PERFORMANCE ANALYSIS OF TEA, RSA, AND AES ENCRYPTION ALGORITHMS IN EMBEDDED MICROCONTROLLER SYSTEM

Ravi M K[1], Dr. Amit Kumar Jain[2]

Department of Electronics and Communication Engineering

[1,2]Himalayan University, Itanagar (Arunachal Pradesh) – India

Abstract

Embedded systems are an integral part of our society today. Each part of day to day life is some way or another subject to an embedded system. Utilities from power and water to gas and internet all depend intensely on an immense system of sensors and specific embedded systems to keep our reality running. Regardless of how basic or complex these systems might be, they are connected in an interconnected web holding up our lifestyle. An issue with any single connection can wreak destruction. Encryption is a significant apparatus for protecting against satirizing, altering, denial, information misfortune; DOS assaults, and benefit heightening. Encryption is the first line of barrier in protecting our country's embedded framework from being assaulted from inside by the systems that control its operation. The three algorithms profiled in this proposal, TEA, AES, and RSA, all fill extraordinary needs in the encryption field. TEA exceeds expectations in its speed and effortlessness and enables designers to rapidly execute encryption in the most controlled asset conditions.

1. INTRODUCTION

Encryption in embedded systems fills three essential needs. First and premier it ensures intellectual property. Besides, it secures delicate data. In conclusion, it ensures against unapproved utilize. The six threats to embedded systems, ridiculing, altering, revocation, information divulgence, denial of service and rise of benefit, are specific cases of assaults misusing at least one of the reasons for encryption. Encryption is only one device for battling these assaults, yet extraordinary in the way that it is the only device that can be utilized to battle all threats. The cross assault effectiveness of encryption takes into account the thoughts and investigations introduced in this proposal not exclusively to be connected to an extensive variety of threats yet additionally to make a structure for concentrate different devices and to describe their effectiveness against an assault. The first two motivations behind encryption are very commonplace. Intellectual property is regularly profoundly secured [1]. Take, for instance, the Coke formula put away in a profoundly secure vault in Atlanta, GA. The intellectual property of an embedded system is frequently its aggregate source code. Touchy data is regular data put away on the system, for example, encryption keys or usernames and passwords [2].

Discussion of Algorithm Types

The two main types of encryption algorithms for embedded use are symmetric and asymmetric algorithms. Each algorithm type has its advantages and disadvantages. Symmetric and asymmetric algorithm architectures are discussed below [3].

- **Symmetric Algorithms**

Symmetric algorithms go back similarly as 1900 BC, starting with substitution figures, wherein is supplanted by A + delta, et cetera. Symmetric algorithms encode and decode data utilizing a similar key or mystery. This is most commonly epitomized by performing rehashed rounds of a scientifically invertible operation, for example, XOR, include, subtract, increase or separation, after that in unscrambling, the same round is

rehashed utilizing a similar key, yet with the converse numerical operation.

- **Asymmetric Algorithms**

Asymmetric algorithms, in which encryption is included both mysteries on non-mystery divides, is new, start with the Diffie Hellman algorithm in 1976. Asymmetric algorithms scramble and unscramble data utilizing a key match. A key combine contains one private key and one public key. Asymmetric algorithms, known as public key encryption algorithms right now don't have known time-efficient answers for breaking. These algorithms depend on huge numerical calculations on prime numbers or elliptic bends, requiring improved handling power, for both encryption and decoding. Unlike symmetric algorithms, a public key can be partaken in plain content without fundamentally utilizing a safe trade.Frequently these algorithms are utilized to provide methods for secure key trade for symmetric algorithms. This expels the disadvantage of symmetric algorithms, requiring a key trade, however, holds the speed of symmetric algorithms.

- **Encryption Cost and Performance Factors**

Embedded encryption is bound by many cost vs. performance trade-offs. The costs discussed in this thesis are [4]

(1) Algorithm size (measured in ROM and RAM usage bytes),
(2) Energy consumption and
(3) Data integrity.

The performance factors discussed in this thesis are:

1. Encryption time,
2. Decryption time and
3. Brute force security.

The algorithms described in the following three components concentrate on these cost and performance metrics. These metrics provide an approach to look at the algorithms notwithstanding their extraordinary executions and complexity. Comparative metrics ought to be produced or examined by designers of encryption modules to decide the security conspires that augments performance while diminishing expenses. It is recommended that algorithm measure has been given fairly lesser weight than energy utilization for most embedded systems. This is particularly valid for embedded systems that work on battery control.

2. ANALYSIS OF THE TINY ENCRYPTION ALGORITHM (TEA)

The Tiny Encryption Algorithm (TEA) is a symmetric encryption algorithm designed for high speed and low memory usage. The algorithm makes use of a Feistel type structure for the manipulation of data blocks for encryption. TEA was proposed by Wheeler et al. in 1994 as an algorithm designed to “get security with simplicity” [5]. The algorithm relies on the functional building blocks of most microcontrollers and only uses shift, add, subtract and multiply instructions to manipulate data. These operations are practically universal and are supported across most programming languages and handled directly in hardware on most microcontrollers. The encryption routine provided by Wheeler et al. is written for the C programming language. Data is provided in v[0] and v, 32-bit word vectors. The 128-bit encryption key is provided in 32-bit words, K[0] – K. Figure 1 demonstrates the C routine used to implement the TEA algorithm for this thesis. The input n is the cycle count, used as a security scale factor. Delta, is a constant, suggest by the designers to ensure variability of the cipher text when single bit changes occur in the plain text. Finally, v is a vector representing the data to encryption and k a vector key to encrypt the data.

```
void encrypt (uint32_t* v, uint32_t * k)
{
   uint32_t y=v[0],z=v[1], sum=0,                    /* set up */
   delta=0x9e3779b9 ,                                /* a key schedule constant */
   n=32 ;                                            /* number of rounds */
   while (n-->0)
   {                                                 /* basic cycle start */
      sum += delta ;
      y += ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]) ;  /*Feistel Rounds */
      z += ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]) ;
   }                                                 /* end cycle */
v[0]=y ; v[1]=z ;
}
```

Figure 1: TEA Encryption Routine

The key, K, is 128 bits, split into four 32-bit sub key vectors K[0], K[1], K[2], K[3], where K[0] is the most significant 32 bits of K. The plain content D, 64 bits, is separated into two equivalent parts, D[0] and D[1], where D[0] is the most significant 32 bits of D. The keys and data split at word limits and are recombined as bitwise links. Every half, like this, is then used to seed the other half for the Feistel round.

This figure is regardless of clock rate, resulting in a predicable delta between encryption and decryption cycle times. This allows the necessary buffers between encryption and decryption modules to remain fixed size, void decrypt (uint32_t* v, uint32_t * k) { uint32_t y=v[0],z=v[1], sum=0, /* set up */ delta=0x9e3779b9 , /* a key schedule constant */ n=32 ; sum = delta * 32; while (n-->0) { /* basic cycle start */ sum -= delta ; y -= ((z<>5)+k[3]) ; z -= ((y<>5)+k[1]) ; } /* end cycle */ v[0]=y ; v[1]=z ; } 46 regardless of clock speed. An advantage of this fact is that the clock rate can be adjusted to meet low power needs, without modifying buffer size.

Performance			Cost		
Clock (Mhz)	Encrypt Time(s)	Decrypt Time(s)	Avg. I (mA)	Power(w)	Energy (J)
24	0.01864	0.01812	0.013387	0.0441771	0.00082346
12	0.03743	0.03681	0.00729	0.024057	0.00090045
8	0.05665	0.05608	0.005264	0.0173712	0.00098408
6	0.07473	0.07404	0.004248	0.0140184	0.0010476
4	0.11206	0.11163	0.003312	0.0109296	0.00122477

Table 1: Performance Metrics Gathered on the TEA Algorithm for A Data Size 1024 Bytes on A Cypress CY858LP

3. ANALYSIS OF THE ADVANCED ENCRYPTION STANDARD (AES) ALGORITHM

AES works in rounds, like a customary square figure 2. For AES-128, ten rounds are performed on a 128-piece square. All rounds are played out the same, except the last round. This applies to the last round for all keys sizes [6]. The info data piece D is known as the information state cluster. The state cluster has the type of a 4x4 network, as appeared in Figure 4.6. The info D is isolated into 16 single bytes. Every section is known as a word.

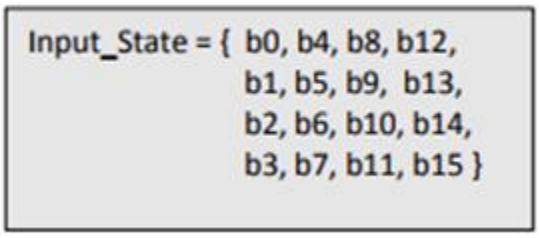

Figure 2: AES State Array

The key extension algorithm is a designed to ensure that solitary piece changes in the info key create a key calendar development those outcomes in another round key for more than one round. The key extension for a 128-piece key creates a 4x44=176 byte round key. AES utilizes a round constant characterized as a 32-bit word. This current word's lower three bytes are constantly zero and the upper byte is 2,where I is the round number. This is normally a different capacity signified as "rcon(i)". This constant performs an indistinguishable part from the delta constant in TEA and keeps back to back rounds from creating a similar figure value. The key extension routine is best actualized as multiple smaller capacities to ensure legitimate operation.

Performance			Cost		
Clock (Mhz)	Encrypt Time(s)	Decrypt Time(s)	Avg. I (mA)	Power(w)	Energy (J)
24	0.036782	0.023939	0.012731	0.0420123	0.0015453
12	0.072354	0.047718	0.007206	0.0237798	0.00172056
8	0.109221	0.074144	0.005306	0.0175098	0.00191244
6	0.145631	0.095262	0.004355	0.0143715	0.00209294
4	0.219004	0.142983	0.003404	0.0112332	0.00246012

Table 2: Performance Metrics Gathered On the AES Algorithm for a Data Set Size 1024 Bytes

4. ANALYSIS OF THE RSA ENCRYPTION ALGORITHM

This examination proposes that 2048-piece keys, and more prominent ought to be utilized to provide security for timespans of over 20 years. The security and prime factorization issue are outside the extent of this exploration, and a designer can sensibly put stock in distributed reports of RSA's security. In the survey, the private and public values required for key generation, encryption and decoding are compressed [7]. Public values are those required for encryption, yet that can be shared uninhibitedly enabling anybody to encode a message. Private values are those required for decoding and should be kept mystery by the unscrambling party.

RSA Cryptosystem		
Integer	Public	Private
P and Q		▲
N (modulus)	▲	
E (public key exponent)	▲	
D (private key exponent)		▲
Φ		▲

Table 3: RSA Cryptosystem Integer Summary

The transmitter utilizes the public key, public (n,e) to encode the data. The data P, plain content, is first changed to a message m, with the accompanying limitation: 0 ≤ m<n and gcd(m, n) = 1. This change happens with regards to a pre-characterized cushioning plan. The cushioning plan is utilized to keep the m value from being numerically basic and acquaints haphazardness with the encryption message, guaranteeing encryptions create semantically secure outcomes. The cushioning plan isn't viewed as a safe component of the algorithm, as the two parties must know the cushioning plan. It is guessed that extra security can be accomplished with a restricted cushioning plan, in which the converse of the cushioning plan can be kept mystery. Following change of the plaintext P, into number m, the figure C is registered as: C(m) = $memod(n)$.

These outcomes are compressed in Table 4.5. The encryption time patterns demonstrate that RSA has data rates five times slower than those of faster symmetric algorithms. The contrast amongst encryption and decoding module time necessities exhibits a pattern like that of TEA in which the encryption time – unscrambling time delta is a constant. The performance of key generation can't be disregarded all together. Key check time is non-paltry, as talked about more prominent than 30 seconds on the hardware utilized for this exploration without optimization. With optimization, sub-second generations can be accomplished. These outcomes were not tried and are left for promote analysis [8].

Performance			Cost		
Clock (Mhz)	Encrypt Time(s)	Decrypt Time(s)	Average Current (mA)	Power(w)	Energy (J)
24	0.10091	0.08654	0.012416	0.040973	0.004135
12	0.20217	0.18764	0.007027	0.023189	0.004688
8	0.30281	0.28818	0.005183	0.017104	0.005179
6	0.40394	0.38921	0.004263	0.014068	0.005683
4	0.60533	0.52021	0.003454	0.011398	0.0069

Table 4: Performance Metrics Gathered On the RSA Algorithm for a Data Set Size 1024 Bytes

5. COMPARISON OF TEA, RSA, AND AES ENCRYPTION ALGORITHMS

- **Security**

Level of security is the most important performance normal for any algorithm, yet is frequently the hardest to quantify. The two algorithms compose symmetric and asymmetric, fill distinctive needs in encryption. High data rates are best served by symmetric algorithms, while extreme security is best served by asymmetric algorithms. Looking at their security, in the most oversimplified terms, prompts a conclusion that symmetric algorithms are more secure and asymmetric algorithm is less secure. In reality, designers must realize that the assets expected to execute asymmetric algorithms, for example, RSA, are not realistic in each circumstance.

- **Encryption and Decryption Time**

Comparing TEA, AES, and RSA, the results demonstrated similar performance for the two symmetric algorithms TEA and AES. In all cases, RSA took significantly longer to transform the data. Encryption for all algorithms took slightly longer that the decryption routines. To generate the plots in Figures 4.8 and 4.9, below, the clock speed was varied from 4 – 24 Mhz and the time to encrypt 1024 bytes was measured. The difference in speed was most apparent at lower clock speeds. If higher clock speeds can be support by architecture and power requirements the higher security of AES and RSA can be easily achieved with minimal delays

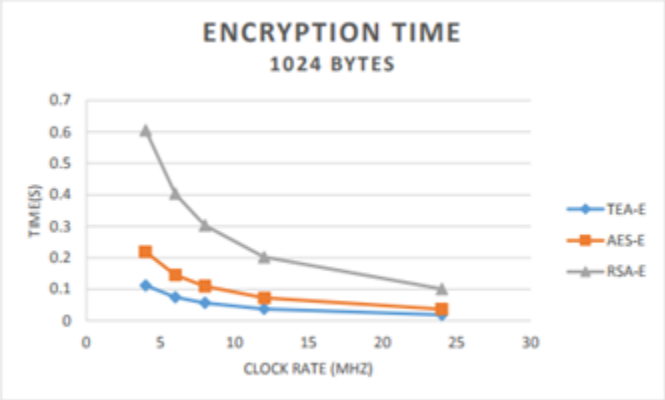

Figure 3: Comparison of Encryption Time

To enhance the security of TEA or AES, RSA can be used to transmit private encryption keys. Following a secure key exchange the faster symmetric algorithms can be used for transferring large data sets efficiently. In real world scenarios, the increased encryption time of RSA might not justify the security level increase over AES. One might be better served to increase the key size of AES to achieve the same level of security.

- **Power Consumption**

Power analysis of each of the three algorithms shows an advantage to running at higher clock speeds for battery worked device. Power utilization is higher, expanded speed diminishes encryption/unscrambling time, bringing about general less energy utilized. The utilization of processor rest and low power states can expand and realize these increases. Low power and rest states ought to be utilized whenever hinder or time-driven operation is possible. The power utilization and energy insights are for the encryption schedules only and are only estimated for the duration of encryption. TEA and RSA additionally don't require extensive stack portions amid operation since they don't require any key developments. While AES modules could be side loaded from flash memory when required, applications requiring AES ought to know about the extensive measure of brief space required for the key development. It ought to be noticed that a

joined encode and decode routine for AES was utilized as a part of this work to permit code sharing and decrease memory utilization.

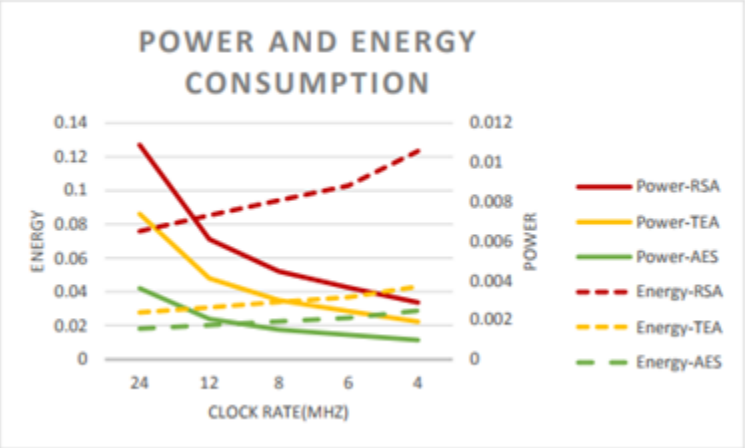

Figure 4.10: power vs. Energy consumption

	Encrypt Routine		Decrypt Routine	
	ROM	Stack	ROM	Stack
TEA	172	29	178	29
AES	856	244	856	244
RSA	30	16	26	16

6. CONCLUSION

It is concluded that AES provides military-level security to designers, yet legitimate execution requires great research of algorithm operation and usage is very complex. RSA, notwithstanding a complex key generation process, provides the most elevated amount of security, requiring no private key trade to happen. RSA's slow speed keeps it from being utilized for time-touchy applications, however its powerful security makes RSA perfect for handshake exchanges before data is transmitted utilizing a technique, for example, AES. The way in which these portrayal and modeling structures have been introduced is expected to be a system for any security arrangement. The system is an establishment, talked about with the expectation that it can be adjusted to fit any security technique in the embedded field.

REFERENCES

[1] Datta S, Stojmenovic I, Jie Wu, "Internal Node and Shortcut Based Routing with Guaranteed Delivery in Security Microcontroller system", International Conference on Distributed Computing Systems Workshop, pp.461-466, 2001.

[2] Shanti Chilukuri et al (2009), "DGRAM: A Delay Guaranteed Routing and MAC Protocol for Wireless Sensor Networks", available at: http://dspace.library.iitb.ac.in/xmlui/b

itstream/handle/10054/1317/DGRAM:%20a%20delay%20guaranteed%20.pdf?sequence=3

[3] Yanwei Wu, Xiang-Yang Li, YunHao Liu, and Wei Lou, —Energy-Efficient Wake-Up Scheduling for Data Collection and Aggregation IEEE Transaction on Parallel and Distributed System, Vol. 21, No. 2, pp. 275-287, February 2010.

[4] Hong-Yen Yang, Chia-Hung Lin, and Ming-Jer Tsai, "Distributed Algorithm for Efficient Construction and Maintenance of Connected k-Hop Dominating Sets in Microcontroller Ad Hoc microcontroller system", IEEE Transactions on Microcontroller Computing, vol.7, vo.4, pp.444-457, 2008.

[5] Yuanzhu Peter Chen, Arthur Liestman L, "Maintaining Weakly Connected Dominating Sets for Clustering Ad Hoc microcontroller system", Journal of Ad Hoc microcontroller system, vol.3, pp.629-642, 2005.

[6] A.D. Amis, R. Prakash, T.H.P Vuong and D.T. Huynh, "Max-Min DCluster Formation in Wireless Ad Hoc Networks," In proceedings of IEEE Conference on Computer Communications (INFOCOM), vol. 1, pp. 32-41, 2000.

[7] UlagKozatt .C, George Kondylis , Bo Ryu, Mahesh K. Marina, "Virtual Dynamic Backbone for Microcontroller Ad Hoc Microcontroller system", IEEE International Conference on Communications (ICC), pp.250-255,2001

[8] Heinzelman W.B., A.P.Chandrakasan, H. Balakrishnan "App.lication specific protocol architecture for security microembedded microcontroller system" IEEE Trans. On Security Networking 2002.