Securing IoT: Hardware vs Software

Rafael Azuaje
Computer Information Systems, University of the Incarnate Word, San Antonio, Texas, USA
Email: razuaje@uiwtx.edu

***Abstract*—Securing the Internet of Things (IoT): The IoT remains a relative unknown in academia despite its burgeoning financial success in the private industry highlights academia's well-known and problematic struggle to maintain awareness of the private sector and its activities and demands. This research concluded that considering the many features offered by hardware based security implemented by microcontrollers such as PIC24F family, PIC18F 'K42', Arduino Mega2560R3, ARM 7/ARM 9 and others, they do not outweigh the implementation of software based security implemented in System on a Chip (SoC) boards such as the Raspberry Pi Zero ($5.00 US), Banana Pi ($9.00 US), C.H.I.P. ($9.00 US), Onion Omega2 ($7.00 US).**

***Index Terms*—IoT, security, embedded security, Curve25519, (Deffie-Hellman), ECC**

I. Introduction

A global management consulting firm, McKinsey & Company, estimates the total value potential of IoT will reach $4-11 trillion annually by 2025. The lion's share of value will lie in factories where operations and equipment optimization could yield annual savings of $1.23.7 trillion. The IHS Markit forecasts that the IoT market will grow from an installed base of 15.4B devices in 2015 to 30.7B device in 2020 and 75.4B in 2025. The IoT is not a new technology, in and of itself. What makes IoT distinct is a technology that has been available for many decades, however, in the past 20 years it has gained unparalleled popularity in most societies. This technology improves our standard of living but at the same time: it could compromise our security if it is not properly secured. This research outlines the implementation of security in IoT devices by using hardware based, software based or a combination of both approaches. Also, the limitation of the above-mentioned approaches.

ZDNet has reported that in 2017 8.38 billion devices are currently deployed [1]. According to Gartner's Group, this trend will continue (see Fig. 1). Using the trend line equation for the consumer to forecast the IoT related to consumer consumption in the year 2021, we have: Y=2362.3 X+3030.2, substituting X for 5 (year 2021), this yields Y=14,841.00 billion devices. This trend is great for the IT industry, but also presents a great risk: security. Many IoT makers have tried to maximize profits by making their devices less secure, in many cases such devices do not have the proper security updates or fixed bugs that were not detected during the quality control process.

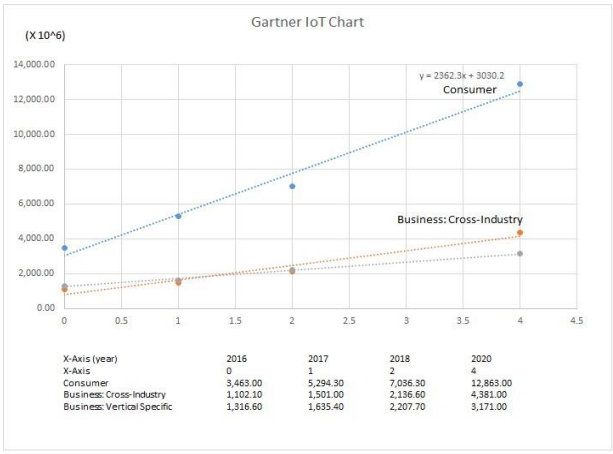


Figure 1. Gartner's IoT 2017

II. IOT Security: Encryption

Implementing software-based security in IoT devices is more difficult than other platforms such as desktops, laptop, tablets, and phones. The difficulties are driven by the constraints of this technology, that is speed, power consumption, cost, and excessive demand worldwide. Cost is one of the main driving force in the design, marketing and deployment of IoT devices. These devices need to be in the marketplace at a fast rate, and normally do not follow Moore's Law. Several schemes of encryption systems are available to designers to implement in IoT devices, the ones covered in this research are Curve25519 (Elliptical Curve Encryption (ECC)/DeffieHellman) and Rivest-Shamir-Adelman (RSA). Both encryption systems have advantages and disadvantages, the RSA is very popular, however, the ECC has a smaller memory profile and it is faster to execute.

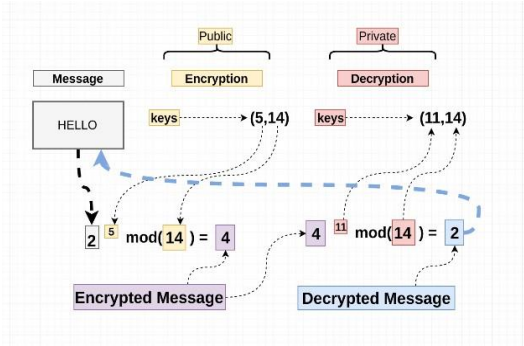


Figure 2. Basic RSA principle

Manuscript received May 6, 2018; revised August 16, 2018.

doi: 10.12720/jait.9.3.79-83

RSA Encryption

Currently, one of the most popular encryption systems is based on the RSA encryption algorithm, used to securely transmit messages over the internet. This system uses two keys, one is public and the other is private. Fig. 2 illustrates in a simple mode the basics of the algorithm [2].

Since RSA is based on using a public and private keys, it is important to have an understating on how to create such keys. This examples illustrates the process [3].

Generating Public Key (n,e):

1.- Select two prime numbers, for example **P**=53 and **Q**=59

2.- **n**=**P***Q=3127

3.- Pick **e** to be a small number (integer and not a factor of n), 1<**e**< Φ(n), lets choose **e**=3. 4.- Public Key = (3127, 3)

To calculate Φ(n):

Φ(n)= [(P-1) * (Q-1)] = (53-1) * (59-1) = 3016

Generating Private Key (d):

d=[(k * Φ(n) +1) /**e**]

For some integer k, choosing a value of 2 for k, **d**=2011

Encrypting data:

Let's assume that we need to encrypt "hi"

1.- Convert letters to numbers based on the position occupied in the alphabet: h=8 and i=9

2.- The encrypted data c= (converted data^{e}) mod n = 89^{3} mod 3127 = 1394

Decrypting data:

Given the encrypted data: 1394

1.- Data = c^{d} mod n = 1394^{2011} mod 3127 =89 Converting 89 to their respective letters, we end up with the original data: "hi".

The main strength of the RSA scheme is the use of large factoring prime numbers. It is based on the principle that it is easy to multiply large prime numbers, but factoring large prime numbers is very difficult. For example, it is easy to check that 9161 multiplied by 8009 yields 73,370,449, but trying to find the prime factors of 73,370,449 is a much longer process [4].

Implementing RSA at 2048 bits on IoT devices with 32-bit processors is not the fastest approach. In order to multiply two 2048 bit numbers requires a 64 word by 64 word long multiply operation. When one considers a 64-bit processor, it will require a 4096 long multiply with 64-bit resultant and accumulate [5]. Therefore, it is best to consider a 64-bit approach instead of a 32-bit.

Source Code for RSA Encryption/Decryption Visual Studio 2017: C# RSA program:

This program was originally posted in itechtuts.com, I do not claim ownership, but humbly pay tribute to such elegant design.

```
using System; using
System.Collections.Generic; using
System.ComponentModel; using
System.Data; using System.Drawing;
using System.Text; using
System.Windows.Forms; using
System.Security.Cryptography;
namespace RSAEncryption
{
public partial class Form1 : Form
{           public
Form1()
{
InitializeComponent();
}
private void Form1_Load(object sender, EventArgs e)
{ }
#region-----Encryptionand Decryption Function----- static
public byte[] Encryption(byte[] Data,
RSAParameters RSAKey, bool DoOAEPPadding)
{ try { byte[]
encryptedData;
using (RSACryptoServiceProvider RSA = new
RSACryptoServiceProvider())
{
RSA.ImportParameters(RSAKey);
encryptedData = RSA.Encrypt(Data,
DoOAEPPadding);
}
return encryptedData;
}
catch (CryptographicException e)
{
Console.WriteLine(e.Message);
return null;
}  }
static public byte[] Decryption(byte[] Data,
RSAParameters RSAKey, bool DoOAEPPadding)
{ try { byte[]
decryptedData;
using (RSACryptoServiceProvider RSA = new
RSACryptoServiceProvider())
{
RSA.ImportParameters(RSAKey);
decryptedData = RSA.Decrypt(Data,
DoOAEPPadding);
}
return decryptedData;
}
catch (CryptographicException e)
{
Console.WriteLine(e.ToString());
return null;
}
}
#endregion
#region--variables area
UnicodeEncoding ByteConverter = new
UnicodeEncoding();
```

```
RSACryptoServiceProvider RSA =
new RSACryptoServiceProvider();
byte[] plaintext; byte[] encryptedtext;
#endregion
#region-- Function Implemantation private void
button1_Click(object sender, EventArgs e)
{ plaintext =
ByteConverter.GetBytes(txtplain.Text
); encryptedtext =
Encryption(plaintext,
RSA.ExportParameters(false), false);
txtencrypt.Text =
ByteConverter.GetString(encryptedtext);
}
private void button2_Click(object sender,
EventArgs e) { byte[] decryptedtex =
Decryption(encryptedtext,
RSA.ExportParameters(true), false);
txtdecrypt.Text =
ByteConverter.GetString(decryptedtex);
}
#endregion
}
}
```

Elliptical Curve Encryption (ECC)

Unlike RSA, which uses prime factors to determine the value of the public key (see RSA Encryption above). ECC, uses the equation of an ellipse (y2 = x3 + ax + b) to generate its key [6]. The main benefit of using ECC compared to RSA is that it provides a high level of security using a shorter key. Shorter keys use less memory, which consequently decrease the computational requirements [7]. Fig. 3 shows the graph of the equation of an ellipse.

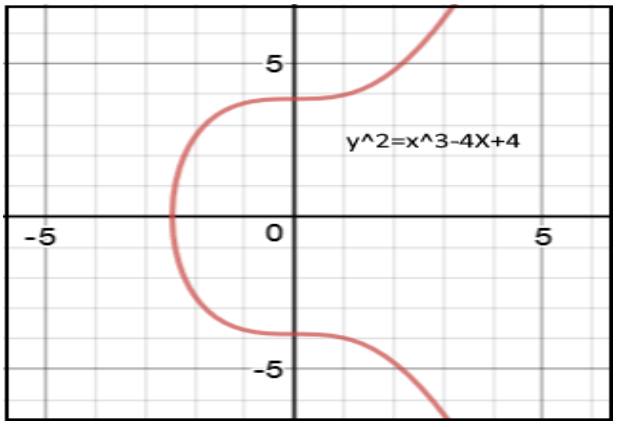


Figure 3. Elliptical curve for Y2=X3-4x+4

Basic Implementation of ECC Encryption/Decryption

The sender must first encode any message m as a point on the elliptic curve (y2 = x3 + ax + b) Pm, important to note that the ciphertext is a pair of points on the elliptic curve. The sender masks the message using random k, but also sends along a key allowing the receiver who knows the private key to recover k and hence the message. For an attacker to recover the message, the attacker would have to compute k given G and kG, which is hard to calculate [8]. Steps to implement ECC:

1.- First encode any message m as a point on the elliptic curve (Pm)
2.- Select suitable curve & point G as in D-H
3.- A & B select private keys nA<n, nB<n
4.- Compute public keys: PA= nAG, PB= nBG 5.- A encrypts Pm : Cm={kG, Pm + kPB}, k: random positive integer
PB: B's public key
6.- B decrypts Cm compute:
Pm + kPB – nB(kG) = Pm + k(nBG) – nB(kG) = Pm

Example:
p = 751; Eq(-1, 188) → 2^3 = −x+188; and G = (0, 376).
A → B: Pm = (562, 201) A selects the random number k = 386.
B's Public key: PB = (201, 5). A computes: 386(0, 376) = (676, 558) and (562, 201) + 386(201, 5) = (385, 328).
A sends the cipher text: {(676, 558), (385, 328)}.

Source Code for ECC Encryption/Decryption

This source code is based on standard C, developed by Geeksforgeeks.org. The author does not claim ownership. It is used to illustrate the basic algorithms ECC encryption.

```
//Source: https://ide.geeksforgeeks.org/index.php
/* This program calculates the Key for two persons
using the Diffie-Hellman Key exchange algorithm
*/
#include<stdio.h>
#include<math.h>
// Function to return value of x1 ^ y1 mod P long long int
power(long long int x1, long long int y1,long long int P)
{
        if (y1 == 1)
        return x1;
        else
        return (((long long int)pow(x1, y1)) % P);
}
int main()
{
        long long int P, G, x, a, y, b, ka, kb;
// Both persons will be agreed upon the
        // public keys G and P
P = 97; // A prime number P is taken
printf("The value of P : %lld\n", P);
G = 9; // A primitve root for P, G is taken
printf("The value of G : %lld\n\n", G);
// Alice will choose the private key a
a = 4; // a is the chosen private key
printf("The private key a for Alice : %lld\n", a);
x = power(G, a, P); // gets the generated key

// Bob will choose the private key b
        b = 3; // b is the chosen private key
printf("The private key b for Bob : %lld\n\n", b);
```

```
y = power(G, b, P); // gets the generated key
// Generating the secret key after the exchange
        // of keys
        ka = power(y, a, P); // Secret key for Alice
        kb = power(x, b, P); // Secret key for Bob

printf("Secret key for the Alice is : %lld\n", ka);
        printf("Secret Key for the Bob is : %lld\n",
kb);
        return 0;
}
```

III. IOT Security: Hardware Based

The hardware-based approach allows designers to produce high-quality IoT devices in a timely manner on the market. This approach offers built-in resources such as optimized Cryptographic Engine, code protection (prevents reverse engineering), and error-free security models, since, the Cryptographic Engine uses its own built-in memory, this consequently allows designers to use more main memory to incorporate additional features. The main drawback of hardware-based encryption is the increase of cost compared to the software-based approach. Also, since the Cryptographic Engine is hard-wired; it is very difficult to update when new cryptographic security models emerge or are revised.

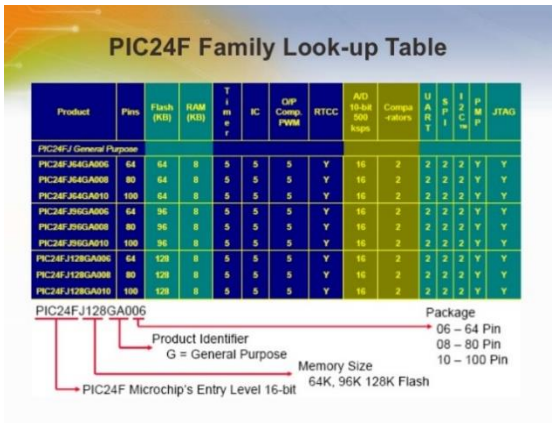
The choices available in microcontrollers and Systems On a Chip having a cryptography engine are limited compared to the rest of such systems. These Systems On a Chip with built-in cryptography engines can save considerable effort, time and money. Microchip has integrated several security features into the PIC24F GB2 family of microcontrollers to protect embedded data [9]. The main features of this Cryptographic Engine are:

*AES Engine with 128,192 or 256-Bit Key
*Supports ECB, CBC, OFB, CTR and CFB128 modes
*DES/Triple DES (TDES) Engine: Supports 2-Key and 3-Key EDE or DED TDS
*Supports up to Three Unique Keys for TDES
*True Random Number Generator
*Pseudorandom Number Generator
*Non-Readable, On-Chip, OTP Key Storages

Table I display the main characteristics of the PIC24F family of microcontrollers [10].

The fully featured hardware crypto engine supports the AES, DES and 3DES standards to reduce software overheads, lower power consumption and enable faster throughput. A Random Number Generator is also implemented which can be used to create random keys for data encryption, decryption, and authentication. The One-Time-Programmable (OTP) key storage prevents the encryption key from being read or overwritten. This microcontroller has many features highly suitable for embedded encryption but its cost ($3.95 each) may not be suitable in some cases [11]. Other microcontrollers with built-in cryptography are PIC18F ‘K42’, Arduino Mega2560R3, ARM 7/ARM 9.

TABLE I. PIC24F Family



IV. IOT Security: Software Based

The software-based approach allows companies to reduce cost in hardware, develop encryption systems in less time, use extensive open source libraries, and incorporate security features that could be easily updated if the need arises. Also, the ability to implement software-based solution in ready-to-run small boards such as the Raspberry Pi Zero ($5.00), Banana Pi ($9.00) and Onion Omega 2 ($9.00) is very appealing to developers who are familiar with Linux operating system and programming languages such as C, C++, PERL, ERLANG, Python and other programming languages.

The software-based approach makes easier to update locally or remotely, does not create hardware dependency (most cases) and is able to be used in multiple platforms. Table II, displays a comparison between these small boards.

TABLE II. Comparison of Small Computer Boards

Raspberry Pi Zero $5 US	Banana Pi $9 US	Onion Omega 2 $ 7.00 US	C.H.I.P $ 9.00 US
1GHz, Single-core CPU	A20 ARM Cortex™-A7 Dual-Core	CPU MIPS 32 24Kc	ARM 7 1GHz
512MB RAM	GPUARM Mali400MP2Complies with OpenGL ES 2.0/1.1	Clock 580 MHz	512MB RAM
Onboard Storage SD (Max. 64GB)	Memory (SDRAM) 1GB DDR3 (shared with GPU)	RAM 64 MB	4 GB Built-in
Mini HDMI and USB On-The-Go ports	Onboard Storage SD (Max. 64GB) / MMC card slot UP to 2T on 2.5 SATA disk	Storage 16 MB	No HDMI
Micro USB power	Onboard Network 10/100/1000 Ethernet RJ45,optional WIFI	2 USB 2.0	2 USB 2.0
HAT-compatible 40-pin header	Video Input,CSI input connector allows for the connection of a designed camera module	SD Slot 64 GB Max.	WiFi 802.11b/g/n
Composite video and reset headers	Video Outputs: HDMI, CVBS , LVDS/RGB	WiFi b/g/n	Bluetooh4.0
CSI camera connector	Audio Output 3.5 mm Jack and HDMI	GPIO 15	GPIO 80
	Power Source 5 volt via MicroUSB(DC In Only) and/or MicroUSB (OTG)	PWM 2	
	2 X USB 2.0 Ports	UART 2	
		I2C 1	
		SPI 1	
		I2S 1	

V. Conclusion

The conclusion of this research is that considering the many features offered by hardware based security implemented by microcontrollers such as PIC24F family, PIC18F 'K42', Arduino Mega2560R3, ARM 7/ARM 9 and others, they do not outweigh the implementation of software based security implemented in System on a Chip (SoC) boards such as the Raspberry Pi Zero ($5.00 US), Banana Pi ($9.00 US), C.H.I.P. ($9.00 US), Onion Omega2 ($7.00 US) [12]. These systems in general are faster and provide more resources such as memory, Input and Output interfaces, open source: operating system (Linux), development tools, libraries.

However, either software base or hardware-based cryptography may have a short life once quantum computing becomes readily available. More research is necessary to address future cryptography schemes capable of being immune to quantum computing [13].

References

[1] L. Tung. (2017). IoT devices will outnumber the world's population this year for the first time. [Online]. Available: http://www.zdnet.com/article/iot-deviceswill-outnumber-the-worlds-population-this-yearfor-the-first-time/

[2] J. Garcia. [Online]. Available: https://hackernoon.com/how-does-rsa-workf44918df914b

[3] RSA Algorithm in Cryptography. [Online]. Available: http://www.geeksforgeeks.org/rsaalgorithm-cryptography/

[4] SIlicon Labs. (2016). IoT Security Part 7: Key Exchange using Elliptical Curve Cryptography. [Online]. Available: https://www.silabs.com/community/blog.entry.html/2016/05/26/iot_security_part7-UeMh

[5] Cancallme. (2014). Implement RSA Algorithm in C#. [Online]. Available: http://itechtuts.com/implement-rsa-algorithm-c/

[6] D. Gloag, Elliptic Curve Cryptography (ECC): Encryption & Example. [Online]. Available: https://study.com/academy/lesson/elliptic-curvecryptography-ecc-encryption-example.html

[7] J. Chu. (2016). The beginning of the end for encryption schemes? [Online]. Available: http://news.mit.edu/2016/quantumcomputer-end-encryption-schemes-0303

[8] Cryptography. [Online]. Available: http://ae.hc.cust.edu.tw/new_website/attachments/article/244/Lecture%2015_elliptic%20curves%20II.pdf

[9] PIC24FJ128GB204. [Online]. Available: https://www.microchip.com/wwwproducts/en/PI C24FJ128GB204

[10] (2011). Introduction to the 16-bit PIC24F Microcontroller Family. [Online]. Available: https://www.slideshare.net/element14/introduction-to-the-16bit-pic24f-microcontroller-family

[11] microchipdirect.com. [Online]. Available: https://www.microchipdirect.com/product/PIC24FJ128GB204

[12] (2017). [Online]. Available: https://www.mouser.com/ProductDetail/Microch ip/PIC12F509-I-P/?qs=B6asEe1xA0UMsIMa1jMkYA%3D%3D&gclid=Cj0KCQiAsqLSBRCmARIsAL4Pa9T7X1tjmqOpTT99M7Fp77TQiiUDGLcFw30joQXla7zwiWfSzWSnSg0aAqU3EALw_wcB

[13] J. Temperton, (2017). d-wave-2000qquantum-computer. [Online]. Available: http://www.wired.co.uk/article/dwave-2000q-quantum-computer

Rafael Azuaje was born in Valera, Venezuela, in 1957, currently an US citizen. He received the B.S. degree in Computer Science from the Universidad Politecnica de Madrid, Madrid, Spain in 1979 and the M.S. degree in Computer Information Systems from the Saint Mary's University in San Antonio, Texas and a PhD in Management of Information Systems from Northcentral University at Prescott, AZ in 2004. His research interests include Artificial Intelligence, Parallel Programming, Cyber Security and Quantum Computing.