An overview of the XTR public key system

Arjen K. Lenstra, Eric R. Verheul

Abstract. XTR is a new method to represent elements of a subgroup of a multiplicative group of a finite field. Application of XTR in cryptographic protocols leads to substantial savings both in communication and computational overhead without compromising security. This paper describes and explains the techniques and properties that are relevant for the XTR cryptosystem and its implementation. It is based on the material from [10, 11, 12, 28].

1. Introduction

XTR stands for 'ECSTR', which is an abbreviation for Efficient and Compact Subgroup Trace Representation. It is a novel method that makes use of traces to represent and calculate powers of elements of a subgroup of a finite field. XTR is not the first method to do so. The LUC cryptosystem (cf. [25], and also [13, 17, 18, 22]) uses the trace over $\mathrm{GF}(p)$ to represent elements of the order $p+1$ subgroup of $\mathrm{GF}(p^2)^*$. Compared to the traditional representation this leads to a factor 2 reduction in the representation size. The variant described in [7] uses the subgroup of order p^2+p+1 of $\mathrm{GF}(p^3)^*$ instead, but as a result sizes are reduced by only a factor 1.5. XTR uses the trace over $\mathrm{GF}(p^2)$ to represent elements of the order p^2-p+1 subgroup of $\mathrm{GF}(p^6)^*$, thereby achieving a factor 3 size reduction. Also, the resulting calculations are appreciably faster than using the standard representation. The factor 3 size reduction was first achieved—at much lower speed than XTR, but using the same subgroup—in the predecessor paper [3].

From a security point of view, XTR is a traditional discrete logarithm system: for its security it relies on the difficulty of solving discrete logarithm related problems in the multiplicative group of a finite field. Thus, XTR is not based on any new primitive or new allegedly hard problem—on the contrary, it is based on the primitive underlying the very first public key cryptosystem, the Diffie-Hellman key agreement protocol. Other advantages of XTR are its very fast parameter and key selection (much faster than RSA, orders of magnitude faster than ECC), small key sizes (much smaller than RSA, comparable with ECC for current security settings), and speed (overall comparable with ECC for current security settings). Combined with its very easy programmability, this makes XTR an excellent public key system for a very wide variety of environments, ranging from smart cards to web servers, without the need to share system parameters with other users (as is often the case in ECC systems). For actual implementation results and comparisons with other cryptosystems, see [10, Subsection 4.4].

The purpose of this paper is to present a unified description of the XTR results obtained so far (cf. [10, 11, 12, 28]). Proofs are included only inasmuch they are required to implement XTR; all other proofs can be found in the original papers on which this survey is based. An outline of the paper is as follows:

- Section 2 introduces the mathematics of XTR including the basic parameters and the fundamental algorithms to calculate traces of powers.
- Section 3 describes the selection of XTR parameters and keys.
- Section 4 contains XTR-based encryption schemes, providing confidentiality services.
- Section 5 focuses on XTR-based digital signature schemes, providing authentication and non-repudiation services, and includes methods to reduce public key sizes for these applications.
- Section 6 describes how to efficiently verify that data exchanged during XTR-based protocols are correctly formatted. Such checks can be used to prevent so-called subgroup based attacks.
- Section 7 summarizes the most relevant security properties of XTR. This includes a result providing evidence that the XTR group is more secure than an algebraically isomorphic supersingular elliptic curve over $\mathrm{GF}(p^2)$, thereby settling a problem posed by [16].

2. Fundamentals of XTR

2.1. XTR supergroup and XTR group

Many cryptographic protocols used to be based on a generator of the full multiplicative group of a finite field. Schnorr introduced the idea to replace this generator by the generator of a relatively small subgroup of sufficiently large prime order q (cf. [24]). This same idea is used in XTR in a specific setting, namely XTR uses a subgroup of prime order q of the order $p^2 - p + 1$ subgroup of $\mathrm{GF}(p^6)^*$. The latter group is referred to as the *XTR supergroup* and the order q subgroup $\langle g \rangle$ generated by g is referred to as the *XTR (sub)group*. The XTR supergroup is not contained in any proper subfield of $\mathrm{GF}(p^6)$ (cf. [9]). Combined with the choice of q it follows that computing discrete logarithms in $\langle g \rangle$ is as hard, in general, as it is in $\mathrm{GF}(p^6)^*$ (cf. [10, Section 5]).

The reason that XTR uses this specific subgroup $\langle g \rangle$ is not just that it provides the full $\mathrm{GF}(p^6)^*$ security, but also that the elements of the XTR supergroup, and thus of $\langle g \rangle$, allow a very efficient representation, at a small cost: if one is willing to give up the distinction between elements and their conjugates over $\mathrm{GF}(p^2)$, then not only elements of the XTR supergroup can be represented using an element of $\mathrm{GF}(p^2)$ as opposed to $\mathrm{GF}(p^6)$ (i.e., just a third of the usual number of bits). But also calculations take place in $\mathrm{GF}(p^2)$ instead of $\mathrm{GF}(p^6)$ and can thus be performed much faster than usual. As shown in Subsection 2.3 below, this is all a consequence of the particularly nice form of the minimal polynomial over $\mathrm{GF}(p^2)$ of the elements of the XTR supergroup. First, however, it is described how computations in $\mathrm{GF}(p^2)$ can be done efficiently.

2.2. Arithmetic operations in $\mathrm{GF}(p^2)$

As set forth in Subsection 2.1 above, a representation of $\mathrm{GF}(p^2)$ is needed that allows efficient arithmetic operations, where p is a prime such that $p^2 - p + 1$ has a sufficiently large prime factor q. Here it is indicated how this can be done, irrespective of the existence of q. Selection of p (and q) is described in Section 3.

Let p be a prime that is $2 \bmod 3$. It follows that $(X^3-1)/(X-1) = X^2+X+1$ is irreducible over $\mathrm{GF}(p)$ and that the roots α and α^p form an optimal normal basis for $\mathrm{GF}(p^2)$ over $\mathrm{GF}(p)$, i.e., $\mathrm{GF}(p^2) \cong \{x_1\alpha + x_2\alpha^p\colon x_1, x_2 \in \mathrm{GF}(p)\}$. With $\alpha^i = \alpha^{i \bmod 3}$ it follows that

$$\mathrm{GF}(p^2) \cong \{x_1\alpha + x_2\alpha^2\colon \alpha^2 + \alpha + 1 = 0 \text{ and } x_1, x_2 \in \mathrm{GF}(p)\}.$$

Note that in this representation of $\mathrm{GF}(p^2)$ an element t of $\mathrm{GF}(p)$ is represented as $-t\alpha - t\alpha^2$, e.g. 3 is represented as $-3\alpha - 3\alpha^2$. The cost of arithmetic operations in $\mathrm{GF}(p^2)$ follows from Lemma 2.2.1 below.

Lemma 2.2.1 [10, Lemma 2.1.1]. *Let $x, y, z \in \mathrm{GF}(p^2)$ with $p \equiv 2 \bmod 3$. Not counting additions or subtractions in* $\mathrm{GF}(p)$:

(i) *computing x^p is for free*;

(ii) *computing x^2 takes two multiplications in* $\mathrm{GF}(p)$;

(iii) *computing $x \cdot y$ takes three multiplications in* $\mathrm{GF}(p)$;

(iv) *computing $x \cdot z - y \cdot z^p$ takes four multiplications in* $\mathrm{GF}(p)$.

Proof. Let $x = x_1\alpha + x_2\alpha^2$, $y = y_1\alpha + y_2\alpha^2$, $z = z_1\alpha + z_2\alpha^2 \in \mathrm{GF}(p^2)$. To prove (i) observe that

$$x^p = x_1^p\alpha^p + x_2^p\alpha^{2p} = x_2\alpha + x_1\alpha^2.$$

To prove (ii), write

$$(x_1\alpha + x_2\alpha^2)^2 = x_2(x_2 - 2x_1)\alpha + x_1(x_1 - 2x_2)\alpha^2.$$

Under the reasonable assumption that a squaring in $\mathrm{GF}(p)$ takes 80% of the time of a multiplication in $\mathrm{GF}(p)$ (cf. [5]), this is faster than the three squarings in $\mathrm{GF}(p)$ that would result if x^2 were computed using the Karatsuba-like approach that is used for (iii). To compute

$$x \cdot y = (x_2y_2 - x_1y_2 - x_2y_1)\alpha + (x_1y_1 - x_1y_2 - x_2y_1)\alpha^2,$$

compute $x_1 \cdot y_1$, $x_2 \cdot y_2$, and $(x_1 + x_2) \cdot (y_1 + y_2)$, so that $x_1 \cdot y_2 + x_2 \cdot y_1$ and thus $x \cdot y$ follow using four subtractions. This proves (iii). Finally, (iv) follows from

$$\begin{aligned} x \cdot z - y \cdot z^p = &(z_1(y_1 - x_2 - y_2) + z_2(x_2 - x_1 + y_2))\alpha \\ &+ (z_1(x_1 - x_2 + y_1) + z_2(y_2 - x_1 - y_1))\alpha^2. \end{aligned} \qquad \square$$

2.3. Traces

The *conjugates* over $\mathrm{GF}(p^2)$ of $h \in \mathrm{GF}(p^6)$ are h, h^{p^2}, and h^{p^4}. The sum of the conjugates over $\mathrm{GF}(p^2)$ of $h \in \mathrm{GF}(p^6)$ is known as the *trace* $\mathrm{Tr}(h) = h + h^{p^2} + h^{p^4}$ over $\mathrm{GF}(p^2)$ of h. From this definition and the fact that $h^{p^6} = 1$ it follows that $\mathrm{Tr}(h)^{p^2} = \mathrm{Tr}(h)$, so that $\mathrm{Tr}(h) \in \mathrm{GF}(p^2)$. Furthermore,

$$\mathrm{Tr}(h_1 + h_2) = \mathrm{Tr}(h_1) + \mathrm{Tr}(h_2) \text{ and } \mathrm{Tr}(c \cdot h_1) = c \cdot \mathrm{Tr}(h_1)$$

for $h_1, h_2 \in \mathrm{GF}(p^6)$ and $c \in \mathrm{GF}(p^2)$. That is, the trace over $\mathrm{GF}(p^2)$ is $\mathrm{GF}(p^2)$-linear. Unless specified otherwise, conjugates and traces in this paper are over $\mathrm{GF}(p^2)$.

Now let $g \in \mathrm{GF}(p^6)^*$ be of order > 3 and dividing $p^2 - p + 1$. As argued in Subsection 2.1 the subgroup $\langle g \rangle$ is as secure, with respect to the discrete logarithm related problems, as the full multiplicative group $\mathrm{GF}(p^6)^*$, assuming a sufficiently large prime divides the order of g. For that reason g is later chosen as an element of order q dividing $p^2 - p + 1$. For the present purposes, however, q is not needed and it suffices to take g of order > 3 and dividing $p^2 - p + 1$.

Because $p^2 \equiv p - 1 \bmod p^2 - p + 1$ and $p^4 \equiv -p \bmod p^2 - p + 1$, the conjugates of g are g, g^{p-1} and g^{-p}, so that $\mathrm{Tr}(g) = g + g^{p-1} + g^{-p}$. It follows that the product of the conjugates equals 1, so that the polynomial $(X - g)(X - g^{p-1})(X - g^{-p})$ has the form $X^3 - \mathrm{Tr}(g)X^2 + uX - 1$, where

$$u = g \cdot g^{p-1} + g \cdot g^{-p} + g^{p-1} \cdot g^{-p} = g^p + g^{1-p} + g^{-1} = \mathrm{Tr}(g)^p \in \mathrm{GF}(p^2)$$

(the last equality follows from $1 - p \equiv -p^2$ and $-1 \equiv p^2 - p$, both modulo $p^2 - p + 1$). Thus

$$(X - g)(X - g^{p-1})(X - g^{-p}) = X^3 - \mathrm{Tr}(g)X^2 + \mathrm{Tr}(g)^p X - 1 \in \mathrm{GF}(p^2)[X]$$

is actually the minimal polynomial of g over $\mathrm{GF}(p^2)$, and this polynomial—and thereby g's conjugates—is fully determined by $\mathrm{Tr}(g)$. This is the fundamental observation underlying XTR. The same holds for any power of g: for any integer n the conjugates of g^n are the roots of $X^3 - \mathrm{Tr}(g^n)X^2 + \mathrm{Tr}(g^n)^p X - 1 \in \mathrm{GF}(p^2)[X]$, and the latter polynomial is fully determined by $\mathrm{Tr}(g^n)$.

This observation is useful for cryptographic purposes if there is a way to efficiently compute $\mathrm{Tr}(g^n)$ given $\mathrm{Tr}(g)$: in cryptographic protocols $g^n \in \mathrm{GF}(p^6)$ can then be replaced by $\mathrm{Tr}(g^n) \in \mathrm{GF}(p^2)$, thereby obtaining a saving of a factor 3 in the representation size. It is shown in Algorithm 2.3.5 below that $\mathrm{Tr}(g^n)$ can indeed be computed quickly given $\mathrm{Tr}(g)$—it turns out that this computation can be done much faster than computing g^n given g, so that a considerable speed advantage is obtained as well.

Definition 2.3.1 [10, Definition 2.3.1]**.** For $c \in \mathrm{GF}(p^2)$ define

$$F(c,X) = X^3 - cX^2 + c^pX - 1 \in \mathrm{GF}(p^2)[X],$$

and define $\tau(c,n) = h_0^n + h_1^n + h_2^n$ for $n \in \mathbf{Z}$, where h_0, h_1, h_2 are the (not necessarily distinct) roots of $F(c,X)$ in $\mathrm{GF}(p^6)$. The shorthand c_n is used for $\tau(c,n)$.

The definition of $F(c,X)$ for all c in $\mathrm{GF}(p^2)$ is more general than that implied by the argument before Definition 2.3.1. There only c are considered of the form $\mathrm{Tr}(g)$ for g of order > 3 and dividing $p^2 - p + 1$. The more general definition allows the application in Section 3. For the present purposes, if $c = \mathrm{Tr}(g)$ then $c_n = \mathrm{Tr}(g^n)$ (as argued above), which makes fast computation of c_n relevant. This can be done based on the following properties of c_n and $F(c,X)$.

Lemma 2.3.2 [10, Lemmas 2.3.2 and 2.3.4]**.**

(i) $c = c_1$.

(ii) $c_{-n} = c_{np} = c_n^p$ *for* $n \in \mathbf{Z}$.

(iii) $c_n \in \mathrm{GF}(p^2)$ *for* $n \in \mathbf{Z}$.

(iv) $c_{u+v} = c_u \cdot c_v - c_v^p \cdot c_{u-v} + c_{u-2v}$ *for* $u, v \in \mathbf{Z}$.

(v) *Either all* h_j *have order dividing* $p^2 - p + 1$ *and* > 3 *or all* $h_j \in \mathrm{GF}(p^2)$. *In particular,* $F(c,X)$ *is irreducible if and only if its roots have order dividing* $p^2 - p + 1$ *and* > 3.

(vi) $F(c,X)$ *is reducible over* $\mathrm{GF}(p^2)$ *if and only if* $c_{p+1} \in \mathrm{GF}(p)$.

Corollary 2.3.3 [10, Corollary 2.3.5]**.** *Let* c, c_{n-1}, c_n, *and* c_{n+1} *be given.*

(i) *Computing* $c_{2n} = c_n^2 - 2c_n^p$ *takes two multiplications in* $\mathrm{GF}(p)$.

(ii) *Computing* $c_{n+2} = c \cdot c_{n+1} - c^p \cdot c_n + c_{n-1}$ *takes four multiplications in* $\mathrm{GF}(p)$.

(iii) *Computing* $c_{2n-1} = c_{n-1} \cdot c_n - c^p \cdot c_n^p + c_{n+1}^p$ *takes four multiplications in* $\mathrm{GF}(p)$.

(iv) *Computing* $c_{2n+1} = c_{n+1} \cdot c_n - c \cdot c_n^p + c_{n-1}^p$ *takes four multiplications in* $\mathrm{GF}(p)$.

Proof. Use Lemmas 2.2.1 and 2.3.2. □

Definition 2.3.4 [10, Definition 2.3.6]**.** Let $S_n(c) = (c_{n-1}, c_n, c_{n+1}) \in \mathrm{GF}(p^2)^3$.

Algorithm 2.3.5 (Computation of $S_n(c)$ given n and c [10, Algoritm 2.3.7])**.**

- If $n < 0$, apply this algorithm to $-n$ and c, and apply Lemma 2.3.2.(ii) to the resulting value.
- If $n = 0$, then $S_0(c) = (c^p, 3, c)$ (cf. Lemma 2.3.2.(ii)).
- If $n = 1$, then $S_1(c) = (3, c, c^2 - 2c^p)$ (cf. Corollary 2.3.3.(i)).
- If $n = 2$, use Corollary 2.3.3.(ii) and $S_1(c)$ to compute c_3 and thereby $S_2(n)$.
- Otherwise, to compute $S_n(c)$ for $n > 2$ define $\bar{S}_i(c) = S_{2i+1}(c)$ and let $\bar{m} = n$. If $\bar{m}$ is even, then replace $\bar{m}$ by $\bar{m} - 1$. Let $\bar{m} = 2m + 1$, $k = 1$, and compute $\bar{S}_k(c) = S_3(c)$ using Corollary 2.3.3.(ii) and $S_2(c)$.
  Let $m = \sum_{j=0}^{r} m_j 2^j$ with $m_j \in \{0, 1\}$ and $m_r = 1$. For $j = r-1, r-2, \ldots, 0$ in succession do the following:

— If $m_j = 0$ then use

$$\bar{S}_k(c) = (c_{2k}, c_{2k+1}, c_{2k+2}) \text{ to compute } \bar{S}_{2k}(c) = (c_{4k}, c_{4k+1}, c_{4k+2})$$

(using Corollary 2.3.3.(i) for c_{4k} and c_{4k+2} and Corollary 2.3.3.(iii) for c_{4k+1}).

— If $m_j = 1$ then use

$$\bar{S}_k(c) = (c_{2k}, c_{2k+1}, c_{2k+2}) \text{ to compute } \bar{S}_{2k+1}(c) = (c_{4k+2}, c_{4k+3}, c_{4k+4})$$

(using Corollary 2.3.3.(i) for c_{4k+2} and c_{4k+4} and Corollary 2.3.3.(iv) for c_{4k+3}).

Comment. The great similarity between the computation for $m_j = 0$ and $m_j = 1$ makes this algorithm much less susceptible to environmental attacks than usual exponentiation routines.

— Replace k by $2k + m_j$.

After this iteration $k = m$ and $S_{\bar{m}}(c) = \bar{S}_m(c)$. If n is even use

$$S_{\bar{m}}(c) = (c_{\bar{m}-1}, c_{\bar{m}}, c_{\bar{m}+1}) \quad \text{to compute} \quad S_{\bar{m}+1}(c) = (c_{\bar{m}}, c_{\bar{m}+1}, c_{\bar{m}+2})$$

(using Corollary 2.3.3.(ii)) and replace $\bar{m}$ by $\bar{m}+1$. As a result $S_n(c) = S_{\bar{m}}(c)$.

Theorem 2.3.6 [10, Theorem 2.3.8]**.** *Given the sum c of the roots of $F(c, X)$, the sum c_n of the n^{th} powers of the roots of $F(c, X)$ can be computed in $8\log_2(n)$ multiplications in* $\mathrm{GF}(p)$.

Thus, given the representation $\mathrm{Tr}(g) \in \mathrm{GF}(p^2)$ of the conjugates of g, the representation $\mathrm{Tr}(g^n) \in \mathrm{GF}(p^2)$ of the conjugates of the n^{th} power of g can be computed at the cost of $8\log_2(n)$ multiplications in $\mathrm{GF}(p)$, for any integer n. This compares quite favorably to the speed of the computation of $g^n \in \mathrm{GF}(p^6)$ given $g \in \mathrm{GF}(p^6)$ (cf. [10, Subsection 2.4]).

Before cryptographic applications of this alternative representation of the elements of $\langle g \rangle$ can be discussed, it remains to show how p, q, and $\mathrm{Tr}(g)$ are chosen.

3. XTR parameter and key selection

3.1. Selection of p and q

As indicated in Subsection 2.1, primes p and q have to be selected in such a way that q divides $p^2 - p + 1$, and such that the resulting fields and subgroups are large enough to withstand known attacks. Furthermore, in order to be able to use the fast $\mathrm{GF}(p^2)$ arithmetic described in Subsection 2.2, the prime p should be $2 \bmod 3$. Primes p that are $1 \bmod 3$ can be used as well, but they may not always achieve the same speed.

Let P and Q denote the bit lengths of the primes p and q to be generated, respectively. The prime p should be such that the field $\mathrm{GF}(p^6)$ cannot be effectively attacked using the Discrete Logarithm variant of the Number Field Sieve, and the prime q should be such that an order q subgroup cannot be effectively attacked using Pollard's rho method. For current security levels, a choice where $6P$ is close to 1024 and Q is close to 160 is acceptable. Choosing P much smaller than Q cannot be recommended given current cryptanalytic methods.

In principle p may also be a non-trivial prime power (cf. [10, Section 6]). This is, however, incompatible with the first two methods presented in this subsection, and makes selection of a proper p and q in general much harder. It can be used in conjunction with Algorithm 3.1.3, but not efficiently if q must be much smaller than intended there.

A detailed analysis of the run times of the algorithms in this subsection is straightforward and left to the reader.

Algorithm 3.1.1 (Selection of q and 'nice' p [10, Algorithm 3.1.1]).

1. Find $r \in \mathbf{Z}$ such that $q = r^2 - r + 1$ is a Q-bit prime.
2. Find $k \in \mathbf{Z}$ such that $p = r + k \cdot q = kr^2 + (1 - k)r + k$ is a P-bit prime that is $2 \bmod 3$.

Algorithm 3.1.1 is very fast and can be used to find primes p that satisfy a second degree polynomial with small coefficients. Such p lead to fast arithmetic operations in $\mathrm{GF}(p)$. In particular if the search for k is restricted to $k = 1$ (i.e., search for an r such that both $r^2 - r + 1$ and $r^2 + 1$ are prime and such that $r^2 + 1 \equiv 2 \bmod 3$, thereby slowing down Algorithm 3.1.1 considerably) the primes p have a very nice form; note that in this case r must be even and $p \equiv 1 \bmod 4$.

On the other hand, such 'nice' p may be undesirable from a security point of view because they may make application of the Discrete Logarithm variant of the Number Field Sieve easier. Another method to generate p and q that does not have this disadvantage (and thus neither the advantage of fast arithmetic modulo p) and that is about equally fast, is the following.

Algorithm 3.1.2 (Selection of q and p [10, Algorithm 3.1.2]).

1. First, select a Q-bit prime $q \equiv 7 \bmod 12$.
2. Find the roots r_1 and r_2 of $X^2 - X + 1 \bmod q$.
   Comment. It follows from $q \equiv 1 \bmod 3$ and quadratic reciprocity that r_1 and r_2 exist. Since $q \equiv 3 \bmod 4$ they can be found using a single $\frac{q+1}{4}^{\text{th}}$ powering modulo q.
3. Find $k \in \mathbf{Z}$ such that $p = r_i + k \cdot q$ for $i = 1$ or 2 is a P-bit prime that is $2 \bmod 3$.

In Section 3.5 a fast algorithm is given to verify that an element $c \in \mathrm{GF}(p^2)$ is the trace of an element of the XTR supergroup: according to Theorem 3.5.5 this can be done at the cost of about $1.8 \log_2(p)$ multiplications in $\mathrm{GF}(p)$. Verifying XTR subgroup membership, however, amounts to checking that $c_q = 3$. This costs $8 \log_2(q)$ multiplications in $\mathrm{GF}(p)$ (cf. Algorithm 2.3.5), and is thus substantially more expensive than XTR supergroup membership verification. In some applications (and in particular to avoid so-called subgroup attacks, cf. Subsection 6.1), XTR subgroup membership verification is required. In that case it turns out to be practical to choose the size Q of the XTR subgroup close to the size of the XTR supergroup: the amount of damage can in general be bounded by $2P - Q$, so that only limited damage can be done if $2P - Q$ is small and the 'cheap' XTR supergroup test is carried out instead of the expensive XTR subgroup test. Furthermore, by using 'short exponents' (e.g. of size 170 bits, cf. [23]) in this setting, the use of a large XTR subgroup does not have a negative impact on the speed of cryptographic operations. The best that can be achieved (given that $p = 2 \bmod 3$) is to choose p such that $(p^2 - p + 1)/3$ is prime (and equal to q). The following straightforward algorithm determines satisfactory primes p and q for such applications.

Algorithm 3.1.3 (Selection of large q and p)**.** Select a P-bit prime p until $p^2 - p + 1$ is of the form $q \cdot s$ where q is a prime number and s is small.

Algorithm 3.1.3 can be improved by choosing p in such a way that some fixed number of small primes does not divide $(p^2 - p + 1)/3$.

An alternative way to render subgroup attacks mostly ineffective, is by choosing p and q such that $(p^2 - p + 1)/q$ is a small multiple of a prime of the same order of magnitude as q. Assuming that Q is only slightly smaller than P, finding such p and q can be achieved by the following simple (but on average considerably slower) adaptation of Algorithm 3.1.2.

Algorithm 3.1.4 (Selection of subgroup attack resistant q and p)**.**

1. First, select a Q-bit prime $q \equiv 7 \bmod 12$.
2. Find the roots r_1 and r_2 of $X^2 - X + 1 \bmod q$.
3. Find $k \in \mathbf{Z}$ such that $p = r_i + k \cdot q$ for $i = 1$ or 2 is a P-bit prime that is $2 \bmod 3$ and $(p^2 - p + 1)/q$ is of the form $s \cdot q'$ for a small s and prime q' of at least Q bits.

3.2. Basic subgroup selection

Given p and $q > 3$, it remains to find an element $c \in \mathrm{GF}(p^2)$ such that $c = \mathrm{Tr}(g)$ for an element $g \in \mathrm{GF}(p^6)$ of order q dividing $p^2 - p + 1$. Note that finding $\mathrm{Tr}(g)$ suffices and that g itself is not needed. But, given $\mathrm{Tr}(g)$, a generator g of the XTR subgroup (cf. Subsection 2.1) can be found by determining any root of $F(\mathrm{Tr}(g), X)$ (cf. Definition 2.3.1).

According to Lemma 2.3.2.(v), if $c \in \mathrm{GF}(p^2)$ is such that $F(c, X)$ is irreducible, then c is the trace of an element $h \in \mathrm{GF}(p^6)^*$ of order > 3 dividing $p^2 - p + 1$. Thus, if furthermore $c_{(p^2-p+1)/q} \neq 3$ (cf. Definition 2.3.1), which can be verified using Algorithm 2.3.5 at the cost of $8 \log_2((p^2 - p + 1)/q)$ multiplications in $\mathrm{GF}(p)$, then $c_{(p^2-p+1)/q}$ is the trace of an element of order q. Consequently, $\mathrm{Tr}(g)$ can be defined as $c_{(p^2-p+1)/q}$. It remains to find $c \in \mathrm{GF}(p^2)$ such that $F(c, X)$ is irreducible. According to the following lemma, this can be done by randomly picking c's until $F(c, X)$ is irreducible.

Lemma 3.2.1 [10, Lemma 3.2.1]**.** *For a randomly selected* $c \in \mathrm{GF}(p^2)$ *the probability that* $F(c, X) \in \mathrm{GF}(p^2)[X]$ *is irreducible is about one third.*

As shown in Subsections 3.3 and 3.5, testing $F(c, X)$ for irreducibility for a randomly selected $c \in \mathrm{GF}(p^2)$ can be done very fast, but those methods require additional code. The following method, based on Lemma 2.3.2.(vi), requires only Algorithm 2.3.5 and thus hardly any additional code.

Algorithm 3.2.2 (Computation of $\mathrm{Tr}(g)$ [10, Algorithm 3.2.2])**.**

1. Apply Algorithm 2.3.5 to $n = p + 1$ and a random $c \in \mathrm{GF}(p^2) \setminus \mathrm{GF}(p)$ to compute c_{p+1}.
2. If $c_{p+1} \in \mathrm{GF}(p)$ then return to Step 1.
3. Apply Algorithm 2.3.5 to $n = (p^2 - p + 1)/q$ and c to compute $d = c_{(p^2-p+1)/q}$.
4. If $d = 3$, then return to Step 1.
5. Let $\mathrm{Tr}(g) = d$.

Theorem 3.2.3 [10, Theorem 3.2.3]**.** *Algorithm* 3.2.2 *computes an element of* $\mathrm{GF}(p^2)$ *that equals* $\mathrm{Tr}(g)$ *for some* $g \in \mathrm{GF}(p^6)$ *of order* q. *It can be expected to require* $\frac{3q}{q-1}$ *applications of Algorithm* 2.3.5 *with* $n = p + 1$ *and* $\frac{q}{q-1}$ *applications with* $n = (p^2 - p + 1)/q$.

The next subsection contains a faster method to find the trace of a generator of the XTR subgroup, based on a more direct method to test $F(c, X)$ for irreducibility.

3.3. Subgroup selection using an irreducibility test

The roots of a third-degree equation can be computed directly by means of one of Cardano's classical formulas, more precisely Scipione del Ferro's method (cf. [19, page 559]).

Algorithm 3.3.1 (Scipione del Ferro, ∼1465–1526)**.** To compute the roots of $f(X) = aX^3 + bX^2 + dX + e$ in a field of characteristic $p \neq 2, 3$, do the following.

1. Compute the polynomial
$$\frac{f(X - \frac{b}{3a})}{a} = X^3 + f_1 X + f_0$$
with
$$f_1 = \frac{3ad - b^2}{3a^2} \text{ and } f_0 = \frac{27a^2e - 9abd + 2b^3}{27a^3}.$$
2. Compute the discriminant $\Delta = f_0^2 + 4(\frac{f_1}{3})^3$ of the polynomial $X^2 + f_0 X - (\frac{f_1}{3})^3$, and compute its roots $r_{1,2} = \frac{-f_0 \pm \sqrt{\Delta}}{2}$ using the standard method to compute the roots of a second-degree equation.
3. If $r_1 = r_2 = 0$, then let $u = v = 0$. Otherwise, let $r_1 \neq 0$, compute a cube root u of r_1, and let $v = -\frac{f_1}{3u}$ be a cube root of r_2.

4. The roots of $f(X)$ are
$$u+v-\frac{b}{3a}, \quad uw+vw^2-\frac{b}{3a}, \quad uw^2+vw-\frac{b}{3a},$$
where $w \in \mathrm{GF}(p^2)$ is a non-trivial cube root of unity, i.e., $w^3=1$ and $w^2+w+1=0$.

Lemma 3.3.2. *With Algorithm 3.3.1 applied to $f(X)=F(c,X)\in \mathrm{GF}(p^2)[X]$:*

(i) $\Delta \in \mathrm{GF}(p)$ (*cf.* [11, *Lemma* 3.3]).

(ii) *Δ is a quadratic residue in $\mathrm{GF}(p)$ if and only if either $F(c,X)$ is irreducible in $\mathrm{GF}(p^2)[X]$ or all roots in $\mathrm{GF}(p^2)$ of $F(c,X)$ have order dividing $p+1$* (*cf.* [11, *Lemma* 3.6]).

(iii) *$F(c,X)\in \mathrm{GF}(p^2)[X]$ is reducible over $\mathrm{GF}(p^2)$ if and only if r_1 satisfies $r_1^{p(p+1)/3}=r_1^{(p+1)/3}$ (i.e., r_1 is a cube in $\mathrm{GF}(p^2)$)* (*cf.* [11, *Corollary* 3.4]).

Algorithm 3.3.1 and Lemma 3.3.2 lead to the following irreducibility test.

Algorithm 3.3.3 (Irreducibility test [11, Algorithm 3.5])**.** To test $F(c,X)$ in $\mathrm{GF}(p^2)[X]$ for irreducibility over $\mathrm{GF}(p^2)$ with $p\neq 2,3$, do the following.

1. Compute
$$f_0=\frac{-27+9c^{p+1}-2c^3}{27} \text{ and } f_1=c^p-\frac{c^2}{3}\in \mathrm{GF}(p^2).$$
2. If $\Delta=f_0^2+4(\frac{f_1}{3})^3\in \mathrm{GF}(p)$ (cf. Lemma 3.3.2.(i)) is a quadratic non-residue in $\mathrm{GF}(p)$ then $F(c,X)$ is reducible (cf. Lemma 3.3.2.(ii)).
   Comment. This step requires the computation of a Jacobi symbol.
3. Otherwise, compute $r_1=\frac{-f_0+\sqrt{\Delta}}{2}\in \mathrm{GF}(p^2)$.
4. Compute $y=r_1^{(p+1)/3}\in \mathrm{GF}(p^2)$, then $F(c,X)$ is irreducible if and only if $y\neq y^p$ (cf. Lemma 3.3.2.(iii)).

Algorithm 3.3.4 (Computation of $\mathrm{Tr}(g)$)**.**

1. Pick a random $c\in \mathrm{GF}(p^2)\setminus \mathrm{GF}(p)$ and use Algorithm 3.3.3 to test $F(c,X)$ for irreducibility.
2. If $F(c,X)$ is reducible, then return to Step 1.
3. Apply Algorithm 2.3.5 to $n=(p^2-p+1)/q$ and c to compute $d=c_{(p^2-p+1)/q}$.
4. If $d=3$, then return to Step 1.
5. Let $\mathrm{Tr}(g)=d$.

Theorem 3.3.5 [11, Theorem 3.7]**.** *Finding the trace of a generator of the XTR group, using Algorithm 3.3.4 takes an expected number*
$$\frac{q}{q-1}\big(7.2\log_2(p)+8\log_2((p^2-p+1)/q)\big)$$
plus a small constant number of multiplications in $\mathrm{GF}(p)$.

Proof. According to Lemma 3.2.1, Algorithm 3.3.3 is called, on average, $\frac{3q}{q-1}$ times. For half the calls, on average, Δ in Step 2 is a quadratic non-residue, and the cost is a small constant number of multiplications in $\mathrm{GF}(p)$. For the other calls, first $\sqrt{\Delta}$ is computed at an expected cost of $\log_2(p)$ squarings and $0.5\log_2(p)$ multiplications in $\mathrm{GF}(p)$ (for a total of $1.3\log_2(p)$ multiplications in $\mathrm{GF}(p)$, cf. assumption in the proof of Lemma 2.2.1). This is followed by the computation of y at an expected cost of $\log_2(p)$ squarings and $0.5\log_2(p)$ multiplications in $\mathrm{GF}(p^2)$ (for a total of $3.5\log_2(p)$ multiplications in $\mathrm{GF}(p)$, cf. Lemma 2.2.1). Thus, for random c application of Algorithm 3.3.3 costs

$$\frac{1.3+3.5}{2}\log_2(p) = 2.4\log_2(p)$$

plus a small constant number of multiplications in $\mathrm{GF}(p)$, on average. □

Thus, Algorithm 3.3.4, based on Scipione del Ferro's method in Algorithm 3.3.1, is more than 50% faster than Algorithm 3.2.2. Though useful, Algorithm 3.3.4 and Theorem 3.3.5 are just a side result of a more important consequence of Algorithm 3.3.1, namely the key recovery method from Subsection 5.5. The next two sections contain two even faster methods to find $\mathrm{Tr}(g)$. The first method poses the additional restriction that $p \equiv 2$ or $5 \bmod 9$ (i.e., $p \not\equiv 8 \bmod 9$).

3.4. Subgroup selection when $p \not\equiv 8 \bmod 9$

If $p \not\equiv 8 \bmod 9$, then $(Z^9-1)/(Z^3-1) = Z^6+Z^3+1 \in \mathrm{GF}(p)[Z]$ is irreducible over $\mathrm{GF}(p)$, so that $\mathrm{GF}(p^6)$ can be represented as $\mathrm{GF}(p)(\zeta)$ where $\zeta^6+\zeta^3+1=0$. This representation of $\mathrm{GF}(p^6)$ allows symbolic calculation, i.e., irrespective of the value of p, of the trace of the $(p^6-1)/(p^2-p+1)^{\mathrm{th}}$ power of elements of the form $\zeta+a$, for random $a \in \mathrm{GF}(p)$. This follows from a more general argument due to H.W. Lenstra, Jr. In particular (cf. [11, Proposition 4.3]),

$$\mathrm{Tr}\left((\zeta+a)^{(p^6-1)/(p^2-p+1)}\right) = \frac{-3}{a^6-a^3+1}\left((a^2-1)^3\alpha + a^3(a^3-3a+1)\alpha^2\right),$$

where it is shown that $a^6-a^3+1 \neq 0$. It follows that, for any $a \in \mathrm{GF}(p)^*$, $a \neq \pm 1$, the right hand side expression is the trace of an element of order dividing p^2-p+1 (cf. [11, Corollary 4.4]). With $a=2$ and $a=1/2$, this leads to the following very fast method to initialize $\mathrm{Tr}(g)$—obviously any $a \in \mathrm{GF}(p)^*$, $a \neq \pm 1$, can be used instead.

Algorithm 3.4.1 (Computation of $\mathrm{Tr}(g)$ [11, Algorithm 4.5])**.**

1. Let $c = \frac{27\alpha+3\alpha^2}{19} \in \mathrm{GF}(p^2)$.
2. Apply Algorithm 2.3.5 to $n=(p^2-p+1)/q$ and c to compute $d = c_{(p^2-p+1)/q}$.
3. If $d \neq 3$, then let $\mathrm{Tr}(g) = d$ and return success.
4. Otherwise, if $d=3$, then replace c by $\frac{-27\alpha-24\alpha^2}{19} \in \mathrm{GF}(p^2)$.
5. Apply Algorithm 2.3.5 to $n=(p^2-p+1)/q$ and c and recompute $d = c_{(p^2-p+1)/q}$.
6. If $d \neq 3$, then let $\mathrm{Tr}(g) = d$ and return success.
7. Otherwise, if $d=3$, then return failure.

The probability of failure of Algorithm 3.4.1 may be expected to be q^{-2}, i.e., negligibly small. Its expected cost is about $8\log_2((p^2-p+1)/q)$ multiplications in $\mathrm{GF}(p)$.

Although Algorithm 3.4.1 is a very fast method to find a proper $\mathrm{Tr}(g)$, it is less general than the method from Subsection 3.3, and in particular does not provide a faster $F(c,X)$ irreducibility test. Because fast irreducibility testing has other applications than just $\mathrm{Tr}(g)$-initialization, improving the test from Subsection 3.3 is relevant. In Subsection 3.5 below this is done by reformulating the third-degree $\mathrm{GF}(p^2)$-irreducibility test as a third-degree $\mathrm{GF}(p)$-irreducibility test, and by carefully analyzing the cost of the latter.

3.5. Subgroup selection using a faster irreducibility test

Definition 3.5.1 [12, Definition 2.1]**.** For $c \in \mathrm{GF}(p^2)$ let

$$P(c,X) = X^3 + (c^p + c)X^2 + (c^{p+1} + c^p + c - 3)X + c^{2p} + c^2 + 2 - 2c^p - 2c.$$

It easily follows that $P(c,X)$ is a polynomial in $\mathrm{GF}(p)$. The following result indicates why it is relevant to consider $P(c,X)$.

Corollary 3.5.2 [12, Corollary 2.5]**.** *$F(c,X)$ is irreducible over $\mathrm{GF}(p^2)$ if and only if $P(c,X)$ is irreducible over $\mathrm{GF}(p)$.*

As shown in [12, Section 3], efficient application of Algorithm 3.3.1 to $P(c,X)$ requires an equivalent of Algorithm 2.3.5 for traces over $\mathrm{GF}(p)$. It is well known how that is done; the details are given in Algorithm 3.5.3 below.

Algorithm 3.5.3 [12, Algorithm 3.4]**.** To compute the trace $\mathrm{Tr}(y^n) \in \mathrm{GF}(p)$ over $\mathrm{GF}(p)$ of $y^n \in \mathrm{GF}(p^2)$, given an integer $n > 0$ and the trace $\mathrm{Tr}(y) \in \mathrm{GF}(p)$ over $\mathrm{GF}(p)$ of $y \in \mathrm{GF}(p^2)$ of order dividing $p+1$. This algorithm takes $1.8\log_2(p)$ multiplications in $\mathrm{GF}(p)$ (cf. assumption in the proof of Lemma 2.2.1).

1. Let $n = \sum_{i=0}^{k} n_i 2^i$ with $n_i \in \{0,1\}$ and $n_k \neq 0$ and let $v = \mathrm{Tr}(y) \in \mathrm{GF}(p)$.
2. Compute $w = (v^2 - 2) \in \mathrm{GF}(p)$.
3. For $i = k-1, k-2, \ldots, 0$ in succession, do the following.
   - If $n_i = 1$, then first replace v by $vw - \mathrm{Tr}(y)$ and next replace w by $w^2 - 2$.
   - If $n_i = 0$, then first replace w by $vw - \mathrm{Tr}(y)$ and next replace v by $v^2 - 2$.
4. Return $\mathrm{Tr}(y^n) = v$.

Algorithm 3.5.4 [12, Algorithm 3.5]**.** To test $P(c,X) = X^3 + p_2X^2 + p_1X + p_0 \in \mathrm{GF}(p)[X]$ for irreducibility over $\mathrm{GF}(p)$ with $p \neq 2,3$, do the following.

1. Compute
$$f_0 = \frac{27p_0 - 9p_2p_1 + 2p_2^3}{27},\ f_1 = p_1 - \frac{p_2^2}{3} \in \mathrm{GF}(p).$$
2. If $\Delta = f_0^2 + 4(\frac{f_1}{3})^3 \in \mathrm{GF}(p)$ is a quadratic residue in $\mathrm{GF}(p)$, then $P(c,X)$ is reducible (cf. [12, Lemma 3.2]).
   Comment. This step requires the computation of a Jacobi symbol.

3. Otherwise, compute $s = 2\frac{f_0^2+\Delta}{f_0^2-\Delta} \in \mathrm{GF}(p)$.
   Comment. According to [12, Lemma 3.3], s is the trace of r_1^{p-1} over $\mathrm{GF}(p)$, where $r_1 = \frac{-f_0+\sqrt{\Delta}}{2}$.
4. Apply Algorithm 3.5.3 to $n = \frac{p+1}{3}$ and $\mathrm{Tr}(y) = s$ to compute $\mathrm{Tr}(y^{(p+1)/3})$. If $\mathrm{Tr}(y^{(p+1)/3}) = 2$, then $P(c, X)$ is reducible.
   Comment. If the trace over $\mathrm{GF}(p)$ of $(r_1^{p-1})^{(p+1)/3}$ equals 2, then r_1 is a cube in $\mathrm{GF}(p^2)$ and thus, according to [12, Lemma 3.2], $P(c, X)$ is not irreducible.
5. Otherwise, Δ is a quadratic non-residue and r_1 is not a cube in $\mathrm{GF}(p^2)$ so that, according to [12, Lemma 3.2], $P(c, X)$ is irreducible over $\mathrm{GF}(p)$.

In the worst case Algorithm 3.5.4 costs $1.8\log_2(p)$ plus a small constant number of multiplications in $\mathrm{GF}(p)$. For half the random $c \in \mathrm{GF}(p^2)$, however, Δ is a quadratic residue, and the cost is just a small constant number of multiplications in $\mathrm{GF}(p)$. The proof of Theorem 3.5.5 below and its corollary follow.

Theorem 3.5.5 [12, Theorem 3.6]. *For $c \in \mathrm{GF}(p^2)$ the irreducibility of the polynomial $F(c, X) = X^3 - cX^2 + c^pX - 1$ over $\mathrm{GF}(p^2)$ can be tested at the cost of at most $m + 1.8\log_2(p)$ multiplications in $\mathrm{GF}(p)$, for some small constant m.*

Corollary 3.5.6 [12, Corollary 3.7]. *Finding the trace of a generator of the XTR group can be expected to take about*

$$\frac{q}{q-1}\big(2.7\log_2(p) + 8\log_2((p^2 - p + 1)/q)\big)$$

multiplications in $\mathrm{GF}(p)$ (*cf. Theorem* 3.3.5).

The result of Corollary 3.5.6 is only about $2.7\log_2(p)$ multiplications in $\mathrm{GF}(p)$ slower than Algorithm 3.4.1, but is more general since it applies to all $p \equiv 2 \bmod 3$ and not only to $p \equiv 2, 5 \bmod 9$.

Remark 3.5.7. Algorithm 3.5.4 provides an efficient way to test if $c \in \mathrm{GF}(p^2)$ is the trace of an element of the XTR supergroup. That is, by choosing the size of the XTR group close to the XTR supergroup, one obtains an efficient way to determine XTR group membership modulo a small error. In Algorithm 3.1.3 it is explained how such XTR parameters p and q can be found. In Section 6 it is shown that this has an application in the prevention of 'subgroup based attacks'.

4. XTR cryptographic schemes for confidentiality services

In any cryptosystem that relies on the (subgroup) discrete logarithm problem the ordinary representation of subgroup elements can be replaced by the XTR representation of subgroup elements of a multiplicative group of equivalent security. This section contains a description of some applications of XTR that provide confidentiality services: Diffie-Hellman key agreement in Subsection 4.1 and ElGamal encryption in Subsection 4.2. In both schemes random exponents modulo q are used. If q is chosen to be close to $p^2 - p + 1$ (cf. Algorithm 3.1.3), then much shorter random exponents, say of 170 bits, can be used instead if that is desirable for computational efficiency (cf. [23]).

4.1. XTR-DH

4.1.1. XTR-DH key agreement. Let p, q, $\mathrm{Tr}(g)$ be shared XTR public key data. If Alice and Bob want to agree on a secret key K they do the following.

1. Alice selects a random integer $a \in [2, q-3]$, applies Algorithm 2.3.5 to $n = a$ and $c = \mathrm{Tr}(g)$ to compute
$$S_a(\mathrm{Tr}(g)) = \left(\mathrm{Tr}(g^{a-1}), \mathrm{Tr}(g^a), \mathrm{Tr}(g^{a+1})\right) \in \mathrm{GF}(p^2)^3,$$
and sends $\mathrm{Tr}(g^a) \in \mathrm{GF}(p^2)$ to Bob.
2. Bob receives $\mathrm{Tr}(g^a)$ from Alice, selects a random integer $b \in [2, q-3]$, applies Algorithm 2.3.5 to $n = b$ and $c = \mathrm{Tr}(g)$ to compute
$$S_b(\mathrm{Tr}(g)) = \left(\mathrm{Tr}(g^{b-1}), \mathrm{Tr}(g^b), \mathrm{Tr}(g^{b+1})\right) \in \mathrm{GF}(p^2)^3,$$
and sends $\mathrm{Tr}(g^b) \in \mathrm{GF}(p^2)$ to Alice.
3. Alice receives $\mathrm{Tr}(g^b)$ from Bob, applies Algorithm 2.3.5 to $n = a$ and $c = \mathrm{Tr}(g^b)$ to compute
$$S_a(\mathrm{Tr}(g)^b) = \left(\mathrm{Tr}(g^{(a-1)b}), \mathrm{Tr}(g^{ab}), \mathrm{Tr}(g^{(a+1)b})\right) \in \mathrm{GF}(p^2)^3,$$
and determines K based on $\mathrm{Tr}(g^{ab}) \in \mathrm{GF}(p^2)$ (but see Remark 7.1.3).
4. Bob applies Algorithm 2.3.5 to $n = b$ and $c = \mathrm{Tr}(g^a)$ to compute
$$S_b(\mathrm{Tr}(g)^a) = \left(\mathrm{Tr}(g^{a(b-1)}), \mathrm{Tr}(g^{ab}), \mathrm{Tr}(g^{a(b+1)})\right) \in \mathrm{GF}(p^2)^3,$$
and determines K based on $\mathrm{Tr}(g^{ab}) \in \mathrm{GF}(p^2)$.
Comment. The 'neighboring' elements $\mathrm{Tr}(g^{(a-1)b})$ and $\mathrm{Tr}(g^{(a+1)b})$ computed by Alice are in general different from $\mathrm{Tr}(g^{a(b-1)})$ and $\mathrm{Tr}(g^{a(b+1)})$, the neighboring elements computed by Bob.

The communication and computational overhead of XTR-DH key agreement 4.1.1 are both about one third of traditional implementations of the Diffie-Hellman protocol that are based on subgroups of multiplicative groups of finite fields, and that achieve the same level of security (cf. Section 7.1).

4.2. XTR-ElGamal encryption

4.2.1. XTR-ElGamal encryption (cf. [6]). Let p, q, $\mathrm{Tr}(g)$ be XTR public key data, either owned (and made public) by Alice or shared by all parties. Furthermore, let $\mathrm{Tr}(g^k)$ be a value computed and made public by Alice, for some integer k selected (and kept secret) by Alice. Given $(p, q, \mathrm{Tr}(g), \mathrm{Tr}(g^k))$, Bob can encrypt a message M intended for Alice as follows.

1. Bob selects at random $b \in [2, q-3]$ and applies Algorithm 2.3.5 to $n = b$ and $c = \mathrm{Tr}(g)$ to compute
$$S_b(\mathrm{Tr}(g)) = \left(\mathrm{Tr}(g^{b-1}), \mathrm{Tr}(g^b), \mathrm{Tr}(g^{b+1})\right) \in \mathrm{GF}(p^2)^3.$$
2. Bob applies Algorithm 2.3.5 to $n = b$ and $c = \mathrm{Tr}(g^k)$ to compute
$$S_b(\mathrm{Tr}(g^k)) = \left(\mathrm{Tr}(g^{(b-1)k}), \mathrm{Tr}(g^{bk}), \mathrm{Tr}(g^{(b+1)k})\right) \in \mathrm{GF}(p^2)^3.$$
3. Bob determines a symmetric encryption key K based on $\mathrm{Tr}(g^{bk}) \in \mathrm{GF}(p^2)$.
4. Bob uses an agreed upon symmetric encryption method with key K to encrypt M, resulting in the encryption E.
5. Bob sends $(\mathrm{Tr}(g^b), E)$ to Alice.
Comment. The message sent by Bob consists of an 'overhead part' $\mathrm{Tr}(g^b)$ and a message part E. The length of the former is independent of the length of M, but the length of the latter depends strongly on the length of M and the type of symmetric encryption used.

4.2.2. XTR-ElGamal decryption. Using her knowledge of k, Alice decrypts the message $(\mathrm{Tr}(g^b), E)$ encrypted using XTR-ElGamal encryption 4.2.1 as follows.

1. Alice applies Algorithm 2.3.5 to $n = k$ and $c = \mathrm{Tr}(g^b)$ to compute
$$S_k(\mathrm{Tr}(g^b)) = (\mathrm{Tr}(g^{b(k-1)}), \mathrm{Tr}(g^{bk}), \mathrm{Tr}(g^{b(k+1)})) \in \mathrm{GF}(p^2)^3.$$
2. Alice determines symmetric encryption key K based on $\mathrm{Tr}(g^{bk}) \in \mathrm{GF}(p^2)$.
3. Alice uses the agreed upon symmetric encryption method with key K to decrypt E, resulting in the encryption M.

The communication and computational overhead of XTR-based ElGamal encryption 4.2.1 and decryption 4.2.2 (with communication overhead as explained in Step 5 of 4.2.1) are both about one third of traditional implementations of the ElGamal encryption and decryption protocols that are based on subgroups of multiplicative groups of finite fields, and that achieve the same level of security (cf. Section 7.1).

Remark 4.2.3. The type of encryption described in 4.2.1 is commonly referred to as 'hybrid encryption', because the key K is used in conjunction with an agreed upon symmetric key encryption method. In the non-hybrid version the message is restricted to the key space and 'encrypted' using an invertible operation that takes place in the key space, such as multiplication by the key. In 4.2.1 and with $K = \mathrm{Tr}(g^{bk})$ this would amount to requiring that $M \in \mathrm{GF}(p^2)$ and computing E as $K \cdot M \in \mathrm{GF}(p^2)$. Compared to non-hybrid traditional ElGamal encryption, non-hybrid XTR-ElGamal encryption saves a factor three on the length of both parts of the encrypted message, for messages that fit in the key space (of one third of the 'traditional' size).

Remark 4.2.4. As is customary it is implicitly assumed in the decryption that the first component of an ElGamal encrypted message represents a conjugate of a power of g. In some situations this should be verified explicitly. A value, say c, can be checked by verifying that $c \in \mathrm{GF}(p^2) \setminus \mathrm{GF}(p)$ (implying, in particular, that $c \neq 3$), by applying Algorithm 2.3.5 to $n = q$ and c to compute $S_q(c) = (c_{q-1}, c_q, c_{q+1})$, and by verifying that $c_q = 3$. Other and more efficient techniques are discussed in Section 6.

5. XTR cryptographic schemes for non-repudiation services

5.1. Introduction

In this section two XTR applications are described that provide non-repudiation services: Nyberg-Rueppel message recovery digital signatures in Subsection 5.3 and XTR-DSA in Subsection 5.4. Both schemes require computation of the product of two powers of g. For the standard representation this can easily be done using well known multi-exponentiation techniques in substantially less time than required for two separate exponentiations. But if traces are used it is a relatively complicated operation. In Subsection 5.2 below it is described how this computation may be carried out in common cryptographic applications such as the ones in Subsections 5.3 and 5.4.

5.1.1. XTR public key data for signature verification. As in Subsection 4.2, Alice's XTR public key data for digital signatures consist of p, q, $\mathrm{Tr}(g)$, and $\mathrm{Tr}(g^k)$ for a secret integer k that is known only to Alice. However, in addition it is assumed that not only $\mathrm{Tr}(g^k)$ but also $\mathrm{Tr}(g^{k-1})$ and $\mathrm{Tr}(g^{k+1})$ (and thus $S_k(\mathrm{Tr}(g))$) are available to the verifier. These additional $\mathrm{GF}(p^2)$ elements are either part of the public key, or they are reconstructed by the verifier. As shown in Subsection 5.5, $\mathrm{Tr}(g^{k-1})$ (or $\mathrm{Tr}(g^{k+1})$) can be reconstructed from p, q, $\mathrm{Tr}(g)$, $\mathrm{Tr}(g^k)$, and $\mathrm{Tr}(g^{k+1})$ (or $\mathrm{Tr}(g^{k-1})$) using an explicit and easily computed formula. Reconstruction of $\mathrm{Tr}(g^{k+1})$ (or $\mathrm{Tr}(g^{k-1})$) given just $(p, q, \mathrm{Tr}(g), \mathrm{Tr}(g^k))$ requires additional assumptions and a slightly more involved computation (cf. Subsection 5.5).

5.2. Computing the trace of a product

Let $\mathrm{Tr}(g) \in \mathrm{GF}(p^2)$ and $S_k(\mathrm{Tr}(g)) \in \mathrm{GF}(p^2)^3$ be given for some secret integer k with $0 < k < q$. In Algorithm 5.2.7 below it is shown that $\mathrm{Tr}(g^a \cdot g^{bk})$ can be computed efficiently for any $a, b \in \mathbf{Z}$ given $\mathrm{Tr}(g)$ and $S_k(\mathrm{Tr}(g)$, i.e., without knowing k.

Definition 5.2.1 [10, Definition 2.4.1]**.** Let $C(V)$ denote the center column of a 3×3 matrix V and let

$$A(c) = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & -c^p \\ 0 & 1 & c \end{pmatrix} \text{ and } M_n(c) = \begin{pmatrix} c_{n-2} & c_{n-1} & c_n \\ c_{n-1} & c_n & c_{n+1} \\ c_n & c_{n+1} & c_{n+2} \end{pmatrix}$$

be 3×3-matrices over $\mathrm{GF}(p^2)$ with c and c_n as in Definition 2.3.1.

Lemma 5.2.2 [10, Lemma 2.4.2]**.** *For $n, m \in \mathbf{Z}$*

$$S_n(c) = S_m(c) \cdot A(c)^{n-m} \text{ and } M_n(c) = M_m(c) \cdot A(c)^{n-m}.$$

Corollary 5.2.3 [10, Corollary 2.4.3]**.**

$$c_n = S_m(c) \cdot C(A(c)^{n-m}).$$

Lemma 5.2.4 [10, Lemma 2.4.4]**.** *The determinant of $M_0(c)$ equals*

$$D = c^{2p+2} + 18c^{p+1} - 4(c^{3p} + c^3) - 27 \in \mathrm{GF}(p).$$

If $D \neq 0$ then

$$M_0(c)^{-1} = \frac{1}{D} \cdot \begin{pmatrix} 2c^2 - 6c^p & 2c^{2p} + 3c - c^{p+2} & c^{p+1} - 9 \\ 2c^{2p} + 3c - c^{p+2} & (c^2 - 2c^p)^{p+1} - 9 & (2c^{2p} + 3c - c^{p+2})^p \\ c^{p+1} - 9 & (2c^{2p} + 3c - c^{p+2})^p & (2c^2 - 6c^p)^p \end{pmatrix}.$$

Lemma 5.2.5 [10, Lemma 2.4.5]**.**

$$\det(M_0(\mathrm{Tr}(g))) = (\mathrm{Tr}(g^{p+1})^p - \mathrm{Tr}(g^{p+1}))^2 \neq 0.$$

Combination of these results leads to the following corollary.

Corollary 5.2.6 [10, Lemma 2.4.6, Corollary 2.4.7]**.** *Given $\mathrm{Tr}(g)$ and $S_n(\mathrm{Tr}(g))$,*

$$C(A(\mathrm{Tr}(g))^n) = M_0(\mathrm{Tr}(g))^{-1} \cdot (S_n(\mathrm{Tr}(g)))^T$$

can be computed at the cost of a small constant number of operations in $\mathrm{GF}(p^2)$.

Algorithm 5.2.7 (Computation of $\mathrm{Tr}(g^a \cdot g^{bk})$ [10, Algorithm 2.4.8])**.**
Let $\mathrm{Tr}(g)$, $S_k(\mathrm{Tr}(g))$ (for unknown k), and $a, b \in \mathbf{Z}$ with $0 < a, b < q$ be given.

1. Compute $e = a/b \bmod q$.
2. Apply Algorithm 2.3.5 to $n = e$ and $c = \mathrm{Tr}(g)$ to compute $S_e(\mathrm{Tr}(g))$.
3. Use Corollary 5.2.6 to compute $C(A(\mathrm{Tr}(g))^e)$ based on $\mathrm{Tr}(g)$ and $S_e(\mathrm{Tr}(g))$.
4. Use Corollary 5.2.3 to compute $\mathrm{Tr}(g^{e+k}) = S_k(\mathrm{Tr}(g)) \cdot C(A(\mathrm{Tr}(g))^e)$.
5. Apply Algorithm 2.3.5 to $n = b$ and $c = \mathrm{Tr}(g^{e+k})$ to compute $S_b(\mathrm{Tr}(g^{e+k}))$.
6. Return $\mathrm{Tr}(g^{(e+k)b}) = \mathrm{Tr}(g^a \cdot g^{bk})$.

Theorem 5.2.8 [10, Theorem 2.4.9]**.** *Given $M_0(\mathrm{Tr}(g))^{-1}$, $\mathrm{Tr}(g)$, and*

$$S_k(\mathrm{Tr}(g)) = (\mathrm{Tr}(g^{k-1}), \mathrm{Tr}(g^k), \mathrm{Tr}(g^{k+1}))$$

the trace $\mathrm{Tr}(g^a \cdot g^{bk})$ of $g^a \cdot g^{bk}$ can be computed at a cost of

$$8 \log_2(a/b \bmod q) + 8 \log_2(b) + 34$$

multiplications in $\mathrm{GF}(p)$.

Remark 5.2.9. Assuming that $M_0(\mathrm{Tr}(g))^{-1}$ is computed once and for all (at the cost of a small constant number of operations in $\mathrm{GF}(p^2)$), it follows that $\mathrm{Tr}(g^a \cdot g^{bk})$

can be computed at a cost of $16\log_2(q)+34$ multiplications in $\mathrm{GF}(p)$. This is still substantially faster than traditional computation of $g^a \cdot g^{bk}$ in $\mathrm{GF}(p^6)$ using multi-exponentiation, but the speed-up factor drops from 3 (for a single exponentiation) to about 1.75.

5.3. XTR-Nyberg-Rueppel signatures

This subsection contains a description of the XTR version of the Nyberg-Rueppel (NR) message recovery signature scheme. XTR can in a similar way be used in other 'ElGamal-like' signature schemes as illustrated in Subsection 5.4 below.

5.3.1. XTR-NR signature generation. To sign a message M containing an agreed upon type of redundancy using the XTR version of the NR protocol, Alice does the following:

1. Alice selects a random integer $u \in [2, q-3]$, and applies Algorithm 2.3.5 to $n = u$ and $c = \mathrm{Tr}(g)$ to compute
$$S_u(\mathrm{Tr}(g)) = (\mathrm{Tr}(g^{u-1}), \mathrm{Tr}(g^u), \mathrm{Tr}(g^{u+1})) \in \mathrm{GF}(p^2)^3.$$
2. Alice determines a symmetric encryption key K based on $\mathrm{Tr}(g^u) \in \mathrm{GF}(p^2)$.
3. Alice uses an agreed upon symmetric encryption method with key K to encrypt M, resulting in the encryption E.
4. Alice computes the (integer valued) hash h of E.
5. Alice computes $s = (k \cdot h + u) \bmod q \in \{0, 1, \ldots, q-1\}$.
6. Alice's resulting signature on M is (E, s).

5.3.2. XTR-NR signature verification. It is assumed that Alice's XTR public key is as described in 5.1.1 and thus contains $S_k(\mathrm{Tr}(g))$. To verify Alice's signature (E, s) and to recover the signed message M, verifier Bob does the following.

1. Bob checks that $0 \le s < q$; if not failure.
2. Bob computes the hash h of E.
3. Bob replaces h by $-h \bmod q \in \{0, 1, \ldots, q-1\}$.
4. Bob applies Algorithm 5.2.7 to $\mathrm{Tr}(g)$, $S_k(\mathrm{Tr}(g))$ (with k unknown to Bob), $a = s$, and $b = h$ to compute $\mathrm{Tr}(g^s \cdot g^{hk})$ (which equals $\mathrm{Tr}(g^u)$).
5. Bob determines a symmetric encryption key K based on $\mathrm{Tr}(g^s \cdot g^{hk}) \in \mathrm{GF}(p^2)$.
6. Bob uses the agreed upon symmetric encryption method with key K to decrypt E resulting in M.
7. The signature is accepted if and only if M contains the agreed upon redundancy.

XTR-NR signature generation 5.3.1 and verification 5.3.2 are both considerably faster than traditional implementations of the NR scheme that are based on subgroups of multiplicative groups of finite fields of the same security level: XTR-NR signature generation 5.3.1 is about three times faster than traditional NR signature generation, and XTR-NR signature verification 5.3.2 is about 1.75 faster than the traditional method (cf. Remark 5.2.9). The length of the signature is identical to other variants of the hybrid version of the NR scheme (cf. Remark 4.2.3): an overhead part of length depending on the desired security (i.e., the subgroup size) and a message part of length depending on the message itself and the agreed upon redundancy and symmetric encryption.

5.4. XTR-DSA signatures

This subsection contains a description of the XTR version of the DSA signature scheme. As in the original standard, [21], we assume that the size of q is 160 bits, the same size as the SHA-1 secure hash [20].

5.4.1. XTR-DSA signature generation. To sign a message M using the XTR version of DSA, Alice does the following:

1. Alice selects a random integer $u \in [2, q-3]$.
2. Alice applies Algorithm 2.3.5 to $n = u$ and $c = \mathrm{Tr}(g)$ to compute
$$S_u(\mathrm{Tr}(g)) = \big(\mathrm{Tr}(g^{u-1}), \mathrm{Tr}(g^u), \mathrm{Tr}(g^{u+1})\big) \in \mathrm{GF}(p^2)^3.$$
3. Alice writes $\mathrm{Tr}(g^u) = x_1\alpha + x_2\alpha^2$ and computes $r = (x_1 + p \cdot x_2) \bmod q$. If $r = 0$, then Alice goes back to Step 1.
4. Alice computes $u^{-1} \bmod q$.
5. Alice computes the hash h of M.
   Comment. The secure hash function SHA-1 (cf. [20]) is used in the Digital Signature Standard (cf. [21]).
6. Alice computes $s = u^{-1}(h + k \cdot r) \bmod q$. If $s = 0$, then Alice goes back to Step 1.
7. Alice's resulting signature on M is (r, s).

5.4.2. XTR-DSA signature verification. It is assumed that Alice's XTR public key is as described in 5.1.1 and thus contains $S_k(\mathrm{Tr}(g))$. To verify Alice's signature (r, s) on message M, verifier Bob does the following.

1. Bob checks that $0 < r, s < q$; if not failure.
2. Bob computes $w = s^{-1} \bmod q$.
3. Bob computes the hash h of M.
4. Bob computes $u_1 = w \cdot h \bmod q$ and $u_2 = r \cdot w \bmod q$.
5. Bob applies Algorithm 5.2.7 to $\mathrm{Tr}(g)$, $S_k(\mathrm{Tr}(g))$ (with k unknown to Bob), $a = u_1$, and $b = u_2$ to compute $v_0 = \mathrm{Tr}(g^{u_1} \cdot g^{k \cdot u_2})$ (which equals $\mathrm{Tr}(g^u)$).
6. Bob writes $v_0 = z_1\alpha + z_2\alpha^2$ and computes $v = (z_1 + p \cdot z_2) \bmod q$.
7. Bob accepts the signature if and only if $v = r$.

Remark 5.4.3. Note that if (r,s) is a valid signature on message M, then so are $(r, s\cdot p^2 \bmod q)$ and $(r, s\cdot p^4 \bmod q)$. This is similar to the property of the ECC-DSA signature scheme (cf. [1]) that if (r,s) is a valid signature, then so is $(r, -s \bmod q)$. In applications where this may cause problems, uniqueness can be achieved by selecting the signature for which $s\cdot p^{2i} \bmod q$, with $i = 0,1,2$, is minimal. Of course, the verification should then check this as well.

5.5. Key size reduction

In this subsection it is shown that $\mathrm{Tr}(g^{k+1})$ and $\mathrm{Tr}(g^{k-1})$ can be derived from $\mathrm{Tr}(g)$ and $\mathrm{Tr}(g^k)$, if the private key k is properly chosen. Throughout this section let $c = \mathrm{Tr}(g)$ and $c_n = \mathrm{Tr}(g^n)$ for $n \in \mathbf{Z}$. First of all, c_{k-1} (or c_{k+1}) can be computed from c, c_k, and c_{k+1} (or c_{k-1}) in a small constant number of operations in $\mathrm{GF}(p)$. This follows from Theorem 5.5.1 and Algorithm 5.5.2 below.

Theorem 5.5.1 [11, Theorem 5.1].
1. *If $k \neq p, 1-p \bmod p^2-p+1$ then $c^p c_{k-1} - c c_k \neq 0$ and*

$$c_{k+1} = \frac{c_k^p(c^2-3c^p) - c_{k-1}^p(c^{2p}-3c) - c_{k-1}^2 c + c_k^2(c^p - c^2) + c_k c_{k-1} c^{p+1}}{c^p c_{k-1} - c c_k}.$$

2. *If $k \neq -p, p-1 \bmod p^2-p+1$ then $c c_{k+1} - c^p c_k \neq 0$ and*

$$c_{k-1} = \frac{c_k^p(c^{2p}-3c) - c_{k+1}^p(c^2-3c^p) - c_{k+1}^2 c^p + c_k^2(c - c^{2p}) + c_k c_{k+1} c^{p+1}}{c c_{k+1} - c^p c_k}.$$

Algorithm 5.5.2 (Inversion in $\mathrm{GF}(p^2)$ [11, Algorithm 5.2]).
Let $x = x_1\alpha + x_2\alpha^2 \in \mathrm{GF}(p^2)$. Compute $t = (x_1x_2 + (x_1-x_2)^2)^{-1} \in \mathrm{GF}(p)$, then $1/x = t(x_2\alpha + x_1\alpha^2) \in \mathrm{GF}(p^2)$.

It follows that under a mild restriction on the private key k, if c and c_k are part of the public key, then either c_{k+1} or c_{k-1} suffices to compute c_{k-1} or c_{k+1}, in order to reconstruct the value $S_k(\mathrm{Tr}(g))$ required for signature verification. It is not hard to see that neither c_{k+1} nor c_{k-1} must in principle be part of the public key: they can both be computed by multiplying or dividing the roots of $F(c,X)$ and $F(c_k,X)$, leading to 3 possible representations c_{k+1} (and c_{k-1}). Two bits in the public key suffice to indicate which of the representations is the correct one. In general, finding the roots of $F(c,X)$ and $F(c_k,X)$ requires a more substantial computation than is acceptable for the recipient of the public key. If $p \not\equiv 8 \bmod 9$, then this idea can be made to work much more efficiently, however, and this can be done in such a way that the two additional bits are not even required. This is shown in the remainder of this subsection. The description focuses on reconstructing c_{k+1} from c and c_k, but works in a very similar way for c_{k-1}.

Roughly speaking, Algorithm 3.3.1 is used to compute explicit representations of g and g^k in $\mathrm{GF}(p^6) = \mathrm{GF}(p)[X]/(X^6+X^3+1)$ (cf. Section 3.3) based on their representations c and c_k, respectively. The value of c_{k+1} then follows as the trace over $\mathrm{GF}(p^2)$ of $g \cdot g^k \in \mathrm{GF}(p^6)$, where k is chosen in such a way that c_{k+1} is 'minimal' in some sense. How this is done is shown first, in 5.5.3 below, after which the reconstruction of c_{k+1} is described in 5.5.4.

5.5.3. Selecting k. After selecting the private key k, its owner applies Algorithm 2.3.5 to $n = k$ and $c = \mathrm{Tr}(g)$ to compute $S_k(\mathrm{Tr}(g)) = (c_{k-1}, c_k, c_{k+1})$. Because g^k, g^{kp^2}, and g^{kp^4} are conjugates, the same c_k is obtained for $n = k$, $n = kp^2 \bmod q$, and $n = kp^4 \bmod q$. The side result c_{k+1} obtained from the computation of c_k, however, is in general not the same for $n = k$, $n = kp^2 \bmod q$, and $n = kp^4 \bmod q$, simply because $\mathrm{Tr}(g^{k+1})$, $\mathrm{Tr}(g^{kp^2+1})$, and $\mathrm{Tr}(g^{kp^4+1})$ are, in general, not the same. Namely, unless $k = 0 \bmod q$, the elements g^{k+1}, g^{kp^2+1}, and g^{kp^4+1} are not conjugates over $\mathrm{GF}(p^2)$, despite the fact that g^k, g^{kp^2}, and g^{kp^4} are conjugates over $\mathrm{GF}(p^2)$.

It follows that for any pair (c, c_k) there are in principle three different possible values for c_{k+1}: one that corresponds to the selected private key k, and two that correspond to the related but 'wrong' values $kp^2 \bmod q$ and $kp^4 \bmod q$. This ambiguity, which will have to be resolved by any method to recover c_{k+1} from (c, c_k), is avoided in the following simple manner: the owner of k computes all three values $\mathrm{Tr}(g^{k+1})$, $\mathrm{Tr}(g^{kp^2+1})$, and $\mathrm{Tr}(g^{kp^4+1})$ for the k of his choice, and next replaces k by k, $kp^2 \bmod q$, or $kp^4 \bmod q$ depending on which of $\mathrm{Tr}(g^{k+1})$, $\mathrm{Tr}(g^{kp^2+1})$, and $\mathrm{Tr}(g^{kp^4+1})$ is the 'smallest'. As a consequence, c_{k+1} is the 'smallest' possibility given the pair (c, c_k). The security is not affected by changing the initially selected k in this way. It remains to define what is meant by 'smallest' and how $\mathrm{Tr}(g^{k+1})$, $\mathrm{Tr}(g^{kp^2+1})$, and $\mathrm{Tr}(g^{kp^4+1})$ are computed.

For $x \in \mathrm{GF}(p)$ let $\pi_0(x) \in \{0, 1, \ldots, p-1\}$ be the image of x under the 'natural' bijection between $\mathrm{GF}(p)$ and $\{0, 1, \ldots, p-1\}$. For $x = x_1\alpha + x_2\alpha^2 \in \mathrm{GF}(p^2)$ let $\pi(x) = \pi_0(x_1) + p \cdot \pi_0(x_2)$ (cf. Subsection 2.2) be a bijection from $\mathrm{GF}(p^2)$ to $\{0, 1, \ldots, p^2-1\}$. The mapping π induces an ordering on $\mathrm{GF}(p^2)$ and 'smallest' is defined as smallest with respect to this ordering.

To compute $\mathrm{Tr}(g^{k+1})$, $\mathrm{Tr}(g^{kp^2+1})$, and $\mathrm{Tr}(g^{kp^4+1})$ the owner of k can simply apply Algorithm 2.3.5 three times, namely with $n = k$, $n = kp^2 \bmod q$, and $n = kp^4 \bmod q$, at a total cost of about $24\log_2(q)$ multiplications in $\mathrm{GF}(p)$ (cf. Theorem 2.3.6). A conceptually more complicated method that saves about $8\log_2(q)$ multiplications in $\mathrm{GF}(p)$ is as follows. Compute (c_{k-1}, c_k, c_{k+1}) and $(c_{-p-1}, c_{-p}, c_{-p+1})$, at the cost of $16\log_2(q)$ multiplications in $\mathrm{GF}(p)$, followed by $c_{k\pm2}$ and $c_2 = c^2 - 2c^p$ (cf. Corollary 2.3.3.(ii) and (i)). Use these values to compute $\mathrm{Tr}(g^{kp^2+1})$ by observing that

$$\mathrm{Tr}(g^{kp^2+1}) = \mathrm{Tr}(g^{kp^2-p^3}) = \mathrm{Tr}(g^{(k-p)p^2}) = \mathrm{Tr}(g^{k-p})$$

and

$$\begin{pmatrix} \mathrm{Tr}(g^{k-p-1}) \\ \mathrm{Tr}(g^{k-p}) \\ \mathrm{Tr}(g^{k-p+1}) \end{pmatrix}^T = \begin{pmatrix} c_{-p-1} \\ c_{-p} \\ c_{-p+1} \end{pmatrix}^T \begin{pmatrix} c_2^p & c^p & 3 \\ c^p & 3 & c \\ 3 & c & c_2 \end{pmatrix}^{-1} \begin{pmatrix} c_{k-2} & c_{k-1} & c_k \\ c_{k-1} & c_k & c_{k+1} \\ c_k & c_{k+1} & c_{k+2} \end{pmatrix}$$

(cf. Lemmas 5.2.2 and 5.2.5). This takes a small constant number of multiplications in $\mathrm{GF}(p)$. With

$$\mathrm{Tr}(g^{kp^2-1}) = \mathrm{Tr}(g^{k+p})$$

a similar matrix identity involving (c_{p-1}, c_p, c_{p+1}) (obtained using $c_{-n} = c_n^p$, cf. Lemma 2.3.2(ii)) is used to compute $\mathrm{Tr}(g^{kp^2-1})$. The same method is then used to compute $\mathrm{Tr}(g^{kp^4+1})$ based on $(\mathrm{Tr}(g^{kp^2-1}), \mathrm{Tr}(g^{kp^2}), \mathrm{Tr}(g^{kp^2+1}))$ (where $\mathrm{Tr}(g^{kp^2}) = c_k$) and $(c_{-p-1}, c_{-p}, c_{-p+1})$.

5.5.4. Reconstructing c_{k+1}. Given (c, c_k) with k chosen as explained in 5.5.3, the correct (i.e., the 'smallest') c_{k+1} is reconstructed by means of repeated application of Algorithm 3.3.1. To get Algorithm 3.3.1 to work for this specific application, two auxiliary algorithms are needed.

Algorithm 5.5.5 (Exponentiation in $\mathrm{GF}(p^2)$ [11, Algorithm 5.3]**.** To compute $x^e \in \mathrm{GF}(p^2)$ given $x \in \mathrm{GF}(p^2)$ and positive integer e, do the following.

1. Let $e_0, e_1 \in \{0, 1, \ldots, p-1\}$ be such that $e_0 + e_1 p = e \bmod (p^2-1)$ and let $e_i = \sum_j e_{ij} 2^j$, with $e_{ij} \in \{0,1\}$ for $i = 0,1$ and $j \geq 0$, be the binary representations of e_0 and e_1.
2. Let n be the largest index such that $e_{in} \neq 0$ for $i = 0$ or 1.
3. Compute $x^{p+1} = x \cdot x^p \in \mathrm{GF}(p)$.
4. Let $y = 1$ in $\mathrm{GF}(p^2)$. For $j = n, n-1, \ldots, 0$ in succession do the following:
   - if $e_{0j} = 1$ and $e_{ij} = 1$, then replace y by $y \cdot x^{p+1}$;
   - if $e_{0j} = 1$ and $e_{1j} = 0$, then replace y by $y \cdot x$;
   - if $e_{0j} = 0$ and $e_{1j} = 1$, then replace y by $y \cdot x^p$;
   - if $j > 0$, then replace y by y^2.

   Comment. Note that this is similar to multi-exponentiation.
5. Return $y = x^e \in \mathrm{GF}(p^2)$.

Lemma 5.5.6 [11, Lemma 5.4]**.** *The expected cost of Algorithm* 5.5.5 *is* $4\log_2(p)$ *multiplications in* $\mathrm{GF}(p)$.

Algorithm 5.5.7 (Cube root in $\mathrm{GF}(p^2)$ if $p \not\equiv 8 \bmod 9$ [11, Algorithm 5.5])**.** To compute a cube root in $\mathrm{GF}(p^6)$ of $r \in \mathrm{GF}(p^2)$ perform the following steps.

1. If $p \equiv 2 \bmod 9$, then let $e = \frac{8p^2-5}{9}$, otherwise, if $p \equiv 2 \bmod 9$ then let $e = \frac{p^2+2}{9}$.
2. Apply Algorithm 5.5.5 to $x = r$ and e to compute $t = r^e \in \mathrm{GF}(p^2)$.
3. Compute $s = t^3 \in \mathrm{GF}(p^2)$ and determine $j = 0, 1$ or 2 such that $\alpha^j s = r$.
4. Return a cube root $\zeta^j t \in \mathrm{GF}(p^6)$ of r, where ζ is as in Subsection 3.3.

   Comment. The result is in $\mathrm{GF}(p^2)$ if $j = 0$.

Algorithm 5.5.8 (Key recovery [11, Algorithm 5.6])**.** To compute the 'smallest' c_{k+1} corresponding to (c, c_k), do the following.

1. Apply Algorithm 3.3.1 to $f(X) = F(c, X)$ to compute a single root $g \in \mathrm{GF}(p^6) = \mathrm{GF}(p)(\zeta)$ of $F(c, X)$.

Comment. The representation of elements of $\mathrm{GF}(p^6)$ is explained in Subsection 3.3. In Step 3 of Algorithm 3.3.1 a cube root is computed using Algorithm 5.5.7. As a result, u is a $\mathrm{GF}(p^2)$-multiple of a power of ζ, so that Algorithm 5.5.2 can be used for the division by u in the same step.

2. Apply Algorithm 3.3.1 to $f(X) = F(c_k, X)$ to compute all three roots $y_1, y_2, y_3 \in \mathrm{GF}(p^6) = \mathrm{GF}(p)(\zeta)$ of $F(c_k, X)$
   Comment. Same comments as above. Furthermore, $w = \alpha$ in Step 4 of Algorithm 3.3.1.
3. For $i = 1, 2, 3$ compute the trace t_i over $\mathrm{GF}(p^2)$ of $gy_i \in \mathrm{GF}(p^6)$.
   Comment. The trace over $\mathrm{GF}(p^2)$ of $\sum_{j=0}^{5} a_j\zeta^j \in \mathrm{GF}(p^6)$ equals
   $$3(a_3 - a_0)\alpha - 3a_0\alpha^2 \in \mathrm{GF}(p^2)$$
   (cf. [11, Lemma 4.1]).
4. Let c_{k+1} be the 'smallest' of t_1, t_2, and t_3, under the ordering induced by π as in 5.5.3.

Theorem 5.5.9 [11, Theorem 5.7]. *Algorithm* 5.5.8 *can be expected to require* $10.6\log_2(p)$ *multiplications in* $\mathrm{GF}(p)$.

5.5.10. Key recovery summary. It follows from 5.5.3 and 5.5.4 that $\mathrm{Tr}(g^{k-1})$ and $\mathrm{Tr}(g^{k+1})$ do not have to be included in the XTR public key data for digital signature or authentication applications, as long as

1. the private key k is selected as explained in 5.5.3 above,
2. p, q, $\mathrm{Tr}(g)$, and $\mathrm{Tr}(g^k)$ are included in the public key,
3. the recipient of the public key is willing and able to perform Algorithm 5.5.8 to compute $\mathrm{Tr}(g^{k+1})$ followed by an application of Theorem 5.5.1 to compute $\mathrm{Tr}(g^{k-1})$.

Actually, there are three options for XTR public keys used for digital signatures or authentication: include one, two, or all three of the values $\mathrm{Tr}(g^{k-1})$, $\mathrm{Tr}(g^k)$, $\mathrm{Tr}(g^{k+1})$. In some applications, e.g. issuance of a certificate by a Certificate Authority, it may be required that the relative correctness of these components can be verified by a third party. A method to do this is described in Remark 6.2.10 below.

6. Correctness verification of XTR data formats

6.1. Subgroup based attacks against XTR

Security of cryptographic protocols may be endangered if elements of a certain group are exchanged but group membership is not properly verified by the recipient. Examples of such attacks can be found in [2, 4, 12, 14]. In XTR, subgroup attacks refer to attacks that take advantage of the omission to verify membership of the XTR (sub)group (cf. Subsection 2.1). As argued in [12], subgroup attacks can be rendered mostly ineffective if either

1. p and q are such that $(p^2 - p + 1)/q$ is small (cf. Algorithm 3.1.3), or
2. p and q are such that $(p^2 - p + 1)/q$ is a small multiple of a prime of the same order of magnitude as q (cf. Algorithm 3.1.4).

In either case, however, membership of the XTR supergroup, i.e., the order p^2-p+1 group containing the XTR group (cf. Subsection 2.1), still must be verified. In Subsection 6.2 below it is shown how that can be done. Note that if $(p^2-p+1)/q$ is small, short exponents may be used in the XTR versions of cryptographic protocols to maintain efficiency, as mentioned in the introduction to Section 4 (cf. [23]).

6.2. Prevention of subgroup attacks against XTR

Let G denote the order q XTR group and H the order p^2-p+1 XTR supergroup, as defined in Subsection 2.1. This section contains efficient methods to determine if an element $d \in \mathrm{GF}(p^2) \setminus \mathrm{GF}(p)$ is the trace of an element of H. As explained in Subsection 6.1 such methods are useful to prevent XTR subgroup attacks.

It follows from Lemma 2.3.2.(v) that it suffices to check that $F(d, X)$ is irreducible. According to Theorem 3.5.5 this can be done at the cost of $1.8 \log_2(p)$ plus a small constant number of multiplications in $\mathrm{GF}(p)$. Thus, checking membership of H can be done at the cost of a small overhead compared to the cost of the regular XTR cryptographic operations. As shown in the remainder of this section, the overhead can be reduced to just a small constant number of operations in $\mathrm{GF}(p)$, at the cost however of a small amount of additional communication: if d, the element to be checked, equals $\mathrm{Tr}(h)$ for $h \in H$, and $\mathrm{Tr}(h \cdot g)$ is sent along with d, then the fact that d is indeed the trace of an element of H can be verified in a small constant number of operations in $\mathrm{GF}(p)$ (cf. Corollary 6.2.8). Here it is assumed that the trace $\mathrm{Tr}(g)$ of an element $g \in H$ is known.

Definition 6.2.1 [12, Definition 5.1 and Lemma 5.2]**.** For third-degree monic polynomials

$$R(X) = \prod_{i=0}^{2}(X - \alpha_i) \in \mathrm{GF}(p^2)[X] \text{ and } S(X) = \prod_{j=0}^{2}(X - \beta_j) \in \mathrm{GF}(p^2)[X]$$

with $\alpha_i, \beta_j \in \mathrm{GF}(p^6)^*$ for $0 \leq i, j < 3$, the *root-product* $\Re(R, S)$ is the ninth-degree polynomial

$$\Re(R, S) = \prod_{i,j=0}^{2} (X - \alpha_i \beta_j) \in \mathrm{GF}(p^2)[X]$$

with non-zero constant term.

Lemma 6.2.2 [12, Lemma 5.3]**.** *With $R(X)$ and $S(X)$ as in Definition* 6.2.1*,*

$$\Re(R, S) = (\beta_0 \cdot \beta_1 \cdot \beta_2)^3 R(X \cdot \beta_0^{-1}) \cdot R(X \cdot \beta_1^{-1}) \cdot R(X \cdot \beta_2^{-1}),$$

and if $S(X)$ *is irreducible over* $\mathrm{GF}(p^2)$ *then*

$$\Re(R,S) = \beta_0^{3(p^4+p^2+1)} R(X \cdot \beta_0^{-1}) \cdot R(X \cdot \beta_0^{-p^2}) \cdot R(X \cdot \beta_0^{-p^4}).$$

In the application of Lemma 6.2.2 the polynomial $F(c,X)$ plays the role of $S(X)$, where $c = \mathrm{Tr}(g)$ for some $g \in H$, so that $F(c,X) = S(X)$ is irreducible (cf. Lemma 2.3.2.(v)). This implies that $\mathrm{GF}(p^6)$ can be represented as $\mathrm{GF}(p^2)(g)$, and

$$\Re(R, F(c,X)) = R(X/g) \cdot R(X/g^{p-1}) \cdot R(X/g^{-p})$$

(since $g^{3(p^4+p^2+1)} = 1$, $g^{p^2} = g^{p-1}$, and $g^{p^4} = g^{-p}$) can be computed using a constant number of operations in $\mathrm{GF}(p^6) = \mathrm{GF}(p^2)(g)$, if a representation of $g^p \in \mathrm{GF}(p^2)(g)$ is known. The following result shows how such a representation can be obtained. As before, $\mathrm{Tr}(g^i)$ is abbreviated to c_i.

Proposition 6.2.3 [12, Proposition 5.4 and Corollary 5.5]**.** *Let* $c = \mathrm{Tr}(g)$ *for some* $g \in H$ *and let* $c_{p-2} = \mathrm{Tr}(g^{p-2}) \in \mathrm{GF}(p^2)$ *be given. Then* $g^p = Kg^2 + Lg + M \in \mathrm{GF}(p^2)(g)$ *can be computed in a small constant number of multiplications in* $\mathrm{GF}(p)$ *using*

$$\begin{pmatrix} M \\ L \\ K \end{pmatrix} = \begin{pmatrix} c_{-2} & c_{-1} & c_0 \\ c_{-1} & c_0 & c_1 \\ c_0 & c_1 & c_2 \end{pmatrix}^{-1} \cdot \begin{pmatrix} c_{p-2} \\ c \\ c^p \end{pmatrix},$$

where the inverse of the matrix on the right hand side exists (*cf. Lemma* 5.2.5) *and is given in Lemma* 5.2.4.

Theorem 6.2.4 [12, Theorem 5.6]**.** *Let* $R(X) \in \mathrm{GF}(p^2)[X]$ *be a monic third-degree polynomial with non-zero constant term and let* $c = \mathrm{Tr}(g)$ *for some element* $g \in H$. *Given* $\mathrm{Tr}(g^{p-2}) \in \mathrm{GF}(p^2)$, *the root-product* $\Re(R(X), F(c,X))$ *can be computed at the cost of a small constant number of operations in* $\mathrm{GF}(p)$.

Proof. As argued above, this follows from Lemma 6.2.2 and Proposition 6.2.3. □

The value c_{p-2} plays an important role, so it could be precomputed and stored, independent of the value d to be checked. Note that $c_{p+1} = c \cdot c_p - c^p \cdot c_{p-1} + c_{p-2}$, $c_p = c^p$, and $c_{p-1} = c$, so that $c_{p-2} = c_{p+1}$. The following results shows that c_{p-2} can quickly be recovered from a single bit.

Proposition 6.2.5 [12, Proposition 5.7]**.** *Let* $c = \mathrm{Tr}(g)$ *for some element* $g \in H$. *Then* $\mathrm{Tr}(g^{p-2}) = c_{p-2}$ *can be computed at the cost of a square-root computation in* $\mathrm{GF}(p^2)$, *assuming one bit of information to resolve the square-root ambiguity.*

Proof. With $c_{p-2} = x_1\alpha + x_2\alpha^2$ it simply follows that

$$(c_{p-2} - c_{p-2}^p)^2 = -3(x_1 - x_2)^2.$$

Combination with $c_{p-2} = c_{p+1}$, the identity for $(c_{p+1} - c_{p+1}^p)^2$ given in Lemmas 5.2.4 and 5.2.5, leads to

$$-3(x_1 - x_2)^2 = c^{2p+2} + 18c^{p+1} - 4(c^{3p} + c^3) - 27 \in \mathrm{GF}(p).$$

On the other hand $c_{p-2} + c_{p-2}^p = -(x_1 + x_2)$. With $c_{p-2} = g^{p-2} + g^{(p-2)p^2} + g^{(p-2)p^4} = g^{p-2} + g^{-2p+1} + g^{p+1}$, it follows that $c_{p-2}^p = g^{-p-1} + g^{-p+2} + g^{2p-1}$. Now,

$$\begin{aligned} c^{p+1} &= c \cdot c^p = (g + g^{p-1} + g^{-p})(g^p + g^{-1} + g^{-p+1}) \\ &= g^{p+1} + g^{p-2} + g^{-2p+1} + g^{-p-1} + g^{-p+2} + g^{2p-1} + 3 \\ &= c_{p-2} + c_{p-2}^p + 3. \end{aligned}$$

Thus, $x_1 + x_2 = 3 - c^{p+1} \in \mathrm{GF}(p)$. Combining the identities involving $x_1 - x_2$ and $x_1 + x_2$ it follows that c_{p-2} and its conjugate over $\mathrm{GF}(p)$ can be computed at the cost of a square-root calculation in $\mathrm{GF}(p^2)$. To distinguish $c_{p-2} = x_1\alpha + x_2\alpha^2$ from its conjugate $x_2\alpha + x_1\alpha^2$ over $\mathrm{GF}(p)$ a single bit that is on if and only if $x_1 > x_2$ suffices. □

Algorithm 6.2.6 [12, Proof of Lemma 5.8]**.** Let $c = \mathrm{Tr}(g)$ for some element $g \in H$ and let $\mathrm{Tr}(g^{p-2}) = c_{p-2}$ be given. Given $d, d' \in \mathrm{GF}(p^2)$, to check if there exists an element $h \in H$ such that $d = \mathrm{Tr}(h)$ and $d' = \mathrm{Tr}(h \cdot g)$, do the following.

1. If $F(d, \alpha^i) = 0$ for either $i = 0$, 1, or 2, then the statement is not true.
2. Otherwise, if $F(d, \alpha^i) \neq 0$ for $i = 0, 1, 2$, then compute the root-product $\Re(F(d, X), F(c, X))$.
3. If $\Re(F(d, X), F(c, X))$ is divisible by $F(d', X)$, then the statement is true, otherwise it is false.

Lemma 6.2.7 [12, Lemma 5.8]**.** *Algorithm* 6.2.6 *takes a small constant number of operations in* $\mathrm{GF}(p)$.

Corollary 6.2.8 [12, Corollary 5.9]**.** *Let* $c = \mathrm{Tr}(g)$ *for some element* $g \in H$ *and let* $\mathrm{Tr}(g^{p-2})$ *be given. Given the trace values of an alleged element* $h \in H$ *and its 'successor'* $g \cdot h$, *it takes a small constant number of operations in* $\mathrm{GF}(p)$ *to verify that indeed* h *in* H.

Corollary 6.2.9 [12, Corollary 5.10]**.** *Let* $c = \mathrm{Tr}(g)$ *where* g *is known to be a generator of the XTR group, let* d *be the trace of an element that is known to be in the XTR group* $\langle g \rangle$, *and let* d' *be some element of* $\mathrm{GF}(p^2)$. *Then it can efficiently be verified if* d *and* d' *are of the form* $\mathrm{Tr}(g^x)$ *and* $\mathrm{Tr}(g^{x+1})$, *respectively, for some integer* x, $0 < x < q$.

Remark 6.2.10. An XTR public key meant for digital signatures takes the form (p, q, c, d, d'), where p and q are primes satisfying the usual XTR conditions, $c = \mathrm{Tr}(g)$ for a generator g of the XTR group, $d = \mathrm{Tr}(g^k)$ for a secret key k, and $d' = \mathrm{Tr}(g^{k+1})$ (cf. Section 5). Corollary 6.2.9 implies that a Certificate Authority can efficiently verify the consistency of an XTR signature public key presented by a client, before issuing a certificate on it. More specifically, if a client provides a Certificate Authority with XTR public key data (p, q, c, d, d'), then the Certificate Authority checks that these data satisfy the conditions given above, using the following two step approach. First the Certificate Authority checks that p and q are well-formed and that $c, d \in \mathrm{GF}(p^2) \setminus \mathrm{GF}(p)$ are indeed traces of elements of the XTR group by verifying that $c_q = d_q = 3$ using standard XTR arithmetic (cf. Algorithm 2.3.5). Secondly, the Certificate Authority uses Corollary 6.2.9 to verify that d and d' are traces of consecutive powers of the generator corresponding to c. Note that the Certificate Authority does not obtain information about the secret key k.

7. Security of XTR

7.1. Security of the trace representation

In the XTR versions of 'subgroup discrete logarithm' based cryptographic protocols the subgroup elements are replaced by their traces. This implies that the security is no longer based on the regular and well known subgroup Discrete Logarithm (DL), Diffie-Hellman (DH), or Decision Diffie-Hellman (DDH) problems, but on their XTR counterparts. The *XTR-DH* problem is the problem of computing $\mathrm{Tr}(g^{xy})$ given $\mathrm{Tr}(g^x)$ and $\mathrm{Tr}(g^y)$. Given $\mathrm{Tr}(g^x)$ and $\mathrm{Tr}(g^y)$, the XTR-Diffie-Hellman value $\mathrm{Tr}(g^{xy})$ is denoted by $XDH(\mathrm{Tr}(g^x), \mathrm{Tr}(g^y))$. The *XTR-DDH* problem is the problem of determining whether $XDH(a,b) = c$ given $a, b, c \in \mathrm{Tr}(\langle g \rangle)$. The *XTR-DL* problem is to find $0 \le x < q$ such that $a = \mathrm{Tr}(g^x)$ given $a \in \mathrm{Tr}(\langle g \rangle)$. Note that if x satisfies $a = \mathrm{Tr}(g^x)$, then so do $x \cdot p^2 \bmod q$ and $x \cdot p^4 \bmod q$.

Theorem 7.1.1 [10, Theorem 5.2.1]**.** *The following equivalences hold*:

1. *The XTR-DL problem is* $(1,1)$*-equivalent to the DL problem in* $\langle g \rangle$*,*
2. *The XTR-DH problem is* $(1,2)$*-equivalent to the DH problem in* $\langle g \rangle$*,*
3. *The XTR-DDH problem is* $(3,2)$*-equivalent to the DDH problem in* $\langle g \rangle$*,*

where $\mathcal{A}$ *is* (a,b)-equivalent *to* $\mathcal{B}$*, if any instance of* $\mathcal{A}$ (*or* $\mathcal{B}$) *can be solved by at most* a (*or* b) *calls to an algorithm solving* $\mathcal{B}$ (*or* $\mathcal{A}$).

Remark 7.1.2. An algorithm solving DL, DH, or DDH with non-negligible probability can be transformed in an algorithm solving the corresponding XTR problem with non-negligible probability, and vice versa (cf. [10, Proof of Theorem 5.2.1]).

Despite the fact that, according to Theorem 7.1.1, XTR-DH is $(1,2)$-equivalent to ordinary DH, in many practical situations a single call to an XTR-DH solving algorithm would suffice to solve a DH problem. An example is DH key agreement where the resulting key is actually used after it has been established.

Remark 7.1.3. Theorem 7.1.1.2 states that determining the (small) XTR-DH key is as hard as determining the whole DH key in the representation group $\langle g \rangle$. From the results in [27] it actually follows that determining the image of the XTR-DH key under any non-trivial $\mathrm{GF}(p)$-linear function is also as hard as the whole DH key. This means that, for example, finding the coefficient of α or α^2 of the XTR-DH key is as hard as finding the whole DH key, implying that cryptographic applications may be based on just one of the coefficients. Note that in 4.1.1 both coefficients are used.

7.2. Relation between the XTR group and supersingular elliptic curves

The number of points over $\mathrm{GF}(p^2)$ (including the point at infinity) on an elliptic curve over $\mathrm{GF}(p^2)$ takes the form p^2-t+1 for some integer $-2p \leq t \leq 2p$. It is well known that there exist so-called supersingular elliptic curves over $\mathrm{GF}(p^2)$ where this order is equal to p^2-p+1 and that there exist efficiently computable (i.e., in polynomial time and space in the input length), injective homomorphisms based on the Weil pairing, of such curves onto the XTR supergroup. Such homomorphisms are known as MOV embeddings. See for instance [15] for further reference.

At the Crypto 2000 rump session (cf. [16]) it was suggested that the inverses of these homomorphisms might be efficiently computable too, and it was mentioned that this would imply that the XTR (sub)group is just an instance of the (sub)group of a supersingular elliptic curve. Thus, an attack affecting elliptic curve cryptosystems would affect XTR-based cryptosystems as well, implying that the security of XTR cryptosystems would not be not better than that of elliptic curve cryptosystems.

More precisely, the suggestion made at the Crypto 2000 rump session can be formulated as the following assumption:

X2C One can efficiently find a supersingular elliptic curve over $\mathrm{GF}(p^2)$, such that the group of points C over $\mathrm{GF}(p^2)$ (including the point at infinity) is of order p^2-p+1 and an efficiently computable, injective homomorphism from the XTR subgroup into C.

A similar problem is posed by N. Koblitz in [8, Remark on page 328]. It was shown in [28], however, that the suggested assumption is most likely false, because it would contradict several generally accepted hardness assumptions. The following is one of the results shown in [28].

Theorem 7.2.1. *Under the* **X2C** *assumption, the following problems are efficiently computable*:

1. *The Diffie-Hellman problem in the XTR subgroup.*
2. *The Diffie-Hellman problem in the group of points of order q on a supersingular elliptic curve over $\mathrm{GF}(p^2)$ of order p^2-p+1.*

This result gives evidence that the security provided by the XTR subgroup is better than that provided by the isomorphic group on supersingular elliptic curves. Additional evidence is provided by the fact that the Decision Diffie-Hellman problem is efficiently computable in the latter group, while this problem is believed to be hard in the XTR subgroup. See [28].

Note added in proof. The XTR exponentiation methods as presented in this paper can be improved considerably. The basis for the improvement is an alternative version of the XTR double exponentiation method from Algorithm 5.2.7 that is not only twice as fast on average, but that also does not require the matrix methods as introduced in Subsection 5.2. This improved version of Algorithm 5.2.7

can be used to formulate a new XTR single exponentiation method, improving Algorithm 2.3.5 of the present paper by more than 16%, or by more than 50% if a small one time precomputation is performed. As a result XTR signature applications become 50% faster and all other applications become significantly faster as well. For the details refer to [26].

References

[1] American National Standards Institute (ANSI) X9.62-1998, Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA). 1998.

[2] Biehl, I., Meyer, B., Müller, V., Differential fault attacks on elliptic curve cryptosystems. In: Advances in Cryptology – CRYPTO 2000 (ed. by M. Bellare; Lecture Notes in Comput. Sci. 1880), 131–146. Springer, Berlin 2000.

[3] Brouwer, A.E., Pellikaan, R., Verheul, E.R., Doing more with fewer bits. In: Advances in Cryptology – ASIACRYPT '99 (ed. by K.Y. Lam, E. Okamoto, C. Xing; Lecture Notes in Comput. Sci. 1716), 321–332. Springer, Berlin 1999.

[4] Burmester, M.V.D., A remark on the efficiency of identification schemes. In: Advances in Cryptology – EUROCRYPT '90 (ed. by I.B. Damgård; Lecture Notes in Comput. Sci. 473), 493–495. Springer, Berlin 1991.

[5] Cohen, H., Miyaji, A., Ono, T., Efficient elliptic curve exponentiation using mixed coordinates. In: Advances in Cryptology – ASIACRYPT '98 (ed. by K. Ohta and D.Y. Pei; Lecture Notes in Comput. Sci. 1514), 51–65. Springer, Berlin 1998.

[6] ElGamal, T., A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans. Inform. Theory 31 (1985), 469–472.

[7] Gong, G., Harn, L., Public key cryptosystems based on cubic finite field extensions. IEEE Trans. Inform. Theory 45 (1999), 2601–2605.

[8] Koblitz, N., An elliptic curve implementation of the finite field digital signature algorithm. In: Advances in Cryptology – CRYPTO '98 (ed. by H. Krawczyk; Lecture Notes in Comput. Sci. 1462), 327–337. Springer, Berlin 1998.

[9] Lenstra, A.K., Using cyclotomic polynomials to construct efficient discrete logarithm cryptosystems over finite fields. In: Information Security and Privacy (ACISP '97) (ed. by V. Varadharajan, J. Pieprzyk, Y. Mu; Lecture Notes in Comput. Sci. 1270), 127–138. Springer, Berlin 1997.

[10] Lenstra, A.K., Verheul, E.R., The XTR public key system. In: Advances in Cryptology – CRYPTO 2000 (ed. by M. Bellare; Lecture Notes in Comput. Sci. 1880), 1–19. Springer, Berlin 2000; available from `www.ecstr.com`.

[11] —, —, Key improvements to XTR. In: Advances in Cryptology – ASIACRYPT 2000 (ed. by T. Okamoto; Lecture Notes in Comput. Sci. 1976), 220–233. Springer, Berlin 2000; available from `www.ecstr.com`.

[12] —, —, Fast irreducibility and subgroup membership testing in XTR. In: Public Key Cryptography (PKC 2001, Cheju Island, Korea) (ed. by K. Kim; Lecture Notes in Comput. Sci. 1992). Springer, Berlin 2001; available from `www.ecstr.com`.

[13] Lidl, R., Müller, W.B., Permutation polynomials in RSA-cryptosystems. In: Advances in Cryptology (CRYPTO '83) (ed. by D. Chaum), 293–301. Plenum Press, New York 1984.

[14] Lim, C.H., Lee, P.J., A key recovery attack on discrete log-based schemes using a prime order subgroup. In: Advances in Cryptology – CRYPTO '97 (ed. by B.S. Kaliski; Lecture Notes in Comput. Sci. 1294), 249–263. Springer, Berlin 1997.

[15] Menezes, A., Elliptic Curve Public Key Cryptosystems. Kluwer Academic Publishers, Boston 1993.

[16] Menezes, A., Vanstone, S., ECSTR (XTR): Elliptic Curve Singular Trace Representation. Rump session of CRYPTO 2000.

[17] Müller, W.B., Polynomial functions in modern cryptology. In: Contributions to General Algebra 3 (Vienna, 1984) (ed. by G. Eigenthaler et al.), 7–32. Hölder-Pichler-Tempsky, Vienna 1985.

[18] Müller, W.B., Nöbauer, R., Cryptanalysis of the Dickson-scheme. In: Advances in Cryptology – EUROCRYPT '85 (ed. by F. Pichler; Lecture Notes in Comput. Sci. 219), 50–61. Springer, Berlin 1986.

[19] Nicholson, W.K., Introduction to Abstract Algebra. PWS–Kent Publishing Company, Boston 1993.

[20] National Inst. of Standards and Technology, Secure Hash Standard (SHA-1). Federal Information Processing Standards Publication 180–1, 1995; available from `www.itl.nist.gov/fipspubs/fib180-1.htm`.

[21] —, Digital Signature Standard. Federal Information Processing Standards Publication 186, 1994.

[22] Nöbauer, R., Cryptanalysis of the Rédei scheme. In: Contributions to General Algebra 3 (Vienna, 1984) (ed. by G. Eigenthaler et al.), 255–264. Hölder-Pichler-Tempsky, Vienna 1985.

[23] van Oorschot, P.C.., Wiener, M.J., On Diffie-Hellman key agreement with short exponents. In: Advances in Cryptology – EUROCRYPT '96 (ed. by U. Maurer; Lecture Notes in Comput. Sci. 1070), 332–343. Springer, Berlin 1996.

[24] Schnorr, C.P., Efficient signature generation by smart cards. J. Cryptology 4 (1991), 161–174.

[25] Smith, P., Skinner, C., A public-key cryptosystem and a digital signature system based on the Lucas function analogue to discrete logarithms. In: Advances in Cryptology – ASIACRYPT '94 (ed. by J. Pieprzyk, R. Safavi-Naini; Lecture Notes in Comput. Sci. 917), 357–364. Springer, Berlin 1995.

[26] Stam, M., Lenstra, A.K., Speeding up XTR, submitted for publication; available from `www.ecstr.com`.

[27] Verheul, E., Certificates of recoverability with scalable recovery agent security. In: Public Key Cryptography (PKC 2000, Melbourne) (ed. by H. Imai, Y. Zhang; Lecture Notes in Comput. Sci. 1751), 258–275. Springer, Berlin 2000.

[28] —, Evidence that XTR is more secure than supersingular elliptic curve cryptosystems. In: Advances in Cryptology – EUROCRYPT 2001 (ed. by B. Pfitzmann; Lecture Notes in Comput. Sci. 2045), 195–210. Springer, Berlin 2001.

Arjen K. Lenstra
Citibank, N.A.
and Technische Universiteit Eindhoven
1 North Gate Road
Mendham, NJ 07945-3104, U.S.A.
`arjen.lenstra@citicorp.com`

Eric R. Verheul
PricewaterhouseCoopers
GRMS Crypto Group
Goudsbloemstraat 14
5644 KE Eindhoven, The Netherlands
`eric.verheul@[nl.pwcglobal.com,pobox.com]`