Universität Karlsruhe (TH)
Institut für Telematik

TELEMATICS TECHNICAL REPORTS

Efficient Implementation of Elliptic Curve Cryptography for Wireless Sensor Networks

Erik-Oliver Blaß, Martina Zitterbart
{blass,zit}@tm.uka.de

March, 18th 2005

TM-2005-1

ISSN 1613-849X
`http://doc.tm.uka.de/tr/`

Institute of Telematics, University of Karlsruhe
Zirkel 2, D-76128 Karlsruhe, Germany

Efficient Implementation of Elliptic Curve Cryptography for Wireless Sensor Networks

Erik-Oliver Blaß and Martina Zitterbart

Institut für Telematik, Universität Karlsruhe
Zirkel 2, 76128 Karlsruhe, Germany
{blass, zit}@tm.uka.de
http://www.tm.uka.de

Abstract. One of the huge problems for security in sensor networks is the lack of resources. Typical sensor nodes such as the quite popular MICA and MICA2 Motes from UC Berkeley [1] are based on a microcontroller architecture with only a few KBytes of memory and severe limited computing ability. Strong public-key cryptography is therefore commonly seen as infeasible on such devices. In contrast to this prejudice this paper presents an efficient and lightweight implementation of public-key cryptography algorithms relying on elliptic curves. The code is running on Atmels 8Bit ATMEGA128 microcontroller, the heart of the MICA2 platform. The key to our fast implementation is the use of offline pre-computation and handcrafting.

1 Introduction

Wireless sensor networks are one of the key technologies of the ubiquitous computing vision. Physically small sensor devices are able to communicate with each other using radio interfaces. They cooperate and offer pervasive services to the user. Tiny sensors are often embedded into ordinary items of our daily live such as clothes or attached to office rooms and thus allowing assistance in day-to-day situations. There exists a large number of different scenarios for sensor networks one can imagine: examples are environment observation, medical monitoring of body functions as well as office and even military applications.

Furthermore there is a large number of scenarios where the data transported and exchanged between sensor nodes is critical. Imagine someone fooling around with the values a heartbeat sensor transmits to an alarm station or a business competitor coming to know important and valuable information a user dictates to autonomous microphones in his office. Such data has to be protected against threats in a way so classic security properties like integrity, authenticity or confidentiality can be guaranteed. To accomplish such security goals in modern networks like the Internet or a companies local area network cryptographic primitives like encryption/decryption as well as signature schemes are usually needed. The question is whether these primitives can be used in sensor networks as well.

Node hardware used in wireless sensor networks differs a lot from hardware commonly seen in desktop or sever systems. As cost and energy saving are of paramount importance most commonly utilized sensor hardware is based around a battery powered microcontroller like e.g. Atmels 8Bit ATMEGA128. Systems like UC Berkleys MICA platform [1] or our own sensor nodes K-SNeP [2], [4], [3], [5], [6] are built around this core. Other nodes are based on Texas Instruments' MSP430F149 [7] or Intels ARM technology [8]. All these sensor nodes share one common problem as soon as it comes to cryptographic operations: their extreme limited hardware configuration. For example the ATMEGA128 features only 4KBytes of main memory (RAM), 4KByte of EEPROM and 128KByte of ROM. The chip is clocked to about 7Mhz by default allowing 7 million instructions per second (MIPS).

Because of these restrictions cryptography is commonly considered as a delicate problem throughout the community. While there is consensus that symmetric ciphers might work (see related work), even though limited, there is a prejudice against the feasibility of well known asymmetric methods based on the RSA-problem or the Diffie-Hellman-problem. This is due to the fact that a pleasing implementation of asymmetric algorithms giving satisfying performance together with minimal memory consumption is yet missing. Algorithms like RSA, El-Gamal or DSA seem to overstrain tiny sensor hardware. These asymmetric algorithms are known to consume a lot of memory and computing time. Especially computing time is critical within sensor nodes, as they are battery powered and thus not chargeable at any time. For the design of new security protocols for sensor networks however this knowledge of algortihm usability is crucial: cryptographic primitives are the fundamental building blocks of every secure protocol. If you can show that asymmetric encryption is feasible in sensor networks at a certain level of cost it becomes possible to invent new security protocols or adapt existing ones for sensor networks. Such protocols would not need complex detours to substitute asymmetry by symmetric operations like [30] or [10].

The contribution of this work is an implementations of asymmetric encryption and signature generation schemes for the 8Bit ATMEL sensor platform that features acceptable run-time and memory consumption while preserving a level of acceptable security for sensor networks. This allow the design of new security protocols utilizing public-key techniques.

Hereby our implementation is based on elliptic curves – a relatively new cryptographic technique which promises faster and memory efficient encryption, decryption and signature generation than other traditional approaches. Our fast implementation is based on handcrafted optimization on the one hand and the precomputation of certain points on the other.

2 Related work

As wireless sensor networks are becoming an attractive research field the problem of securing them emerges more and more as a hot topic. Most security work focuses on the search for and development of alternatives to classical public-key algorithms and public-key infrastructures. The need for an alternative to public-key infrastructures is due to the fact that the paradigm of sensor networks seems contrary to a centralized *certificate authority* (CA) or any kind of centralized communication. Sensors might meet spontaneously in an ad-hoc manner and may be not be able to access anything infrastructure like at any time. See [30], [3], [5] or [31] for an overview.

Secondly as no satisfying implementation of efficient asymmetric cryptography on microcontroller based sensor hardware exits there is apparently the need to look for alternatives. Most of the published work focuses on mimicking centralized CA, certificate and key distribution as well as common signatures schemes and asymmetric algorithms by decentralized key distribution and pure symmetric cryptography. Publications like [10] mimic asymmetric signatures schemes by a relatively complex scheme of two party hash chains, so do [32], [33] and [34]. Other work like [35] try to establish pairwise secret keys to avoid public and private key schemes or Diffie-Hellman like key exchanges.

All of this publications assume first of all the complete impracticability of public-key algorithms and take this as a legitimation or motivation for their work. In [11] and [12] the authors implement elliptic curve cryptography for sensor networks. However the underlying hardware is quite sophisticated consisting of 16 Bit microcontrollers with 16 MHz clock frequency. Therefore the results are only of limited value as typical sensor hardware does not dispose of such powerful computing resource. As mentioned before sensor networks can in general not afford high clock frequencies and potent CPUs, because of cost and energy saving issues associated.

In [36] a high-performance microcontroller offerings 24 MIPS, i.e. 3 times more than the usual ATMEGA 128, is utilized. The work is based on special Galois fields called *optimal extension fields* where field multiplication can be done quite efficiently. However the security of this fields is unclear because of the Weil descent attack [37].

The workings trying to implement elliptic curves on 8 Bit ATMEGA128 chips like in [15] and[16] only reaches extremely poor performance: e.g. for a signature generation over 1:08min of expensive computing and battery time has to be spent, which surely is not affordable. In addition the cost for necessary field operations are not mentioned at all.

3 Elliptic Curve Cryptography

Elliptic curves are an algebraic structure whose use for cryptography was first mentioned in [13] and [14]. They feature properties which allow the setup of a problem similar to the well known discrete logarithm problem of finite fields – also knows as *Galois* fields. The subsequent section gives a brief and rough mathematical background to understand our implementation.

3.1 Overview

An elliptic curves is a set points over a field that satisfies a certain equation. Is a curve defined over the field $\mathbb{F}$ all of its points (x, y) with $x, y \in \mathbb{F}$ satisfy the so called

Weierstraß-equation

$$y^2 + a_1xy + a_3y \equiv x^3 + a_2x^2 + a_4x + a_6,\ a_1, a_2, a_3, a_4, a_6 \in \mathbb{F}$$

Here a_i are the parameters of the curve. Figure 1 is an example of an elliptic curve

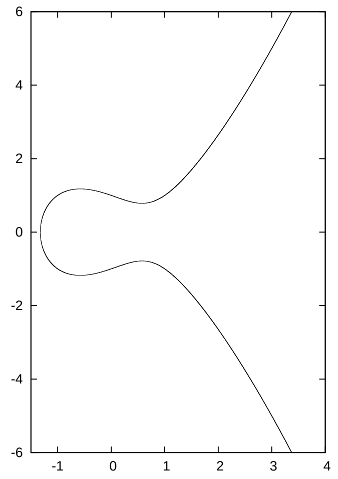

Fig. 1. Example of an elliptic curve of the reals. Curve parameters are here: $a_4 = -1$ and $a_6 = 1$.

defined over the (infinite) field $\mathbb{R}$. Here curve parameters are: $a_1 = 0$, $a_2 = 0$, $a_3 = 0$, $a_4 = -1$ and $a_6 = 1$. As we will see in the next section usually Galois fields are used for elliptic curve cryptography (ECC) as they allow the introduction of a problem similar to the discrete logarithm problem. For curves over typically used Galois fields $\mathbb{F}_{2^p}$ the Weierstraß equation applies in a reduced form: $y^2 + xy \equiv x^3 + a_2x^2 + a_6$ [17].

3.2 Elliptic Curve Discrete Logarithm Problem: ECDLP

The problem to find logarithms is difficult to compute in finite fields. Assume eg. an large prime p, an out of it resulting finite field $\mathbb{F}_p$ and the equation

$$a = b^c \bmod p,\ a, b \in \mathbb{F}_p,\ 0 \leq c \leq p - 2$$

the task is to find c with given a and b. There is no known algorithm to compute a so called *discrete* logarithm in polynomial time. The complexity to find a discrete logarithm is linear with number of elements in $\mathbb{F}_p$ ie. complexity rises exponential with the size of p in bits.

Addition and Multiplication Something similar holds for elliptic curves over finite fields. You can *add* two points $A, B \in K$ from a curve K which results in a new point

$C \in K$. We won't go much into details here but the addition of two points A, B is done by drawing a line through A and B. This line intersects K at exactly on point $C' \in K$. The sum C of $A + P$ is point C' mirrored at the y-axis. Point subtraction looks similar as this is like an addition with mirrored y-coordinate. It is not necessary to introduce the exact definition of a two point addition here any further, but only of value to notice that such point additions include certain field operations like field addition, field multiplication and even field inversion with x and y coordinates of a point P.

Together with a point 0 at infinity addition of points on elliptic curves gives an algebraic structure called a group. Within this group a problem analog to the discrete logarithm problem can be introduced.

A multiplication

$$Q = \underbrace{P + \cdots + P}_{k\ times,\ k \in \mathbb{N}}$$

of a point P with an integer k can be seen as the multiple addition of one and the same point. The resulting product Q is again another point on the curve. Given a curve K over a finite field F_{2^p}, a point $P \in K$ and a product Q the problem is to find a $k \in \mathbb{N}$ that holds $Q = kP$. This problem is called *Elliptic Curve Discrete Logarithm Problem ECDLP* and hard to solve [13]. For example with a finite field $\mathbb{F}_{2^p}$ with 2^p elements you need about $O(2^{\frac{p}{2}})$ operations to find k [18].

On top of this problem cryptographic algorithms can be set up like Diffie-Hellman [19], El-Gamal [20], DSA [21] and so on. The description of user keys ie. keys a user possesses like his private/public key pair is up to the sections below describing different algorithms based on ECC. The size of such a key is simply the bit size necessary to hold the key – in convenience with all the cryptography you already know. E.g. an elliptic curve over F_{2^p} consists of 2^p elements (points) – here a user key would be $log_2 2^p = p$ Bits as known from traditional methods.

Elliptic curves offer the same security than traditional based algorithms but consume a lot less of memory and computing power. Lenstra published in [18] a comparison between different cryptographic approaches regarding key size and security. As an example: RSA with a key size of 622 Bits offers the same security against attacks as an elliptic curve with only 105 Bit key size – while consuming a lot more computing time and memory. This is what makes elliptic curves attractive for wireless sensor networks.

4 Implementation details

This section deals with implementation internals and describes all the details done to achieve maximum performance while preserving main memory. The reader may skip to section 5 to learn about the results of the implementation.

As a starting point Rosings source code from [22] was taken. This code is designed for conventional PC and desktop computer hardware the key to an efficient and working sensor network implementation is to slim, optimize and tweek such an implementation by hand.

4.1 Key Size

One of the first and most important decision to take regards key size. In general it is of course better to choose a very large key size as this automatically means better security – on the other hand larger key sizes mean even more expensive computation time, thus a tradeoff has to be found. The smallest but secure key size has to be found.

As a good starting point one has to think about what minimum key size is commonly seen as a *secure* key size which means it is understood as infeasible to break for modern hardware. There are a lot of theoretical discussion about this topic (see [18]) but only a few attempts to actually break ECDLP in real life.

The largest broken ECDLP yet had 109 Bit key size ie. over the finite field $\mathbb{F}_{2^{109}}$ – it took 17 months[9]. Although 17 months with huge computing power (2600 workstations) is a lot of time, security of 109 Bit keys size is now debatable and one should choose larger keys for securing very sensible information even in sensor networks.

Of course for hardware restrained sensor networks smaller key sizes might be possible, e.g. [15][16] use initially 33 Bit keys, the question to answer here is whether adversaries in sensor networks are limited likewise their fair-minded victims – restricted sensor nodes. Typically you must consider adversaries as by far more potent in terms of computing power and memory storage in comparison to your own abilities. E.g. adversary nodes in a sensor network could have a link to a mighty base station or people using equipment for eavesdropping could catch all sensor network communication and feed their super computers with it.

The next greater than 109 Bit possible key size/curve that can be described using a very memory saving optimal normal base (see below) technique is 113 Bit, ie. a curve over $\mathbb{F}_{2^{113}}$. This curve offers about $2^4 = 16$ times more security than 109 Bit which seems enough security for todays hardware. With $a_2 = 1$ and $a_6 = 1$ the number of points n on such a curve is relatively large with $n \approx 1.038 \cdot 10^{31}$ points. Because of the need to choose the smallest possible but larger than 109 Bit key and to consume least main memory we propose 113 Bit keys, as a secure length for ECC keys in sensor networks.

4.2 Optimal Normal Bases

The next decision that has to be made before implementing ECC over $F_{2^{113}}$ is whether to use a polynomial base representation of curve element coordinates $(x, y) = P$ with $x, y \in F_{2^{113}}$ or so called optimal normal base (ONB) representation. Both variants have their pros and cons (see [23]) but in general you can say that ONB is better suited for target platforms where memory consumption is critical [22]. Although it is assumed that software implementations are a little faster with polynomial representation our first step with polynomial base was quite disappointing: we could not shrink this implementation below ≈7 KByte of RAM so it would not fit into the total of 4 KByte of main memory available on an ATMEL128 microcontroller. Therefore and because embedding of data on a curve which is needed in section 5.1 is significant simpler we choose ONB.

Point coordinates are elements of our finite field $F_{2^{113}}$ and are commonly described with polynomials and polynomial mathematics. Coordinate $\beta \in F_{2^{113}}$ can be seen as the polynomial

$$\beta = a_{112}x^{112} + a_{111}x^{111} + \cdots + a_1x^1 + a_0$$

with a_i being the coefficients of β. The set $B = \{\beta, \beta^2, \beta^{2^2}, \cdots, \beta^{2^{112}}\}$ is called a normal basis for $F_{2^{113}}$. Every other element $e \in F_{2^{113}}$ can be written as a linear combination of βs elements [22]:

$$e = e_{112}\beta^{2^{112}} + e_{111}\beta^{2^{111}} + \cdots + e_1\beta^{2^1} + e_0\beta.$$

This makes later on frequently needed squaring of e very simple - it is simply a left rotation of e. Addition is a simple XOR of coefficients from both elements. Multiplication of two different elements $a, b \in F_{2^{113}}$ is way more complicated and we will only give a formula for this:

$$c_k = \sum_{i=0}^{112}\sum_{j=0}^{112} a_{i+k}b_{j+k}\lambda_{ij0}.$$

Here λ_{ij0} is called a multiplication table and depends only on the chosen normal base. As you initially chose such a normal base λ_{ij0} for all upcoming field multiplications it is constant for each and every sensor node using a certain curve for communication. Hence it is ideally suited to swap out of main memory as you might already anticipate. This multiplication table comes regains importance in section 4.3. An optimal normal base for $F_{2^{113}}$ is a normal base with 225 non-zero elements in λ_{ij0}.

4.3 RAM to ROM

One key point to save main memory is moving all larger unchangeable data from RAM to ROM which is generally supported on the Atmel platform. Using a special compiler directive `ROM` as a modifier to data declarations forces compiler, linker and finally the device programmer to place this data into flash-ROM regions next to ordinary program code. Later on small fragments of this region can be copied temporarily from ROM to RAM using the `memcpy_P` command.

In addition Atmels ATMEGA 128 microcontroller features 4 KByte of EEPROM. So EEPROM can also be used to store variables as an alternative to main RAM and later be copied temporarily back in small parts to RAM. EEPROM has the advantage of being re-programmable from the microcontroller during normal sensor operation.

As you might have noticed in section 4.2 every sensor node agreeing to communicate securely using a certain curve over a finite field has the same multiplication table λ_{ij0} which does not change during sensor lifetime. It was therefore very reasonable to:

1. precompute this table offline and distribute it to every sensor node prior to deployment and
2. move λ_{ij0} out of valuable main memory to (flash-)ROM.

Access to sensor hardware is most likely possible in every sensor network scenario as individual nodes need individual informations e.g. ID and hardware address, individual program code and so on during what you might call an *personalization* process.

Therefore λ_{ij0} was once computed for our 113 Bit curve and wrapped into a standard include `.h` file declared as ROM. Each sensor using cryptographic routines includes this `.h` file.

This saved about 908 Bytes of valuable RAM which does not sound much but means 22% of entire main memory after all. For every field multiplication the according 16 Bit values are copied 113 times from ROM to RAM. This seems unnecessary and expensive but did not result in noticeably higher performance penalty in contrast to copying the whole table from ROM to RAM at once and simply indexing its elements afterwards.

The same goes for field inversion operations in $F_{2^{113}}$. The algorithm used here is an implementation [24] which needs two unchangeable arrays for its work. By shifting these to ROM using the same techniques as described above additional 1164 Bytes, which is 28% of main memory, were saved.

4.4 Point Multiplication

As ECDLP is based on a lot of point multiplications this is the most crucial operation in ECC. One multiplication $Q = kP$, $1 \leq k \leq 2^{113}$ implies k point additions which in turn means a lot of field operations. Looking at the description of the different implemented algorithms from section 5 you notice that occurring point multiplication can generally be classified into two different types:

1. point multiplications with an always *fixed* point P (e.g. so-called *base point*) as well as
2. point multiplications with arbitrary non-fixed varying points P.

Class 2 multiplications are implemented using an ECC version of the popular square-and-multiply algorithm for large number exponentiation as described in [25] and [26]. Due to the lack of a more sophisticated and more efficient faster algorithm we have to use this *double-and-add* principle which is relatively slow compared to class 1.

Class 1 multiplications do have an advantage of allowing the use of precomputation – which is our key to speed here. Consider the binary representation of k as:

$$k = k_{112}2^{112} + k_{111}2^{111} + \cdots + k_1 2^1 + k_0 2^0,\ k_i \in \{0, 1\}.$$

As P is considered as a fixed point here for all communication we can set up a buffer where all products $2^i P$ with $0 \leq i < 113$ are stored. The resulting pseudocode looks like this:

```
fixed Point P; //Our forever fixed point
```

```
Point buffer[113]; //Buffer with all the P-doublings

//Precompute points - store them in buffer
buffer[0]=P;
Q’=P;
for (i=1;i<113;i++) {
 Q’=doublePoint(Q’);
 buffer[i]=Q’;
}
```

This one time initial computation of the precomputation table can again be done offline and written to sensor nodes prior to deployment. In comparison to the multiplication tables however such a precomputation table is unique for every sensor node, because P is treated as a secret key in ECC algorithms. Nevertheless `buffer` can be computed and transferred to sensor hardware prior to its deployment e.g. as part of its personalization process. `buffer` has a size of a 3616 Bytes but again perfectly fits into program memory regions of flash-ROM and is usable.

In very rare cases, e.g. if a private key is compromised, `buffer` contents might need to be changed. In a scenario where nodes with compromised keys should be able to generate new ones `buffer` could also fit into EEPROM. The complete calculation of `buffer` would only take 12.96s of computing time on sensor hardware. In a situation where everything has to be kept in main memory 3616 Bytes mean 88% of RAM leaving not much space for other computations besides ECC. Anyhow: both EEPROM and even RAM storage of `buffer` is avoidable in most sensor scenarios because of the possibilities of one time precomputation and pre-deployment access to hardware as mentioned before.

For a new multiplication $Q = k'P$ this means for every Bit k'_i that is set to 1: add 2^iP to Q. Pseudocode:

```
//From now on for every multiplication Q=k’P do:
Q=pointZero;
for (i=0;i<113;i++) {
 if (k’ mod 2 == 0)
  Q=addPoint(Q, buffer[i]);
 shift_right(k’);
}
return Q;
```

Instead of adding P together k'-times, k' is a large number with $1 \leq k \leq 2^{113}$, we can simply use our table of precomputed points and simply add no more than 113 times to obtain Q.

As we will see in table 1 our class 1 multiplication is faster by a factor of 2.56 than class 2 multiplication.

Remark: for both types of multiplication necessary point addition is done with [24].

4.5 Further optimizations

Another way to gain more speed is handcrafting a source to the target platform – this is often underestimated. Making a run-time profile from the source code on a PC revealed that certain helper functions like copying points from on location in memory to another are heavily used by point multiplication or addition.

- All smaller loops within the identified functions were unrolled to avoid expensive loop checks and jumps.
- All invariant computations inside the loop, i.e. computations that do not change per cycle at all, were moved outside the loop.
- Special compiler directives that optimize code at the cost of compile time were turned on. These directives are `-fexpensive-optimizations`, `-fthread--jumps` and `-O3`.
- Finally all of the helper functions were *completely* inlined into the code so that costly functions calls and involved stack operations are omitted.

All this together instantly halved execution times. This comes with the cost of a larger ROM image as seen in table 2.

5 Results of implemented algorithms

This section describes implementation results of various ECC algorithms that run on our sensor hardware. The implemented algorithms were chosen because of their popularity throughout the community. ECC versions of Diffie-Hellman, El-Gamal and DSA are commonly utilized[22].

As you will see each method heavily relies mainly on costly point multiplication. Additional operations e.g. other field operations seem negligible in comparison to point multiplication.

Considering only computational overhead (e.g. point multiplication) and memory overhead might seem a little close minded. Of course there are a lot of other factors that make security mechanisms expensive. For example the results presented here do not take communication overhead into account, ie. memory consumption and any times for pure sending and receiving data over the air are not considered. We see the sending of a Diffie-Hellman key taking place instantaneously – although in reality a sending node or receiving node would have to wait for a communication to complete before continuing operations. This makes sense in anyway as our work concentrates on the implementation and analysis of a number of algorithmic primitives for sensor networks. In addition one has to send e.g. his complete signature to a verifier in either case and this does not depend on how optimized or speedy algorithm an algorithm is. So we omit communication overhead here.

All results are summarized in table 1.

Operation	Time[s]	Standard derivation/s	Estimated results as of [15][s]
Initial setup precomputation	12.96	0.01	-
Point multiplication (fixed)	6.74	0.67	$\approx$34
Key generation	6.74	0.67	$\approx$34
Point multiplication (random)	17.28	0.47	$\approx$34
ECDSA signature	6.88	0.46	$\approx$34
ECDSA verification	24.17	0.72	$\approx$68
Diffie-Hellman key exchange	17.28 (24.02)	0.57	$\approx$68
El-Gamal encryption	24.07	0.94	$\approx$68
El-Gamal decryption	17.87	0.03	$\approx$34

Table 1. Average times for different operations

Elliptic Curve Diffie-Hellman ECDH This well known algorithm from [19] is quite important in modern protocols as a key exchange and can be adopted for ECC:

Consider two parties Alice and Bob willing to exchange a common secret key without making this one known to a passive eavesdropper. Both have agreed to a common and publicly known curve K over a finite field eg. F_{2^p} as well as to a base point $P \in K$ in advance.

- Alice randomly chooses $k_A,\ 1 < k < 2^p$ and Bob accordingly $k_B,\ 1 < k < 2^p$. Now k_A is considered as Alices, k_B is Bobs private key.
- Alice computes her public key: $Q_A = k_A P$, Bob does: $Q_B = k_B P$.
- Alice sends Q_A to Bob, Bob sends Q_B to Alice.
- Alice can now compute the shared secret for her and Bob by $secret = k_A Q_B$ and Bob also by $secret = k_B Q_A$.

An eavesdropper knows only Q_A and Q_B but is not able to compute the $secret$ from that.

As you can see ECDH needs two point multiplications for each Alice and Bob. One multiplication is with a fixed base point P and the other one with the received peers public key. There are no further operations necessary. A complete Diffie-Hellman therefore takes an average time of 24.02sec. Since Alice and Bobs public keys are likely not to be changed during sensor lifetime and could be reused for key exchanges with different communications partners it is possible to precompute them offline prior to sensor deployment. In this case a complete Diffie-Hellman would take only one class 2 point multiplication with a received public key in 17.28sec.

5.1 El-Gamal

Taher ElGamal described a popular asymmetric encryption algorithm in [27] back in 1985 which relies on traditional DLP and that has been adopted to elliptic curves and the ECDLP. This algorithm allows asymmetric encryption and decryption of data using public and private keys.

In our case Alice wants to encrypt and send secret data to Bob in a way only Bob can decrypt it. Similar to Diffie-Hellman both have agreed to a curve K over F_{2^p} and a base point $P \in K$ in advance. Both chose their private key k_A, $1 < k_A < 2^p$ and k_B, $1 < k_B < 2^p$ respectively and compute their public keys $Q_B = k_B P$ and $Q_B = k_B P$.

1. The secret data m Alice wants to transfer to Bob is first of all *embedded* into a point $P_m \in K$. Alice chooses m as the x-coordinate of a new point $P_m \in K$ and computes the corresponding y-coordinate of P_m in a way that P_m holds the Weierstraß equation. This is not quite as straight forward to do, but see [22] for a more detailed description.
2. Alice picks another random number r, $1 < r < 2^p$ and computes $P_r = rP$ and $P_h = P_m + rQ_B$. Alice sends both P_h and P_r to Bob.

Obviously encryption of m uses one (fast) multiplication with base point P, one (slow) multiplication with a random point and a little bit of other operations for data embedding or point addition. Encryption takes 24.07sec on average. Simply adding both average multiplication times 6.74sec+17.28sec=24.02sec gives a remaining 24.07sec-24.02sec=0.05sec for the other operations. You can see that these operations are not time-critical.

Bob tries to decrypt m by computing $P_s = k_B P_r$ and $P_m = P_h - P_s$. He can now extract m out of P_m as this is the x-coordinate. Decryption consumes 17.87sec of time. Bob only needs one slow point multiplications and one point subtraction which done in an average of 17.87sec-17.28sec=0.59sec.

5.2 Elliptic Curve Digital Signature Algorithm

Finally the *Digital Signature Standards* (DSS) algorithm was implemented on our target ATMEGA128 platform. DSS is a standard for obtaining digital signatures from electronic documents which describes every component involved in a signature generation. Thus it specifies details of the signature algorithm *Digital Signature Algorithm* (DSA), a signature generation technique whose security is based on a problem similar to Diffie-Hellman or DLP [28] Therefore DSA can be transformed to use ECDLP [22].

This time Alice wants to generate a signature for document m on the one hand. On the other hand Bob is willing to verify Alice signature. As usual both participants agreed to a curve K over F_{2^p} as well as to a base point $P \in K$ before. Also known in advance is the total number of points n on the curve, see section 4.1.

Alice owns a secret key $k,\ 1 < k < 2^p$ and publishes here public key $Q = kP$ to Bob. To generate a signature Alice proceeds as follows:

1. She takes a hash function H and produces the hash value of her electronic document m to $e = H(m)$. As an example hash function SHA-1 can be used which can also efficiently be implemented on sensor hardware without a problem. So hashing is not to big of a problem.
2. Alice picks a random value $s,\ 1 < s < 2^p$ and calculates $R = sP$. Let the x-coordinate of R be x.
3. Now she can compute $c \equiv x \bmod n$ and $d \equiv s^{-1}(e + kc) \bmod n$ which gives her a signature $S = (c, d, R)$ for her document. She sends S together with m to Bob.

Signature generation consists of only one point multiplication of the fixed point P – thus a fast multiplication. The rest of the appertaining operations are simply large number modulo multiplications which do not cost a lot. Therefore signature generation takes only around 6.88sec. Let alone 6.75sec for multiplication, the large number modulo multiplications do only take very short 0.14sec on average case.

For verification, ie. to check whether S is a valid signature of m from Alice, Bob subsequently does the following calculations:

1. At first Bob himself hashes m to $e' = H(m)$.
2. Bob now executes the following field modulo operations: $h = d^{-1} \bmod n$, $h_1 = e'h \bmod n$ and $h_2 = ch \bmod n$. Finally these values h_i are used to generate another point $R' \in K$ by $R' = h_1P + h_2Q$.
3. The signature is assumed valid only if the x-coordinate of R' equals the x-coordinate of R. (For a proof see again [22].)

Besides some large number modulo operations ECDSA verification needs one fixed point multiplication and one random point multiplication. A complete verification step takes about 24.17sec. Adding 6.74sec for fixed point multiplication and 17.28sec for random point multiplications gives a total of 24.02sec already. As you can see costs for non point multiplication operations are again negligible.

Remark: We implemented and use the SHA-1[29] hash function. The implementation of the reference source-code delivers ≈4.4 KBit/s data throughput. The timings presented here take this into account.

5.3 Memory Consumption

Besides computing time memory consumption is an important criteria for the use in sensor networks. Table 2 gives an overview over the memory use of our complete implementation where all algorithms (ECDSA, ECDH, El-Gamal) are taken together.

Memory segment .bss holds declared but uninitialized variables, .data holds already initialized variables. Added together a total of 208 Bytes of main RAM (=0.05%) is

Type	Size
.bss RAM	164
.data RAM	44
Total RAM	208
EEPROM	0
.text ROM	75088

Table 2. Total RAM/ROM consumption in bytes

what a sensor node has to spent for ECC permanently. Although stack size is difficult to measure in general it seems not be a problem here. As there is no recursion in the code the stack is only slightly utilized for function calls.

The ATMEGA128 features 4 KByte of EEPROM which is not used in our current implementation, but can be accessed in a similar way as flash-ROM is. Therefore it is perfectly suited to store other nodes public keys or certificates or whatsoever. It can even take a sensor nodes private key, curve details and multiplication tables in case these might change from time to time.

While ROM-memory use is not quite as critical as main RAM memory it is interesting to see how much space is consumed due to the use of loop-unrolling and inlining of functions for speed optimization. A total of 73 KBytes of flash-ROM is permanently utilized for ECC operations which is about 57% of available ROM. This leaves 55 KByte for normal sensor code which is still quite a lot and should not make any problem. Microcontroller code is normally very space saving – if speed optimizations like extensive loop-unrolling or inlining like in our ECC code is omitted.

6 Conclusion

This work presents a fast and efficient implementation of elliptic curves for microcontroller based sensor networks. The key for efficiency are memory optimizations like moving data from expensive RAM to ROM as well as a precomputation of *base points* for faster execution.

This work concludes that public-key cryptography *is* possible in sensor networks. Existing security protocols detouring asymmetric primitives with complicated symmetric constructs have to be reconsidered as there is now the chance to develop new security protocols for sensor networks which might be based on more elegant hybrid techniques. Asymmetry is of course more expensive than symmetry but hybrid protocols could make use of asymmetric operations only at very few and rare occasions e.g. to exchange a symmetric session key by ECDH.

All this does not solve the trust or key bootstrapping problem in sensor networks, i.e. how to initially spread trust by (public) keys in an ad-hoc formed network without infrastructure. But future work can now tackle this problem without turning public key cryptography aside.

References

1. University of California Berkeley: Tiny OS Hardware Designs, 2004
`http://www.tinyos.net/scoop/special/hardware`
2. Hurler, B., Zitterbart, M.: A Flexible Concept to Program and Control Wireless Sensor Networks. First European Workshop on Wireless Sensor Networks, Berlin, 2004
3. Hof, H.-J., Blaß, E.-O., Fuhrmann, T., Zitterbart, M.: Design of a Secure Distributed Service Directory for Wireless Sensornetworks. First European Workshop on Wireless Sensor Networks, Berlin, 2004
4. Blaß, E.-O., Hof, H.-J., Zitterbart, M.: S-CAN: Sicheres Overlay für Sensornetze. 2. GI/ITG KuVS Fachgespräch Drahtlose Sensornetze, Karlsruhe, 2004.
5. Hof, H.-J., Blaß, E.-O., Zitterbart, M.: Secure Overlay for Service Centric Wireless Sensor Networks. First European Workshop on Security in Ad-Hoc and Sensor Networks, Heidelberg, 2004
6. Blaß, E.-O., Hof, H.-J., Hurler, B., Zitterbart, M.: Erste Erfahrungen mit der Karlsruher Sensornetz-Plattform. 1. GI/ITG KuVS Fachgespräch Drahtlose Sensornetze, Berlin, 2003
7. Freie Universität Berlin: ScatterWeb, 2004
`http://www.scatterweb.net/`
8. Intel Research: Development of an enhanced universal embedded node platform for wireless sensor networks, 2004
`http://www.intel.com/research/exploratory/motes.htm`
9. Certicom: Press Release – Certicom Announces Elliptic Curve Cryptosystem (ECC) Challenge Winner, 1997
`http://www.certicom.com/index.php?action=company,press_archive&view=121`
10. Anderson, R., Bergadano F., Crispo, B., Lee, J., Manifavas C., Needham R.: A New Family of Authentication Protocols. ACMOSR: ACM Operating Systems Review, 1998
11. Huang Q., Cukier J., Kobayashi, H., Liu B., Zhang, J.: Fast Authenticated Key Establishment Protocols for Self-Organizing Sensor Networks. International Conference on Wireless Sensor Networks and Applications, 2003
12. Huang Q., Kobayashi, H.: Energy/security scalable mobile cryptosystem. IEEE Personal, Indoor and Mobile Radio Communications, 2003
13. Koblitz, N.: Elliptic curve cryptosystems. Mathematics of Computation, Vol. 48, 1987
14. Miller, V.S.: Use of elliptic curves in cryptography. Advances in cryptology CRYPTO 85, 1986
15. Malan D. J., Welsh, M., Smith, M. D.: A Public-Key Infrastructure for Key Distribution in TinyOS Based on Elliptic Curve Cryptography. First IEEE International Conference on Sensor and Ad Hoc Communications and Networks, 2004
16. Lorincz, K., Malan, D. J., Fulford-Jones, T. R. F., Nawoj, A., Clavel, A., Shnayder, V., Mainland, G., Moulton, S., Welsh, M.: Sensor Networks for Emergency Response: Challenges and Opportunities, IEEE Pervasive Computing, 2004
17. Lopez, J., Dahab, R.: An Overview of Elliptic Curve Cryptography. Technical Report, Institute of Computing, State University of Campinas, 2000
`http://citeseer.ist.psu.edu/lop00overview.html`
18. Lenstra, A. K., Verheul, E. R.: Selecting Cryptographic Key Sizes. Journal of Cryptology: the journal of the International Association for Cryptologic Research, 2001
19. Diffie, W., Hellman, M. E.: New Directions in Cryptography. IEEE Transactions on Information Theory, 1976
20. El Gamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. Proceedings of CRYPTO 84 on Advances in cryptology, 1985

21. Federal Information Processing Standards Publication 186: Digital Signature Standard (DSS). National Institute of Standards and Technology, 1994
22. Rosing, M.: Implementing Elliptic Curve Cryptography. Manning Publications Co., 1999
23. Menezes, A. J.: Elliptic Curve Public Key Cryptosystems. The Kluwer International Series in Engineering and Computer Science, 1993
24. Schroeppel, R., Orman, H., O'Malley, S., Spatscheck, O.: Fast Key Exchange with Elliptic Curve Systems. 15th Annual International Cryptology Conference on Advances in Cryptology, 1995
25. Koblitz, N..: CM-Curves with Good Cryptographic Properties. 11th Annual International Cryptology Conference on Advances in Cryptology, 1991
26. Solinas, J. A.: An Improved Algorithm for Arithmetic on a Family of Elliptic Curves. 17th Annual International Cryptology Conference on Advances in Cryptology, 1997
27. El Gamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. Proceedings of CRYPTO 84 on Advances in cryptology, 1985
28. National Institute of Standards and Technology: Digital Signature Standard (DSS). Federal Information Processing Standards Publication 186, 1994
29. National Institute of Standards and Technology: Secure Hash Standard. Federal Information Processing Standards Publication 180-1, 1995
30. Perrig, A., Szewczyk, R., Wen, V., Culler, D. E.,Tygar, J. D.: SPINS: security protocols for sensor netowrks. Mobile Computing and Networking, 2001
31. Andersson, R., Chan, H., Perrig, A.: Key Infection: Smart Trust for Smart Dust. IEEE International Conference on Network Protocols, 2004
32. Weimerskirch, A., Westhoff, D.: Zero Common-Knowledge Authentication for Pervasive Networks. Selected Areas in Cryptography, 2003
33. Weimerskirch, A., Westhoff, D.: Identity Certified Authentication for Ad-hoc Networks. 10th Workshop on Security of Ad Hoc and Sensor Networks, 2003
34. Balfanz, D., Smetters, D., Stewart, P., Wong, H.: Talking to strangers: Authentication in adhoc wireless networks. Symposium on Network and Distributed Systems Security, 2002
35. Liu, D., Ning, P.: Establishing Pairwise Keys in Distributed Sensor Networks. 10th Computer and Communications Security, 2003
36. Kumar, S., Girimondo, M., Weimerskirch, A., Paar, C., Patel, A., Wander, S.: Embedded End-to-End Wireless Security with ECDH Key Exchange. The 46th IEEE Midwest Symposium On Circuits and Systems, 2003.
37. Menezes, A.: Evaluation of Security Level of Cryptography: The Elliptic Curve Discrete Logarithm Problem (ECDLP). Technical Report `www.ipa.go.jp/security/enc/CRYPTREC/fy15/doc/1028_ecdlp.pdf`