Extended abstract published in J. Luoluo, Ed., *2012 International Conference on Affective Computing and Intelligent Interaction (ICACII 2012)*, vol. 10 of *Lecture Notes in Information Technology*, pp. 289–294, IERI, 2012.

GQ2 vs. ECC: A Comparative Study of Two Efficient Authentication Technologies

Louis C. Guillou[1,⋆] and Marc Joye[2]

[1] Independent researcher, Bourgbarré, France
louisguillou@orange.fr
[2] Technicolor, Security & Content Protection Labs
1 avenue de Belle Fontaine, Cesson-Sévigné Cedex, France
marc.joye@technicolor.com

Abstract. The now classical GQ scheme is a zero-knowledge proof of knowledge of a (plain) RSA signature. With a new set of keys but with exactly the same protocol, the GQ2 scheme is a zero-knowledge proof of knowledge of a decomposition of the RSA modulus in use. For dynamic authentication, efficiency reasons suggest to use GQ2 rather than RSA or GQ or even, in certain cases, ECC. This paper provides a thorough comparison between GQ2 and ECC and specifies when a given technology should be preferred over the other one.

Keywords: Authentication protocols, GQ2, ECDSA, PKI, implementation.

1 Introduction

Authentication schemes are cryptographic schemes enabling a prover holding a private key to authenticate herself to a verifier holding the matching public key. In their seminal paper, Fiat and Shamir [4] showed how zero-knowledge techniques can help in the design of such schemes. The prover convinces the verifier in a 3-pass protocol that she possesses some secret information without revealing information whatsoever about her secret.

The GQ scheme by Guillou and Quisquater [6] is one of the most efficient extensions of the original Fiat-Shamir technique. The GQ scheme is RSA based, it uses e-th modulo N with e coprime to $\phi(N)$ (whereas the Fiat-Shamir scheme uses square roots modulo N). It is known to be (honest-verifier) zero-knowledge and a proof of knowledge of a plain RSA signature (i.e., an e-th root modulo N of the prover's public key) [7]. What is less known is that the GQ authentication protocol can be used with 2^k-roots modulo N. The corresponding protocol is then referred to as the GQ2 authentication protocol. It appears in ISO standard ISO/IEC 9798-5 [10]. GQ2 is also a signature scheme using the Fiat-Shamir heuristic. GQ2 signature appears in ISO standard ISO/IEC 14888-2 [9]. The main

⋆ Retired from France Telecom R&D.

advantage of GQ or GQ2 signature over RSA signature is that it is typically one order of magnitude faster. Compared to GQ, GQ2 offers the advantage of being compatible with RSA. A user possessing an RSA key pair can use it with GQ2 for dynamic authentication or signature.

Yet another way to obtain efficient schemes is to rely on elliptic curve cryptography (ECC). There is no standardized authentication protocol based on elliptic curves. But there is a standard for digital signature, the Elliptic Curve Digital Signature Algorithm (ECDSA) [1,5,8], which can be used for dynamic authentication purposes as follows. The verifier chooses a message uniformly at random and requests the prover to produce a valid signature on the message.

Elliptic curve cryptography is becoming more and more popular. Nevertheless, RSA is still dominating the security marketplace for public-key cryptography. Moving from RSA to ECC implies changing the underlying public-key infrastructure (PKI). This has a cost. A simpler alternative would be to move from RSA to GQ2. But what is the expected performance? How does GQ2 compare to ECC? This paper answers this latter question. We provide a detailed analysis of the two technologies and present concrete implementations.

Outline The rest of this paper is organized as follows. In the next section, we present the GQ2 authentication protocol and signature. We show that the authentication protocol is indeed a zero-knowledge proof of knowledge of the factorization of an RSA modulus. In Section 3, we review the ECDSA signature based on elliptic curves. In Section 4, we describe the underlying algorithms and study their complexity and memory requirements. We offer detailed, optimized implementations for GQ2 and ECDSA. Finally, in Section 5, we compare the two technologies for various parameters and different settings.

2 GQ2 Authentication and Signature

2.1 Squares and square roots

Consider a prime number p. The multiplicative group of integers modulo p, namely $\mathbb{Z}_p^*$, has $p-1$ elements: $\{1,\dots,p-1\}$. An element $a \in \mathbb{Z}_p{}^*$ is a square if and only if there exists some $\alpha \in \mathbb{Z}_p^*$ such that $a = \alpha^2$. Fermat theorem says that

$$a^{p-1} = 1 \pmod p$$

for any $a \in \mathbb{Z}_p^*$. Therefore, a is a square in $\mathbb{Z}_p^*$ (or a quadratic residue modulo p) if and only if $a^{(p-1)/2} = 1 \pmod p$. This is known as Euler's criterion and gives rise to the Legendre symbol:

$$\left(\frac{a}{p}\right) = a^{(p-1)/2} \bmod p = \begin{cases} 1 & \text{if } a \text{ is a quadratic residue modulo } p \\ -1 & \text{if } a \text{ is a quadratic nonresidue modulo } p \end{cases} .$$

Suppose further that prime p is odd (i.e., $p \neq 2$). Then we can write

$$p = 2^b t + 1 \quad \text{with } t \text{ odd and } b \geqslant 1 .$$

When $b = 1$ — or equivalently when $p \equiv 3 \pmod 4$, if a is a quadratic residue modulo p then its two square roots are

$$\alpha = a^{(p+1)/4} \pmod p \quad \text{and} \quad -\alpha \pmod p \ .$$

Among these two roots, it is easily verified that α is itself a quadratic residue modulo p and that $-\alpha$ is not. In other words, square roots exist and are unique in the subgroup of quadratic residues modulo p.

More generally, if we define the cyclic subgroup

$$\mathbb{S}_p = \{x^{2^b} \mid x \in \mathbb{Z}_p^* \text{ where } p = 2^b t + 1 \text{ with } t \text{ odd } \} \subset \mathbb{Z}_p^*$$

then square roots exist and are unique in $\mathbb{S}_p$. Given an element $a \in \mathbb{S}_p$, its square root (in $\mathbb{S}_p$) is given by[3]

$$\alpha = a^{(t+1)/2} \ .$$

Indeed, since any element a in $\mathbb{S}_p$ is 2^b-power residue, it follows that $a^t = a^{(p-1)/2^b} = 1$, and consequently $\alpha^2 = a^{t+1} = a^t \cdot a = a$. Iterating the process, for any $k \geqslant 1$, 2^k-th roots exist and are unique in $\mathbb{S}_p$: $a = \beta^{2^k}$ is equivalent to $\beta = a^x$ with $x = (\frac{t+1}{2})^k \bmod t$.

2.2 Basic protocol

Key generation Any GQ scheme makes use of a generic equation

$$G\,Q^v \equiv 1 \pmod N \qquad (1)$$

where G is a public number, Q a private number, N an RSA-type modulus, and v a verification exponent.

Let $N = p_1 p_2$ where $p_1 = 2^{b_1} t_1 + 1$ and $p_2 = 2^{b_2} t_2 + 1$ are prime, with t_1, t_2 odd and $b_1, b_2 \geqslant 1$. Let g be a small prime number such that

$$\begin{cases} (\frac{g}{p_1}) = -1, (\frac{g}{p_2}) = 1 & \text{if } b_1 > b_2 \\ (\frac{g}{p_1}) = 1, (\frac{g}{p_2}) = -1 & \text{if } b_1 < b_2 \\ (\frac{g}{p_1}) = -(\frac{g}{p_2}) & \text{if } b_1 = b_2 \end{cases} .$$

For example, the following values are pertinent:

- $g = 2$ with $p_1 \equiv 3 \pmod 8$ and $p_2 \equiv 7 \pmod 8$, i.e., $N \equiv 5 \pmod 8$;
- $g = 3$ with $p_1 \equiv 1 \pmod 3$ and $p_2 \equiv 2 \pmod 3$, i.e., $N \equiv 2 \pmod 3$;
- $g = 5$ with $p_1 \equiv \pm 2 \pmod 5$ and $p_2 \equiv \pm 1 \pmod 5$, i.e., $N \equiv \pm 2 \pmod 5$; and so on.

[3] Notice that $(t+1)/2$ is an integer.

Define $b = \max(b_1, b_2)$ and let $G = g^{2^b}$. Remark that G is in $\mathbb{S}_{p_1} \times \mathbb{S}_{p_2}$ while g is not since g is a quadratic nonresidue in $\mathbb{Z}_N^*$. Let $v = 2^{k+b}$ be the verification exponent where $k \geqslant 1$ is a security parameter. Finally, let $Q_i = G^{x_i} \bmod p_i$ with $x_i = -(\frac{t_i+1}{2})^{k+b} \bmod t_i$ ($i \in \{1, 2\}$). By construction, letting now $Q = \mathrm{CRT}(Q_1, Q_2)$ (i.e., $Q \equiv Q_1 \pmod{p_1}$ and $Q \equiv Q_2 \pmod{p_2}$) yields $G\,Q^v \equiv 1 \pmod N$ as per Eq. (1).

Public key is $\{N, b, k, g\}$ and private key is $\{p_1, p_2, Q_1, Q_2\}$.

Authentication GQ2 authentication protocol between a prover, say Peggy, and a verifier, say Victor, is given by the following 3-pass scheme.

$$\begin{array}{lcl}
\textbf{Peggy} & & \textbf{Victor} \\
r \in_R \{1, \ldots, N-1\} & & \\
W = r^v \bmod N & \xrightarrow{\quad W \quad} & \\
 & \xleftarrow{\quad d \quad} & d \in_R \{0,1\}^k \\
D = rQ^d \bmod N & \xrightarrow{\quad D \quad} & \\
 & & W \stackrel{?}{=} D^v G^d \bmod N
\end{array}$$

The exchanged quantities, W, d, D, are called witness, challenge and response, respectively. The protocol can be iterated several times.

Signature Companion GQ2 signature scheme is obtained thanks to the Fiat-Shamir heuristic [4] (see also [16]). The challenge is replaced with the hash value of the witness and of the message M to be signed. So, the signature σ on a message M is given by

$$\sigma = (d, D) \quad \text{with} \begin{cases} d = \mathcal{H}(W, M) & \text{where } W = r^v \bmod N \\ D = rQ^d \bmod N \end{cases}$$

for some cryptographic hash function $\mathcal{H} : \{0,1\}^* \to \{0,1\}^k$. The validity of signature $\sigma = (d, D)$ is then verified by checking whether $d = \mathcal{H}(W', M)$ with $W' = D^v G^d \bmod N$.

2.3 Security considerations

In the authentication protocol, Peggy proves in a zero-knowledge fashion that she knowns a solution Q to Eq. (1), which, in turn, implies the knowledge of the factorization of N. Indeed, suppose without loss of generality that $(\frac{g}{p_1}) = -1$ and $(\frac{g}{p_2}) = 1$ (and thus $b_2 \leqslant b_1$ and $b = b_1$). Letting $h = g^{2^{b-1}} \bmod N$ and $z := (Q^{-1})^{2^{k+b-1}} \bmod N$, Equation (1) yields

$$h^2 \equiv z^2 \pmod N \iff (h - z)(h + z) \equiv 0 \pmod{\{p_1, p_2\}} \ .$$

Since $k \geqslant 1$, first note that z is a 2^b-power residue modulo N. But $h \not\equiv z \pmod{p_1}$ because h is not a 2^b-power residue modulo p_1 (i.e., $h \notin \mathbb{S}_{p_1}$) since $(\frac{g}{p_1}) = -1$, and $h \equiv z \pmod{p_2}$ because $h \in \mathbb{S}_{p_2}$ since $(\frac{g}{p_2}) = 1$. Hence,

$$\gcd(h - z, N)$$

will disclose p_2.

The three security properties zero-knowledge proofs of knowledge must fulfill are completeness, soundness, and zero-knowledge.

Completeness Completeness means that given an honest prover and an honest verifier, the protocol succeeds with overwhelming probability. For GQ2, it is easy to see that $G\,Q^v \equiv 1 \pmod{N}$ implies $r^v \equiv (rQ^d)^v\,G^d \pmod{N}$ for any challenge d.

Soundness Soundness means that a dishonest prover should not be able to convince an honest prover. More formally, soundness means there exists a (polynomial-time) knowledge extractor $\mathcal{K}$ such that if a dishonest prover successfully executes the protocol with non-negligible probability then $\mathcal{K}$ can recover some information allowing successful subsequent protocol executions.

From two accepting conversations sharing the same witness in GQ2 authentication, say (W, d, D) and (W, d', D'), knowledge extractor $\mathcal{K}$ can factor RSA modulus N as follows.

1. Let $\gcd(v, d - d') = 2^\delta$ where $0 \leqslant \delta < k - 1$ (since $d, d' \in \{0,1\}^k$ and $d \neq d'$) and write $d - d' = 2^\delta \tau$ where τ is odd;
2. Observing that $W \equiv D^v G^d \equiv (D')^v G^{d'} \pmod{N} \implies \left(\frac{D'}{D}\right)^v \equiv G^{d-d'} \pmod{N} \implies \left(\frac{D'}{D}\right)^{2^{k+b}} \equiv (g^\tau)^{2^{\delta+b}} \pmod{N}$, compute

$$H = (g^\tau)^{2^{b-1}} \bmod N\,, \quad Z = \left(\frac{D'}{D}\right)^{2^{k+b-\delta-1}} \bmod N\,,$$

   and $\gcd(H - Z, N)$, which yields a non-trivial factor of N. It is worth noting here that H^2 and Z (and thus any power thereof) are in $\mathbb{S}_{p_1} \times \mathbb{S}_{p_2}$.

Zero-knowledge The property of zero-knowledge ensures that the prover does not disclose any information about its secret knowledge while interacting with the verifier. For GQ2, it is possible to simulate conversations that are indistinguishable from real conversations (i.e., from those resulting from protocol executions with the real prover). This can be achieved by first randomly picking $d^* \in_R \{0,1\}^k$ and $D^* \in_R \mathbb{Z}_N^*$ and next by computing $W^* = (D^*)^v G^{d^*} \bmod N$. If (W, d, D) denotes a real conversation, it is easily verified that the two distributions

$$(W^*, d^*, D^*) \quad \text{and} \quad (W, d, D)$$

are indistinguishable.

2.4 Compatibility with RSA

One of the main advantages of GQ2 technology resides in its compatibility with RSA. Compatibility with RSA is important as it allows GQ2 to rely on the public-key infrastructure (PKI) already deployed for RSA.

Public parameters Any RSA modulus N and its associated certificate can be used as it is within GQ2. The other public parameters, namely (b, k, g), are either given by the user or defined by the application.

Remark that one can verify that g is a quadratic nonresidue modulo N from its Jacobi symbol; i.e., by checking that $(\frac{g}{N}) = -1$. If $N = p_1 p_2$, this ensures that $(\frac{g}{p_1}) = -(\frac{g}{p_2})$. Remark also that $(\frac{g}{N}) = -1$ implies that the very existence of a solution Q to Eq. (1) shows that N is composite.

Private parameters When RSA is implemented with Chinese remaindering (a.k.a. RSA in CRT mode), the user knows secret primes p_1 and p_2 of RSA modulus N and can therefore derive the corresponding private key, $\{p_1, p_2, Q_1, Q_2\}$.

In contrast, when RSA is implemented in standard mode, the user has only access to the triple (N, e, d) where $ed \equiv 1 \pmod{\phi(N)}$. In this case, the user does not have directly the knowledge of primes p_1 and p_2. But the user can easily recover them thanks to Miller's algorithm [13] because a multiple of the group order, $\#\mathbb{Z}_N^* = \phi(N)$, is known; i.e., $ed - 1$.

2.5 Extended protocol

The efficiency of the basic protocol can be improved using a multi-prime RSA-type modulus. Define $N = \prod_{i=1}^{m+1} p_i$ for large (equal-size) primes $p_i = 2^{b_i} q_i + 1$ with q_i odd and $b_i \geqslant 1$.[4] For better efficiency, it is advantageous to choose $b_i = 1$. Parameter $m \geqslant 1$ is an additional security parameter. The global security is given by the product $k \times m$. Typical values, depending on the application context, are:

- $k \times m = 8$: weak authentication mode;
- $k \times m = 32$: strong authentication mode;
- $k \times m = 80$: signature mode (or more generally, $k \times m \geqslant 80$).

Parameter b is now defined as $b = \max_{1 \leqslant i \leqslant m} b_i$. There is also a set of m quadratic nonresidues, $\mathcal{G} = \{g_1, \ldots, g_m\}$, that are selected in a way similar as for the basic protocol.

3 Elliptic Curve Cryptography

Discrete-log based cryptosystems can be transposed in the setting of elliptic curves over a finite field. Since the underlying problem is substantially harder

[4] Note that the basic protocol corresponds to the case $m = 1$.

in the case of elliptic curve cryptography, smaller parameters can be used but with the same expected security level [11, 14]. The main schemes based on elliptic curves are ECDSA for digital signatures, ECDH for key exchange and ECIES for encryption.

Let E be an elliptic curve defined over the prime field $\mathrm{GF}(p)$, of order $\#E(\mathrm{GF}(p)) = hn$ for some cofactor $h \in \{1, 2, 3, 4\}$ and prime n. Let also $G \in E$ be a base point of order n. The domain parameters for ECDSA are $\{E, p, G, n, h\}$. An ECDSA key pair is associated with a given set of domain parameters. Each user chooses a random element $d \in_R \{1, \dots, n-1\}$ and computes the point $Q = [d]G \in E$. The public key is Q and the private key is d.

Before detailing the signing/verification processes, it is useful to introduce some notation. The neutral element of elliptic curve E (namely, the point at infinity) is denoted by $\mathcal{O}$. Given a point $P \in E \setminus \{\mathcal{O}\}$, its x-coordinate is denoted by $\mathrm{x}(P)$.

The ECDSA signature on a message m is obtained as follows:

1. randomly choose $k \in_R \{1, \dots, n-1\}$;
2. compute $r = \mathrm{x}([k]G) \bmod n$; if $r = 0$ go to Step 1;
3. compute $s = k^{-1}\big(\mathcal{H}(m) + d\,r\big) \bmod n$; if $s = 0$ go to Step 1;
4. return $\sigma = (r, s)$.

The validity of $\sigma = (r, s)$ is then checked using public key Q as follows:

1. verify that $r, s \in \{1, \dots, n-1\}$;
2. compute $w = s^{-1} \bmod n$, $u_1 = \mathcal{H}(m)\,w \bmod n$ and $u_2 = rw \bmod n$;
3. compute $X = [u_1]G + [u_2]Q$;
4. accept the signature if and only if $X \neq \mathcal{O}$ and $\mathrm{x}(X) \equiv r \pmod{n}$.

4 Algorithmic Aspects

We assume hereafter that the algorithms are implemented on Ω-bit processors. A number A is represented as $A = \sum_i A_i \beta^i$ with $\beta = 2^\Omega$ and $0 \leqslant A_i < \beta$. We call *word* a value coded on Ω bits. We call *CPU operation* an operation performed on words; for example, a CPU multiplication refers to the multiplication of two words. CPU multiplication will used as unit when comparing GQ2 and ECC technologies; its cost is denoted by M.

4.1 Basic operations

Modular multiplication An efficient algorithm for modular reduction was proposed by Montgomery [15]. For a number co-prime with a modulus N, it represents equivalent classes of $A \pmod{N}$ as $\widetilde{A} = AR \pmod{N}$ and re-defines the modular multiplication as

$$\texttt{Mont\_Mult}(A, B, N) = ABR^{-1} \bmod N \ .$$

So, it turns out that $\texttt{Mont\_Mult}(\widetilde{A},\widetilde{B},N) = \widetilde{A \cdot B}$ is isomorphic to the usual multiplication modulo N. Montgomery algorithm relies on the observation that

$$TR^{-1} \bmod N = \frac{T + MN}{R} - \epsilon N$$

where $\epsilon \in \{0,1\}$ and $M = TN'_0 \bmod N$ with $N'_0 = -N^{-1} \bmod R$. In particular, when N is odd and R is chosen as a power of $\beta = 2^{\Omega}$ on a Ω-bit processor, Montgomery reduction is easily evaluated since divisions by R become shifts and reductions modulo R are obtained with a series of additions.

We give below the interleaved version of the algorithm, as presented in [12, § 14.3.2]. It has the advantage of being compact. In addition to the memory space for storing A, B, N and N'_0, it merely requires $\underline{(n+1)\text{ words}}$ of working memory.

Algorithm 1 Montgomery modular multiplication

Input: $N < \beta^n$ odd, with $\beta < 2^{\Omega}$, and $N'_0 = -N^{-1} \bmod \beta$
$A = \sum_{j=0}^{n-1} A_j \beta^j$ and $B = \sum_{j=0}^{n-1} B_j \beta^j$ with $0 \leqslant A, B < N$
Output: $\texttt{Mont\_Mult}(A,B,N) = ABR^{-1} \bmod N$ with $R = \beta^n$

```
1: $R_0 \leftarrow 0$
2: for $j = 0$ to $n-1$ do
3:     $u \leftarrow \big((R_0 \bmod \beta) + A_j \cdot B_0\big)N'_0 \bmod \beta$
4:     $R_0 \leftarrow (R_0 + A_j \cdot B + u \cdot N) \operatorname{div} \beta$
5: end for
6: if $(R_0 \geqslant N)$ then $R_0 \leftarrow R_0 - N$
7: return $R_0$
```

The evaluation of u (Line 3) requires 2M. The evaluation of R_0 (Line 4) requires 2 multiplications of an n-word value by a word, such a multiplication can be obtained with nM. As a result, the Montgomery modular multiplication costs $n(2\mathsf{M} + 2 \cdot n\mathsf{M}) = \underline{2n(n+1)\mathsf{M}}$.

Remark 1. Note that the interleaved version does not permit to optimize the modular squaring. Hence, in this paper, no distinction will be made between modular squaring and modular multiplication for complexity analysis.

Integer multiplication Another useful operation is the multiplication of two n-word integers A and B. We give hereafter an "in-place" method based on the schoolboy method that evaluates the product $A \cdot B$ with rewriting in the buffer containing A:

$$(C, A) \leftarrow A \cdot B$$

where C is an n-word memory register.

Algorithm 2 In-place integer multiplication

Input: $A = \sum_{j=0}^{n-1} A_j \beta^j$, $B = \sum_{j=0}^{n-1} B_j \beta^j$, and $C = \sum_{j=0}^{n-1} C_j \beta^j$
Output: `IntP_Mult`$(A, B) = (C, A) \leftarrow A \cdot B$

```
1: $C \leftarrow 0; c \leftarrow 0$
2: **for** $j = 0$ to $n - 1$ **do**
3:     $tmp \leftarrow A_j$
4:     $(c, A_j) \leftarrow tmp \cdot B_0 + C_0$
5:     **for** $i = 0$ to $n - 2$ **do** $(c, C_i) \leftarrow tmp \cdot B_{i+1} + C_{i+1} + c$
6:     $C_{n-1} \leftarrow c$
7: **end for**
8: **return** $(C, A)$
```

It appears that merely $\underline{n \text{ words}}$ of working memory (i.e., register C) are required. Further, operand B remains available in memory, output of the algorithm. The cost of the algorithm is $n(1\mathsf{M} + (n-1)\mathsf{M}) = \underline{n^2\mathsf{M}}$.

4.2 GQ2 technology

A GQ2 authentication or signature requires a first modular exponentation for the computation of witness $W = r^v \bmod N$ and a second modular exponentiation for the computation of response $D = r\,Q^d \bmod N$.

The most economic algorithm (memory-wise) for exponentiation is the "square & multiply" algorithm. Moreover, it leaves available the value of x (in the computation of $x^d \bmod N$), output of the algorithm. Adapted to the Montgomery multiplication, it runs as depicted in Alg. 3. Normalization and denormalization steps are necessary. The computation of $R^2 \bmod N$ in the normalization step will be neglected in the complexity analysis. Likewise, the cost for the computation of N_0' will be neglected; its value can be given as an input of the algorithm. So, the cost of the normalization (resp. denormalization) is estimated to 1 Montgomery modular multiplication.

Algorithm 3 Square & multiply algorithm (with Montgomery multiplication)

Input: $N < \beta^n$ odd, with $\beta < 2^\Omega$, and $N_0' = -N^{-1} \bmod \beta$
$x < N$ and $d = \sum_{j=0}^{\ell-1} d_j 2^j$ with $d_j \in \{0, 1\}$ and $d_{\ell-1} = 1$
Output: $x^d \bmod N$

```
1: $R_0 \leftarrow$ Mont_Mult$(x, R^2 \bmod N, N)$; $R_1 \leftarrow R_0$        ▹ Normalization
2: **for** $j = \ell - 2$ down to 0 **do**
3:     $R_0 \leftarrow$ Mont_Mult$(R_0, R_0, N)$
4:     **if** $(d_j = 1)$ **then** $R_0 \leftarrow$ Mont_Mult$(R_0, R_1, N)$
5: **end for**
6: $R_0 \leftarrow$ Mont_Mult$(R_0, 1, N)$        ▹ Denormalization
7: **return** $R_0$
```

Furthermore, an evaluation modulo $N = p_1 p_2$ can be efficiently computed from p_1 and p_2 using the Chinese remainder theorem (CRT):

$$A \bmod N = A_1 + p_1\Big(I_{p_1}(A_2 - A_1) \bmod p_2\Big)$$

with $A_1 = A \bmod p_1$, $A_2 = A \bmod p_2$, and $I_{p_1} = {p_1}^{-1} \bmod p_2$.

Complexity analysis Using Montgomery modular exponentiation (Alg. 3), the evaluation of $\widetilde{W_i} := r^v R \bmod p_i$ with $v = 2^{k+b}$ requires $(k+b)$ squarings modulo p_i plus 1 multiplication modulo p_i for the normalization of $r \bmod p_i$; $i \in \{1, 2\}$. Note that as is detailed in Appendix A.1, a single denormalization step is necessary to obtain W through CRT from $\widetilde{W_1}$ and $\widetilde{W_2}$.

Challenge d is uniformly distributed over $\{0, 1\}^k$; its expected bit-length is therefore $\ell = \frac{1}{2^k} + \sum_{j=1}^{k} \frac{j}{2^{k+1-j}} \approx k - 1$. Using Algorithm 3, the evaluation of $Q^d \bmod p_i$ requires $\ell - 1$ squarings modulo p_i and, on average, $(\ell - 1)/2$ multiplications modulo p_i for the main loop plus 1 multiplication modulo p_i for the normalization of $Q \bmod p_i$; $i \in \{1, 2\}$. Multiplying the so-obtained result with $r \bmod p_i$ directly yields $D_i := rQ^d \bmod p_i$; there is no need to denormalize. Consequently, setting $\ell = k-1$, the cost to get D_i is $1+(k-2)+(k-2)/2+1 = k+(k-1)/2$ Montgomery modular multiplications. Now, by normalizing I_{p_1}, a single Montgomery multiplication allows one to get D through CRT from D_1 and D_2 (again see Appendix A.1 for details).

To sum up, under the following assumptions:

A1. Chinese remaindering is used for the evaluation of W and D:
  - modulus $N = p_1 p_2$ is balanced: $|p_1|_2 = |p_2| = |N|_2/2 \propto \Omega$;
  - primes $p_1, p_2 \equiv 3 \pmod 4$;
  - the inverse of p_1 modulo p_2, I_{p_1}, is precomputed (as is the case for RSA applications using Chinese remaindering);

A2. the "square & multiplication" algorithm with Montgomery multiplication (see Alg. 3) is used for the computation of W and of D (modulo p_1 and p_2), where parameters N'_0 and R^2 (modulo p_1 and p_2) are precomputed;

A3. a modular squaring has the same cost as a modular multiplication (see Remark 1);

A4. the cost of reading/writing data in various memory types is neglected;

A5. the cost of drawing random number r is neglected;

A6. the cost of exchanges between the prover and the verifier in authentication mode is neglected;

A7. the cost of the hash function in signature mode is neglected;

a GQ2 authentication or signature with security parameter k requires

$$\underbrace{2\cdot\underbrace{(\overbrace{1}^{\text{norm.}}+k+\overbrace{b}^{=1})}_{\text{eval. of }\widetilde{W_i}}+\underbrace{1+\overbrace{1}^{\text{denorm.}}}_{\text{CRT}}}_{\text{evaluation of }W}+\underbrace{2\cdot\underbrace{(k+(k-1)/2)}_{\text{eval. of }D_i}+\underbrace{\overbrace{1}^{\text{norm. }I_{p_1}}+1}_{\text{CRT}}}_{\text{evaluation of }D}=5k+7$$

Montgomery modular multiplications (on average) plus 2 integer multiplications for the CRT recombinations, all involving $|N|_2/2$-bit values, that is, a total cost of

$$\boxed{\Big(n(\tfrac{n}{2}+1)(5k+7)+\tfrac{n^2}{2}\Big)\mathsf{M}}$$

where n represents the word-length of modulus N.

Memory requirements Regarding the amount of working memory, in addition to a few CPU registers, under assumptions A1–A7 and

A8. writing in non-volatile memory (EEPROM-type) is not allowed;

a GQ2 authentication or signature with security parameter k requires

$$\boxed{\Big(\tfrac{7n}{2}+\lceil\tfrac{k}{\Omega}\rceil+1\Big)\text{ words of working memory}}$$

where n represents the word-length of modulus N. A detailed implementation can be found in Appendix A.1.

Illustration As an example, we get for a 1024-bit modulus N on an 8-bit processor (i.e., $\Omega = 8$, $n = 128$) the following figures.

Authentication mode	Weak	Strong	Signature
Parameter k	8	32	80
# CPU multiplications (M)	399,232	1,397,632	3,394,432
RAM memory (bytes)	450	453	459

4.3 ECC technology

In order to avoid divisions in GF(p), projective representations are preferred. In [8], the Jacobian representation is suggested as it provides fast arithmetic. With this representation, a point doubling requires 9 multiplications in GF(p) in the general case — and 8 multiplications in GF(p) when curve parameter $a = -3$. The addition of two different points requires 16 multiplications in GF(p). When one of the two points has its Z-coordinate equal to 1, the number of multiplications in GF(p) drops to 11; such a point addition is known as a *mixed point addition*. A good trade-off is to use *modified* Jacobian coordinates [2]. A point doubling then requires 8 multiplications in GF(p) — regardless of curve parameter a — and a mixed point addition requires 13 multiplications in GF(p). See [3] for detailed formulas.[5]

[5] As we will rely on the interleaved version of Montgomery modular multiplication (Alg. 1), no distinction is made between squaring and multiplication in GF(p); cf. Remark 1.

Complexity analysis With the modified w-ary algorithm (Alg. 4, Appendix A.2), the evaluation of $[k]G$ requires $(\ell-1)w$ point doublings and $(\ell-1)(2^w-1)/2^w$ point additions, on average, where $\ell = \left\lceil \frac{|k|_2}{w} \right\rceil$. If the accumulator (i.e., R_0) is represented with modified Jacobian coordinates and if the 2^{w-1} precomputed points (i.e., R_1 and R_{2i+1} for $1 \leqslant i \leqslant 2^{w-1}-1$) are represented in affine coordinates, then a point doubling requires 8 multiplications in GF(p) and a point addition requires 13 multiplications in GF(p). Consequently, the computation of $[k]G$ requires, on average, $\left(\left\lceil \frac{|k|_2}{w} \right\rceil - 1\right)\left(8w + 13\,\frac{2^w-1}{2w}\right)$ multiplications in GF(p).

An application of Fermat theorem yields the (affine) x-coordinate of $[k]G$ by an inversion modulo p followed by 2 multiplications modulo p; that is, on average, $3(|p|_2 - 1)/2 + 2$ multiplications modulo p, assuming the inverse is computed with the "square & multiply" algorithm. Similarly, component s of an ECDSA signature requires, on average, $3(|n|_2 - 1)/2 + 2$ multiplications modulo n.

Furthermore, with Montgomery modular multiplication, one has to take into account possible normalizations and denormalizations.

Remark 2. Cryptographic standards recommend elliptic curves E over GF(p) such that $\#E(\mathrm{GF}(p)) = h\,n$ where cofactor $h \in \{1, 2, 3, 4\}$. Hasse's theorem says that $p + 1 - 2\sqrt{p} \leqslant \#E(\mathrm{GF}(p)) \leqslant p + 1 + 2\sqrt{p}$. For those curves, it thus follows that $|n|_2 \approx |p|_2$. For the complexity analysis, we assume $|n|_2 = |p|_2$.

So, under the following assumptions:

B1. elliptic curve E is defined over finite field GF(p) for a random prime p ($|p|_2 \geqslant 160$, $|p|_2 \propto \Omega$);
B2. base point G is of order n with $|n|_2 = |p|_2$;
B3. computations in GF(p) are carried out using Montgomery representation, where parameters N'_0 and R^2 (modulo p and n) are precomputed;
B4. the modified w-ary algorithm (see Alg. 4) (appropriately adapted for Montgomery modular multiplication) is used for the point multiplication $[k]G$:
  - 2^{w-1} points $G_i = [2i-1]G$ ($1 \leqslant i \leqslant 2^{w-1}$) are precomputed and given as input in affine coordinates;
  - the accumulator makes use of modified Jacobian coordinates;
  - the cost for initializing the accumulator, R_0, is neglected;
  - the cost to get $\nu(k_j)$ and $\overline{k_j}$ for digits k_j is neglected;
  - [2] represents a doubling in modified Jacobian coordinates and [+] represents a mixed point addition (modified Jacobian coordinates/affine coordinates);
  - the output is given with modified Jacobian coordinates: $[k]G = (X_1, Y_1, Z_1, a{Z_1}^4)$;
B5. the "square & multiply" algorithm is used for computing modular inverses (in the computation of $x_1 := \mathrm{x}([k]G)$ and of s);
B6. a modular squaring has the same cost as a modular multiplication (see Remark 1);
B7. the cost of the modular reduction $x_1 \bmod n$ is neglected (cf. Remark 2);
B8. the cost of modular additions/subtractions is neglected;

B9. the cost of reading/writing data in various memory types is neglected;
B10. the cost of drawing random number k is neglected;
B11. the cost of hashing message M is neglected;

an ECDSA signature requires

$$\underbrace{2^w+1}_{\text{norm.}}+\underbrace{\left(\left\lceil\frac{|p|_2}{w}\right\rceil-1\right)\left(8w+13\,\frac{2^w-1}{2^w}\right)}_{\text{point multiplication}}+$$
$$\underbrace{3(|p|_2-1)/2+2+1}_{r}+\underbrace{1+3(|p|_2-1)/2+2}_{s}$$
$$=\left(\left\lceil\frac{|p|_2}{w}\right\rceil-1\right)\left(8w+13\,\frac{2^w-1}{2^w}\right)+3|p|_2+2^w+4$$

modular multiplications with $|p|_2$ bits, on average; or

$$\boxed{2l(l+1)\left[\left(\left\lceil\frac{\Omega l}{w}\right\rceil-1\right)\left(8w+13\,\frac{2^w-1}{2^w}\right)+3\Omega l+2^w+4\right]\mathsf{M}}$$

where l is the word-length of prime p, on an Ω-bit processor.

Memory requirements Regarding the amount of working memory, in addition to a few CPU registers, under assumptions B1–B11 and

B12. writing in non-volatile memory (EEPROM-type) is not allowed;

an ECDSA signature with the modified w-ary algorithm requires

$$\boxed{\left((2^w+10)l+1\right)\text{ words of working memory}}$$

where l represents the word-length of prime p. A detailed implementation can be found in Appendix A.2.

Illustration As an example, we get for a 160-bit ECDSA signature with the modified w-ary algorithm on an 8-bit processor (i.e., $\Omega = 8, l = 20$) the following figures, for different values for w.

	$w = 1$	$w = 3$	$w = 4$
# CPU multiplications (M)	2,344,860	1,988,175	1,867,582
RAM memory (bytes)	241	361	521

5 Comparison

The next three tables gives a comparative of 1024-bit GQ2 and 160-bit ECC technologies on 8-, 16- and 32-bit processors. For a given amount of RAM, they list the number of needed CPU multiplications, under assumptions A1–A8 for GQ2 and B1–B12 for ECC.

Table 1. GQ2 vs. ECC: 8-bit processor ($\Omega = 8$)

RAM (bytes)	GQ2 technology			ECC technology
	Weak auth.	Strong auth.	Signature	Signature
240	—	—	—	$2,344,860$
450	$399,232$	$1,397,632$	—	$1,988,175$
460	↓	↓	$3,394,432$	↓
520	↓	↓	↓	$1,867,582$

Table 2. GQ2 vs. ECC: 16-bit processor ($\Omega = 16$)

RAM (bytes)	GQ2 technology			ECC technology
	Weak auth.	Strong auth.	Signature	Signature
240	—	—	—	$614,130$
450	$101,312$	$354,752$	—	$520,713$
460	↓	↓	$861,132$	↓
520	↓	↓	↓	$489,129$

Table 3. GQ2 vs. ECC: 32-bit processor ($\Omega = 32$)

RAM (bytes)	GQ2 technology			ECC technology
	Weak auth.	Strong auth.	Signature	Signature
240	—	—	—	$167,490$
450	$26,080$	$91,360$	—	$142,013$
460	↓	↓	$221,920$	↓
520	↓	↓	↓	$133,399$

GQ2 technology requires at least 450 bytes of RAM memory for enabling its implementation. In environments with fewer RAM memory, only ECC technology is available (the implementation we gave requires at least 240 bytes of RAM memory).

When the amount of available RAM memory is of the order of 500 bytes (or more), GQ2 in strong authentication mode outperforms ECC. This may appear counter-intuitive as GQ2 makes use of much larger key sizes. This is especially true when the processor size (i.e., Ω) is large. Besides, the table assumes the basic GQ2 protocol, which is compatible with RSA. Better performance (in both memory and speed) can be obtained using the extended GQ2 protocol (cf. § 2.5). Yet another advantage of GQ2 is that it offers a fine-grained control on the authentication strength (parameter k) for better efficiency/security trade-offs.

In summary, GQ2 technology is particularly well suited to environments featuring moderate-size RAM memory (500 bytes or more). Moreover, it can make use of the already-deployed PKI for RSA. ECC technology is more intended to very constrained environments.

References

1. ANSI X9.62. Public key cryptography for the financial services industry, the elliptic curve digital signature algorithm (ECDSA). American National Standards Institute, 2005.
2. Henri Cohen, Atsuko Miyaji, and Takatoshi Ono. Efficient elliptic curve exponentiation using mixed coordinates. In K. Ohta and D. Pei, editors, *Advances in Cryptology – ASIACRYPT '96*, volume 1514 of *Lecture Notes in Computer Science*, pages 51–65. Springer, 1998.
3. Explicit-formulas database. http://www.hyperelliptic.org/EFD/.
4. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *Advances in Cryptology – CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1987.
5. FIPS 186-2. Digital signature standard (DSS). National Institute for Standards and Technology, 2000.
6. Louis C. Guillou and Jean-Jacques Quisquater. A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory. In C. G. Günther, editor, *Advances in Cryptology – EUROCRYPT '88*, volume 330 of *Lecture Notes in Computer Science*, pages 123–128. Springer, 1988.
7. Louis C. Guillou and Jean-Jacques Quisquater. A "paradoxical" indentity-based signature scheme resulting from zero-knowledge. In S. Goldwasser, editor, *Advances in Cryptology – CRYPTO '88*, volume 403 of *Lecture Notes in Computer Science*, pages 216–231. Springer, 1990.
8. IEEE P1363. Standard specifications for public-key cryptography. IEEE Standards Association, 2000.
9. ISO/IEC 14888-2. Information technology – Security techniques – Digital signatures with appendix – Part 2: Integer factorization based mechanisms. International Organization for Standardization, 2008.
10. ISO/IEC 9798-5. Information technology – Security techniques – Entity authentication – Part 5: Mechanisms using zero-knowledge techniques. International Organization for Standardization, 2009.
11. Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987.
12. Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.
13. Gary L. Miller. Riemann's hypothesis and tests for primality. *Journal of Computer and System Sciences*, 13(3):300–317, 1976.
14. Victor S. Miller. Use of elliptic curves in cryptography. In H. C. Williams, editor, *Advances in Cryptology – CRYPTO '85*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426. Springer, 1985.

15. Peter L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985.
16. David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, 2000.

A Detailed Implementations

A.1 GQ2

Let n denote the word-length of modulus N on an Ω-bit processor. Let $\mathtt{MEM0}, \ldots, \mathtt{MEM3}, \mathtt{TMP1}, \mathtt{TMP2}$ be $n/2$-word registers, $\mathtt{TMP0}$ be a $(n/2+1)$-word register and $\mathtt{mem1}$ be a k-bit register.

A GQ2 authentication or signature can be executed with the memory requirements as per § 4.2 with the following sequence of instructions:

1. load p_1 in $\mathtt{MEM0}$ and draw a random element r_1 $(:= r \bmod p_1)$ in $\mathtt{MEM2}$;
   $[\mathtt{MEM0} \leftarrow p_1, \mathtt{MEM2} \leftarrow r_1]$
2. load $R^2 \bmod p_1$ in $\mathtt{TMP1}$;
   $[\mathtt{TMP1} \leftarrow R^2 \bmod p_1]$
3. normalize r_1 using $\mathtt{TMP0}$ as working buffer and recopy the result, namely $\mathtt{Mont\_Mult}(\mathtt{MEM2}, \mathtt{TMP1}, \mathtt{MEM0})$, from $\mathtt{TMP0}$ to $\mathtt{TMP1}$;
   $[\mathtt{TMP1} \leftarrow \widetilde{r_1} := rR \bmod p_1]$
4. perform $(k+b)$ modular squarings
   $$\mathtt{TMP1} \leftarrow \mathtt{Mont\_Mult}(\mathtt{TMP1}, \mathtt{TMP1}, \mathtt{MEM0})$$
   using $\mathtt{TMP0}$ as working buffer;
   $[\mathtt{TMP1} \leftarrow \widetilde{W_1} := WR \bmod p_1]$
5. repeat Steps 1–4 modulo p_2 by replacing $\mathtt{MEM2}$ with $\mathtt{MEM3}$ and $\mathtt{TMP1}$ with $\mathtt{TMP2}$;
   $[\mathtt{MEM0} \leftarrow p_2, \mathtt{MEM3} \leftarrow r_2, \mathtt{TMP2} \leftarrow \widetilde{W_2}]$
6. compute $\widetilde{W_2} - \widetilde{W_1} \pmod{p_2}$ in $\mathtt{TMP2}$;
   $[\mathtt{TMP2} \leftarrow (W_2 - W_1)R \bmod p_2]$
7. load I_{p_1} in $\mathtt{MEM1}$, compute $\mathtt{Mont\_Mult}(\mathtt{MEM1}, \mathtt{TMP2}, \mathtt{MEM0})$ with $\mathtt{TMP0}$ as working buffer and recopy the result in $\mathtt{MEM1}$;
   $[\mathtt{MEM1} \leftarrow I_{p_1}(W_2 - W_1) \bmod p_2]$
8. load p_1 in $\mathtt{TMP2}$;
   $[\mathtt{TMP2} \leftarrow p_1]$
9. compute $\mathtt{IntP\_Mult}(\mathtt{MEM1}, \mathtt{TMP2})$ with $\mathtt{MEM0}$ as working buffer (see Alg. 2);
   $[(\mathtt{MEM0}, \mathtt{MEM1}] \leftarrow p_1\big(I_{p_1}(W_2 - W_1) \bmod p_2\big)]$
10. denormalize $\widetilde{W_1}$ with $\mathtt{TMP0}$ as working buffer to get W_1 and add it to $(\mathtt{MEM0}, \mathtt{MEM1})$ to get W;
    $[(\mathtt{MEM0}, \mathtt{MEM1}) \leftarrow W]$
11. **authentication mode** send Q and receive d,
    signature mode compute $d = \mathcal{H}(W, M)$;
    and put d in $\mathtt{mem1}$;
    $[\mathtt{mem1} \leftarrow d]$

12. load p_1 in `MEM0` and $R^2 \bmod p_1$ in `TMP1`;

    [`MEM0` $\leftarrow p_1$, `TMP1` $\leftarrow R^2 \bmod p_1$]

13. load $Q \bmod p_1$ in `MEM1`, normalize it using `TMP0` as working buffer and recopy the result, $\widetilde{Q_1} = QR \bmod p_1$, in `MEM1`;

    [`MEM1` $\leftarrow \widetilde{Q_1}$]

14. apply the "square & multiply" exponentiation algorithm (Alg. 3) with `TMP1` as accumulator and `TMP0` as working buffer;

    [`TMP1` $\leftarrow Q^d R \bmod p_1$]

15. compute `Mont_Mult(MEM2,TMP1,MEM0)` with `MEM0` as working buffer and recopy the result in `MEM2`;

    [`MEM2` $\leftarrow D_1 := rQ^d \bmod p_1$]

16. repeat Steps 12–15 modulo p_2 by replacing `MEM2` with `MEM3` and `TMP1` with `TMP2` (solely in Step 12);

    [`MEM0` $\leftarrow p_2$, `MEM3` $\leftarrow D_2 := rQ^d \bmod p_2$, `TMP2` $\leftarrow R^2 \bmod p_2$]

17. compute $D_2 - D_1 \pmod{p_2}$ in `MEM3`;

    [`MEM3` $\leftarrow D_2 - D_1$]

18. load I_{p_1} in `MEM1`, normalize it as `Mont_Mult(MEM1,TMP2,MEM0)` with `TMP0` as working buffer and recopy the result in `TMP1`;

    [`MEM1` $\leftarrow I_{p_1} R \bmod p_2$]

19. compute `Mont_Mult(TMP1,MEM3,MEM0)` with `TMP0` as working buffer abd recopy the result in `MEM1`;

    [`MEM1` $\leftarrow I_{p_1}(D_2 - D_1) \bmod p_2$]

20. load p_1 in `MEM3`;

    [`MEM3` $\leftarrow p_1$]

21. compute `IntP_Mult(MEM1,MEM3)` with `MEM0` as working buffer (see Alg. 2);

    [(`MEM0`, `MEM1`] $\leftarrow p_1\big(I_{p_1}(D_2 - D_1) \bmod p_2\big)$]

22. add `MEM2` to get D;

    [(`MEM0`, `MEM1`) $\leftarrow D$]

23. **authentication mode** send D,
    signature mode return (d, D).

A.2 ECC

Point multiplication The central operation in elliptic curve cryptography is the point multiplication. Several algorithms are available for this computation. We selected the modified (additive) w-ary algorithm as it is both efficient and easy to implement. It generalizes (in additive notation) the "square & multiply" algorithm by considering w-bit windows. For the computation of $[k]G$, it repeatedly updates an accumulator R_0 as $[2^w]R_0$ and add some precomputed point according the value of the current digit. A modification of the algorithm allows one to only precompute *odd* multiples of G: $G_i := [2^{2i+1}]G$ for $1 \leqslant i \leqslant 2^{w-1} - 1$. This results in memory savings. Moreover, precomputed points can be represented with affine coordinates, resulting in further memory savings and speeding up the computation since general point additions are then advantageously replaced with mixed point additions.

In Alg. 4, an integer $k_j > 0$ is uniquely written under the form $k_j = 2^{\nu(k_j)}\overline{k_j}$ with $\overline{k_j}$ odd. Moreover, to avoid confusion, point addition is henceforth noted with $[+]$.

Algorithm 4 Modified w-ary algorithm on elliptic curve E

Input: p prime, $G \in E(\mathrm{GF}(p))$, $w > 1$
$k = \sum_{j=0}^{\ell-1} k_j(2^w)^j$ with $k_j \in \{0,1\}^w$ and $k_{\ell-1} \neq 0$
Output: $[k]G$

```
 1: $R_1 \leftarrow G; R_0 \leftarrow [2]R_1$
 2: for $i = 1$ to $2^{w-1} - 1$ do $R_{2i+1} \leftarrow R_{2i-1}$ [+] $R_0$
 3: $R_0 \leftarrow R_{\overline{k_{\ell-1}}}; R_0 \leftarrow [2^{\nu(k_{\ell-1})}]R_0$
 4: for $j = \ell - 2$ down to 0 do
 5:     if $(k_j = 0)$ then $t \leftarrow w$
 6:     else $t \leftarrow \nu(k_j); R_0 \leftarrow [2^{w-t}]R_0; R_0 \leftarrow R_0$ [+] $R_{\overline{k_j}}$
 7:     end if
 8:     $R_0 \leftarrow [2^t]R_0$
 9: end for
10: return $R_0$
```

ECDSA Let l denote the word-length of p (defining prime field $\mathrm{GF}(p)$) on an Ω-bit processor. Let also $u = 2^w + 1$. Let $\mathtt{MEM0}, \ldots, \mathtt{MEM}u, \mathtt{TMP1}, \ldots, \mathtt{TMP6}, \mathtt{mem0}$ be l-word registers and TMP0 be a $(l+1)$-word register. The Jacobian representation of $[k]G$ is denoted by (X_1, Y_1, Z_1) and its affine representation by (x_1, y_1) where $x_1 = X_1/{Z_1}^2$ and $y_1 = Y_1/{Z_1}^3$.

An ECDSA signature can be executed with the memory requirements as per § 4.3 with the following sequence of instructions:

1. load p in MEM0 and draw a random element k in mem0;
   $[\mathtt{MEM0} \leftarrow p, \mathtt{mem0} \leftarrow k]$
2. load $R^2 \bmod p$ in TMP1;
   $[\mathtt{TMP1} \leftarrow R^2 \bmod p]$
3. put curve parameter a in MEMu and, for $1 \leqslant i \leqslant 2^{w-1}$, put $G_i := [2i-1]G$ in $(\mathtt{MEM}(2i-1), \mathtt{MEM}(2i))$;
   $[(\mathtt{MEM}_1, \mathtt{MEM2}) \leftarrow G, \ldots, (\mathtt{MEM}(u-2), \mathtt{MEM}(u-1)) \leftarrow G_{2^w-1}, \mathtt{MEM}u \leftarrow a]$
4. normalize G_i $(1 \leqslant 2^{w-1})$ and a with TMP0 as working buffer, $\mathtt{Mont\_Mult(MEM}j$, TMP1, MEM0) for $1 \leqslant j \leqslant u$, and recopy the result in input register;
   $[(\mathtt{MEM}_1, \mathtt{MEM2}) \leftarrow \widetilde{G}, \ldots, (\mathtt{MEM}(u-2), \mathtt{MEM}(u-1)) \leftarrow \widetilde{G_{2^w-1}}, \mathtt{MEM}u \leftarrow \widetilde{a}]$
5. apply modified w-ary algorithm (Alg. 4) with $(\mathtt{MEM}u, \mathtt{TMP1}, \mathtt{TMP2}, \mathtt{TMP3})$ as accumulator, TMP0 as (Montgomery) working buffer, and TMP4, TMP5, TMP6 as temporary registers for point operations;
   $[\mathtt{MEM}u \leftarrow \widetilde{a{Z_1}^4}, \mathtt{TMP1} \leftarrow \widetilde{X_1}, \mathtt{TMP2} \leftarrow \widetilde{Y_1}, \mathtt{TMP3} \leftarrow \widetilde{Z_1}]$
6. recopy $\widetilde{Z_1} := Z_1R \bmod p$ in MEM1 and put $p - 2$ in MEM3 by performing $\mathtt{MEM3} \leftarrow \mathtt{MEM0} - 2$;
   $[\mathtt{MEM1} \leftarrow Z_1R \bmod p, \mathtt{MEM2} \leftarrow p - 2]$

7. apply the "square & multiply" algorithm with TMP3 as accumulator and TMP0 as working buffer, to get ${Z_1}^{p-2}R \bmod p$;

   [TMP3 $\leftarrow {Z_1}^{-1}R \bmod p$]
8. compute $(X_1/{Z_1}^2)R \bmod p$ in two steps as TMP3 $\leftarrow$ `Mont_Mult`(TMP3, TMP3, MEM0) followed by TMP3 $\leftarrow$ `Mont_Mult`(TMP3, TMP1, MEM0) with TMP0 as working buffer; denormalize TMP3 in MEM1, with TMP0 as working buffer, to get x_1 (supposed $< n$) and thus r;

   [MEM1 $\leftarrow r$]
9. load n in MEM0 and $R^2 \bmod n$ in TMP1; [MEM0 $\leftarrow n$, TMP1 $\leftarrow R^2 \bmod n$]
10. normalize r and k by respectively computing MEMu $\leftarrow$ `Mont_Mult`(MEM1, TMP1, MEM0) and mem0 $\leftarrow$ `Mont_Mult`(mem0, TMP1, MEM0) with TMP0 as working buffer;

    [MEMu $\leftarrow \widetilde{r} := rR \bmod n$, mem0 $\leftarrow \widetilde{k} := kR \bmod n$]
11. put $n - 2$ in TMP2 (by computing TMP2 $\leftarrow$ MEM0 $- 2$) and apply the "square & multiply" algorithm with TMP1 as accumulator and TMP0 as working buffer, to get $k^{n-2}R \bmod n$ and recopy the result in mem0;

    [mem0 $\leftarrow k^{-1}R \bmod n$]
12. load secret parameter d in MEM2, compute `Mont_Mult`(MEM2, MEMu, MEM0) with TMP0 as working buffer and recopy the result in MEMu;

    [MEMu $\leftarrow dr \bmod n$]
13. compute $\mathcal{H}(M)$ in working buffers TMP0, TMP1, ..., TMP6 and write it in MEM2;

    [MEM2 $\leftarrow \mathcal{H}(M)$]
14. add MEM2 to MEMu (modulo n) in MEM2;

    [MEM2 $\leftarrow \mathcal{H}(M) + dr \bmod n$]
15. compute MEM2 $\leftarrow$ `Mont_Mult`(mem0, MEM2, MEM0) with TMP0 as working buffer, to get s;

    [MEM2 $\leftarrow s$]
16. return (r, s).