The Montgomery Powering Ladder

Marc Joye[1] and Sung-Ming Yen[2,⋆]

[1] Gemplus Card International, Card Security Group
Parc d'Activités de Gémenos, B.P. 100, 13881 Gémenos Cedex, France
marc.joye@gemplus.com
http://www.gemplus.com/smart/ − http://www.geocities.com/MarcJoye/
[2] Laboratory of Cryptography and Information Security (LCIS)
Dept of Computer Science and Information Engineering
National Central University, Chung-Li, Taiwan 320, R.O.C.
yensm@csie.ncu.edu.tw
http://www.csie.ncu.edu.tw/~yensm/

Abstract. This paper gives a comprehensive analysis of Montgomery powering ladder. Initially developed for fast scalar multiplication on elliptic curves, we extend the scope of Montgomery ladder to any exponentiation in an abelian group. Computationally, the Montgomery ladder has the triple advantage of presenting a Lucas chain structure, of being parallelized, and of sharing a common operand. Furthermore, contrary to the classical binary algorithms, it behaves very regularly, which makes it naturally protected against a large variety of implementation attacks.

Keywords. Exponentiation algorithms, Montgomery powering ladder, constrained environments, cryptographic implementations, fault attacks, side-channel attacks.

1 Introduction

Exponentiation or powering algorithms are of central importance in cryptography as they are the basis of (nearly) all public-key cryptosystems. Although numerous exponentiation algorithms have been devised, algorithms for constrained devices are scarcely restricted to the square-and-multiply algorithm and its right-to-left counterpart. A less-known algorithm due to Peter Montgomery is also not much greedy for memory. Developed for fast elliptic curve multiplication, this algorithm is of full generality and applies to any abelian group. Furthermore, it presents several useful features not available in the classical binary algorithms.

This paper is aimed at giving a thorough analysis of Montgomery ladder, considering both the efficiency and security issues. Among other things, we show how it reduces the memory requirements for elliptic curve computations or how it speeds up by a factor of up to 50% the evaluation of full Lucas sequences. For

⋆ Supported in part by the Ministry of Education of the Republic of China under contract EX-91-E-FA06-4-4.

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 291–302, 2003.

(modular) exponentiation, we show that Montgomery ladder can be combined with the common-multiplicand technique, leading to a 33% speed-up factor. The Montgomery ladder is also prone to parallel implementations; on a bi-processor device, the running time is divided by two, compared to the non-parallel version. Last but not least, Montgomery ladder is a prime choice for a secure exponentiation as its high regularity makes it naturally resistant to various side-channel and fault attacks. A slight variant protected against the M safe-error attack is presented.

The rest of this paper is organized as follows. The next section presents the Montgomery ladder in terms of group-theoretic language. In Section 3, we analyze the efficiency of Montgomery ladder and compare it to the classical binary ladders. Section 4 analyzes the security of Montgomery ladder towards implementation attacks. Finally, we conclude in Section 5.

2 Montgomery Ladder

Originally, the so-called Montgomery ladder [16] was proposed as a means to speed up scalar multiplication in the context of elliptic curves. It has been then rediscovered several times and applied to different settings.

To ease the discussion, we give hereafter a higher description of the algorithm. We consider the general problem of computing $y = g^k$ in a (multiplicatively written) abelian group $\mathbb{G}$, on input g and k. Let $\sum_{i=0}^{t-1} k_i\, 2^i$ be the binary expansion of exponent k. The Montgomery ladder relies on the following observation. Defining $L_j = \sum_{i=j}^{t-1} k_i\, 2^{i-j}$ and $H_j = L_j + 1$, we have

$$L_j = 2L_{j+1} + k_j = L_{j+1} + H_{j+1} + k_j - 1 = 2H_{j+1} + k_j - 2$$

and so we obtain

$$(L_j, H_j) = \begin{cases} (2L_{j+1}, L_{j+1} + H_{j+1}) & \text{if } k_j = 0\,, \\ (L_{j+1} + H_{j+1}, 2H_{j+1}) & \text{if } k_j = 1\,. \end{cases} \tag{1}$$

Suppose that, at each iteration, a first register, say R_0, is used to contain the value of g^{L_j} and that a second register, say R_1, is used to contain the value of g^{H_j}. Equation (1) implies that

$$(g^{L_j}, g^{H_j}) = \left((g^{L_{j+1}})^2, g^{L_{j+1}} \cdot g^{H_{j+1}}\right) \quad \text{if } k_j = 0$$

and

$$(g^{L_j}, g^{H_j}) = \left(g^{L_{j+1}} \cdot g^{H_{j+1}}, (g^{H_{j+1}})^2\right) \quad \text{if } k_j = 1\ .$$

Remarking that $L_0 = k$, this leads to an elegant algorithm for evaluating $y = g^k$: the Montgomery ladder.

For cryptographic applications, the group $\mathbb{G}$ may be taken as ${\mathbb{Z}_N}^*$ (e.g., for RSA or Rabin encryption/signature), ${\mathbb{F}_q}^*$ (e.g., for DH key exchange), the elements of a Lucas sequences (e.g., for LUC signature), the points of an elliptic curve (e.g., for ECDSA signature), ... Other practical applications include fast primality tests and factorization algorithms.

```
Input: g, k = (k_{t-1}, ..., k_0)_2
Output: y = g^k
R_0 ← 1; R_1 ← g
for j = t − 1 downto 0 do
  if (k_j = 0) then
    R_1 ← R_0 R_1; R_0 ← (R_0)^2
  else [if (k_j = 1)]
    R_0 ← R_0 R_1; R_1 ← (R_1)^2
return R_0
```

Fig. 1. Montgomery Ladder.

3 Efficiency Analysis

The most widely used algorithm for computing g^k are the square-and-multiply algorithm, which processes the bits of exponent k from the left to the right (Fig. 2 (a)), and its right-to-left counterpart (Fig. 2 (b)). We restrict our attention to constrained environments and do not consider more sophisticated exponentiation algorithms (see [7] for a survey).

```
Input: g, k = (k_{t-1}, ..., k_0)_2
Output: y = g^k
R_0 ← 1; R_1 ← g
for j = t − 1 downto 0 do
  R_0 ← (R_0)^2
  if (k_j = 1) then R_0 ← R_0 R_1
return R_0
```

(a) Left-to-right binary algorithm.

```
Input: g, k = (k_{t-1}, ..., k_0)_2
Output: y = g^k
R_0 ← 1; R_1 ← g
for j = 0 to t − 1 do
  if (k_j = 1) then R_0 ← R_0 R_1
  R_1 ← (R_1)^2
return R_0
```

(b) Right-to-left binary algorithm.

Fig. 2. Classical Binary Ladders.

From a computational perspective, the Montgomery ladder (Fig. 1), in its basic version, appears inferior to the classical binary algorithms as it requires $2t$ multiplications instead of $1.5t$ multiplications, on average. Nevertheless, in some cases, the Montgomery ladder may reveal itself more efficient by observing that

1. the value R_1/R_0 is invariant throughout the algorithm (and so equals g);
2. at each iteration, the two multiplications are independent;
3. at each iteration, the two multiplications share a common operand.

3.1 Lucas Chains

The key property of Montgomery ladder (Fig. 1) is that the relation $R_1/R_0 = g$ (or equivalently, $R_1 = R_0\, g$) is maintained invariant. This was noticed by

Montgomery [16,17] and applied to the ECM factorization method for a special class of elliptic curves.

```
Input: G, k = (1, k_{t-2}, ..., k_0)_2
Output: Y = kG
R_0 ← G; R_1 ← 2G
for j = t − 2 downto 0 do
  if (k_j = 0) then
    R_1 ← R_0 + R_1; R_0 ← 2R_0
  else [if (k_j = 1)]
    R_0 ← R_0 + R_1; R_1 ← 2R_1
return R_0
```

Fig. 3. Montgomery Ladder for Elliptic Curves.

Let $\boldsymbol{R_0}$ and $\boldsymbol{R_1} \in E(\mathbb{K})$ be two points on an elliptic curve E defined over a field $\mathbb{K}$. If the difference $\boldsymbol{G} := \boldsymbol{R_1} - \boldsymbol{R_0}$ is known then the x-coordinate of point $\boldsymbol{Y} = k\boldsymbol{G}$ can be computed from the x-coordinate of $\boldsymbol{R_0}$, the x-coordinate of point $\boldsymbol{R_1}$ and the x- and y-coordinates of point $\boldsymbol{G}$ [16]. Agnew *et al.* [2] (see also [13]) later observed (for binary fields $\mathbb{K}$) that the y-coordinate of $\boldsymbol{R_0}$ can easily be recovered when point $\boldsymbol{G}$ and the x-coordinates of $\boldsymbol{R_0}$ and of $\boldsymbol{R_0} + \boldsymbol{G}$ $(= \boldsymbol{R_1})$ are known. This was extended to fields $\mathbb{K}$ of characteristic $p > 3$ in [18,19] (see also [3,6,8] for general Weierstraß elliptic curves).

Because the computations can be carried out with the x-coordinates only, a lot of multiplications (in field $\mathbb{K}$) are saved, resulting in an algorithm faster than the classical binary algorithms (Fig. 2). Additionally, fewer memory is required since the y-coordinates need not to be handled (and thus stored) during the computation of $x(k\boldsymbol{G})$. The y-coordinate of $k\boldsymbol{G}$, $y(k\boldsymbol{G})$, is computed at the end of the algorithm from $\boldsymbol{G}$, $x(k\boldsymbol{G})$ and $x(k\boldsymbol{G} + \boldsymbol{G})$.

A similar technique exists for Lucas sequences. The special Lucas sequence $\{V_k(P, 1)\}$ with parameter $Q = 1$ is considered in [27] and the general case, $\{V_k(P, Q)\}$ along with the 'sister' sequence $\{U_k(P, Q)\}$ is addressed in [9] (see also [1, Section A.2.4]).

Analogously to elliptic curves, given $(V_1, U_1) = (P, 1)$, V_k and V_{k+1}, the value of U_k can be recovered as $U_k = (2V_{k+1} - P\,V_k)/\Delta$ with $\Delta = P^2 - 4Q$. Provided that division by Δ is inexpensive or that the value of Δ^{-1} is precomputed, this saves one multiplication per iteration compared to [9], resulting in a 22% improvement in the general case and a 50% improvement when $Q = 1$.

3.2 Parallel Computing

A second property of Montgomery ladder is its intrinsic disposition of being parallelized. This feature may reveal very useful in the near future as recent

```
Input: P, Q, k = (k_{t-1}, ..., k_0)_2
Output: y = V_k(P, Q)
V_0 ← 2; V_1 ← P; q_0 ← 1; q_1 ← 1
for j = t − 1 downto 0 do
  q_0 ← q_0 q_1
  if (k_j = 0) then
    q_1 ← q_0
    V_1 ← V_0 V_1 − P q_0; V_0 ← V_0^2 − 2q_0
  else [if (k_j = 1)]
    q_1 ← Q q_0
    V_0 ← V_0 V_1 − P q_0; V_1 ← V_1^2 − 2q_1
return V_0
```

Fig. 4. Montgomery Ladder for Lucas Sequences.

cryptographic tokens come equipped with several arithmetical co-processors [6, Section 5].

To exhibit the parallel nature of Montgomery ladder, we simplify the presentation of Fig. 1. Using k_j and its negation $\neg k_j$ as register indexes, the two different cases can be rewritten into a single statement as

$$R_{\neg k_j} \leftarrow R_0\, R_1; R_{k_j} \leftarrow (R_{k_j})^2 \ .$$

Hence, we clearly see that the two multiplications can be evaluated independently.

```
Input: g, k = (k_{t-1}, ..., k_0)_2
Output: y = g^k
R_0 ← 1; R_1 ← g
for j = t − 1 downto 0 do
   /* Processor 1 */ | /* Processor 2 */
   R_{¬k_j} ← R_0 R_1 | R_{k_j} ← (R_{k_j})^2
return R_0
```

Fig. 5. Parallel Montgomery Ladder.

For a modular exponentiation, if we denote by M the time for performing a multiplication, an optimized squaring takes roughly $0.8M$. So, on a bi-processor device, each iteration is performed in time M. As a result, the parallel version of the Montgomery ladder nearly attains the optimal 200% speed-up factor, over the standard Montgomery ladder (Fig. 1), for an RSA-type implementation. For elliptic curve implementations, the addition of two points or the doubling are further dissimilar [8], so that the expected gain seems sub-optimal; it is however possible to combine the operations of addition and doubling to lower the number of (field) multiplications [6] to nearly obtain the 200% speed-up factor.

3.3 Common-Multiplicand Multiplication

A third property of Montgomery ladder is that the two multiplications share a common operand: both multiplications involve R_0 when $k_j = 0$ and R_1 when $k_j = 1$. The 'common-multiplicand multiplication' method [26] is thus applicable. The method was initially developed to speed up the right-to-left binary algorithm (Fig. 2 (b)). Generalizations and improvements can be found in [22, 23].

The basic idea consists in rewriting the two involved multiplications with logical operators. Defining $R_{com} = (R_0 \,\mathsf{AND}\, R_1)$ and $R_{i,c} = (R_i \,\mathsf{XOR}\, R_{com})$, we have

$$R_i = R_{i,c} + R_{com}, \quad i \in \{0, 1\} \ . \tag{2}$$

Assume $k_j = 1$ (the case $k_j = 0$ is similar). Then the Montgomery ladder requires the computation of $R_0 \leftarrow R_0 R_1$ and $R_1 \leftarrow R_1 R_1$. From Eq. (2), this can be evaluated as $R_0 \leftarrow R_{0,c} R_1 + R_{com} R_1$ and $R_1 \leftarrow R_{1,c} R_1 + R_{com} R_1$. On average, the Hamming weight (i.e., the number of nonzero bits) of R_{com} and $R_{i,c}$ is twice smaller to that of R_i [27]. Consequently, each multiplication now requires half less binary additions, on average, that is, a 33% expected gain since the common multiplication $R_{com} R_1$ is only evaluated once.

For a modular exponentiation, the common-multiplicand method is particularly suited in certain hardware realizations (when logical operations can be processed in parallel with arithmetical operations). When the group law is more involved (as on elliptic curves), it may lead to software optimizations as well as several common (basic) operations (e.g., a field multiplication) may be saved [15].

4 Security Analysis

This section analyzes the security of Montgomery ladder towards implementation attacks. We distinguish two types of implementation attacks: side-channel attacks and fault attacks.

4.1 Side-Channel Attacks

Side-channel attacks are based on the observation that some side-channel information (e.g., timing [12] or power consumption [11]) depends on the instruction being executed and/or the data being handled.

The standard binary ladders (Fig. 2) contains conditional branchings. If the conditional branching is driven by secret data (namely, if the bits of exponent k in the computation of $y = g^k$ are secret) and if the two branches behave differently regarding some side-channel analysis (e.g., simple power analysis (SPA)) then secret data can be retrieved. To this end, dummy operations are added to the basic algorithms, so that they behave more regularly [4].

As it clearly appears in the next figure, the Montgomery ladder is already highly regular. Whatever the processed bit, there is always a multiplication followed by a squaring.

```
Input: g, k = (k_{t-1}, ..., k_0)_2
Output: y = g^k
R_0 ← 1; R_2 ← g
for j = t − 1 downto 0 do
  b ← ¬k_j
  R_0 ← (R_0)^2; R_b ← R_b R_2
return R_0
```

(a) Left-to-right binary algorithm.

```
Input: g, k = (k_{t-1}, ..., k_0)_2
Output: y = g^k
R_0 ← 1; R_2 ← g
for j = 0 to t − 1 do
  b ← ¬k_j
  R_b ← R_0 R_2; R_2 ← (R_2)^2
return R_0
```

(b) Right-to-left binary algorithm.

Fig. 6. (Simple) Side-Channel Protected Classical Binary Ladders.

```
Input: g, k = (k_{t-1}, ..., k_0)_2
Output: y = g^k
R_0 ← 1; R_1 ← g
for j = t − 1 downto 0 do
  R_{¬k_j} ← R_0 R_1; R_{k_j} ← (R_{k_j})^2
return R_0
```

Fig. 7. (Simple) Side-Channel Protected Montgomery Ladder.

Provided that the writing in registers R_0 and R_1 (resp. that the squaring of registers R_0 and R_1) cannot be distinguished from a *single* side-channel measurement, the Montgomery ladder can be implemented to prevent a given [simple] side-channel attack. It is worth noting that protections against simple side-channel attacks do not ward off the differential attacks, consisting in acquiring *several* side-channel measurements of different executions of the same algorithm and after that in performing some statistical treatment. For example, the attack of [4, § 3.2] conducted against the protected standard binary ladders (Fig. 6) readily applies the above protected Montgomery ladder. However, standard blinding techniques (e.g., [14,4]) easily prevent differential attacks.

Memory-wise, compared to the protected standard binary ladders, the protected Montgomery ladder requires one less register. Furthermore, it enjoys the useful features mentioned in Section 3.

Remark 1. The Montgomery ladder for Lucas sequences (see Fig. 4) does not behave regularly. This is however not a issue for cryptographic applications as known cryptosystems based on Lucas sequences ([20,21]) use for parameter Q the value $Q = 1$. Variables q_0 and q_1 are therefore useless.

4.2 Fault Attacks

An important lesson taught in [25] is that countermeasures must be considered globally (see also [10]). This was illustrated with the C safe-error attack in [25] and with the M safe-error in [24]. The next paragraphs analyze the security of the

Montgomery ladder regarding to the C and M safe-error attacks and highlight the interplay between different countermeasures.

Security against C safe-error attack. It was well known that a countermeasure developed against one implementation attack does not necessarily thwart another kind of implementation attack automatically. More surprisingly, in [25], it was pointed out that a countermeasure developed against a given attack, if not carefully examined, may benefit another physical attack tremendously. In that paper, a new type of computational safe-error attack (called a C safe-error attack) was mounted against the classical, side-channel protected exponentiation algorithms of Fig. 6. The C safe-error attack is developed by inducing *any* temporary random computational fault(s) inside the ALU.

It is easy to see that the protected algorithm of Fig. 6 (a) (commonly known as the square-and-multiply-always exponentiation algorithm) is susceptible to a C safe-error attack. This follows by observing that since the algorithm runs in constant time, an attacker can more easily locate the exact moment of the second multiplication "$R_b \leftarrow R_b\, R_2$" for each iteration. Moreover, when the current exponent bit, say k_j, is equal to 0, then this multiplication is a dummy operation and so has no influence on the final result. Therefore, if an attacker induces any kind of computational fault into the ALU during the operation of $R_b \leftarrow R_b\, R_2$ at jth iteration, then according to whether the final result is incorrect or not, she may deduce if $k_j = 1$ or $k_j = 0$. We note however that this attack only reveal one bit of exponent k and may be made, in some circumstances, much harder by randomizing exponent k prior to the exponentiation.

The same attack holds for the right-to-left protected algorithm of Fig. 6 (b).

For the Montgomery ladder (Fig. 7), the situation is different as there are no dummy operations. Consequently, any fault induced into the ALU will result in an incorrect exponentiation result.

Security against M safe-error attack. The M safe-error pointed out in [24] can be illustrated on the modular multiplication, $B \leftarrow A \cdot B \bmod N$, by calling the program routine listed in Fig. 8 as $B \leftarrow \texttt{MUL}(A, B, N)$. In this routine, it is assumed that multiplier B is represented in a 2^T-ary form as $B = \sum_{j=0}^{m-1} B_j\, (2^T)^j$, and both multiplicand A and multiplier B are sent to the routine `MUL` by passing their location address (i.e., the call by address technique). This call by address assumption is reasonable since it is popular for both high-level programming language (e.g., C) and all instruction-level language implementations.

The idea behind the M safe-error can be understood as follows. The value of multiplier B will be correct after the assignment operation $B \leftarrow A \cdot B \bmod N$, *even* if some blocks B_j (or Y_j with the notations of Fig. 8) of the multiplier are modified after they have been employed in the modular multiplication algorithm. As suggested in [24], this M safe-error can be avoided if B is assigned as the multiplicand, i.e., by calling the routine as $B \leftarrow \texttt{MUL}(B, A, N)$. It should be noted that the M safe-error attack needs to induce a temporary memory fault inside a register or memory location. Compared to the C safe-error attack, this

```
Input: X, Y, N
Output: R = MUL(X, Y, N)
R ← 0
for j = m − 1 downto 0 do
   R ← (R · 2^T + X · Y_j) mod N
output R
```

Fig. 8. M Safe-Error on Interleaved Modular Multiplication.

implies stronger cryptanalytic assumptions, namely a higher controllability of fault location and timing.

As presented in Fig. 7, the Montgomery ladder for modular exponentiation is vulnerable to the M safe-error attack, no matter R_0 or R_1 is passed to the routine as the multiplier in the multiplication $R_{\neg k_j} \leftarrow R_0\, R_1$. To prove above claim, we consider the two following possible implementations. Suppose first that R_1 is assigned as the multiplier (that is, exactly the algorithm of Fig. 7): $[R_{\neg k_j} \leftarrow R_0\, R_1\,;\ R_{k_j} \leftarrow (R_{k_j})^2]$. Within this design, when $k_j = 1$, the two operations at iteration j are $R_0 \leftarrow R_0\, R_1$ and $R_1 \leftarrow (R_1)^2$. Any error induced into R_1 cannot be an M safe-error. On the other hand, when $k_j = 0$, the two operations are $R_1 \leftarrow R_0\, R_1$ and $R_0 \leftarrow (R_0)^2$. An error carefully induced into the higher part of R_1 is an M safe-error (because the error in register R_1 is cleared after the assignment $R_1 \leftarrow R_0\, R_1$) and so do not influence the computation. Based on the two distinct behaviors, an attacker can recover the value of bit k_j. Likewise, if R_0 is now assigned as the multiplier, depending on whether of an error carefully induced into R_0 at iteration j is an M safe-error or not, an attacker can recover the value of bit k_j.

As mentioned in § 3.2, the Montgomery ladder can be implemented in parallel when two ALU's are available. It can be easily verified that the above M safe-error attack still applies in this parallelized implementation if these two ALU's share the source information of R_0 and R_1.

It is fairly easy to modify Montgomery ladder in order to counteract the aforementioned M safe-error attack. It suffices to perform $R_{\neg k_j} \leftarrow R_{\neg k_j}\, R_{k_j}$ at each iteration instead of $R_{\neg k_j} \leftarrow R_0\, R_1$ or $R_{\neg k_j} \leftarrow R_1\, R_0$.

```
Input: g, k = (k_{t−1}, ..., k_0)_2
Output: y = g^k
R_0 ← 1; R_1 ← g
for j = t − 1 downto 0 do
   b ← ¬k_j
   R_b ← R_b R_{k_j}; R_{k_j} ← (R_{k_j})^2
return R_0
```

Fig. 9. (Simple) Side-Channel and M Safe-Error Protected Montgomery Ladder.

When $k_j = 0$ (and $b = 1$), $R_1 \leftarrow R_1 R_0$ is executed (by calling the routine $R_1 \leftarrow \texttt{MUL}(R_1, R_0)$ with R_0 as multiplier). On the other hand, when $k_j = 1$ (and $b = 0$), $R_0 \leftarrow R_0 R_1$ is executed (by calling the routine $R_0 \leftarrow \texttt{MUL}(R_0, R_1)$ with R_1 as multiplier). It can be verified that no matter $k_j = 0$ or $k_j = 1$, any error induced into R_0 or R_1 cannot be an M safe-error. The proposed modification foils thus the above M safe-error attack.

5 Conclusion

This paper gave a generic description of Montgomery ladder in an abelian group $\mathbb{G}$. It thoroughly analyzed its main features for fast computation and secure implementation on constrained devices. We hope having convinced the reader that Montgomery ladder may be a first-class substitute of the celebrated square-and-multiply algorithm.

References

1. IEEE Std 1363-2000. *IEEE Standard Specifications for Public-Key Cryptography*. IEEE Computer Society, August 29, 2000.
2. G.B. Agnew, R.C. Mullin, and S.A. Vanstone. An implementation of elliptic curve cryptosystems over $F_{2^{155}}$. *IEEE Journal on Selected Areas in Communications*, 11(5):804–813, June 1993.
3. Éric Brier and Marc Joye. Weierstraß elliptic curves and side-channel attacks. In D. Naccache and P. Paillier, editors, *Public Key Cryptography*, volume 2274 of *Lecture Notes in Computer Science*, pages 335–345. Springer-Verlag, 2002.
4. Jean-Sébastien Coron. Resistance against differential power analysis for elliptic curve cryptosystems. In Ç.K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES '99)*, volume 1717 of *Lecture Notes in Computer Science*, pages 292–302. Springer-Verlag, 1999.
5. Richard Crandall and Carl Pomerance. *Prime Numbers: A Computational Perspective*. Springer-Verlag, 2001.
6. Wieland Fischer, Christophe Giraud, Erik Woodward Knudsen, and Jean-Pierre Seifert. Parallel scalar multiplication on general elliptic curves over $\mathbb{F}_p$ hedged against non-differential side-channel attacks. Report 2002/007, Cryptology ePrint Archive, January 2002.
7. Daniel M. Gordon. A survey of fast exponentiation methods. *Journal of Algorithms*, 27:129–146, 1998.
8. Tetsuya Izu and Tsuyoshi Takagi. A fast parallel elliptic curve multiplication resistant against side channel attacks. In D. Naccache and P. Paillier, editors, *Public Key Cryptography*, volume 2274 of *Lecture Notes in Computer Science*, pages 280–296. Springer-Verlag, 2002.
9. Marc Joye and Jean-Jacques Quisquater. Efficient computation of full Lucas sequences. *Electronics Letters*, 32(6):537–538, March 1996.
10. Marc Joye, Jean-Jacques Quisquater, Sung-Ming Yen, and Moti Yung. Observability analysis: Detecting when improved cryptosystems fail. In B. Preneel, editor, *Topics in Cryptology – CT-RSA 2002*, volume 2271 of *Lecture Notes in Computer Science*, pages 17–29. Springer-Verlag, 2002.

11. Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In M. Wiener, editor, *Advances in Cryptology – CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer-Verlag, 1999.
12. Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In N. Koblitz, editor, *Advances in Cryptology – CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer-Verlag, 1996.
13. Julio López and Ricardo Dahab. Fast multiplication on elliptic curves over $GF(2^m)$ without precomputation. In Ç.K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems*, volume 1717 of *Lecture Notes in Computer Science*, pages 316–327. Springer-Verlag, 1999.
14. Thomas S. Messerges, Ezzy A. Dabbish, and Robert H. Sloan. Power analysis attacks of modular exponentiation in smartcards. In Ç.K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES '99)*, volume 1717 of *Lecture Notes in Computer Science*, pages 144–157. Springer-Verlag, 1999.
15. Atsuko Miyaji, Takatoshi Ono, and Henri Cohen. Efficient elliptic curve exponentiation. In Y. Han, T. Okamoto, and S. Qing, editors, *Information and Communications Security (ICICS '97)*, volume 1334 of *Lecture Notes in Computer Science*, pages 282–290. Springer-Verlag, 1997.
16. Peter L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48(177):243–264, January 1987.
17. Peter L. Montgomery. Evaluating recurrences of form $X_{m+n} = f(X_m, X_n, X_{m-n})$ via Lucas chains. Unpublished manuscript, January 1992.
18. Katsuyuki Okeya, Hiroyuki Kurumatani, and Kouichi Sakurai. Elliptic curves with the Montgomery form and their cryptographic applications. In H. Imai and Y. Zheng, editors, *Public Key Cryptography*, volume 1751 of *Lecture Notes in Computer Science*, pages 238–257. Springer-Verlag, 2000.
19. Katsuyuki Okeya and Kouichi Sakurai. Efficient elliptic curve cryptosystems from a scalar multiplication algorithm with recovery of the y-coordinate on a Montgomery-form elliptic curve. In Ç.K. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 126–141. Springer-Verlag, 2001.
20. P. Smith. Cryptography without exponentiation. *Dr. Dobb's Journal*, (4):26–30, April 1994.
21. Peter J. Smith and Michael J.J. Lennon. LUC: A new public key system. In E.G. Douglas, editor, *Ninth IFIP Symposium on Computer Security*, pages 103–117. Elsevier Science Publishers, 1993.
22. Tzong-Chen Wu and Yuh-Shihng Chang. Improved generalisation of common-multiplicand algorithm of Yen and Laih. *Electronics Letters*, 31(20):1738–1739, September 1995.
23. Sung-Ming Yen. Improved common-multiplicand multiplication and fast exponentiation by exponent decomposition. *IEICE Trans. Fundamentals*, E80-A(6):1160–1163, June 1997.
24. Sung-Ming Yen and Marc Joye. Checking before output may not be enough against fault-based cryptanalysis. *IEEE Trans. on Computers*, 49(9):967–970, September 2000.
25. Sung-Ming Yen, Seung-Joo Kim, Seon-Gan Lim, and Sang-Jae Moon. A countermeasure against one physical cryptanalysis may benefit another attack. In K. Kim, editor, *Information Security and Cryptology – ICISC 2001*, volume 2288 of *Lecture Notes in Computer Science*, pages 414–427. Springer-Verlag, 2002.

26. Sung-Ming Yen and Chi-Sung Laih. Common-multiplicand multiplication and its application to public-key cryptography. *Electronics Letters*, 29(17):1583–1584, August 1993.
27. Sung-Ming Yen and Chi-Sung Laih. Fast algorithms for LUC digital signature computation. *IEE Proc.-Comput. Digit Tech.*, 142(2):165–169, March 1995.