Comparison of Innovative Signature Algorithms for WSNs

Benedikt Driessen
escrypt GmbH
Lise-Meitner-Allee 4
44801 Bochum, Germany
bdriessen@escrypt.com

Axel Poschmann
Horst-Görtz-Institute for IT-Security
Ruhr-University Bochum, Germany
poschmann@crypto.rub.de

Christof Paar
Horst-Görtz-Institute for IT-Security
Ruhr-University Bochum, Germany
cpaar@crypto.rub.de

ABSTRACT

For many foreseen applications of *Wireless Sensor Networks* (WSN) – for example monitoring the structural health of a bridge – message integrity is a crucial requirement. Usually, security services such as message integrity are realized by symmetric cryptography only, because asymmetric cryptography is often stated as impracticable for WSN. However, the proposed solutions for symmetric key establishment introduce a significant computation, storage, and–most important–communication overhead. Digital signatures and key-exchange based on asymmetric algorithms would be very valuable though. In the literature nearly only RSA and ECC are implemented and compared for sensor nodes, though there exist a variety of innovative asymmetric algorithms. To close this gap, we investigated the efficiency and suitability of digital signature algorithms based on innovative asymmetric primitives for WSN. We chose XTR-DSA and NTRUSIGN and implemented both (as well as ECDSA) for MICAZ motes.

Categories and Subject Descriptors

E.3 [**Data Encryption**]: Public key cryptosystems; D.2.8 [**Software Engineering**]: Metrics—*performance measures*

General Terms

Algorithms, Measurement, Performance, Security

1. INTRODUCTION

The benefits of *Wireless Sensor Networks* (WSN) are discussed widely these days. The foreseen applications range from military to vehicular scenarios. However, many of these applications process sensitive data, for example a WSN that measures the structural health of a bridge. Since adversaries can easily eavesdrop and also manipulate or inject messages, security is a crucial requirement for these applications.

Protocols based on symmetric cryptography can assure integrity, authentity, and confidentiality of messages sent by the sensor nodes very efficiently. However, one problem with symmetric solutions is the key management: neither an individual key (shared only between a sensor and the sink) nor a network-wide key (shared between all sensors) is a suitable solution. The first approach, though it offers the highest resiliency, suffers from a significant pre-distribution overhead, whereas the latter one offers a good scalability with no resiliency. Many probabilistic solutions have been proposed to deal with these issues [3, 14, 21]. All these schemes introduce a significant computation, storage, and – most important – communication overhead.

Using protocols based on asymmetric cryptography eases the establishment of keys. Furthermore, digital signatures with all their benefits can be realized by asymmetric cryptography. However, asymmetric algorithms such as RSA [26] have much longer operand lengths compared to symmetric algorithms (1024 bit vs 80 bit). This in turn results in three orders of magnitude longer processing times on a typical 8-bit micro-controller such as MICAZ compared to symmetric algorithms [10]. Consequently, despite the fact that the usage of asymmetric cryptography would solve many problems, it is often stated as impracticable for WSN.

Beside RSA exist a variety of asymmetric algorithms, such as NTRU [13], XTR [19], ECC [23], and the so called *MQ* algorithms to name just a few. Interestingly, virtually all publications dealing with asymmetric algorithms for WSN have been focusing on RSA and ECC only. To close this gap, we investigated the efficiency and suitability of digital signature algorithms based on innovative asymmetric primitives for WSN. We chose to evaluate the ECDSA [16] scheme which is quite wide spread and well examined, XTR-DSA [18, 19, 28] because of its compact signatures and comparable speed, and NTRUSIGN [12, 15] due to its simple and fast core operation, promising significant computational advantages over the two other schemes. Despite this feature, NTRUSIGN is often ignored because previous versions of this scheme have already been broken [8] and a rigorous security proof has not been made, yet.

A brief description of each schemes' core arithmetic can be found in Section 2. For further details the interested reader is referred to the original publications and [7]. Section 3 is dedicated to briefly outlining the target platform, the tools used, and the main optimizations applied to the schemes. Subsequently, our implementation results are presented and compared to other published implementation results in Section 4. Finally, in Section 5 we give our conclusions and close the paper.

2. CORE ARITHMETIC

In this section we provide a brief overview of the arithmetic used by the signature schemes. Note that ECDSA and XTR-DSA are based on the Digital Signature Algorithm (DSA) which was specified in a U.S. Government "Federal Information Processing Standard" (FIPS) called the "Digital Signature Standard" (DSS) [1]. NTRUSign is defined by a standard for financial services [24].

2.1 ECDSA

The ECDSA scheme [16] is a signature scheme based on elliptic curve cryptography (ECC). An elliptic curve E over the finite field K is defined by the simplified "Weierstrass Equation",

$$E(K): y^2 = x^3 + ax + b, \texttt{char}(K) \neq 2, 3$$

where $a, b \in K$. The points $P = (x, y) \in E(K)$ on the elliptic curve E form an abelian group, where certain rules ("group law") apply. The following basic mathematical operations are perfomed on points $P \in E(K)$:

Addition Let $P_1 = (x_1, y_1), P_2 = (x_2, y_2) \in E(K)$ be two distinct points. The sum $Q = (x_3, y_3) = P_1 + P_2$ is defined as follows:

$$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2,$$

$$y_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3) - y_1.$$

Doubling Let $P = (x_1, y_1) \in E(K)$ be a point on $E(K)$. The double $Q = (x_2, y_2) = 2P$ is derived by the following equation:

$$x_2 = \left(\frac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1,$$

$$y_2 = \left(\frac{3x_1^2 + a}{2y_1}\right)(x_1 - x_2) - y_1.$$

By using these basic operations we can construct two point multiplication methods, which are required by the ECDSA scheme.

Single (scalar) point multiplication Let $P \in E(K)$ be a point on $E(K)$, $k \in \texttt{ord}(K)$ is an integer. Single point multiplication $Q = kP$ may be performed by k additions; $Q = \underbrace{P + P + \cdots + P}_{k}$.

Simultaneous (scalar) point multiplication Let $P_1, P_2 \in E(K)$ be two distinct points on $E(K)$ where $k, l \in \texttt{ord}(K)$ are integers. Simultaneous point multiplication $Q = kP_1 + lP_2$ may be performed by applying single point multiplication to obtain $X_1 = kP_1$, $X_2 = lP_2$ and adding X_1 and X_2; $X_1 + X_2 = Q$.

Single point multiplication is performed once during the signature generation process, while the verification algorithm requires the simultaneous multiplication method.

2.2 XTR-DSA

The XTR-DSA signature scheme is based on the XTR public key scheme. XTR has been proposed by Lenstra and Verheul in 2000 [19] and is an abbreviation for "Efficient and Compact Subgroup Trace Representation" (ECSTR). XTR uses the trace $\texttt{Tr}(g) \in \mathbb{Z}_{p^2}$ of $g \in \mathbb{Z}_{p^6}$ to represent and calculate powers of elements of the order $p^2 - p + 1$ subgroup of $\mathbb{Z}^*_{p^6}$. p and q are primes where q is chosen such that $q > 6$ and $q | p^2 - p + 1$ is satisfied. XTR's main arithmetic is performed in the q order subgroup ("XTR-subgroup") of the $p^2 - p + 1$ order subgroup ("XTR-supergroup") of $\mathbb{Z}^*_{p^6}$. According to the authors of XTR, the scheme is able to achieve the same security level as RSA-1024 [26] by operating on XTR-subgroup elements with 160-170 bits size.

The arithmetic in XTR is performed on elements $c = c_1\alpha + c_2\alpha^2 \in \mathbb{Z}_{p^2}$ and an element $t \in \mathbb{Z}_p$ is represented as $-t\alpha - t\alpha^2$. We have observed that a complete XTR-DSA signature scheme can be built on top of three atomic operations which we denote by $\texttt{Op1}(\cdot)$, $\texttt{Op2}(\cdot)$, and $\texttt{Op3}(\cdot)$ in the remainder of this document. The following arithmetic in $\mathbb{Z}_{p^2}$ and $\mathbb{Z}_p$ is performed by these operations:

Op1(x,y,z,w) This operation requires four elements $w, x, y, z \in \mathbb{Z}_{p^2}$. The resulting element $c \in \mathbb{Z}_{p^2}$ is computed using two multiplications in $\mathbb{Z}_{p^2}$,

$$c = \texttt{Op1}(x, y, z, w) = xz - yz^p + w.$$

The actual arithmetic is performed by finite field operations in $\mathbb{Z}_p$. For this purpose four modular multiplications in $\mathbb{Z}_p$ are required. $c = c_1\alpha + c_2\alpha^2$ is computed by,

$$c_1 = w_1 + (z_1(y_1 - x_2 - y_2) + z_2(x_2 - x_1 + y_2)) \bmod p,$$

$$c_2 = w_2 + (z_1(x_1 - x_2 + y_1) + z_2(y_2 - x_1 - y_1)) \bmod p.$$

Op2(x) $c = \texttt{Op2}(x)$ is computed by one squaring in $\mathbb{Z}_{p^2}$,

$$c = \texttt{Op2}(x) = x^2 - 2x^p.$$

Note that computation of x^p does not require any arithmetic in $\mathbb{Z}_p$, because $x^p = x_1^p\alpha^p + x_2^p\alpha^{2p} = x_1\alpha^p + x_2\alpha$, hence x_1 and x_2 are just swapped – only two multiplications are performed instead,

$$c_1 = x_2(x_2 - 2x_1) - 2x_2 \bmod p,$$

$$c_2 = x_1(x_1 - 2x_2) - 2x_1 \bmod p.$$

Op3(x) This operation is based on $\texttt{Op2}(\cdot)$ and can be computed as

$$c = \texttt{Op3}(x) = (\texttt{Op2}(x) - x^p)x + 3 = x^3 - 3x^{p+1} + 3.$$

We compute c_1, c_2 in two steps, first we compute t_1, t_2,

$$t_1 = x_2(x_2 - 2x_1) - 3x_2 \bmod p,$$

$$t_2 = x_1(x_1 - 2x_2) - 3x_1 \bmod p.$$

From t_1, t_2 we can compute c_1, c_2 directly. By applying the ideas of Karatsuba [2] we compute $(x_1x_2), (t_1t_2)$, and $(x_1 + x_2)(t_1 + t_2)$ and therefore only need a total of five multiplications in $\mathbb{Z}_p$.

Given these operations, we can construct exponentiation algorithms.

Single exponentiation For $g \in \mathbb{Z}_{p^6}^*$ and $k \in \mathbb{Z}_q$ the single exponentiation algorithm computes $\mathtt{Tr}(g^k)$.

Double exponentiation Computing $\mathtt{Tr}(g^{bk+a})$ for two integers $a, b \in \mathbb{Z}_q$ and the "generator" $g \in \mathbb{Z}_{p^6}^*$ requires the knowledge of $c_k = \mathtt{Tr}(g^k)$, which is part of the public key.

Single exponentiation is performed once by the signature generation method, whereas double exponentiation is performed once by the verification method. For a more detailed explanation of XTR's mathematical background, the interested reader is referred to [18, 19, 28].

2.3 NTRUSign

NTRUSign is a digital signature scheme by Hoffstein *et al.* The first version of NTRUSign was published in 2003 [12]. NTRUSign is the successor of the "NTRU Signature Scheme" (NSS), which was published [13] and successfully attacked [8] in 2001. NTRUSign's main arithmetic operation is the convolution of two polynomials $f, g \in \mathbb{Z}[X]/X^N - 1$, where $h = f * g$ such that

$$h_k = \sum_{i+j \equiv k \bmod N} f_i g_j, \quad h_k, f_i, g_j \in \mathbb{Z}.$$

This operation can be performed efficiently in software, which makes NTRU-based schemes (NSS, NTRUSign) promising candidates for fast signature schemes on constrained devices such as the MICAz mote.

3. IMPLEMENTATION

We have developed modular ECDSA-, NTRUSign-, and XTR-DSA-components in NesC (`1.2.8a`). The core arithmetic was implemented in assembly specifically optimized for the MICAz platform. All implementations were optimzed for speed rather than for code size, because a shorter processing time directly reduces the energy consumption. Benchmarks have been performed using the AVRStudio development suite [5] for micro-controllers.

3.1 ECDSA

Our implementation of ECDSA is optimized for use with the "SECP160R1" curve, a parameter set standardized by "Standards for Efficient Cryptography" (SEC2) [25]. The supported curve is supposed to provide a security level equivalent to a symmetric encryption scheme with an 80-bit key. We can divide the ECDSA scheme in three layers: (**1**) finite field arithmetic (in $\mathbb{Z}_n$ or $\mathbb{Z}_p$), (**2**) ECC arithmetic (*e.g.*, point addition, point multiplication, extitetc.), and (**3**) the ECDSA protocol layer.

The finite field arithmetic on the first layer was implemented using Uhsadel's previous work on fast and highly efficient arithmetic routines in assembly [29, 30]. However, Uhsadel's routines can only be applied to calculations in $\mathbb{Z}_p$, so we had to complement them with arithmetic for modular addition, subtraction, inversion and multiplication in $\mathbb{Z}_n$, which is only required by the third layer of ECDSA. We implemented a variant of Euclid's binary extended gcd-algorithm [22, Alg. 14.62] as inversion algorithm and a combination of Comba multiplication [4] with Barrett reduction [22, Alg. 14.42] for modular multiplication.

Most of our optimizations took place on the second layer. Point addition and doubling is performed in mixed (Jacobian/affine) coordinates. We assume that the keypair changes infrequently and the curve is fixed by the parameters defined by SECP160R1. Under these assumption we can neglect the computational costs of the pre-computation phases required by the fixed-point methods we used for generating and verifying a signature. For single multiplication (*i.e.*, kP), we implemented an adaption of the "fixed-base comb method" [11, Alg. 3.44] with a single table. While optimizing our implementation of the so-called "Shamir's trick" [11, Alg. 3.48] we derived two further variants of this simultaneous scalar multiplication algorithm which computes $kP+lQ$. By pre-computing $\Gamma_{i,j} = iP + jQ$ for all $i, j \in [0, 2^w - 1]$ we can accelerate the multiplication-step. Our first variant [7, Alg. 42] is based on the observation that the original algorithm scales badly, *i.e.*, increasing the window-size w by one quadruples the size of the pre-computation table. Our modification truncates the pre-computation table by computing $\Gamma_{i',j'}$ for all $i', j' \in [0, t-1]$ with $2^{w-1} < t < 2^w$. This way, we can increase the table size in smaller steps, *e.g.*, choosing $w = 3$ and $t = 6 < 2^3$ yields a table with $6^2 = 36$ instead of $(2^3)^2 = 64$ points. Once pre-computation is complete, our multiplication algorithm starts by scanning the scalars k and l from the left with window size w, thereby obtaining integers $K, L \in [0, 2^w - 1]$. In case $K \geq t$ or $L \geq t$ we can reduce K and L by right-shifting them by one bit. This effectively shrinks the width of the window that was used to obtain K and L by one, therefore $K, L \in [0, 2^{w-1} - 1]$. Due to $t > 2^{w-1}$ we can now safely use K and L to index our pre-computation table. We also derived a second variant [7, Alg. 43] by combining the features of the fixed-point method we previously mentioned and "Shamir's trick". Let $t = \lceil \mathtt{log2}(n) \rceil$, $d = \lceil t/w \rceil$; we applied the idea of the original fixed-point method to "Shamir's trick" by pre-computing $\Gamma_{\alpha,\beta} = aP + bQ$ where

$$a = (\overbrace{0, \cdots, 0}^{d-1}, \alpha_{w-1}, \cdots \overbrace{0, \cdots, 0}^{d-1}, \alpha_1, \overbrace{0, \cdots, 0}^{d-1}, \alpha_0)_2$$

$$b = (\underbrace{0, \cdots, 0}_{d-1}, \beta_{w-1}, \cdots \underbrace{0, \cdots, 0}_{d-1}, \beta_1, \underbrace{0, \cdots, 0}_{d-1}, \beta_0)_2$$

for all $\alpha = (\alpha_{w-1}, \cdots, \alpha_0)_2, \beta = (\beta_{w-1}, \cdots, \beta_0)_2 \in [0, 2^w - 1]$. To compute $kP + lQ$ we proceed as in the single fixed-point method, with the difference that we write $k = (K^{w-1}, \cdots, K^0)_{2^d}$ and $l = (L^{w-1}, \cdots, L^0)_{2^d}$ and scan K^i, L^i simultaneously for all $i \in [0, w-1]$.

For further details, the complete algorithms (including pre-computation methods), and a performance evaluation the interested reader is referred to [7, p. 42 ff.].

3.2 XTR-DSA

Our implementation of XTR-DSA uses two primes p and q satisfying $q > 6$ and $q | p^2 - p + 1$, where

$$\lceil \mathtt{log2}(p) \rceil = 170, \quad \lceil \mathtt{log2}(q) \rceil = 160.$$

The security provided by these parameters is assumed to equal the security of our ECDSA and NTRUSign implementation [18, 19]. Similarly to the ECDSA scheme, we can divide XTR-DSA in three layers: (**1**) finite field arithmetic (in $\mathbb{Z}_p$ or $\mathbb{Z}_q$), (**2**) XTR arithmetic ($\mathtt{Op1}(\cdot)$, $\mathtt{Op2}(\cdot)$, $\mathtt{Op3}(\cdot)$, $\mathtt{Tr}(g^k)$, *etc.*), and (**3**) the XTR-DSA protocol layer.

We will now list the most important optimizations on the first layer. For arithmetic in $\mathbb{Z}_q$ we are actually re-using the algorithms from our ECDSA implementation. Barrett's reduction in combination with Comba's multiplication algorithm were implemented. Euclid's binary gcd-algorithm computes inverses in $\mathbb{Z}_q$ and $\mathbb{Z}_p$. The performance critical arithmetic in $\mathbb{Z}_p$ is performed by separate algorithms. We chose to perform calculations in Montgomery representation, therefore we can use faster Montgomery multiplication algorithms instead of regular modular multiplication. We have compared the "Finely Integrated Product Scanning" (FIPS) to the "Coarsely Integrated Operand Scanning" (CIOS) method [17] and found[1] that the FIPS method should be preferred on MICAz motes. We have implemented the FIPS multiplication method, modular addition, and subtraction in assembly, thereby significantly speeding up $\texttt{Op1}(\cdot)$, $\texttt{Op2}(\cdot)$, and $\texttt{Op3}(\cdot)$.

Our implementation of the second layer is based on a paper by Stam [28]. $\texttt{Op1}(\cdot)$, $\texttt{Op2}(\cdot)$, and $\texttt{Op3}(\cdot)$ were implemented in a straightforward manner according to the equations as defined in Section 2.2, we only had to substitute modular multiplication by Montgomery multiplication. Our final implementation uses the "XTR Single Exponentiation with Precomputation" [28, Alg. 4.3] and the improved "Simultaneous XTR Double Exponentiation" algorithm [28, Alg. 3.1]. Changes to the latter algorithms were only made in order to transform the XTR-(sub)group elements to Montgomery representation.

For more details and some additional "tricks" the interested reader is referred to [7, p. 20 ff.].

3.3 NTRUSign

The optimization of our NTRUSign implementation is based on a technical report by Silverman [27] and a paper on parameter generation by Hoffstein *et al.* [15]. From the latter we have chosen the follwing parameters,

$$N = 127, \quad d = 31, \quad q = 256, \quad \texttt{NormBound} = 122,$$

$$B = 1, \quad t = \text{"}\texttt{transpose}\text{"},$$

in combination with trinary polynomials. Our choice is estimated to provide a security level equivalent to a 80-bit symmetric key [15, Table 5]. Although implementing the keypair generation methods took most of the time, we will now focus on the signature generation and verification methods. For details on the keypair generation process, the interested reader is referred to [7, p. 60 ff.].

The "high speed multplication algorithm" presented by Silverman is based on Karatsuba's idea [2] and reduces the number of single-precision multiplications for two N-digit integers from N^2 to $\frac{3}{4}N^2$. By applying Karatsuba's idea recursively, the complexity can be reduced even more. However, due to this recursive behaviour, a straightforward implementation of Silverman's algorithm on MICAz is not advisable. We have identified two problems that lead to excessive RAM usage when the algorithm is applied recursively. (**1**) The resulting polynomial a can have up to $2N$ coefficients, in this case the relation $X^N = 1$ is applied. (**2**) The algorithm requires three intermediate polynomials $a1$, $a2$, and $a3$ (with n coefficients each) for every recursive iteration (where n is halved). Our solution to these problems results in 50% less RAM usage. We have noticed that $X^N = 1$ is only applied before Silverman's algorithm returns. We have modified the algorithm so that the calling function needs to reserve only N coefficients for the resulting polynomial a. The idea of our modification is to split the step where a is "assembled" from $a1$, $a2$, and $a3$ into two cases. In case $2n - 1 \leq N$, we assemble a straightforward according to Karatsuba's formula. In case $2n-1 > N$, we assemble a in a similar fashion, but apply $X^N = 1$ by adding the respective coefficients of $a1$, $a2$, and $a3$ directly instead of adding them according to Karatsuba's idea and applying the reduction step afterwards. By observing which coefficients of $a1$, $a2$, and $a3$ are added/subtracted and carefully sorting the order of these operations we were able to substitute the three polynomials (and hence three memory-buffers in every recursive iteration) by one polynomial, see [7, Alg. 59].

Additionally, we have re-ordered the signature generation algorithm [7, Alg. 55] and found that the previously discussed algorithm can be modified in order to compute two convolutions in parallel [7, Alg. 63], when both convolution operations share one input polynomial.

4. RESULTS

We will now shortly analyze the implemented signature schemes with respect to (**1**) energy consumption, (**2**) RAM and ROM size as estimated by the NesC compiler, (**3**) the length of signatures and keys, and (**4**) the performance (signature generation and verification) achieved by our implementation. Table 1 summarizes our results and compares them to other implementations.

We assume MICAz motes clocked at $7.37MHz$ to be used. At a supply voltage of $3V$ the MICAz mote draws a current of $12mA$ when the CPU is operating. From this we can calculate the energy consumption $E_{\texttt{CPU}}(t)$ for the timespan t (in seconds) by

$$E_{\texttt{CPU}}(t) = 12mA \cdot 3V \cdot ts = (36 \cdot t)mJ.$$

The computation energy figures $E_{\texttt{CPU}}(t_{\texttt{sign}})$ and $E_{\texttt{CPU}}(t_{\texttt{verify}})$ give the amount of energy required to perform one signing or verification operation.

Furthermore, we assume its radio device (Chipcon's CC2420) to be operated with $0dB$. From [6] we see that the current consumption for sending and receiving is $17.4mA$ and $18.8mA$, respectively. The radio device is compliant with the 802.15.4 standard (ZigBee) and has an effective data rate of $250kb/s$. Therefore we can calculate the energy consumption $E_{\texttt{s}}$ for sending and $E_{\texttt{r}}$ for receiving one bit by

$$E_{\texttt{s}} = \frac{17.4mA \cdot 3V}{250kb/s} = 209\mu J/b,$$

$$E_{\texttt{r}} = \frac{18.8mA \cdot 3V}{250kb/s} = 226\mu J/b.$$

In TinyOS by default each message consists of a $10B$ header and a $29B$ payload. ZigBee specifies a maximum packet length of $128B$. Therefore the maximum payload is theoretically $118B$. If more than $118B$ have to be transmitted the message has to be split and sent in chunks. The overhead for sending a payload of x total bytes can be calculated as $\rho(x) = 10B \cdot \lceil \frac{xB}{118B} \rceil$. The total energy $E_{\texttt{t}}(x)$ for transmitting (*i.e.*, sending and receiving) a message m of $x = |m|$

[1]in contrast to the authors' recommendation in [17]

bytes can be estimated by calculating

$$E_{\mathtt{t}}(x) = (E_{\mathtt{s}} + E_{\mathtt{r}}) \cdot (\rho(x) + x) \cdot 8\frac{b}{B}.$$

The communication energy figures $E_{\mathtt{t}}(|Sig|)$ and $E_{\mathtt{t}}(|k_{\mathtt{pub}}|)$ are listed in Table 1. It is clearly visible that the communication cost of longer public keys plays a minor role – in our scenario – compared to the computation costs. However, with larger networks there might be a point where this relation turns into the opposite.

It is not surprising that the NTRUSign module is quite small; it has only one core operation (convolution) and is therefore rather simple to implement. The ECDSA and the XTR-DSA module have a more complex arithmetic, therefore a lot more code is required. The RAM estimation seems to account for the memory statically allocated by the keypair and pre-computed values such as the big pre-computation table of our ECDSA simultaneous fixed-point method. The RAM allocated by the XTR-DSA module is mostly used to store the Montgomery representation of the public key.

The signature sizes of XTR-DSA and ECDSA are identical because a modulus of the same size is used to calculate the signature values. The size of their private key is also quite comparable. In order to accelerate XTR-DSA, we chose to include the surrounding powers of c_k in the public key, thereby drastically increasing the keysize. NTRUSign requires the largest keypair, this is due to the use of "Perturbation bases", *i.e.*, we have to store two distinct private keys. Note that we have encoded trinary polynomials in strings of $64B$, see [7, p. 64]

NTRUSign is superior to the other signature schemes when comparing signature generation and verification time. Recall that these results are achieved with a pure NesC implementation without any pre-computation. XTR-DSA and ECDSA are heavily assembly-enhanced and nearly equally fast in generating a signature, but ECDSA wins the verification benchmark. However, ECDSA's symmetric behaviour requires a long pre-computation phase, because the simultaneous fixed-point method calculates 64 points in advance. All of our implementations outperform TinyECC, while the RSA signature verification due to Gura *et al.* is faster than ECDSA and XTR-DSA.

5. CONCLUSIONS

Despite the fact that there exist a variety of asymmetric algorithms, publications dealing with implementation of public key schemes for WSN only focus on RSA and ECC. In order to close this gap we have investigated the efficiency and suitability of two innovative asymmetric algorithms for WSN – namely XTR-DSA and NTRUSign – in direct comparison to the de-facto standard for lightweight signature applications ECDSA. However, past and present attacks on the NTRU-based scheme weakened the trust into NTRUSign, it should be awaited if NTRUSign becomes stable before using it in security critical applications. Nevertheless, our implementation results show that NTRUSign requires $619ms$ to generate and $78ms$ to verify a signature. This is significantly less than any other digital signature scheme published so far. At the same time the memory requirements of $11.3kB$ ROM and $542B$ RAM are moderate making it an interesting candidate for usage in WSN.

Acknowledgement

The work presented in this paper was supported in part by the European Commission within the STREP UbiSec&Sens of the EU Framework Programme 6 for Research and Development (`www.ist-ubisecsens.org`). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the UbiSec&Sens project or the European Commission.

6. REFERENCES

[1] Digital Signature Standard (DSS). Federal Information Processing Standards (FIPS) Publication 186, National Institute of Standards and Technology (NIST), May 1994.

[2] A. A. Karatsuba and Y. Ofman. Multiplication of Many-Digital Numbers by Automatic Computers. *Soviet Physics-Doklad*, 7:595–596, 1963.

[3] H. Chan, A. Perrig, and D. Song. Random Key Predistribution Schemes for Sensor Networks. In *Proceedings of the IEEE Security and Privacy Symposium 2003*, 2003.

[4] P. G. Comba. Exponentiation Cryptosystems on the IBM PC. *IBM Systems Journal 29*, 4:526–536, 1990.

[5] Atmel Corporation. AVR Studio, 4.12.490, Service Pack 3. `http://www.atmel.com/dyn/products/tools_card.asp?tool\_id=2725`.

[6] Crossbow Technology Inc. MPR-MIB Users Manual - Revision A. available via `http://www.xbow.com/Support/Support_pdf_files/MPR-MIB_Series_Users_Manual.pdf`, June 2007.

[7] B. Driessen. Efficient Embedded Implementation of Security Solutions for ad-hoc Networks. Master's thesis, Ruhr-University Bochum, `http://www.crypto.rub.de/theses.html`, September 2007.

[8] C. Gentry, J. Jonsson, J. Stern, and M. Szydlo. Cryptanalysis of the NTRU Signature Scheme (NSS) from Eurocrypt '01. In *Advances in Cryptology – Asiacrypt 2001, Proceedings*, volume 2248, pages 123–131. Springer, 2001.

[9] N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz. Elliptic curve cryptography and RSA on 8-bit CPUs. In *Proceedings of Workshop on Crpytographic Hardware and Embedded Systems (CHES 2004), 6th International Workshop*, pages 119 – 132, 2004.

[10] N. Gura, A. Patel, A. Wander, H. Eberle, and S.C. Shantz. Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs. *Proceedings of Workshop on Cryptographic Hardware and Embedded Systems (CHES 2004), 6th International Workshop, pages 119–132*, 2004.

[11] D. Hankerson, A. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography.* Springer, 2004.

[12] J. Hoffstein, N. Howgrave-Graham, J. Pipher, J. H. Silverman, and W. Whyte. NTRUSign: Digital signatures using the NTRU lattice. In *Proceedings of the RSA conference*, 2003.

[13] J. Hoffstein, J. Pipher, and J. H. Silverman. NSS: An NTRU Lattice-Based Signature Scheme. In *Advances in Cryptology - EUROCRYPT 2001, Proceedings*, volume 2045, pages 211–228, 2001.

[14] Joengmin Hwang and Yongdae Kim. Revisiting random key pre-distribution schemes for wireless sensor networks. In *SASN '04: Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks*, pages 43–52, New York, NY, USA, 2004. ACM Press.

[15] J. Hoffstein and N. Howgrave-Graham and J. Pipher and J. H. Silverman and W. Whyte. Performance Improvements and a Baseline Parameter Generation Algorithm for NTRUSign, 2005.

[16] D. Johnson and A. Menezes. The ellipcit curve digital signature algorithm (ECDSA), 1999.

[17] C. K. Kocç, T. Acar, and B. S. Kaliski Jr. Analyzing and comparing Montgomery multiplication algorithms. *IEEE Micro*, 16(3):26–33, 1996.

[18] A. K. Lenstra and E. R. Verheul. An overview of the XTR public key system. In *The Proceedings of the Public Key Cryptography and Computational Number Theory Conference*, 2000.

[19] A. K. Lenstra and E. R. Verheul. The XTR public key system. *Lecture Notes in Computer Science*, 1880:1+, 2000.

[20] A. Liu and P. Ning. TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks. Technical Report TR-2007-36, North Carolina State University, Department of Computer Science, November 2007.

[21] D. Liu and P. Ning. Location-based Pairwise Key Establishments for Static Sensor Networks. *2003 ACM Workshop on Security in Ad Hoc and Sensor Networks (SASN '03)*, 720082, 2003.

[22] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of applied cryptography.* CRC Press, 1996. Available at `www.cacr.math.uwaterloo.ca/hac/`.

[23] V.S. Miller. Use of Elliptic Curves in Cryptography. In *Advances in Cryptology - Proceed- ings of CRYPTO'85*, pages 417–426, 1986.

[24] N/A. DSTU X9.98 Draft 9 - Lattice-Based Polynomial Public Key Establishment Algorithm for the Financial Services Industries. Technical report, Accredited Standards Commitee X9 Incorporated, 22. 2007.

[25] Certicom Research. SEC 2: Recommended Elliptic Curve Domain Parameters. Standards for Efficient Cryptography Version 1.0, 2000.

[26] R. L. Rivest, A. Shamir, and L. M. Adleman. *A method for obtaining digital signatures and public-key cryptosystems*, pages 120 – 126. 1978.

[27] J. H. Silverman. High Speed Multiplication of (Truncated) Polynomials. Technical Report 10, NTRU, 1999.

[28] M. Stam and A. K. Lenstra. Speeding up XTR. In *Advances in Cryptology – Asiacrypt 2001*, pages 125–143. Springer Verlag, 2001.

[29] L. Uhsadel. Comparison of Low-Power Public Key Cryptography on MICAz 8-bit Micro Controllers. Master's thesis, Ruhr-University Bochum, 2007.

[30] Leif Uhsadel, Axel Poschmann, and Christof Paar. Enabling Full-Size Public-Key Algorithms on 8-bit Sensor Nodes. In *Proceedings of ESAS 2007*, volume 4572 of *LNCS*, pages 73–86, 2007.

Author(s)	**Scheme**	**ROM**	**RAM**	**|Sig|**	$\|k_{\mathtt{priv}}\|$	$\|k_{\mathtt{pub}}\|$	$t_{\mathtt{sign}}$	$t_{\mathtt{verify}}$	$E_t(\|Sig\|)$	$E_t(\|k_{\mathtt{pub}}\|)$	$E_{\mathrm{CPU}}(t_{\mathtt{sign}})$	$E_{\mathrm{CPU}}(t_{\mathtt{verify}})$
This work, see also [7]	NTRUSIGN	$11.3kB$	$542B$	$127B$	$383B$	$127B$	$0.619s$	$0.078s$	$511\mu J$	$511\mu J$	$22.3mJ$	$2.81mJ$
	ECDSA	$43.2kB$	$3.2kB$	$40B$	$21B$	$40B$	$0.918s$	$0.938s$	$174\mu J$	$174\mu J$	$33.0mJ$	$33.8mJ$
	XTR-DSA	$24.3kB$	$1.6kB$	$40B$	$20B$	$176B$	$0.965s$	$2.009s$	$174\mu J$	$681\mu J$	$34,7mJ$	$72.3mJ$
Liu *et al.*, [20]	ECDSA	$19.3kB$	$1.5kB$	$40B$	$21B$	$40B$	$2.001s$	$2.436s$	$174\mu J$	$174\mu J$	$72.0mJ$	$87.7mJ$
Gura *et al.*, [9]	RSA	$7.4kB$	$1.1kB$	$128B$	$128B$	$131B$	$10.99s$	$0.43s$	$514\mu J$	$525\mu J$	$396mJ$	$15.5mJ$

Table 1: Implementation results of NTRUSign, ECDSA, and XTR-DSA for MICAz