On Route to Secure Sensor Networks

Embedded Encryption

Johan Dams
jd@puv.fi

Why ?

- Encrypting data stream in sensitive environments
- Both for sensor networks and embedded systems
  - Healthcare: remote patient monitoring
  - Security applications

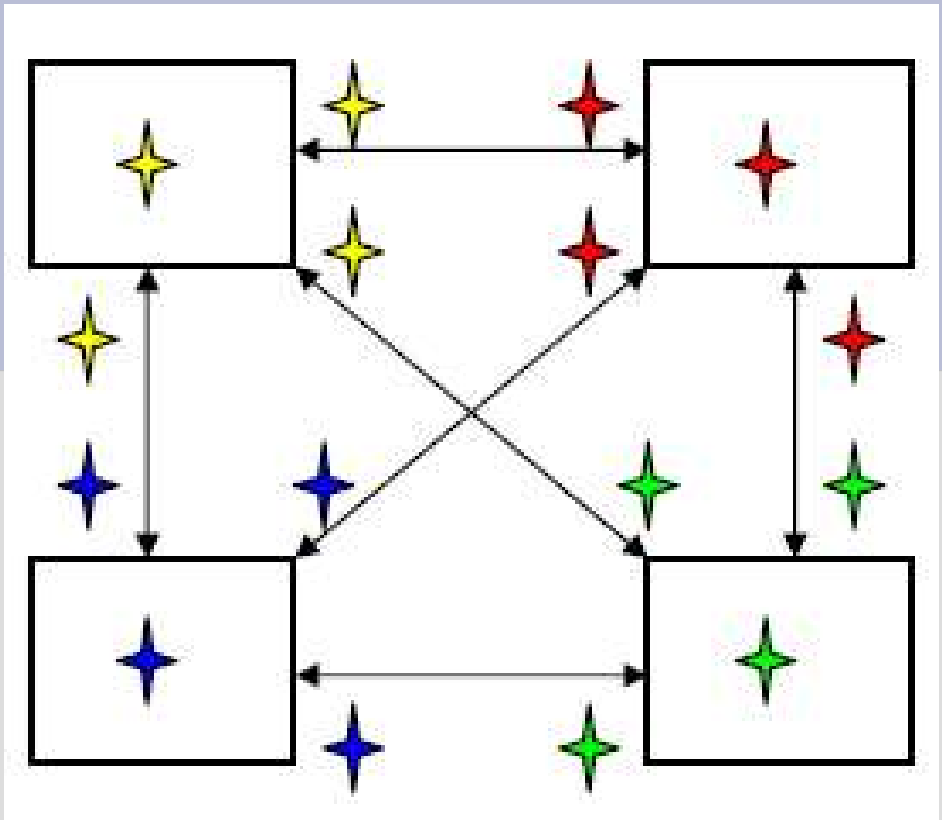
How ?

- Encryption methods are very intensive on memory and computing power
  - Lots of mathematical operations going on
  - On really big numbers
- Small embedded systems and sensor nodes very limited
  - Small amounts of memory
  - 8-bit / 16-bit, limited clock speed
  - Often Battery Powered
    - Energy concerns

Types of Encryption

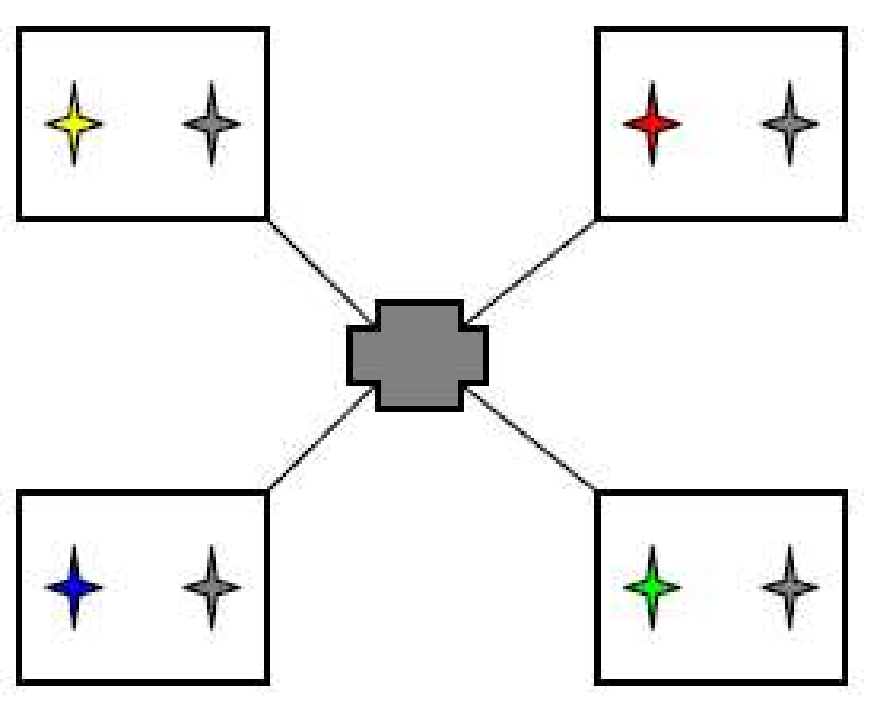
- Symmetric Key Encryption
  - Both parties need to have access to the key before communication
  - Same key for encryption and decryption
- For embedded systems and sensor networks in particular:
  - not useful
    - Adding and deleting nodes 'on the fly' would be highly impractical
    - Large number of keys need to be stored on each chip

(see figure next slide)



- Maintaining unique symmetric keys for each communicating pair of entities would require the management of: N x (N –1) / 2
- Thus 1000 users = 499,500 keys
- Way around this? See later...

- Asymmetric Key Cryptography (Public Key)
  - Private key kept secret
  - Public key can be freely distributed
  - Decryption key is not be deducible from the encryption key
- Advantages
  - Simplifies key exchange enormously
  - Smaller number of keys (see next slide)
  - In sensor networks, nodes could be added or removed without key changes on other nodes
- Disadvantages
  - Mathematical operations are more complex



- N = Number of Key Pairs
- Thus: 1000 nodes = 1000 key pairs

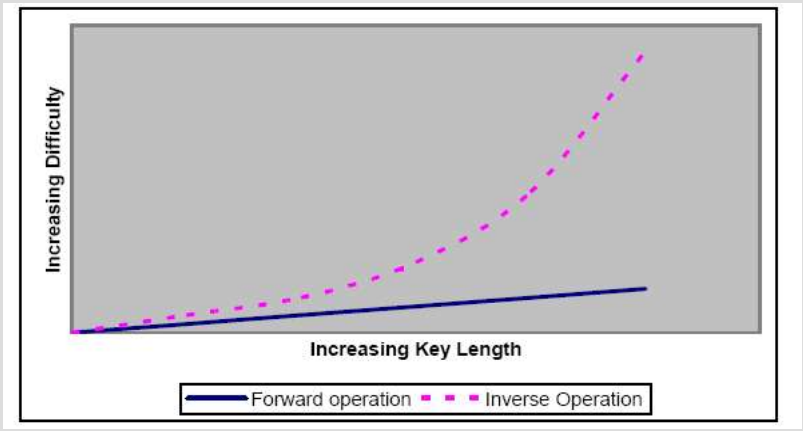
- What can be done: Hybrid Crypto-system
  - Use asymmetric crypto to provide a common key to all nodes
  - After that, use simpler and less intensive math using this symmetric key
- Disadvantage
  - All host use the same key
    - Unacceptable in security applications
      - e.g., wireless sensor network for security applications: If one node is compromised, the entire network cannot be trusted anymore
  - Lack of authentication methods
    - Distinct feature of asymmetric crypto systems
  - Messy and difficult to manage

What to do ?

- Seems no good solution possible
  - Symmetric:
    - Easy mathematically
    - But:
      - How to distribute the keys?
  - Asymmetric
    - Easy Key Distribution
    - But:
      - Mathematically heavy
- Other problem not discussed before:
  - Key length!
    - The size of keys make it inefficient on limited hardware

Interesting though...

- Asymmetric Encryption works like this:
  - There are two operations
    - Forward: Easy
    - Inverse: Hard
  - e.g., RSA
    - Two large prime numbers:
      - 65731 and 83497 (still small for real use, normally 200 digits)
      - Easy operation: 65731 x 83497 = 5488341307
      - Hard operation: Given 5488341307, find the 2 primes that were multiplied
    - Keys are functions of the product and of the primes
      - Operations we wish to be easy require performing the relatively easy forward function; multiplication.
      - The operations we wish to make difficult - finding the plaintext from the cipher-text using only the public key - require performing the inverse operation; solving the factoring problem.



A Solution

- Elliptic Curve Cryptography (ECC)
  - Public Key => Easy distribution of keys
  - Short key sizes possible
  - Mathematically still very heavy... But there are ways to make this work.

ECC?

- Evolved from Diffie Hellman
  - Diffie Hellman uses a problem known as the discrete logarithm problem as its central, asymmetric operation
  - The discrete log problem concerns finding a logarithm of a number within a finite field arithmetic system
    - Prime fields are fields whose sets are prime, that is, they have a prime number of members.
    - Over a prime field, exponentiation turns out to be a relatively easy operation, while the inverse, computing the logarithm, is very difficult.

- Diffie-Hellman Method for Key Agreement
  - allows two hosts to create and share a secret key
  - Step 1)
    - Both sides need to get the parameters
      - Prime number 'p' > 2
      - Base 'g', integer < 'p'
    - These can be hard-coded, or fetched from a server

- Step 2)
  - The hosts each secretly generate a private number called 'x', which is less than "p – 1"
- Step 3)
  - The hosts generate the public keys, 'y'. They are created with the function:

    y = g^x % p
- Step 4)
  - The two host now exchange the public keys ('y') and the exchanged numbers are converted into a secret key, 'z'

    z = y^x % p

- 'z' can now be used as the key for whatever encryption method is used to transfer information between the two hosts
  - Mathematically, the two hosts have generated the same value for 'z'.

    z = (g^x % p)^x' % p = (g^x' % p)^x % p
    - All of these numbers are positive integers

  - The discrete logarithm problem, using the values in the equation above, is simply finding x given only y, g and p.

- Back to ECC...
  - also uses a discrete log problem in a finite group.
  - Difference is that ECC defines its group differently.
    - it is the difference in how the group is defined — and particularly how the mathematical operations within the group are defined — that gives ECC its greater security for a given key size

- ECC's advantage:
  - Its inverse operation gets harder, faster, against increasing key length than do the inverse operations in Diffie Hellman and RSA
  - This means ECC offers greater security bit-for-bit

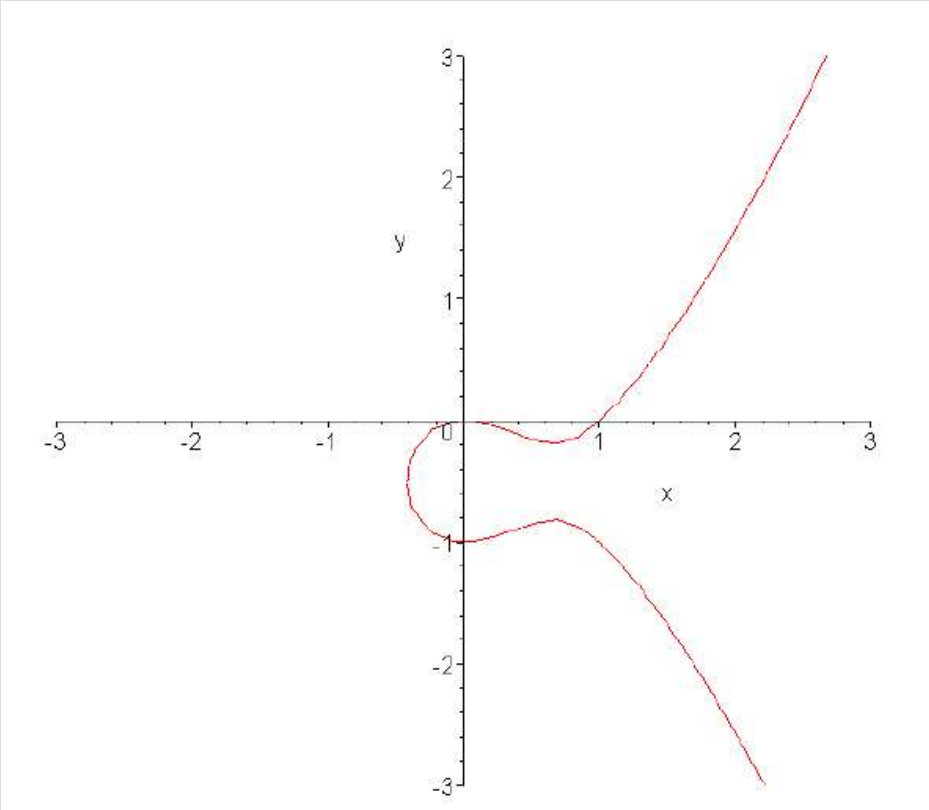
ECC key size (bits)	RSA key size (bits)	Key size ratio
163	1024	1/6
256	3072	1/12
384	7680	1/20
512	15360	1/30

Table 1: Equivalent key sizes for ECC and RSA, supplied by NIST to ANSI X9F1

How ECC works

- An elliptic curve is defined in a standard, two dimensional x,y Cartesian coordinate system by an equation of the form:

  y^2 + y = x^3 - x^2



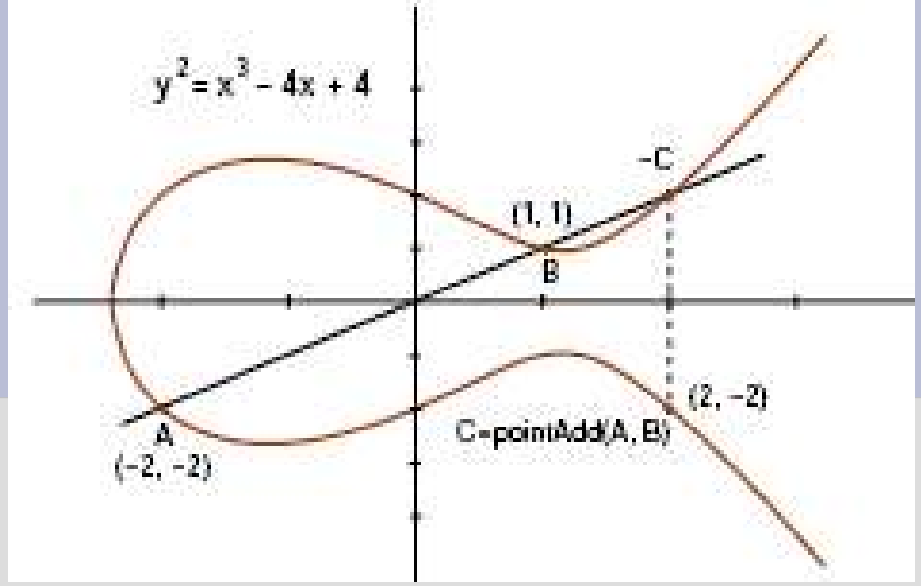
- The elliptic curve is used to define the members of the set over which the group is calculated, as well as the operations between them, which define how math works in the group
- It goes like this:
  - imagine a graph labeled along both axes with the numbers of a large prime field
    - a square graph, p x p in size, where p is a very large prime number
    - This is hard to imagine, so use e.g., 17 as the prime

- Fp is the field of integers modulo p, and consists of all the integers from 0 to p-1
- So you'd be looking at a graph 17x17 (example) units in size.
  - If you define an elliptic curve so that there are points (x, y) on the curve that satisfy the condition that both x and y are members of the prime field
  - you have implicitly created a group from the set of integer points on the curve
    - It is a subset of all the points in the p by p matrix created when you drew the graph, specifically the ones the curve passes directly through

- Note:
  - unlike the groups used in Diffie Hellman, the elements of the set aren't integers, but points
  - It contains a set of elements (points, in this case), and when you add one point to another, or subtract one from another, there are rules that say what point in the set you wind up at when you do so.
    - Just as for the integers in the groups used in Diffie Hellman.

The rules of the game

- Point Multiplication
  - The dominant operation in ECC cryptographic
  - critical operation which is in itself fairly simple, but whose inverse is very difficult
  - Point multiplication is simply calculating kP, where k is an integer and P is a point on the elliptic curve defined in the prime field
- This is done by a series of point additions and doublings



y^2 = x^3 + ax + b

- Algebraically, the result of adding points $A(x_A, y_A)$ and $B(x_B, y_B)$ is $C(x_C, y_C)$ such that
  - $x_C = s^2 - x_A - x_B$, $y_C = -y_A + s(x_A - x_C)$
  - where $s = (y_A - y_B)/(x_A - x_B)$ is the slope of the line through A and B.
  - When A equals B, the line through A and B degenerates to the tangent at A and $s = (3x_A^2 + a)/2y_A$.
  - The result of adding A and -A is defined to be a special point called the point at infinity.

Point multiplication Q = kP

Repeated point addition and doubling: 9P = 2(2(P)) + P

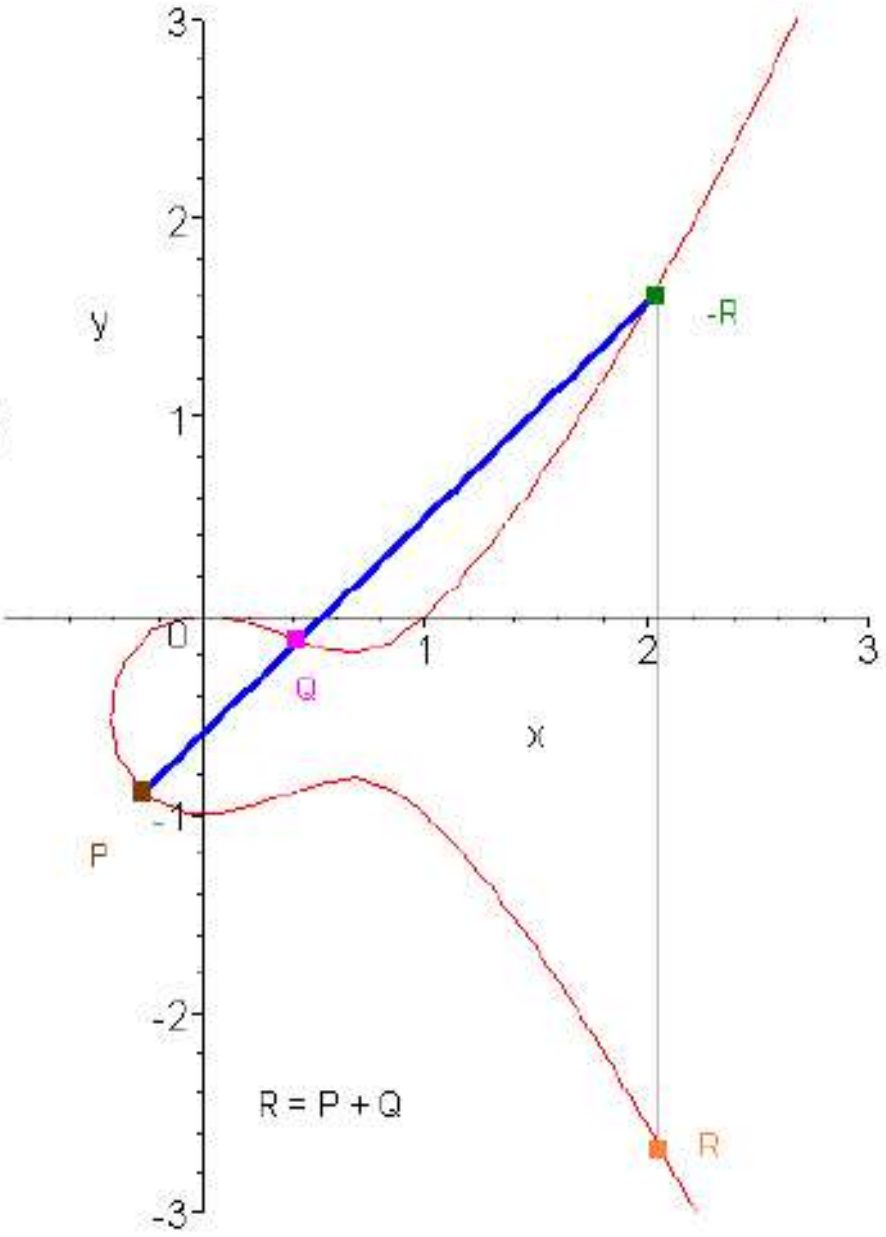
Public key operation
Q(x,y) = kP(x,y)

- Q = Public key
- P = Base point (curve parameter)
- k = Private key
- n = Order of P

Elliptic curve discrete logarithm
Given public key kP, find private key k

Best known attack: Pollard's rho method with running time:

$$((Pi \times n)^{(1/2)})/2$$



- Elliptic curve groups used in cryptography are defined over two kinds of fields:
  - GF(p), where p is a prime,
  - and GF(2^m) where each element is a binary polynomial of degree m (that can be represented as an m-bit string sinc coefficient is either 0 or 1);

- Third type: Optimal Extension Fields
  - Choose p of the form 2^n +/- c, for n; c arbitrary positive integers, where
    - $\log_2(c) <= [½ n]$
    - In this case, one chooses p of appropriate size to use the multiply instructions available on the target microcontroller
    - In addition, m is chosen so that an irreducible binomial P(x) = x^m - w exists, w in GF(p).
  - Generating good curves over OEF's ?
    - Schoof's algorithm

A practical Implementation

- Goal: To provide an ECC system that can be implemented on 8-bit microcontrollers
  - Here 8051
    - Very common
    - quite high-speed available
      - e.g. Dallas Semiconductor DS80C400
    - Easily available for testing
- Secondary goal: Scalable for higher-end Controllers (not yet finished and tested, but should be no problem)

The field

- implementation is based on the use of the Optimal Extension Fields GF((2^8 – 17)^17)
  - Why?
    - 2^8 – 17 = 239, largest prime able to fir into an 8-bit register
  - For 16-bit micros : GF((2^13 – 1)^13)

Implementing

- Optimisation – Some algorithms
  - Extension field multiplication is the most costly basic arithmetic function in OEFs
  - For a given extension field of order n, n^2 subfield multiplications are required to multiply two values using traditional polynomial multiplication

- Demonstration:
  - Given two degree-1 polynomials, A(x) and B(x), we can demonstrate:
    - A(x) = a1x + a0
    - B(x) = b1x + b0
    - we must calculate the product of each possible pair of coefficients.
      - D0 = a0b0
      - D1 = a0b1
      - D2 = a1b0
      - D3 = a1b1
    - Now we can calculate the product C(x) = A(x) . B(x) as:
      - C(x) = D3x^2 + (D2 + D1)x + D0

- We can do this faster: Karatsuba Method
  - begins by taking the same two polynomials, and calculating the following three products:
    - E0 = a0b0
    - E1 = a1b1
    - E2 = (a0 + a1)(b0 + b1)
  - These are then used to assemble the result C(x) = A(x) . B(x):
    - C(x) = E1x^2 + (E2 - E1 - E0)x + E0

- Comparison
  - The traditional method requires four multiplications and one addition
  - while the Karatsuba method requires three multiplications and four additions.

  - Thus we have traded a single multiplication for three additions. If the cost to multiply on the target platform is as least three times the cost to add, then the method is effective.
  - Not on 8051, but on several 16 bit and more CPU's

- Point Multiplication
  - the primary operation in an elliptic curve cryptosystem is point multiplication
  - C = kP. For large k, computing kP is costly
  - ordinary integer exponentiation can be adapted to this setting
  - The most basic of these algorithms is the binary-double-and-add algorithm
    - On average 1,5 $\log_2(k)$

- Other optimisations are possible
  - e.g. Somehow using de Rooij
    - Very useful with digital signatures, as the point can be known ahead of time
    - Use on smartcard
  - Needs more work...
- Itoh-Tsujii inversion
  - reduction of the extension field inversion to a subfield inversion
    - subfield inverse can be calculated by efficient means, such as table-lookup or the Euclidean algorithm, given a small order of the subfield.

Some numbers

Field	appr. Field Order	# Cycles for Multiply
GF(2^135)	2^135	19,000
GF((2^8)^17)	2^136	7,000
GF((2^8 – 17)^17)	2^134	5,000

- Generic binary fields offer performance which lags behind
- certain composite fields have been shown to have cryptographic weaknesses
- OEF best for our application

Conclusion

- A lot of work still needs to be done
- Implementing ECCs on the 8051 is a challenging task... But it can be done!
- Standard 8051, 256 bytes RAM, ... Tight fit.
  - Better 8051 implementations will yield better results

- Each curve point in our group occupies 34 bytes of RAM
  - 17 bytes each for the X and Y coordinates.
  - To make the system as fast as possible, the most intensive field operations, such as multiplication, squaring, and inversion, operate on fixed memory addresses in the faster, lower half of RAM.

- During a group operation, the upper 128 bytes are divided into three sections for the two input and one output curve points
  - the available lower half of RAM is used as a working area for the fieeld arithmetic algorithms.
  - A total of four 17-byte coordinate locations are used
- Finally, six bytes are used to keep track of the curve points, storing the locations of each curve point in the upper RAM

- Using these pointers, we can optimize algorithms that must repeatedly call the group operation, often using the output of the previous step as an input to the next step.
- Instead of copying a resulting curve point from the output location to an input location,
  - which involves using pointers to move 34 bytes around in upper RAM,
- we can simply change the pointer values and effectively reverse the inputs and outputs of the group operation

- Code size to implement
  - Approx. 16KB
    - Can be optimised

- Some statistics (12Mhz Intel 8051) (approx.)

		t(uS)
– Multiplication	C(x) = A(x)B(x)	5300
– Squaring	C(x) = A(x)^2	3300
– Addition	C(x) = A(x) + B(x)	300
– Inversion	C(x) = A(x)^-1	25000
– Scalar Mult.	C(x) = sA(x)	700

The End