Using Cyclotomic Polynomials to Construct Efficient Discrete Logarithm Cryptosystems over Finite Fields

Arjen K. Lenstra

Citibank, N.A., 4 Sylvan Way, Parsippany, NJ 07054, U.S.A.
Email: arjen.lenstra@citicorp.com

Abstract. We show how to use cyclotomic polynomials to construct subgroups of multiplicative groups of finite fields that allow very efficient implementation of discrete logarithm based public key cryptosystems. Depending on the type of application and implementation, the resulting schemes may be up to three times faster than the fastest known schemes of comparable security that use more conventional choices of subgroups or finite fields.

1 Introduction

In this paper we consider public key cryptosystems that are based, for their security, on the difficulty of the discrete logarithm problem in the multiplicative group of a finite field. Several such cryptosystems have been proposed. Examples are the signature and encryption algorithms due to Taher ElGamal (cf. [3]) and variations thereof. We propose a new variation that has distinct computational advantages over previously proposed schemes.

The most time consuming operation in the ElGamal schemes is exponentiation in the finite field. Therefore variants have been proposed that make the exponentiation faster without affecting the security. Claus Schnorr proposed to work in a small subgroup of the multiplicative group of a prime field of large characteristic (cf. [13]). Such subgroups not only make the signature considerably shorter, but they also make the exponentiation considerably faster by using short exponents. Supposedly, using subgroups does not affect the security of the scheme if the subgroup order is prime and sufficiently large (though small compared to the characteristic). A variant of Schnorr's proposal was adopted in the U.S. Government's Digital Signature Standard (cf. [6]). All these variants allow efficient software implementations.

Gordon Agnew et al. proposed to work in a large extension of the field of two elements (cf. [2]). With a so-called 'optimal normal basis' representation of the extension field multiplication becomes very fast and squaring reduces to a circular shift, so that exponentiation can be done very efficiently. Although this representation does not affect the security of the scheme, fields of characteristic two are generally considered to be more vulnerable to attacks than other fields of comparable sizes. The security of Agnew's variant can be boosted by working in a quadratic extension, an idea that is similar to Schnorr's. To take full advantage of this variant a hardware implementation is strongly recommended.

We show how the ideas of using subgroups and optimal normal bases can be combined by showing how cyclotomic polynomials can be used to efficiently construct cryptosystems that not only use small subgroups (and thus short signatures and short exponents) but that also allow very efficient exponentiation. We show that our variant is asymptotically faster than both Schnorr's and Agnew's variants. We also show that an actual software implementation of our variant can be expected to be about three times faster than Schnorr's variant; for this comparison both variants used '960-bit security', which is considered to be acceptable today. For signature generation the performance comparison to RSA is also very favorable, even if RSA is implemented using Chinese remaindering.

This paper is organized as follows. In Section 2 we introduce our notation and review some well known facts about finite fields, optimal normal bases, discrete logarithms, and cyclotomic polynomials. In Section 3 we present our new variant. Its performance is analyzed and compared to other variants in Section 4. We conclude with some additional observations in Section 5.

2 Preliminaries

(2.1) Finite fields. Let $F(p^t)$ denote a finite field of odd characteristic p and cardinality p^t. The multiplicative group $F(p^t)^*$ of $F(p^t)$ has cardinality p^t-1. We assume that t+1 is prime (but see (5.1)) and that p is a primitive root modulo t+1 (i.e., p modulo t+1 generates $F(t+1)^*$). Then the zeros α^i for $i=1,p,\ldots,p^{t-1}$ of the polynomial $(X^{t+1}-1)/(X-1)=X^t+X^{t-1}+\ldots+X+1$ form an optimal normal basis for $F(p^t)$ over F(p) (cf. [4]). For the definition of an optimal normal basis we refer to [9]. For our purposes we are interested in the following properties of these bases.

Choice of basis for $F(p^t)$ over F(p). Because p is a primitive root modulo t+1 and because $\alpha^i=\alpha^{i \bmod t+1}$, the basis $\{\alpha^i : i=1,p,\ldots,p^{t-1}\}$ for $F(p^t)$ over F(p) is the same as the basis $\{\alpha^i : i=1,2,\ldots,t\}$ for $F(p^t)$ over F(p). Note that the latter basis is different from but similar to the more traditional power basis $\{\alpha^i : i=0,1,\ldots,t-1\}$ for $F(p^t)$ over F(p). The identities $\alpha^t=-\alpha^{t-1}-\alpha^{t-2}-\ldots-\alpha-1$ and $\alpha^0=-\alpha^t-\alpha^{t-1}-\ldots-\alpha$ can be used to switch quickly between these two bases. We prefer to use the basis $\{\alpha^i : i=1,2,\ldots,t\}$ for $F(p^t)$ over F(p) because it makes p-th powering essentially for free (see below). Sometimes, however, it might be more convenient to use $\{\alpha^i : i=0,1,\ldots,t-1\}$ (cf. (5.3)).

Multiplication in $F(p^t)$. Using the basis $\{\alpha^i : i=1,2,\ldots,t\}$ for $F(p^t)$ over F(p), we can multiply elements of $F(p^t)$ in t^2 multiplications and $O(t^2)$ additions in F(p): exponents of α that are >t can be reduced modulo t+1 (since $\alpha^i=\alpha^{i \bmod t+1}$), and $\alpha^0=-\alpha^t-\alpha^{t-1}-\ldots-\alpha$ can be used to handle exponents of α that are zero modulo t+1. Thus, the reduction stage of the multiplication (and squaring) in $F(p^t)$ takes only 2t−1 additions in F(p). This is a consequence of the optimality of the basis. The power basis $\{\alpha^i : i=0,1,\ldots,t-1\}$ for $F(p^t)$ over F(p) works just as quickly for multiplication and squaring: for the reduction stage exponents can still be taken modulo t+1, and the identity $\alpha^t=-\alpha^{t-1}-\alpha^{t-2}-\ldots-\alpha-1$ can be used to handle exponents that are equal to t modulo t+1.

p-th Powering in $F(p^t)$. The elements of $F(p)$ are left unchanged by p-th powering, and the mapping that maps i to (p*i modulo t+1) acts as a permutation on the set of exponents $\{1,2,\ldots,t\}$ of α. Therefore, p-th powering of elements of $F(p^t)$ that are represented using the basis $\{\alpha^i : i=1,2,\ldots,t\}$ for $F(p^t)$ over $F(p)$ involves only a permutation of the elements of $F(p)$ representing $F(p^t)$ over the basis $\{\alpha^i : i=1,2,\ldots,t\}$, and is thus virtually for free in practice. This is a consequence of the fact that $\{\alpha^i : i=1,p,\ldots,p^{t-1}\}$ is a normal basis for $F(p^t)$ over $F(p)$.

Exponentiation in $F(p^t)$. To raise an element x of $F(p^t)$ to the e-th power, for some $e>p$, we first write e in the basis p, i.e., $e=e(k)p^k+e(k-1)p^{k-1}+\ldots+e(1)p+e(0)$ with $k\approx\log_2 e/\log_2 p$ and $0\le e(i)<p$ for $i=0,1,\ldots,k$. By scanning the k+1 exponents e(i) from least to most significant bit, we next compute $x^{e(i)}$ in $F(p^t)$ for $i=0,1,\ldots,k$ using $[\log_2 p]$ squarings and approximately $w(e)-k$ multiplications in $F(p^t)$, where w(e) is the sum of the Hamming weights of e(i) for $i=0,1,\ldots,k$. The result x^e is then computed using k (free) p-th powerings and approximately k multiplications in $F(p^t)$. It follows that e-th powering can trivially be done in $\log_2 p$ squarings and $w(e)-1$ multiplications in $F(p^t)$ if we use the basis $\{\alpha^i : i=1,2,\ldots,t\}$ for $F(p^t)$ over $F(p)$. For an asymptotically faster method see (5.2).

We conclude that for appropriately chosen medium-sized p and t exponentiation in $F(p^t)$ can be carried out very efficiently, not only because of the lower than usual number of operations in $F(p^t)$, but also because those operations themselves can be performed very quickly.

(2.2) Discrete logarithms. The discrete logarithm problem (DL) in $F(p^t)^*$ is: given g and y in $F(p^t)^*$, if possible find x such that $g^x=y$ in $F(p^t)$. If g generates $F(p^t)^*$ then such an x with $0\le x<p^t-1$ exists. More in general, if g generates a subgroup of order k of $F(p^t)^*$, then x (with $0\le x<k$) exists if and only if y is contained in $\langle g\rangle$, the subgroup generated by g.

Let

$$L[x,u,v] = \exp(v(\ln(x))^u(\ln\ln(x))^{1-u}).$$

Let s be the smallest divisor of t such that $\langle g\rangle$ can be embedded in $F(p^s)$. If $s<t$ then $F(p^s)$ is a true subfield of $F(p^t)$; note also that the order k of g must divide p^s-1. Then DL can be solved in heuristic expected asymptotic time $L[p^s,1/3,1.923+o(1)]$ for $p^s\to\infty$, which is subexponential in p^s (this follows from a standard linear algebra argument and [1,11]). Alternatively, DL can be solved in $O(q^{1/2})$ elementary operations in $F(p^s)$, where q is the largest prime dividing k (cf. [7]). Today an acceptable level of security would be obtained if t, p, k, s, and q are chosen such that $p^s\ge 2^{800}$ and $q\ge 2^{160}$, because it is believed that these choices would make it infeasible to solve DL with current algorithms and hardware (cf. [6]). Smaller values of p^s, but still with $p^s\ge 2^{500}$, provide marginal security (cf. [12]).

(2.3) Cyclotomic polynomials. The irreducible factorization of X^t-1 in $\mathbf{Z}[X]$ is given by

$$X^t-1 = \Pi_{d|t}\Phi_d(X),$$

where $\Phi_d(X)$ is the d-th cyclotomic polynomial (cf. [10]). The factor $\Phi_t(X)$ is the only irreducible factor of X^t-1 that does not appear in the factorization of X^s-1 for divisors s of t with s<t. Therefore the factors of the cardinality p^t-1 of $F(p^t)^*$ that are not factors of p^s-1 for divisors s of t with s<t, must be factors of $\Phi_t(p)$. Also, it follows from the Lemma that a prime factor >t of $\Phi_t(p)$ cannot be a factor of p^s-1 for any divisor s of t with s<t.

(2.4) Lemma. *Let* q>t *be a prime factor of* $\Phi_t(p)$. *Then* q *does not divide any* $\Phi_s(p)$ *for divisors* s *of* t *with* s<t.

Proof. Since q divides $\Phi_t(p)$, we find that $p^t\equiv 1 \bmod q$. Combined with q>t and the primality of q it follows that t divides q−1. Therefore $\Phi_t(X)$ and the $\Phi_s(X)$'s are among the distinct irreducible factors of $X^{q-1}-1$. The polynomial $X^{q-1}-1$ factors into q−1 distinct linear factors in F(q)[X], one of which is (X−p mod q) since q divides $\Phi_t(p)$ (i.e., p is a root of $\Phi_t(X)$ in F(q)[X]). Thus, no other factor of $X^{q-1}-1$ over F(q)[X] equals (X−p mod q), and therefore p is not a root of any of the $\Phi_s(X)$'s in F(q)[X], which proves the lemma.

It follows that if one wishes to find a prime q dividing p^t-1 such that q does not divide any p^s-1 for divisors s of t with s<t, it suffices to take a prime factor q>t of $\Phi_t(p)$. Because the degree of $\Phi_t(X)$ equals $\phi(t)$, with ϕ Euler's totient-function, and because $n>\phi(n)>n/(6\ln\ln(n))$ for n>6, we find that $\Phi_t(p)$ grows at least as fast as $p^{t/(6\ln\ln(t))}$.

3 Efficient DL-based cryptosystems

3.1) Combining advantages. We propose to combine the advantages of the variants proposed by Schnorr and by Agnew et al. by using $F(p^t)$ for appropriately chosen medium-sized p and t as the underlying finite field for the DL-based public key cryptosystems. In Schnorr's variant t is equal to 1 and large p's allowing appropriately sized subgroups can easily be determined; the advantage consists of short exponents and short signatures. In the variant by Agnew et al. p is equal to 2 and the advantage consists of the very efficient exponentiation that is obtained by using an optimal normal basis.

In (2.1) we have seen how medium-sized p and t can be chosen such that we get very efficient exponentiation by using an optimal normal basis for $F(p^t)$ over F(p). It remains to be shown how p and t can be selected in such a way that we can easily determine an appropriately sized subgroup that cannot be embedded in any true subfield of $F(p^t)$. From the discussions in (2.2), (2.3) and Lemma (2.4) it follows that it suffices to select p and t in such a way that not only the conditions in (2.1) are satisfied, but such that also $p^t\geq 2^{800}$ and $\Phi_t(p)$ has a prime factor $q\geq 2^{160}$.

An entirely straightforward way to find such p and t is as follows: fix some value for t such that t+1 is prime, and generate primes p of the required length until one is

found that is primitive modulo t+1 and for which it can easily be recognized that $\Phi_t(p)$ contains a sufficiently large prime factor q. The latter can be done using trial division with the primes up to, say, 10^5; any other reasonable bound or method will do. All events have sufficiently high probability for this approach to be effective. As can be seen below, this method works sufficiently quickly in practice. Once p, t, and q have been found, a generator of a subgroup of order q can be constructed in the usual fashion by computing the $((p^t-1)/q)$-th power g of some randomly selected element of $F(p^t)^*$ until $g\neq 1$. Note that $\langle g\rangle$ cannot be embedded in any $F(p^s)$ for divisors s of t with $s<t$ because the order q of g is constructed as a factor of p^t-1 that does not divide any p^s-1 for divisors s of t with $s<t$.

(3.2) Runtimes. The table below gives the runtimes in seconds on a Pentium 166MHz processor to generate 10 satisfactory triples p, t, q using various security levels: one using marginal security (t=18 and $\lceil \log_2 p \rceil$=32), one with slightly better security (t=10 and $\lceil \log_2 p \rceil$=64), two with acceptable security (t=30 and $\lceil \log_2 p \rceil$=32, and t=18 and $\lceil \log_2 p \rceil$=64), and one with high security (t=30 and $\lceil \log_2 p \rceil$=64). Note that $\Phi_{10}(X)=X^4-X^3+X^2-X+1$, $\Phi_{18}(X)=X^6-X^3+1$, and $\Phi_{30}(X)=X^8+X^7-X^5-X^4-X^3+X+1$. The last column gives the bit-length range of the resulting q's.

t	$\lceil \log_2 p \rceil$	$\phi(t)$	seconds	$\lceil \log_2 q \rceil$ in
18	32	6	74	[163,189]
10	64	4	114	[210,254]
30	32	8	120	[214,251]
18	64	6	209	[340,380]
30	64	8	240	[446,506]

(3.3) Remarks. Our construction is not as fast as the construction of primes p in Schnorr's variant, but since it is a one-time cost this is not a serious issue. Note that, if we are willing to invest more time in the construction of p, t, and q, we can aim for a fixed value for $[\log_2 q]$ that depends on and grows with the sizes of p and t (cf. (2.3)). Fixing the value of $[\log_2 q]$ irrespective of p and t is not realistic because it will quickly make the construction of satisfactory pairs p, t infeasible, unless one is willing to spend a lot of time on the factoring attempts of $\Phi_t(p)$. As an example, for t=30 and $\lceil \log_2 p \rceil$=32 it took three hours to find the 161-bit prime

$$244827772225842250509825320865863609168809687572 1$$

dividing $\Phi_{30}(2718153587)$, as opposed to just 120 seconds for 10 q's in the 214-bit range dividing $\Phi_{30}(p)$ for 10 different p's in the 32-bit range.

Thus, our construction is also not as general as Schnorr's one, because our subgroup sizes grow with p and t. It can be argued, however, that this is from a security point of view only natural and a distinct advantage of our proposal, and that variants (like [6]) that keep the subgroup sizes fixed are, on the long term, bound to run into security problems. Irrespective of the size of the multiplicative group, DL in

an order q subgroup can be solved in $O(q^{1/2})$ field operations, as mentioned above. Therefore, the security of systems using a fixed value for $[\log_2 q]$ will decrease with time. Keeping the security level constant implies that not only the size of the multiplicative group, but also the size of the subgroup has to grow with time.

A possible disadvantage of our variant is that there are fewer appropriate pairs p,t than there are primes p for Schnorr's variant (assuming comparable levels of security). If $\log_2 p$ is large enough ($\log_2 p \geq 64$, say) this is not a serious problem. For smaller p's it depends on the type of application whether or not this is a serious disadvantage.

(3.4) Security. The security of DL-based public key cryptosystems that work in subgroups of finite fields as proposed here is based on the observations from (2.2), (2.3), and Lemma (2.4). We are not aware of faster ways to solve the discrete logarithm problem in these subgroups than the methods mentioned above.

4 Performance analysis, comparison, and runtimes

Exponentiation is the most important and most time consuming operation in DL-based public key cryptosystems. In (4.1) we therefore analyze the time needed for a single exponentiation in a scheme that uses a subgroup and finite field as proposed in this paper. In (4.2) we compare the asymptotic performance of exponentiations using our, Schnorr's, and Agnew's variants, and in (4.3) we compare the number of multiplications required by a single exponentiation (i.e., a single signature generation) in various schemes using a realistic example; actual runtimes are given as well. More runtimes are given in (4.5).

(4.1) Analysis. As explained in (2.1), multiplication in $F(p^t)$ can be done in t^2 multiplications in F(p) (throughout this section we only count multiplications) and squaring can be done in $t+t(t-1)/2$ multiplications in F(p). Exponentiations will be done in the order q subgroup, so that exponents have at most $\lceil \log_2 q \rceil$ bits. Therefore, using the straightforward exponentiation method from (2.1), a single exponentiation can be expected to take approximately

$$t^2([\log_2 p] + \log_2 q)/2$$

multiplications in F(p). Here we make the reasonable assumption that on average $w(e)=(\log_2 q)/2$ for an exponent e, with w(e) as in (2.1).

(4.2) Asymptotic comparison. We compare the asymptotic performance of our variant with Schnorr's and Agnew's variants of the ElGamal signature scheme, assuming comparable levels of security. The expected number of multiplications in F(p) needed for one exponentiation in a subgroup of order q as constructed in (3.1) is given in (4.1), where we do not use any fast multiplication or exponentiation techniques (except that we make use of the fact that p-th powering is for free).

Comparison with Schnorr's variant. Let r be a prime close to p^t. One exponentiation in an order $\approx q$ subgroup of $F(r)^*$ can be expected to take

$$(¾+½)(\log_2 q) = 1.25 \log_2 q$$

multiplications in F(r). Here we assume that a squaring in F(r) takes (asymptotically) approximately 75 percent of the time of a multiplication in F(r), and that only half of the bits of the exponents are on. Again not assuming fast multiplication techniques, a single multiplication in F(r) takes approximately the same time as t^2 multiplications in F(p). We find that a single exponentiation using Schnorr's variant takes the same time as

$$1.25\, t^2 \log_2 q$$

multiplications in F(p), as opposed to

$$t^2(\log_2 p + \log_2 q)/2$$

for our variant. Thus, for realistic choices of $\log_2 p$ ($32 \le \log_2 p \le 64$, say) and $\log_2 q$ ($\log_2 q \ge 160$), our variant can be expected to be approximately twice as fast as Schnorr's variant. In an actual software implementation, our variant may even do better, as shown below.

Comparison with Agnew's variant. The relative performance of our variant and the one by Agnew et al. must be assessed in a different manner, because of the entirely different type of operations involved. If we do a simple operation count of both variants (and do not incorporate the possible effects of parallelized implementation, cf. [2]), we can say the following. Omitting constants, a single exponentiation using our variant takes

$$O(t^2(\log_2 p + \log_2 q)(\log_2 p)^2) = O(t^2(\log_2 p)^3) + O(t^2(\log_2 q)(\log_2 p)^2)$$

elementary operations (where we do not use fast multiplication techniques, cf. (4.1)). To get the same level of security as our variant, Agnew's variant has to be implemented in an extension field of degree approximately $t*\log_2 p$ over F(2). Using standard multiplication techniques, an exponentiation in a comparable realization of Agnew's scheme therefore takes

$$O(t^3(\log_2 p)^3)$$

elementary operations. Thus, our variant can be expected to have better asymptotic behavior than Agnew's variant because, based on existing methods to solve DL (cf. (2.2)), $t*\log_2 p$ should grow faster than $\log_2 q$.

(4.3) Comparison by example. The table below gives the approximate average number of standard multiplications (in millions) required by a single digital signature

generation as computed by a generic software implementation of various DL-based cryptosystems, all with $\lceil \log_2 p' \rceil = 960$. RSA with a 960-bit modulus and a full-length exponent (with or without use of 'Crt', the Chinese remaindering theorem) is also included. The actual runtimes in seconds on a Pentium 166MHz processor and the resulting signature lengths are given in the last two columns. Note that the signature verification runtime for each of the DL-based systems equals about twice the signature generation runtime (but only 25% more time than signature generation if multi-exponentiation is used); for RSA the signature verification runtime is negligible if a short public exponent is used.

By 'standard multiplications' we mean multiplications on 32-bit integers (`long` in C) or 56-bit mantissa floating point numbers (`double` in C). By 'generic software implementation' we mean that all programs were written in plain C without assembly language routines, and that no fast multiplication or m-ary exponentiation methods were used (see (5.2)). In the implementation used, the 32×32→64-bit multiply needed for long integer arithmetic takes three standard multiplies, and a 32×32→32-bit modular multiply with a 32-bit modulus takes four standard multiplies. The most advanced tool that was used is Montgomery multiplication for division-free modular arithmetic on long integers (cf. [8]). Of course, using more sophisticated methods would affect the data given in the table. Experiments indicated, however, that the relative performance is hardly affected.

Row (I) refers to an ordinary ElGamal signature scheme over a prime field with a 960-bit characteristic. Row (II) refers to Schnorr's variant of it, using a 224-bit subgroup of the same field. Row (III) refers to the variant proposed in this paper, with t=30 and a 32-bit characteristic. Schemes based on the variant proposed by Agnew et al. use bit-operations and require a hardware implementation to reach their full potential; therefore we have not included them in this comparison.

	t	$\lceil \log_2 p \rceil$	$\lceil \log_2 q \rceil$	million multiplies	seconds	signature length
(I)	1	960	N/A	6.6	0.83	1920 bits
(II)	1	960	224	1.5	0.23	448 bits
(III)	30	32	224	0.5	0.09	448 bits
RSA-Crt	((480+480)-bit modulus)			1.6	0.24	960 bits
RSA	(960-bit modulus)			6.6	0.82	960 bits

It can be seen from the table that the variant proposed in this paper requires substantially fewer multiplies and less runtime, compared to the other schemes considered. It follows from the data in the table that using ElGamal (i.e., working in the full multiplicative group) in a degree 30 extension of a 32-bit prime field that is represented with an optimal normal basis, results in about $(960/224)*0.5 \approx 2.1$ million multiplies and $(960/224)*0.09 \approx 0.39$ seconds per signature generation; this is much better than ElGamal over a prime field of comparable size (I), but inferior to Schnorr's variant with a 224-bit subgroup (II).

Of course, the performance of our new variant strongly depends on the efficiency of the multiplication in F(p). On current 32-bit architectures a relatively high degree of

efficiency can be achieved for 32-bit primes p. We anticipate that the emerging 64-bit architectures will allow us to efficiently use 64-bit primes as well.

(4.4) Remark. A small additional advantage of our variant is that the size of the public key is somewhat shorter than in Schnorr's variant. In the latter the public key consists of p,q,g,y, where p is a (large) prime, q a divisor of p−1, g a generator of a subgroup of order q, and y an element of that subgroup (i.e., $y=g^x$ for the secret key x). For a 960-bit p and 224-bit q, the public key takes 960+224+960+960=3104 bits (i.e., 388 bytes). Using our variant with t=30 and p a 32-bit prime, this can immediately be reduced to 32+30+224+960+960=2206 bits (i.e., 276 bytes). A small additional saving can be obtained by replacing q in the public key by its much smaller cofactor with respect to $\Phi_t(p)$; the value for q can then be computed using the cofactor and $\Phi_t(p)$.

(4.5) More runtimes. We conclude this section with more extensive examples of actual runtimes of the ElGamal signature scheme, Schnorr's variant, and the newly proposed variant. The rows refer to (I), (II), and (III) as in (4.3). Per column the value of t and the numbers of bits of p^t and q are given: the third column refers to t=30 and a 953-bit p^t with a 165-bit prime q. The signature generation runtimes are given in seconds on a Pentium 166MHz processor, averaged over only 10 examples (because the standard deviation is very small).

	18,512,160	30,900,220	30,953,165	36,960,315	36,1032,307
(I)	0.18	0.68	0.83	0.83	1.00
(II)	0.06	0.22	0.18	0.34	0.38
(III)	0.02	0.08	0.07	0.13	0.14

5 Additional remarks

(5.1) Adding flexibility. Considerably more t's can be used if we do not require an optimal normal basis, but simply carefully select the way we represent the t-th degree extension of F(p). Note that we may assume that t itself is not prime, because if that were the case we would not be able to easily find a small (but large enough) prime factor of $\Phi_t(p)$. If t has a large prime factor s such that s+1 is prime and k=t/s is small, then we can represent $F(p^t)$ as a k-th degree extension of $F(p^s)$. Assuming that p is primitive modulo s+1we represent $F(p^s)$ using an optimal normal basis over F(p). In this representation of $F(p^t)$ p-th powering is again almost for free; the reduction stage of the multiplication (and squaring) of elements of $F(p^t)$ takes slightly more than 2t−1 additions in $F(p^t)$, but is still relatively cheap.

This can for instance be seen as follows. Let k=2 and X^2+c be irreducible in $F(p^s)[X]$. We compute $X^p \bmod X^2+c$ once and for all as c_1X+c_0 in $F(p^s)[X]$. Then $(a_1X+a_0)^p$ for a_1X+a_0 in $F(p^s)[X]/(X^2+c)$ (which is isomorphic to $F(p^t)$) can be computed as $(a_1)^pX^p+(a_0)^p = c_1(a_1)^pX+c_0(a_1)^p+(a_0)^p$. Thus p-th powering in $F(p^t)$ can be

done in two (free) p-th powerings in $F(p^s)$, two multiplications in $F(p^s)$, and one addition in $F(p^s)$. The product of a_1X+a_0 and b_1X+b_0 in $F(p^s)[X]/(X^2+c)$ equals $(a_1b_0+a_0b_1)X+ a_0b_0-a_1b_1c$, so that multiplication in $F(p^t)$ can easily be done in five multiplications and two additions in $F(p^s)$. The performance figures for row (III) in the table from (4.3) are hardly affected if k is, say, at most 3. Note that subgroups can still be constructed as indicated above.

If no optimal normal basis can be used at all, one can try to find a minimal polynomial of the form $X^t+g(X)$ where $g(X)$ is a sparse low-degree monic polynomial, because that makes the reduction stage of the multiplication (and squaring) in $F(p^t)$ cheap. Subgroups can of course still be constructed as usual, but p-th powering is no longer for free. The performance figures from row (III) in the table from (4.3) are about 50 percent worse in this case. This may still be competitive with other variants.

Alternatively, one can use a normal basis for $F(p^t)$ over $F(p)$, so that p-th powering is still a permutation; the reduction stage may become more expensive unless the normal basis is chosen with care.

(5.2) m-ary Exponentiation. There are several methods to speed up exponentiation which can be applied to improve the figures in rows (I), (II), and the two RSA rows in the table from (4.3), and also those in row (III). For the former rows one can use the well known m-ary exponentiation (cf. [5: 4.6.3]), for row (III) a variant that applies to the situation in (2.1) can be found in [5: exercise 4.6.3.32]. The overall effect of application of these methods is that all figures in the table from (4.3) can be improved by about 20 percent. The same observation applies to the figures in the table from (4.5).

(5.3) Inversion in $F(p^t)$**.** We consider the computation of the inverse of a non-zero element $x\in F(p^t)$ that is represented using the optimal normal basis $\{\alpha^i : i=1,2,\ldots,t\}$ for $F(p^t)$ over $F(p)$. Although inversion of elements of $F(p^t)$ is not needed for the DL-based cryptosystems over finite fields discussed in this paper, it might be useful if the approach presented here is used in other circumstances. The traditional extended Euclidean algorithm can be used in the standard fashion after converting the representation of x to the power basis $\{\alpha^i : i=0,1,\ldots,t-1\}$ (remember that the identities $\alpha^t= -\alpha^{t-1}-\alpha^{t-2}-\ldots-\alpha-1$ and $\alpha^0= -\alpha^t-\alpha^{t-1}-\ldots-\alpha$ can be used to switch quickly between these two bases, cf. (2.1)). Thus, computing x^{-1} can be done in at most, approximately, t inverses in $F(p)$, t division-with-remainder computations on polynomials of degree $<t$ over $F(p)$, t multiplications in $F(p^t)$, and t^2 multiplications in $F(p)$. A division-free binary variant can also be used; it requires approximately the same number of computations.

Both these algorithms require substantial special purpose software. We discuss an exponentiation-based method because software for exponentiation is usually already available. From $x^m=x$ for $m=p^t$ it follows that $x^{-1}=x^{m-2}$, so that inversion can be done with a single exponentiation with exponent $m-2$. Writing $m-2$ in basis p, we can take advantage of the normal basis representation of x, and compute x^{m-2} using $\log_2 p$ squarings and $w(m-2)-1$ multiplications in $F(p^t)$, as in (2.1). This estimate can be improved by observing that

$$m-2 = p^t-2 = p^t-p+p-2 = (p-1)(p^{t-1}+ p^{t-2}+\ldots+p)+p-2,$$

i.e., the radix p digits of m–2 are all equal to p–1 except for the least significant one which equals p–2. Thus, with one exponentiation with exponent p–2 followed by a single multiplication we may compute x^{p-2} and x^{p-1}. The result x^{-1} then follows after t–2 multiplications and t–1 p-th powerings in $F(p^t)$.

It depends on the relative sizes of p and t if this exponentiation-based approach is faster than the straightforward application of the extended Euclidean algorithm. If $\log_2 p$ and t are of comparable size, then the speed of the two methods should be comparable.

(5.4) Computing random powers. In many digital signature schemes the most time consuming step of the signature generation is computing g^k, where g generates a subgroup of order q of the multiplicative group of a finite field and k is a random positive integer less than q. Various methods have been proposed to speed up this message independent step, without affecting the unpredictability of the random power k. If the finite field is $F(p^t)$ and it can be represented using a normal basis over F(p) (as the fields considered in this paper), it should be possible to develop new methods to quickly compute g^k for random (and known) k's because p-th powering is for free. We give an example of a possible approach, but leave this subject for further research.

First note that the $\log_2 p$ consecutive squares of g (as needed in the exponentiation, cf. (2.1)) can be tabulated, which would save all squaring steps. For Schnorr's variant this is less attractive because p is much bigger there. Further, we could restrict ourselves to k's with small w(k) (cf. (2.1)), because the number of multiplications in $F(p^t)$ to be carried out is w(k)–1. However, this would limit the space of k's and make them too predictable. A solution may be to consider exponents k' that are larger than q (but still smaller than p^t-1) with low w(k'), and to obtain k as k' modulo q. With the proper parameter setting the number of k's thus obtained can easily be made large enough. It would be interesting to see if the k's obtained are indeed sufficiently random.

References

1. M. Adleman, J. DeMarrais, *A subexponential algorithm for discrete logarithms over all finite fields*, Proceedings Crypto'93, Lecture Notes in Comp. Sci. **773**. 147-158 (1994).
2. G. Agnew, R.C. Mullin, I.M. Onyszchuk, S.A. Vanstone, *An implementation for a fast public-key cryptosystem*, Journal of Cryptology, **3**, 63-79 (1991).
3. T. ElGamal, *A public key cryptosystem and a signature scheme based on discrete logarithms*, IEEE Transactions on Information Theory, **31**, 469-472 (1985).
4. S. Gao, H.W. Lenstra, Jr., *Optimal normal bases*, Designs, Codes and Cryptography, **2**, 315-323 (1992).

5. D.E. Knuth, *The art of computer programming*, volume 2, *Seminumerical algorithms*, second edition, Addison-Wesley, Reading, Massachusetts, 1981.
6. D.W. Kravitz, *Digital signature algorithm*, U.S. Patent # 5,231,668, 27 Jul 1993.
7. A.K. Lenstra, H.W. Lenstra, Jr., *Algorithms in number theory*, J. van Leeuwen, editor, Handbook of Theoretical Computer Science, 674-715, Elsevier Science Publishers, 1990.
8. P.L. Montgomery, *Modular multiplication without trial division*, Math. Comp., **44**, 519-521 (1985).
9. R.C. Mullin, I.M. Onyszchuk, S.A. Vanstone, R.M. Wilson, *Optimal normal bases in GF(p^n)*, Discrete Appl. Math., **22**, 149-161 (1988/89).
10. H. Riesel, *Prime numbers and computer methods for factorization*, Birkhäuser, 1985.
11. O. Schirokauer, *Using number fields to compute logarithms in finite fields*, to appear.
12. O. Schirokauer, D. Weber, T. Denny, *Discrete logarithms: the effectiveness of the index calculus method*, H. Cohen, editor, Preproceedings ANTS II, Algorithmic number theory symposium, 327-351, Université de Bordeaux I, 1996.
13. C.P. Schnorr, *Efficient signature generation by smart cards*, Journal of Cryptology, **4**, 161-174 (1991).