Efficient Subgroup Exponentiation in Quadratic and Sixth Degree Extensions

Martijn Stam[1,⋆] and Arjen K. Lenstra[2]

[1] Technische Universiteit Eindhoven
P.O.Box 513, 5600 MB Eindhoven, The Netherlands
stam@win.tue.nl
[2] Citibank, N.A. and Technische Universiteit Eindhoven
1 North Gate Road, Mendham, NJ 07945-3104, U.S.A.
arjen.lenstra@citigroup.com

Abstract. This paper describes several speedups for computation in the order $p+1$ subgroup of $\mathbf{F}_{p^2}^*$ and the order p^2-p+1 subgroup of $\mathbf{F}_{p^6}^*$. These results are in a way complementary to LUC and XTR, where computations in these groups are sped up using trace maps. As a side result, we present an efficient method for XTR with $p \equiv 3 \bmod 4$.

Keywords: XTR, LUC, finite field, cyclotomic polynomial.

1 Introduction

Many cryptographic protocols rely on the assumed hardness of the discrete logarithm problem in certain groups. Well known examples are prime order subgroups of $\mathbf{Z}_p^*$ or of elliptic curves. Let G_x, for a positive integer x, denote a cyclic (sub)group of order x. In this paper we focus on subgroups G_q of $\mathbf{F}_{p^d}^*$ with q a prime dividing the d-th cyclotomic polynomial Φ_d evaluated at p. The cryptographic relevance of these groups was already pointed out in [11]: other subgroups of $\mathbf{F}_{p^d}^*$ can be embedded in a true subfield of $\mathbf{F}_{p^d}^*$, thereby making the discrete logarithm computation substantially easier.

Computation in finite fields is a well studied problem. However, research tends to emphasize on bilinear complexity [10], asymptotic complexity [24], or binary characteristic [1]. The case of large prime characteristic with small extension degree has been studied less extensively [5,11,2]. Moreover, usually the entire field is discussed, while hardly any attempt is made to look closely at the cryptographically interesting cyclotomic subgroup. An exception is the aforementioned article [11], but there the problem is not addressed in full detail. In this paper, we consider the groups $\mathsf{G}_{p+1} \subset \mathbf{F}_{p^2}^*$ and $\mathsf{G}_{p^2-p+1} \subset \mathbf{F}_{p^6}^*$.

Currently the fastest exponentiation methods in the subgroups G_{p+1} and G_{p^2-p+1} use trace maps, resulting in respectively LUC [20] and XTR [13]. They have the additional benefit of reducing the size of the representation of subgroup elements to a half respectively a third of the traditional representation. Application of LUC and XTR is advantageous in protocols where the subgroup operations are restricted to additions, and single or double exponentiations. But for

⋆ The first author is sponsored by STW project EWI.4536

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 318–332, 2003.

more involved protocols that also require ordinary multiplications of subgroup elements or triple (or larger) exponentiations, they may lead to cumbersome manipulations that outweigh the computational advantages. As a consequence, using trace based representations in more complicated protocols may be inconvenient (unless of course the small representation size is crucial).

For that reason, we consider in this paper how exponentiation speedups in G_{p+1} and G_{p^2-p+1} can be achieved in such a way that other operations are not affected, i.e., while avoiding trace based compression methods. For quadratic extensions we show that for both $p \equiv 2 \bmod 3$ and $p \equiv 3 \bmod 4$ inversions in $\mathsf{G}_{p+1} \subset \mathbf{F}_{p^2}^*$ come for free, and that squaring in G_{p+1} is cheaper than in the field $\mathbf{F}_{p^2}$. This results in single and double exponentiations that cost about 60% and 75%, respectively, of traditional methods. Both methods are still considerably slower than LUC (see also [22]).

Our main result concerns sixth degree extensions, i.e., the case $\mathsf{G}_{p^2-p+1} \subset \mathbf{F}_{p^6}^*$. We show that for both $p \equiv 2 \bmod 9$ and $p \equiv 5 \bmod 9$ inversions in G_{p^2-p+1} are very cheap, while squaring in G_{p^2-p+1} is substantially faster than in $\mathbf{F}_{p^6}$. Moreover, the methods from [8,22] can be used to transform a k-bit single exponentiation into a $k/2$-bit double exponentiation (i.e., the product of two $k/2$-bit exponentiations). Using appropriate addition chains this results in a vastly improved single exponentiation routine, that takes approximately 26% of the time cited in [13, Lemma 2.1.2.*iii*]. The improvement for double exponentiation is less spectacular, requiring an estimated 33% compared to [13, Lemma 2.1.2.*iv*]. Our methods are slightly slower than the improved version of XTR [22], but faster than the original XTR [13].

Our proposed methods do not have the compressed-representation benefits or disadvantages of LUC or XTR. Protocols where our methods compare well to LUC and XTR are especially those based on homomorphic ElGamal encryption [7] such as Brands' protocols [3] and Schoenmakers' verifiable secret sharing scheme [19]. Another example is the Cramer-Shoup protocol [6].

Another consideration is the cost of subgroup membership checking, since the security of several cryptographic protocols stands or falls with the correctness of the generators and proper subgroup membership of other elements. For LUC the cost of the subgroup membership test is negligible. For XTR it is small but not really negligible. Testing membership of G_{p+1} and G_{p^2-p+1} as proposed in this paper only costs a small constant number of operations in the underlying field and is thus negligible, as in LUC.

The proposed methods can also be used in conjunction with LUC and XTR. Given an element in G_{p+1} or G_{p^2-p+1} the cost of computing the LUC respectively XTR representation is negligible. Going from LUC to G_{p+1} requires a square root computation in $\mathbf{F}_p$, going from XTR to G_{p^2-p+1} can be done by computing the roots of a third degree polynomial over $\mathbf{F}_{p^2}$. In both cases extra information is needed to resolve root ambiguities.

Unless indicated otherwise, all logarithms in this paper are natural.

2 Preliminaries

2.1 Computational Model

Throughout this paper we use the following conventions to measure the costs of operations. Let l be a positive integer that will be clear from the context. We use M for the cost of multiplying two l-bit numbers (without modular reduction), S for the cost of squaring an l-bit number (idem), D for reducing a $2l$-bit number modulo an l-bit number, A_1 for adding two l-bit numbers (including a reduction if needed), and A_2 for adding two $2l$-bit numbers (no reduction). A modular addition (of cost A_1) typically boils down to two or three plain l-bit additions (which makes it hard to determine whether $A_1 > A_2$ or vice versa). Consequently, the stated numbers of additions should be taken with a grain of salt. As another example, in Lemma 3.24.*iv* the cost of subgroup squaring is approximated as $2S + 2D + A_1$, assuming that the cost of subtracting one or multiplying by two is negligible compared to A_1 and A_2. Furthermore, the reduction (of cost D) is sometimes fed numbers slightly larger than $2l$-bits.

Anyway, for exponentiations we always switch back to the simplified case $A_1 = A_2 = 0$, $M = D = 0.5$, and $S = 0.3$, assuming some fixed value for l. This corresponds to the model where an l-bit modular multiplication is the unit of measurement, a squaring costs 80% of a modular multiplication, and additions are considered negligible. This simplification facilitates comparisons with other results given in the literature.

2.2 Discrete Logarithm Problem

In this paper it is assumed that the discrete logarithm problem in the order q subgroup G_q of $\mathbf{F}_{p^d}^*$ is sufficiently difficult. Here we briefly review the well known implications of this assumption for the choice of q given p^d.

It follows from the Pohlig-Hellman algorithm [15] that q is best chosen as a prime number. Furthermore, it follows from the Pollard-ρ method [16,23] that $\sqrt{q}$ should be sufficiently large, say at least 2^{80} or 2^{100} depending on the security requirements. Finally, it was shown in [11] that q divides $\Phi_d(p)$ if and only if G_q cannot be embedded in a true subfield of $\mathbf{F}_{p^d}$, under the assumption that $q > d$. This implies that, if a sufficiently large q divides $\Phi_d(p)$, then index calculus method attacks on the discrete logarithm problem in G_q cannot be mounted in any true subfield of $\mathbf{F}_{p^d}$ but must take place in the field $\mathbf{F}_{p^d}$. Thus, such attacks can be expected to take time

$$\exp((1.923 + o(1))(\log p^d)^{1/3}(\log\log p^d)^{2/3}),$$

for $p \to \infty$ and d fixed [9,17,18].

Summarizing, we find that the order q must be a prime of at least, say, 160 bits, irrespective of the value of d. For $d = 2$ we have the additional requirement that q divides $\Phi_2(p) = p + 1$ and that the bit length of the prime p is at least, say, 512. For $d = 6$ the order q divides $\Phi_6(p) = p^2 - p + 1$ and p must be a prime of bit length at least, say, 170.

2.3 Finite Field Representation

In cryptography, d-th degree extensions of finite fields are most commonly represented using either polynomial or normal bases (see [14] for definitions and details). With a proper choice of minimal polynomial (such as a trinomial with small coefficients), polynomial bases allow relatively efficient multiplication and squaring in the sense that the usual reduction stage from a degree $2d-2$ product to the degree $d-1$ result can be performed at the cost of cd additions in the underlying prime field, for a very small constant c. In general, this is not the case for normal bases, but they have the advantage that the Frobenius automorphism can be computed for free. For polynomial bases the Frobenius automorphism can be computed at a small but non-negligible cost. A class of polynomial bases combining the best of both worlds is featured in [5]. They are based on cyclotomic fields. The following theorem, a slight adaptation of [14, Theorem 2.47(ii)], says something about the extension degrees one obtains using cyclotomic fields.

Theorem 2.31 *Given a field $\mathbf{F}_{p^e}$ with p prime and some n coprime to p. Then the n-th cyclotomic field over $\mathbf{F}_{p^e}$ is isomorphic to $\mathbf{F}_{p^{ed}}$ where d is the least positive integer such that $p^{ed} \equiv 1 \bmod n$.*

This theorem implies $d|\phi(n)$. We fix $e = 1$. Furthermore, we concentrate on $d = \phi(n)$, i.e., the case that $p \bmod n$ generates $\mathbf{Z}_n^*$. This requires $\mathbf{Z}_n^*$ to be cyclic, so that n is either 2, 4, the power of an odd prime, or twice the power of an odd prime. We ignore $n = 2$, since it does not lead to a proper extension.

Actually, [5] is concerned with rings $\mathbf{Z}[\gamma]/p\mathbf{Z}[\gamma]$ where n is a prime power, γ is a primitive n-th root of unity, and p is an integer of which primality is to be determined. If p is indeed a prime generating $\mathbf{Z}_n^*$ then $\mathbf{Z}[\gamma]/p\mathbf{Z}[\gamma]$ is isomorphic to $\mathbf{F}_p[\gamma]$ supporting identical representations.

Let $\Gamma = (\gamma, \gamma^2, \ldots, \gamma^d)$ with γ as above, then Γ is a basis of $\mathbf{F}_{p^d}$ over $\mathbf{F}_p$. It is understood that an element $a \in \mathbf{F}_{p^d}$ is represented as $\bar{a} = (a_0, \ldots, a_{d-1}) \in (\mathbf{F}_p)^d$, where $a = \Gamma \cdot \bar{a}^T$. We abuse notation by identifying a and $\bar{a}$.

We are interested in finding fast single and double exponentiations for G_q, where $q|\Phi_d(p)$ (cf. Section 2.2). For that purpose we formulate fast multiplication and squaring methods for $\mathbf{F}_{p^d}$, show that squaring in $\mathsf{G}_{\Phi_d(p)}$ can be done even faster, and that the cost of p-th powering in $\mathbf{F}_{p^d}$ (and thus of inversion in $\mathsf{G}_{\Phi_d(p)}$) is virtually negligible. Of independent interest is membership testing for $\mathsf{G}_{\Phi_d(p)}$.

If $d < 105$, then $\Phi_d(p) = \sum_{i\in\mathcal{P}} p^i - \sum_{i\in\mathcal{N}} p^i$ for appropriate index sets $\mathcal{P}$ and $\mathcal{N}$. Let $a \in \mathbf{F}_{p^d}$. Since $\mathbf{F}_{p^d}^*$ is cyclic, $a \in \mathsf{G}_{\Phi_d(p)}$ if and only if $a^{\Phi_d(p)} = 1$, which is equivalent to $\prod_{i\in\mathcal{P}} a^{p^i} = \prod_{i\in\mathcal{N}} a^{p^i}$. Testing this condition requires at most d applications of the Frobenius automorphism and $|\mathcal{P}| + |\mathcal{N}| - 1$ multiplications in $\mathbf{F}_{p^d}$. Thus, for fixed d testing $\mathsf{G}_{\Phi_d(p)}$-membership costs at most $\phi(d)$ multiplications in $\mathbf{F}_{p^d}$. Membership $x \in \mathsf{G}_q$ can be established by verifying that $x^q = 1$.

The relation $\prod_{i\in\mathcal{P}} a^{p^i} = \prod_{i\in\mathcal{N}} a^{p^i}$ gives rise to d possibly dependent relations of degree $|\mathcal{P}|$. In some cases these relations can be used to speed up $|\mathcal{P}|$-th powering in $\mathsf{G}_{\Phi_d(p)}$. This is exploited to get fast squaring in $\mathsf{G}_{\Phi_d(p)}$ for $d = 2, 6$.

A major ingredient when calculating modulo Γ is writing powers $> d$ of γ as linear combinations in Γ. This reduction is performed in two stages. First, all powers higher than n are reduced using $\gamma^n = 1$; next the relation $\Phi_n(\gamma) = 0$ is used to map everything to powers of γ between 1 and $\phi(n)$. Since $d = \phi(n)$, we are done. Note that only additions and subtractions are needed for the reduction.

In [11] only pairs (p, n) are considered for which n is prime and for which p generates $\mathbf{Z}_n^*$, because they lead to so-called optimal normal bases. The relevance of such bases for characteristics > 2 is limited, and the 'cheap' reduction they achieve (just $2d-1$ additions in $\mathbf{F}_p$) is almost met by the somewhat wider class considered above.

2.4 Key Generation

Given n and $d = \phi(n)$ and a desired level of security, key generation consists of two phases: sufficiently large primes p and q have to be found with p generating $\mathbf{Z}_n^*$ and q dividing $\Phi_d(p)$, after which a generator of G_q has to be found.

2.41 Finding p and q. For small d, as in this paper, standard security requirements lead to $\log p > \log q$, cf. Section 2.2. In this case the obvious generalization of the method from [13] can be used. First, an appropriately sized prime q is selected, where $q|\Phi_d(p)$ may impose a priori restrictions on q (e.g., $q \equiv 1 \bmod 3$ for $d = 6$). Next, a root r of $\Phi_d[x] \in \mathbf{F}_q[x]$ is found and p is determined as $r + \ell q$ for $\ell \in \mathbf{Z}_{\geq 0}$ such that p is a large enough prime that generates $\mathbf{Z}_n^*$.

With larger d (or $e > 1$, cf. Theorem 2.31) one may aim for primes p that fit in a computer word (i.e., $\log_2(p) = 32 \text{ or } 64$). Although this may be advantageous, $\log p$ becomes substantially smaller than $\log q$. We are not aware of an efficient method to find such p and q. If q is selected first, the probability is negligible that an appropriate p exists such that $q|\Phi_{de}(p)$. If p is selected first, there is only a very slim probability that $\Phi_{de}(p)$ has an appropriate prime factor, and finding it leads to an unattractive integer factorization problem. In this paper the possibility $\log p < \log q$ is not further discussed.

2.42 Finding a generator of G_q. This problem is easily solved by selecting $h \in \mathbf{F}_{p^d}$ at random until $g = h^{(p^d-1)/q} \neq 1$, at which point g is the desired generator. A faster method is described in [12, credited to H.W. Lenstra, Jr.]. First an element $h \in \mathsf{G}_{\Phi_d(p)}$ is constructed directly and next $g = h^{\Phi_d(p)/q}$ is computed. If $g = 1$ another h has to be generated. The specifics follow.

Let $f \in \mathbf{F}_p$ and let γ be a primitive n-th root of unity as in Section 2.3. Consider $h_f = (\gamma+f)^{(p^d-1)/\Phi_d(p)} \in \mathsf{G}_{\Phi_d(p)}$. Since $\Phi_d(p)$ divides p^d-1 irrespective of p, we can write $(p^d-1)/\Phi_d(p)$ as $r_+(p) - r_-(p)$ where r_+ and r_- are both polynomials with positive coefficients. The equation $(\gamma+f)^{r_+(p)} = h_f(\gamma+f)^{r_-(p)}$ gives rise to a system of d equations in the coefficients of h_f. Since the system only depends on p's congruency class modulo n (and not on p itself), solving the system can be done before actually picking p. The resulting h_f corresponding to several different choices for f can be hardcoded in the program. In Section 4.4 the details for G_{p^2-p+1} with $p \equiv 2 \mod 9$ are presented.

2.5 LUC and XTR

For completeness, we give a very brief description of LUC and XTR. LUC [20] is based on the subgroup $\mathsf{G}_{p+1} \subseteq \mathbf{F}_{p^2}^*$ and the trace map $\mathrm{Tr} : \mathbf{F}_{p^2} \to \mathbf{F}_p$ defined by $\mathrm{Tr}(g) = g + g^p$. Since $g \in \mathsf{G}_{p+1}$ implies that $(X - g)(X - g^p) = X^2 - (g + g^p)X + g^p g = X^2 - \mathrm{Tr}(g)X + 1$, the roots of the polynomial $X^2 - \mathrm{Tr}(g)X + 1$ are g and its conjugate g^p. Define $V_n = \mathrm{Tr}(g^n)$, then it can easily be verified that $V_{n+m} = V_n V_m - V_{n-m}$ using $g^p = g^{-1}$ for $g \in \mathsf{G}_{p+1}$. Thus, computation of V_{n+m} from V_n, V_m, and V_{n-m} costs a multiplication (a squaring if $n = m$) in $\mathbf{F}_p$. The V_n coincide with a special instance of the Lucas-function.

XTR [13] is based on the subgroup $\mathsf{G}_{p^2-p+1} \subseteq \mathbf{F}_{p^6}^*$ and the trace map $\mathrm{Tr} : \mathbf{F}_{p^6} \to \mathbf{F}_{p^2}$ defined by $\mathrm{Tr}(g) = g + g^{p^2} + g^{p^4}$. In this case, $g \in \mathsf{G}_{p^2-p+1}$ and its conjugates g^{p^2} and g^{p^4} are the roots of the polynomial $X^3 - \mathrm{Tr}(g)X^2 + \mathrm{Tr}(g)^p X - 1$. Define $c_n = \mathrm{Tr}(g^n)$, then it can be verified that $c_{n+m} = c_n c_m - c_m^p c_{n-m} + c_{n-2m}$. Since the c_n are elements of $\mathbf{F}_{p^2}$, efficient computation of c_{n+m} requires a suitable representation for $\mathbf{F}_{p^2}$ (in particular one that supports cheap Frobenius).

Both LUC and XTR compute $\mathrm{Tr}(g^n)$ instead of g^n and in case of a double exponentiation this would be $\mathrm{Tr}(g^n h^m)$ instead of $g^n h^m$. The necessity of knowing V_{n-m} respectively c_{n-m} and c_{n-2m} makes ordinary exponentiation routines unapplicable. Nevertheless, in either case efficient exponentiation methods exist. However, the shortest addition chain is typically considerably longer than the shortest one. For further details, see [22] and the references contained therein.

3 Quadratic Extensions

In this section we discuss computing in $\mathbf{F}_{p^2}$ and $\mathsf{G}_{p+1} \subset \mathbf{F}_{p^2}^*$. Fast computations in the full field $\mathbf{F}_{p^2}$ with $p \equiv 2 \bmod 3$ are important for XTR and have been discussed in [13]. We show that the field arithmetic for $p \equiv 3 \bmod 4$ from [5, Case $p^k = 4$] can be used for XTR without significant loss of efficiency compared to $p \equiv 2 \bmod 3$. The subgroup G_{p+1} is not relevant for XTR, but it is the subgroup on which LUC is based. We show that it yields some extra computational benefits that are, however, still not competitive with LUC.

We first discuss the field arithmetic for $p \equiv 2 \bmod 3$ in general and then focus on the subgroup. The case $p \equiv 3 \bmod 4$ is dealt with similarly, first the field arithmetic and then the subgroup arithmetic. Suitable exponentiation routines that apply to either case conclude this section.

3.1 Field Representation for $p \equiv 2 \bmod 3$

3.11 Field arithmetic. Let p and q be primes with $p \equiv 2 \bmod 3$ and $q | p + 1$. Then p generates $\mathbf{Z}_3^*$ and $\Phi_3(x) = x^2 + x + 1 | x^3 - 1$ is irreducible in $\mathbf{F}_p$. Let γ denote a root of $\Phi_3(x)$, then $\gamma^n = \gamma^{(n \bmod 3)}$ and in particular $\gamma^p = \gamma^2$. Hence $\Gamma = (\gamma, \gamma^2)$ is an optimal normal basis of $\mathbf{F}_{p^2}$ over $\mathbf{F}_p$. Using Γ instead of $(1, \gamma)$ leads to slightly fewer additions than the basis $(1, \gamma)$ discussed in [5, Case

$p = 3$]. The following lemma is easily implied by the formulas from [13, Section 2.1] (cf. [13, Lemma 2.1.1], [22, Lemma 2.2], and [5, Case $p = 3$]).

Lemma 3.12 *Let $a, b, c \in \mathbf{F}_{p^2}$ with $p \equiv 2 \bmod 3$.*

i. Computing a^p is free.
ii. Computing a^2 costs $2M + 2D + 3A_1$.
iii. Computing ab costs $3M + 2D + 2A_1 + 2A_2$.
iv. Computing $ac - bc^p$ costs $4M + 2D + 6A_1 + 2A_2$.

3.13 Subgroup arithmetic. Because $x^{p+1} = 1$ for $x \in \mathsf{G}_{p+1}$, we find that inversion in G_{p+1} is equivalent to p-th powering and thus for free. Let $a = a_0\gamma + a_1\gamma^2$ with $a_0, a_1 \in \mathbf{F}_p$, so $a \in \mathbf{F}_{p^2}$. Then $a \in \mathsf{G}_{p+1}$ if and only if $a^{p+1} = a^p \cdot a = 1$, i.e., $(a_1\gamma + a_0\gamma^2)(a_0\gamma + a_1\gamma^2) = 1$. This is equivalent to $a_0^2 - a_0a_1 + a_1^2 = 1$, so that G_{p+1}-membership testing costs $M + S + D + A_1 + A_2$ plus a comparison with one. This relation can also be exploited to speed up squaring in G_{p+1}, since the value of a_0a_1 follows from a_0^2 and a_1^2 using only a handful of additions. More specifically, $a^2 = (2-2a_0^2-a_1^2)\gamma+(2-a_0^2-2a_1^2)\gamma^2$, which costs $2S+2D+2A_1+3A_2$.

Free inversion in G_{p+1} also results in an advantage for simultaneous computation of ab and ab^{-1} for $a \in \mathbf{F}_{p^2}$ and $b \in \mathsf{G}_{p+1}$: since there are only four possible combinations a_ib_j, four multiplications suffice.

Lemma 3.14 *Let G_{p+1} be the order $p + 1$ subgroup of $\mathbf{F}_{p^2}^*$ with $p \equiv 2 \bmod 3$ and let $a = a_0\gamma + a_1\gamma^2 \in \mathbf{F}_{p^2}$ with $\Phi_3(\gamma) = 0$.*

i. The element a is in $\mathbf{F}_p$ if and only if $a_0 = a_1$.
ii. The element a is in G_{p+1} if and only if $a_0^2 - a_0a_1 + a_1^2 = 1$. Testing this costs $M + S + D + A_1 + A_2$.
iii. Computing a^{-1} for $a \in \mathsf{G}_{p+1}$ is free.
iv. Computing a^2 for $a \in \mathsf{G}_{p+1}$ costs $2S + 2D + 2A_1 + 3A_2$.
v. Computing ab and ab^{-1} for $b \in \mathsf{G}_{p+1}$ costs $4M + 4D + 6\min(A_1, A_2)$.

3.2 Field Representation for $p \equiv 3 \bmod 4$

3.21 Field arithmetic. Let p and q be primes with $p \equiv 3 \bmod 4$ and $q|p + 1$. Then p generates $\mathbf{Z}_4^*$ and $\Phi_4(x) = x^2 + 1$ is irreducible in $\mathbf{F}_p$. Let γ denote a root of $\Phi_4(x)$, then $\Gamma = (1, \gamma)$ is a basis of $\mathbf{F}_{p^2}$ over $\mathbf{F}_p$. (Since $\gamma^2 = -1$ the basis (γ, γ^2) looks contrived and leads to slightly more complicated reductions.) This field representation is identical to [5, Case $p^k = 4$], although the number of additions in our cost functions is slightly different.

Let $a \in \mathbf{F}_{p^2}$ be represented by $(a_0, a_1) \in (\mathbf{F}_p)^2$, i.e., $a = \Gamma \cdot (a_0, a_1)^T = a_0 + a_1\gamma$. From $\gamma^n = \gamma^{(n \bmod 4)}$ and thus $\gamma^p = \gamma^3 = -\gamma$ it follows that $a^p = a_0^p + a_1^p\gamma^p = a_0 - a_1\gamma$ so that p-th powering costs a modular negation. The cost of multiplication is $3M + 2D + 2A_1 + 3A_2$ since $ab = a_0b_0 - a_1b_1 + ((a_0 + a_1)(b_0 + b_1) - a_0b_0 - a_1b_1)\gamma$. The cost of squaring is $2M + 2D + 2A_1$ since $a^2 = (a_0 + a_1)(a_0 - a_1) + 2a_0a_1\gamma$. The cost of computing $ac - bc^p$ for $a, b, c \in \mathbf{F}_{p^2}$ is $4M + 2D + 2A_1 + 2A_2$ since $ac - bc^p = (b_0 + a_0)c_0 + (b_1 - a_1)c_1 + ((a_1 - b_1)c_0 + (a_0 + b_0)c_1)\gamma$. By analogy with Lemma 3.12 we get the following.

Lemma 3.22 *Let $a, b, c \in \mathbf{F}_{p^2}$ with $p \equiv 3 \bmod 4$.*

i. Computing a^p costs A_1.
ii. Computing a^2 costs $2M + 2D + 2A_1$.
iii. Computing ab costs $3M + 2D + 2A_1 + 3A_2$.
iv. Computing $ac - bc^p$ costs $4M + 2D + 2A_1 + 2A_2$.

It follows from Lemmas 3.12 and 3.22 and [13] that XTR can be generalized to $p \equiv 3 \bmod 4$ without loss of efficiency compared to $p \equiv 2 \bmod 3$ as in [13].

3.23 Subgroup arithmetic. As in 3.13, inversion in G_{p+1} is equivalent to p-th powering; it costs A_1. Let $a = a_0 + a_1\gamma$ with $a_0, a_1 \in \mathbf{F}_p$, so $a \in \mathbf{F}_{p^2}$. Then $a \in \mathsf{G}_{p+1}$ if and only if $a^{p+1} = a^p \cdot a = 1$, i.e., $(a_0 - a_1\gamma)(a_0 + a_1\gamma) = 1$ which is equivalent to $a_0^2 + a_1^2 = 1$. So, G_{p+1}-membership testing costs $2S + D + A_2$. It also follows that $a^2 = 2a_0^2 - 1 + ((a_0 + a_1)^2 - 1)\gamma$ for $a \in \mathsf{G}_{p+1}$, which implies that squaring in G_{p+1} can be done faster than in $\mathbf{F}_{p^2}$.

Lemma 3.24 *Let G_{p+1} be the order $p+1$ subgroup of $\mathbf{F}_{p^2}^*$ with $p \equiv 3 \bmod 4$ and let $a = a_0 + a_1\gamma \in \mathbf{F}_{p^2}$ with $\Phi_4(\gamma) = 0$.*

i. The element a is in $\mathbf{F}_p$ if and only if $a_1 = 0$.
ii. The element a is in G_{p+1} if and only if $a_0^2 + a_1^2 = 1$. Testing this costs $2S + D + A_2$.
iii. If $a \in \mathsf{G}_{p+1}$, then computing a^{-1} costs A_1.
iv. If $a \in \mathsf{G}_{p+1}$, then computing a^2 costs $2S + 2D + A_1$.
v. Computing ab and ab^{-1} for $b \in \mathsf{G}_{p+1}$ costs $4M + 4D + 6\min(A_1, A_2)$.

3.3 Subgroup Exponentiation

For a single exponentiation we have to compute a^m, where m has roughly the same bitlength k as q. With signed flexible windows [4] of size 5, this requires about $k+1$ squarings and $7 + k/6$ multiplications in G_q. With Lemmas 3.14.*iv* and 3.24.*iv* the squaring cost is $\approx (3S + 2D)(k + 1)$ and with Lemmas 3.12.*iii* and 3.22.*iii* the multiplication cost is $\approx (3M + 2D)(7 + k/6)$. Under the assumption that $M \approx D$ and $S \approx 0.3M$ the resulting number of $\mathbf{F}_p$-multiplications is 19.1 for the precomputation plus 2.0 per exponent bit.

For double exponentiation we have to compute $a^m b^n$ for m and n of roughly equal size and with m as above. This can be computed using Solinas' trick [21, See also Appendix A], resulting in k squarings and $k/2$ multiplications in G_q. With G_q-arithmetic as above, this becomes $(2S+2D)k + (3M+2D)k/2 \approx 2.85k$ multiplications in $\mathbf{F}_p$. The precomputation of ab and ab^{-1} uses Lemmas 3.14.*v* and 3.24.*v*. Combination of these observations leads to the following theorem.

Theorem 3.31 *Let p and q be primes with $q|p+1$, $p \equiv 2 \bmod 3$ or $p \equiv 3 \bmod 4$, and $\lceil \log_2 q \rceil = k$. Let a, b be in the order q subgroup G_q of $\mathbf{F}_{p^2}^*$ and $m, n \in (0, q)$. Assuming that $M \approx D$ and $S \approx 0.3M$,*

i. computing a^m costs on average $19.1 + 2k$ multiplications in $\mathbf{F}_p$, and
ii. computing $a^m b^n$ costs on average $4 + 2.85k$ multiplications in $\mathbf{F}_p$.

These results improve previously reported ones, but the resulting exponentiations are less efficient than the LUC exponentiations. So, even though we have several related results concerning improved key selection and other choices of p, we leave the subject of quadratic extensions and move on to sixth degree extensions because there our methods appear to have a more substantial impact.

4 Sixth Degree Extension

In this section fast exponentiation routines for the group $\mathsf{G}_{p^2-p+1} \subset \mathbf{F}_{p^6}^*$ with $p \equiv 2 \bmod 9$ are described. Let f be a sixth degree irreducible polynomial over some ground field, with root γ. Consider the extension induced by γ and represented by a polynomial basis consisting of six consecutive powers of γ, such as $(1, \gamma, \ldots, \gamma^5)$ or $(\gamma, \gamma^2, \ldots, \gamma^6)$. The cost of computation in this representation depends on the general question of how many ground field multiplications are needed to multiply two degree five polynomials, and on the specific question of what f looks like. Therefore, a short word on the multiplication of fifth degree polynomials in general, before going into details about the field representation and the benefits the group offers. These results are then used in the subsequent exponentiation routines. We conclude this section with an improved key selection method.

4.1 Multiplication of Fifth Degree Polynomials

Multiplication of two polynomials of degree five can be done in 18 multiplications plus a handful of additions [2,5]. Indeed, let $G(x) = \sum_{i=0}^{5} g_i x^i$ and $H(x) = \sum_{i=0}^{5} h_i x^i$ be two fifth degree polynomials. Write $G = G_0 + G_1 x^3$ and $H = H_0 + H_1 x^3$ where G_0, G_1, H_0, and H_1 are second degree polynomials. Then

$$GH = G_0 H_0 + (G_0 H_1 + G_1 H_0) x^3 + G_1 H_1 x^6,$$

so that, with $C_0 = G_0 H_0$, $C_1 = G_1 H_1$, and $C_2 = (G_0 - G_1)(H_0 - H_1)$, it follows that

$$GH = C_0 + (C_0 + C_1 - C_2) x^3 + C_1 x^6. \tag{1}$$

Each of the C_i can be computed using 6 multiplications in the ground field. For example, because $G_0 = g_0 + g_1 x + g_2 x^2$ and $H_0 = h_0 + h_1 x + h_2 x^2$,

$$C_0 = g_0 h_0 + (g_1 h_0 + g_0 h_1) x + (g_2 h_0 + g_1 h_1 + g_0 h_2) x^2 + (g_2 h_1 + g_1 h_2) x^3 + (g_2 h_2) x^4,$$

so that, with $c_0 = g_0 h_0$, $c_1 = g_1 h_1$, $c_2 = g_2 h_2$, $c_3 = (g_0 - g_1)(h_0 - h_1)$, $c_4 = (g_0 - g_2)(h_0 - h_2)$, and $c_5 = (g_1 - g_2)(h_1 - h_2)$, we have that

$$C_0 = c_0 + (c_0 + c_1 - c_3) x + (c_0 + c_1 + c_2 - c_4) x^2 + (c_1 + c_2 - c_5) x^3 + c_2 x^4.$$

With similar expressions for C_1 and C_2 it follows that 18 ground field multiplications (or squarings) suffice to compute the product GH (or the square G^2).

If the g_i and h_i are l-bit numbers, and one is interested in an (unreduced) product with $2l$-bit or sligthly larger coefficients, then computing C_0 costs $6M + 6A_1 + 7A_2$ and the cost of computing GH as in (1) is $18M + 24A_1 + 21A_2$.

It remains to reduce GH modulo f, at a cost depending on f. This is discussed in the remainder of this section for several ground fields $\mathbf{F}_p$. In that case the resulting coefficients must be reduced modulo p at a cost of $6D$ for l-bit p.

4.2 Field Representation for $p \equiv 2 \bmod 9$

4.21 Field arithmetic. Let p be prime with $p \equiv 2 \bmod 9$. Then p generates $\mathbf{Z}_9^*$ and $\Phi_9(x) = x^6 + x^3 + 1$ is irreducible in $\mathbf{F}_p$. Let γ denote a root of $\Phi_9(x)$, then $\Gamma = (\gamma, \gamma^2, \ldots, \gamma^6)$ is a basis for $\mathbf{F}_{p^6}$ over $\mathbf{F}_p$ (in [5, Case $p^k = 9$] the similar basis $(1, \gamma, \ldots, \gamma^5)$ is used).

Let $a = \sum_{i=0}^{5} a_i \gamma^{i+1} \in \mathbf{F}_{p^6}$. From $\gamma^n = \gamma^{n \bmod 9}$ and thus $\gamma^p = \gamma^2$ it follows with $\Phi_9(\gamma) = 0$ that $a^p = a_4\gamma + (a_0 - a_3)\gamma^2 + a_5\gamma^3 + a_1\gamma^4 - a_3\gamma^5 + a_2\gamma^6$. Thus, p-th powering costs A_1. In a similar way it follows that p^3-th powering costs $2A_1$. For multiplication in $\mathbf{F}_{p^6}$ the method from Section 4.1 is used, with proper adjustment of the powers of x, e.g., $G = G_0 x + G_1 x^4$. It follows with straightforward bookkeeping that collecting corresponding powers of x in Relation (1) combined with the modular reductions costs $12A_2 + 6D$. (For the basis $(1, \gamma, \ldots, \gamma^5)$ we find that the collecting phase costs $14A_2$, which slightly improves the $18A_2$ reported in [5].) With Section 4.1 it follows that multiplication can be done for $18M + 6D + 24A_1 + 33A_2$. Doing more elaborate collecting reduces the $33A_2$ to $29A_2$. Squaring follows by replacing $18M$ by $18S$, but it can be done substantially faster by observing that

$$G^2 = (G_0\gamma + G_1\gamma^4)^2 = (G_0 - G_1)(G_0 + G_1)\gamma^2 + (2G_0 - G_1)G_1\gamma^5,$$

with $G_0, G_1 \in \mathbf{F}_p[\gamma]$ of degree two. Computing this costs $9A_1$ for the preparation of the multiplicands, two polynomial multiplications costing $6M + 6A_1 + 7A_2$ each, $7A_2$ for the collection, and $6D$ for the final reductions. It follows that squaring can be done for $12M + 6D + 21A_1 + 21A_2$. (This is A_2 more than reported in [5] for $(1, \gamma, \ldots, \gamma^5)$.)

Lemma 4.22 *Let $a, b \in \mathbf{F}_{p^6}$ with $p \equiv 2 \bmod 9$.*

i. Computing a^p or a^{p^5} costs A_1.
ii. Computing a^{p^2}, a^{p^3}, or a^{p^4} costs $2A_1$.
iii. Computing a^2 costs $12M + 6D + 21A_1 + 21A_2$.
iv. Computing ab costs $18M + 6D + 24A_1 + 29A_2$.

4.23 Subgroup arithmetic. Let $a = \sum_{i=0}^{5} a_i \gamma^{i+1} \in \mathbf{F}_{p^6}$. Membership of one of the three proper subfields of $\mathbf{F}_{p^6}$ is characterized by one of the equations $a^{p^i} = a$ for $i = 1, 2, 3$. Specifically, $a \in \mathbf{F}_p$ if and only if $a^p = a$ which is equivalent to the system of linear equations $(a_0, a_1, a_2, a_3, a_4, a_5) = (a_4, a_0 - a_3, a_5, a_1, -a_3, a_2)$. The solution $a_0 = a_1 = a_3 = a_4 = 0$ and $a_2 = a_5$ is not surprising since

$1+\gamma^3+\gamma^6=0$, so an element $c \in \mathbf{F}_p$ takes the form $-c\gamma^3 - c\gamma^6$. Similarly, $a \in \mathbf{F}_{p^2}$ if and only if $a^{p^2}=a$, which is equivalent to $a=a_2\gamma^3+a_5\gamma^6$, and $a \in \mathbf{F}_{p^3}$ if and only if $a^{p^3}=a$ or $a=(a_3-a_4)\gamma+(-a_3+a_4)\gamma^2+a_5\gamma^3+a_3\gamma^4+a_4\gamma^5+a_5\gamma^6$.

More interesting for cryptographic purposes is the order p^2-p+1 subgroup G_{p^2-p+1} of $\mathbf{F}_{p^6}^*$, because that subgroup cannot be embedded in a true subfield of $\mathbf{F}_{p^6}$. The G_{p^2-p+1}-membership condition $a^{p^2-p+1}=1$ is equivalent to $a^{p^2}a=a^p$, which can be verified at a cost of, essentially, a single $\mathbf{F}_{p^6}$-multiplication. From $a^{p^3}=a^{-1}$ it follows that inversion in G_{p^2-p+1} costs $2A_1$.

Computing $a^{p^2}a-a^p=\sum_{i=0}^{5} v_i\gamma^{i+1}$ symbolically produces

$$\begin{aligned}
v_0 &= a_1^2 - a_0a_2 - a_4 - a_4^2 + a_3a_5,\\
v_1 &= -a_0 + a_1a_2 + a_3 - 2a_0a_3 + a_3^2 - a_2a_4 - a_1a_5,\\
v_2 &= -a_0a_1 + a_3a_4 - a_5 - 2a_2a_5 + a_5^2,\\
v_3 &= -a_1 - a_2a_3 + 2a_1a_4 - a_4^2 - a_0a_5 + a_3a_5,\\
v_4 &= a_0^2 + a_1a_2 + a_3 - 2a_0a_3 - a_4a_5,\\
v_5 &= -a_2 + a_2^2 - a_1a_3 - a_0a_4 + a_3a_4 - 2a_2a_5.
\end{aligned} \tag{2}$$

If $a \in \mathsf{G}_{p^2-p+1}$, then $v_i=0$ for $0 \le i < 6$ and the resulting six relations can be used to significantly reduce the cost of squaring in G_{p^2-p+1}. Let $V=(v_0,v_1,\ldots,v_5)$ be the vector consisting of the v_i's. Then for any 6×6-matrix M, we have that $a^2+\Gamma\cdot(M\cdot V^T)=a^2$ if $a \in \mathsf{G}_{p^2-p+1}$, because in that case V is the all-zero vector. Carrying out this computation symbolically, involving the expressions for the v_i's for a particular choice of M yields the following:

$$a^2 = a^2 + 2\Gamma\cdot\begin{pmatrix} 0&0&0&-1&0&0\\ 0&-1&0&0&1&0\\ 0&0&1&0&0&0\\ 1&0&0&-1&0&0\\ 0&-1&0&0&0&0\\ 0&0&0&0&0&1 \end{pmatrix}\cdot V^T = \Gamma\cdot\begin{pmatrix} 2a_1+3a_4(a_4-2a_1)\\ 2a_0+3(a_0+a_3)(a_0-a_3)\\ -2a_5+3a_5(a_5-2a_2)\\ 2(a_1-a_4)+3a_1(a_1-2a_4)\\ 2(a_0-a_3)+3a_3(2a_0-a_3)\\ -2a_2+3a_2(a_2-2a_5) \end{pmatrix}. \tag{3}$$

Given that we are working over a sixth degree extension, the six multiplications and reductions required for (3) seem optimal. The additions can be taken care of in several ways; a reasonable solution results in $6M+6D+9A_1+12A_2$.

Lemma 4.24 *Let G_{p^2-p+1} be the order p^2-p+1 subgroup of $\mathbf{F}_{p^6}^*$ with $p \equiv 2 \bmod 9$ and let $a=a_0\gamma+a_1\gamma^2+\cdots+a_5\gamma^6 \in \mathbf{F}_{p^6}$ with $\Phi_9(\gamma)=0$.*

i. The element a is in $\mathbf{F}_p$ if and only if $a=a_2\gamma^3+a_2\gamma^6$.
ii. The element a is in $\mathbf{F}_{p^2}$ if and only if $a=a_2\gamma^3+a_5\gamma^6$.
iii. The element a is in $\mathbf{F}_{p^3}$ if and only if $a=(a_3-a_4)\gamma+(-a_3+a_4)\gamma^2+a_5\gamma^3+a_3\gamma^4+a_4\gamma^5+a_5\gamma^6$.
iv. The element a is in G_{p^2-p+1} if and only if in relations (2) $v_i=0$ for $0 \le i < 6$. This can be checked at a cost of essentially $18M+6D$.
v. Computing a^{-1} for $a \in \mathsf{G}_{p^2-p+1}$ costs $2A_1$.
vi. Computing a^2 for $a \in \mathsf{G}_{p^2-p+1}$ costs $6M+6D+9A_1+12A_2$.

4.3 Subgroup Exponentiation

For a single exponentiation we have to compute a^m, where m has roughly the same bitlength k as q. For the case $q|p^2-p+1$ it is shown in [22, Section 4.4] that m can quickly be written as $m \equiv m_1 + m_2 p \bmod q$ with m_1 and m_2 of bitlength $k/2$. Hence a^m can be rewritten as $a^{m_1}(a^p)^{m_2}$. This can be computed using Solinas' trick [21] at the cost of $k/2$ squarings and $k/4$ multiplications in G_q. Tanja Lange pointed out to us that the precomputation only requires one group multiplication, since $a^p a^{-1} = a^{p^2}$. With Lemmas 4.24.*vi* and 4.22.*iv* the squaring and multiplication costs become $\approx (6M + 6D)k/2$ and $\approx (18M + 6D)k/4$, respectively. Assuming that $M \approx D$ this results in six $\mathbf{F}_p$-multiplications per exponent bit.

A double exponentiation $a^m b^n$, with $\log m \approx \log n$ and m as above, can be rewritten as $a^{m_1}(a^p)^{m_2} b^{n_1}(b^p)^{n_2}$ with $\approx k/2$-bit m_1, m_2, n_1, and n_2. This quadruple exponentiation can be computed using Solinas' trick simultaneously on two pairs of two exponents (paired in any way), resulting in a total of $k/2$ squarings and twice $k/4$ multiplications. With Lemmas 4.24.*vi* and 4.22.*iv* this becomes $(6M + 6D)k/2 + 2(18M + 6D)k/4 \approx 9k$ multiplications in $\mathbf{F}_p$. Combination of these obeservations leads to the following theorem.

Theorem 4.31 *Let p and q be primes with $q|p^2 - p + 1$, $p \equiv 2 \bmod 9$, and $\lceil \log_2 q \rceil = k$. Let a, b be in the order q subgroup G_q of $\mathbf{F}_{p^6}^*$ and $m, n \in (0, q)$. Assuming that $M \approx D$,*

i. *computing a^m costs on average $6 + 6k$ multiplications in $\mathbf{F}_p$, and*
ii. *computing $a^m b^n$ costs on average $12 + 9k$ multiplications in $\mathbf{F}_p$.*

The cost of this $\mathbf{F}_{p^6}$-exponentiation is comparable to XTR, cf. Section 5.

4.4 Key Selection

We elaborate on the improved key selection mentioned in Section 2.42 and similar to [12, Algorithm 4.5]. With $f \in \mathbf{F}_p$ and $h_f = (\gamma + f)^{(p^6-1)/\Phi_6(p)}$ it follows that $h_f(\gamma+f)(\gamma+f)^p = (\gamma+f)^{p^3}(\gamma+f)^{p^4}$. Solving this equation for the coefficients of h_f gives

$$h_f = \frac{\Gamma}{f^6 - f^3 + 1} \cdot \begin{pmatrix} -f + f^2 + 3f^3 - f^4 - 2f^5 \\ -f - 2f^2 + 3f^3 + 2f^4 - 2f^5 \\ (1 - f^2)^3 \\ f - f^2 + f^4 - f^5 \\ f - f^2 + f^4 - f^5 \\ -f^3(1 - 3f + f^3) \end{pmatrix}. \tag{4}$$

This gives $h_{1/2} = \frac{1}{19}(0, -12, 9, 6, 6, 1)$ and $h_2 = -\frac{3}{19}(6, 2, 3, 2, 2, 8/3)$.

Given either $h \in \mathsf{G}_{\Phi_d(p)}$, compute $g = h^{(p^2-p+1)/q}$ using [4] and Lemmas 4.24.*vi* and 4.22.*iv*. Assuming that p is only slightly larger in size than q this takes $8 \log p + 90$ ground field multiplications (note that Theorem 4.31 does not apply). The resulting g generates G_q unless $g = 1$. The probability of failure may be expected to be q^{-1}, independently for each h. This is negligible.

Remark 4.41 Our methods work, and result in identical runtimes, as long as $p \bmod 9$ generates $\mathbf{Z}_9^*$. Since $\phi(\phi(9)) = 2$, the only other case is $p \equiv 5 \bmod 9$. Several other choices of p can be handled in a similar fashion.

5 Timings

All methods were implemented to verify their correctness and runtime characteristics. The table below summarizes runtimes for $\mathsf{G}_q \subset \mathsf{G}_{p^2-p+1} \subset \mathbf{F}_{p^6}^*$ for 170-bit p and q, and compares them to the XTR timings from [13,22]. They are in milliseconds on a 600 MHz Pentium III NT laptop, averaged over 100 random p, q pairs and 100 exponentiations per pair. The timings confirm that our new methods for $\mathbf{F}_{p^6}$-subgroup exponentiation are superior to the original XTR and almost competitive with the faster version of XTR from [22]. This shows that the main reason to use XTR would no longer be its speed, but mostly its compact — and sometimes inconvenient — representation.

Table 1. XTR and $\mathsf{G}_q \subset \mathsf{G}_{p^2-p+1}$ runtimes.

	XTR in [13]	XTR in [22]	G_q
key generation	64 ms	62 ms	85 ms
single exponentiation	10 ms	7.4 ms	8.9 ms
double exponentiation	21 ms	8.6 ms	13 ms

Acknowledgements. We would like to thank Jeroen Doumen, Jan Draisma and Tanja Lange for fruitful discussions, and the CHES 2002 reviewers for their useful comments. Special thanks go to Peter Beelen, whose remarks stimulated this research and improved the group arithmetic for quadratic extensions.

References

1. G. Agnew, R. Mullin, and S. Vanstone. Fast exponentiation in $GF(2^n)$. In C. G. Günther, editor, *Advances in Cryptography—Eurocrypt'88*, volume 330 of *Lecture Notes in Computer Science*, pages 251–255. Springer-Verlag, 1988.
2. D. Bailey and C. Paar. Efficient arithmetic in finite field extensions with application in elliptic curve cryptography. *Journal of Cryptology*, 2000.
3. S. A. Brands. *Rethinking Public Key Infrastructures and Digital Certificates Building in Privacy.* PhD thesis, Technische Universiteit Eindhoven, 1999.
4. H. Cohen. Analysis of the flexible window powering algorithm. Submitted for publication, available from `http://www.math.u-bordeaux.fr/~cohen`, 2001.
5. H. Cohen and A. K. Lenstra. Supplement to implementation of a new primality test. *Mathematics of Computation*, 48(177): S1–S4, 1987.

6. R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In H. Krawczyk, editor, *Advances in Cryptography—Crypto'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 13–25. Springer-Verlag, 1998.
7. T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.
8. R. Gallant, R. Lambert, and S. Vanstone. Faster point multiplication on elliptic curves with efficient endomorphisms. In J. Kilian, editor, *Advances in Cryptography—Crypto'01*, volume 2139 of *Lecture Notes in Computer Science*, pages 190–200. Springer-Verlag, 2001.
9. D. M. Gordon. Discrete logarithms in GF(p) using the number field sieve. *SIAM J. Discrete Math.*, 6(1):124–138, 1993.
10. A. Lempel, G. Seroussi, and S. Winograd. On the complexity of multiplication in finite fields. *Theoretical Computer Science*, 22:285–296, 1983.
11. A. K. Lenstra. Using cyclotomic polynomials to construct efficient discrete logarithm cryptosystems over finite fields. In V. Varadharajan, J. Pieprzyk, and Y. Mu, editors, *ACISP'97*, volume 1270 of *Lecture Notes in Computer Science*, pages 127–138. Springer-Verlag, 1997.
12. A. K. Lenstra and E. R. Verheul. Key improvements to XTR. In T. Okamoto, editor, *Advances in Cryptography—Asiacrypt'00*, volume 1976 of *Lecture Notes in Computer Science*, pages 220–233. Springer-Verlag, 2000.
13. A. K. Lenstra and E. R. Verheul. The XTR public key system. In M. Bellare, editor, *Advances in Cryptography—Crypto'00*, volume 1880 of *Lecture Notes in Computer Science*, pages 1–19. Springer-Verlag, 2000.
14. R. Lidl and H. Niederreiter. *Introduction to finite fields and their applications.* Cambridge University Press, 1994.
15. S. C. Pohlig and M. E. Hellman. An improved algorithm for computing logarithms over $gf(p)$ and its cryptographic significance. *IEEE Transactions on Information Theory*, 24:106–110, 1978.
16. J. Pollard. Monte carlo methods for index computation (mod p). *Mathematics of Computation*, 32(143):918–924, 1978.
17. O. Schirokauer, Mar. 2000. Personal communication.
18. O. Schirokauer, D. Weber, and T. F. Denny. Discrete logarithms: the effectiveness of the index calculus method. In H. Cohen, editor, *ANTS II*, volume 1122 of *Lecture Notes in Computer Science*, pages 337–361. Springer-Verlag, 1996.
19. B. Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic voting. In M. Wiener, editor, *Advances in Cryptography—Crypto'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 148–164. Springer-Verlag, 1999.
20. P. Smith and C. Skinner. A public-key cryptosystem and a digital signature system based on the Lucas function analogue to discrete logarithms. In J. Pieprzyk and R. Safavi-Naini, editors, *Advances in Cryptography—Asiacrypt'94*, volume 917 of *Lecture Notes in Computer Science*, pages 357–364. Springer-Verlag, 1995.
21. J. A. Solinas. Low-weight binary representations for pairs of integers. Technical report, CACR (University of Waterloo) preprint series, 2001.
22. M. Stam and A. K. Lenstra. Speeding up XTR. In C. Boyd, editor, *Advances in Cryptography—Asiacrypt'01*, volume 2248 of *Lecture Notes in Computer Science*, pages 125–143. Springer-Verlag, 2001.
23. E. Teske. On random walks for Pollard's rho method. *Mathematics of Computation*, 70:809–825, 2001.

24. J. von zur Gathen and M. Nöcker. Exponentiation in finite fields: theory and practice. In T. Mora and H. Mattson, editors, *AAECC-12*, volume 1255 of *Lecture Notes in Computer Science*, pages 88–133. Springer-Verlag, 1997.

A Solinas' Trick

We briefly discuss Solinas' trick for performing a double exponentiation $g^a h^b$ in a group where inversion is cheap. This naturally occurs in the context of elliptic curve cryptography, and also applies to the groups G_q as discussed in this paper.

Let g be a group element and a some exponent. Let the binary expansion of a be $\sum_{i=0}^{k} a_i 2^i$ where all $a_i \in \{0,1\}$. On average, half of the a_i's will be nonzero, hence the square-and-multiply method requires k squarings and $k/2$ multiplications.

If inversion, i.e., the computation of g^{-1} is cheap, single exponentiation can be sped up by using a signed digit representation for the exponent. Once again, write $a = \sum_{i=0}^{k} a_i 2^i$, but relax the condition on the a_i to $a_i \in \{-1,0,1\}$. The representation is no longer unique, but the non-adjacent form (NAF) is. On average, only a third of the a_i's of the NAF will be nonzero. This improves the square-and-multiply method to k squarings and $k/3$ multiplications.

A double exponentiation, i.e., the computation of $g^a h^b$ for given group elements g and h and exponents a and b, can be performed faster than two separate exponentiations using Shamir's trick. If $a = \sum_{i=0}^{k} a_i 2^i$ and $b = \sum_{i=0}^{k} b_i 2^i$ with all $a_i, b_i \in \{0,1\}$, switching to a vector notation $\underline{c} = (ab)^T$ and $\underline{c}_i = (a_i b_i)^T$ leads to $\underline{c} = \sum_{i=0}^{k} \underline{c}_i 2^i$, where $\underline{c}_i \in \{\binom{0}{0}, \binom{0}{1}, \binom{1}{0}, \binom{1}{1}\}$. Assuming a and b to be independent, about three quarter of the columns $\underline{c}_i$ will be nonzero. By precomputing the value gh corresponding to $\underline{c}_i = \binom{1}{1}$ this yields a runtime of k squarings and $3k/4$ multiplications for the square-and-multiply method.

If inversion is for free, one could consider combining the NAF with Shamir's trick. Given two random exponents, each having a NAF of length about k and an expected number of $2k/3$ zeroes, on average in $\frac{4}{9}$ of the positions both a_i and b_i be zero. This leaves $\frac{5}{9}k$ nonzero $\underline{c}_i$'s, resulting in an improvement of the square-and-multiply method to k squarings and $5k/9$ multiplications.

In [21], Solinas noted that computing the NAF's independent of each other might not be optimal to minimize the number of nonzero $\underline{c}_i$'s. As an alternative, the joint sparse form is proposed, that satisfies the following properties:

1. There are at most two consecutive nonzero columns.
2. Adjacent terms do not have opposite sign, i.e., $a_i a_{i+1} \neq -1$ and $b_i b_{i+1} \neq -1$ for all i.
3. If $a_i a_{i+1} = 1$, then $b_i = 0$ and $b_{i+1} = \pm 1$. Similarly for $b_i b_{i+1} = 1$.

An efficient algorithm is given in [21] that computes the Joint Sparse Form and it is proven that on average, half of the resulting columns $\underline{c}_i$ will be nonzero. The running time of the square-and-multiply method thus becomes k squarings and $k/2$ multiplications.