ECDSA and QUARTZ in Comparison with RSA

Patrick Lieb[1], Markus Schmidt[1], and Alex Wiesmaier[1,2,3]

[1] TU Darmstadt
[2] AGT International
[3] Hochschule Darmstadt

Abstract. Standard digital signature schemes like RSA cannot be used in IoT. This work investigates two algorithms, ECDSA and QUARTZ; that are supposedly suitable for resource-constrained devices. In addition, the German variant ECGDSA and the Korean variant ECKCDSA are evaluated.

Keywords: ECDSA, ECGDSA, ECKCDSA, QUARTZ, RSA, IoT

1 Introduction

Digital signature algorithms are cryptographic algorithms that are supposed to provide data authenticity, integrity and non-repudiation. RSA is the most common standard for asymmetric encryption and decryption, as well as for signing and verifying. However, the RSA signature scheme is unsuitable for use in lightweight devices with low bandwidth, low computing power and little memory. In Section 3, an examination of ECDSA is shown together with its two variants ECGDSA in Section 3.3 and ECKCDSA in Section 3.3, containing an analysis of its security in Section 3.4. In Section 4, QUARTZ is presented, containing an analysis of its security in Section 4.4. The Sections 3.6 and 4.6 are dealing with the comparison between RSA and ECDSA and QUARTZ, respectively. The advantages and disadvantages of ECDSA and QUARTZ are shown as well as suitable areas of application are given in Section 5.

2 Related Work

The Elliptic Curve Digital Signature Algorithm (ECDSA) is a very well known algorithm that has already been investigated in a few papers. An important introduction to elliptic curve cryptography is given by Hankerson et al. [17] with their "Guide to Elliptic Cryptography". The elliptic curve digital signature algorithm ECDSA was originally presented by Johnson et al. [22]. In addition, similar variants have been introduced such as the ECGDSA by Hess et al. [19] and the BSI [4]; or ECKCDSA that was introduced by Lim et al. [26]. Vaudeny [40] performed an extensive security evaluation of ECDSA. A good comparison of ECDSA and ECGDSA is given by Sarath et al. [37]. However, there is no paper that examines ECDSA, including its variants ECGDSA and ECKCDSA, that considers an evaluation of security and performance.

The QUARTZ signature scheme was originally described by Courtois et al. [33] and later updated [9], targeting performance issues. In addition to the description, there is a categorization of attacks and performance data. An important examination of the impact of the perturbation operations "v" and "-" on basic HFE and an implicit security analysis on QUARTZ is given by Courtois et al. [8]. Ding et al. [11] introduced the GUI signature scheme, an improvement of QUARTZ also based on HFEv-. Additionally, QUARTZ and its underlying basics are well described and a comparison of GUI, QUARTZ and RSA is presented. Other signature algorithms based on HFE are GUI, SFLASH or PFLASH.

Tame Transformation Signatures, introduced as TTS, that are based on the tame transformation method [6], belong to the extended family of successors of the Matsumoto-Imai signature scheme C*.

The balanced and unbalanced Oil and Vinegar signature schemes are based on multivariate polynomials over a finite field. Principles of this signature scheme are used in QUARTZ for the perturbation operation "v". The Rainbow signature scheme [12] describes a generalization of the Oil-Vinegar signature scheme.

3 Elliptic Curve Digital Signature Algorithm ECDSA

Elliptic curve cryptography was independently invented by Victor Miller [29] and Neal Koblitz [25], in 1985 and 1987 respectively. The security of elliptic curve cryptography is based on the elliptic curve discrete logarithm problem ECDLP. The elliptic curve digital signature algorithm ECDSA is a variation of DSA. However, unlike DSA it is based on elliptic curves. It was first approved by ANSI in 1999 and later also accepted by IEEE and NIST. Over the years it has gained in popularity, especially for resource-constrained devices.

In order to explain the mechanisms of ECDSA, it is first necessary to explain the mathematical principles of elliptic curves.

3.1 Elliptic Curves Over Finite Fields

Elliptic curve cryptography is based on elliptic curves over finite fields. A finite field is defined as a finite set of elements $\mathbb{F}_q$. The order q of $\mathbb{F}_q$ describes the number of elements in $\mathbb{F}_q$. An elliptic curve $\mathbb{E}$ over a field $\mathbb{F}_q$ is a cubic curve that is defined by the Weierstrass Equation

$$\mathbb{E} : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \ , \tag{1}$$

with $a_1, a_2, a_3, a_4, a_6 \in \mathbb{F}_q$ and $\Delta \neq 0$, where Δ describes the discriminant of $\mathbb{E}$. The discriminant Δ is given by the following equation:

$$\Delta = -b_2^2b_8 - 8b_4^3 - 27b_6^2 + 9b_2b_4b_6 \ , \tag{2}$$

with

$$\begin{aligned}
b_2 &= a_1^2 + 4a_2 \ , \\
b_4 &= 2a_4 + a_1 a_3 \ , \\
b_6 &= a_3^2 + 4a_6 \text{ and} \\
b_8 &= a_1^2 a_6 + 4a_2 a_6 - a_1 a_3 a_4 + a_2 a_3^2 - a_4^2 \ .
\end{aligned}$$

There are three different types of curves that are used in ECDSA [22]; elliptic curves over prime fields, elliptic curves over binary fields and Koblitz curves.

Elliptic Curves over Prime Field $\mathbb{F}_p$. A field of odd prime order $q = p$ with $p > 3$ is called prime field $\mathbb{F}_p$. In general, the Weierstrass Equation can be simplified if the characteristic of a field is not equal to 2 or 3. Since prime fields are satisfying this condition, the elliptic curve $\mathbb{E}$ over prime field $\mathbb{F}_p$ can be simplified to

$$\mathbb{E} : y^2 = x^3 + ax + b \quad \text{where } a, b \in \mathbb{F}_p \text{ and } 4a^3 + 27b^2 \not\equiv 0 \pmod{p} \ . \tag{3}$$

The elliptic curve $\mathbb{E}(\mathbb{F}_p)$ is defined by all points (x, y) with $x, y \in \mathbb{F}_p$ that satisfy (3). The term $4a^3 + 27b^2$ is based on the discriminant of $\mathbb{E}(\mathbb{F}_p)$ that can be specified as $\Delta = -16(4a^3 + 27b^2)$. Based on the definition of elliptic curves, Δ must not be 0. The special point $\mathcal{O}$ is denoted as *point of infinity.*

Elliptic Curves over Binary Field $\mathbb{F}_{2^m}$. A field of characteristic equal to 2 is called binary field, or characteristic two finite field $\mathbb{F}_{2^m}$ with m labeled as the extension degree of the field. Since the characteristic of $\mathbb{F}_{2^m}$ is equal to 2, the elliptic curve $\mathbb{E}$ over binary field $\mathbb{F}_{2^m}$ can be described by

$$\mathbb{E} : y^2 + xy = x^3 + ax^2 + b \quad \text{where } a, b \in \mathbb{F}_{2^m} \text{ and } b \neq 0 \ . \tag{4}$$

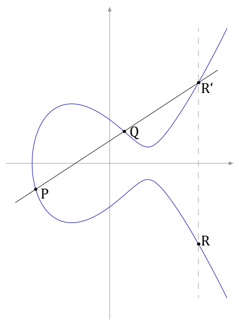
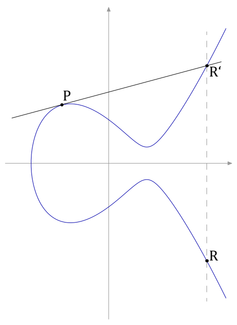
The finite set of elements $\mathbb{E}(\mathbb{F}_{2^m})$ is defined by all points (x, y) with $x, y \in \mathbb{F}_{2^m}$ that satisfy (4). The special point $\mathcal{O}$ is again denoted as *point of infinity.*

Koblitz Curves. In addition, there is a special type of elliptic curves over binary fields called Koblitz curves [25], or binary anomalous curves. These curves are also defined over $\mathbb{F}_{2^m}$ but a and b have to be either 0 or 1. Since (4) defines that b must not be 0, these curves can be described by the following equation:

$$\mathbb{E}_a : y^2 + xy = x^3 + ax^2 + 1 \quad \text{where } a \in \{0, 1\} \ . \tag{5}$$

Koblitz curves have an important property: If the point $(x, y) \in \mathbb{E}_a$ with $a \in 0, 1$, then the point $(x^2, y^2) \in \mathbb{E}_a$. Based on this, the cost of point doubling can be reduced to almost zero.

Table 1. Arithmetic Operations on Elliptic Curves

Point Addition $P + Q = R \; with \; P, Q, R \in \mathbb{F}_q$	**Point Doubling** $P + P = R \; with \; P, R \in \mathbb{F}_q$
1. find line l through P and Q 2. R' is the unique intersection of l with $\mathbb{F}_q$ 3. R is reflection of R' on x-axis	1. create tangent t of $\mathbb{F}_q$ at P 2. R' is the unique second intersection of t with $\mathbb{F}_q$ 3. R is reflection of R' on x-axis
	

Arithmetic Operations. The two binary operations, point addition and point multiplication, are defined on all elliptic curves. Table 1 shows a simplified version of point addition and point doubling in elliptic curves based on the algorithms of Hankerson et al. [17]. It is important to say that point doubling can be seen as a special case of point addition on a single point. Point doubling is supposed to be faster than point addition. This performance difference is likely to be exploited in attacks on ECDSA.
Point multiplication is defined by repeated point doubling. Thus, equation $R = kP$ is equal to $R = \underbrace{P + P + P + P + \cdots + P}_{k \text{ times}}$.

3.2 Generation and Verification Algorithms of ECDSA

ECDSA described by [22] consists of three different algorithms, key generation, signature generation and signature verification. All three algorithms have to operate on the same global domain parameters. These parameters are denoted as $\mathbb{E}$, P, n and $\mathcal{H}$. The first domain parameter $\mathbb{E}$ is the elliptic curve over field $\mathbb{F}_q$ with $q = p$ or $q = 2^m$. Therefore, this parameter includes the choice of a and b as well as p for prime fields and m for binary fields. Moreover, a base point P on $\mathbb{E}(\mathbb{F}_q)$ is required. The parameter n describes the order of the base point P. It has to satisfy the condition $4\sqrt{q} < n < 2^{160}$. The last domain parameter is the hash function $\mathcal{H}$. It has to be one-way and collision-resistant. All these parameters are public parameters.

Key Generation. The key generation algorithm does not require any input parameters except the domain parameters. This algorithm generates a key pair

(Q, d) with Q labeled as the public key and d denoted as the private key. It is defined as follows:

1. Compute random integer d with $0 < d < n$
2. Calculate $Q = dP$

Signature Generation. The signature generation algorithm takes the private key d and the message m as input parameters. It generates the signature (r, s) of message m:

1. Compute random integer k with $0 < k < n$
2. Compute $kP = (x_R, y_R)$
3. Compute $r = x_R \bmod n$; if $r = 0$ go to 1
4. Compute $k' = k^{-1} \bmod n$
5. Compute $h = \mathcal{H}(m)$
6. Compute $s = (k'h + rd) \bmod n$; if $s = 0$ go to 1
7. Signature of m is (r, s)

Signature Verification. The signature verification algorithm verifies based on the public key Q if the signature (r, s) belongs to the message m:

1. Verify that $0 < r < n$ and $0 < s < n$
2. Compute $h = \mathcal{H}(m)$
3. Compute $s' = s^{-1} \bmod n$
4. Compute $t_1 = s'h \bmod n$ and $t_2 = s'r \bmod n$
5. Compute $R = (x_R, y_R) = t_1P + t_2Q$
6. If $R = 0$ then reject signature
7. If $x_R \bmod n = r$ then accept, otherwise reject

3.3 ECDSA Variants

There are two other variants that are very similar to ECDSA that will be examined in this paper. First, the German variant, the Elliptic Curve German Digital Signature Algorithm ECGDSA [19] and second, the Korean variant, the Korean Certificate Digital Signature Algorithm ECKCDSA [26].

Elliptic Curve German Digital Signature Algorithm ECGDSA. The German Variant ECGDSA is described by Hess et al. [19] and the BSI [4]. It is derived from the original ECDSA. Its main difference is the inverted private key d^{-1} that is used to compute the public key Q. This simplifies the signature generation algorithm.

Key Generation. The key generation algorithm does not have any input parameters. Only the domain parameters are needed to compute the key pair (Q, d) with Q labeled as the public key and d denoted as the private key. The algorithm is as follows:

1. Compute random integer d with $0 < d < n$
2. Compute $d' = d^{-1} \bmod n$
3. Calculate $Q = d'P$

Signature Generation. The ECGDSA signature generation is very close to the signature generation in ECDSA. As the private key in ECGDSA is inverted in order to compute the public key,the signature generation algorithm does not need to compute the inverse of the secret random integer k:

1. Compute random integer k with $0 < k < n$
2. Compute $kP = (x_R, y_R)$
3. Compute $r = x_R \bmod n$; if $r = 0$ go to 1
4. Compute $h = \mathcal{H}(m)$
5. Compute $s = d(kr - h) \bmod n$; if $s = 0$ go to 1
6. Signature of m is (r, s)

Signature Verification. The significant difference to ECDSA is that r is inverted instead of s. The rest of the algorithm is equivalent to ECDSA:

1. Verify that $0 < r < n$ and $0 < s < n$
2. Compute $h = \mathcal{H}(m)$
3. Compute $r' = r^{-1} \bmod n$
4. Compute $t_1 = r'h \bmod n$ and $t_2 = r's \bmod n$
5. Compute $R = (x_R, y_R) = t_1P + t_2Q$
6. If $R = 0$ then reject signature
7. If $x_R\ mod\ n = r$ then accept, otherwise reject

Elliptic Curve Korean Certificate Digital Signature Algorithm ECKCDSA. This algorithm was invented by Lim et al. [26] in 1998. It is very similar to ECDSA and ECGDSA. However, it also uses the certificate of the signer in order to generate and verify signatures.

Key Generation. The key generation algorithms is similar to that of ECGDSA. It also does not have any input parameters. Only the domain parameters are needed to compute the key pair (Q, d) with Q labeled as the public key and d denoted as the private key. The key generation algorithm is as follows:

1. Compute random integer d with $0 < d < n$
2. Compute $d' = d^{-1} \bmod n$
3. Calculate $Q = d'P$

Signature Generation. The signature generation algorithm of ECKCDSA includes the hash value of the signer's certificate *cert*. The cryptographically secure hash function $\mathcal{H}$ has to produce hashes with l bit lengths. Lim et al. recommend using large subgroups of order q with $q > 2^l$:

1. Compute random integer k with $0 < k < n$
2. Compute $kP = (x_R, y_R)$
3. Compute $r = \mathcal{H}(x_R)$
4. Compute $h = \mathcal{H}(m \| cert)$
5. Compute $w = r \oplus h \bmod n$; if $w < n$ go to 7
6. Compute $w = w - n$
7. Compute $s = d(k - w) \bmod n$; if $s = 0$ go to 1
8. Signature of m is (r, s)

Signature Verification. The significant difference of the ECKCDSA signature verification algorithm compared to both other signature verification algorithms is the verification step of r. In ECKCDSA, r has to be smaller than 2^l. The rest of the algorithm is similar to the algorithms of ECDSA and ECGDSA:

1. Verify that $0 < r < 2^l$ and $0 < s < n$
2. Compute $h = \mathcal{H}(m \| hcert)$
3. Compute $w = r \oplus h \mod n$
4. Compute $R = (x_R, y_R) = sQ + wP$
5. If $\mathcal{H}(x_R) = r$ then accept, otherwise reject

3.4 Security

Vaudenay [40] describes that ECDSA has a high vulnerability if it is used in a poor way. Therefore, four necessary security conditions have to be satisfied in order to receive a secure digital signature scheme.
First, the discrete logarithm in the subgroup spanned by P, has to be hard. If it is not hard, the discrete logarithm of the public key can be computed easily. Thus, an attacker can also easily compute the secret key d.
Second, the hash function $\mathcal{H}$ used in the signing process needs to be one-way and collision resistant. If an attacker is able to find a collision of two hashes over different messages, he can sign one message but declare his signature on the other.
Third, the pseudo-random generator for k has to be unpredictable. If the random or pseudo-random generator used to generate k is not secure, it could happen that the same ephemeral key k is used to sign two different messages m_1 and m_2. A possible key recovery attack is described in Sect. 3.5. For that reason, the pseudo-random generator for k has to be cryptographically strong.
Lastly, all domain parameters need to be validated and securely stored. Without a proper validation, it would be possible to hide trapdoors in $\mathbb{E}$, P or n [40]. The validation can be performed by the "Explicit Validation of a Set of EC Domain Parameters" algorithm that is given by Johnson et al. [22]. Their algorithm checks if q is an odd prime and n is a prime with $n > 2^160$ and $n > 4\sqrt{q}$ among other things. Another way is to generate and validate the domain parameters by a trusted third party.

The security conditions of RSA are similar to the security conditions of ECDSA. The hash function $\mathcal{H}$ used in the signing process also needs to be one-way and collision resistant. Moreover, the random or pseudo-random generator to generate k has to be unpredictable as well. The main difference is that the security of RSA is based on the factorization problem. This problem describes the difficulty of factoring the product of two large prime numbers.

If the random or pseudo-random generator used in ECDSA in order to generate k is secure and the hash function is one-way and collision-resistant, the attacker has to solve the Elliptic Curve Discrete Logarithm Problem ECDLP in order to break ECDSA. This means he has to find d in $Q = dP$ with $0 < d < n$ where n is denoted as the order of the elliptic curve. There are many known attacks against ECDLP. A few examples are the exhaustive search, Pohlig-Hellman, Baby-Step Giant-Step and Pollard's Rho algorithm. However, the fastest algorithms to solve the ECDLP have a fully-exponential runtime, whereas the running times of integer factorization and the discrete logarithm problem are sub-exponential [41]. For that reason, ECDSA is more secure than RSA since it takes exponential time to break, compared to RSA which can be broken in sub-exponential time.

In addition to these four security conditions, every public key should be validated before usage. It has to be validated that Q lies on the elliptic curve and x_Q and y_R are properly represented elements of the field $\mathbb{F}_q$. Moreover, Q nor nQ must not be the special point of infinity. This validation is necessary for obvious reasons. First, it prevents the insertion of a malicious public key that might enable some attacks. Second, the public key validation is able to detect errors that might occur during the transmission. This validation can either be performed by means of the "Explicit Validation of an ECDSA Public Key Algorithm" introduced by Johnson et al. [22] or a trusted party.

Furthermore, an attacker E should not be able to claim the public key of another person A. If that would be possible, the attacker E could insist that all messages originally signed by A were signed by E. Thus, in order to prevent illegal claiming of public keys, every certificate authority CA always has to request a proof of possession of the private key corresponding to the public key that should be certified. For instance, this can be done by a zero-knowledge-proof or by a challenge response requested by the CA.

Comparable Key Sizes. ECDSA keys are significantly smaller than RSA keys providing the same security level. In elliptic curve cryptography, key size usually refers to the field size. Table 2 shows comparable key sizes of symmetric cryptography, RSA and ECDSA that offer the same security strength. The values are based on NIST SP 800-57 [31]. Although ECDSA keys still have to be at least twice as long as symmetric keys in order to offer the same security, it is obvious that the keys of ECDSA are significantly smaller than the keys of RSA that offer the same security. Based on the recommendations of NIST, ECDSA with key sizes between 160 to 223 bits provide a bit security of 80 bits. This implies that ECDSA over binary fields or prime fields with key sizes of 160 to 223 bits achieve a similar bit security of 80 bits. The necessary key sizes to achieve bit securities of 80, 112, 128, 192 and 256 can also be obtained from Table 2.

Recommended Curves. FIPS PUB 186-4 published by NIST [30] recommends ECDSA with elliptic curves over prime fields as well as ECDSA with elliptic curves over binary fields and Koblitz curves for governmental use. However, the recommendation of elliptic curves by the German ECC-Brainpool [27], RFC5349 for use of ECC in Kerberos [46], RFC6460 for use of ECC in TLS [36] and the

Table 2. Comparable Key Sizes (in bits) of Symmetric Algorithms (equal to security bits), RSA and ECDSA based on NIST SP 800-57 [31]

Symmetric (security bits)	RSA	ECDSA
80	1024	160-223
112	2048	224-255
128	3072	256-383
192	7680	384-511
256	15360	512+

Fact Sheet Suite B Cryptography written by NSA [32] do not include any curves over binary fields or Koblitz curves. These curves are only recommended by NIST FIPS PUB 186-4 [30] and Standards for Efficient Cryptography, SEC 2 [2]. Therefore, they do not seem to satisfy the highest security conditions anymore.

Table 3. Recommended Key Sizes (in bits) for ECC based on NIST [30], SEC 2 [2], ECC-Brainpool [27], RFC5349 [46], RFC6460 [36] and NSA [32]

	Curves over Prime Fields								Curves over Binary Fields or Koblitz Curves				
NIST [30]		192	224	256		384		521	163	233	283	409	571
SEC 2 [2]		192	224	256		384		521	163	233	283	409	571
ECC-Brainpool [27]	160	192	224	256	320	384	512						
RFC5349 [46]				256		384							
RFC6460 [36]				256		384							
NSA [32]						≥ 384							

All in all, one can say that ECDSA offers the same level of security as RSA with lower key size since the hardness of RSA is based on the prime factorization problem that can be broken in sub-exponential time, whereas the hardness of ECDSA depends on the ECDLP that can only be broken in fully-exponential time. It is important to say that the factorization problem and ECDLP are upper limits. An attacker is not necessarily able to solve the prime factorization problem if he is able to break RSA. This is also applicable to ECDSA and ECDLP.

3.5 Attacks

One of the most famous attacks on ECDSA is probably the hack of the Sony Playstation 3 that was presented on the 27th Chaos Communication Congress Console Hacking in 2010 [5]. However, this attack could have been prevented easily. Sony did not follow the necessary security conditions introduced by Vaudenay [40] that have been explained in Sect. 3.4. The ephemeral key k that was used in the Playstation 3 was not unpredictable since Sony was using a fixed value for k. Therefore, it was easy to recover the secret key d.

Attack on Weak Pseudo-Random Generator for k. The basics of the attack on the Playstion 3 are explained by Sarath et al. [37]. They showed that it is possible to recover d if the same k is used to generate two different signatures (r, s_1) and (r, s_2) for two messages m_1 and m_2. It is possible to calculate s_1 and s_2 by means of $s_1 = k^{-1}(h_1 + dr)$ and $s_2 = k^{-1}(h_2 + dr)$, where $h_1 = \mathcal{H}(m_1)$ and $h_2 = \mathcal{H}(m_2)$ with $\mathcal{H}$ denoted as the hash function. Thus, it is possible to compute $ks_1 - ks_2 = h_1 + dr - h_2 - dr$ and $k = \frac{h_1 - h_2}{s_1 - s_2}$. This leads to the equation $d = \frac{ks-h}{r}$. Thus, the pseudo-random generator must not be weak. It must not generate the same k for two different messages.

Side-channel Attacks. Braun and Kargl [1] presented a doubling attack to compute the secret key d. Their attack is based on the addition formula of two points being slightly different from the point duplication in elliptic curves. It is possible to perform this attack by measuring the required time since point doubling is supposed to be faster than point addition. Thus, a verifier is able to detect if $t_1 P = t_2 Q$ in step 5 of the signature verification algorithm of ECDSA. Hence, this yields to $t_1 = t_2 d$ and it is easily possible to compute the secret key d. Nevertheless, this attack is only possible in the very special and improbable case that $t_1 P = t_2 Q$. This attack can also be prevented easily. The signer only has to validate that $t_1 P \neq t_2 Q$ which implies that $h \neq rd$.

Brumley et al. [3] also introduced a timing attack against elliptic curve over binary fields. Their attack recovers the private key by exploiting implementations that do not run in constant time. They assume that the Montgomery's ladder is used for scalar multiplication of points on elliptic curves over binary fields. This ladder implementation introduces a timing attack vulnerability since there is a direct correlation between the time needed to compute a scalar multiplication and the logarithm of k. Their attack consists of two phases. In the first phase, an attacker is collecting a filtered set of signatures based on the time correlation of scalar multiplication and logarithm of k. In the second phase, the attacker is able to perform a lattice attack on the filtered set collected in the previous phase. Lattices describe mathematical objects that are often used in cryptography. One application field of lattices is to attack schemes with partially known secret data. It is possible to find small solutions of underdetermined systems of equations by means of lattices. The repeated use of this lattice attack might lead an attacker to be able to recover the corresponding private key of the signature. Brumley et al. are assuming that an attacker is able to collect enough ECDSA signatures generated with the same ECDSA key. Additionally, an attacker must be able to measure the running time of the sign operation for each signature.
Another type of side-channel attacks that are applicable to ECDSA are template attacks. Template attacks also consist of two phases. First, a possible attack has to build templates of the device under attack, for instance statistical models.

Then, the attacker matches the templates with the traces gained from the attacked device. In 2008, Medwed and Oswald [28] presented a template-based simple power analysis attack against ECDSA. They combined a template attack with a power analysis attack. In their attack, only a few bits of the ephemeral key k are needed in order to get the ECDSA secret key d.
Obviously, it is also possible to run power analysis attacks that are not based on template attacks. Hutter et al. [21] constructed two differential power analysis attacks in which they are able to gain the private key during the signature generation. Their first attack performs a power analysis, while the second attack is based on an electromagnetic analysis.

Fault Attack. In 2009, Schmidth and Medwed [38] introduced a fault attack on ECDSA. By means of manipulating the program flow, it is possible the compute some bits of the ephemeral key k. This is already enough for a lattice attack to be applied to recover the private signature key. A possible countermeasure against that attack is to introduce a check value in order to prevent the device from releasing fault values. Schmidth and Medwed achieved this by changing the point representation in ECDSA.

3.6 Performance

We show a comparison of ECDSA with RSA on three resource-constrained devices. The running times of Table 4 are based on three different sources. Westhoff et al. [43] are measuring the running time of ECDSA over binary fields and RSA on a Sharp Zaurus SL-5500G Personal Digital Assistant operating on Linux with an integrated Intel SA-1110 StrongARM CPU running on 206 MHz. They are using OpenSSL (Developer Snapshot 20021202) to measure the running times for RSA and their own speed optimized ECDSA version. They are comparing ECDSA over binary fields on field sizes from 113 to 233 bits with RSA on comparable key sizes.
The running times of the Ultra-80 and Yopy devices are taken from the paper of Gupta et al. [16]. Yopy is a Linux Personal Digital Assistant with a 200 MHz StrongARM CPU and Ultra-80 is a Sun server with an integrated 450 MHz UltraSPARC II CPU. They are using the OpenSSL0.9.6b speed program to measure the ECDSA and RSA running times. They had to enhance the OpenSSL0.9.6b speed program on their own in order to be able to measure ECDSA. On both devices, only elliptic curves over binary fields with a field size of 163 and 193 bits are measured.
The running times of supercop-20140622 are taken from the eBATS project [13]. Supercop-20140622 is a Cortex-A8 2011 TI Sitara AM3359 with 720 MHz built in the armeabi architecture. The eBATS project uses the OpenSSL implementation of RSA and ECDSA with their own wrapper. In their implementations, RSA performs message recovery. Originally, the running times are measured in *cycles*. In order to easily compare them against the running times of the other

two papers, we have transformed them into ms by means of the equation

$$runTime = cycles * \frac{1}{frequency * 10^3} \ , \qquad (6)$$

where $frequency$ is given in hz. The eBATS project includes the most complete and recent comparison between ECDSA and RSA. They measured ECDSA with binary fields and prime fields with field sizes from 160 to 571 bits and RSA with key sizes from 1024 to 4096 bits. Additionally, the eBATS project measured ECDSA with Koblitz curves. These implementations are approximately 10% faster than ECDSA over binary fields. This time difference increases slightly with bigger key sizes. However, they are not shown in Table 4 for the sake of better readability.

Table 4. Performance of ECDSA over binary and prime fields and RSA on different resource-constrained processors

Device	Key/Field Size (bits)			Signature Generation (ms)			Signature Verification (ms)		
	bin. Field	prim. Field	RSA	bin. Field	prim. Field	RSA	bin. Field	prim. Field	RSA
Sharp Zaurus SL-5500G [43]	113		512	2.8		13.7	7.5		1.3
	131		704	3.8		32.4	11.5		2.5
	163		1024	5.7		78.0	17.9		4.3
	193		1536	7.6		251.9	26.0		9.7
	233		2240	10.1		731.8	37.3		20.4
Ultra-80 [16]	163		1024	6.8		32.1	13.0		1.7
	193		2048	9.2		205.5	18.1		6.1
Yopy [16]	163		1024	24.5		188.7	46.5		10.8
	193		2048	39.0		1273.8	76.6		39.1
supercop-20140622 [13]			512			2.7			0.2
			768			5.9			0.2
	163	160	1024	5.3	3.2	10.9	10.2	3.7	0.3
		192	1024		4.4	10.9		5.1	0.3
			1536			29.0			0.5
	233	224	2048	9.6	5.7	61.2	18.5	6.7	0.8
	283	256	3072	17.5	7.5	183.5	30.8	8.8	1.5
			4096			412.4			2.5
	409	384	7680	38.8	18.7		76.4	22.3	
	571	521	15360	89.7	41.8		178.3	49.5	

On the Sharp Zaurus SL-5500G, ECDSA is up to 72.5 times faster than RSA-2240 on the biggest measured key size of 233 bits. Even with a key size of 113 bits, ECDSA is still 4.9 times faster than RSA on the smallest measured key size. Gupta et al. only measured the running times with key sizes of 163 and 193 bits on Ultra-80 and Yoppy. In their measurement, ECDSA is respectively 4.7 and 22.3 times faster than RSA on Ultra-80. The performance difference of ECDSA and RSA is slightly higher on Yoppy, with ECDSA being up to 32.7 times faster. The benchmark of supercop shows an improved version of RSA

that uses message recovery, which is why the time ratio of ECDSA to RSA is significantly smaller. ECDSA over binary curves is only up to 10.5 times faster. ECDSA over prime curves is up to 24.5 times faster. All measurements show that the signature generation in ECDSA is noticeably faster than in RSA. This is based on the significant smaller key size that ECDSA is using.
The signature verification is the drawback of ECDSA. RSA is faster in all measurements. The running times of supercop show the biggest difference. In these measurements, RSA is up to 34 times faster when ECDSA is performed over binary fields with a 163 bits field size. Nevertheless, ECDSA over prime fields is significantly faster. It is only a maximum of 12.9 times slower than RSA. The time difference is much slower on the other three devices since they are using speed optimized versions of ECDSA. Apparently, they are also using older implementations of RSA because RSA is significantly slower on their devices compared to supercop.
The running times of signature generation and verification performed on supercop are supposed to be the most relevant since it is the most recent benchmark that is using current RSA and ECDSA implementations. Furthermore, only our comparison based on supercop is following the recommendation of comparable key sizes by NIST SP 800-57 that is shown in Table 2. Thus, supercop is almost always using bigger field sizes. That is another reason why the performance difference between RSA and ECDSA is smaller on supercop.

Table 5. Signature length comparison (in bits) of RSA and ECDSA over prime and binary fields on different bit security levels measured by the eBATS project [13]

Bit security	RSA-l	ECDSA-bin p	ECDSA-prime b
80	1024 $[l = 1024]$	336 $[b = 163]$	320 $[b = 160]$
112	2048 $[l = 2048]$	480 $[b = 233]$	448 $[b = 224]$
128	3072 $[l = 3072]$	576 $[b = 283]$	512 $[b = 256]$

In addition, Table 5 shows the signatures lengths of RSA and ECDSA over binary and prime fields on bit security levels of 80, 112 and 128 bits. The signatures of ECDSA over prime fields are a bit shorter than the one generated by ECDSA over binary fields. However, ECDSA signatures are in general significantly smaller than RSA signatures. On 128 security bits, the ECDSA signatures are more than five times smaller than RSA signatures.

All in all, the measurements show that ECDSA is better than RSA in terms of key size and signature generation. However, RSA is much faster in signature verification. If it is required to have a compromise between key size, signature generation and signature verification, ECDSA is the better choice since it is able to use much smaller keys and the time differences in the signature generation are much higher than in the signature verification step. For instance, ECDSA with a key size of 233 bits takes 9.6ms to generate a signature, while RSA-2048 takes

61.2ms. In the signature verification, ECDSA-233 takes 18.5ms and RSA-2048 needs 0.8ms. In total, ECDSA-233 takes 28.1ms and RSA-2048 needs 62.0ms to perform both operations. However, signatures are usually created once and may be verified often. In an IoT scenario, signatures may be created by small devices but verified by big servers that have access to special cryptographic hardware. For that reason, the performance for signature generation might be more important on resource-constrained devices in an IoT scenario. Hence, ECDSA is the better choice for resource-constrained devices.
It is important to say that prime fields should not always be faster than binary fields. In general, ECDSA over binary curves is supposed to be even faster than ECDSA over prime curves because binary curves have shorter formulas and binary squaring is usually very cheap. The reason why ECDSA over prime curves are much faster on supercop-20140622 is due to the processor architecture. General purpose processors usually have a giant integer multiplier circuit that can compute arithmetic operations on prime curves very quickly.

Table 6. Time Measurements of the ECDSA Variants in seconds performed on Pentium(R) Dual-Core CPU with 2.30 GHz and a 192-bit key [37]

	Key Generation	Signature Generation	Verification
ECDSA [37]	78 ms	93 ms	125 ms
ECGDSA [37]	83 ms	78 ms	125 ms
ECKCDSA	$\approx$ 83 ms	$\geq$ 78 ms	$\geq$ 125 ms

Furthermore, a performance comparison of ECDSA and its two variants, ECGDSA and ECKCDSA, is presented. Table 6 shows the times taken for key generation, signature generation and signature verification for ECDSA and ECGDSA, based on the performance evaluation of Sarath et al. [37]. They are using a Pentium(R) Dual-Core CPU with 2.30 GHz and a key size of 192 bits. It is significant that the signature generation of ECGDSA is about 16% faster than the signing algorithm of ECDSA. The running times of the signature verification of both algorithms are almost the same and the key generation of ECGDSA is about 6% slower. The time values of ECKCDSA are not measured, but estimated by us. We assume that the key generation performance of ECGDSA and ECKCDSA is similar since they are using the same algorithm. However, ECKCDSA is supposedly slightly slower in signature generation and verification because of the additional bits of the certificate in the hash function.

Hitchcock et al. [20] showed that it is possible to implement ECDSA effectively and compactly in hardware. They implemented an efficient ECDSA algorithm over prime fields on a smart card. In general, ECC can be implement more effectively in hardware than RSA. Table 7 shows the minimum number of gates required to implement RSA and ECC in hardware for 80 and 128 security bits presented by Zhang et al. [44]. ECC requires up to ten times less gates than RSA. Thus, ECC can be implemented on smaller chips that generate less heat

Table 7. Minimum number of gates required to implement ECC and RSA in hardware with 80 and 128 security bits

	80 security bits		128 security bits	
Algorithm	RSA-1024	ECC-163	RSA-3072	ECC-283
Gate count	34,000	3,260	50,000	6,660

and consume less power [23] than bigger RSA chips. Additionally, it is possible to implement ECDSA effectively in software. All in all, ECDSA is best suitable for IoT devices with low bandwidth, low computing power and little memory.

4 QUARTZ

QUARTZ, submitted to the European NESSIE project by Patarin and Courtois 2000, is a multivariate signature scheme. It is based on HFE (Hidden Field Equations) or, to be more precise, HFEv-, where the perturbation operations "v" and "-" are added.

HFEv- has two independently adjustable security parameters d and n, so that a signature length of 128 bits can be achieved by choosing a fixed, small n, while d can be adjusted to achieve the desired security level. Additionally, QUARTZ offers long-term security since it is quantum computer resistant [8]. To avoid the birthday paradox problem in QUARTZ, there are four iterations of signature generation (for messages m, $H(m||0x00)$, $H(m||0x01)$ and $H(m||0x02)$) and accordingly four iterations during the verification process.

4.1 Multivariate Cryptography

In this section, a quick explanation is given of how multivariate cryptography based on the MQ-Problem and especially such public key cryptosystems work. For a more comprehensive description see [11], [42] or [45].
Like RSA or ECC, multivariate encryption and signature schemes are asymmetric. However, unlike them they do not depend on assumptions that break as soon as quantum computer exist. Therefore, they offer long-term security. To do so, they work with systems of multivariate quadratic polynomials and are based on the so called MQ-Problem.

MQ-Problem. Let $p_1(\mathbf{x}), ..., p_m(\mathbf{x})$ be m multivariate quadratic polynomials with n variables $x_1, ..., x_n$. The goal is to find a common zero $\mathbf{x_0}$ of the polynomials $p_1, ..., p_m$, i.e. a solution $\mathbf{z} = (z_1, ...z_n)$ such that $p_1(\mathbf{z}) = ... = p_m(\mathbf{z}) = 0$. The MQ-problem is proven to be NP-hard for $m \approx n$.

Public Key Cryptosystem. For multivariate encryption and signature schemes, there has to be an easily invertible multivariate quadratic map called "central map". Let $\mathcal{G} : \mathbb{K}^n \mapsto \mathbb{K}^m$ be this map, then there are two affine transformations

$\mathcal{S}$ and $\mathcal{T}$, such that a new multivariate quadratic map $\mathcal{F} : \mathbb{K}^n \mapsto \mathbb{K}^m$ can be computed as

$$\mathcal{F} = \mathcal{T} \circ \mathcal{G} \circ \mathcal{S} \ . \tag{7}$$

This hides the structure of $\mathcal{G}$, so that to anyone, who doesn't know $\mathcal{S}$ and $\mathcal{T}$, the map $\mathcal{F}$ looks like a random map. Therefore, $\mathcal{F}$ being a trapdoor one-way function is used as the public key, while $\mathcal{T}$, $\mathcal{G}$ and $\mathcal{S}$ together form the private key. [11,45]

For QUARTZ, $\mathcal{G}$ is chosen in a special way being a member of the BigField family. This means $\mathcal{G}$ is an easily invertible map over a degree n extension Field $\mathbb{E}$ of $\mathbb{K}$. To get a quadratic map $\bar{\mathcal{G}} : \mathbb{K}^n \mapsto \mathbb{K}^n$, an isomorphism $\varphi : \mathbb{K}^n \mapsto \mathbb{E}$ is used:

$$\bar{\mathcal{G}} = \varphi^{-1} \circ \mathcal{G} \circ \varphi \ . \tag{8}$$

Therefore, the hiding of the central map and thus the public key of the scheme looks like

$$\mathcal{F} = \mathcal{T} \circ \bar{\mathcal{G}} \circ \mathcal{S} = \mathcal{T} \circ \varphi^{-1} \circ \mathcal{G} \circ \varphi \circ \mathcal{S} \ . \tag{9}$$

Sign: Let $m \in \mathbb{K}^n$ be the message to be signed. The signature is then computed as follows:

$$\sigma = \mathcal{S}^{-1}(\varphi^{-1}(\mathcal{G}^{-1}(\varphi(\mathcal{T}^{-1}(m))))) \in \mathbb{K}^n \ . \tag{10}$$

For a schematic presentation see Fig. 1.

Verify: Let $\sigma \in \mathbb{K}^n$ be a signature supposed to belong to the message $m \in \mathbb{K}^n$. To verify the signature, it is checked if $m = m'$, where $m' = F(\sigma) \in \mathbb{K}^n$. If it holds, the signature is accepted, otherwise rejected. For a schematic presentation see Fig. 1.

sign

$$\begin{tikzcd}
 & \mathbb{E} \arrow[r, "\mathcal{G}^{-1}"] & \mathbb{E} \arrow[d, "\varphi^{-1}"] & \\
\mathbb{K}^n \arrow[r, "\mathcal{T}^{-1}"] & \mathbb{K}^n \arrow[u, "\varphi"'] \arrow[r, "\bar{\mathcal{G}}^{-1}"] & \mathbb{K}^n \arrow[r, "\mathcal{S}^{-1}"] & \mathbb{K}^n \arrow[lll, bend left, "\mathcal{F}"]
\end{tikzcd}$$

verify

Fig. 1. Schematic representation of sign and verify in BigField multivariate signature scheme, derived from [11, Fig. 1]

Minus: The "minus" perturbation operation removes a small number of equations from the public key to prevent linearization attacks. [34]

Vinegar: The "vinegar" perturbation operation adds additional variables to the central map to hinder direct attacks.

4.2 Parameters

QUARTZ being a special case of HFEv- has the same parameters. More specifically QUARTZ is a HFEv- scheme with a special choice of parameters.

$$(\mathbb{K}, n, d, r, v) = (GF(2), 103, 129, 3, 4) \ , \tag{11}$$

where $\mathbb{K}$ is the underlying finite field, n is the size of the extension field, d is the degree of the hidden polynomial, r is the number of equations removed and v is the number of vinegar variables.

In QUARTZ, $\mathbb{E} = \mathbb{L} = \mathbb{K}^{103}$ is used as extension field of $\mathbb{K}$, like described in Sect. 4.1, more specifically it is defined by $\mathbb{L} = \mathbb{K}[X]\left(X^{103} + X^9 + 1\right)$.

4.3 Algorithm Specifications

The algorithm itself is described by Courtois et al. [33]. There is also an updated version [9].

Let the definitions from Sect. 4.2 apply. Furthermore, let φ be the bijection between $\mathbb{K}^{103}$ and $\mathbb{L}$:

$$\varphi(x_0, ..., x_{102}) = \sum_{0}^{102} \left(x_i * X^i\right) \quad \left(\text{mod } X^{103} + X^9 + 1\right) \ . \tag{12}$$

The central map is defined by a family of secret functions $\mathcal{F}_V(Z) : \mathbb{L} \mapsto \mathbb{L}$ with $V \in \mathbb{K}^4$:

$$\mathcal{F}_V(Z) = \sum_{0 \le i < j < 103}^{2^i + 2^j \le 129} \alpha_{i,j} * Z^{2^i + 2^j} + \sum_{0 \le i < 103}^{2^i \le 129} \beta_i(V) * Z^{2^i} + \gamma(V) \ , \tag{13}$$

where $\alpha_{i,j} \in \mathbb{L}$, $\beta_i : \mathbb{K}^4 \mapsto \mathbb{L}$ being affine transformations and $\gamma : \mathbb{K}^4 \mapsto \mathbb{L}$ being a quadratic transformation.

To hide the structure of the central map in the public key, there are two affine transformations $\mathcal{S} : \mathbb{K}^{107} \mapsto \mathbb{K}^{103}$ and $\mathcal{T} : \mathbb{K}^{103} \mapsto \mathbb{K}^{100}$. The public key $\mathcal{P} : \mathbb{K}^{107} \mapsto \mathbb{K}^{100}$ is then a quadratic map defined by

$$\mathcal{P} = \mathcal{T} \circ \varphi^{-1} \circ \mathcal{F}_V(Z) \circ \varphi \circ \mathcal{S} \ , \tag{14}$$

whereas the private key consists of $\mathcal{T}$, $\mathcal{F}_V(Z)$ and $\mathcal{S}$.

Signature Generation. Given a message M, the corresponding signature is calculated as follows:

First, four 100-bit strings H_1, H_2, H_3, H_4 are built from M by computing

$$\begin{aligned} M_0 &= H(M) \\ M_1 &= H(M_0||0x00) \\ M_2 &= H(M_0||0x01) \\ M_3 &= H(M_0||0x02) \ , \end{aligned} \tag{15}$$

where H is some hash function outputting 160 bits, and then picking H_1, H_2, H_3, H_4 from $\bar{M} = M_1||M_2||M_3$ such that $\bar{H} = H_1||H_2||H_3||H_4$ are the first 400 bits of $\bar{M}$ and each H_i is exactly 100 bits long. This procedure is represented in a more visual way in the upper part of the schematic representation of signature generation in Fig. 2.

Then, the actual signature process is applied to every H_i $(1 \leq i \leq 4)$ successively (see cycle in middle part of Fig. 2). Therefore, a pre-image of $Y = H_i \oplus \tilde{S}$ under $\mathcal{T}$ is computed, where $\tilde{S}$ is a 100-bit string, which is initialized to 0 and overwritten after every iteration. The result is then transformed into $\mathbb{L}$ by applying φ.

$$B = \varphi(\mathcal{T}^{-1}(Y)) \in \mathbb{L} \ . \tag{16}$$

After that, values for vinegar variables $\mathbf{V} = (v_1, v_2, v_3, v_4) \in \mathbb{K}^4$ have to be chosen at random or, like it is done in [33], by picking specific bits from the hash of Y and a random 80-bit string. The univariate polynomial equation in Z

$$\mathcal{F}_V(Z) = B \ , \tag{17}$$

where V is the chosen $\mathbf{V} = (v_1, v_2, v_3, v_4) \in \mathbb{K}^4$, has to be solved for example by Berlekamp's algorithm [11]. If no solution is found, new vinegar variables are chosen again at random (or by re-hashing and picking again) and the calculation goes back to step (17). Otherwise, let $A_1, ..., A_\delta \in \mathbb{L}$ be the found solutions, the solution A is then the one with the smallest hash.

The intermediate result of one iteration is given by transforming A back into $\mathbb{K}^{107}$ using φ^{-1} and then applying $\mathcal{S}^{-1}$ to the result.

$$X = \mathcal{S}^{-1}(\varphi^{-1}(A)) \in \mathbb{K}^{107} \ . \tag{18}$$

The new value of $\tilde{S}$ for the next iteration consists of the first 100 bits of X, while the subsequent 7 bits form X_i, which contributes to the final signature.

The final signature is composed of the $\tilde{S}$ of the last iteration and all X_i as it can be seen in the lower part of Fig. 2:

$$\sigma = \tilde{S}||X_4||X_3||X_2||X_1 \ . \tag{19}$$

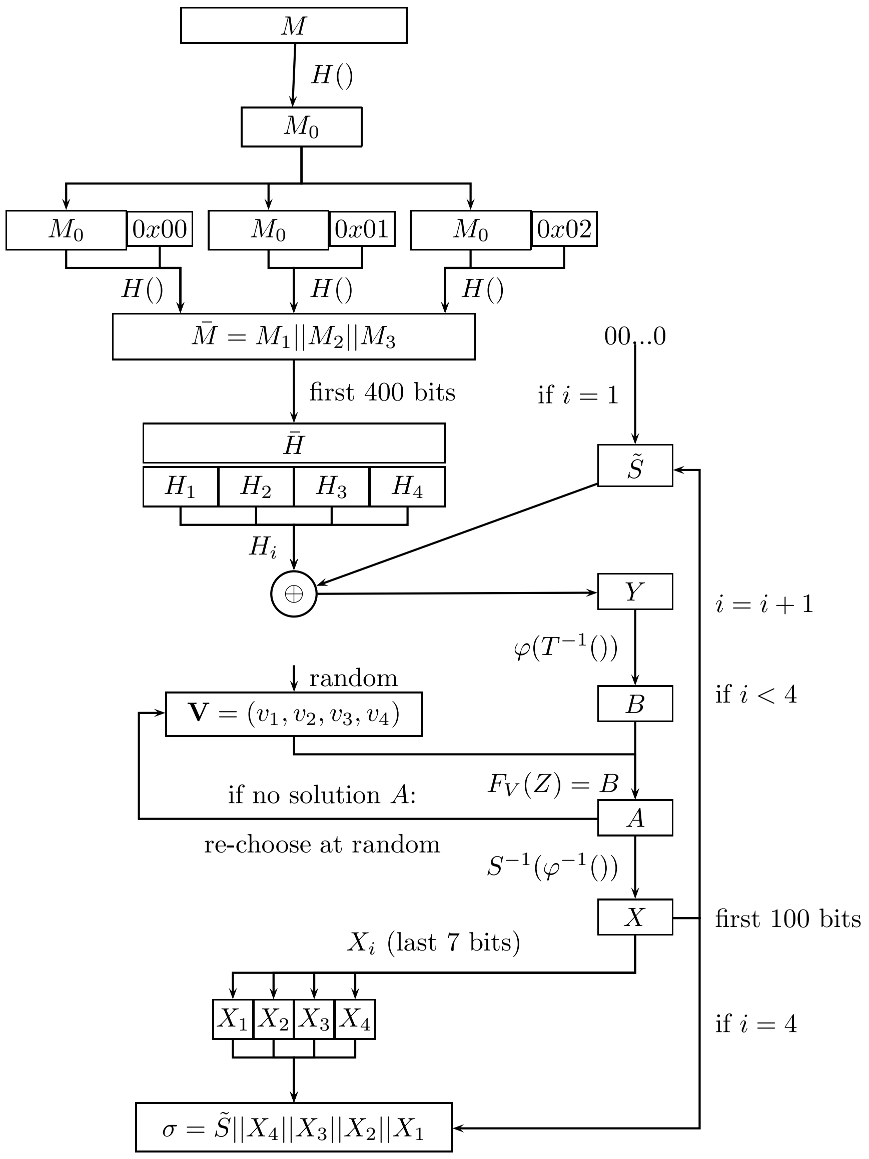


Fig. 2. Schematic representation of signature generation in QUARTZ, derived from [9, Fig. 1]

Signature Verification. Let σ be the 128-bit signature that supposedly belongs to the message M. To verify the signature as described in Sect. 4.1, it has to be checked if the public key applied to σ results in M.

In the signature generation of QUARTZ, there are four iterations to create a signature and so are there for signature verification, analogously. Therefore, let H_1, H_2, H_3, H_4 be defined as in Sect. 4.3. Furthermore, let $\tilde{S}$, X_1, X_2, X_3 and X_4 be defined such that $\tilde{S}||X_4||X_3||X_2||X_1 = \sigma$, where $\tilde{S}$ is 100 bits long and X_i consists of 7 bits each.

For each iteration over X_i from $i = 4$ down to $i = 1$, calculate the 100-bit string U defined as

$$U = \mathcal{P}(U_{old}||X_i) \oplus H_i \ , \tag{20}$$

where U_{old} is equal to the U of the last iteration and is initialized to $U_{old} = \tilde{S}$ for the first iteration[4].

The signature σ is then accepted if U after the last iteration equals the 100-bit string of zeros 00...0, otherwise it is rejected.

The schematic representation of signature verification is shown in Fig. 3. When comparing signature generation (Fig. 2) and verification (Fig. 3), it is easy to see, that the preparation part at the top is the same and the main part of each of them consists of a cycle which is executed 4 times. In signature verification however, the procedure counts backwards from 4 to 1 to verify the steps of signature generation in correct order.

4.4 Security

Even though QUARTZ can't be proven to be equivalent to any well defined and known problem, there are indicators, which implicate reasonable confidence in its security.

QUARTZ is based on the MQ-problem which is known to be NP-hard [42]. Therefore, it is difficult in worst-case and even more, it seems to be difficult in average, since there are no known algorithms for solving the MQ-problem significantly better than exhaustive search when the number of equations is approximately equal to the number of variables. The hardest instances of the MQ-problem currently known are random systems of quadratic equations. If the degree d of the hidden polynomial $\mathcal{F}$ increases, the trapdoor fades away, converging to a random system of quadratic equations at $d \rightarrow q^n$, where q is the number of elements of the underlying finite field (in QUARTZ $q = 2$). Thus d chosen to be rather large in QUARTZ contributes to its security. [8, 33]

In the initial publication of QUARTZ, Courtois [33] claims a bit security of 2^{80} for QUARTZ. Later, however, he investigates the security with respect to so-called Gröbner bases [8] and comes to the conclusion, that to achieve a security level of at least 2^{80}, the security parameter d has to be increased from 129 to 257 (see end of this section).

[4] reminder: $\mathcal{P}$ is the public key defined as $\mathcal{P} = \mathcal{T} \circ \varphi^{-1} \circ \mathcal{F}_V(Z) \circ \varphi \circ \mathcal{S}$

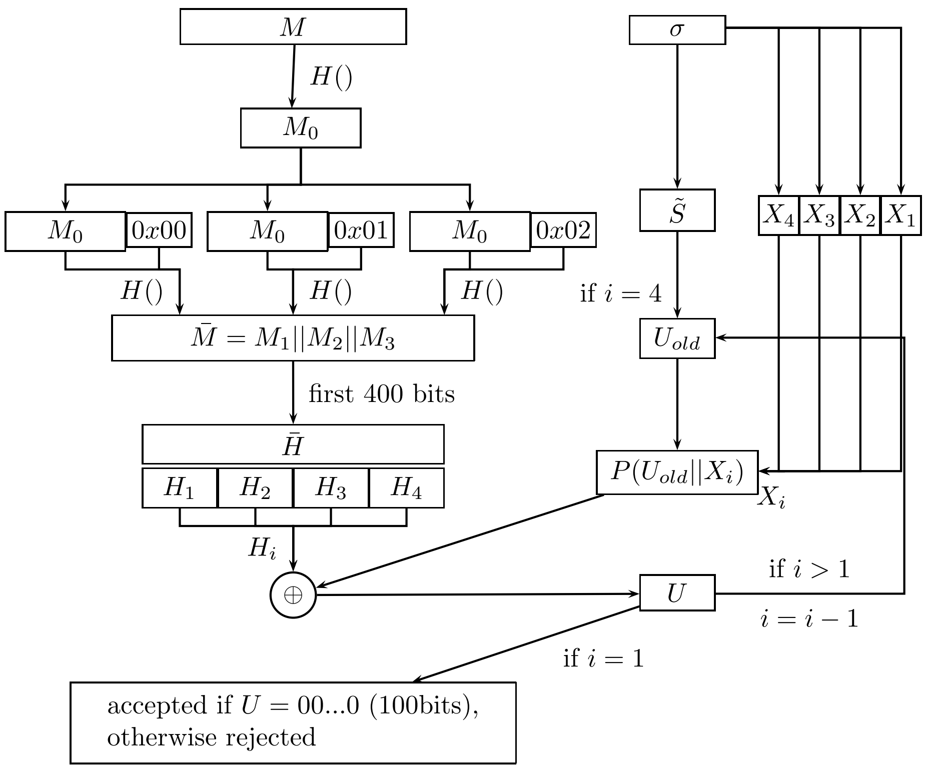


Fig. 3. Schematic representation of signature verification in QUARTZ, derived from [9, Fig. 2]

Even though the signature generation process of QUARTZ looks complex compared to the underlying HFEv-, breaking QUARTZ is equivalent to breaking HFEv- [11]. So attacks on the underlying basic HFE scheme may also apply to QUARTZ. There are many known attacks on the basic HFE scheme, either by recovering the secret key or by inverting the trapdoor function directly. Examples for methods recovering the secret key are the original Shamir-Kipnis attack [24] and the improved version by Courtois [7], which solves the MinRank problem (cf. Sec. 4.5) more efficient. These give the complexities $n^{O(\log_q^2 d)}$ and $n^{3\log_q d+O(1)}$, respectively. Ding et al. [11] states a complexity of $O\left(q^{n*(\lfloor \log_q d-1\rfloor+1+v+r-1)} * (n-r)^3\right)$ for solving the MinRank problem. The direct attack by Courtois as an example for inverting the trapdoor function directly is in $n^{\frac{3}{2}*\log_q d+O(1)}$ [7]. Furthermore, there are attacks that try to solve polynomial systems by computing so-called Gröbner bases with Faugère's attack using the F5/2 algorithm [14,39], currently being the most efficient. See Sec. 4.5 for further information about the direct attack, Gröbner bases and the F5/2 algorithm.

There are no known specific attacks on HFEv-. However, the direct attack by Courtois and all attacks using Gröbner bases can be applied to HFEv- without changes. Nevertheless it seems the perturbations indeed increase the security, since even the best known attack on basic HFE by Faugère is unsuccessful because of the increased complexity [8].

Influence of Perturbation Operations. Courtois et al. [8] made several observations regarding the influence of perturbation operations "minus" and "vinegar" on pure HFE systems, especially for those of low degree d:

– Both operations have similar effects concerning randomness added to the original basic HFE system.
– Nevertheless, there are differences concerning the total time of attacks. While the "vinegar" operation increases the time of currently known attacks, the "minus" operation may even decrease the time of attacks solving systems, since it reduces the size of the system to solve. However, it should not be omitted, because this operation helps against other attacks like the Patarins attack on Matsumoto-Imai.
– It might be better for a fixed number of perturbations to use many of type "vinegar" and few "minus" perturbations, so that there is a mixture of both, while using the advantage of "vinegar".

This supports the choice of parameters $r = 3$ and $v = 4$ in QUARTZ, even though there should be more "vinegar" perturbations according to the last statement. However, there are restrictions [9, 33] in QUARTZ preventing an alteration of these choices.

Special Case of Signing on Gröbner Bases. Gröbner bases can be seen like a Gaussian elimination procedure for multivariate, non-linear systems. Applying

Gröbner bases to HFEv- in the special case of signing allows a simplification (concerning the amount of possible solutions) by fixing some variables. For a more detailed description see Sec. 4.5. Faugère's attack using the F5/2 algorithm (being the currently best attack on basic HFE) applied to QUARTZ gives an extrapolated $O(n^{10}/4)$. The working factor $WF(HFEv-)$ including perturbations is calculated by

$$WF(HFEv-) \approx \frac{2^7 * (n-3)^{10}}{4} \approx 2^{71} \ , \tag{21}$$

which has to be done 4 times in QUARTZ to forge a signature. Because of the repeating method of fixing variables, there is an additional factor of 1.6 on average, see Sect. 4.5. Therefore, the working factor for QUARTZ $WF(QUARTZ)$ is calculated as follows:

$$WF(QUARTZ) = 4 * 1.6 * WF(HFEv-) \approx 2^{74} \ . \tag{22}$$

This is equivalent to approximately 2^{65} Triple-DES computations[5]. To ensure a security level of at least 2^{80}, Courtois et al. [8] proposes that the degree d of the hidden polynomial F in QUARTZ should be changed from 129 to 257. This would result in a working factor $WF(QUARTZ_{257})$ of

$$WF(QUARTZ_{257}) \ = 2^{87} \ , \tag{23}$$

which is equivalent to approximately 2^{78} TDES computations. [8]

As a result, Courtois et al. recommend to increase the parameter d in QUARTZ from 129 to 257 to achieve the desired security level.

4.5 Attacks

The attacks on QUARTZ can be categorized into 3 main groups. [33]

First, attacks trying to recover the secret key, which are mainly attacks on the underlying basic HFE scheme. This includes for example exhaustive search and the Shamir-Kipnis [24] attack or MinRank [7] attack.

Second, direct attacks targeting the trapdoor itself, i.e. inverting the trapdoor, to compute a signature directly only using the public key. This results in the problem of solving an instance of the MQ problem, which is NP-hard for random instances [42]. However, there are methods like Gröbner bases, XL [10] or FXL [10] designed to solve this problem. The currently most efficient attack of this category is Faugère's attack using the F5/2 algorithm [14, 39].

Third, attacks like affine multiple attacks or higher degree attacks, which try to exploit detected differences of the public key compared to a general system of quadratic equations.

The most important attacks on QUARTZ are the MinRank attack and direct attacks by inverting the trapdoor-function, especially by using Gröbner bases, which are thus described hereinafter.

[5] Measurement unit in the NESSIE project

MinRank attack. [7, 11, 24] The MinRank problem is, given a field $\mathbb{K}$, two integers $m, n \in \mathbb{N}$, a value $r < n$ and m matrices $M_1, ..., M_m$ of dimension $n \times n$ over $\mathbb{K}$, find a linear combination $\alpha \in \mathbb{K}^m$ of small rank $rank(\sum_i \alpha_i M_i) \leq r$.

The MinRank attack is an improvement of the Shamir-Kipnis attack (cf. [24]) using methods to solve the underlying MinRank problem in the second step directly instead of using relinearization. The main idea in this attack is to lift the maps $\mathcal{T}$, $\mathcal{S}$ and the public key (also a map) $\mathcal{P}$ to the extension field $\mathbb{L}$, called $\mathcal{T}^*$, $\mathcal{S}^*$ and P^* hereinafter. The function $\mathcal{P}^*$ can then be represented as the quadratic form

$$\mathcal{P}^*(X) = \underline{X} G \underline{X}^T \ , \tag{24}$$

where the matrix G consists of coefficients of the representation of the public key $\mathcal{P}$ as univariate polynomial and $\underline{X} = \left(X^{2^0}, X^{2^1}, ..., X^{2^{n-1}}\right)$. Since $\mathcal{T}$ and $\mathcal{S}$ are linear maps, $\mathcal{T}^*$ and $\mathcal{S}^*$ can be represented as

$$\mathcal{S}^*(X) = \sum_{i=1}^{n-1} s_i * X^{2^i} \quad \text{and} \quad \mathcal{T}^*(X) = \sum_{i=1}^{n-1} t_i * X^{2^i} \ , \tag{25}$$

where $s_i, t_i \in \mathbb{L}$. From $\mathcal{P}^* = \mathcal{T}^* \circ \mathcal{F} \circ \mathcal{S}^* \Rightarrow \mathcal{T}^{*-1} \circ \mathcal{P} = \mathcal{F} \circ \mathcal{S}^*$ it is clear that

$$G' = \sum_{k=0}^{n-1} t_k G^{*k} = W * F * W^T \ , \tag{26}$$

where G^{*k} are the variants of G computed by raising entries of G to various powers and cyclically rotating its rows and columns, W is a $n \times n$ matrix and F is the $n \times n$ matrix representing the central map $\mathcal{F}$. Let $k = \lfloor \log_q d - 1 \rfloor + 1$, then, because only the top-left $k \times k$ entries of F are non-zero and $k \ll n$, the rank of G' collapses to at most $rank(G') \leq k + r + v$. Therefore, the coefficients t_k can be computed by applying a solver for the MinRank problem, as it is described by Courtois [7].

Direct attack. A direct attack on the trapdoor function of QUARTZ means to solve the equation

$$\mathcal{P}(\sigma) = m \ , \tag{27}$$

where $\mathcal{P}$ is the public key, m the message to sign and σ the signature to be found, directly. There are several methods to solve such a system like Gröbner bases or XL. Since Faugère's attack using the F5/2 algorithm, being the currently most efficient, is based on Gröbner bases, this method is presumably the most interesting.

Gröbner bases: [8,39] The Gröbner bases algorithm can be used to solve systems of equations. Let $1 \leq i \leq m$, $p_i \in GF(q)[x_1, ..., x_n]$ and $y_i \in GF(q)$ so that there are m polynomial equations

$$p_i(x_1, ..., x_n) = y_i \ . \tag{28}$$

This can be solved by calculating the set of all common zeros of the polynomials

$$\tilde{p_i} = p_i - y_i \tag{29}$$

over the algebraic enclosure. For that, a Gröbner basis can be used, which is a special kind of generating set for the ideal generated by the polynomials $\tilde{p_i}$. Gröbner basis can be seen as a kind of Gaussian elimination, except it applies to multivariate, non-linear systems. Just like in Gaussian elimination, a triangular structure in the Gröbner basis is produced (lexicographical term ordering), i.e. for each $1 \leq i \leq n$ there is (at least) one polynomial, which only depends on monomials with $x_1, ..., x_n$.

The first algorithm to compute a Gröbner basis is the Buchberger algorithm it was published with. While Buchberger algorithm gives double exponential worst case complexity and single exponential worst case complexity for multivariate cryptography settings [8], there are better alternatives, e.g. by Faugère's algorithms F4, F5 and F5/2 [14, 39].

The F5 algorithm computes Gröbner basis incremental by starting with a pair of generator polynomials of the ideal and then taking into account one more in each step. In this process useless reductions of polynomials to zero, which are the most expensive operations in an algorithm computing Gröbner basis, are avoided. Therefore, it is much more efficient than for example, the Buchberger algorithm, and it is even possible to brake the HFE Challenge 1 using F5/2 [18].

In the special case of signing, an effective simplification can be made since only one solution is needed instead of expected q^{r+v}, where r is the number of removed equations, v is the number of vinegar variables and $q = 2$ in QUARTZ, from solving the system in QUARTZ. Since the HFEv- system consists of $n + v$ variables and $n - r$ equations, the number of computed solutions can be reduced by fixing $(n + v) - (n - r)$ variables with some arbitrary values. The resulting system with $n - r$ variables and $n - r$ equations is expected to have one solution on average and therefore saves on computation time enormously. [8]

4.6 Performance

The most expensive step in QUARTZ' signature generation is the inversion of $\mathcal{F}$ [11], which is usually done by applying the Berlekamp algorithm with complexity $O(d^3 + n * d^2)$ [35]. Since this has to be done four times, the signature generation of QUARTZ is rather slow.

Let the parameters of QUARTZ be defined as in Sect. 4.2, the signature length is then given by [11]

$$|\sigma| = (n - r) + 4 * (r + v) = 128\text{bit} \ . \tag{30}$$

Even though such a small signature length is desirable with respect to transport (e.g. network congestion) and storage (e.g. memory), unfortunately, it comes with several drawbacks, especially key sizes and execution time. In [9], Courtois et al. state that the length of the public and private keys are about 71 KBytes and 3 KBytes. Additionally, the time on a Pentium III 500 MHz to sign and verify[6] is specified as 10 seconds on average and less than 1ms respectively. In the same setting, the time for key generation is stated as 4 seconds.

Due to the improvements made in [9] compared to [33], the signature scheme is not only 40% faster but the probability of not finding a signature for a message is cut down from 2^{-83} to 2^{-183}.

In [8], Courtois et al. state that signing using a computer with 2 GHz takes 2 seconds for the original parameter $d = 129$, while for the proposed alteration $d = 259$ it would take 6 seconds.

Being an improvement of QUARTZ, the signature scheme GUI presented by Ding et al. [11] claims to produce signatures of size down to 120 bits, along with the cutting down of signature generation time to one hundredth of the time needed by QUARTZ, while still satisfying the same security level of 2^{80}. However, verification speed as well as public and private key sizes remain nearly identical to QUARTZ.

There is nothing to be found about hardware implementations of QUARTZ or GUI. Therefore, no comparison of hardware and software implementations can be shown here and, for the same reason, no comparison between QUARTZ and RSA in terms of hardware implementation can be done.

Nevertheless, depending on software implementations, a comparison of the signature schemes QUARTZ, GUI-95 and RSA-1024 are shown in Table 8. Each of them satisfies the security level 2^{80}, but have different characteristics concerning the sizes of signatures, public- and private-key as well as signing and verifying speed. The values in k-cycles for the time needed to generate or verify a signature are measured on AMD Opteron 6212, 2.5 GHz / Intel Xeon CPU E5-2620, 2.0 GHz.

Table 8. Comparison GUI - QUARTZ - RSA - ECDSA [11]

scheme	security level (bits)	signature size (bits)	public key size (bytes)	private key size (bytes)	signing time (k-cycles)	verifying time (k-cycles)
QUARTZ	80	128	75,514	3,774	167,485 / 168,266	375 / 235
GUI-95	80	120	60,600	3,053	1,479 / 1,186	325 / 230
RSA-1024	80	1024	128	128	2,080 / 2,115	74 / 64

As shown, QUARTZ produces very small signatures compared to RSA, but for the price of a very slow signature generation and large keys. It needs ≈ 80 times as much time as RSA when generating signatures and is ≈ 5 times slower when verifying signatures. The public and private keys in QUARTZ are more than 500 times and ≈ 30 times bigger than those of RSA respectively. Even though GUI performs much better than QUARTZ, it still is, apart from the signature size, only equal to or better than RSA in the time needed for signature

[6] For a message of length <512 bits.

generation. Verification is nearly as slow as in QUARTZ and the keys are also very big compared to RSA.

So, in summary it can be said, that as long as in an IoT scenario the limited computation and memory capabilities of constrained devices is the limiting factor and not the network bandwidth, QUARTZ does not seem to be suitable for IoT scenarios. Even if such small signature sizes would be needed at any prize, QUARTZ's improvement GUI is a much better alternative, mainly because of its signature generation speed being better than RSA's.

5 Conclusion

Table 9 shows a comparison of the signature schemes QUARTZ, ECDSA and GUI. To visualize the comparison, Fig. 4 and Fig. 5 show signature and key sizes plotted. The data for keys setup, signing and verifying in milliseconds are measured on a 500 Mhz Pentium III [6]. The values in kcycles are taken from Table 8.

Table 9. Comparison of ECDSA [6], QUARTZ [6] and GUI [11]

scheme	security level (bits)	signature size (bits)	public key size (bytes)	private key size (bytes)	keys setup (ms)	signing (ms/kcycles)	verifying (ms/kcycles)
RSA	80	1024	128	128	-	-/2080	-/74
ECDSA	80	326	48	24	1.6	1.9/-	5.1/-
QUARTZ	80	128	>71000	>3700	3100	11000/167485	0.24/375
GUI	80	120	>60000	>3000	-	-/1479	-/325

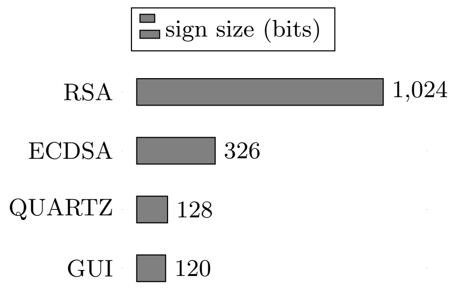


Fig. 4. Comparison of signature sizes of RSA, ECDSA, QUARTZ and GUI in bits

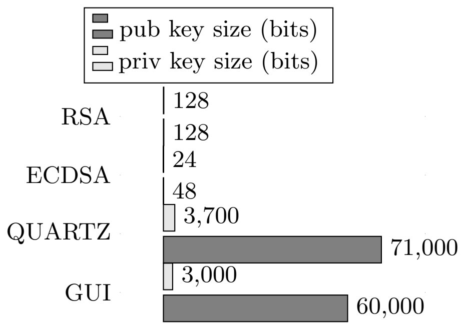


Fig. 5. Comparison of key sizes of RSA, ECDSA, QUARTZ and GUI in bits

ECDSA has approximately 2.5 times bigger signatures than QUARTZ, but performs better in every other aspect except verification time. The public and private keys in QUARTZ are respectively 1500 times and 150 times bigger than that of ECDSA. Furthermore, QUARTZ takes 2000 times more time to generate keys and is more than 5000 times slower when signing a message. Verification however is approx. 20 times faster in QUARTZ than in ECDSA. Since GUI is over 100 times faster than QUARTZ while signing, the ratio for the signature generation speed between GUI and ECDSA cuts down to under 50. For all other aspects, QUARTZ and GUI are in the same magnitude and therefore, the GUI to ECDSA comparison yields similar ratios as QUARTZ to ECDSA. Thus the key sizes are the most significant drawback of GUI compared to ECDSA.

The main advantages of ECDSA over RSA are key size and signature generation speed while the main disadvantage compared to RSA is verification speed. In summary, the performance data of ECDSA are more balanced than those of RSA in terms of key size, signature generation speed and verification speed while providing a much smaller signature size. The main drawback of QUARTZ/GUI in comparison to RSA is the size of the keys whereas the main advantage is the smaller signature size. Unlike QUARTZ, GUI can compete with RSA because of its signature generation speed being in the same magnitude as RSA's, leading to the trade-off between the signature size ($^1/_8$ times RSA's) and the verification speed (≈ 5 times RSA's).

As mentioned in Sec. 4.6, QUARTZ is absolute since GUI achieves the same advantages over RSA and ECDSA with fewer disadvantages. Therefore, only GUI and ECDSA are considered in the following recommendations.

In scenarios of IoT and constraint devices, the characteristic values to evaluate the suitability of signature algorithms mainly are signature size, key sizes, generation and verification speed. Since multiple entities may need to verify the same signature or the same signature may need to be verified multiple times, verification speed is important. Nevertheless, most of the verification in IoT scenarios is done on big servers, so even though ECDSA and GUI are slower than RSA in terms of signature verification, there are still IoT scenarios for which they are suitable.

So if verification speed is not a crucial factor, ECDSA is preferred over RSA because of its small signature size, small key size and signature generation speed, which help to prevent network congestion as well as suite the limited resources of constraint devices better. Also, ECDSA provides the best signature generation speed compared to GUI and RSA and is therefore suitable for devices that often send, but (nearly) never receive signed messages, such as sensors (which occur frequently in an IoT-scenario).

If devices have enough space to handle large keys, GUI outperforms ECDSA in every case except the last mentioned case, where a device only signs messages, because of its small signature size, small verification time and tolerable signature generation speed.

In summary it can be said, therefore, that ECDSA and GUI, under conditions mentioned above, are suitable for usage in an IoT-context.

References

1. Braun, M., Kargl, A.: A Note on Signature Standards. In: IACR Cryptology ePrint Archive 2007. (2007)
2. Brown, D.R.L: Standards for Efficient Cryptography, SEC 2: Recommended Elliptic Curve Domain Parameters. Certicom Research, Certicom Corp. (2010)
3. Brumley, BB., Tuveri, N.: Remote timing attacks are still practical. In: Computer Security-ESORICS 2011 Sep 12, pp. 355-371, Springer Berlin Heidelberg. (2011)
4. Bundesamt für Sicherheit in der Informationstechnik: Elliptic Curve Cryptography, Technical Guideline TR-03111, Version 2.0. Federal Office for Information Security. (2012)
5. Bushing, Marcan, Segher, Sven.: Console Hacking 2010, PS3 Epic Fail. In 27th Chaos Communication Congress. (2010)
6. Chen, J., Yang, B.: A More Secure and Efficacious TTS Signature Scheme. ICISC 2003, LNCS vol. 2971, pp. 320-338. Springer. (2003)
7. Courtois, N.: The security of Hidden Field Equations (HFE). In Cryptographers' Track RSA Conference, volume 2020 of Lectures Notes in Computer Science, pp. 266-281. (2001)
8. Courtois, N., Daum, M., Felke, P.: On the Security of HFE, HFEv- and Quartz. PKC 2003, LNCS vol. 2567, pp. 337-350. Springer. (2002)
9. Courtois, N., Goubin, L., Patarin, J.: Quartz, an asymmetric signature scheme for short signatures on PC. Submission to NESSIE, second revised version, October 2001, can be found at `http://www.minrank.org/quartz-b.pdf`.
10. Courtois, N., Klimov, A., Patarin, J., Shamir, A.: Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations. EUROCRYPT 2000, LNCS vol. 1807, pp. 392-407. Springer. (2000)
11. Ding, J., Petzoldt, A., Chen, M., Yang, B.: Gui: Revisiting Multivariate Digital Signature Schemes based on HFEv-. 2015, `http://csrc.nist.gov/groups/ST/post-quantum-2015/papers/session1-ding-paper.pdf`.
12. Ding, J., Schmidt, D: Rainbow, a new multivariable polynomial signature scheme, In ACNS'05 Proceedings of the Third international conference on Applied Cryptography and Network Security, pp. 164-175. (2005)
13. eBATS: ECRYPT Benchmarking of Asymmetric Systems. Part of eBACS: ECRYPT Benchmarking of Cryptographic Systems. Virtual Applications and Implementations Research Lab. `http://bench.cr.yp.to` Accessed on 17-02-2016
14. Faugère, J.: A new efficient algorithm for computing Gröbner bases without reduction to zero (F5). Proceeding ISSAC '02 Proceedings of the 2002 international symposium on Symbolic and algebraic computation. pp 75-83. (2002)
15. Goldwasser, S., Micali, S., and Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. In: SIAM J. Comput. 17, 2 (April 1988), pp. 281-308. (1988)
16. Gupta, V., Gupta, S., Chang, S. and Stebila, D.: Performance analysis of elliptic curve cryptography for SSL. In: Proceedings of the 1st ACM workshop on Wireless security, 2002, September, (pp. 87-94). ACM. (2002)
17. Hankerson, D., Menezes, A.J., Vanstone, S.: Guide to Elliptic Curve Cryptography. Springer-Verlag New York Inc. (2004)
18. HFE Challenge 1. `http://hfe.minrank.org/`. Accessed on 15-2-2016.
19. Hess E., Schafheutle, M., and Serf, P.: The Digital Signature Scheme ECGDSA. Siemens AG, Corporate Technology, Dept. CT IC 3. (2006)

20. Hitchcock, Y., Dawson, E., Clark, A., Montague, P.: Implementing an efficient elliptic curve cryptosystem over GF(p) on a smart card. In: ANZIAM J. 44(E):C354-C377. (2003)
21. Hutter, M., Medwed, M., Hein, D., Wolkerstorfer, J.: Attacking ECDSA-enabled RFID devices. In: Applied Cryptography and Network Security 2009 Jun 2, 519-534, Springer Berlin Heidelberg. (2009)
22. Johnson, D., Menezes, A., Vanstone, S.A.: The Elliptic Curve Digital Signature Algorithm (ECDSA). In: Int. J. Inf. Sec. 1 (1). pp. 36-63. (2001)
23. Khalique, A., Singh, K., Sood. S.: Implementation of Elliptic Curve Digital Signature Algorithm. In: Int. J. Com. App. Vol. 2 (2), 21-27. (2010)
24. Kipnis, A., Shamir, A.: Cryptanalysis of the HFE Public Key Cryptosystem by Relinearization. CRYPTO 99, LNCS vol. 1666, pp. 19-30. Springer. (1999)
25. Koblitz, N.: Elliptic curve cryptosystems, Mathematics of Computation 48 (177), pp. 203-209. (1987)
26. Lim, C.H., Lee, P.J.: The Korean certificate-based digital signature algorithm. In: Computers & Electrical Engineering, Vol. 4. pp. 249-265. (1998)
27. Lochter, M., Merkle, J.: RFC 5639: Elliptic Curve Cryptography ECC Brainpool Standard Curves and Curve Generation. (2010) `http://tools.ietf.org/html/rfc5639`
28. Medwed, M., Oswald, E.: Template attacks on ECDSA. In: Information Security Applications 2008 Sep 23. pp. 14-27, Springer Berlin Heidelberg. (2008)
29. Miller, V.: Use of elliptic curves in cryptography. In: CRYPTO, Lecture Notes in Computer Science 85. pp. 417-426. (1985)
30. National Institute of Standards and Technology: FIPS PUB 186-4, Digital Signature Standard (DSS). (2013)
31. National Institute of Standards and Technology: NIST Special Publication 800-57 Part 1 Revision 4, Recommendation for Key Management Part 1: General. (2016)
32. National Security Agency: Fact Sheet Suite B Cryptography. NSA. (2015)
33. Patarin, J., Courtois, N., Goubin, L.: QUARTZ, 128-Bit Long Digital Signatures. CTRSA 2001, LNCS vol. 2020, pp. 282-297. Springer. (2001) See [9] for the updated version.
34. Patarin, J., Courtois, N., Goubin, L.: Flash, a fast multivariate signature algorithm. CTRSA 2001, LNCS vol. 2020, pp. 298 - 307. Springer. (2001)
35. Richards, C.: Algorithms for Factoring Square-Free Polynomials over Finite Fields. Master Thesis, Simon Fraser University (Canada). (2009)
36. Salter, M.: RFC 6460: Suite B Profile for Transport Layer Security (TLS). (2012) `https://tools.ietf.org/html/rfc6460`
37. Sarath, G. Jinwala, D.C., Patel, S.: A Survey on Elliptic Curve Digital Signature Algorithm and its Variants. In: Second International Conference on Computational Science and Engineering (CSE-2014) Dubai, UAE. pp. 121-136. (2014)
38. Schmidt, JM., Medwed, M.: A fault attack on ECDSA. In: Fault Diagnosis and Tolerance in Cryptography (FDTC), 2009 Workshop on 2009 Sep 6, IEEE. pp. 93-99. (2009)
39. Stegers, T., Buchmann, J.: Faugère's F5 Algorithm Revisited. Thesis For The Degree Of Diplom-Mathematiker, Technische Universität Darmstadt. (2005)
40. Vaudenay, S.: The Security of DSA and ECDSA. In: Public Key Cryptography - PKC 2003: 6th International Workshop on Practice and Theory in Public Key Cryptography Miami, FL, USA, January 6–8, 2003 Proceedings. pp. 309-323. (2003)
41. Vanstone, S.A.: Next generation security for wireless: elliptic curve cryptography. In: Computers & Security 22.5: 412-415. (2003)

42. Wolf, C., Preneel, B.: Taxonomy of Public Key Schemes based on the problem of Multivariate Quadratic equations. (2005) http://eprint.iacr.org/2005/077.pdf.
43. Westhoff, D., Lamparter, B., Paar, C., Weimerskirch, A.: On digital signatures in ad hoc networks. European transactions on telecommunications, 16(5), pp.411-425. (2005)
44. Zhang, X., Zhou, M., Zhuang, J. X., Li, J.: Implementation of ECC-Based Trusted Platform Module. In: 2007 International Conference on Machine Learning and Cybernetics, Hong Kong, pp. 2168-2173. (2007)
45. Yasuda, T., Dahan, X., Huang, Y., Takagi, T., Sakurai, K.: MQ Challenge: Hardness Evaluation of Solving Multivariate Quadratic Problems. Cryptology ePrint Archive, Report 2015/275. (2015)
46. Zhu, L., Jaganathan, K., Lauter, K.: Elliptic Curve Cryptography (ECC) Support for Public Key Cryptography for Initial Authentication in Kerberos (PKINIT). (2008) https://tools.ietf.org/html/rfc5349