# Hardware - Software Implementation of Public-Key Cryptography for Wireless Sensor Networks

**Gerard Murphy[1] Aidan Keeshan[2] Rachit Agarwal[3] Emanuel Popovici[4]**

*Department of Microelectronic Engineering,*
*University College Cork,*
*Cork,*
*Ireland*

*E-mail:* [1]*g.d.murphy@student.ucc.ie* [2]*a.keeshan@student.ucc.ie*
[3]*rachit.agarwal@ue.ucc.ie* [4]*e.popovici@ucc.ie*

*Abstract* - **Security in wireless sensor networks is currently provided through symmetric key cryptography. Although the low computational complexity involved in private key algorithms is advantageous, session keys must be embedded in the sensor nodes before the nodes can be deployed. Protocols are also necessary to ensure synchronization of keys between the devices on a network. These protocols require significant communication and storage overhead. The limitation of such a cryptosystem is that it is not possible to guarantee the confidentiality of the session keys.**

**It is commonly perceived that public key algorithms are slow, consume a lot of power and require a significant amount of architectural overhead. In this paper we show that it is possible to implement public key algorithms on resource constrained sensor node platforms. Using a hardware/software codesign approach, we have successfully mapped a public key cryptosystem based on Rabin's scheme onto the motes developed by Tyndall National Institute. Our implementation focuses on efficient architectures that execute the public key algorithms using minimal resources.**

*Keywords* – **Rabin's Scheme, public-key cryptography, wireless sensor networks**

## I INTRODUCTION

Interest in wireless sensor networks has increased dramatically in recent years due to the vast range of possible applications for the technology. It is expected that wireless sensors will soon be incorporated into medical, personal and military applications on a large scale. Distributed embedded systems that can communicate wirelessly have the potential to behave far more intelligently than isolated nodes. Due to the potentially sensitive nature of communications between sensor nodes, the confidentiality and the integrity of data must be assured. Both can be achieved by encrypting the communications.

Security in wireless sensor networks is currently provided through private key cryptography. Private-key cryptosystems need complicated key-scheduling algorithms to establish symmetric keys. In private key cryptography, parties which intend to communicate must agree on a secret symmetric key. This is currently achieved by embedding session keys within the sensor nodes before they are deployed. In this scenario, the capture of a single node by an adversary may compromise the security of the entire network.

It is widely believed that the complexity cost of public-key algorithms make them unsuitable for resource constrained applications. In this paper we propose a public-key cryptographic architecture based on Rabin's scheme [1] that provides efficient encryption of data. Using public-key cryptography we can eliminate the necessity for predetermined session keys.

In communications, the original message before encryption is referred to as the plaintext ($P$) and is usually of a fixed length. In public-key cryptography an entity generates a pseudo-random private key and

a corresponding public-key. The public-key is not secret and can be used by anyone to encrypt communications intended for the entity associated with that public key. After encryption the resulting ciphertext (*C*) can be sent over an insecure channel, but only the entity that generated the keys can decrypt the ciphertext. The private key is used to decrypt the communications and is known only by the entity that generated the keys. This process is shown in Fig. 1.

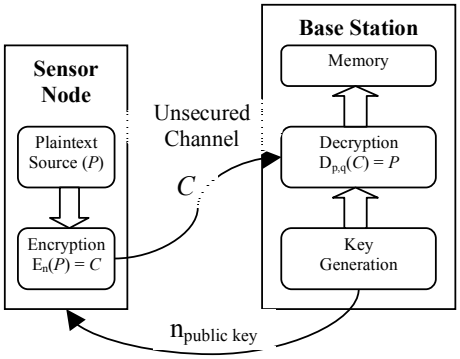


**Fig. 1**: Public Key Cryptography in Sensor Networks

It is usual in sensor network applications that the sensor nodes need only encrypt communications and not decipher them. Our implementation exploits this detail by utilizing a public-key scheme which has a very efficient encryption algorithm. It is possible to incorporate our public key architecture into a hybrid cryptosystem for applications were decryption on the nodes is necessary. The public key encryption algorithm can be used to establish symmetric session keys, which can then be used with a private key cryptographic algorithm to encrypt and decrypt communications.

## II RABIN'S SCHEME

The security of Rabin's scheme is inherent in the difficulty of factorizing large numbers and is therefore similar to the security of RSA with a modulus of the same size. The size of the modulus determines the security of the cipher. RSA is the standard encryption algorithm used in public key cryptographic applications. The disadvantage of RSA is that the algorithm necessary to implement both encryption and decryption is computational intensive. Rabin's scheme has asymmetric computational costs. It is only necessary to perform a simple modulo squaring operation to encrypt a message. Therefore encryption can be preformed relatively efficiently, while the computational cost of the decryption algorithm is comparable to RSA. This asymmetrical property of Rabin's scheme means it is of significant practical importance in applications such as wireless sensor networks where encryption must be done on very low power, often remote, nodes.

We present next the encryption and decryption steps as they appear in [2].

*Key generation for Rabin's Scheme*

The base station generates a private key pair and a corresponding public key.

1. Choose two large strong random prime numbers, p and q, both congruent to 3 mod 4.
   $p, q \equiv 3 \bmod 4$
2. Compute the product $n = p \cdot q$
3. The base station's public key is n. The private key is the pair (p,q)

*Rabin's public-key encryption*

To encrypt a message for the base station

1. Obtain the base station's public key n.
2. Represent the message (*M*) as a binary integer, less than the length of n
3. Compute $C = M^2 \bmod n$
4. Transmit the ciphertext C to base station

*Rabin's public-key decryption*

Decryption involves finding the four square roots of $C = M^2 \bmod n$. To recover the plaintext:

1. Use the extended Euclidean algorithm to find integers a and b that satisfy the equation:
   $a.p + b.q = 1$

   Note that a and b can be computed during the key generation stage
2. Compute $r = C^{(p+1)/4} \bmod p$
3. Compute $s = C^{(q+1)/4} \bmod q$
4. Compute $x = (a.p.s + b.q.r) \bmod n$
5. Compute $y = (a.p.s - b.q.r) \bmod n$
6. The four square roots of *C* modulo *n* are:

   $M1 = x$
   $M2 = -x \bmod n$
   $M3 = y$
   $M4 = -y \bmod n$

The plaintext was M1, M2, M3 or M4. The base station somehow decides which one of these was the actual message.

The drawback of Rabin's public-key scheme is that the receiver must select the correct plaintext from among four possibilities. This can be overcome in practice by adding predetermined redundancy to the original plaintext before encryption [2]. For example, a specified header can be added to the plaintext or instead the last 32 bits can be replicated. With this approach, we can determine the actual message sent with high probability. Although this does generate significant transmission overhead, it does act as a simple error detection protocol for the receiver. If an error occurs in the transmitted bit sequence the decryption process will not generate a square root with the specified redundancy.

The extra bits also have the benefit of circumventing weaknesses in certain cryptographic protocols [3]. Certain chosen-ciphertext attacks exist against Rabin's scheme where an adversary can recover some information about a plaintext. In this scenario, the adversary presents a certain ciphertext to the decryption machine. The decryption machine decrypts the ciphertext and returns some plaintext. The adversary can build up information about the private keys which can then be used to help recover information about legitimate plaintexts. The decryption machine can be implemented so that it only returns legitimate messages. If it decrypts a ciphertext that does not have the specified redundancy, the ciphertext is considered fraudulent and the message is discarded.

Rabin's scheme encryption is regarded as an extremely fast operation in terms of public-key encryptions algorithms. It involves a single modulo squaring operation while RSA encryption with e = 3 requires one modular multiplication and two modular squaring calculation. While Rabin's scheme decryption is slower than RSA decryption, the operations are comparable. In applications such as sensor networks, decryption may be done on dedicated base stations that are not be constrained by resources. Sensor nodes nominally do have limited resources. But the low computational complexity of this public-key encryption algorithm means that it can be feasibly implemented on such a platform.

## III IMPLEMENTATION

The implementation was prototyped on 25mm cube modules provided by the Tyndall National Institute [3]. These cubes consist of several different programmable modules that are interchangeable. The cubes were configured so that each node included a low power 8-bit Atmel microcontroller, a Spartan-IIE FPGA and a RF transceiver. For our implementation the different modules were synchronized by a shared 4 MHz clock. This experimentation platform was ideal for verifying the public-key architecture and testing the overall cryptosystem. An overview of the system platform is illustrated in Fig. 2.

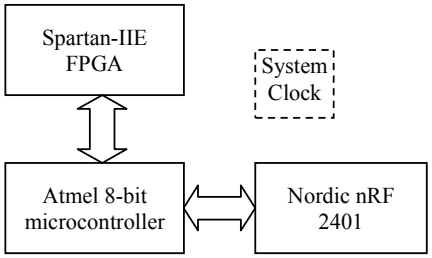


**Fig. 2**: System Platform

### a) *Encryption - Hardware*

The architecture was designed to be parametrizable in regard to the length of the modulus used. For the purpose of this paper the cryptosystem is presented with a 512-bit modulus.

Rabin's scheme encryption is based on a simple modular squaring operation. An inefficient approach to this calculation would involve squaring the 512-bit number and performing a large division operation. An optimized algorithm was devised with consideration for the timing and memory constraints of the platform. This algorithm (**Modular Squaring**) performs the calculation using repeated additions and subtractions, eliminating the need for a large multiplier.

**Modular Squaring** *modSquare(A, M)*
**Input:** Integers A, M; $0 \le A < M < 2^n$
**Output:** $R = A^2 \bmod M$ $(0 \le R < M)$

```
01. R <= 0
02. B <= A
03. for i = 0 upto n-1 do
04.    if (b_i = 1)
05.       R <= R + A;
06.    A <= A(n-1 downto 0)  & '0';
07.    while (A ≥ M  OR  R ≥ M) do
08.       if  (A ≥ M  AND  R < M) then A <=  A - M;
09.       else if  (A < M  AND  R ≥ M) then R <= R - M;
10.       else if  (A ≥ M  AND  R ≥ M) then
11.          R <= R - M;
12.          A <= A - M;
13. return R;
```

The encryption algorithm was implemented on a Spartan-IIE FPGA. The encryption operation requires the plaintext and the public key as inputs. A standard handshaking protocol is used to interface the FPGA with the microcontroller. The two 512-bit numbers are sent from the microcontroller to the FPGA via an 8-bit bus. The FPGA signals the microcontroller to read back the ciphertext when the

encryption procedure is complete. The encryption architecture is shown in Fig. 3.

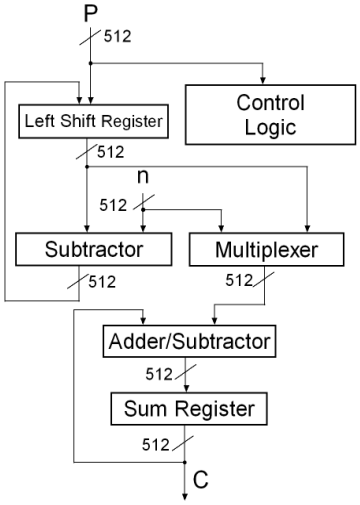


**Fig. 3:** Encryption Architecture

The Spartan-IIE has 64k-bits of RAM that is divided into sixteen 4k-bit dual-port blocks. These blocks can be configured to specific port widths. Due to the limited design space available on the FPGA it was necessary to utilize one of these RAM blocks for the encryption process. The Ram block was configured to write data via an 8-bit parallel bus. The data was consequently read serially by the control logic during the calculation process.

The main advantage of using a FPGA to prototype a hardware design is the ability to test the physical implementation. Unfortunately the power consumed by FPGA's make them unsuitable for sensor network architectures. Actual sensor nodes are commonly realized in ASIC format. Their appearance in this implementation is purely for timing and area considerations. Power consumption is not indicative of actual sensor node performance.

### b) *Encryption - Software*

It was necessary to employ a power efficient methodology in the design of the software. The advantage of using the Atmel microcontroller was its low power RISC architecture. Consideration for the specialized instruction set of Atmega 128 helped to exploit this feature. Memory usage was also minimized to avoid the latency and power consumption of memory access. The power saving conventions for this design included:

- Minimizing repetitive address computation
- Assigning the most frequently accessed parameters to registers.

Auxiliary to this convention was the optimization of all numerical calculations in order to achieve the most efficient implementation. All software was implemented using a C variant developed for embedded systems. The maximum radix that can be represented on an 8-bit microcontroller is 256. This is the same numerical radix of extended ASCII codes as represented by the char type in C. It was necessary to develop multi-precision functions to represent numbers greater than the processor's radix.

Additions, subtraction, compare, multiplication and modulo functions were required to implement Rabin's Scheme. All these functions were developed using classical algorithms [4], with the exception of the modulo function. This modulo function utilized the compare and subtraction functions in order to resolve the remainder. The full design implementation revealed the computational complexity of this modulo function. An algorithm was devised to make this function operate more efficiently. By using pre-multiplication and right-shift operations it was possible to reduce the number of subtraction per byte from a maximum of 256, down to a maximum of 8 subtractions.

### c) *Decryption – Hardware/Software Codesign*

The decryption architecture was implemented on the sensor node platform by adopting a hardware/software codesign approach. The architecture was partitioned so that the most computationally intensive processes were implemented in hardware while the microcontroller managed the execution of the algorithm and performed the final mathematical transformations.

The most demanding calculations in the decryption process are the two 256-bit modular exponential operations. The computation architecture which was implemented is based on the binary method algorithm. This algorithm (**Modular Exponentiation**) performs modular exponentiation by using repeated modular squaring and modular multiplication (**Modular Multiplication**) operations. Modular multiplication is almost identical to modular squaring. It was possible to implement the decryption hardware by reusing the encryption architecture and combining it with a more sophisticated control logic block and memory management system.

**Modular Multiplication** *modMult(A, B, M)*
**Input:** Integers A, B, M; $0 \leq A < M < 2^n$, $0 \leq B < E < 2^n$
**Output:** R = A.B mod M $(0 \leq R < M)$

```
01. R <= 0
02. for i = 0 upto n-1 do
03.     if (b_i = 1)
04.         R <= R + A;
05.     A <= A(n-1 downto 0) & '0';
06.     while (A ≥ M  OR  R ≥ M) do
07.         if  (A ≥ M  AND  R < M) then A <=  A - M;
08.         else if (A < M  AND R ≥ M) then R <= R - M;
09.         else if (A ≥ M  AND R ≥ M) then
10.             R <= R - M;
11.             A <= A - M;
12. return R;
```

**Modular Exponentiation** *modExp(A, E, M)*
**Input:** Integers A, E, M; $0 \le A < M < 2^n$, $0 \le E < 2^n$
**Output:** $R = A^E \bmod M$ $(0 \le R < M)$

```
01. R <= 1
02. for i = n-1 downto 0 do
03.     R <= modSquare(R, M)
04.     if (Ei =1)
05.         R <= modMult(R, A, M)
06. return R;
```

Extended precision addition, subtraction and modulo operations were implemented on the microcontroller to perform the final mathematical operations. The microcontroller selected the correct plaintext from among the four possible plaintexts by scanning the solutions for a pre-specified header.

By exploiting the potential of hardware/software codesign it was possible to implement the decryption architecture relatively efficiently. Encryption can also be performed by using the modular exponentiation hardware and setting the exponent to 2. However the area constraints of the Spartan-IIE FPGA meant that it was only possible to fit a 256-bit modulo exponentiation architecture onto the nodes. It is also worth noting that RSA encryption and decryption can both be performed using a single modular exponentiation operation.

## IV RESULTS

The public-key cryptosystem was successfully implemented and tested on the 25mm cube platform. Each FPGA design was synthesized and implemented using ISE8 software. The results for the encryption architecture are shown in tables 1 and 2.

| **Design** bits | **Max Freq.** MHz | **LUT** No. (%) | **Slices** No. (%) | **Ram** bits |
|---|---|---|---|---|
| 256 | 17.8 | 2401 (39%) | 1454 (47%) | 256 |
| 512 | 11.7 | 4,736 (77%) | 2,879 (93%) | 512 |

**Table 1**: Spartan-IIE FPGA Implementation of Encryption
Post Place & Route Results

| **Design** bits | **No. of clock cycles** excluding load | **Maximum Throughput** Kbits/s |
|---|---|---|
| 256 | 692 | 769 |
| 512 | 1385 | 769 |

**Table 2**: Spartan-IIE FPGA Implementation of Encryption
Timing Results

Table 1 shows results for resource usage in terms of LUT's, slices and Ram bits used. Table 2 shows the timing results of the encryption architecture clocked with a frequency of 4 MHz. The number of clock cycles needed to perform the encryption procedure does not include the time necessary to load the data into the FPGA. The maximum throughput results however do include the communication overhead necessary to interface the FPGA and the microcontroller.

The synthesis results for a 1024-bit design were also obtained. Unfortunately, it was not possible to implement this design as it required more area resources than the Spartan-IIE FPGA had available. The results for this design are shown in table 3.

| **Design** bits | **Max Freq.** MHz | **LUT** No. (%) | **Slices** No. (%) | **Ram** bits |
|---|---|---|---|---|
| 1024 | 13.3 | 8352 (135%) | 4574 (148%) | 1024 |

**Table 3**: Spartan-IIE FPGA Implementation of Encryption
Synthesis Results

Preliminary timing analysis show that encryption in software is two orders of magnitude slower than encryption in hardware.

The results for the modular exponentiation architecture that was used in the decryption process are shown in tables 4 and 5. Rabin's Scheme encryption with a 512-bit modulus requires a 256-bit modular exponentiation architecture for decryption.

| **Design** bits | **Max Freq.** MHz | **LUT** No. (%) | **Slices** No. (%) | **Ram** bits |
|---|---|---|---|---|
| 128 | 29.6 | 1674 (27%) | 903 (29%) | 256 |
| 256 | 19.8 | 3238 (52%) | 1769 (57%) | 512 |

**Table 4**: Spartan-IIE FPGA Implementation of Modular Exponentiation
Post Place & Route Results

| **Design** bits | **No. of clock cycles** excluding load | **Maximum Throughput** Kbits/s |
|---|---|---|
| 128 | 86,090 | 5.92 |
| 256 | 344,356 | 2.97 |

**Table 5**: Spartan-IIE FPGA Implementation of Modular Exponentiation
Timing Results

The high level of computational complexity involved in modular exponentiation is revealed in the results shown above. The results obtained are averages of possible inputs where the exponent is the

largest possible length i.e. the same length as the modulus.

Table 6 shows the synthesis results for a 512-bit modular exponential FPGA design.

| **Design** bits | **Max Freq.** MHz | **LUT** No. (%) | **Slices** No. (%) | **Ram** bits |
|---|---|---|---|---|
| 512 | 13.3 | 6347 (103%) | 3542 (115%) | 1024 |

**Table 6**: Spartan-IIE FPGA Implementation of Modular Exponentiation
Synthesis Results

The rest of the decryption algorithm was implemented on the microcontroller. The software partitioned decryption process involves several extended-precision multiplication and modulo operations, one addition and three subtractions. The estimated results for the software partitioned design indicate that the time necessary to perform these calculations is similar to the amount of time that is needed for the FPGA to perform the modulo exponentiation operations.

# V CONCLUSIONS

Good cryptographic implementations are essential in wireless sensor networks if the security of the communications is to be assured. As wireless sensor technologies begin to be incorporated into applications on a large scale, the secure encryption of data is becoming even more vital. Public-key cryptography permits such security and its flexibility enables the formation of ad hoc networks. Our results show that it is possible to implement public-key cryptosystems on resource constrained platforms such as wireless sensor networks.

While the hardware implementation of the encryption algorithm is much faster than the software implementation, more power is consumed due to the high power requirements of the FPGA. Software implementations of the algorithm are also realizable and have the benefit of low cost and high flexibility. However the time necessary to perform encryption and decryption is significantly increased by using a software only approach. The results show that the inherited benefit of adopting a hardware-software partitioned cryptographic system is a great advantage for efficient implementations on resource constrained sensor nodes.

# VI REFERENCES


[1] Rabin, M.O., ”Digitalized signatures and public key functions as intractable as factorization”, Mit/lcs/tr-212, Massachusetts Institute of Technology (1979)

[2] A. Menezes, P. van Oorschot, and S. Vanstone, “Handbook of Applied Cryptography”, Second Edition, CRC Press, 2001

[3] B. O'Flynn, S. Bellis, K. Delaney, J. Barton, S.C. O'Mathuna, A.M. Barroso, J. Benson, U. Roedig, C. Sreenan, “The development of a novel minaturized modular platform for wireless sensor networks”, Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on 15 April

[4] D. E. Knuth, “The Art of Computer Programming”, Volume II Seminumerical Algorithms, 1997

[5] B.Schneier, “Applied Cryptography”, Second Edition, John Wiley and Sons, Inc, 1996

[6] G. Gaubatz, J.-P. Kaps and B. Sunar, “Public Key Cryptography in Sensor Neworks – revisited”, In *1ST European Workshop on sSecurity in Ad-Hoc and Sensor Networks (ESAS 2004),* 2004.