行政院國家科學委員會補助專題研究計畫 ■ 成 果 報 告
□期中進度報告

安全無線感測網路：感測網路串流密碼系統之設計

Wireless Sensor Network: Design of Stream Ciphers in Sensor Network

計畫類別：□ 個別型計畫　■ 整合型計畫
計畫編號：NSC　94－2213－E－009－090
執行期間：　2005 年　8 月　1 日至　2006 年　7 月　31 日

計畫主持人：陳榮傑
共同主持人：
計畫參與人員：林志賢、林家瑋、林嘉軒、劉用翔、蔡佩娟

成果報告類型(依經費核定清單規定繳交)：■精簡報告　□完整報告

本成果報告包括以下應繳交之附件：
□赴國外出差或研習心得報告一份
□赴大陸地區出差或研習心得報告一份
□出席國際學術會議心得報告及發表之論文各一份
□國際合作研究計畫國外研究報告書一份

處理方式：除產學合作研究計畫、提升產業技術及人才培育研究計畫、列管計畫及下列情形者外，得立即公開查詢
□涉及專利或其他智慧財產權，□一年□二年後可公開查詢

執行單位：國立交通大學 資訊工程學系

中　華　民　國　　95 年　　10 月　　31 日

1. 中文摘要

無線微型感測網路的主要組成元件是感測器。感測器具備有偵測資訊以及無線通訊的能力，透過無線電的傳輸可以將收集到的相關資訊即時、動態的傳送。在無線微型感測器網路普及的環境下，網路安全將成為很重要的議題，安全的網路讓使用者可以盡情的享受無線微型感測安全網路的便利性、功能性、穩定性及隱密性。不同於以往的安全技術設計，因為受限於無線微型感測器上極有限的記憶體空間、計算能力與低功率需求，需要大量且複雜運算的公鑰密碼系統，例如 RSA 或橢圓曲線密碼系統，並不適合運用在無線微型感測器網路上。甚至於密鑰密碼系統中的塊狀密碼系統 (Block Cipher System)，也因為架構複雜且運算耗時，也不適合運用在無線微型感測器網路上。而串流密碼系統(Stream Cipher System) 有別於塊狀密碼系統，是種架構相當簡單且加密速度非常快速的密碼系統，非常適用於記憶體及運算資源有限的感測器上。

本計畫將針對密碼學中的串流密碼進行徹底且深入的研究，了解各種串流密碼的結構與性質，分析各種串流密碼技術應用在微型感測器上的優缺點，並且利用各種模擬檢測或已知的攻擊方法進行安全性的驗證。最後，我們將對各種設定進行細部的調整作最佳化，達到使用最少的資源而有最大的加解密速度與安全性，並加入省電機制的考量，希望在微型感測器上極有限的記憶體空間、計算能力與低功率需求的限制下，提供最佳的安全密碼技術。

關鍵詞：感測網路、串流密碼、線性反饋移位暫存器

2. Abstract

Network is one of the most important research topics in recent years. Sensor is the main component of this network architecture which collects the environment information and transmitted to the read-end data center through RF signaling and ad-hoc routing methodology to make more advanced data analysis. Security is a very important issue in wireless sensor network. Due to the limitation of memory and computation power in the sensors, public-key cryptosystems (e.g., RSA and ECC) and block cipher systems (e.g., DES) are difficult to be embedded in the sensor network. Stream cipher systems with simple structures and high-speed encryption have the advantage for severe constraints on the amount of processing power and memory space.

In this project, we will study all of stream cipher systems and give stream cipher based candidates for the security technique in wireless sensor network. To verify the security, we use the statistic testing and known attack methods to analysis our proposed techniques. Eventually, we will develop a secure and power-saving encryption mechanism that meets the hardware limitation of sensor node.

Keywords : Sensor Network, Stream Cipher, LFSR Wireless Sensor

3. Design of Stream Ciphers on Sensors

On Wireless Sensor Networks there are several severe challenges – these sensors have limited processing power, storage, bandwidth, and energy. So we must choose a fast and low-storage cryptosystem. The stream cipher is very fast and can lower power consumption. Since the stream cipher encrypts each character under a time varying function of the key, it prevents deletion, insertion or replay of ciphertext, as well as ciphertext searching. We will design the stream cipher to resist all the attacks such as correlation attacks [8], the best affine approximation attack [9], algebraic attacks [2] and so on. A stream cipher is to use the generator to produce the pseudo-random keystream. The generator has the filter generator and combination generator. We will discuss which generator is more suitable for Wireless Network Sensors. There are two main components to construct the generator: one is LFSR and the other is a Boolean function.

Boolean functions must satisfy some conditions to resist all kinds of attacks. They must be of high resilient, nonlinearity, algebraic degree, algebraic immunity and balancedness. We will use the method of [1] to construct the Boolean function we want.

Using the method of [1] we can get

$f_n = x_n (f'_{n-1} \oplus f''_{n-1}) \oplus f'_{n-1}$, $n \equiv 2 \pmod 3$ ……..……. (1)

where $f'_{n-1} = (x_{n-3} \oplus 1)f'_{n-4} \oplus x_{n-3}f''_{n-4} \oplus x_{n-2} \oplus x_{n-1}$

$f''_{n-1} = (x_{n-2} \oplus x_{n-1} \oplus 1)f'_{n-4} \oplus (x_{n-2} \oplus x_{n-1}) f''_{n-4} \oplus x_{n-3} \oplus x_{n-2}$

and $f'_1 = 0$, $f''_1 = x_1$. By the theorem in [1] the function f_n is (2n-7)/3-resilient function on V^n, $n \equiv 2 \pmod 3$, with the nonlinearity $2^{n-1} – 2^{(2n-4)/3}$ and an algebraic degree of each variable in f_n is (n+4)/3. The scheme of the function f_n contains 2n – 4 XOR and (2n - 1)/3 AND. It is linear on n. The number of XOR and AND in other functions constructed by usual methods, in general, is exponential on n. It uses less storage and operations and is faster. The Boolean function constructed by this method is (n, (2n-7)/3, (n+4)/3, $2^{n-1} – 2^{(2n-4)/3}$), that is (n, resilient, algebraic degree, nonlinearity). If we choose n to be 11, it is (11, 5, 5, 960) and from [2] we know the algebraic immunity of this Boolean function is 4. We expect these values are enough to be a secure stream cipher.

Then we talk about LFSR. On sensors, the key length is not too big, so we choose it to be 128. In the filter generator, we only need to find one primitive polynomial of degree 128. Its period is $2^{128} – 1$, and we believe it is long enough. But if we want to resist the inversion attack and the conditional correlation attack, the LFSR tapset must be FPDS [4,5]. Therefore we choose

$$c = x^{128} + x^{99} + x^{59} + x^{31} + x^9 + x^7 + 1$$

T = {7, 9, 31, 59, 99, 128} is FPDS. And the filter generator tapset Γ also need to be FPDS. Therefore, we choose Γ = {0 1 3 7 12 20 30 44 66 82 127}, and it is FPDS.

In the combination generator, we must choose 11 primitive polynomials and sum of their degree is 128, e.g. {6, 8, 9, 11, 12, 13, 14, 14, 15, 16, 17}.

3.1 Implementing the stream cipher with software

In this subsection, we will discuss how to implement the filter generator and the

combination generator and decide which one is better for Wireless Sensor Networks. The hardware specification of the sensor we use is as follows:

CPU	8051, 8-bit, 12MHz
Storage	512 bytes RAM
	16k bytes ROM
	64k flash RAM
OS	MicroC OSII
compiler	Keil C

Table 1: Characteristics of prototype sensors

We first write c program and compile it with Keil C and load the hex file Keil C produces into ROM of sensors. Because the memory of sensors is small, we expect the code size of our stream cipher is smaller.

We first choose the filter generator to implement. Let n be 11 and the Boolean function can be got from (1). It is as follows:

f = x11**{**(x8+x9+x10) **[**(x5+x6+x7)(x1(x2+ x3+x4)+x2+x4)+x5+x7**]**+x8+x10**}**
+ x8 [(x5+x6+x7) (x1(x2+x3+x4)+x2+x4)+x5+x7] +x5(x1(x2+x3+x4) + x2+x4)
+x1x2+x3+x4+x6+x7+x9+x10 ……..……..(2)

The connection polynomial of LFSR is $x^{128} + x^{99} + x^{59} + x^{31} + x^9 + x^7 + 1$. And the filter generator tapset is Γ = {0 1 3 7 12 20 30 44 66 82 127}. Let this stream cipher be *StreamCipher1* as in Figure 2.

Then we use “Pointer and circular buffer” to StreamCipher1 to become *StreamCipher2*. “Pointer and circular buffer” [3] is as follows:

*Pointer and circular buffer***:** is based on the idea of having a pointer pointing at the beginning of the LFSR in memory. When we clock the LFSR once we do not shift all the values one step in memory, but rather, we only move the pointer one position. This gives a compact code description of the LFSR sequence generation, and is faster than StreamCipher1.

Next we implement the combination generator. The Boolean function is the same as StreamCipher1. And we need 11 connection polynomials of LFSR as follows:

$X^6 + X + 1$ $X^6 + X + 1$

$X^8 + X^5 + X^4 + X^3 + 1$ $X^9 + X^4 + 1$

$X^{10} + X^3 + 1$ $X^{11} + X^2 + 1$

$X^{12} + X^7 + X^4 + X^3 + 1$ $X^{13} + X^4 + X^3 + X^1 + 1$

$X^{14} + X^{12} + X^{11} + X + 1$ $X^{14} + X^5 + X^3 + X + 1$

$X^{15} + X + 1$ $X^{16} + X^5 + X^3 + X^2 + 1$

We use these LFSRs and the Boolean function f to construct the *StreamCipher3* as Figure 3.

In Table 2 it is obvious that *StreamCipher2* is the smallest. Therefore, we choose the filter generator as our generator in the stream cipher. At last we implement the filter generator with 8051 assembly code and optimize it to be the smallest by using reuse and loop and so on. It will produce code size of 799 bytes. Key setup and running 128-bit keystream totally approximately cost 0.031426 seconds. The data rate of producing the keystream is 128 / 0.031426 = 4073 bits/s.

We just xor the plaintext with the keystream to complete encrypting.

	StreamCipher1	StreamCipher2	StreamCipher3
Code Size	3.56k bytes	2.56k bytes	5.77k bytes
time	0.050614 s	0.050409 s	0.167840 s

Table 2: Code size of three stream cipher

3.2 Implementing the stream cipher with hardware

We devise these sensors to last as long as possible on Wireless Sensor Networks. We want to lower power consumption when sensors encrypt with the stream cipher. In the same CPU clock rate the algorithm of the faster encrypting consumes the lower power. Therefore, we want to make our encryption algorithm faster to lower power consumption. Of course, we also want our algorithm to use less memory. In order to save power consumption and memory we may change the hardware specification to better fit our stream cipher.

First we write the stream cipher algorithm with 8051 assembly code. We use the filter generator as our stream cipher generator. This filter generator consists of one LFSR of length 128 and one 11-variable Boolean function. LFSR of length 128 needs 16 addresses to be stored (one address is 8bits), that is from KEY0 to KEYF. The Boolean function needs 11 inputs from LFSR and one input is one bit. Therefore we often extract one bit from some address and it needs to do 11 times. Doing one time needs many operations as follows.

```
MOV     A, 30H
ANL     A, #08H
RR  A
RR  A
RR  A
MOV     R2, A
```

Doing these is only to move fourth bit of the value in address 30H to register 2. The connection polynomial of LFSR also needs to do these operations to compute the next state. So doing these operations in the filter generator needs totally 17 times. If we can increase one new instruction to replace these operations, the code size will reduce much and the speed of encryption will be faster.

After observing the stream cipher program with assembly code, we find that increasing this instruction, MOV Rn, ADDRESS.m, is a good idea. This instruction means to move m-th bit of the value in address ADDRESS to the register n; that is, this instruction can replace the above all instructions.

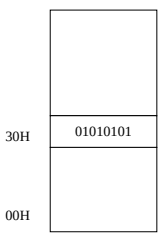

Figure 4: Memory

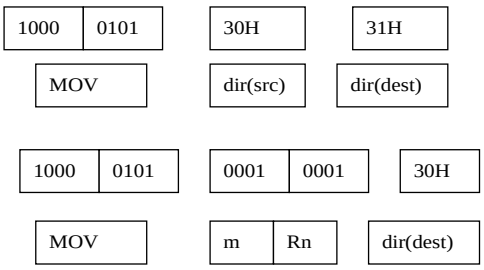

Figure 5: New instruction

For example MOV R1, 30H.0 => R1 = 1, MOV R2, 30H.4 => R2 = 0 in Figure 4. By using this instruction we will reduce code size and increase the speed of the stream cipher. How do we increase this new instruction in 8051? We first find out the source code of 8051 and modify it to increase this instruction. The source code of 8051 is VHDL or may be Verilog. But we only simulate it and do not implement it in reality. So we find a simulation of 8051 written by C++ and modify it to increase this instruction. How do we modify the simulation of 8051? If we can find out opcode which is not used in 8051, then we use this opcode as the opcode of our new instruction. The easier method is to modify the instruction which we do not use in the stream cipher algorithm to become our new instruction. In this case, we modify MOV DIRECT, DIRECT as Figure 5. In this Figure 5, the original instruction MOV dir, dir, is to move the value in address 30H to the value in address 31H. The modified instruction, MOV m, Rn, dir, is to move m-th bit of the value in address dir to register n, Rn. In Figure 5, MOV 11H, 30H is to move second bit (begin from 0) of the value in address 30H to R1. We take Figure 5 as an example, that is, R1 is equal to 0. By using this instruction we can largely decrease the code size of the stream cipher and increase the speed of the stream cipher. Table 3 compares the stream cipher not using the modified instruction and one using the modified instruction with regard to the code size and execution time. This modification improves by 799 – 578 = 221 (bytes) and 0.031426 – 0.024642 = 0.006784 (s). The improved rate of code size is 221/799 = 27.7% and the improved rate of execution time is 0.016784/0.031426 = 21.6%.

	original	modified
code size (bytes)	799	578
execution time (s)	0.031426	0.024642

Table 3: Code size and execution time of the stream cipher

3.3 Analyzing security

In the previous subsection, we obviously know the code size of the filter generator is smaller than one of the combination generator. On sensors memory is very critical. Because the filter generator is a special case of the combination generator and they share the same Boolean function f in (2), that is, they have the same nonlinearity, resilient, algebraic immunity, so they have the same power to resist some attacks, such as the BAA attack, the correlation attack, and the algebraic attack. Therefore we will choose the filter generator as our cryptosystem on sensors.

The structure of the filter generator is shown in Figure 2. Because the connection polynomial c is a primitive polynomial so the period of the sequence s and z are $2^k – 1$ if f is balanced [6]. In the filter generator the period of the keystream is $2^{128} – 1$. We believe it is long enough.

test	degree of freedom	passing range	other paratmeter	results	
frequency test	1	-3.84 ~ 3.83	no	pass	X1=1
serial test	2	-5.99 ~ 5.99	no	pass	X2=1
poker test	7	-14.067 ~ 14.067	m = 3	pass	X3=4
runs test	14	-23.685 ~ 23.685	k = 8	pass	X4=11.4
autocorrelation	no	-1.96 ~ 1.96	d = 500	pass	X5=1

Table 4: Statistical tests table of the filter generator

We also hope the keystream the generator produces possesses the randomness. While it is

impossible to give a mathematical proof that a generator is indeed a random bit generator, the statistical tests [7] help detect certain kinds of weakness the generator may have. Table 4 shows that our choosing filter generator is very probable to random.

Then we hope our filter generator can resist all kinds of attacks. The Boolean function f in the filter generator is (11, 5, 5, 960) and its AI is 4. We believe 5-resilient is big enough to resist all correlation attacks. Nonlinearity is equal to 960 and in the BAA attack a = 0.0625 and the sequence the BAA attack [9] generates is similar with the original keystream with probability of 0.53125. This value is low enough to resist the BAA attack. The generator filter tapset and the LFSR tapset are FPDS to resist the inversion attack and the conditional correlation attack.

attack	algebraic	BDD	Investion	tradeoff
complexity	$O(2^{65})$	$O(2^{114})$	$O(2^{82})$	$O(2^{85})$

Table 5: Complexity of attacks

Table 5 shows the complexity of other attacks. Let CPU clock rate be 4G, and it computes at most 2^{48} instructions in one day. Therefore if all complexity is larger than 2^{64}, we say the stream cipher is secure. So our stream cipher is secure.

Compared with RC5 in [10] the filter generator uses less code size and is faster. RC5 was used as the cryptosystem on Wireless Sensor Network in [10]. The faster the operations of encrypting are in the same clock rate, the less power consumption is. This is also very important on sensors and this makes sensors survive longer. At last, we compare the filter generators, RC5 and A5. RC 5 and A5 are implemented by 8051 assembly code. The filter generator is implemented by our modified 8051 assembly code. The result is as follows.

	filter generator	RC5	A5
code size (bytes)	578	1789	1071
data rate (bits/s)	5194	600	3318

Table 6: Comparison among the filter generator, RC5, and A5

Obviously, our filter generator is faster than RC5 and A5 and uses less code size.

4. Conclusion

The stream cipher is divided into the combination generator and the filter generator. The filter generator needs less code size than the combination generator. The filter generator is also faster than the combination generator. Therefore, we choose the filter generator as our cryptosystem on Wireless Sensor Networks. And our designing stream cipher can be loaded into sensors and it is secure.

5. Reference

[1] Turiy Tarannikov, “On resilient Boolean functions with maximal possible,” Crypto ePrint Archive, http://eprint.iacr.org, No. 2000/005.

[2] Anne Canteaut, Kapaleeswaran Viswanathan, “Results on Algebraic Immunity for Cryptographically Significant Boolean Functions,” Progress in Cryptology - INDOCRYPT

2004: 5th International Conference on Cryptology in India, Chennai, India, December 20-22, 2004. Proceedings.

[3] P. Ekdahl and T. Hohansson, “SNOW-a new stream cipher,” in Proceedings of First Open NESSIE Workshop, KU-Leuven, 2000.

[4] Jovan Dj. Golic, ”On the security of nonlinear filter generators,” In Dieter Gollmann, editor, Fast Software Encryption (FSE 1996), LNCS 1039, pages 173-187. Springer-Verlag, 1996.

[5] B. Löhlein, “Analysis of modifications of the conditional correlation attack,” 1999. Accepted at 3rd IEEE/ITG Conference on Source and Channel Coding, 17-19 Jan. 2000, Munich.

[6] Markus Schneider, “Methods of generating binary pseudo-random sequences for stream cipher encryption (in German),” PhD thesis, Faculty of Electrical Engineering, University of Hagen, Germany, September 1999. Berichte ausder Kommunikationstechnik, Band 4, Shaker Verlag.

[7] A. J. Menezes, P. C. V. Oorschot, and S. A. Vanstone, Handbook of Applied Cryptography, pp.169-190, 1996.

[8] T. Siegenthaler, “Decrypting a class of stream ciphers using ciphertext only,” IEEE Transaction on Computers, 1985, C-34, pp. 81-85.

[9] R. A. Rueppel, “Analysis and design of stream cipher,” Springer-Verlag, Berlin etc., 1986.

[10] Perrig, R. Szewczyk, V. Wen, D. culler, and J. Tygar, “SPINS: Security Protocols for Sensor Networks,” In Seventh Annual ACM International Conference on Mobile Computing and Networks (Mobicom 2001), Rome Italy, July 2001.

6. 成果自評

依前幾節所述之結果，我們達成了此計畫預期的目標。此計畫的研究結果不僅針對串流密碼的核心技術提供理論上的安全檢測標準，更具體提出幾個建構的方向，並且提供在軟體及硬體上的實作方式，可應用在記憶體及運算資源有限的感測器上。成果極具有學術上的價值與貢獻，相當適合學術期刊上發表。