# Montgomery-Suitable Cryptosystems

[Published in G. Cohen, S. Litsyn, A. Lobstein, and G. Zémor, Eds., *Algebraic Coding*, vol. 781 of *Lecture Notes in Computer Science*, pp. 75-81, Springer-Verlag, 1994.]

David Naccache and David M'Raïhi

Gemplus Card International, Crypto-Team
1 Place de Navarre, F-95200, Sarcelles, France.
Email: {`100142.3240, 100145.2261`}`@compuserve.com`

**Abstract.** Montgomery's algorithm [8], hereafter denoted $M_n(\cdot,\cdot)$, is a process for computing $M_n(A,B) = ABN \bmod n$ where $N$ is a constant factor depending only on $n$.
Usually, $AB \bmod n$ is obtained by $M_n(M_n(A,B), N^{-2} \bmod n)$ but in this article, we introduce an alternative approach consisting in pre-integrating $N$ into cryptographic keys so that a single $M_n(\cdot,\cdot)$ will replace directly each modular multiplication.
Except the advantage of halving the number of Montgomery multiplications, our strategy skips the pre-calculation (and the storage) of the constant $N^{-2} \bmod n$ and reveals to be particularly efficient when a hardware device implementing $M_n(\cdot,\cdot)$ is the basic computational tool at one's command.

## 1 Introduction

Since the discovery of public-key cryptography, considerable endeavours were invested in the design of efficient modular reduction algorithms. Reducing the time and memory complexities of the operation $a \bmod n$ is a challenging problem on which relies the practical feasibility of most signature, identification and encryption schemes.

In 1987, Montgomery [8] introduced a very elegant $O(\log^2(n))$−per-constant-hardware algorithm for computing $AB2^{-\ell(n)} \bmod n$ where $\ell(n)$ denotes the length of $n$ in bits.
As this scheme outputs $AB2^{-\ell(n)} \bmod n$ and not $AB \bmod n$, a correction of the result is done by a re-multiplication of the output by the pre-calculated constant $K = 4^{\ell(n)} \bmod n$ (in other words, standard modular reduction is reconstituted by $AB \bmod n = M_n(K, M_n(A,B))$ where $M_n(A,B)$ denotes the Montgomery operator).

In itself, the algorithm $M_n(A,B)$ is very simple:

```
input : an odd $n$, $A=\sum_{i=0}^{\ell(n)-1} 2^i A[i]$, $B$
output: $AB2^{-\ell(n)} \bmod n$
$R \leftarrow A$
for $i \leftarrow 0$ to $\ell(n)-1$
  if $A[i]=1$ then $R \leftarrow R+B$
  if $R$ is odd then $R \leftarrow R+n$
  $R \leftarrow R/2$
if $R<n$ then return($R$) else return($R-n$)
```

A deeper explanation of the method and its variants can be found in [1], [4], [6], [8] and [14].

Replacing the modular multiplications in the 'square & multiply' exponentiation algorithm by $M_n(A,B)$s we get a new operator $E_n(A,p)$ wherein $p-1$ parasite factors are accumulated with the final result (the accumulation of $p-1$ factors, and not $p$, is a a consequence of the fact that for multiplying $p$ numbers one must perform $p-1$ multiplications):

```
input : an odd $n$, $A$, $p>0$
output: $A^p 2^{-\ell(n)(p-1)} \bmod n$
$e \leftarrow A$
first $\leftarrow$ #True
while ($p \neq 0$)
  if $p$ is odd then
    if first then
      $z \leftarrow e$
      first $\leftarrow$ #False
    else $z \leftarrow M_n(z,e)$
  $p \leftarrow \lfloor p/2 \rfloor$
  $e \leftarrow M_n(e,e)$
return($z$)
```

In the next sections, we will show how to use $E_n(A,p)$ and $M_n(A,B)$ in the implementation of a variety of cryptosystems without getting rid of the $N$s nor pre-calculating the constant $K$. This approach is new since instead of designing computational tools for the comfortable execution of protocols we modify the cryptosystems to meet the computational particularities of a modular reduction tool. Technically, this is achieved by pre-integrating the $N$s directly into the keys.

[1] shows that the hardware, memory and time complexities of $M_n(A,B)$ are equivalent to that of the operation $AB$ (without modular reduction). From this remark, it appears that a transformation of cryptosystems wherein modular multiplications are replaced by $M_n$s is equivalent in practice to performing these schemes when multiplications are done in $\mathbf{Z}$ instead of $\mathbf{Z}_n$.

Hereafter, we will successively apply the protocol re-writing strategy to the RSA [12], Diffie-Hellman [3], Fiat-Shamir [7], Guillou-Quisquater [11], Schnorr

[13], El-Gamal [5], OSS (Naccache's repaired version) [9], DSS [10] and Benaloh-de Mare's one-way accumulators [2].

In this paper, $n$ is an RSA modulus, $N = 2^{-\ell(n)} \bmod n$, $p$ and $q$ are primes, $P = 2^{-\ell(p)} \bmod p$ and $Q = 2^{-\ell(q)} \bmod q$.

## 2 An RSA-like Scheme

Let $m$ and $c$ represent a message and its ciphertext and denote by $\{e, d\}$ a couple of RSA [12] keys.

The sender encrypts $m$ by computing $c = E_n(m, e) = m^e N^{e-1} \bmod n$ and the receiver decrypts $c$ by calculating:

$$E_n(c, d) \equiv c^d N^{d-1} \equiv (m^e N^{e-1})^d N^{d-1} \equiv m^{ed} N^{ed-1} \equiv m \bmod n$$

The security of the modified scheme, formally equivalent to that of the original RSA, is based on the fact that the encryption operation is a successive application of:

1. A public permutation of the message space (multiplication by the public factor $N$)
2. Encryption with a standard RSA
3. And a public reverse-permutation of the ciphertext space (division by $N$).

The decryption consists in applying successively the inverse transformations.

## 3 A Diffie-Hellman-like Scheme

Alice and Bob share the same prime $p$ and a common exponentiation base $a$ [3].

- Alice computes $r_1 = E_p(a, x) \equiv a^x p^{x-1} \bmod p$
- Bob computes $r_2 = E_p(a, y) \equiv a^y p^{y-1} \bmod p$
- Alice and Bob swap $r_1$ and $r_2$
- Alice computes $k = E_p(r_2, x) \equiv (a^y P^{y-1})^x P^{x-1} \equiv a^{xy} P^{xy-1} \bmod p$
- Bob computes $k = E_p(r_1, y) \equiv (a^x P^{x-1})^y P^{y-1} \equiv a^{xy} P^{xy-1} \bmod p$

Again, the security of this Montgomery variant is based on the observation that the scheme is equivalent to a standard Diffie-Hellman with a new exponentiation base $\alpha = aP$ where both parties obtain the common key $\alpha^{xy} \bmod p$ and then perform a public permutation of the key space (division by the public constant $P$).

## 4 A Fiat-Shamir-like Protocol

Redefine the Fiat-Shamir keys [7] by: $\nu_j s_j^2 N^3 \equiv 1 \bmod n$

- The prover sends $z = M_n(r, r) \equiv r^2 N \bmod n$ to the verifier

- The verifier sends a random binary word $e$ of size $k$ (let $\hbar(e)$ be the Hamming weight of $e$).
- The prover sends $y = M_n(s_{i_1}, M_n(s_{i_2}, \ldots M_n(s_{i_{\hbar(e)}}, r) \ldots)$ where the $i_j$s denote the $\hbar(e)$ indices selected by vector $e$.
- The verifier compares $z$ and $M_n(\nu_{i_1}, M_n(\nu_{i_2}, \ldots M_n(\nu_{i_{\hbar(e)}}, M_n(y, y)) \ldots)$

Note that this can be applied to the digital signature as well.

As in the original Fiat-Shamir, one can show that breaking this protocol is equivalent to the extraction of modular roots and thus to factoring. For simplicity, we will only discuss the case $k = 1$ since generalisation is straightforward and only involves heavier notations:

Cheating implies the existence of an efficient algorithm $\mathcal{A}(\nu, n) = \{Z, y_0, y_1\}$ where:

$$y_0^2 N \bmod n \equiv Z \quad \text{and} \quad y_1^2 N \nu N \equiv Z \bmod n \ .$$

From these two equations, we get directly: $\frac{y_0}{y_1} = \sqrt{\nu N} \bmod n$.

To transform $\mathcal{A}$ to a root-extracting algorithm, firstly obtain $\sqrt{N} \bmod n$ from $\mathcal{A}(1, n)$ (this is needed only if $\ell(n)$ is odd. If not, compute directly $\sqrt{N} \equiv 2^{-\hat{n}/2} \bmod n$) and then calculate $\sqrt{\nu} = \frac{y_0}{y_1 \sqrt{N}} \bmod n$ from $\mathcal{A}(\nu, n)$.

## 5 A Naccache-Ong-Schnorr-Shamir-like Signature Scheme

The relation between the secret keys and the public keys is $\nu_j s_j^2 N^3 \equiv 1 \bmod N$

- The prover sends $x = r + M_n(m, \frac{1}{r} \bmod n) \equiv r + \frac{mN}{r} \bmod n$ and the verifier replies with $e$.
- The prover sends $y = M_n(s_{i_1}, M_n(s_{i_2}, \ldots M_n(s_{i_{\hbar(e)}}, r - M_n(m, \frac{1}{r} \bmod n)) \ldots)$
- The verifier computes $z = M_n(x, x) - M_n(\nu_{i_1}, M_n(\nu_{i_2}, \ldots M_n(\nu_{i_{\hbar(e)}}, M_n(y, y)) \ldots)$ and compares if: $z = M_n(4, m)$.

The proof of security is similar to that of the section 4 (see [9] as well).

## 6 A Guillou-Quisquater-like Protocol

The new relation between $J$, $B$ and $\nu$ [11] is: $f(ID)B^\nu N^{\nu+1} \equiv 1 \bmod n$, modular multiplications are replaced by $M$s and exponentiations by $E$s.

- The prover picks $r$ and sends $ID$ and $T = E(r, \nu)$ to the verifier.
- The verifier picks $d < \nu$, computes $J = f(ID)$ and replies with $d$.
- The prover sends $U = M_n(E_n(B, d), r)$
- The verifier checks that $T = M_n(E_n(J, d), E_n(U, \nu))$

The zero-knowledge property of the scheme is preserved since the external view of the protocol can be simulated by a probabilistic Turing machine that picks $d$ and $U$ from a random tape, computes $T = J^d U^\nu N^{d+\nu-1}$ and outputs the triple $\{T, d, U\}$ which is indistinguishable from a communication between the prover and the verifier.

## 7 A Schnorr-like Protocol

Schnorr's protocol [13], presented in Crypto'89, is a DLP-based system where the relation between the public ($\nu$) and the secret ($s$) keys are: $\nu = \alpha^{-s} \bmod p$ and $\alpha^q \equiv 1 \bmod p$.

Redefine: $\nu P^{s+1} \alpha^s \equiv 1 \bmod p$ and give to the signer a new secret-key $\sigma = sQ^{-1} \bmod q$

- The prover picks $r$ and sends $x = E_p(\alpha, r)$
- The verifier picks $e$ and sends it to the prover.
- The prover sends back $y = r + M_q(\sigma, e) \bmod q = r + se \bmod q$
- The verifier tests that $x = M_p(E_p(\alpha, y), E_p(\nu, e))$

Security is guaranteed by a zero-knowledge simulator that picks at random $y$ and $e$, computes $x = \alpha^y \nu^e P^{y+e-1} \bmod p$ and and outputs the indistinguishable triple $\{x, e, y\}$.

## 8 An El-Gamal-like Signature Scheme

In El-Gamal's scheme [5] the public key is $p = g^s \bmod q$ (where $s$ is the secret key), $q$ and $g$ are system parameters and $m$ is signed by $\{u, v\}$ where: $u = g^k \bmod q$, $k$ is random and $v$ is such that $m = su + kv \bmod (q-1)$. Upon reception, the signature is checked by comparing $p^u u^v$ to $g^m$ (modulo $q$).

To make El-Gamal's scheme Montgomery-suitable, redefine $p = E_q(g, s)$ but let $v$ remain unchanged.

For signing $m$, compute $u = E_q(g, k)$ and for checking the signature, make sure that:

$$M_q(E_q(p, u), E_q(u, v)) = E_q(g, m) \ .$$

It is still unclear if this scheme is strictly equivalent to the original El-Gamal in terms of security.

## 9 A Data Signature Standard-like Algorithm

The Digital Signature Algorithm [10] was proposed by the US National Institute of Standards and Technology to *provide an appropriate core for applications requiring a digital rather than written signature.*

DSA parameters are:

1. A prime modulus $p$ where $2^{L-1} < p < 2^L$ for $512 \leq L \leq 1024$ and $L = 64\alpha$ for some $\alpha$
2. A prime $q$ such that $2^{159} < q < 2^{160}$ and $p - 1$ is a multiple of $q$.
3. A number $g$, of order $q$ modulo $p$ such that $g = h^{\frac{p-1}{q}} \bmod p$, where $h$ is any integer such that $1 < h < p - 1$ and $g = h^{\frac{p-1}{q}} \bmod p > 1$
4. A number $x$, generated randomly and a number $y$ defined by the relation: $y = g^x \bmod p$.

5. A number $k$ generated randomly such that $0 < k < q$.

The integers $p$, $q$ and $g$ are system parameters and can be public or common to a group of users. The signer's private and public keys are respectively $x$ and $y$. Parameters $x$ and $k$ are used for signature generation only and must be kept secret. Parameter $k$ must be regenerated for each signature.

In order to sign a message $m$ (hashed value of a primitive file $M$), the signer computes the signature $\{r, s\}$ by:

$$r = \left(g^k \bmod p\right) \bmod q \quad \text{and} \quad s = \frac{m + xr}{k} \bmod q$$

After making sure that $r \neq 0$ and $s \neq 0$, the signature $\{r, s\}$ is sent to the verifier who computes:

1. $w = \frac{1}{s} \bmod q$
2. $u_1 = mw \bmod q$
3. $u_2 = rw \bmod q$
4. $v = (g^{u_1} y^{u_2} \bmod p) \bmod q$
5. And checks if $v$ and $r$ match to accept or reject the signature.

Redefine $r = E_p(g, M_q(x, 1)) = g^{Qx} P^{Qx-1} \bmod p$

The signer computes:

$$r = M_q(E_p(g, k), 1) = (g^k P^{k-1} \bmod p)Q \bmod q \quad \text{and}$$

$$s = M_q(\frac{1}{k}, m + M_q(x, r)) = \frac{mQ + xrQ^2}{k} \bmod q$$

and sends $\{r, s\}$ to the verifier who controls the signature by computing $w = s^{-1} \bmod q$ and comparing $M_q(M_p(E_p(g, M_q(m, w)), E_p(y, M_q(r, w))), 1)$ to $r$. Note that the strategy of using $M_q$ "upstairs" and $M_p$ "downstairs" can also be applied to El-Gamal's algorithm upon a proper modification of the keys. Also, modular inverses can be computed by $M_q(M_q(E_q(x, q-2), 1), 1) = x^{-1} \bmod q$ or $M_q(E_q(x, q-2), Q) = x^{-1} \bmod q$ (if the storage of $Q$ is believed to be a good trade-off compared to one $M_q$).

As in section 9, it is unclear if this scheme is equivalent to the original DSA in terms of security.

## 10 Using $E_n$ as a One-way Accumulator

At Eurocrypt'93, Benaloh and de Mare [2] introduced a new family of one-way functions which satisfy a quasi-commutativity property that allows them to be used as accumulators. This property allows to design protocols in which the need for a trusted central authority can be eliminated.

Typical examples of practical applications of the concept are document time-stamping and membership testing.

Accumulators should satisfy the property: $h(h(x, y), z) = h(h(x, z), y)$ which is true for $E_n(x, y) = h(x, y)$

## 11 Conclusion and Implementation Details

[1] showed that it is possible to compute $ABN \bmod n$ in $\ell(n)+1$ clock cycles. That is, a modular multiplication is performed with the same complexity (timewise and hardwarewise) as that of a standard multiplication, It is thus possible to execute all the schemes mentioned in our article in time and hardware equivalent to their execution without modular reductions at all.

Field experiments confirm this estimation since on a 68HC05 running at 3.5 MHz (internal clock), the $M$ operation (for $\ell(n) = 512$) is executed in about 135 ms (RAM usage is less than 70 bytes) and a special $M^2$ (squaring) version runs at 85 ms but requires a double RAM space.

## 12 Acknowledgment

The authors would like to thank Beni Arazi for his numerous suggestions and remarks concerning this work.

## References

1. B. ARAZI, *Modular multiplication is equivalent in complexity to a standard multiplication*, Fortress U&T Internal Report (1992) available from Fortress U&T Information Safeguards, P.O. Box 1350, Beer-Sheva, IL-84110, Israel.
2. J. BENALOH & M. de MARE, *One-way accumulators: A decentralised alternative to digital signatures*, Advances in Cryptology: Proceedings of Eurocrypt'93, Lecture Notes in Computer Science, Springer-Verlag, to appear.
3. W. DIFFIE & M. HELLMAN, *New directions in cryptography*, IEEE TIT, vol. 22, (1976), pp 644–654.
4. S. DUSSE & B. KALISKI, *A cryptographic library for the Motorola DSP56000.* In Advances in Cryptology – Eurocrypt'90, pp. 230–244, Springer-Verlag, New-York, 1990.
5. T. EL-GAMAL, *A public-key cryptosystem and a signature scheme based on the discrete logarithm*, IEEE TIT, vol. 31, No. 4, (1985), pp. 469–472.
6. S. EVEN, *Systolic modular multiplication*, In Advances in Cryptology – Crypto'90, pages 619-624, Springer-Verlag, New-York, 1991.
7. A. FIAT & A. SHAMIR, *How to prove yourself: Practical solutions of identification and signature problems*, Advances in Cryptology: Proceedings of Crypto'86, Lecture Notes In Computer Science, Springer-Verlag, Berlin, 263 (1987), pp 186–194.
8. P. MONTGOMERY, *Modular multiplication without trial division*, Mathematics of Computation, vol. 44 (170), pp. 519–521 1985.
9. D. NACCACHE, *Can OSS be repaired ?*, Advances in Cryptology: Proceedings of Eurocrypt'93 , Lecture Notes in Computer Science, Springer-Verlag, to appear.
10. National Institute of Standards and Technology, Publication XX: announcement and specifications for a digital signature standard (DSS), Federal Register, August 19, 1992.

11. J.J. QUISQUATER & L. GUILLOU, *A practical zero-knowledge protocol fitted to security microprocessor minimising both transmission and memory*, Advances in cryptology: Proceedings of Eurocrypt'88 (C. Günter, ed.), Lecture Notes in Computer Science, Springer-Verlag, Berlin, 330 (1988), pp 123–128.
12. R. RIVEST, A. SHAMIR & L. ADLEMANN, *A method for obtaining digital signatures and public-key cryptosystems*, CACM, vol. 21 (1978), pp. 120–126.
13. C. SCHNORR, *Efficient identification and signatures for smart-cards*, Advances in Cryptology: Proceedings of Eurocrypt'89 (G. Brassard ed.), Lecture Notes in computer science, Springer-Verlag, Berlin, 435 (1990), pp. 239–252.
14. M. SHAND & J. VUILLEMIN, *Fast implementations of RSA cryptography*, 11th IEEE Symposium on Computer Arithmetic, 1993. To appear.