchange,CHAP*

1.0 INTRODUCTION

Modbus is an application layer protocol that provides master and slave or client and server communications between devices connected to different buses or networks, it's basically a network protocol, whose primary purpose was to build a network from PLCs but its scope has grown exponentially as it is now used in plethora of different applications. There are three variations of the Modbus protocol according to the messages sending on the channel, i.e. Modbus RTU, Modbus ASCII and Modbus TCP/IP. Usually, the Modbus message contains three main fields, including recipient address, protocol data unit (PDU) and error checking field. The Modbus protocol has been extensively used in industrial control systems because of its ability to be fast and reliable. The Modbus TCP/ IP version is being used extensively because so many of our devices are today Ethernet and TCP-compatible. Thus work was done on the TCP/ IP version of the Modbus protocol. The Modbus protocol provides real-time communication between field devices that are located far away from each other via the Internet. The Modbus protocol consists of an OSI application layer messaging protocol and base with a client/server architecture model. The interconnectivity between devices can be achieved by employing the TCP/IP protocols that enable an exchange of messages via the Internet. Communication is started at the client by building an application data unit (ADU), and function codes that are placed in order to define the Modbus messaging meaning and the actions that shall be taken by the target device [1,6].

*varun.mrao@gmail.com, PES Institute of Technology, Bangalore, India.
** gpr@cdac.in, rajeshk@cdac.in, Centre for development of advanced computing, Bangalore

In figure 1, a header that is referred to as the Modbus application protocol header (MBAP) is employed to identify the Modbus ADU while the data is carried over the TCP/IP network; this is added irrespective of whether it's a request from the client side or a response from the server end. In Modbus TCP/IP, the CRC and the Slave ID get dropped and replaced by the MPAB header as can be seen in figure 1.

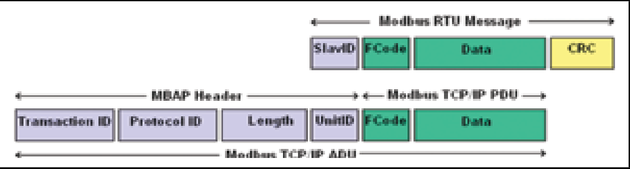


FIG 1: MODBUS TCP/IP AND MODBUS RTU/ASCII PDU

Modbus memory addresses which are used are referenced as coils (read/write - boolean), discrete input (read only - boolean), input registers (read only - int) and holding registers (read/write - int). Modbus protocol defines several function codes to access and manipulate the data in these memory addresses. Like for instance Modbus function code 1 is used to request the state of the Coils which is a boolean value (0 or 1).

Section two of this paper dwells into the vulnerabilities associated with the protocol and thus justifies the hardening of the protocol. Section three mainly talks about the experimental setup used for simulations before and after hardening from which the delays associated with hardening could be determined and thus draw inferences from our experiments. Section four elaborates the hardening methods and section five talks about the results and delays associated with different layers of hardening. Section six lays down the conclusions that can be drawn from observations in section five.

2.0 MODBUS VULNERABILITIES

The Modbus TCP protocol lacks provisions for protecting confidentiality and for verifying the integrity of messages sent between a master and slaves. Modbus TCP does not authenticate the master and slaves. Gabor, et al through a series of case studies [4] explains to us that notwithstanding that the Modbus protocol has been susceptible to forms of security attacks because of the absence of security provisions within the protocol in the past, [4] that he believes that the expanse of the internet has contributed to the increase in security attacks on the devices communicating via Modbus, thus justifying the need for hardening of the protocol. The protocol does not incorporate any anti-repudiation or anti-replay mechanisms. The security limitations of Modbus can be exploited by attackers to wreak havoc on industrial control systems.[2,3,5]

Some of the vulnerabilities with Modbus protocol are:

- Unauthenticated Command Execution: The lack of authentication of the master and slaves means that an attacker can send forged Modbus messages. In order to execute this attack, the attacker must be able to access the network that hosts the SCADA servers or the field network that hosts the slaves.
- Modbus Denial-of-Service Attacks: An example attack involves impersonating the client and sending meaningless messages to the server that cause them to expend processing resources and eventually crash.
- Man-in-the-Middle Attacks: The lack of integrity checks enables an attacker who has access to the production network to modify legitimate messages or fabricate messages and send them to the server and also view communications that go on across the network.
- Replay Attacks: The lack of security mechanisms enables an attacker to reuse legitimate Modbus messages sent to or from clients.

3.0 EXPERIMENTAL SETUP

To understand the impact analysis of an attack, an experimental setup is required [9,10]. Before hardening, a Modbus simulator needed to be chosen so that protocol could be hardened and tested on the simulator and also aid in measuring the performance of the hardened protocol. Using this criterion uModbus was selected as the simulator for our project and various versions of the simulator were developed for hardening and testing.

First, one simulator setup was developed which had a UI and the user could observe the hardening. Another version was designed with optimized codes specifically for measuring delay before and after hardening. Finally a version was designed to prove that the simulator works in a "real world environment" as it runs ideally for an infinite time period memory locations certain real world values and data type and proved the validity of the system.

For the purpose of simulating a "real world environment", the memory was broken up into multiple real world instances like voltage, frequency, circuit breakers etc. The server was also pre-initialized with values in the range of these real world values. JSON's were used for defining the instances and also the type (analog/ digital), names, tags, data types and whether or not these values can be rewritten or not, data addresses.

The idea was to prove the robustness of the concept of hardening in a real world scenario, so the client and server were running forever .The hardened Modbus protocol worked just as well as the normal unhardened Modbus protocol with a slight delay. The program did not crash and thus proving that the simulator was working without any glitches and can run in a real world scenario where data is continuously being updated. In the process floating point data support was also integrated into the simulator. The float value was split and stored in two memory addresses.

4.0 HARDENING OF THE PROTOCOL

The TCP version of the Modbus protocol introduces complexity with respect to managing the reliable delivery of messages and at the same time maintaining strong real-time constraints. As was seen in section three on Modbus vulnerabilities, the protocol is susceptible to many different types of attacks; the Unauthorized Command Execution and Man in the Middle attacks were identified as the problems that this hardening approach would be able to tackle.

This paper suggests a novel approach to harden the Modbus protocol. To harden the protocol against attacks a twofold approach was used:-

1. Encrypt the data so that the data in flight can be secured.

2. Authenticate the clients with tailor made multiple authentication schemes for modbus protocol.

The encryption helps prevent people intercepting the data from understanding the functions being performed and thus shields the user from a Man in the Middle attack. The authentication helps prevent unauthenticated clients from accessing information and generating random packets. Both authentication schemes are compared to measure efficiency.

4.1. Encryption

Encryption is a process of disguising a message or information with a certain way so unauthorized people do not understand the content of the message or information. Encryption has a close relation to the decryption process. There are thousands of encryption schemes out there and a decision needed to be made on what encryption scheme could be chosen and this was done taking into consideration two important features in the encryption schemes:-

1. It needs to be extremely fast because of the Modbus protocol's real time constraints and the nature of applications where the protocol is used. A high delay would defeat the point of the hardened protocol.

2. It needs to be reliable, unbroken and "tried and tested". This will ensure there is no possibility of people being able to break the encryption and obtain the data anyways, again defeating the purpose of the entire hardened protocol.

Based on these requirements Advanced Encryption Scheme (AES) was chosen as the encryption scheme instead of ones like DES, Blowfish, Hummingbird etc.[7,8] since it's extremely fast and reliable.

For this implementation, the pyCrypto module was used which the AES (Advanced Encryption Scheme) algorithm had written in python, so this was integrated into the server and client code wherever there was a transmission of data via the Modbus port 502. All the requests and responses were encoded using AES(Advanced Encryption Scheme)and then sent over the port 502, thus ensuring the safety of data while in flight. The AES-CBC mode with 256-bit key size was used and the key was randomly generated and the IV was hard coded because the key kept changing it was okay for the IV to remain the same.

Another aspect to the AES encryption was the padding the message before encryption, AES works only on multiples of 16-bit data and so if the request or response wasn't big enough it needed to be padded to increase the size to 16 bits so that it could be encrypted using AES(Advanced Encryption Scheme). For this the PCKS5 packing scheme was used, it basically finds a value to use for padding based on the difference to the nearest 16th multiple and request/ response length. Using this scheme the messages were padded using some special character which is different for every message and because it was padded it could be sent in for encryption using AES(Advanced Encryption Scheme).

Thus even in the event that there is a Man in the Middle Attack and the hacker intercepts the packet, he doesn't understand the contents of the packet and thus cannot figure out what is going on. If he randomly changes the value of the packet the decryption at the other end fails, thus ensuring that there is nothing that the Man in the Middle can do to disrupt the transmission of data whilst it's in flight.

4.2 Diffie-Hellman Key Exchange

The Diffie-Hellman algorithm helps generate cryptographic keys across a public channel [8]. It was used in order to make the AES-CBC mode full proof and almost unbreakable from threats like guessing of initialization vector and key. Thus the key at the Server and the Client didn't need to be hardcoded. This communication between Server and Client used a different port so that there is no possibility of corruption of data during transmission. Port 9015 which is not a reserved port, was used for this purpose. A python package pyDHE was used for this key exchange and the code needed for the calling of this function was integrated at the all necessary situations at both client and server.

Thus after servicing a certain number of requests, this is configurable and can be set according to the usage and how much delay can be levied. The client and server first generate their own public and private keys. They then exchange their respective public keys to be able to generate a shared key which is generated using a previously agreed prime number a primitive root and the other side's public key. Thus using the shared key as part of the AES (Advanced Encryption Scheme)algorithm, it makes the hardening full proof and can't be understood by a "Man in the Middle".

```
Server Pseudo-Code:
client-connected = false
While true
  Wait for client connection on port 502
  client-connected = true
  While client-connected {
    Generate key pair (public/private)
    Exchange public keys
    Generate shared session key
num-requests = 0
     While client-connected AND num-requests <
requests-per-session {
      While client session active {
```

```
      process decrypted client request
      Send encrypted response to client
      increment num-requests
    }
    if connection terminated {
      // client terminated connection
      client-connected = false
    } else {
      break
    }
  } } }
```

Client Pseudo-Code:

```
server-connected = false
Set up socket and connect with server on port 502
server-connected = true
While server-connected {
  Generate key pair (public/private)
  Exchange public keys
  Generate shared session key
num-requests = 0
  While server-connected AND num-requests < requests-per-session {
    While server session active {
      Generate and encrypt client request
      Decrypt server responses
      increment num-requests
    }
    if connection terminated {
      // client terminates connection
      server-connected = false
    } else {
      break
    }
  }
}
```

4.3 Authentication Schemes

uModbus in itself has a low-level authentication scheme on the basis of the slave ID of the client. For making sure the authentication is more secure the Challenge Handshake Authentication Protocol (CHAP) was used because it's considered reliable and fast means of authentication. Again for this, the port 9015 was used so that there is no corruption of actual data being communicated via the Modbus protocol.

In the CHAP (Challenge Handshake Authentication Protocol) one decides the kind of Challenge that is sent be the server to authenticate the client and in this regard, two different challenges and two different authentication criteria were tried in this regard.

In the first challenge the concept of "RSA-digital signatures" was used, the server generates a random number, calculates it's hash value using SHA and sends the random sequence to the client. When the client when it's being authenticated for the first time, it shares a public key with the server. The client uses a private key which is generated and signs the hash generated using SHA of the random number sequence. This is then sent to the server and even if someone intercepts and changes the packet, the hash value calculated will change and the protocol will fail. This was implemented using the pyCrypto module which has the libraries for RSA-digital signatures as well as SHA, therefore both signing and calculation of hash were done using the module.

Pseudo Code of RSA-Signature Scheme – Server Side

```
if (num_reqests_serviced>authentication_interval) {
Public_key = Receives public key from client
challenge = Generates random number as challenge
  Sends the challenge to the client
hash_value_server = SHA ( challenge )
  Receives response client
hash_vlaue_client = Decrypt (Response from server, Public_key)
    if (hash_value_server == hash_value_client ){
Client is verified and the next request is accepted
    }
    else {
      Terminate connection with client
    }
}
```

Pseudo Code of RSA-Signature Scheme – Client Side

```
if (num_reqests_serviced>authentication_interval) {
 Generate key pair (Public and Private)
 Send the Public Key to Server
   challenge = Recieves random number as challenge
hash_value_client = SHA ( challenge )
  response = RSA (hash_value_client, Private_key)
 Send encrypted response to server
   if (verified ){
      Client can continue to generate new requests
   }
   else {
      Client's connection is terminated and can't continue to generate new requests
   }
}
```

The second challenge was a novel approach devised thinking of knowledge that would exist at only the server and at authentic clients. It basically involves authenticating the client based on the previous history of requests that have been serviced. The server and client store all the requests that are being/ going to be serviced. To authenticate the client, the server generates a random number between 0 and the number of requests that have been serviced up to that point in time, this is then encrypted using AES and sent over to the client with a different initialization vector but the same key which was generated using the Diffie-Hellman key exchange. The client then decrypts the request using the same initialization vector and the key. Then the request is taken and it's encrypted and sent over to the server again using the same initialization vector and key. The server decrypts the encrypted request and verifies it against the list of requests that it has serviced. Thus the client is authenticated. The question then arises as to how to authenticate a first time client. The solution to this is pretty simple; the arrays which store the serviced requests are pre-initialized with some random value. The first time authentication is done using a pre-initialized value because only authentic clients will know this response when asked for the first-time authentication.

Pseudo Code of Previous history of serviced requests

Signature Scheme – Server Side

```
if (num_reqests_serviced>authentication_interval) {
  // number_of_requests_serviced array keeps getting updated every time a request is serviced
// Session key from Diffie-Hellman is used
        challenge= Random(0,Length(number_of_requests_serviced -1 ))
encrypted_request = AES_Encrypt ( challenge, session key )
 Sends the challenge to the client
resp = Receives response client
client_resp = AES_Decrypt (resp,session key )
 if (client_resp == number_of_requests_serviced[challenge ] ){
        Client is verified and the next request is accepted
   }
   else {
     Terminate connection with client
   }
}
```

Pseudo Code of Previous history of serviced requests

Signature Scheme – Client Side

```
// number_of_requests_generated array keeps getting updated every time a request is serviced
if (num_reqests_serviced>authentication_interval) {
    challenge = Receive challenge from server
challenge_index = AES_Decrypt( challenge )
        response = number_of_requests_generated[challenge_index]
encrypted_response = AES_Encrypt (response)
 Send encrypted_response to server
   if (verified ){
Client can continue to generate new requests
```

} else {
Client's connection is terminated and can't continue to generate new requests
} }

TABLE 1

MODBUS DELAY THEORETICAL PERFORMANCE EVALUATION

Operations/ Modbus Functions/ Value being written	Request/ Response packet size	Number of Coils/Registers being accessed	Delay without Hardening (ms)	Delay with Hardening -Only AES w/o Diffie (ms)	Delay with Hardening - Only AES with Diffie (ms)	Delay because of Diffie Key Exchange (ms)	Delay using CHAP-1 (ms) with AES [RSA Digital Signatures]	Delay using CHAP-2 (ms) with AES [Challenged with prev responses]	Delay using CHAP-2 (ms) with AES ♦Diffie Delay	Delay using CHAP - 1 (ms) with AES +Diffie Delay
Read Coils										
	12/ 16	50	0.2430	0.2974	0.7243	0.4269	36.6863	0.3661	0.7930	37.1132
	12/22	100	0.4054	0.4886	0.8756	0.3871	37.8271	0.5279	0.9150	38.2141
	12/28	150	0.6571	0.7032	1.1308	0.4276	37.3587	0.7380	1.1656	37.7883
	12/34	200	0.8735	0.9449	1.5885	0.6436	37.7688	1.4418	2.0853	38.4124
Read Holding Registers										
	12/49	20	0.1341	0.2501	0.6062	0.3561	39.4592	0.2728	0.6289	39.8153
	12/ 109	50	0.2002	0.2848	0.7320	0.4472	37.2518	0.3375	0.7847	37.6990
	12/ 159	75	0.2734	0.3277	0.8057	0.4781	37.4217	0.4042	0.8823	37.8998
	12/209	100	0.3593	0.4017	0.8717	0.4700	37.7473	0.4840	0.9540	38.2173
Write Single Coil										
0	12/12	1	0.0913	0.1453	0.5270	0.3817	37.5449	0.2038	0.5855	37.9266
1	12/ 12	1	0.0955	0.1401	0.5327	0.3926	38.4735	0.2197	0.6123	38.8861
Write Single Register										
50	12/ 12	1	0.1005	0.1464	0.5227	0.3763	37.5372	0.2075	0.5838	37.9134
100	12/12	1	0.0965	0.1432	0.5354	0.3922	37.1873	0.2185	0.6107	37.5794
500	12/12	1	0.0965	0.1409	0.5803	0.4394	36.5825	0.2193	0.6586	37.0218
1000	12/ 12	1	0.0945	0.1419	0.5512	0.4093	37.7341	0.2161	0.6254	38.1434
2000	12/ 12	1	0.0997	0.1421	0.5876	0.4456	37.4849	0.2168	0.6624	37.9304
10000	12/ 12	1	0.1046	0.1391	0.5707	0.4316	37.7697	0.2211	0.6527	38.2013
Writing 10000 multiple registers	53/12	20	0.1750	0.2219	0.6346	0.4127	36.8150	0.2924	0.7050	37.2277
	113/ 12	50	0.2868	0.4062	0.7728	0.3666	37.6507	0.4190	0.7856	38.0173
	163/12	75	0.3997	0.5058	0.8717	0.3659	37.2369	0.5290	0.8950	37.6028
	213/12	100	0.5069	0.6248	1.0254	0.4005	36.7746	0.6662	1.0667	37.1752

5.0 RESULTS

5.1 Setup for Calculation of Delay

The optimized code was written with only necessary features so that the delay which was being added because of hardening could be calculated. This was done by getting rid of all the print statements and also changed the value of prime used in the Diffie-Hellman key exchange to a 16 digit prime number (1111235916285193) because e ^(prime) was a cost intensive operation and was taking a lot of time. After trying a few combinations it was found to be reasonably complex and also fast to compute, so the 16 digit prime was used everywhere. The server code was written such that it did not have any pre-initialized values or prints. For the client, individual scripts were written for each of the Modbus functions and each of the hardening scenarios, so that the delays for each function could be computed individually. For the evaluation, these functions were performed over 1000 iterations, measured the time taken every time and then went ahead and averaged it. For each stage, the delay was calculated showing us how much time delay each segment of the hardening was adding.

5.2 Graphs and Analysis from results

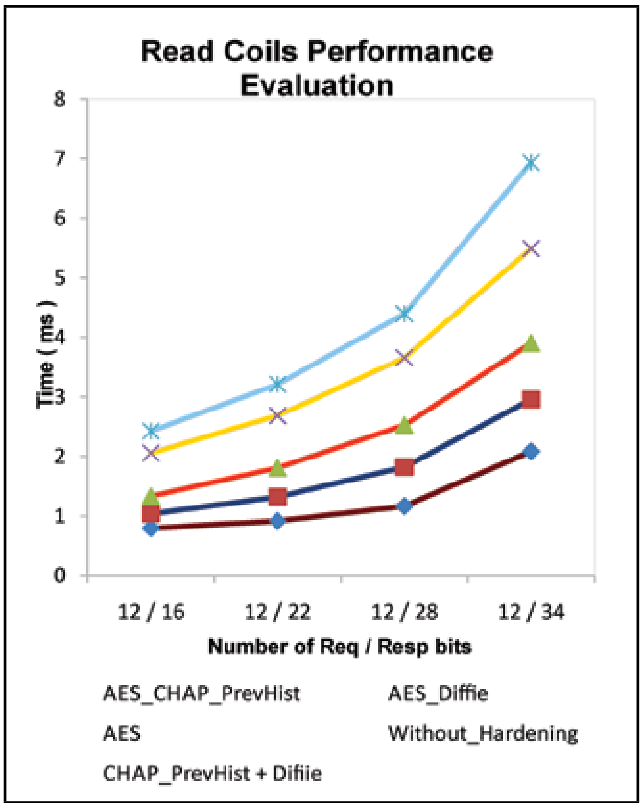
Using the optimized server and wrote specific codes to evaluate performance of the hardened Modbus protocol for each function of Modbus and also for each and every hardening scenario – Only AES, AES + Diffie, Diffie, AES + CHAP etc.

Notes :

1. Only the read coils command graph has been plotted because the trend across all other graphs is similar as well, as can be seen in the values mentioned in the table.
2. In the first graph the AES_CHAP_With_RSA signatures has been divided by 14, so that the other values can be comparable on the graph.

Although the delays associated with all functions were computed from the table it can be observed that the trend of delay associated with all the parts of hardening is showing a similar trend. Thus one set of function codes has been graphed to divulge necessary information using a graphical approach. From this, a few points can be observed -

From the table it can be seen that CHAP authentication using the previous request history is much faster than the traditional CHAP using the RSA signature scheme. It's more than 14 times faster than the CHAP using the RSA-digital signature scheme which can be seen from the first graph. Thus it's a better fit in situations where the time constraints are in the order of nanoseconds. When the time constraint is in the order of a few seconds, the RSA digital signature approach can be used. Also, it can be seen from the second graphthat the delay keeps increasing as more elements or security layers are added to the hardening. Each layer adds its



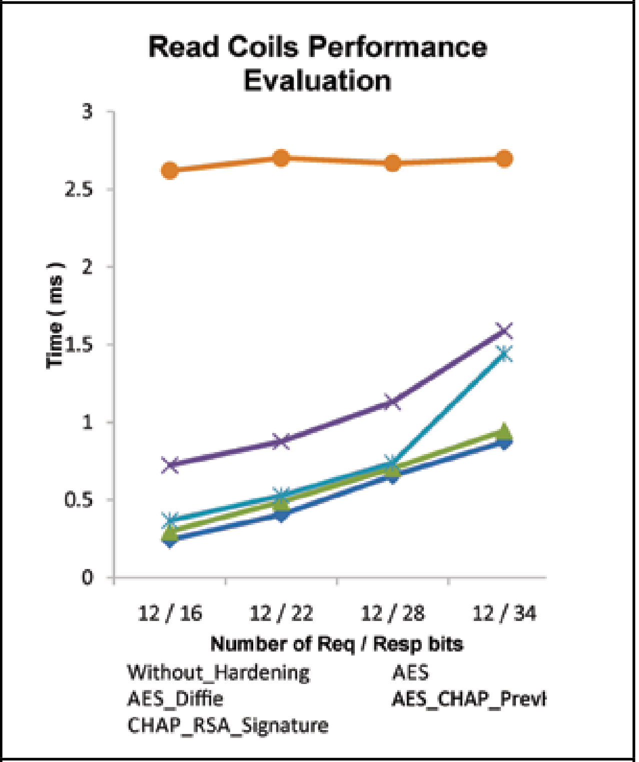


FIG 2: GRAPHS OF READ COILS PERFORMANCE

own delay and adds to the total delay associated with the hardened protocol; this can be seen from the graph as the delay keeps on increasing with each new layer of hardening added to the protocol. Also it can be observed that after the request/ response bits goes above 12/28 the time taken increases and shape of the graph changes, this might be system related eccentricities and further experimentation is required in a host of different testing scenarios to be able to explain the reason for the graph taking such a shape. It can also be seen that the Diffie-Hellman key exchange is adding a significant delay as compare to the simple AES encryption scheme and so for each application an acceptable frequency of key exchange needs to be determined which at the same time not compromise on the security of the algorithm.

6.0 CONCLUSION

The objective of the project was to be able to harden and increase reliability and security of the Modbus protocol. This was achieved by using already existing cryptography schemes and techniques to device a new and more secure approach to transmission via the Modbus protocol building on pre-existing knowledge of the protocol and adhering to the protocol requirements. The results at each stage of hardening have been tabulated; therefore people who are looking to add security layers to their transmission can do so by looking at the paper and using the level of hardening which suits them the best.

As expected it was seen that on adding multiple layers to the hardening of the protocol the delay associated with the hardening went up correspondingly. But also that the CHAP (Challenge Handshake Authentication Protocol) authentication scheme using the previous history of serviced requests was way faster than the RSA-digital signature authentication scheme. Another added benefit of the hardened simulator is the nature of ability to configure the number of times the key exchange and authentication happen between cycles by simply changing the JSON configurations.

REFERENCES

[1] Modbus, Modbus application protocol specification V1.1b3,

[2] Aamir Shahzad, Malrey Lee, Young-Keun Lee Suntae Kim, Naixue Xiong Jae Young Choi and Younghwa Cho, "Real Time Modbus Transmissions and Cryptography Security Designs and Enhancements of Protocol Sensitive Information" in Symmetry Open Access Journal, 2015.

[3] Igor Nai Fovino, Andrea Carcano, Marcelo Masera and Alberto Trombetta, "design and implementation of a secure modbus protocol" in ICCIP: Critical Infrastructure Protection, pp. 83-96, 2009.

[4] Gabor jakaboczki, eva adamko, "vulnerabilities of modbus rtu protocol – a case study" in annals of the oradea university fascicle of management and technological engineering issue #1, May 2015.

[5] Zakarya drias, Ahmed serhrouchni and Olivier vogel, "Taxonomy of attacks on Industrial Control protocols" in Conference on Protocol Engineering (ICPE) and International Conference on New Technologies of Distributed Systems (NTDS), IEEE, 2015.

[6] The Modbus Organization. Modbus Messaging on TCP/IP Implementation Guide V1.0a; Modbus Organization: Hopkinton, MA, USA, pp. 2–15, 2004.

[7] Nikita Arora, Yogita Gigras, Block and Stream Cipher Based Cryptographic Algorithms: A Survey in International Journal of Information and Computation Technology. ISSN 0974-2239 Vol. 4, No. 2, pp. 189-196, 2014.

[8] Stallings, William. Cryptography and Network Security Principles and Practice. Boston: Pearson, 2011.

[9] Abhiram Amaraneni, Mahendra Lagineni, Rajesh Kalluri, Senthil kumar R.K, Ganga Prasad G.L "Transient analysis of cyber-attacks on Power SCADA using RTDS" the Journal of CPRI, Vol. 11, No. 1, March 2015

[10] Samanth P, R Kalluri, RK Senthil Kumar, BS Bindhumadhava. 'SCADA communication protocols: vulnerabilities, attacks and possible mitigations' at CSI Transactions on ICT ISSN 2277-9078 during April 2013