XTEA based Secure Authentication Protocol for RFID Systems

Gul N. Khan, Jack Yu and Fei Yuan
Department of Electrical and Computer Engineering, Ryerson University,
Toronto, Ontario M5B 2K3 Canada

***Abstract* - RFID technology has been widely used in logistic, automation and authentication applications, but it still has many potential issues such as the risks of privacy and security. This paper presents a novel RFID security protocol based on XTEA algorithm. Analysis of the security and privacy of this novel protocol is performed using FPGA based prototyping platform. Different attack models are implemented, and the results show that the protocol is robust and safe against major attacks. Analysis of the performance is compared with related works and shows its advantages in code size, clock cycle, communication cost and scalability.**

Keywords - RFID Security; Product Authentication; Robust and Secure Protocol; XTEA

I. INTRODUCTION

Although many researchers have been working on the area of RFID security and hundreds of articles have been published during recent years, finding a good solution still remains challenging. The essential reason causing the difficulty is the dynamic range of technologies RFID combines with and the wide area of its application. The research work by Juels has made a valuable insight to this reality [1]. Any practical design of security protocol has to strike a balance between the security level, cost and performance.

In this paper, we investigate some related works regarding RFID security techniques and protocols. Then, we propose a novel security protocol based on eXtended Tiny Encryption Algorithm (XTEA).The security protocol is prototyped and analyzed using FPGA platform. We present the experimental results of different types of major attacks on our scheme to evaluate its security and privacy level. We also evaluate the performance of our scheme by comparing with similar works that also use symmetrical key algorithms. We conclude the paper by summarizing its main features and by pointing out some future directions.

II. RELATED RESEARCH

A security protocol consists of the encryption algorithm used to hide the message from attackers and the processes of how the messages are exchanged. The encryption cipher used in a protocol is very important. Since a back-end server has more computational resource than a tag, the efficient implementation of an RFID protocol for passive tags relies on lightweight ciphers. The ideal cipher must have a small size, consume less power and provide satisfactory level of security.

Some previous research concentrated on efficient implementation of AES (Advanced Encryption Standard). Martin et al. introduced an efficient implementation of AES algorithms in CMOS 0.35 µm process [2]. The results show that the AES-128 data path implementation achieves a current consumption of 8.5 µAmp at frequency of 100 kHz. It needs 1016 clock cycles to encrypt a 128-bit data block and the hardware takes about 3595 gates. Based on these results, some previous works on RFID protocols based on AES algorithm were proposed. Toiruul et al. proposed an advanced mutual authentication protocol to address three major problems: forgery of the tags, unwanted tracking of the tags, and unauthorized access to tag memory [3]. Kim et al. also proposed a RFID authentication protocol using step by step symmetric key change that based on AES encryption and decryption method [4].

However, Tiny Encryption Algorithm (TEA), which also belongs to symmetric key category in cryptography, has become of interest for RFID security solutions during recent years. TEA was proposed by Roger Needham and David Wheeler in 1994 [5]. However, it has been reported that it suffers from key attacks and its actual key size is reduced to 126 bits [6]. In order to overcome these weaknesses, Roger Needham and David Wheeler proposed XTEA algorithm [7]. Although XTEA remains as compact as TEA, the key scheduling has been changed and its key material is introduced more slowly.

Jens-Peter Kaps's work made an in-depth investigation of ultra low power implementation of XTEA algorithm on FPGA [8]. An efficient implementation of XTEA and TinyXTEA-3 were introduced and developed. His work uses Xilinx FPGA devices for prototyping. He also compared with other AES related implementations on FPGA. The smallest AES implementation is the 8-bit AES by Good and Benaissa [9]. It consumes a similar amount of FPGA area as compared to XTEA-3 implementation, but it takes more clock cycles. In another work by Chodowiec and Gaj, AES algorithm consumes 112 clock cycles, which is the same as of XTEA-3, however its FPGA area is around 522 slices [10]. It indicates that XTEA algorithm is more efficient in ultra-low power and low resource environment than the AES algorithm. However, very few RFID security protocols are based on XTEA algorithm so far.

III. XTEA BASED SECURITY PROTOCOL

A. System Assumptions

To solve RFID security and privacy issues using a robust solution, we propose a novel security protocol based on XTEA algorithm. In this protocol, it is assumed that the RF-tag has a 64-bit non-volatile ID and a 128-bit key set that can be dynamically updated. It is also assumed that in the application, the last 30 bits of the tag ID are different and they are enough to represent each tag in a system. The RF-tag has the encryption and decryption capability using

XTEA algorithm. One key set consists of four sub-keys of 32 bits each. Each tag and back-end server share a random secret key set at initialization.

B. Robust Protocol Design

Our proposed RFID protocol has various steps that are depicted in Fig. 1. The protocol message exchange process is described as given below:

1. The reader sends a query message to read the tag.
2. Once being queried, the tag transmits an encrypted tag ID encrypted using XTEA algorithm with current key set k0.
3. The reader forwards this message to the back-end database for authentication.
4. The back-end database checks the database of expected response of known tags to find the corresponding tag ID. Then the back-end database immediately generates a key update message and its acknowledgement message. The message format of key update message is shown in Fig. 2. It consists of a field containing the last 30 bits of the tag ID, a 2-bit field for the sub-key number to be updated as well as the sub-key value of 32 bits. The message is of 64 bits in total length and must be encrypted using the current key set k0. The encrypted message is 64 bit in length too that is unreadable for outsiders. The back-end server computes the tag response for the next query based on the new key set k1. Meanwhile, it backups the current response just in case the key update failed for some reasons.

   The back-end server also generates a key update acknowledgement in a format shown in Fig. 3 that is also 64 bits wide and is encrypted with next key set k1. It consists of the first 32 bits of the tag ID and next 32 bits contain the previous sub-key that is to be replaced. The reader sends the tag ID, the key update message and the key update acknowledgement to the reader.
5. The reader forwards this key update message to the tag. The tag decrypts the message using the current key set k0 and verifies the field of the last 30 bits of tag ID to confirm a match.

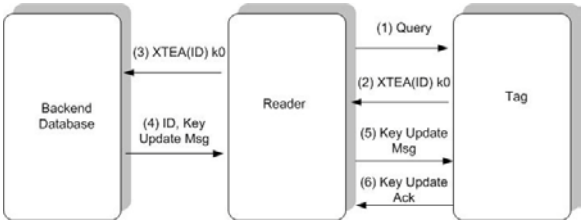

Figure 1. Novel Protocol Based on XTEA Algorithm

Last 30 bits of Tag ID	Subkey#	New Subkey Value

Figure 2. Key Update Message Format

If verification succeeds, it continues to extract the sub-key value and number to update the key set to k1. If this process fails, it continues to wait for another key update message for a timeout window.

6. After receiving the key update and verification passes, the tag generates a key update acknowledgement in the format shown in Fig. 3 and sends it back to the reader. The encryption uses previous key set.
7. The reader verified if the correct key update acknowledgement is received using the current key set. Successfully receiving the acknowledgement message terminates the authentication process.
8. If the reader fails to receive the acknowledgement it resends the key update message. The tag will respond with acknowledgement again even if it has already sent out the acknowledgement previously. In the real implementation, the reader can re-send this key update message several times before warning the back-end the tag is faked. If the key-update process fails because the tag is really a faked one, the tag record in the back-end database should be reverted to the previous value.

IV. Implementation of Attack Models

In this section, we implement our scheme under active scanning, eavesdropping, replay, denial-of-service, and man-in-the-middle attack models using Altera DE2 board [11]. The FPGA hardware system is generated using SOPC builder tool and programmed onto the device using the programmer utility provided in the Quartus-II environment. Communication is achieved through JTAG UART interface.

A. Replay Attack

In replay attack, the reader and the tag are communicating in a channel which is insecure and attacker can intercept the message coming from the tag for the reader. Fig. 4 shows the Nios-II system of this model. First, the reader queries the tag and the attacker records the tag response during the authentication process. The reader then queries the attacker to see if the attacker can be authenticated by replaying the message it records.

The initial tag ID is {0x22332212, 0x21314783} and the key set is {0x247dc618, 0x70bc93e4, 0x902c729a, 0xab47856c}.The sample of result in the first 3 sessions is summarized in the Table I. The implementation shows that the tag not only replies with the encrypted message in every session but also update the one sub-key of its key set. The replay attacker tries to replay the previous tag response but cannot be authenticated. The implementation of replay attack model verifies that our novel protocol is safe against replay attack.

First 32 Bits of The Tag ID	Previous Sub-key Switched

Figure 3. Key Update Acknowledgement Format

B. Active Scanning Attack

In active scanning attack, the tag is isolated from the legitimate readers and the attacker is constantly scanning the tag without owner's authorization. The purpose of this attack is to find if it can keep sending useful messages. The message exchanges that happen in the first 3 sessions are shown in Table II. As one can see from tag response, the

reply message is encrypted and it is not meaningful even if it is exposed to the attacker. It indicates that our protocol is safe against active scanning attack.

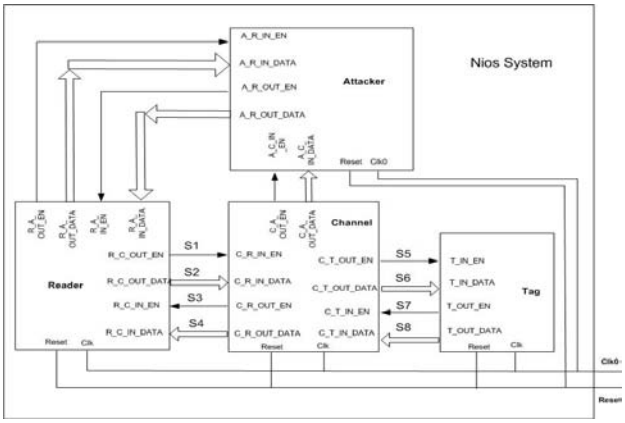

Figure 4. Replay Attack Model

TABLE I. REPLAY ATTACK AND RESPONSE

Session	Tag Response Message	Tag Key Set	Attacker Replay Message
1	B5718393,0 FE7A64B	AB47856C, 902C729A, 70BC93E4 , 247DC618	B5718393, 0FE7A64B
2	AF4D7FD9, DDF60A22	AB47856C, 902C729A, 70BC93E4, DA6B4E51	AF4D7FD9, DDF60A22
3	97E6D6FC, 8072E812	7825DFCB, 902C729A, 70BC93E4, DA6B4E51	97E6D6FC, 8072E812

C. Eavesdropping Attack

In eavesdropping attack, the reader and the tag communicate in a channel which is insecure and attacker can intercept the message coming from the tag and reader. In the implementation, the reader first queries the tag in a normal operation. Then, the reader checks whether the message recorded by the attacker matches the secrets it has or not.

TABLE II. ACTIVE SCANNING ATTACK AND RESPONSE

Session	Message Sent by Attacker	Tag Response Message
1	Tag Query	B5718393,0FE7A64B
2	Tag Query	B5718393,0FE7A64B
3	Tag Query	B5718393,0FE7A64B

As observed from the result shown in Table III, the tag replied with the encrypted message in each new session. The eavesdropping attacker cannot obtain the secret of the tag because it does not know the key set. Our implementation of eavesdropping attack model verified that the protocol is safe against eavesdropping attack.

D. Denial-of- Service Attack Model

In denial-of-service attack, the reader and the tag will communicate via a communication channel, which can be blocked by the attacker. The denial-of-service attacker blocks the key update message and then allows it to pass when the reader resends it. The samples of result in the first 3 sessions of the operation are summarized in the Table IV. The result shows that the key update operation of tag is successful. The result shows our protocol is safe against denial-of-service attack.

E. Man-in- the-Middle Attack Model

In the case of a man-in-the-middle attack, the communication between the reader and the tag is intercepted and modified by the attacker. In the implementation shown, the reader first queries the tag and receives the tag reply. When the key update message arrives, the man-in-the-middle attacker tries to randomly modify and then sends it to the tag. The result from the session 28 to 32 is summarized in Table V.

The tag decrypts the key update message, but it will find the field with the last 30 bits of its tag ID does not match. In the timeout period, the reader does not receive the correct key update acknowledgement message. Therefore it resends the key update message. The key update message acknowledgement from the tag is an important indication that the tag is synchronizing with the reader.

TABLE III. EAVESDROPPING ATTACK AND RESPONSE

Session	Tag Response Message To Query	Tag ID
1	B57183930FE7A64B	2233221221314783
2	AF4D7FD9DDF60A22	2233221221314783
3	97E6D6FC8072E812	2233221221314783

TABLE IV. DENIAL-OF-SERVICE ATTACK AND RESPONSE

Session	Tag Response Message	Tag Key Update Message	Key Update Reader Resend	Key Update ACK
1	B5718393, 0FE7A64B	8EA3D37C, 3F5198CA	8EA3D37C, 3F5198CA	2B6EDC58, 1B49A4D8
2	AF4D7FD9, DDF60A22	BE0B6D17, 27D8F698	BE0B6D17, 27D8F698	A7C5352E, B059EBD
3	97E6D6FC, 8072E812	70F67232, FAA7DB1A	70F67232, FAA7DB1A	110D71D7, 99059299

TABLE V. MAN-IN-THE-MIDDLE ATTACK AND RESPONSE

Session	Tag Response To Query	Tag Key Update Message	Key Update Reader Resend	Tag Key Update ACK
28	BE0F3DF6, DD11E327	7C6F3423, DA1EE993	7C6F3423, DA1EE993	51B7093F, 4E4B5E76
29	2962986F, 2B17C3FA	F6D59CA9, F92AF59B	F6D59CA9, F92AF59B	685BF097, DBECF678
30	D7F46FAE, F0FE0ADF	63C4CFEF, C12C399C	63C4CFEF, C12C399C	A8A9C52, 4F35367B

V. PERFORMANCE COMPARISON

There are various groups working on RFID security protocols that produce numerous protocols each year. It is unreasonable to compare different protocols that use

different encryption algorithms. Therefore, our comparison focuses on those protocols using symmetric key encryption and decryption techniques, due to the same security level of the cipher. In this section, we analyze and compare our protocol with the protocols by Toiruul et al [3] and Kim et al [4].

A. Code Size

We have implemented the protocols by Toiruul et al and Kim et al as well as our security protocol using the same Nios-II multiprocessor hardware platform consisting of only a reader module and a tag module. Such a system is shown in Fig. 5.

The reader and the tag module are implemented as 32-bit RISC Nios II processors which are of economic type in processor option. The communication between reader and tag processors are achieved by PIO (Parallel Input and Output) ports. The communication between CPUs are handled using the following pseudo code:

```
Define  DELAY_CONSTANT   T
Send_Function (Value)
{
    Data_Out_Port = Value;
    Delay(T);
    Set_Data_Out_En;
    Delay(T);
    Clear_Data_Out_En;
    Clear_Data_Out_Port;
    Exit;
}
Receive_Function (Value)
{
   While(Timeout = False)
   {
      If ( Detected Rising Edge)
         Receive_Data = Data_In_Port
   }
   Set_Fail_To_Receive_Flag;
   Exit;
}
```

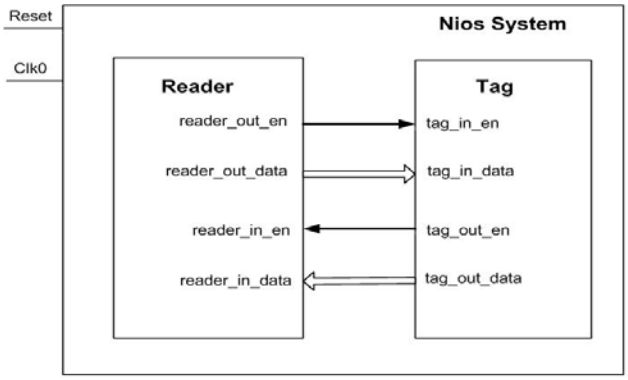

Figure 5. Nios System for Performance Evaluation

It is determined that the size of the reader code size is 3524 bytes and the tag code size is 1944 bytes. The same method is used to evaluate the code size of other two protocols. As for Toiruul et al's protocol, the tag code size is 5700 bytes, while the reader code size is 7836 bytes. For Kim et al's protocol, the tag code size is 6440 bytes, while the reader code size is 9544 bytes. The effective prototyping code size of each protocol is summarized Table VI.

B. Communication Cost

In order to evaluate the communication cost of each protocol, consider the situation where a reader needs to authenticate n number of tags.

In our proposed protocol, when a reader is to query tags, it needs to send out only one query message for all tags to respond. Thus the communication cost of step one is only one. If ***n*** tags are responding, ***n*** tags' response consist of 64**n** bits, then the communication cost in the second step is 64**n** bits. In the same way, it is concluded that the communication cost for step 3 and 4 are all 64**n** bits.

For the Batbold Toiruul et al's protocol communication cost, we evaluate the same scenario as our proposed protocol. When a reader queries n tags, it has to initiate ***n*** messages in step one. However, because its encryption algorithm AES is 128 bits in length, the communication cost of first step is 128**n** bits. In the second step, ***n*** tags send back 128**n** bits to the reader.

The Kyoungyoul Kim et al.'s protocol consists of four steps for over-the-air communication channel. When a reader queries n tags, each step has 128**n** bits in communication cost due to the use of AES encryption algorithm. The comparison of communication cost is summarized in Table VII. It is demonstrated that our scheme is more efficient than the other two schemes in terms of communication cost.

TABLE VI. CODE SIZE COMPARISON

	Our Protocol		**Toiruul et al.'s Protocol**		**Kim et al.'s Protocol**	
	Reader	**Tag**	**Reader**	**Tag**	**Reader**	**Tag**
Code Size	3523	1944	7836	5700	9544	6400

TABLE VII. COMMUNICATION COST

	Our Protocol	**Toiruul et al's Protocol**	**Kim et al's Protocol**
Communication Cost	64+64n bits	2 x 128n bits	4 x 128n bits

C. Clock Cycle

Generally speaking, the clock cycle when implementing a protocol is an important parameter because it is proportional to power consumption level if clock frequency remains unchanged. Fewer clock cycles also means higher throughput and less likelihood being attacked.

The clock cycle can be measured by implementing a timestamp timer in Nios-II SoPC (System-on-Programmable-Chip). A time-stamp timer, Timer_1 is added in the Nios-II system for the reader module in our protocol. The timer is selected as full feature option. Then we start the timestamp timer and obtain the first timestamp at the beginning of the execution. After the execution is completed in each session, we obtain the second timestamp. Then, by calculating the timestamp difference, the execution

time of each session is calculated. To ensure better accuracy, timestamp difference is measured 20 times and the average value is obtained for evaluation. As the communication between reader and tag module also takes time, we have to measure the timestamp difference with different delay constant T. The delay constant is first defined to be an initial value of 60. The timestamp difference is measured as shown in Fig. 6.

Please note that the timer frequency is 0x2faf080 in hexadecimal, which is 50MHz in decimal number. Thus, it is converted that the timestamp difference is 1.838 milliseconds. Since communication between reader processor and tag processor takes certain amount of time, we can only get the protocol execution clock cycle by calculating the ideal execution time when T approaches zero. In fact, if T is set below five in the experiment, the protocol cannot run successfully.

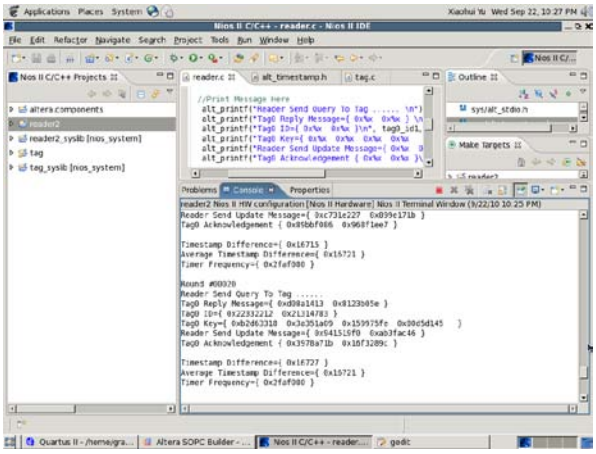

Figure 6. Timestamp Difference Measurement in our Scheme

Table VIII shows the timestamp difference in hexadecimal format obtained with different delay constant values defined in the software. The actual time difference is also computed and shown within brackets. A graph is plotted as depicted in Fig. 7 to explain the relationship between timestamp difference and delay constant T. It is a linear relationship as expected.

It is shown that as T decreases, timestamp difference also decreases linearly and the value drops 0.13ms when T is reduced by 10. We can obtain that if T approaches zero, the timestamp difference approaches 1.060ms. Thus, in an ideal condition when the communication time is zero, our scheme takes 1.060 ms to execute, which is equal to about 53000 cycles.

We apply the same method to analyze the clock cycle of Toiruul et al's protocol and Kim et al's protocol. Graphs of timestamp difference are plotted as shown in Fig. 8 and Fig. 9. It can be computed that if T approaches 0, the timestamp difference for Toiruul et al's protocol converges to 11.382ms and for Toiruul et al's protocol timestamp difference approaches 12.507ms. One session of Toiruul et al's protocol takes 569100 clock cycles to execute, while Kim et al.'s protocol takes about 625350 clock cycles.

TABLE VIII. TIMESTAMP DIFFERENCE FOR OUR SCHEME

Delay Constant T	Timestamp Difference
60	0x16721 (1.838 ms)
50	0x14dda (1.709 ms)
40	0x13482 (1.580 ms)
30	0x11b34 (1.450 ms)
20	0x101d1 (1.320 ms)
10	0xe884 (1.190 ms)

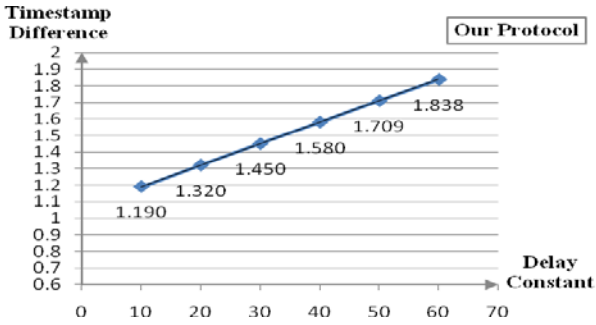

Figure 7. Timestamp Difference for our Protocol

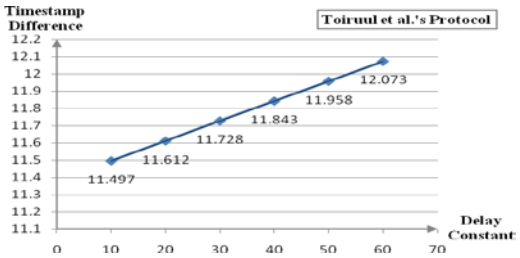

Figure 8. Timestamp Difference Toiruul et al's Protocol

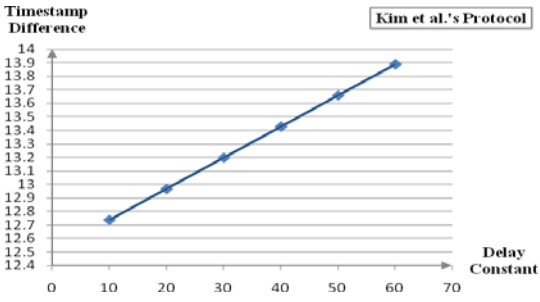

Figure 9. Timestamp Difference Kim et al's Protocol

Our protocol takes far less clock cycles than the protocols proposed by Toiruul et al and Kim et al [3, 4]. The clock cycle based time comparison of each protocol's clock cycle is summarized in Table IX. The advantage of using less clock cycle includes less power consumption, longer reading range, less risk of being attacked.

D. Scalability

We compare our scheme with Toiruul et al's protocol and Kim et al.'s protocol in terms of scalability. In our protocol, when a reader initiates a reading operation, it just needs to issue a query message. The backend server need to compare the tag reply with the list of expected replies from tags. The search time does not increase significantly with more tags added into the system. In Toiruul et al's protocol, the reader relies on other anti-collision protocol to singularize a tag out of many. Therefore, it is considered to be scalable too. In Kim et al.'s protocol, in order to initiate a reading operation, a reader needs to know the tag M secret in advance. If a reader does not know the tag's M, it has to brute force search for known M one by one to read a tag. As the population of tag grows larger, it becomes harder to identify a tag. Obviously, this scheme is not scalable. The results of scalability analysis are also summarized in Table IX.

VI. Conclusions

A novel RFID security protocol based on XTEA encryption algorithm is proposed as a robust solution for RFID security issues. In the protocol, one sub-key of the tag is dynamically updated and which sub-key is to be updated is random in nature. The reader and the tag are capable of authenticating each other mutually. The protocol model and different attack models are prototyped using FPGA. It is proved that the novel protocol is safe against major attacks. Performance is further evaluated against similar and related protocols and the result shows that our scheme is lightweight and more efficient.

As discussed above, XTEA has shown potentials in low power wireless security applications. It is helpful to create mutual authentication and trust between two entities. The future possible efficient application in wireless sensor network will be an interesting area to investigate.

TABLE IX. Timestamp Difference Prototyping and Scalability Analysis for our Proposed Protocol

	Our Proposal	**Toiruul et al.'s Protocol**	**Kim et al.'s Protocol**
Execution Time	1.060ms	11.382ms	12.507ms
Clock Cycle Number	53000	569100	625350
Scalability	Yes	Yes	No

Acknowledgment

The authors acknowledge the financial support from NSERC and Ryerson University. The authors of this work would also like to thank CMC for providing equipment support.

References

[1] A. Juels, "RFID Security and Privacy: A Research Survey" IEEE Journal on Selected Areas in Communications, Vol. 24, No. 2, pp.381-394, February 2006.

[2] Martin Feldhofer and Johannes Wolkerstorfer "Strong Crypto for RFID Tags – A Comparison of Low-Power Hardware Implementations" Institute for Applied Information Processing and Communications, Graz University of Technology Inffeldgasse 16a, 8010 Graz, 2007

[3] Batbold Toiruul and Kyung Oh Lee "An Advanced Mutual-Authentication Algorithm Using AES for RFID Systems" IJCSNS International Journal of Computer Science and Network Security, VOL.6 No.9B, September 2006

[4] Kyoungyoul Kim, Kyungho Chung, Juseok Shin, Hyunwoo Kang, Sejin Oh, Chunho Han, Kwangseon Ahn "A Lightweight RFID Authentication Protocol using Step by Step Symmetric Key Change" 8th IEEE International Conference on Dependable, Autonomic and Secure Computing, 2009

[5] Roger M. Needham and David J. Wheeler, "TEA, a Tiny Encryption Algorithm" Lecture Notes in Computer Science (Leuven, Belgium: Fast Software Encryption: 2nd Int. Workshop), Dec 16, 1994

[6] Vikram Reddy Andem, "A Cryptanalysis of the Tiny Encryption Algorithm", Department of Computer Science in the Graduate School of the University of Alabama, 2003

[7] Roger M. Needham and David J. Wheeler. "TEA Extensions" Technical report, Computer Laboratory, University of Cambridge, October 1997

[8] Jens-Peter Kaps "Chai-Tea, Cryptographic Hardware Implementations of XTEA" Volgenau School of IT&E, George Mason University, Fairfax, VA, USA

[9] T. Good, M. Benaissa, "AES on FPGA from the fastest to the smallest" In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 427–440. Springer, Heidelberg, 2005

[10] P. Chodowiec, K. Gaj, "Very compact FPGA implementation of the AES algorithm."In: Walter, C.D., Ko¸c, C¸ .K., Paar, C. (eds.) CHES 2003. LNCS, vol. 2779, pp. 319–333. Springer, Heidelberg, 2003

[11] Altera Development and Education Board URL: http://www.altera.com/education/univ/materials/boards/