*Wireless Sensor Network Security*
*J. Lopez and J. Zhou (Eds.)*
*IOS Press, 2008*



# Public-key Primitives

Lejla BATINA [a,1], Stefaan SEYS [a], Bart PRENEEL [a], and Ingrid VERBAUWHEDE [a]
[a] *SCD/COSIC, Dept. Electrical Engineering (ESAT), K. U. Leuven, Belgium*

**Abstract.** Wireless sensor networks consist of tiny senor nodes with limited computing and communicating capabilities and, more importantly, with limited energy resources. In this chapter we evaluate the power consumption of Public-key algorithms and investigate whether these algorithms can be used within the power constrained sensor nodes. We evaluate conventional digital signature schemes and encryption schemes, one-time signature schemes and Public-key authentication schemes.

**Keywords.** Public-key cryptography, efficient implementation, power consumption



## 1. Introduction and Motivation

The suitability of Public-key (PK) algorithms for sensor networks is an open research problem as limitations in costs, area and power are quite severe. There exists a common concept of Public-key Cryptography (PKC) being too slow and too expensive for low-cost pervasive applications and most of the protocol proposals deal only with symmetric-key cryptography.

However, the security in wireless sensor networks is becoming more and more relevant as a large number of nodes is exposed in sometimes hostile environments. The key protection is therefore of utmost importance and for that reason there is a clear advantage of using PKC.

Besides, various cryptographic services are required for these applications such as encryption, broadcast authentication, key exchange *etc.* Usual solutions to use symmetric-key algorithms such as AES and MACs are not just complicating issues such as key protection and management but can be at the same time even more expensive with respect to power and energy consumed. In addition, the use of PKC reduces power due to less protocol overhead [1].

To the best of our knowledge very few papers discuss the possibility for PKC in these applications although the benefits of PKC are evident especially for key distribution between the nodes and various authentication protocols. For example, the authentication of the base station is easily performed assuming the Public key of the base station can be stored in each node [2]. One of the reasons is that software-only approach for PKC is very expensive. There has a been a several studies showing that pure software implementations are to slow on some platforms to fulfil the security requirements [4].

[1]Corresponding Author: Lejla Batina, SCD/COSIC, Dept. Electrical Engineering (ESAT), K. U. Leuven, Belgium; E-mail: lejla.batina@esat.kuleuven.be.

However, already a small hardware module to accelerate computationally intensive finite field operations results in a huge improvement in performance of a factor of almost 2 orders of magnitude [5]. Nevertheless, one of the recent works shows that PKC for sensor nets might be possible even as a pure software solution [3].

In this chapter we describe Public-key Cryptography based solutions for security services such as encryption, key-distribution and authentication as required for wireless sensor networks. We discuss suitable protocols and the costs they imply. We restrict ourselves to existing protocols, although we believe that a substantial improvement can be obtained by revisiting and eventually redesigning the existing solutions. So, the selection of algorithms we address in this work is mainly based on previous proposals. We also argue that a custom hardware assisted approach to implement PKC in order to obtain stronger cryptography as well as to minimize the power might be the right one for lightweight applications as proposed by several publications so far [14,1,7]. We discuss and elaborate our design choices on all levels of the protocols hierarchy. Our results show feasible solution for Public-key Cryptography for these applications.

As an example we show that the Elliptic Curve Cryptography (ECC) algorithms that minimize the memory requirements and that require the fewest field operations seem to be a suitable for sensor nets applications. Furthermore, as we focus on hardware assisted approach for ECC implementations we describe a very compact arithmetic and control unit to support ECC protocols [14].

The chapter is organized as follows. In sections 2 and 3 we describe Public-key algorithms that can be used for encryption and digital signatures respectively. Next to traditional signature schemes, section 3 also includes a selection of efficient one-time signature schemes based on a general one-way function (OWF) $f$. Next we describe a number of authentication schemes in section 4. Finally, in section 5 we describe the performance of the different systems we described. This includes both a performance evaluation in software and in hardware.



## 2. Public-key Encryption Schemes

### *2.1. RSA Encryption*

The best known Public-key encryption scheme is RSA. It was invented by Rivest, Shamir and Adleman in 1978 [49]. The textbook version of the RSA encryption scheme is depicted in Alg. 1.

The security of RSA against a chosen-plaintext attack relies on the difficulty of computing the $e$-th root of a ciphertext $c$ modulo a composite integer $n$. This is known as the *RSA problem*. The difficulty of the RSA problem depends on the difficulty of the *integer factorization problem*, i.e., given an odd composite integer with at least two distinct prime factors it is hard to provide one of these prime factors. Clearly, an algorithm that solves the integer factorization problem will solve the RSA problem since this is exactly what happens in the RSA key setup process. However, the converse is still an open problem: can the integer factorization problem be hard if the RSA problem is not hard? For a rigorous security analysis of the RSA scheme and further references we refer to [39].

The naive description of the RSA encryption scheme in Alg. 1 should not be used in practice. Bellare and Rogaway [31] have proposed a provably secure way of encrypting messages using RSA or Rabin (see below), known as the OEAP scheme.

**Algorithm 1** The RSA Public-key encryption system

**Key setup**

1. Generate two large distinct random primes $p$ and $q$ such that $|p| \approx |q|$;
2. compute $n = pq$ and $\phi(n) = (p-1)(q-1)$;
3. generate a random integer $e$ such that $1 < e < \phi(n)$ and $\gcd(e, \phi(n)) = 1$;
4. use the extended Euclidean algorithm to compute the unique integer $d$ such that $1 < d < \phi$ and $ed = 1 \pmod{\phi(n)}$;
5. publish $(n, e)$ as the public key, keep $(p, q, d)$ or $(n, d)$ as the private key.

**Encryption**

Given a public key $(n, e)$, the ciphertext $c$ of message $m \in \mathbb{Z}_n^*$ is

$$c = \mathsf{E}_{(n,e)}(m) = m^e \pmod{n}.$$

**Decryption**

To decrypt the ciphertext $c$ using the secret key $(n, d)$ one computes

$$m = \mathsf{D}_{(n,d)}(c) = c^d \pmod{n}.$$

It is clear from Alg. 1 that the efficienty of the RSA algorithm depend on the selection of the parameters $p, q$ and $e$. The paramters $p$ and $q$ have to be sufficiently large in order for the algorithm to be secure (i.e., $|p| \approx |q| \geq 512$ bit). The minimum version for the public exponent $e$ is theoretically 3, but for encryption this is not deemed secure enough and usually $e = 2^{16} + 1$ is used. The size of the resulting private exponent $d$ is much larger ($|d| \approx |n|$). From this we can already conclude that the RSA encryption (public operation) is much more efficient than RSA decryption (private operation).

### *2.2. Encryption based on EC*

The security of many asymmetric cryptographic primitives (e.g., the DSA) relies on the difficulty of computing a discrete logarithm in a finite cyclic group. In elliptic curve cryptography, this group is provided by an elliptic curve $\mathcal{E}$ defined over $\mathbb{F}_q$ with $q = p^m$ and $p$ a prime number, and a definition of a method to add two points on the curve. The *elliptic curve discrete logarithm problem* can be defined as follows: given the points $P \in \mathcal{E}$ and $Q = \alpha P$ (with $\alpha$ an integer smaller than the order $P$), find the discrete logarithm $\alpha$ of $Q$.

Two well known elliptic curve based encryption schemes are the ECIES [51] (also known as the Elliptic Curve Augmented Encryption Scheme or simply the Elliptic Curve Encryption Scheme) and PSEC [51]. The security of ECIES is based on the difficulty of the *Computational Diffie-Hellman problem for elliptic curves*: Given the points $P \in \mathcal{E}$, $Q = \alpha P$ and $R = \beta P$ (with $\alpha, \beta$ integers smaller than the order $P$), compute $\alpha\beta P$. The ECIES is closely related to the Diffie-Hellman Integrated Encryption Scheme (DHIES) construction in [28]. PSEC is a family of Diffie-Hellman based encryption schemes that are all provably secure in the random oracle model. The security of each member of the

**Table 1.** Elliptic curve, symmetric primitives, RSA and discrete log in $\mathbb{F}_q^*$ key length comparison.

| Symmetric primitive key lengths | Elliptic curve key lengths | discrete log ($\mathbb{F}_q^*$) and RSA key lengths |
|---|---|---|
| 80 | 160 | 1024 |
| 112 | 224 ($\times 1.4$) | 2048 ($\times 2$) |
| 128 | 256 ($\times 1.6$) | 3072 ($\times 3$) |
| 192 | 384 ($\times 2.4$) | 7680 ($\times 7.5$) |
| 256 | 512 ($\times 3.2$) | 15360 ($\times 15$) |

family is based on a different variant of the Diffie-Hellman problem. Note that these encryption schemes have been standardized by different standardization bodies, and unfortunately the versions are not always compatible.

The advantage of elliptic curves is that they can provide the same level of security as RSA or the DSA with substantially smaller key sizes. Note that the signature sizes for both, the DSA and the ECDSA are exactly 2 times the field size. Table 1 lists elliptic curve key lengths and rough estimates of key sizes of symmetric primitives, RSA and discrete log based cryptosystems (both over $\mathbb{F}_q^*$ and elliptic curves) that provide the same level of security. These estimates were obtained from [50], and they are roughly the same as those proposed in a very detailed paper by Lenstra and Verheul [38]. Table 1 also shows how fast the key sizes grow (compared to the security of an 80-bit symmetric key). This means that in the future, the key size advantage of elliptic curve based cryptosystems will only improve.

### *2.3. NTRU encrypt*

NTRU [19] is one of the most efficient Public-key cryptosystems known and it provides encryption as well as signatures. The corresponding protocols are NTRUEncrypt and NTRUSign. The security of NTRU-based schemes depends on the difficulty of certain lattice problems. The authors of the NTRU cryptosystem recommend to use a lattice of dimension $\geq 500$, based on their own experiments [20].

The security of NTRU primitives has been an active research area for the past decade and NTRUSign has been successfully attacked as well as its improvements. On the other hand NTRUEncrypt appeared to be a secure scheme against all known attacks till very recent publication of Gama and Nguyen [26]. The authors introduced new chosen-ciphertext attacks on NTRUEncrypt that work in the presence of decryption failures. NTRU cryptosystems are now evaluated by IEEE standardization group.

The efficiency i.e. the speed and compactness are due to the fact that both protocols rely on inexpensive operations on polynomials with small coefficients. Here we focus on the encryption scheme that seems to have better chances to be adopted for applications such as smart cards, RFID tags and sensor nets due to its suitability for constrained environments as well as stronger security than NTRUSign and the successors of it.

NTRU schemes are based on arithmetic in a polynomial ring $\mathbb{Z}(x)/((x^N-1), q)$ with a parameters $(N, p, q)$ with certain properties [19]. The key operation is multiplication in the ring (here denoted as *) that can be explained as the convolution product of two vectors $a$ and $b$ written as: $a(x) = a_0 + a_1x + a_2x^2 + \ldots + a_{N-1}x^{N-1}$ and $b(x) = b_0 + b_1x + b_2x^2 + \ldots + b_{N-1}x^{N-1}$. Then the product $c(x) = a(x) * b(x) \bmod q, p$ has coefficients $c_k$ that are computed as the sum of partial products $a_ib_j$ where $i + j \equiv$

**Algorithm 2** NTRU Public-key encryption system

**Key setup**

1. Choose a random polynomial $F(x)$ from the ring $R$. $F(x)$ should have small coefficients i.e. either from the set $\{0, 1\}$ (when $p = 2$) or from $\{-1, 0, 1\}$ (when $p = 3$ or $p = x + 2$);
2. Let $f(x) = 1 + pF(x)$. (This construction for $f(x)$ is not necessary but it is recommended in order to decrease the decryption failure rate.);
3. Choose a random polynomial $g(x) \in R$ in the same way as $F(x)$ is chosen;
4. Compute the inverse of $f(x)$, so $f^{-1} \bmod q$;
5. Compute $h(x) = g(x) * f^{-1}(x) \bmod q$;
6. Publish $h(x)$ as the public key and keep $f(x)$ as the private key.

**Encryption**

1. Encode the plaintext message into a polynomial $m(x)$ with coefficients from $\{0, 1\}$ or $\{-1, 0, 1\}$;
2. Choose a random polynomial $\phi(x) \in R$ as above;
3. Compute the ciphertext as polynomial $c(x) = p\phi(x) * h(x) + m(x)(\mathrm{mod} q)$.

**Decryption**

To decrypt the ciphertext $c(x)$ using the secret key $f(x)$ one computes the message polynomial $m'(x) = c(x) * f(x) \bmod p$ and maps the coefficient of $m'$ to plaintext bits.

$k \bmod N$. The moduli for reduction are $p$ and $q$ for decryption and encryption/key setup respectively. For a detailed description as well as mathematical background we refer to [19]. Algorithm 2 shows the details of the encryption algorithm.

Considering performance numbers Gaubatz et al. [2] showed that Rabin's scheme is not a feasible solution while NTRUEncrypt can be implemented in not more than 3000 gates with an average power consumption of less than 20 $\mu W$ at a clock frequency of 500 $kHz$. These figures are obtained for the parameters that are guaranteeing 57 bits of security and they are acceptable for sensor nets applications.



## 3. Digital Signature Schemes

### *3.1. ECDSA*

A well known elliptic curve based digital signature scheme is the ECDSA [36]. The ECDSA is the elliptic curve analogue of the DSA. It was added to the DSS in 2000 (FIPS PUB 186-2). The algorithm is shown in Alg. 3.

The Digital Signature Algorithm (DSA) is well known as the service that uniquely binds a message and a sender. This problem is based on the Discrete logarithm problem (DLP) i.e. on the difficulty of computing logarithms in a large finite field. Another protocol which provides a digital signature, is the ECDSA protocol, which is performed as follows [21]:

**ECDSA protocol:**

The ECDSA is specified by an elliptic curve $E$ defined over $\mathbb{F}_q$ and a publicly known

**Algorithm 3** The ECDSA signature scheme

**Setup of system parameters**

Select an elliptic curve $\mathcal{E}$ defined over $\mathbb{F}_q$ (with $q = p^m$, where $p$ is a prime number), and a publicly known point $G \in \mathcal{E}$ of large prime order $n$.

**Key setup**

1. Select a random integer $d \in_{\mathbb{R}} [1, n[$,
2. Compute $Q = dG$,
3. Publish $Q$ as the public key, and keep $d$ as the private key.

**Signature generation**

1. Select a random integer $k \in_{\mathbb{R}} [1, n[$;
2. Compute $kG = (x_1, y_1)$;
3. Compute $r = x_1 \pmod n$, if $r = 0$ the go back to step 1;
4. Compute $s = k^{-1}(H(m)+dr) \pmod n$, where $H$ is a hash algorithm that maps $\{0,1\}^*$ to $[1, n[$; if $s = 0$ the go back to step 1;
5. the signature of message $m$ is $(r, s)$.

**Signature verification**

Given a public key $Q$, the system parameters and a message-signature pair $(m, (r, s))$, the verifier can verify the signature with the following procedure:

1. Verify that $r$ and $s$ are integers in the interval $[1, n[$;
2. Compute $w = s^{-1} \pmod n$;
3. Compute $u_1 = H(m)w \pmod n$ and $u_2 = rw \pmod n$;
4. Compute $u_1G + u_2Q = (x_0, y_0)$ and $v = x_0 \pmod n$;
5. $\mathsf{Verify}_Q(m, (r, s)) = \mathsf{true}$ if $v = r$.



point $G \in E$ of prime order $n$. A private key of Alice is a scalar $x$ and the corresponding public key is $Q = xG \in E$. The ECDSA requires a hash function which is a computationally efficient function mapping binary strings of arbitrary length to binary strings of some fixed length, called hash-values as defined in Chapter 1.

### *3.2. RSA, Rabin and DSA Signatures*

#### *3.2.1. RSA Signature Scheme*

The key setup of the RSA signature scheme is the same as the key setup of the RSA encryption scheme. The signature of message $m \in \mathbb{Z}_n^*$ is $s = m^d \pmod n$. Given a public key $(n, e)$ and a message-signature pair $(m, s)$, a signature is valid if $m = s^e \pmod n$.

It is easy to see that (for the textbook version) the RSA signing procedure is the same as the RSA decryption procedure, and signature verification is the same as encryption. Without additional measures, it is straightforward to forge a signature (i.e., generate a valid signature without knowledge of the private key on a message that has not been signed by the owner of the private key). Suppose an adversary has obtained

two valid message-signature pairs $(m_1, s_1)$ and $(m_2, s_2)$. By multiplying the signatures she gets a valid signature on the product of the two messages: $s_1 \times s_2 \pmod n = m_1^d \times m_2^d \pmod n = (m_1 \times m_2)^d \pmod n$. Note that the adversary has no option (next to multiplying different message pairs) to manipulate the message on which he obtains a signature. This and other methods of forgery are known as existential forgeries.

A usual method of detecting existential forgeries is to add recognizable redundancy to the message to be signed, which permits a verifier the correct "format" of the signed message. The most common method for adding recognizable information to a message is to apply a cryptographic hash function before applying the signature algorithm. Usually, this hash computation (and other measures such as padding) are an integral part of a digital signature scheme. Bellare and Rogaway [32] have presented a provably secure way of creating signatures with RSA and Rabin (see below), known as the PSS. The proof of security for PSS relies on the random oracle model, in which hash functions are modeled as being truly random functions. In contrast, the method for creating digital signatures with RSA that is described in PKCS #1 [45] has not been proven secure, even if the underlying RSA primitive is secure. PSS-R is a message recovery variant of PSS with the same provable security.

#### *3.2.2. DSA Signature Scheme*

The DSA is part of the DSS which was first announced in 1991 by NIST and published in 1994 (FIPS PUB 186). The security of the DSA is based on the difficulty of computing a discrete logarithm in the finite cyclic group $\mathbb{F}_q^*$ with $q$ prime. This is called the *discrete logarithm problem*: Given the integers $x$ and $y = x^\alpha \in \mathbb{F}_q^*$ (with $x$ a generating element of the group, and $\alpha < q - 1$), find the discrete logarithm $\alpha$ of $y$. As the best algorithms known for the integer factorization and discrete log problems have the same expected running times [40], the required key sizes for RSA and the DSA are the same. The DSA algorithm is the same as the ECDSA algorithm (Alg. 3), but uses the finite cyclic group $\mathbb{F}_q^*$ instead of the group provided by an elliptic curve.

### *3.3. One-time signature schemes*

#### *3.3.1. Introduction*

One-time signatures have been known since the late 1970s. They were introduced by Diffie and Hellman [34], Lamport [37] and Rabin [47]; but they are usually known in the form presented by Merkle [42,43]. These schemes are based on one-way function, rather than on trapdoor functions that are used in traditional schemes such as RSA and the DSA.

In its basic form, the Lamport-Diffie scheme can be used to sign a single bit of data. The secret key consists of two random values $x_0$ and $x_1$, while the public key is obtained by applying the OWF $f$ to the secret values, resulting in the pair $\{f(x_0), f(x_1)\}$. The signature for bit $b$ is $x_b$. The security of this scheme relies on the one-wayness of the function $f$, i.e., given the public key, it is impossible to compute the private key (and thus forge a signature) without breaking the one-way property of $f$. It is also clear that a public/private key pair can only be used once since the signature is equal to part of the private key. To sign longer messages, several instances of this scheme are used. In order to sign an $s$-bit message one requires $2s$ public key values and $2s$ private key values. A signature consists of $s$ values.

**Algorithm 4** Lamport-Diffie signature scheme with Merkle improvement (LDM)

**Setup of system parameters**

Select a OWF $f$ mapping $\{0,1\}^l$ to $\{0,1\}^l$. Let $s$ be the fixed length of the messages to be signed ($s = |m|$).

**Key setup**

1. Generate $t = s + \lceil \log_2(s) \rceil$ random values $x_1, x_2, \ldots, x_t$ with $|x_i| = l$,
2. let $sk = \{x_1, x_2, \ldots, x_t\}$,
3. compute $pk = \{f(x_1), f(x_2), \ldots\ f(x_t)\}$,
4. publish $pk$ as the one-time public key, keep $sk$ as the one-time private key.

**Signature generation**

Let $b_i$ be the $i$-th bit of $\langle m, w \rangle$ with $w$ the Hamming weight of the message $m$. The signature of message $m$ is

$$\sigma = \mathsf{Sign}_{sk}(m) = \{\text{all } x_i \text{ for which } b_i = 1\}\,.$$

**Signature verification**

Given a public key $pk = \{v_1, v_2, \ldots, v_t\}$ and a message-signature pair $(m, \sigma)$, the verifier can verify the signature with the following procedure ($b_i$ is the $i$-th bit of $\langle m, w \rangle$):

$$\mathsf{Verify}_{pk}(m, \sigma) = \mathsf{true} \text{ if } f(\sigma_\alpha) = v_i \text{ for all } i \text{ where } b_i = 1\,.$$

($\sigma_\alpha$ indicates the corresponding value in the signature, e.g., $\alpha = 3$ for the 3rd '1' bit in $\langle m, w \rangle$.)



### *3.3.2. Lamport-Diffie scheme with Merkle improvement*

Merkle [41] proposed an improvement that allows to reduce the key sizes by a factor of two and the signature size by almost a factor of two. Instead of generating two private key values for every bit in the message, Merkle suggests to only generate one value. The public key values are still obtained by applying the function $f$ to the private key values. Now, for every bit in the message that is '1', include the corresponding private key value in the signature; for every bit that is '0', omit the corresponding private key value. On average, this results in a signature that contains $s/2$ private key values. Obviously, the verifier can always claim not to have received a particular private key value, and therefore pretend that some of the '1' bits in the message that was signed were actually '0' bits. This can be remedied by adding the Hamming weight of the message before signing it. This count requires $\log_2(s)$ bits. We will refer to this scheme as the Lamport-Diffie one-time signature scheme with Merkle improvement (LDM) (Alg. 4).

### *3.3.3. Lamport-Diffie scheme with Winternitz improvement*

In [43] Merkle proposes a different variant of the Lamport-Diffie scheme, attributed by Merkle to Winternitz. This scheme reduces the size of the signature at the cost of additional computations. Instead of applying the OWF $f$ once to the private key to obtain the public key, the function $f$ is applied iteratively a fixed number of times. With ev-

ery resulting public/private key value pair it is possible to sign multiple bits. Briefly the scheme works as follows: To sign 4 bits with a single public/private key value pair, we apply the function $f$ 15 times ($= 2^4 - 1$), thus the public key becomes $v = f^{15}(x)$. To sign the message 1001 (9 in decimal), the signer makes $\sigma = f^{15-9}(x)$ public. Anyone can check that $f^9(\sigma) = f^9(f^{15-9}(x)) = v$, thus confirming that $f^{15-9}(x)$ was made public. No one besides the signer could have generated this value. Again extra redundancy has to be added to the signature in order to prevent people from changing the signature on 1001 into a signature on, for example 1000 (8 in decimal), by computing $f(f^{15-9}(x)) = f^{15-8}(x)$.

The complete scheme is described in Alg. 5. Note that a different mechanism is used to add the necessary redundancy to the signature. This solution reduces the signature size even further at the cost of additional computations. The redundancy in this scheme is encoded in the signature value $\sigma_0$ and requires one additional public/private key value pair $\{x_0, v_0\}$. Again we assume that the message $m$ is hashed before it is fed to the signing algorithm.

### *3.3.4. The HORS one-time signature scheme*

Reyzin and Reyzin propose a very efficient one-time signature scheme based on subset selection [48]. Their scheme builds on a construction proposed by Bos and Chaum in [33] that allows to sign $r \geq 1$ messages with a single public/private key pair.

In short, the signature scheme proposed by Bos and Chaum works as follows: the public/private key pair is generated as in the basic Lamport-Diffie scheme, i.e., the public key is obtained by applying a OWF $f$ to each of the $t$ values of the private key. The signing algorithm uses a bijective function $S$ that, on input $m$ ($0 \leq m < \binom{t}{k}$), outputs the $m$-th $k$-element subset of the set $T = \{1, 2, \ldots, t\}$. Let this subset be $\{i_1, i_2, \ldots, i_k\}$. The signature for message $m$ is $\{x_{i_1}, x_{i_2}, \ldots, x_{i_k}\}$. Because each message results in a different $k$-element subset (due to the bijective property of $S$), in order to forge a signature after obtaining a single message-signature pair, the forger will have to invert the OWF $f$ at least once (i.e., for all elements in the forged signature that are not part of the obtained signature). This makes it possible to reduce the security of this scheme to the one-wayness of the function $f$.

Reyzin and Reyzin propose to replace the subset selection function $S$ by a cryptographic hash function $H$. The hash value $h = H(m)$ is split into $k$ parts of equal length. Every part is interpreted as an integer and the collection of all these integers is the subset that will be used to select the private key values to be included in the signature. Reyzin and Reyzin have called this scheme HORS for "Hash to Obtain Random Subset"; this algorithm is described in Alg. 6.

If we assume that the hash function $H$ behaves like a random oracle [30], and that the adversary obtained signatures on $r$ random messages using the same private key. The probability that an adversary is able to forge a signature on a new message (without inverting the OWF $f$) is at most $(rk/t)^k$. This is the probability that after $rk$ elements of $sk$ have been made public, $k$ elements are chosen (at random) that are a subset of them.

*Security level of HORS* The security level of the HORS signature scheme in combination with the hash function $H$, is defined as $\Sigma = -\log_2(P)$. Here, $P$ is the probability of breaking the $(r + 1, k)$ subset resilience of the hash function $H$, assuming that $H$ behaves like a random oracle. This probability $P$ is at most $(rk/t)^k$ .

**Algorithm 5** Lamport-Diffie one-time signature scheme with Winternitz improvement (LDW)

**Setup of system parameters**

Select a OWF $f$ mapping $\{0,1\}^l$ to $\{0,1\}^l$. Let $s$ be the fixed length of the messages to be signed ($s = |m|$). Select the system parameter $g$ such that $g|s$.

**Key setup**

1. Generate $s/g + 1$ random values $x_0, x_1, \ldots, x_{s/g}$ with $|x_i| = l$,
2. let $sk = \{x_0, x_1, \ldots, x_{s/g}\}$,
3. compute $pk = \{f^{(2^g-1)s/g}(x_0), f^{2^g-1}(x_1), \ldots\ f^{2^g-1}(x_{s/g})\}$,
4. publish $pk$ as the one-time public key, keep $sk$ as the one-time private key.

**Signature generation**

1. Split the message $m$ into $s/g$ parts, let these parts be $m_1, m_2, \ldots, m_{s/g}$,
2. interpret each $m_j$ as an integer $I_j$,

The signature of message $m$ is

$$\sigma = \mathsf{Sign}_{sk}(m) = \{\sigma_0, \ldots, \sigma_{s/g}\}$$

$$\text{with } \begin{cases} \sigma_i = F^{2^g-1-I_i}(x_i) \text{ for } 1 \geq i \geq s/g \\ \sigma_0 = F^{\delta}(x_0) \text{ with } \delta = \sum_{i>0} I_i\,. \end{cases}$$

**Signature verification**

Given a public key $pk = \{v_0, v_1, \ldots, v_{s/g}\}$ and a message-signature pair $(m, \sigma)$, the verifier can verify the signature with the following procedure:

1. Split the message $m$ into $s/g$ parts, let these parts be $m_1, m_2, \ldots, m_{s/g}$,
2. interpret each $m_j$ as an integer $I_j$,
3. verify the validity of the signature using

$$\mathsf{Verify}_{pk}(m, \sigma) = \mathsf{true} \text{ if } \begin{cases} v_i = F^{I_i}(\sigma_i) \text{ for } 1 \geq i \geq s/g \\ v_0 = F^{2^g-1-\delta}(\sigma_0) \text{with } \delta = \sum_{i>0} I_i \end{cases}$$

The parameters $k$, $t$ and $s$ cannot be chosen independently, but have to satisfy $s = k \log_2(t)$. Using this, the probability can be rewritten as $2^{-\Sigma}$ with

$$\Sigma = k\big(s/k - \log_2(k) - \log_2(r)\big)\,. \tag{1}$$

As an example, for $s = 160$, $k = 16$ and $r = 1$ (and $t = 1024$), the security level is 96; for $r = 4$ the security level drops to 64. Using Eq. 1 we can compute the number $r$ of signatures we can generate per public key, while maintaining a security level $\bar{\Sigma}$:

**Algorithm 6** The HORS one-time signature scheme

**Setup of system parameters**

Select a cryptographic hash function $H$ with output length $|H| = s$ and a OWF $f$ mapping $\{0,1\}^l$ to $\{0,1\}^l$. Select the system parameters $k$ and $t$ such that $k \log_2(t) = s$.

**Key setup**

1. Generate $t$ random values $x_1, x_2, \ldots, x_t$ with $|x_i| = l$,
2. let $sk = \{x_1, x_2, \ldots, x_t\}$,
3. compute $pk = \{f(x_1), f(x_2), \ldots, f(x_t)\}$,
4. publish $pk$ as the one-time public key, keep $sk$ as the one-time private key.

**Signature generation**

1. Let $h = H(m)$,
2. split $h$ into $k$ substrings $h_1, h_2, \ldots, h_k$ of length $|h_i| = \log_2(t)$,
3. interpret each $h_j$ as an integer $I_j$.

The signature of message $m$ is

$$\sigma = \mathsf{Sign}_{sk}(m) = \{x_{I_1}, x_{I_2}, \ldots, x_{I_k}\}.$$

**Signature verification**

Given a public key $pk$ and a message-signature pair $(m, \sigma)$, the verifier can verify the signature with the following procedure:

1. Let $h = H(m)$, $\sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_k\}$, and $pk = \{v_1, v_2, \ldots, v_t\}$,
2. split $h$ into $k$ substrings $h_1, h_2, \ldots, h_k$ of length $|h_i| = \log_2(t)$,
3. interpret each $h_j$ as an integer $I_j$,
4. verify the validity of the signature using

$$\mathsf{Verify}_{pk}(m, \sigma) = \mathsf{true} \text{ if } f(\sigma_j) = v_{I_j} \text{for } 1 \leq j \leq k\,.$$



$$r = (1/k)2^{(s-\bar{\Sigma})/k}\,. \tag{2}$$

Finally, from Eq. (2) we can compute how many public keys we need to sign $S$ messages:

$$\#pk = S/r = Sk2^{(\bar{\Sigma}-s)/k}\,. \tag{3}$$

As a practical example we will use two different parameter sets with roughly the same security level: HORS-20 with $(s, k, t, r) = (160, 20, 256, 1)$, providing $\Sigma = 73.5$, and HORS-18 with $(s, k, t, r) = (162, 18, 512, 2)$, providing $\Sigma = 68.9$.

### 3.4. One-time Public-key authentication

Two obvious disadvantages of one-time signature schemes are the size of the public key and the fact that it can only be used a limited number of times. All possible verifiers need *authenticated* copies of these public keys, i.e., they need evidence that a particular public key is related to a particular user . For example, when using LDM with $s = 160$ and $l = 80$, the total size of 1000 public keys is about 1.63 $\mathrm{MBytes}$. One obvious mechanism to provide authenticated copies of this public key set is to transfer the complete set to every verifier over some authenticated channel (e.g., using a traditional digital signature such as ECDSA). The disadvantage is that every verifier has to store 1.63 $\mathrm{Mbytes}$ of data for every potential signer. Fortunately, there are more efficient solutions.

#### 3.4.1. Merkle trees

Merkle proposed the use of binary trees to reduce the authentication of a large number of public keys to the authentication of a single value, i.e., the root of the tree [43].

Due to space limitations, we will limit the description of Merkle trees to a few key concepts, for more details we refer to [43].

A Merkle tree is a complete binary tree with an $l$-bit value associated with each node such that each interior node value is a OWF of the node values of its children. The $N$ values that need to be authenticated are placed at the leaves of the tree. Usually these values need to be kept secret. In that case, the hash values of these secret values are placed at the leaves of the tree. The hash values $leaf_i = H(\mathsf{value})$ are called the leaves and the secret values are usually called the leaf pre-images. Every parent's node value is calculated as a OWF $f$ of the concatenation of the two child values. Because of this construction the root of the tree can be used to authenticate the leaves. A particular leaf can be authenticated with respect to the root value and the *authentication path* of the leaf.

*Authentication paths.* Let $sib_i$ be the value of the sibling of the node on height $i$ on the path from the leaf to the root. A leaf has height 0, the OWF of two leaves has height 1, etc., and the root has height $H$ if the tree has $2^H$ leaves. The authentication path is then the set $\{sib_i \mid 0 \leq i \leq H\}$.

The goal of *Merkle tree traversal* is the sequential output of the leaf values and their authentication paths. In [35], Jakobsson et al. present an algorithm which allows a time-space trade-off. When storage is minimized, the algorithm requires about $2\log_2(N)/\log_2(\log_2(N))$ invocations of $f$, and a maximum storage space of $1.5\ \log_2^2(N)/\log_2(\log_2(N))$ outputs of $f$. The Merkle tree traversal algorithm by Jakobsson starts with the calculation of the tree root. During this root calculation, the initial internal state of the algorithm is also calculated and stored in memory. This initialization requires $N - 1$ invocations of $f$.

#### 3.4.2. One-way chains

In the three one-time signature schemes we have described, the public key is computed by applying the OWF $f$ one or more times to the private key, which in turn is nothing more than a set of random values. Another way of authenticating these public keys is using one-way chains. Perrig suggests in [44] the use of these one-way chains to authenticate the public keys that are used in the BiBa (Bins and Balls) one-time signature scheme, but this idea applies equally to other signature schemes. This authentica-

tion mechanism is especially useful in the setting of a single verifier, or a set of "synchronized" verifiers (i.e., verifiers who all receive the same message-signature pairs at the same time). A typical example of the latter is *broadcast authentication* in wireless networks. All nodes within range of a sending node $A$ have an authenticated copy of the root of the one-way chain used by $A$. Node $A$ signs every message it broadcasts with a one-time signature scheme; this signature is verified by all receiving nodes at the same time. For the LDM and the LDW a new public key becomes active after every signature, for the HORS scheme this happens after $r$ signatures. Note that the private key that was used for signature $i$ becomes the public key for signature $i + 1$.

The private key of the three one-time signature schemes we have described is always a set of random values. When using one-time chains for public key authentication, the signer first generates the root private key (i.e., a single set of random values). He then applies the OWF $f$ as many times as required (e.g., for LDM and HORS, the public key is obtained by applying $f$ a single time to the private key; this means that we obtain $n$ public/private key pairs by applying the function OWF $n$ successive times).

The last set of values we obtain is the *first* public key that will be used. This public key is transferred in an authenticated way to all the potential verifiers. The second to last set of values we obtain is the first private key that will be used. Once this private key has been used, it becomes the public key to be used for the next message, etc., until the initial private key is used. At that time, a fresh chain has to be generated and the first public key of this chain has to be transferred to the verifiers.

Note that longer chains are only disadvantageous for the signer, and not for the verifiers. The computational effort for signature generation grows for longer chains since the signer always has to start from the start of the chains. The signer could improve this by storing multiple intermediate private keys in memory. This technique provides a means to exchange storage requirements for computation time. For verifiers the length of the chains has no influence on the performance.



## 4. Authenticated Key Establishment

### *4.1. Diffie-Hellman*

As already known, key management and exchange do not scale well in the case of large number of users as the case is for sensor networks. Also the protection of keys is easily solved by use of PKC. The cost for key exchange in the case of EC operations is one point multiplication (on each side).

We stress here that all mentioned protocols include other cryptographic primitives as well, such as hash functions, MACs *etc.* but for the total cost of a protocol in the case of ECC point multiplication operation is the most substantial. Therefore, for the results we take only number of point multiplications into account.

The **Diffie-Hellman key agreement** protocol gives the first practical solution to the key exchange problem for two parties. The basic version of this protocol is as follows [22]:

**Algorithm 7** Diffie-Hellman key agreement

**One time setup**

1. A prime $p$ such that $(p-1)/2 = p'$, where $p'$ is also a prime and a generator $\alpha$ of $\mathbb{Z}_p^*$, $2 \leq \alpha \leq p-2$ are selected and published;

**Protocol messages**

$A \rightarrow B : \alpha^x \bmod p$ (1)
$B \rightarrow A : \alpha^y \bmod p$ (2)

**Protocol actions**

1. $A$ chooses a random secret $x$, $1 \leq x \leq p-2$ and sends $B$ message (1).
2. $B$ chooses a random secret $y$, $1 \leq y \leq p-2$ and sends $A$ message (2).
3. $B$ receives $\alpha^x$ and computes the shared key as $K = (\alpha^x)^y \bmod p$.
4. $A$ receives $\alpha^y$ and computes the shared key as $K = (\alpha^y)^x \bmod p$.

### *4.2. Protocols for Authentication*

Sensor node identification requires authentication as a cryptographic service. This property can be achieved by symmetric as well as asymmetric primitives. Previously known work considered mainly symmetric-key algorithms e.g. AES [25].

Here we discuss two authentication protocols. They are both written for the case of ECC because we look into performance of ECC in more detail in the last section. The anti-counterfeiting problem can also be rephrased as an authentication problem. This was explored in [18] for RFIDs.

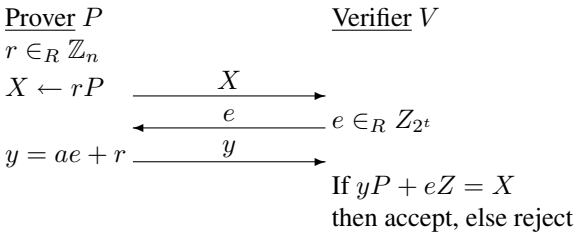
Other protocols found in the literature include Beth's identification protocol [23] and the XDL-IBI scheme in [24]. Beth's protocol only requires one point multiplication but it remains an open problem to prove its security against active adversaries. The XDL-IBI scheme also requires only one point multiplication but is only secure against passive adversaries and concurrent attacks (under a modified assumption). Thus, it seems that by analyzing both Schnorr's and Okamoto's we cover the efficiency of all *available* ID protocols based on the hardness of the DL problem.

#### *4.2.1. Schnorr Identification Protocol based on ECDLP*

This protocol is only secure against passive attacks but it is very efficient as it requires just one ECC point multiplication (for a node).

Here we specify the Schnorr identification protocol [22] based on ECDLP. In this case a node proves its identity to a station in a 3-pass protocol.

1. **Common Input:** The set of system parameters in this case consists of: ($q$, $a$, $b$, $P$, $n$, $h$). Here, $q$ specifies the finite field, $a$, $b$, define an elliptic curve, $P$ is a point on the curve of order $n$ and $h$ is the cofactor. In this case of tag authentication, most of these parameters are assumed to be fixed.
2. **Prover-Tag Input:** The prover's secret $a$ such that $Z = -aP$.
3. **Protocol:** The protocol involves exchange of the following messages:



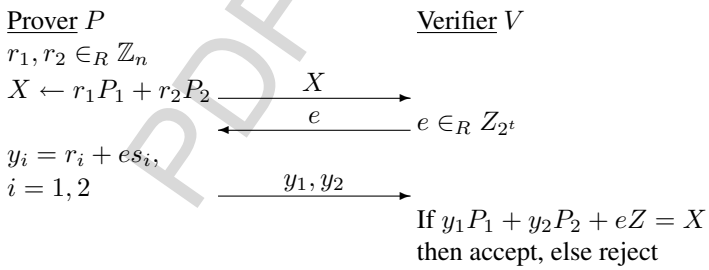

More precisely, the steps of the protocol are:

- *Commitment by a Prover-Tag:* The tag picks $r \in_R \{0, \ldots, n-1\}$, and sends $X = rP$ to the reader.
- *Challenge from a Verifier-Reader:* The reader picks a number $e \in [1, 2^t]$ and sends it to the tag.
- *Response from a Tag:* The tag computes $y = ae + r$ and sends it to the reader.
- The verifier checks that $yP+eZ$ equals $X$. Check: $yP+eZ = (ae+r)P+eZ = aeP + rP + (-eaP) = rP = X$

### 4.2.2. The Okamoto Identification Protocol based on ECDLP

Another option for secure identification is Okamoto's identification protocol. We are considering Okamoto's identification protocol as it provides security against active adversaries and it is based on the hardness of the DL problem.

1. **Common Input:** Common input is the set of system parameters consisting of ($q$, $a$, $b$, $P_1$, $P_2$, $n$, $h$) as before.
2. **Prover-Tag Input:** The prover's secret $(s_1, s_2)$ such that $Z = -s_1P_1 - s_2P_2$.
3. **Protocol:** The protocol involves the exchange of the following messages:




More precisely, the steps of the protocol are:

- *Commitment by a Prover-Tag:* The tag picks $r_i \in_R \{0, \ldots, n-1\}$, $i = 1, 2$ and sends $X = r_1P_1 + r_2P_2$ to the reader.
- *Challenge from a Verifier-Reader:* The reader picks a number $e \in [1, 2^t]$ and sends it to the tag.
- *Response from a Tag:* The tag computes $y_i = r_i + es_i$, $i = 1, 2$ and sends those values to the reader.
- The verifier checks that $y_1P_1 + y_2P_2 + eZ$ equals $X$.
  Check: $y_1P_1+y_2P_2+eZ = (r_1+es_1)P_1+(r_2+es_2)P+e(-s_1P_1-s_2P_2)Z = r_1P_1 + r_2P_2 = X$

**Algorithm 8** Simultaneous point multiplication

**Require:** $k = (k_{t-1}, \ldots, k_0)_2$, $l = (l_{t-1}, \ldots, l_0)_2$, $P, Q$ points on the curve
**Ensure:** $R = kP + lQ$
1: Compute $P + Q$
2: $R \leftarrow \infty$
3: **for** *i* from $t - 1$ downto 0 **do**
4: $R \leftarrow 2R$
5: $R \leftarrow R + (k_i P + l_i Q)$
6: **end for**
7: Return(R)

In [18], the *feasibility* of the ECC version of Schnorr's identification protocol in an RFID system was investigated and area and latency estimates were provided.

As can be seen from Okamoto's scheme, the required computation for a node is of a form $kP + lQ$ *i.e.* so-called multiple-point multiplication. For the purpose of speeding-up this computation one uses Shamir's trick [9]. The scalars $k$ and $l$ are stored in a 2-row matrix in which each row contains binary representation of one of the scalars. All values of the form $iP + jQ$, $0 \leq i, j < 2^w$ are precalculated and stored where $w$ is given width of the window. The algorithm to perform this so-called simultaneous point multiplication is computing at each of $\lceil \frac{t}{w} \rceil$ steps $w$ doublings and 1 addition from the list of the precalculated values of the form $iP+jQ$. As a width of the window $w$ is a variable that allows some trade-off, we chose the smallest window *i.e.* $w = 1$. In this way, the memory requirements are minimized as only 3 points have to be stored: $P, Q, P + Q$. The exact computation is given in Algorithm 8 [9]. The expected running time of the algorithm for $w = 1$ is $\frac{3}{4}t$ point additions and $(t - 1)$ point doublings.

## 5. Performance Evaluation

### 5.1. Performance of RSA, DSA and ECDSA

Table 2 shows power consumption measurements on a 32-bit Intel StrongARM microprocessors by Potlapally *et al.* [46]. Next to the performance of the AES, the performance of SHA-1, the DSA, and the public key algorithms RSA, DSA and ECDSA is presented. These measurements clearly show the performance gap between symmetric and asymmetric techniques.

### 5.2. Efficiency of one-time signature schemes

All the one-time signature and Public-key authentication schemes evaluated in this chapter are based on a general OWF $f$. In order to be able to provide algebraic expressions for the cost (= power requirement) of these schemes we assume that:

- the input size of $f$ is a multiple of $l$ bits; we will refer to a group of $l$ bits as one "block";
- the output size of $f$ is 1 block;
- the cost of $f$ for an input size of $t$ blocks is $t$ BF (for Block Function), i.e., the cost of $f$ grows linearly with respect to the input size.

**Table 2.** Power consumptions of SHA-1, AES, RSA, DSA and ECDSA on a 32-bit Intel StrongARM SA1100 @ 206MHz [46].

| Operation | $\|\text{key}\|$ $\|\text{hash}\|$ | Power consumption | |
|---|---|---|---|
| SHA-1 | 160 | 0.76 $\mu$J/Byte | |
| AES key scheduling | 128 | 7.83 $\mu$J | |
| AES encryption | 128 | 1.21 $\mu$J/Byte | |
| | | Verification | Signing |
| RSA | 1024 | 15.97 mJ | 546.5 mJ |
| DSA | 1024 | 338.02 mJ | 313.6 mJ |
| ECDSA-$\mathbb{F}_{2^{131}}$ | 163 | 196.2 mJ | 134.2 mJ |

Efficient instances of the OWF $f$ can be built from fast block ciphers or cryptographic hash functions. We assume that $f$ maps $n \times 80$ bits to 80 bits, i.e., $l = 80$. Since collision resistance is not required from $f$ we believe that this parameter is sufficient.

We further assume that a cryptographic hash function $H$ is applied to all messages before they are fed to the LDM or LDW signature scheme. The output size of this hash is $|H| = s = 160$ bits. We assume that this is the same hash function that is used in the HORS scheme. As this hash function has to be applied for all three schemes, the cost of this operation is not taken into account for the efficiency evaluation.

In order to compare the efficiency of the one-time signature schemes with elliptic curve based signature schemes, we use the measurements from [46] that are presented in Table 2 on p. 17. Assuming one invocation of the function $f$ requires one invocation of the AES block encryption algorithm, then the cost of an ECDSA verification (signature) is equal to the cost of $10^4$ ($7 \cdot 10^3$) invocations of $f$.

Another important cost factor (certainly for one-time signature schemes) is the *communication cost*. A rigorous performance analysis of the popular Mica2 and Mica2dot motes is presented in [29]. The authors show that the effective throughput available to applications on a Mica2 mote is only 4.6 $\text{kbits/s}$ (a fraction of the nominal bandwidth of 19.2 $\text{kbits/s}$). In order to achieve this, the radio module of the mote requires 48 $\text{mW}$ in receive mode and 54 $\text{mW}$ in transmit mode. Thus, the mote uses 10.4 $\mu\text{J/bit}$ in receive mode and 11.7 $\mu\text{J/bit}$ in transmit mode. This results in the following assumptions we will use for the numeric evaluation:

- The size of the output of the hash function $H$ is 160 bits ($|H| = s = 160$),
- 1 block = 80 bits,
- 1 BF = $16 \times 1.21$ $\mu$J $= 19.36$ $\mu$J,
- the transmission cost of 1 block = 936 $\mu$J,
- the receiving cost of 1 block = 832 $\mu$J.

### 5.2.1. *Efficiency of the LDM*

Looking at Alg. 4, we see that the key setup requires $s + \lceil \log_2(s) \rceil$ BF. The public and private key size is $s + \lceil \log_2(s) \rceil$ blocks. Signature generation is "free". Assuming a uniform distribution of the possible messages in the message space, on average the message and padded redundancy $\langle m, w \rangle$ will contain 50% zeros and 50% ones. This means that the average signature size is $\frac{1}{2}(s + \lceil \log_2(s) \rceil)$ blocks and that verification requires $\frac{1}{2}(s + \lceil \log_2(s) \rceil)$ BF on average.

Note that the secret key $sk = \{x_0, \ldots, x_{m/t}\}$ can be generated with a good pseudo-random generator using a single seed $\underline{sk}$. The entropy of the output of the pseudo-random generator is at most $|\underline{sk}|$, therefore we propose to use a seed with size $2|x_i|$, i.e., 2 blocks. This means that storing the secret key only requires a fraction of the total size of the private key. Obviously this is not true for the public key.

As a practical example, the total cost of signing a message, transmitting this message to the verifier and verifying the message is 148 mJ for the communications and 1.6 mJ for the computations, totalling about 150 mJ.

### 5.2.2. *Efficiency of the LDW*

Looking at Alg. 5, we see that the key setup requires $2(s/g)(2^g - 1)$ BF. The public and private key size is $s/g + 1$ blocks. The costs of signature generation and verification are both $(s/g)(2^g - 1)$ BF, independent of the message. The signature size is $s/g + 1$ blocks.

Notice that the computational cost grows exponentially with the group size $g$, while the communication cost only drops linearly with $g$. This indicates that performance gain, if any, will only be possible for small values of $g$. Table 3 shows the total cost of a signature for different values of $g$. This cost includes signature generation, one signature verification and one transmission from sender to receiver. The minimum cost occurs for $g = 4$, i.e., signing 4 bits with a single public/private key value pair. Using the LDW with $g = 4$ offers a 37% performance gain compared to the LDM.

**Table 3.** Cost of the LDW for different group sizes $g$ (mJ).

| group size $g$ | 1 | 2 | 3 | **4** | 5 | 6 |
|---|---|---|---|---|---|---|
| communications | 285 | 143 | 96 | **72** | 58 | 49 |
| computations | 6.20 | 9.29 | 14.46 | **23.23** | 38.41 | 65.05 |
| total | 291 | 153 | 111 | **96** | 97 | 114 |



### 5.2.3. *Efficiency of the HORS scheme*

We assume that the system-wide parameter $k$ is fixed, and is not included in the public/secret key. The cost of HORS can be summarized as follows:

- The *key setup* requires $t$ evaluations of $f$, resulting in a total computational cost of $t$ BF. The public and private key size is $t$ blocks.
- *Signing* requires no additional operations besides applying the hash function to the message. The signature size is $k$ blocks.
- *Verifying* requires a maximum of $k$ BF if the signature is valid (if the signature is invalid the verification process can be stopped earlier and the cost will be less).

For HORS-20 the total cost for signing a message, transmitting the signature to the verifier and verifying the message is 452.6 mJ for the communications and 387 $\mu$J for the computations, totalling about 453 mJ. For HORS-18, this becomes 905.2 mJ for the communications and 348 $\mu$J for the computations, totalling about 906 mJ. Note that the total cost is dominated by the transmission cost which is 3 orders of magnitude larger than the computation cost.

### 5.3. Efficiency of one-time signature schemes with Public-key authentication

#### 5.3.1. Efficiency of Merkle tree authentication

In order to evaluate the cost of Merkle trees we first describe the different processes involved.

- The algorithm precalc generates the leaf pre-images and requires $precost$ BF.
- The algorithm leafcalc generates the leaves and requires $leafcost$ BF. Note that this includes generation of the leaf pre-images.
- *Root generation* is the process of computing the root node of the tree. This root node will serve as the public key of the signature scheme.
- *Authentication path generation* or *Merkle tree traversal* is the task of generating the authentication paths for successive leaves.

Next to these steps we also consider the signing and verifying processes:

- *Signing* a message consists of (1) regenerating the private key, (2) computing the signature, and (3) generating the authentication path.
- *Verifying* a signature consists of (1) verifying the authenticity of the received public key (i.e., a leaf pre-image), and (2) verifying the signature.

*Authentication path generation and verification cost* The fractal Merkle tree traversal algorithm presented in [35] by Jakobsson *et al.* allows a time-space trade-off. Briefly explained, the algorithm splits the original tree into subtrees of height $h \leq H$. These subtrees are constructed in such a way that the root of one tree is at the same time a leaf of a tree above it, i.e., they are stacked on top of each other. Assuming $h|H$, let $L = H/h$ be the number of subtree *levels*. Exactly one such subtree for each level is kept in memory and all these stacked subtrees together contain the authentication path for the leaf that is being authenticated at the moment. For each output of an authentication path a second new subtree for each level is being constructed. The construction is programmed in such a way that the new subtree will be finished just in time to be used to create the authentication path; at this moment the old subtree is discarded and the construction of a fresh subtree is initiated. Fractal tree traversal requires a maximum of $2(L-1)$ evaluations of $f$ per round, and a maximum space of $2L2^{h+1} + HL/2$ memory units (each unit being the size of the output of $f$, i.e., 1 block). Note that according to our definition of $f$, one evaluation here requires 2 BF. Taking into account the cost of generating the leaves (*leafcost* BF per leaf), the computational cost of fractal tree traversal becomes $(L-1)(2 + leafcost)$ BF per round. The required space is minimized using $h = \log_2(H) = \log_2(\log_2(N))$. The authentication path generation cost then becomes

$$\frac{\log_2(N)}{\log_2(\log_2(N))}(2 + leafcost) \text{ BF per path.} \tag{4}$$

The root of the tree is computed at the initialisation phase of the fractal Merkle tree traversal algorithm. This root generation process requires the computation of the $N$ leaves and all nodes in the tree. By cleverly scheduling the order in which these nodes are computed, the memory requirements can be limited to $\log_2(N) + 1$ blocks. The total root generation cost is

$$N(2 + \mathit{leafcost}) - 2\ \mathrm{BF}. \tag{5}$$

Upon receipt of a leaf pre-image and the corresponding authentication path, the recipients have to compute nodes of the tree until they arrive at the root. The total cost of this verification process is

$$2\log_2(N) + (\mathit{leafcost} - \mathit{precost})\ \mathrm{BF\ per\ path}. \tag{6}$$

#### 5.3.2. Energy cost factors when using Merkle tree authentication

We consider the following energy consumption cost factors:

1. *Key setup cost*. The signer generates the necessary private keys and computes the root of the Merkle tree. The cost of this process is given by Eq. (5). A copy of the resulting root value is transferred to the verifiers over an authenticated channel.
2. *Cost of signing a message.* The signer has to regenerate the private/public key pair, compute the signature, and generate the authentication path for this particular public key.
3. *Cost of verifying a signature.* The verifier has to check the validity of the signature and of the public key (i.e., verifying the authentication path).
4. *Signature size.* A signature consists of the public key that was used, the signature itself, and the authentication path to authenticate the public key.

Due to space limitations, we refer to [27] for a detailed analysis and algebraic expressions of these four cost factors for every one-time signature scheme we mentioned. We refer to Sect. 5.5 for a numeric evaluation and comparison of the different energy costs of the different schemes.

### 5.4. Efficiency of one-way chain authentication

One-way chains may seem more efficient than Merkle trees, as no authentication paths need to be computed, exchanged and verified. On the other hand, the use of one-way chains requires rather strict synchronization between the signer and the verifiers, limiting the applications of this authentication mechanism.

The different processes involved are:

- *One-way chain generation* (key setup) is the process of generating all the hash chains. The output of this process is the root secret key $sk_N$ and the first public key $pk_1$ to be used by the verifiers.
- *Active private key regeneration* is the process of computing the currently active private key.
- *Signing a message* consists of (1) regenerating the active private key and (2) generating the signature.
- *Verifying a signature* consists of verifying the received signature values by walking down the chains until known values are reached (see Sect. 3.4.2).

#### 5.4.1. One-way chain generation and verification

A private key consists of $t+\gamma$ random values. For the LDM and HORS scheme $\gamma = 0$ and for the LDW $\gamma = 1$. These values are generated using a single seed value $\underline{sk}$, requiring

$t + \gamma$ BF. From this root private key, the signer computes the chains. Assuming we wish to sign $S$ messages with a single set of one-way chains and assuming that a single private key can be used $r$ times, the required length $N$ of the chains is $S/r$. The total cost of the *one-way chain generation* process is

$$(t + \gamma) + N(t\alpha + \gamma\beta) \text{ BF}. \tag{7}$$

When using one-way chains, the cost of regenerating the first private key is much higher than the cost of regenerating the last private key. This is because the signer always starts from the root private key and works his way down the chains until he reaches the active private key. Therefore, we compute the *average* cost of the private key regeneration process. In total, $N$ private keys can be verified with a single set of one-way chains. The cost of computing these $N$ private keys consists of computing the root private key and walking down the chains until the active key is reached. The total cost of this process is

$$N(t + \gamma) + \frac{N(N-1)}{2}(t\alpha + \gamma\beta) \text{ BF}.$$

Assuming the signer keeps the active private key in memory until the $r$ signatures are generated, the average cost of private key regeneration *per signature* is

$$\frac{t + \gamma}{r} + \frac{N-1}{2r}(t\alpha + \gamma\beta) \text{ BF}. \tag{8}$$

The cost of signing a message is the cost of generating the active private key and using this key to generate the signature. This last cost depends on the signature scheme that is used.

Assume that the verifiers obtained an authenticated copy of the first public key $pk_1$. Upon reception of a signature, the verifier has to walk down the chains until he reaches a known authenticated value. Once the verifier has checked the authenticity of the received signature, he only keeps the most recently used values for each chain (see Sect. 3.4.2). This way he will have to compute every value in each chain exactly once, except for the root private key and possibly some other values close to this root private key. The total signature verification cost is at most $N(t\alpha + \gamma\beta)$ BF or

$$(t\alpha + \gamma\beta)/r \text{ BF} \tag{9}$$

per signature.

### *5.4.2. Energy cost factors when using one-way chains*

We consider the following energy consumption cost factors:

1. *Key setup cost*. The signer generates the root private key and the first public key. A copy of this public key is transferred to the verifiers over an authenticated channel.
2. *Cost of signing a message.* The signer regenerates the active private key and computes the signature.

3. *Cost of verifying a signature.* The verifier verifies the signature against the public key values he has in memory. Once the the signature has been verified, he updates the public key values.
4. *Signature size.* Only the signature itself has be transferred (public keys nor authentications paths have to be transmitted).

Again we refer to [27] for a detailed analysis and algebraic expressions, and to Sect. 5.5 for a numeric evaluation and comparison of the different energy costs of the different schemes.

### 5.5. Comparison

In order to compare the efficiency of the different one-time signature schemes, including the cost of communications, we will use assumptions presented in Sect. 5.2 which are based on the measurements shown in Table 2. This table also allows us to compare the one-time signature schemes with ECDSA. We have evaluated the energy cost *per signature* when we sign $S$ messages with a single set of one-way chains or Merkle tree. The results in Fig. 1 up to Fig. 5 do not include the cost of bootstrapping the system. We assume that the verifiers have already obtained an authenticated copy of the public key material that they need. Figure 6 shows an example scenario with 10 verifiers, and includes the cost of bootstrapping the system.

Figure 1 shows how the energy cost per *signature generation* varies with the number of signatures. For Merkle based schemes, this cost includes the root generation process (Eq. (5)) next to the cost of generating a signature and the authentication path. The cost of the root generation process is divided over all $S$ signatures. Because of the efficient authentication path generation algorithm, the cost per signature only grows very slowly with the number of signatures ($\sim \log_2(S)/\log_2(\log_2(S))$). For schemes based on one-way chains, the signing cost includes the overhead of computing the first public key (i.e., the last values of the chains, Eq. (7)) next to the cost of generating the signature. In contrast to Merkle based schemes, the cost per signature grows linearly with respect to $S$. This is because of the inefficient private key generation process (Eq. (8)). The ECDSA does not suffer from any overhead and therefore the energy cost per signature is constant. The first column in Table 4 shows the signing cost of the different schemes after the distances between the cost curves have stabilized (at $S = 250$). Note that the one-time cost curves continue rising, thus at some point the ECDSA will be the most efficient option. Obviously this operation point should never be used. For one-way chain based schemes the relative distance never stabilizes.

Figure 2 shows the cost per *signature verification* as a function of the number of signatures $S$. The verification cost for Merkle based schemes grows with $\log_2(S)$ because of the leaf verification process (Eq. (6)). However, this cost is negligible compared to the constant cost of verifying a signature, resulting in a near-constant verification cost. One-way chain based schemes have a constant verification cost. The ECDSA verification cost is also constant. The second column in Table 4 shows the verification cost of the different schemes at $S = 250$.

Comparing Fig 1 with Fig. 2 we see that the verification cost is lower than the signature generation cost for all one-time signature schemes. For one-way chain base schemes this difference grows rapidly ($\sim S$), while for Merkle based schemes the difference remains near-constant.

With respect to *communications or signature size* (Fig. 3) we see that ECDSA is most efficient as the signature size is only 320 bits and there is no overhead. The fact that we need to include the public key and the authentication path in a signature, makes Merkle based schemes less efficient than one-way chain based schemes. The HORS schemes have small signature sizes but relatively large public key sizes. Therefore, they are more efficient than both the LDM and the LDW when using one-way chains, but less efficient than the LDM when using Merkle trees. The third column in Table 4 shows the communications cost in bits/signature of the different schemes at $S = 250$.

Figures 4 and 5 show the total cost, computations and communications, for the signer and the verifier. For the signer, i.e., the transmitter, the communication cost is $11.7\ \mathrm{mJ/kbit}$, while for the verifier it is $10.4\ \mathrm{mJ/kbit}$. As the communication cost is near constant, we obtain shifted versions (with a different shift for every scheme) of Fig 1 and Fig. 2 respectively. For the signer, we see that the communications cost dominates for the Merkle based schemes. For the one-way chain based schemes the computational cost rapidly grows and starts dominating. For the verifier, the communications cost always dominates. For the ECDSA, the computational cost is much larger than the cost of communications.

Figure 6 shows the total cost per signature for an example scenario. In this scenario a single signer transmits signatures to 10 verifiers using a single broadcast message. The one-time schemes are bootstrapped using an ECDSA signature on the root of the Merkle tree or on the first public key of the one-way chains. The cost of this bootstrapping process is divided over the $S$ signatures. Because of the initialization and bootstrapping costs, the cost of the different schemes first decreases with the number of signatures. Depending on the scheme, this decrease levels out at around 50–75 signatures. Beyond this threshold the cost per signature keeps increasing. The rate of increase depends on the *signing* cost of the particular scheme. The most efficient solution for this particular setting is HORS-18 using one-way chains with about 75 signatures per one-way chain. The last column in Table 4 shows the cost of the different schemes for $S = 75$ signatures per Merkle tree or one-way chain. Using multiple instances of HORS-18 (including the bootstrapping cost) will yield a five-fold increase in efficiency compared to using plain ECDSA. Figure 5 shows that ECDSA is less efficient than HORS-18 for the verifier. This means that the efficiency difference between ECDSA and HORS-18 will grow further in the case of more verifiers.

Summarizing we see that (1) one-way chains are most efficient for signature verification, (2) ECDSA and Merkle trees are most efficient for signature generation, and (3) ECDSA is the best candidate with respect to communications costs. If we take the average costs of all one-time signature schemes in Table 4, we obtain $92.8\ \mathrm{mJ/signature}$ for signature generation, $10.4\ \mathrm{mJ/signature}$ for verification and $8.9\ \mathrm{mJ/signature}$ for communications.[2] This shows that signature generation is about ten times more demanding than verification or communications.

### 5.6. An example for a hardware assisted approach for PKC: low-cost and low-power ECC processor

In this section we also roughly estimate PKC performance for the case of ECC processor that performs the point multiplication. This operation is the foundation of all required

[2] At a rounded $10\ \mathrm{mJ/kbit}$.

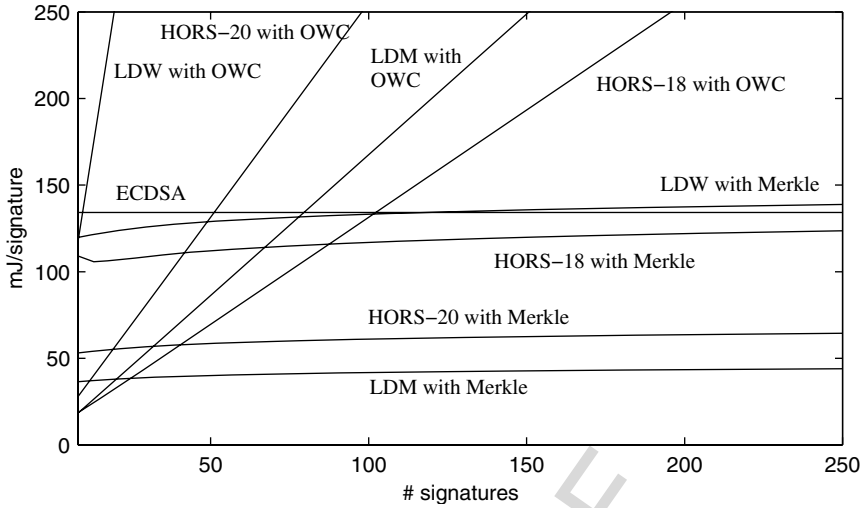



**Figure 1.** Signing efficiency of one-time signature schemes.

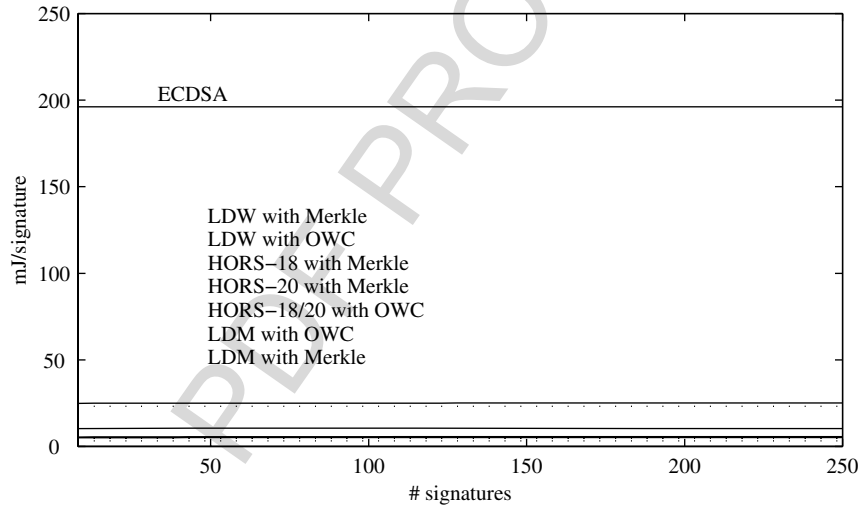



**Figure 2.** Verification efficiency of one-time signature schemes.

protocols as shown above. We discuss design criteria that should lead to a feasible ECC solution for sensor network applications. We do not give details about architectures as hardware implementations are discussed in another chapter.

The main operation in any elliptic curve-based primitive is the scalar multiplication. The hierarchical structure for operations required for implementations of ECC is as follows. Point multiplication is at the top level. At the next (lower) level are the point group operations (addition and doubling). The lowest level consists of finite field operations such as addition, subtraction, multiplication and inversion required to perform the group operations. The levels of ECC hierarchy usually map to typical architectures for low-

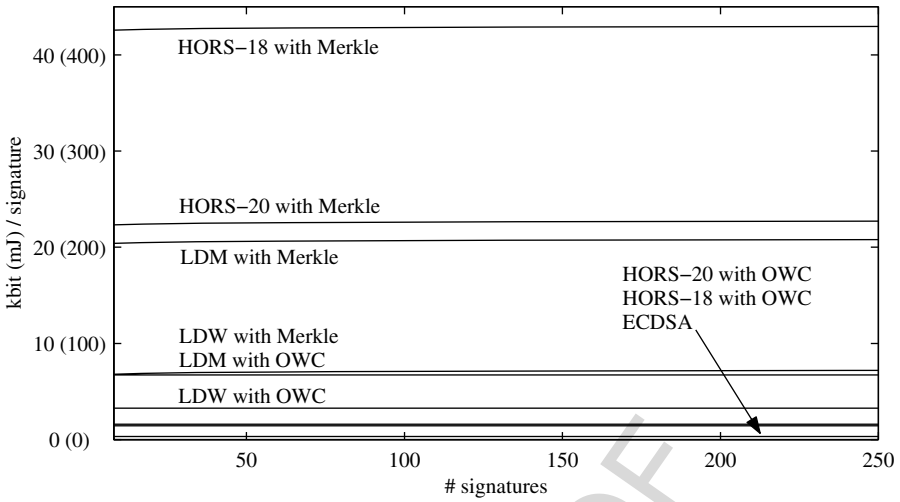



**Figure 3.** Communication efficiency of one-time signature schemes. The energy cost (mJ/signature) is a rough approximation at 10 mJ/kbit.



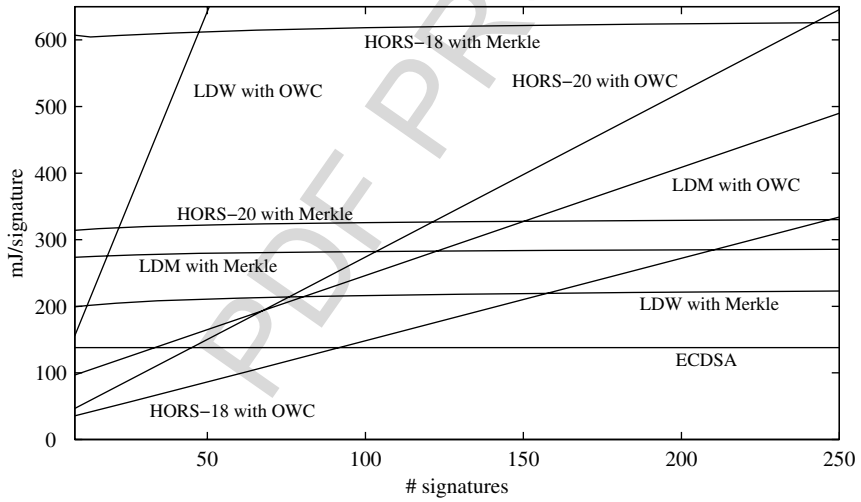



**Figure 4.** Energy consumption of signer (communications and computations).

cost applications in the following way: point multiplication is the building block for all ECC-based protocols and it is realized as a control unit, memory (RAM and ROM) and an arithmetic unit. In ROM the ECC parameters and the constants can be stored. On the other hand, RAM contains all input/output and intermediate variables [18]. Mainly all proposals for hardware implementations of ECC for light weight cryptography deal with ECC over binary fields.

We address each of those levels separately and for on all levels we discuss some

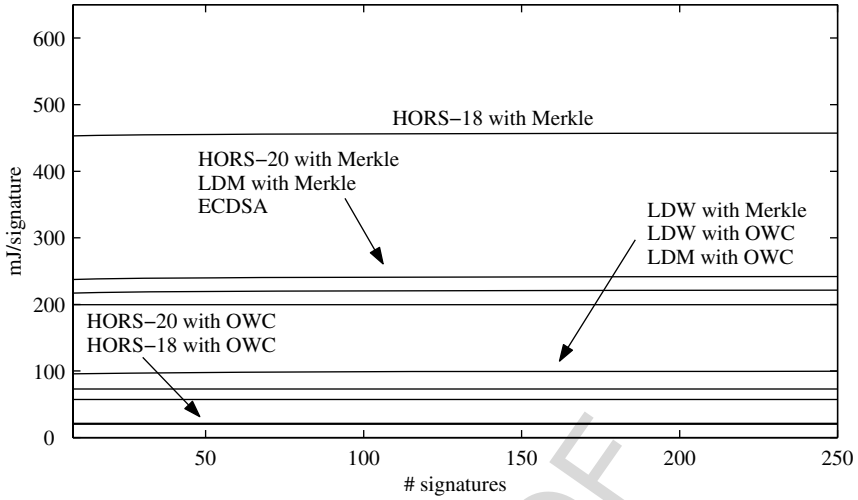



**Figure 5.** Energy consumption of verifier (communications and computations).

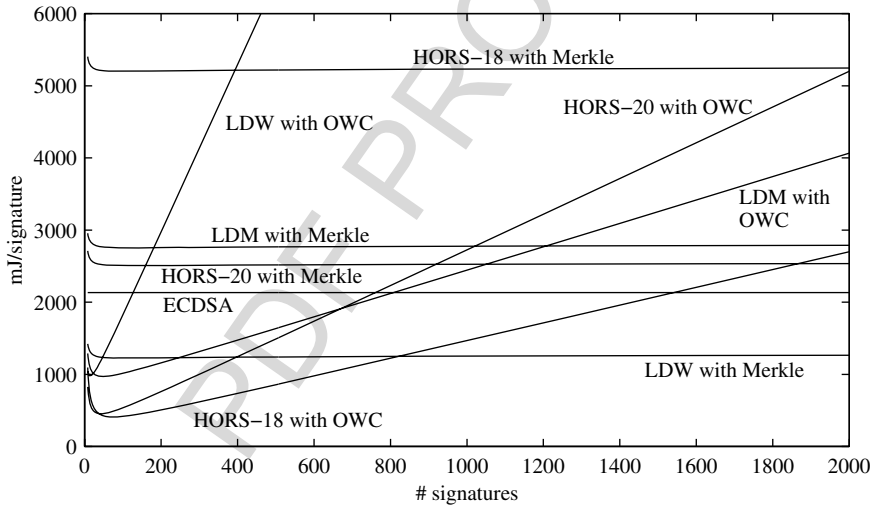



**Figure 6.** Overall energy consumption of one-time signature schemes. This setting assumes one signer and ten verifiers.

algorithmic and architectural choices leading to a feasible solution. One important issue is flexibility. Namely, ECC allows a great choice of fields, curves and other parameters and algorithms. However, it is necessary to limit the flexibility in order to minimize the area and maximize the performance. In this way for binary fields one choice for the field and irreducible polynomial results in the smallest possible arithmetic unit for the given security level. This is, of course dictated by the field size. Another examples leading to optimization in the area and performance are use of special curves (e.g. Koblitz curves), specific algorithms for point multiplication (such as windowing methods), special pro-

**Table 4.** Efficiency of one-time signature schemes.

| # signatures | Signing 250 [mJ] | Verification 250 [mJ] | Comms 250 [bit] | Total 75 [mJ] |
|---|---|---|---|---|
| LDM with Merkle | **42.3** | 5.19 | 20,797 | 2,509.2 |
| LDW with Merkle | 138.9 | 25.09 | 7,197 | 1,226.7 |
| HORS-20 with Merkle | 64.5 | 5.65 | 22,717 | 2,756.6 |
| HORS-18 with Merkle | 123.7 | 10.53 | 42,957 | 5,202.0 |
| LDM with OWC | ↑ | **3.25** | 6,720 | 986.1 |
| LDW with OWC | ↑ | 23.23 | 3,280 | 1,540.6 |
| HORS-20 with OWC | ↑ | 4.96 | 1,600 | 488.1 |
| HORS-18 with OWC | ↑ | 5.61 | 1,440 | **408.3** |
| ECDSA | 134.2 | 196.20 | **320** | 2,133.2 |

The entries are cost per signature.

**Algorithm 9** Algorithm for point multiplication

**Require:** an integer $k > 0$ and a point $P(x, y)$
**Ensure:** $Q = kP$
$k \leftarrow k_{l-1}, ..., k_1, k_0$
$X_1 \leftarrow x,\ Z_1 \leftarrow 1, X_2 \leftarrow x^4 + b, Z_2 \leftarrow x^2.$
**for** $i$ from $l - 2$ downto 0 **do**
If $k_i = 1$ then
$x(P_1) \leftarrow x(P_1 + P_2), x(P_2) \leftarrow x(2P_2)$
Else
$x(P_2) \leftarrow x(P_2 + P_1), x(P_1) \leftarrow x(2P_1)$
**end for**
Return $x(P_1)$



jective coordinates, irreducible polynomial (e.g. trinomial or pentanomial). Here we discuss some special choices such as Montgomery ladder for scalar multiplication, projective coordinates of Lopez and Dahab etc. We also deal here only with the field $\mathbb{F}_{2^{163}}$, which is recommended by many standards and it provides the same level of security as RSA of 1024 bits (or higher [15,16]). We stress again, that fixing arithmetic as well as parameters results in a compact PKC solution for pervasive security.

### *5.6.1. Minimizing memory requirements*

For the point multiplication a good choice is the method of Montgomery (Algorithm 9) [11] that maintains the relationship $P_2 - P_1$ as invariant. It uses a representation where computations are performed on the $x$-coordinate only in affine coordinates (or on the $X$ and $Z$ coordinates in projective representation). That fact allows one to save registers which is one of the main criteria for obtaining a compact solution.

The formulas for point addition/doubling that are suitable for Algorithm 9 are given in the work of Lopez and Dahab [12]. The original formulas in [12] require 2 intermediate variables but it is possible to eliminate one more intermediate register [14]. The formulae for point operations in that case are shown in Algorithm 10. The performance remains intact as one variable can be eliminated by simply reordering the field operations.

**Algorithm 10** EC point operations that minimize the number of registers

**Require:** $X_i, Z_i$, for $i = 1, 2$, $x_4 = x(P_2 - P_1)$
**Ensure:** $X(P_1 + P_2) = X_2$, $Z(P_1 + P_2) = Z_2$
1: $X_2 \leftarrow X_2 \cdot Z_1$
2: $Z_2 \leftarrow X_1 \cdot Z_2$
3: $T \leftarrow X_2 \cdot Z_2$
4: $Z_2 \leftarrow Z_2 + X_2$
5: $Z_2 \leftarrow {Z_2}^2$
6: $X_2 \leftarrow x_4 \cdot Z_2$
7: $X_2 \leftarrow X_2 + T$

**Require:** $c \in \mathbb{F}_{2^n}$, $b = c^2$, $X_1, Z_1$
**Ensure:** $X(2P_1) = X_1$, $Z(2P_1) = Z_1$,
1: $T \leftarrow c$
2: $X_1 \leftarrow X_1^2$
3: $Z_1 \leftarrow Z_1^2$
4: $T \leftarrow T \cdot Z_1$
5: $Z_1 \leftarrow X_1 \cdot Z_1$
6: $T \leftarrow T^2$
7: $X_1 \leftarrow X_1^2$
8: $X_1 \leftarrow X_1 + T$

Algorithm 10 requires only one intermediate variable $T$, which results in 5 registers in total. Namely, the required registers are for the storage of the following variables: $X_1$, $X_2$, $Z_1$, $Z_2$ and $T$. Also, the algorithm shows the operations and registers required if the key-bit $k_i = 0$. Another case is completely symmetric and it can be performed accordingly. More precisely, if the addition operation is viewed as a function $f(X_2, Z_2, X_1, Z_1) = (X_2, Z_2)$ for $k_i = 0$ due to the symmetry for the case $k_i = 1$ we get $f(X_1, Z_1, X_2, Z_2) = (X_1, Z_1)$ and the correct result is always stored in the first two input variables. This is possible due to the property of scalar multiplication based on Algorithm 9. When Algorithm 9 deploys Algorithm 10 we get the following number of operations in $\mathbb{F}_{2^n}$:

$$\begin{aligned} \#registers &= 5 \\ \#multiplications &= 11\lfloor log_2 k \rfloor + 2 \\ \#additions &= 6\lfloor log_2 k \rfloor + 1 \end{aligned}$$

*5.6.2. Reduction of control logic*

The point multiplication and point addition/doubling can be implemented as Finite State Machines (FSMs). The optimization in the number of register requires frequent checking of the value of a key-bit $k_i$. Namely, due to necessary savings in memory the same registers can be used for both point operations and since $k_i \in \{0, 1\}$ in the both cases one has to choose which variable to write/read in available memory locations. This fact enlarges the size of the control logic block but memory requirements are minimized as discussed above.

*5.6.3. Simplifying arithmetic unit*

From the formulae for point operations as given in Algorithm 10 it is evident that one needs to implement only multiplications and additions. Squaring can be considered as a special case of multiplication in order to minimize the area and inversion is avoided by use of projective coordinates. We assume that one inversion that is necessary for conversion of projective to affine coordinates can be computed at the base station's side. Note also that, if necessary, the one inversion that is required can be calculated by use of multiplications i.e. by means of Fermat's theorem [10].

Here we consider polynomial bases, where the elements of $\mathbb{F}_{2^n}$ are polynomials of degree at most $n-1$ over $\mathbb{F}_2$, and arithmetic is carried out modulo an irreducible polynomial $f(x)$ of degree $n$ over $\mathbb{F}_2$. In this case the basis elements have the form $1, \omega, \omega^2, \ldots, \omega^{n-1}$ where $\omega$ is a root in $\mathbb{F}_{2^n}$ of the irreducible polynomial $f(x)$ of degree $n$ over $\mathbb{F}_2$. According to this representation an element of $\mathbb{F}_{2^n}$ is a polynomial of length $n$ and can be written as: $a(x) = \sum_{i=0}^{n-1} a_i x^i = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_1 x + a_0$, where $a_i \in \mathbb{F}_2$.

So-called classical modular multiplication is typically based on the following equation:

$$\begin{aligned} a(x) \cdot b(x) &= (a_{n-1}x^{n-1} + \cdots + a_1 x + a_0) \cdot b(x) \bmod f(x) \\ &= (\cdots (a_{n-1}b(x)x + a_{n-2}b(x))x + \cdots \\ &\quad + a_1 b(x))x + a_0 b(x) \bmod f(x) \end{aligned} \tag{10}$$

which illustrates the Horner scheme for multiplication. This scheme is the basis of the Most Significant Bit-First (MSB) multiplier [13].

The modular addition operation in binary fields is simply the XOR operation which can be performed in one clock cycle. By reusing the logic for multiplication, it is possible to implement modular addition by adding only some more control logic to the existing multiplier [17].

More details about the architecture following the principles above are given in [14].

### *5.6.4. Performance Estimates and Discussion*

In this section we list some numbers for the latency and power consumption for ECC algorithms with the choices for arithmetic as described above [14]. The architecture we refer to has a possibility to deploy the multiplier with various digit-sizes $d$ [17]. For the point multiplication we used Algorithm 1 and for point operations Algorithm 2. The numbers for power assume the operating frequency of 500 $kHz$ for the field $\mathbb{F}_{2^{163}}$. With this frequency the power stays below 30 $\mu W$ which is assumed to be acceptable for sensor networks applications.

The results for the total number of cycles of one point multiplication for fields $\mathbb{F}_{2^{131}}$ and $\mathbb{F}_{2^{163}}$ are given in Table 5. To calculate the time for one point multiplication we need

**Table 5.** The number of cycles required for one point multiplication for ECC [14].

| Field size | *d*=1 | *d*=2 | *d*=3 | *d*=4 |
|---|---|---|---|---|
| 131 | 191 750 | 98 800 | 68 770 | 53 040 |
| 139 | 215 694 | 112 470 | 76 038 | 59 340 |
| 151 | 254 250 | 109 200 | 74 880 | 57 720 |
| 163 | 295 974 | 151 632 | 103 518 | 80 352 |

an operating frequency. However, the frequency that can be used is strictly influenced by the total power. We assumed an operating frequency of 500 $kHz$ as suggested in [2] in order to estimate the actual timing. We get 106 $ms$ for the best case of ECC over $\mathbb{F}_{2^{131}}$ ($d = 4$) and 160.70 $ms$ for the best case of ECC over $\mathbb{F}_{2^{163}}$ ($d = 4$).

The current generation of sensor networks is powered by batteries, so ultra-low power circuitry is a must for these applications. For low-power consumption it is neces-

sary to minimize the switching or dynamic power, the circuit size (for the current CMOS technology) and the operating frequency. This can be achieved by architectural decisions, which we discussed above. However, besides switching power, energy efficiency is even more crucial as sensor nodes are still battery operated. More precisely, the metric that is typically used and that should be minimized accordingly is energy per processed bit $E$. This value can be calculated as:

$$E = \frac{P}{throughput} \left[\frac{J}{bit}\right] .$$

The results for the total power consumption and for energy per bit are given in Figure 7 and Figure 8 respectively for the field $\mathbb{F}_{2^{163}}$. We can observe that both contributions to the total power *i.e.* static and dynamic are close to each other. We also notice that the energy per bit improves as $d$ increases. This can be explained due to a decrease in number of cycles, which is more dominant than an increase in power consumption for higher $d$.

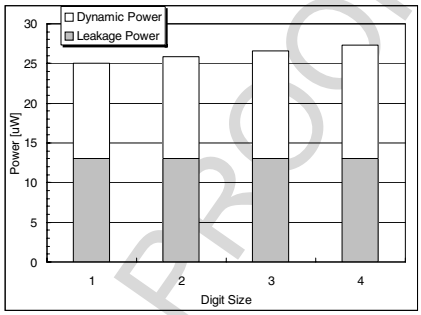



**Figure 7.** The power consumed for various digit sizes for ECC over $\mathbb{F}_{2^{163}}$.

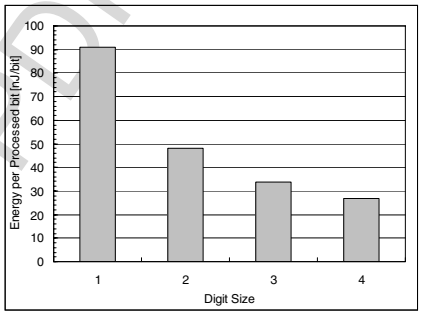



**Figure 8.** Results for energy per encrypted bit for all digit sizes.

The point multiplication is the most consuming operation in ECC-based protocols so we do not need to take other operations into account. Since each protocol consists of a certain number of point multiplication (usually one or two) we can estimate costs per ECC-protocol. As an example we estimate the energy per message for ECDSA (see Table 6). In this way we can also roughly estimate the energy costs per protocol.

**Table 6.** Energy per message for ECDSA for the field $\mathbb{F}_{2^{163}}$.

| Protocol | $d$=1 | $d$=2 | $d$=3 | $d$=4 |
|---|---|---|---|---|
| ECDSA-sign | 14.822 $\mu J$ | 7.837 $\mu J$ | 5.509 $\mu J$ | 4.393 $\mu J$ |
| ECDSA-verify | 26.643 $\mu J$ | 15.674 $\mu J$ | 11.018 $\mu J$ | 8.785 $\mu J$ |

Published results for area, power and latency for PKC are compared in Table 7. The performances given for ECC are for one point multiplication.

**Table 7.** Comparison with other related work.

| Ref. | PKC | Area [$gates$] | Techn. [$\mu m$] | $f$ [kHz] | Perf. [$ms$] | $P$ [$\mu W$] |
|---|---|---|---|---|---|---|
| [7] | ECC over $\mathbb{F}_{2^{131}}$ | 11 969.93 | 0.35 | 13 560 | 18 | - |
| [1] | ECC over $\mathbb{F}_{p_{100}}$ | 18 720 | 0.13 | 500 | 410.45 | $< 400$ |
| [2] | NTRU | 2 850 | 0.13 | 500 | 58.45 | $< 21$ |
| [6] | ECC over $\mathbb{F}_{2^{191}}$, $\mathbb{F}_{p_{192}}$ | 23 000 | 0.35 | 68 500 | 9.89 | n.a. |
| [14] | $\mathbb{F}_{2^{131}}$ | 8104 | 0.13 | 500 | 106 | $< 30$ |



## Acknowledgements

The work described in this document has been partly financially supported by the European Commission through the IST Program under Contract IST-2002-507932 ECRYPT.

It was also supported in part by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy), by FWO projects EMA G.0475.05 and BBC G.0300.07, by the IBBT-QoE project of the IBBT and by the K.U.Leuven-BOF (OT/06/40).



## References

[1] G. Gaubatz, J.-P. Kaps, E. Öztürk and B. Sunar, "State of the Art in Ultra-Low Power Public Key Cryptography for Wireless Sensor Networks", 2nd IEEE International Workshop on Pervasive Computing and Communication Security (PerSec 2005).

[2] G. Gaubatz, J.-P. Kaps and B. Sunar, "Public Key Cryptography in Sensor Networks - Revisited", 1st European Workshop on Security in Ad-Hoc and Sensor Networks (ESAS 2004).

[3] L. Uhsadel, A. Poschmann, and C. Paar, "Enabling Full-Size Public-Key Algorithms on 8-bit Sensor Nodes", In 4th European Workshop on Security and Privacy in Ad hoc and Sensor Networks, Lecture Notes in Computer Science 4572, S. Capkun, C. Meadows, and F. Stajano (eds.), Springer-Verlag, pp. 73-86, 2007, Springer-Verlag.

[4] L. Batina, D and Hwang, A. Hodjat, B. Preneel and I. Verbauwhede, "Hardware/Software Co-design for Hyperelliptic Curve Cryptography (HECC) on the 8051 $\mu P$", Proceedings of 7th International Workshop on Cryptographic Hardware and Embedded Systems CHES 2005, eds. J. R. Rao and B. Sunar, LNCS 3659 pp. 106-118, Springer-Verlag.

[5] A. Hodjat, L. Batina, D. Hwang and I. Verbauwhede, "HW/SW Co-design of a Hyperelliptic Curve Cryptosystem using a $\mu$Code Instruction Set Coprocessor", Elsevier Science Integration the VLSI Journal, vol. 1, nr. 40, pp.45-51, 2006.

[6] J. Wolkerstorfer, "Scaling ECC Hardware to a Minimum", In ECRYPT workshop - Cryptographic Advances in Secure Hardware - CRASH 2005, Leuven, Belgium, Invited talk.
[7] S. Kumar and C. Paar, "Are standards compliant Elliptic Curve Cryptosystems feasible on RFID?", Proceedings of Workshop on RFID Security, 2006, Graz, Austria.
[8] I. Blake, G. Seroussi and N.P. Smart, "Elliptic Curves in Cryptography", Cambridge University Press, 1999, London Mathematical Society Lecture Note Series.
[9] D. Hankerson, A. Menezes and S. Vanstone, "Guide to Elliptic Curve Cryptography", Springer-Verlag, 2004
[10] N. Koblitz, "A Course in Number Theory and Cryptography", Springer-Verlag, 1994
[11] P. Montgomery, "Speeding the Pollard and Elliptic Curve Methods of Factorization", Mathematics of Computation, 1987, Vol. 48, nr. 177, pp. 243-264.
[12] J. López and R. Dahab, "Fast Multiplication on Elliptic Curves over GF($2^m$)", Proceedings of 1st International Workshop on Cryptographic Hardware and Embedded Systems CHES 1999, eds. Ç.K. Koç and C. Paar, LNCS 1717 pp. 316-327. Springer-Verlag.
[13] T. Beth and D. Gollmann, "Algorithm Engineering for Public Key Algorithm", IEEE Journal on Selected Areas in Communications, Vol. 7, nr. 4, pages 458-465, 1989.
[14] L. Batina, N. Mentens, K. Sakiyama, B. Preneel and I. Verbauwhede, "Low-cost Elliptic Curve Cryptography for wireless sensor networks", Proceedings of Third European Workshop on Security and Privacy in Ad hoc and Sensor Networks, eds. L. Buttyan, V. Gligor and D. Westhoff, LNCS 4357 pp. 6-17, Springer-Verlag, 2006.
[15] A. Lenstra and E. Verheul, "Selecting cryptographic key sizes", Proceedings of Third International Workshop on Practice and Theory in Public Key Cryptography (PKC 2000), eds. H. Imai and Y. Zheng, LNCS 1752, pp. 446-465, Springer-Verlag.
[16] ECRYPT, "ECRYPT Yearly Report on Algorithms and Keysizes (2006)".
[17] K. Sakiyama, L. Batina, N. Mentens, B. Preneel and I. Verbauwhede, "Small-footprint ALU for public-key processors for pervasive security", Proceedings of Workshop on RFID Security 2006, Graz, Austria.
[18] P. Tuyls and L. Batina, "RFID-tags for Anti-Counterfeiting", Topics in Cryptology - CT-RSA 2006, ed. D. Pointcheval, LNCS 3860, pp. 115-131, Springer Verlag.
[19] J. Hoffstein, J. Pipher and J.H. Silverman, "NTRU: a ring based public key cryptosystem", In Proceedings of ANTS III, 1998, LNCS 1423, pp. 267Ű288. Springer-Verlag.
[20] ECRYPT, "Hardness of the Main Computational Problems Used in Cryptography", Deliverable 1.2, 2007.
[21] D. Johnson and A. Menezes, The Elliptic Curve Digital Signature Algorithm (ECDSA), Department of Combinatorics & Optimization, 2000, `http://www.cacr.math.uwaterloo.ca`.
[22] A. Menezes, P. van Oorschot and S. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1997.
[23] T. Beth, *Efficient Zero-Knowledge Identification Scheme for Smart Cards*, Advances in Cryptology — EUROCRYPT'88, ed. C. G. Günther, 1988, pp. 77-84.
[24] M. Bellare, C. Namprempre and G. Neven, Security proofs for identity-based identification and signature schemes, Advances in Cryptology — Eurocrypt 2004, eds. C. Cachin and J. Camenisch, LNCS 3027, 2004, pp. 268-286, Springer-Verlag.
[25] M. Feldhofer, S. Dominikus and J. Wolkerstorfer, Strong Authentication for RFID Systems using the AES Algorithm, Proceedings of 6th International Workshop on Cryptographic Hardware in Embedded Systems (CHES), eds. M. Joye and J. -J. Quisquater, LNCS 3156, pp. 357-370, 2004, Springer-Verlag.
[26] P. Nguyen and N. Gama, New Chosen-Ciphertext Attacks on NTRU. In Proceedings of PKC 2007, eds. T. Okamoto and X. Wang, LNCS 4450, 2007, pp. ?, Springer-Verlag.
[27] S. Seys Cryptographic algorithms and protocols for security and privacy in wireless ad hoc networks. PhD thesis, *Katholieke Universiteit Leuven*, B. Preneel (promotor), 172+37 pages, 2006.
[28] M. Abdalla, M. Bellare, and P. Rogaway. The oracle Diffie-Hellman assumptions and an analysis of DHIES. In *Topics in Cryptology – RSA Conference Cryptographers' Track (RSA-CT 2001)*, volume 2020 of *Lecture Notes in Computer Science*, pages 143–158. Springer-Verlag, 2001.
[29] G. Anastasi, A. Falchi, A. Passarella, M. Conti, and E. Gregori. Performance measurements of motes sensor networks. In *Proceedings of the 7th International Symposium on Modeling Analysis and Simulation of Wireless and Mobile Systems (MSWiM 2004)*, pages 174–181. ACM Press, 2004.
[30] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1th ACM Conference on Computer and Communications Security (CCS 1993)*, pages 62–73. ACM Press, 1993.

[31] M. Bellare and P. Rogaway. Optimal asymmetric encryption. In *Advances in Cryptology - EUROCRYPT 1994*, volume 950 of *Lecture Notes in Computer Science*, pages 92–111. Springer-Verlag, 1995.

[32] M. Bellare and P. Rogaway. The exact security of digital signatures: How to sign with RSA and Rabin. In *Advances in Cryptology - EUROCRYPT 1996*, volume 1070 of *Lecture Notes in Computer Science*, pages 399–414. Springer-Verlag, 1996.

[33] J. N. E. Bos and D. Chaum. Provably unforgeable signatures. In *Advances in Cryptology - CRYPTO 1992*, volume 740 of *Lecture Notes in Computer Science*, pages 1–14. Springer-Verlag, 1993.

[34] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[35] M. Jakobsson, T. Leighton, S. Micali, and M. Szydlo. Fractal Merkle tree representation and traversal. In *Topics in Cryptology – RSA Conference Cryptographers' Track (RSA-CT 2003)*, volume 2612 of *Lecture Notes in Computer Science*, pages 314–326. Springer-Verlag, 2003.

[36] D. Johnson and A. Menezes. The elliptic curve digital signature algorithm (ECDSA). Technical report CORR 99-34, Departement of Combinatorics & Optimizations, University of Waterloo, Canada, 1999. Updated: 2000/02/24.

[37] L. Lamport, "Constructing digital signatures from a one-way function," Technical Report CSL-98, SRI International, 1979.

[38] A. K. Lenstra and E. R. Verheul. Selecting cryptographic key sizes. *Journal of Cryptology*, 14(4):255–293, 2001.

[39] W. Mao. *Modern Cryptography – Theory & Practice*. Prentice Hall PTR, 2004.

[40] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[41] R. C. Merkle. *Secrecy, Authentication and Public Key Systems*. UMI Research Press, 1982.

[42] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Advances in Cryptology - CRYPTO 1987*, vol. 293 of *Lecture Notes in Computer Science*, pp. 369–378, Springer-Verlag, 1987.

[43] R. C. Merkle. A certified digital signature. In *Advances in Cryptology - CRYPTO 1989*, volume 435 of *Lecture Notes in Computer Science*, pages 218–238. Springer-Verlag, 1990.

[44] Adrian Perrig. The BiBa one-time signature and broadcast authentication protocol. In *Proceedings of the 8th ACM Conference on Computer and Communications Security (CCS 2001)*. ACM Press, 2001.

[45] PKCS #1 version 2.1: RSA cryptography standard. Public-Key Cryptography Standard 1, RSA Laboratories, 2002.

[46] N. R. Potlapally, S. Ravi, A. Raghunathan, and N. K. Jha. Analyzing the energy consumption of security protocols. In *Proceedings of the 2003 International Symposium on Low Power Electronics and Design (ISLPED 2003)*, pages 30–35, 2003.

[47] M. O. Rabin, "Digitalized signatures," *Foundations of Secure Computation*, pp. 155–168, 1978.

[48] L. Reyzin and N. Reyzin. Better than BiBa: Short one-time signatures with fast signing and verifying. In *Proceedings of the 7th Australian Conference on Information Security and Privacy*, volume 2384 of *Lecture Notes in Computer Science*, pages 144–153. Springer-Verlag, 2002.

[49] Ronald L. Rivest, Adi Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, 1978.

[50] SEC 2: Recommended elliptic curve domain parameters. Version 1.0, Standards for Efficient Cryptography Group, 2000.

[51] V. Shoup. A proposal for an ISO standard for public key encryption. Version 2.1, IBM Zurich research lab, 2001.