Practical Cryptography in High Dimensional Tori

Marten van Dijk[1], Robert Granger[2], Dan Page[2], Karl Rubin[3, ⋆], Alice Silverberg[3, ⋆⋆], Martijn Stam[2], and David Woodruff[1, ⋆⋆⋆]

[1] MIT CSAIL, {marten, dpwood}@mit.edu
[2] Department of Computer Science, University of Bristol, {granger, page, stam}@cs.bris.ac.uk
[3] Department of Mathematics, UC Irvine, {krubin, asilverb}@uci.edu

Abstract. At Crypto 2004, van Dijk and Woodruff introduced a new way of using the algebraic tori T_n in cryptography, and obtained an asymptotically optimal $n/\phi(n)$ savings in bandwidth and storage for a number of cryptographic applications. However, the computational requirements of compression and decompression in their scheme were impractical, and it was left open to reduce them to a practical level. We give a new method that compresses orders of magnitude faster than the original, while also speeding up the decompression and improving on the compression factor (by a constant term). Further, we give the first efficient implementation that uses T_{30}, compare its performance to XTR, CEILIDH, and ECC, and present new applications. Our methods achieve better compression than XTR and CEILIDH for the compression of as few as two group elements. This allows us to apply our results to ElGamal encryption with a small message domain to obtain ciphertexts that are 10% smaller than in previous schemes.

Keywords: torus-based cryptography, discrete-log based cryptography

1 Introduction

When Diffie and Hellman introduced their key agreement scheme in a finite field of prime order, they made the assumption that a birthday attack was the best one can do (the Pohlig-Hellman algorithm [20] was in submission). Hence it made sense to use a subgroup of size about that of the field itself. Since then, the discrete logarithm problem in the multiplicative group of a finite field has been studied with increased interest. For prime order subgroups the best known attacks today are the birthday attack in the subgroup itself and the Number Field Sieve in the full finite field. It is now common practice to use a subgroup whose cardinality is substantially smaller than the field size. This raises the

⋆ Rubin was supported by NSF grant DMS-0140378.
⋆⋆ Silverberg was supported by NSA grant MDA904-03-1-0033.
⋆⋆⋆ Woodruff was supported by an NDSEG fellowship.

question whether it is possible to efficiently represent elements in this subgroup with fewer bits than generic elements of the full field, thus providing compression.

Brouwer, Pellikaan and Verheul [4] showed that compression can be achieved by going up to an extension field. They conjectured that one can attain a compression ratio of $n/\phi(n)$, where n is the degree of the extension. For $n = 2$, the LUC cryptosystem [14] already achieved these savings. For $n = 6$, Brouwer et al. described a system that was later improved upon by Lenstra and Verheul [15, 16], resulting in the XTR public key cryptosystem.

Rubin and Silverberg [23] recast the problem of compression for extension fields in terms of algebraic tori. They showed that if the algebraic torus T_n is rational, the conjectured compression factor $n/\phi(n)$ can in fact be achieved. If n is the product of at most two prime powers then T_n is known to be rational [30, 12]. Based on the rationality of T_6, Rubin and Silverberg [23] developed the CEILIDH public key cryptosystem.

Although T_n is not known to be rational in general, van Dijk and Woodruff [6] show that one can obtain key agreement, signature and encryption schemes with a compression factor asymptotically $n/\phi(n)$ as the number of keys, signatures, or messages grows, *without* relying on the rationality of T_n. This helps explain the potential of, and increasing interest in, torus-based cryptography.

The torus T_n is rational if there are efficiently computable "almost bijections"[4] between $T_n(\mathbb{F}_q)$ and $\mathbb{F}_q^{\phi(n)}$, where ϕ is Euler's totient function. Though the tori T_n in general are only conjectured to be rational, it is known [30] that they are always *stably rational*, i.e., for every n there is an m such that there is an "almost bijection" between $T_n(\mathbb{F}_q) \times \mathbb{F}_q^m$ and $\mathbb{F}_q^{\phi(n)+m}$.

Using the fact that T_n is stably rational, van Dijk and Woodruff [6] developed bijections between $T_n(\mathbb{F}_q) \times \mathbb{F}_q^m$ and $\mathbb{F}_q^{\phi(n)+m}$ with $m = \sum_{d|n,\ \mu(n/d)=-1} d$, where μ is the Möbius function, leading to asymptotically optimal $n/\phi(n)$ savings in bandwidth and storage. However, a major drawback of their solution is its large computational requirements.

The present work gives a new and efficient construction of bijections between $T_n(\mathbb{F}_q) \times \mathbb{F}_q^m$ and $\mathbb{F}_q^{\phi(n)+m}$ with significantly smaller m than in [6], as well as an optimised implementation when $n = 30$. The latter uses some known techniques [9] to efficiently implement CEILIDH.

Note that $n = 30 = 2 \cdot 3 \cdot 5$ is the next cryptographically interesting case, since its compression is up to 20% better than that of systems based on $n = 6$. In addition to our computational savings, in this case we are able to reduce the original affine surplus $m = 32$ [6] to $m = 2$. As we show, this reduction has immediate practical implications.

Since we are interested in the practicality of our construction, we perform timings for exponentiations, compression and decompression for both the new $T_{30}(\mathbb{F}_q)$ system and for a CEILIDH-based $T_6(\mathbb{F}_{q_L})$ system with $q_L \approx q^5$. For an equivalent of 1024-bit RSA security, the computational costs of the operations

[4] The maps may be undefined on a small number of points.

in both systems are comparable, while the compression of our scheme is better by a factor of $5/4 = (30/\phi(30))/(6/\phi(6))$.

Observe that the security of ECC systems is based on a different mathematical problem and (still) has the advantage that the best known attack is via a birthday paradox argument. So, one works in a subgroup that is almost the same size as the elliptic curve itself. Consequently, compression is much less of an issue there. Yet the same ECC systems provide an additional reason to study the compression methods we do, since the rise of pairing-based cryptography leads to elements in the torus (although at present no efficient curves with embedding degree 30 are known). Note that compression for elliptic curves over extension fields has also been considered [22].

Outline: §2 discusses some tools we use. In §3 we present the new mapping. §4 gives cryptographic applications. In §5 we show how to implement our mapping, and in §6 we present simulation results.

2 Preliminaries

In this section we provide some background on the algebraic tori used in cryptography. See [23, 24] for more details. Let $\mathbb{F}_q$ denote the finite field with q elements, let ϕ be Euler's totient function, and let $\Phi_n(x)$ be the n-th cyclotomic polynomial. Write $G_{q,n}$ for the subgroup of $\mathbb{F}_{q^n}^\times$ of order $\Phi_n(q)$. Let $\mathbb{A}^n$ denote n-dimensional affine space. Recall that the Möbius function $\mu(m)$ is 0 if m is not square-free, and is $(-1)^k$ if m is a product of k distinct primes.

2.1 Algebraic Tori

For any positive integer n one can define an algebraic torus T_n over $\mathbb{F}_q$ whose $\mathbb{F}_q$-points consist of the elements of $\mathbb{F}_{q^n}^\times$ whose norm is one down to every proper subfield of $\mathbb{F}_{q^n}/\mathbb{F}_q$. The following provides some useful properties of T_n (see Lemma 7 of [23] and Lemma 1 of [2]).

Lemma 1. *1. $T_n(\mathbb{F}_q) \cong G_{q,n}$, and thus $\#T_n(\mathbb{F}_q) = \Phi_n(q)$.*
2. If $h \in T_n(\mathbb{F}_q)$ has prime order not dividing n, then $h \notin \mathbb{F}_{q^d}$ for any $d \mid n$ with $d < n$.

The variety T_n has dimension $\phi(n)$. Since $T_n(\mathbb{F}_q)$ embeds into $\mathbb{F}_{q^n}^\times$, one can perform the group operation as ordinary multiplication in the field, or use other more efficient possibilities [9]. The subgroup $T_n(\mathbb{F}_q)$ may be regarded as the "primitive" subgroup of $\mathbb{F}_{q^n}^\times$, since by Lemma 1 its elements do not lie in a proper subfield of $\mathbb{F}_{q^n}$. Thus, $T_n(\mathbb{F}_q)$ is believed to be the most cryptographically secure subgroup of $\mathbb{F}_{q^n}^\times$.

2.2 Rationality of Tori over $\mathbb{F}_q$

The interpretation of $G_{q,n}$ as an algebraic torus is motivated by the possibility, for some n, to exploit birational maps between T_n and affine space [23].

Definition 1. *The torus T_n is* **rational** *if there is a birational map $\rho : T_n(\mathbb{F}_q) \to \mathbb{A}^{\phi(n)}$.*

That is, $T_n(\mathbb{F}_q)$ is rational if there are Zariski open subsets $W \subset T_n$ and $U \subset \mathbb{A}^{\phi(n)}$, and rational functions $\rho_1, \ldots, \rho_{\phi(n)} \in \mathbb{F}_q(x_1, \ldots, x_n)$ and $\psi_1, \ldots, \psi_n \in \mathbb{F}_q(y_1, \ldots, y_{\phi(n)})$, such that $\rho = (\rho_1, \ldots, \rho_{\phi(n)}) : W \to U$ and $\psi = (\psi_1, \ldots, \psi_n) : U \to W$ are inverse isomorphisms (as quasi-projective varieties).

The existence of a birational map allows one to represent elements of $T_n(\mathbb{F}_q)$ with just $\phi(n)$ elements of $\mathbb{F}_q$, providing an effective compression factor of $n/\phi(n)$ over the embedding into $\mathbb{F}_{q^n}$. The torus T_n is known to be rational when n is either a prime power or a product of two prime powers [30, 12], and is conjectured to be rational for all n [30].

2.3 CEILIDH

The torus-based public key system CEILIDH was introduced at Crypto 2003 by Rubin and Silverberg [23]. The system is based on the rational torus T_6, and achieves a compression factor of three. They construct an efficiently computable bijection

$$\psi : \mathbb{A}^2(\mathbb{F}_q) \setminus V(f) \to T_6(\mathbb{F}_q) \setminus \{1, a\},$$

together with an efficiently computable inverse

$$\rho : T_6(\mathbb{F}_q) \setminus \{1, a\} \to \mathbb{A}^2(\mathbb{F}_q) \setminus V(f),$$

where $V(f)$ denotes a small subvariety of $\mathbb{A}^2$, and 1 and a in $T_6(\mathbb{F}_q)$ are points excluded by ρ and ψ. This allows one to represent all (bar two) elements of $T_6(\mathbb{F}_q)$ with just two elements of $\mathbb{F}_q$. This system attains the same compression factor as the public key system XTR [15, 29]. Granger, Page and Stam [9] give a comparison of XTR and CEILIDH in the case $q \equiv 2 \bmod 9$ or $q \equiv 5 \bmod 9$.

2.4 Asymptotically Optimal Torus-Based Cryptography

Since T_n is known to be rational only for special values of n, the above ideas do not lead to an optimal compression factor of $n/\phi(n)$ in general. Van Dijk and Woodruff [6] overcome this problem in the case where several elements of T_n are to be compressed. They construct a bijection:

$$\theta : T_n(\mathbb{F}_q) \times (\times_{d|n,\mu(n/d)=-1} \mathbb{F}_{q^d}^\times) \to \times_{d|n,\mu(n/d)=1} \mathbb{F}_{q^d}^\times. \tag{1}$$

Specializing their map to the case $n = 30$ gives

$$T_{30}(\mathbb{F}_q) \times \mathbb{F}_q^\times \times \mathbb{F}_{q^6}^\times \times \mathbb{F}_{q^{10}}^\times \times \mathbb{F}_{q^{15}}^\times \to \mathbb{F}_{q^2}^\times \times \mathbb{F}_{q^3}^\times \times \mathbb{F}_{q^5}^\times \times \mathbb{F}_{q^{30}}^\times,$$

which can be reinterpreted as an "almost bijection" (see [6])

$$T_{30}(\mathbb{F}_q) \times \mathbb{A}^{32}(\mathbb{F}_q) \to \mathbb{A}^{40}(\mathbb{F}_q).$$

One can use this map to achieve an asymptotic compression factor of 30/8. Indeed, to compress m elements of $T_{30}(\mathbb{F}_q)$, one can compress an element x and split its image into $y_1 \in \mathbb{A}^8(\mathbb{F}_q)$ and $y_2 \in \mathbb{A}^{32}(\mathbb{F}_q)$. Then y_1 forms the affine input of the next compression. In the end, $8m+32$ elements of $\mathbb{F}_q$ are used to represent m elements of $T_{30}(\mathbb{F}_q)$. Observe that their map comes from the equation

$$\Phi_{30}(x)(x-1)(x^6-1)(x^{10}-1)(x^{15}-1) = (x^2-1)(x^3-1)(x^5-1)(x^{30}-1), \quad (2)$$

relating the orders of all the different component groups of domain and range. Since these groups are cyclic, one can map to and from their products as long as the orders of the component groups are coprime. For the map above there are some small primes that occur in the order of several component groups, but van Dijk and Woodruff are able to isolate and handle them separately.

3 The New Construction

The bijection (1), while asymptotically optimal, leaves open the question of whether one can obtain better compression for a *fixed* number of elements. Our new compression map, given by (4) below (see Theorems 2 and 4), has this property. Using the fact that $\Phi_n(x) = \prod_{d|n}(x^d-1)^{\mu(n/d)}$, we have

Proposition 1 *If p is a prime, and a is a positive integer not divisible by p, then*

$$\Phi_{ap}(x)\Phi_a(x) = \Phi_a(x^p).$$

The following result can be deduced from Proposition 1 above, using Lemma 6 of [6] (see also pp. 60–61 of [30]). Here, Res denotes the Weil restriction of scalars (see for example [30] or [24]).

Theorem 2 *If p is a prime, q is a prime power, a is a positive integer, qa is not divisible by p, and $\gcd(\Phi_{ap}(q), \Phi_a(q)) = 1$, then*

$$T_{ap}(\mathbb{F}_q) \times T_a(\mathbb{F}_q) \cong (\mathrm{Res}_{\mathbb{F}_{q^p}/\mathbb{F}_q} T_a)(\mathbb{F}_q) \cong T_a(\mathbb{F}_{q^p}).$$

The next result follows from Proposition 1, by doing double induction on the number of prime divisors of n and the number of prime divisors of m.

Theorem 3 *If n is square-free and m is a divisor of n, then*

$$\Phi_n(x) \prod_{d|\frac{n}{m},\, \mu(\frac{n}{md})=-1} \Phi_m(x^d) = \prod_{d|\frac{n}{m},\, \mu(\frac{n}{md})=1} \Phi_m(x^d).$$

The next result follows from Theorem 3, using the ideas in the proof of Theorem 3 of [6].

Theorem 4 *If n is square-free and m is a divisor of n, then there is an efficiently computable bijection (with an efficiently computable inverse)*

$$T_n(\mathbb{F}_q) \times \prod_{d|\frac{n}{m},\, \mu(\frac{n}{md})=-1} T_m(\mathbb{F}_{q^d}) \to \prod_{d|\frac{n}{m},\, \mu(\frac{n}{md})=1} T_m(\mathbb{F}_{q^d}).$$

Note that [6] is based on the case $m = 1$ of Theorem 4. Theorem 4 is most useful to us when T_m is rational. If T_m is rational, then Theorem 4 gives efficiently computable "almost bijections" between T_m and $\mathbb{A}^{\phi(m)}$, and we have

$$T_n \times \mathbb{A}^{D(m,n)} \sim \mathbb{A}^{\phi(n)+D(m,n)} \tag{3}$$

where

$$D(m, n) = \phi(m) \sum_{d \mid \frac{n}{m},\, \mu(\frac{n}{md})=-1} d$$

and $\sim$ denotes efficient "almost bijections". The smaller $D(m, n)$ is, the better for our applications. Given the current state of knowledge about the rationality of the tori T_m, we take m with at most two prime factors. Ideally, $m = 6$. One could also take $m = 2$. When $m = 6$, then (3) gives

$$T_{30} \times \mathbb{A}^2 \sim \mathbb{A}^{10} \qquad \text{and} \qquad T_{210} \times \mathbb{A}^{24} \sim \mathbb{A}^{72}.$$

As a comparison with the original bijection (1) for $n = 30$ which requires $8m+32$ elements of $\mathbb{F}_q$ to represent m elements in $T_{30}(\mathbb{F}_q)$, we see that this provides a considerable improvement.

Even better, using Proposition 1 and induction on the number of prime divisors of n, we also obtain the following.

Theorem 5 *If $n = p_1 \cdots p_k$ is a product of $k \geq 2$ distinct primes, then*

$$\Phi_n(x) \prod_{i=2}^{k-1} \Phi_{p_1 \cdots p_i}(x^{p_{i+2} \cdots p_k}) = \Phi_{p_1 p_2}(x^{p_3 \cdots p_k}).$$

Applying this to $n = 210 = 2 \cdot 3 \cdot 5 \cdot 7$, one can similarly show

$$T_{210}(\mathbb{F}_q) \times T_{30}(\mathbb{F}_q) \times T_6(\mathbb{F}_{q^7}) \sim T_6(\mathbb{F}_{q^{35}}).$$

Now since $T_6 \sim \mathbb{A}^2$, we obtain $T_{210} \times T_{30} \times \mathbb{A}^{14} \sim \mathbb{A}^{70}$. Using $T_{30} \times \mathbb{A}^2 \sim \mathbb{A}^{10}$ now gives $T_{210} \times \mathbb{A}^{22} \sim T_{210} \times \mathbb{A}^{10} \times \mathbb{A}^{12} \sim T_{210} \times (T_{30} \times \mathbb{A}^2) \times \mathbb{A}^{12} \sim T_{210} \times T_{30} \times \mathbb{A}^{14} \sim \mathbb{A}^{70}$, so

$$T_{210} \times \mathbb{A}^{22} \sim \mathbb{A}^{70}.$$

More generally, the above reasoning shows that if $n = p_1 \cdots p_k$ (square-free), then

$$T_n \times \mathbb{A}^{\phi(p_1 p_2) p_3 \cdots p_n - \phi(n)} \sim \mathbb{A}^{\phi(p_1 p_2) p_3 \cdots p_n},$$

which for $6 \mid n$ gives

$$T_n \times \mathbb{A}^{n/3-\phi(n)} \sim \mathbb{A}^{n/3}. \tag{4}$$

Using (4), one can compress m elements of $T_n(\mathbb{F}_q)$ down to just $(m-1)\phi(n)+n/3$ elements of $\mathbb{F}_q$ by either sequential or tree-based chaining as explained in §4.

3.1 Applying the Construction to T_{30}

Henceforth we focus on $n = 30$ since this improves upon previous schemes, has a straightforward parameter generation (see §5), and will be computationally efficient. Note that $\gcd(\Phi_{30}(q), \Phi_6(q)) = 1$. Indeed, using the first paragraph of the proof of Lemma 6 of [6], the only possible prime dividing $\gcd(\Phi_{30}(q), \Phi_6(q))$ is 5, but it is easy to see that regardless of q we have $\Phi_6(q) \bmod 5 \in \{1, 2, 3\}$, which proves our claim. By Theorem 2 we now have

$$T_{30}(\mathbb{F}_q) \times T_6(\mathbb{F}_q) \cong T_6(\mathbb{F}_{q^5}).$$

The compression is based on a sequence of maps

$$T_{30}(\mathbb{F}_q) \times (\mathbb{A}^2(\mathbb{F}_q) \setminus V(f)) \to T_{30}(\mathbb{F}_q) \times T_6(\mathbb{F}_q) \to T_6(\mathbb{F}_{q^5}) \to \mathbb{A}^2(\mathbb{F}_{q^5}) \setminus V(f_5),$$

where $V(f_5)$ denotes $V(f)$ over $\mathbb{F}_{q^5}$. We denote by θ the forward composition of the three maps above, and by θ^{-1} the composition of the inverses. Note that if we have m elements in $T_{30}(\mathbb{F}_q)$, we compress them down to $8m + 2$ elements of $\mathbb{F}_q$. Thus the compression outperforms CEILIDH and XTR when as few as two elements are compressed.

The first and last maps are based on CEILIDH decompression and compression, respectively. We discuss some possibilities for the map $\sigma(\cdot, \cdot)$ between $T_{30}(\mathbb{F}_q) \times T_6(\mathbb{F}_q)$ and $T_6(\mathbb{F}_{q^5})$ in §5 below.

3.2 Missing points

With regard to the functionality of θ, the only remaining issue is that the outer two maps based on CEILIDH do not give everywhere-defined injections.

We can slightly modify the CEILIDH maps, so that for compression we get an injection $\psi' : \mathbb{A}^2(\mathbb{F}_q) \to T_6(\mathbb{F}_q) \times \{0, 1\}$ and for decompression an injection $\rho' : T_6(\mathbb{F}_q) \times \{0, 1\} \to \mathbb{A}^2(\mathbb{F}_q)$. Note that ψ' and ρ' need not be inverses. The two missing points in ρ's domain can easily be added by using a table lookup into two arbitrarily chosen points in $V(f)$. The resulting map is ρ'.

Given the different cardinalities of $T_6(\mathbb{F}_p)$ (namely $p^2 - p + 1$) and $\mathbb{A}^2(\mathbb{F}_p)$ (namely p^2), there are certain points in $\mathbb{A}^2(\mathbb{F}_p)$ that do not decompress. If we concentrate on the case $p \equiv 2 \bmod 9$ or $p \equiv 5 \bmod 9$, then the variety $V(f)$ is defined by $f(v_1, v_2) = 1 - v_1^2 - v_2^2 + v_1 v_2$. For fixed v_2 this has at most 2 roots, and if this is the case then their difference is $(4 - 3v_2^2)^{1/2}$. If this expression equals 2 then $v_2 = 0$, in which case $v_1 \in \{-1, 1\}$. Thus we have a map $\psi' : \mathbb{A}^2(\mathbb{F}_q) \to T_6(\mathbb{F}_q) \times \{0, 1\}$:

- If $f(v_1, v_2) \neq 0$, then $\psi'(v_1, v_2) = (\psi(v_1, v_2), 0)$,
- Else if $v_2 \neq 0$, then $\psi'(v_1, v_2) = (\psi(v_1 + 2, v_2), 1)$,
- Else $\psi'(v_1, v_2) = (\psi(v_1 + 1, v_2), 1)$,

where the extra bit indicates whether the input landed in the variety.

4 Applications

Our new map saves a significant amount of communication in applications where many group elements are transmitted. For instance the compression can be used to agree on a sequence of keys using Diffie-Hellman as in §5.1 of [6]. Other applications include verifiable secret sharing, electronic voting and mix-nets, and private information retrieval.

In our applications we compress many elements. This is done by using part of the output of the i-th element as the affine input for the compression of the $(i+1)$-st element. This sequential chaining is simple, but has the drawback of needing to decompress all elements in order to obtain the first element. Alternatively, one can use trees to allocate the output of previous compressions. For instance, the output of the first compression is split into five pieces, which are subsequently used as the affine input when compressing elements two through six. The output of the second compression is used to compress elements seven through twelve, etc. When compressing m elements, decompressing a specific element now takes $O(\log m)$ atomic decompressions on average.

4.1 ElGamal encryption

Our first application is ElGamal encryption with a small message domain, where we obtain an additional 10% compression over CEILIDH even for the encryption of a single message. This contrasts starkly with the original mapping of [6] that cannot be used to achieve any savings for single-message encryption.

Let q and l be primes such that $l \mid \Phi_{30}(q)$. Let $g \in \mathbb{F}_{q^{30}}^{\times}$ have order l, so that $\langle g \rangle \subseteq T_{30}(\mathbb{F}_q)$. For random a, $1 \leq a \leq l-1$, let a be Bob's private key and $A = g^a$ his public key. Without loss of generality, let $\mathcal{M} = \{0, 1, \ldots, m-1\}$ be the set of possible messages. We assume that m, the cardinality of $\mathcal{M}$, is small. We apply the mapping of §3 to the generalized ElGamal encryption scheme.

Encryption (M):

1. Alice represents the message M as $g^M \in \langle g \rangle$.
2. Alice selects a random integer k, $1 \leq k \leq l$, and computes $d = g^k$.
3. Alice sets $e = g^M \cdot (g^a)^k$.
4. Alice expresses $d \in T_6(\mathbb{F}_{q^5})$ as $(d_1, d_2) \in \mathbb{A}^8(\mathbb{F}_q) \times \mathbb{A}^2(\mathbb{F}_q) \cong \mathbb{A}^2(\mathbb{F}_{q^5})$ by using CEILIDH. Alice compresses $e \in T_{30}(\mathbb{F}_q)$ and $d_2 \in \mathbb{A}^2(\mathbb{F}_q)$ as $\theta(e, d_2) = T$, and outputs (d_1, T).

Decryption (d_1, T):

1. Bob computes $\theta^{-1}(T) = (e, d_2)$ and uses CEILIDH to reconstruct d.
2. Bob uses his private key a to recover $g^M = d^{-a}e$.
3. Bob recovers M from g^M using the fact that M comes from a small domain (e.g., using Pollard lambda or a table lookup).

The ciphertext is represented as 18 symbols in $\mathbb{F}_q$, which is a 10% improvement over a solution in which CEILIDH is used to compress both d and e. Note that the mapping of [6] in §2.4 cannot be used to achieve any savings in this case.

Our scheme preserves homomorphy, that is, without knowing the secret key a one can compute the encryption of $M_1 + M_2$ given encryptions of M_1 and M_2 separately. This is useful in applications such as the efficient two-party computations proposed by Schoenmakers and Tuyls [26] which use homomorphic ElGamal encryption for a small number of messages.

Exactly as for XTR and CEILIDH (with 6 replaced by 30), the security of our schemes follows from the difficulty of the DDH problem in $\mathbb{F}_{q^{30}}^{\times}$, the fact that $T_{30}(\mathbb{F}_q)$ is the primitive subgroup of $\mathbb{F}_{q^{30}}^{\times}$, and the fact that our map and its inverse are efficiently computable.

The representation of M as g^M in $\langle g \rangle$ is not efficient when m is large. We leave it as an open problem to adapt our scheme to handle a larger message domain. We note that one solution is to use hybrid ElGamal encryption instead. Indeed, we may apply the mapping of §3 to hybrid ElGamal encryption, adapting a protocol in §5.3 of [6]. In general, though, this solution does not preserve the homomorphic property.

4.2 ElGamal Signatures

We apply the mapping of §3 to the generalized ElGamal signature scheme, adapting a protocol in §5.2 of [6]. Here the signature has the form (d, e), where $d \in \langle g \rangle$ and $1 \leq e \leq l-1$. The idea is to use part of e in the affine component of θ, which can be done without any loss since $\log_2 e \approx 160$ while $2\log_2 q \approx 70$; see §5.5 for a discussion of parameters. Since the affine component of [6] is much larger, this is not possible in their setting.

For a random a, $1 \leq a \leq l-1$, let a be Alice's private key and $A = g^a$ her public key. Let $h : \{0,1\}^* \to \mathbb{Z}_l$ be a cryptographic hash function. We have the following generalized ElGamal signature scheme for input message M:

Signature Generation (M):

1. Alice selects a random secret integer k, $1 \leq k \leq l$, and computes $d = g^k$.
2. Alice then computes $e = k^{-1}(h(M) - ah(d)) \bmod l$.
3. Alice expresses e as $(e_1, e_2) \in \mathbb{F}_q^2 \times \{0,1\}^*$, computes $\theta(d, e_1) = T$, and outputs (e_2, M, T) as her signature.

Signature Verification (e_2, M, T):

1. Bob computes $\theta^{-1}(T) = (d, e_1)$ and recovers e.
2. Bob accepts the signature if and only if $A^{h(d)} d^e = g^{h(M)}$.

We note that in practice one has the alternative of using Schnorr's signature scheme, which already achieves optimal compression.

4.3 Voting Schemes

We will discuss a recent voting scheme by Kiayias and Yung [11], which is based on the discrete logarithm problem and for which we propose to use T_{30}. We give a comparison with a cutting edge scheme based on Paillier encryption due to Damgård and Jurik [5].

Let L denote the number of voters. Each voter has a secret key a_i, and a public key g^{a_i}. The i-th voter chooses L random exponents $s_{i,j} \in \mathbb{Z}_l$ which satisfy $\sum_j s_{i,j} = 0$, where j ranges from 1 to L (the scheme is self-tallying, which basically means that the voters themselves serve as the talliers). Voter i computes and posts $g^{a_j s_{i,j}}$ for all j, and a zero-knowledge proof that his post is well-formed. Define $t_j = \sum_i s_{i,j}$ and observe that

(a) $\sum_j t_j = 0$,
(b) t_j is a random element in $\mathbb{Z}_l$ if at least one user is honest.

From the posts, the j-th voter can compute $g^{a_j t_j}$, and then by using a_j can also compute g^{t_j}. If f is another public generator of $\langle g \rangle$, the vote of the j-th voter is a bit b_j from which he can calculate and post $g^{t_j} f^{b_j}$. From all such posts, we have $\prod_j g^{t_j} f^{b_j} = f^{\sum_j b_j}$. Since the tally $\sum_j b_j \leq L$, it can be found with Pollard's lambda method in $O(\sqrt{L})$ multiplications (and one already needs $\Theta(L)$ multiplications to compute $\prod_j g^{t_j} f^{b_j}$).

Damgård and Jurik [5] propose a similar scheme as Kiayias and Yung, but use Paillier encryption [19] as a starting point. Again, there are L voters, L secret keys Sk_i and public keys Pk_i. The i-th voter chooses L random integers $s_{i,j}$ in a predefined range with $\sum_j s_{i,j} = 0$. Voter i posts $E_{Pk_j}(s_{i,j})$ for all j, and a zero-knowledge proof that these values are well-formed. Define t_j as above, and observe properties (a) and (b) again hold (the latter statistically). From the posts, voter j computes $E_{Pk_j}(t_j)$, and using Sk_j gets t_j. If his vote is b_j, he then posts $p_j = t_j + b_j$. Observe that $\sum_j p_j = \sum_j b_j$. Hence, tallying is more efficient than in the scheme of Kiayias and Yung, using L additions versus $O(L)$ multiplications.

Although the Paillier-based scheme can be expected to be faster, a scheme based on T_{30} is considerably more compact. We give an analysis of the communication required for both schemes under the assumption that one wants $\log_2 n = 1024$, and that one achieves the same level of security with $30 \log_2 q = 1024$ and a subgroup size of 160 bits.

The communication of the Kiayias-Yung scheme is dominated by the sending of $g^{a_j s_{i,j}}$ for all i, j, together with their zero-knowledge proofs. When looking at the zero-knowledge proofs used, one sees that each voter transmits $4L$ group elements and L exponents. Thus we can use T_{30} compression in $\mathbb{F}_{q^{30}}$ here. This results in roughly $4L \cdot 8 \log_2 q + 160L = 32L \log_2 q + 160L \approx 1253L$ bits per voter.

The communication of the Damgård-Jurik scheme is similarly dominated by the sending of the $E_{PK_j}(s_{i,j})$ for all i, j with their zero-knowledge proofs. We note that their encryption scheme E is not exactly the same as that of Paillier, but a modification of it where the ciphertext size is at least $3 \log_2 n$ bits, n being an RSA modulus. Moreover, each proof that $E_{PK_j}(s_{i,j})$ is well-formed costs at least $5 \log_2 n$ bits. Thus $8L \log_2 n = 8192L$ bits are transmitted per voter in this stage.

Hence our improvement is roughly a factor of 6.5. An additional optimization is for the bulletin board to compress the list of public keys g^{a_i} when distributing this at the beginning. We note that although we improve in communication and bulletin board size, the computational workload has clearly increased.

4.4 Mix-nets

A typical *re-encrypting* mix-net involves M servers that each process n ciphertexts. To process the batch, a server randomly permutes and rerandomises the ciphertexts. In the literature, both ElGamal and Paillier type schemes are used. We save on the communication for both sequential and parallel mix-nets [8]. In a *sequential re-encrypting mix-net*, server i processes the n ciphertexts, then passes them to server $i+1$, and after the M-th server is done, they perform some kind of threshold decryption. Here $n \cdot M$ bits are communicated using either Paillier or ElGamal, but we save using ElGamal together with T_{30} compression. Our savings is a factor of 30/8. In *parallel mixing*, the servers process disjoint batches of input ciphertexts in parallel, and in between processing rounds they transmit n/M ciphertexts to each other, and again we save using T_{30} compression.

5 Representations and Algorithms for T_{30}

In this section we discuss implementation issues concerning field representation, key generation, and efficient exponentiation.

5.1 Field Representations

Since $T_{30}(\mathbb{F}_q) \subset \mathbb{F}_{q^{30}}^{\times}$, we need a model of the latter that permits fast multiplication, squaring, inversion and a fast Frobenius automorphism. We also require arithmetic for T_6, over both $\mathbb{F}_q$ and $\mathbb{F}_{q^5}$. Since $T_{30}(\mathbb{F}_q) \subset T_6(\mathbb{F}_{q^5})$, we may model the arithmetic of $T_{30}(\mathbb{F}_q)$ by the latter, possibly at the risk of losing some optimizations.

The base field $\mathbb{F}_q$ We base our implementation on high performance arithmetic in $\mathbb{F}_q$ using the representational method of Montgomery [17, 3]. For T_{30} one is likely to use characteristics q between 32 and 64 bits long, corresponding to a 2-word value on a 32-bit architecture. We are careful to distinguish between those small, 2-word values required by $T_6(\mathbb{F}_{q^5})$ and more general values of q (which we need for comparison purposes). Essentially, we employ the trivial program specialisation techniques described by Avanzi [1] to construct compact, straight line code sequences for the 2-word case. This affords a significant improvement over code for general sizes of q. Other than the length, we do not rely on assumptions on the value of q, although one could expect incremental improvements by doing so. Also, our choice of extension degree poses some congruence conditions on q.

The extension $\mathbb{F}_{q^5}$ We use a degree five subfield of the degree 10 extension $\mathbb{F}_q[t]/(\Phi_{11}(t))$, and use the Gaussian normal basis $\{t+t^{10}, t^7+t^4, t^5+t^6, t^2+t^9, t^3+t^8\}$. For this to work we require that $q \neq \pm 1 \mod 11$ [18]. Since the extension degree is small, we perform inversions using the Itoh-Tsujii algorithm [10].

The torus T_6 For the torus T_6 we take $q \equiv 2 \mod 9$ or $q \equiv 5 \mod 9$ and use arithmetic based on the degree six extension field defined by adjoining a primitive ninth root of unity to the base field, as in [28, 23, 9]. Note that in T_6 we have virtually free inversion, as it is just the cube of the Frobenius automorphism.

5.2 Compression and Decompression

Our new compression and decompression algorithms require two components: CEILIDH and CRT. We use an implementation of CEILIDH as given in [9].

Although it seems that Chinese remaindering is straightforward, there is some flexibility in choosing the map $\sigma : T_{30}(\mathbb{F}_q) \times T_6(\mathbb{F}_q) \to T_6(\mathbb{F}_{q^5})$. Following [6] we have $\sigma(x, y) = x^\beta y^\alpha$, where $\alpha\Phi_{30}(q) + \beta\Phi_6(q) = 1$. The inverse is given by $\sigma^{-1}(z) = (z^{\Phi_6(q)}, z^{\Phi_{30}(q)})$. The cost of the forward computation (i.e., σ) is an exponentiation in $T_{30}(\mathbb{F}_q)$, an exponentiation in $T_6(\mathbb{F}_q)$, and a multiplication. Depending on the context, the exponentiation in $T_{30}(\mathbb{F}_q)$ may be combined with an exponentiation performed as part of a particular protocol. The inverse is a double exponentiation.

Also attractive is the simple $\sigma'(x, y) = xy$ with inverse $(\sigma')^{-1}(z) = (zy^{-1}, y)$ where $y = z^{\alpha\Phi_{30}(q)}$. Clearly the forward map only costs a multiplication. For the inverse we first compute y using a single exponentiation. Note that the exponent here is larger than in the case of σ, but the total amount of exponent is similar in both cases (although asymptotically it is not the total amount that counts, it is what is relevant in practice). Moreover, we are typically concerned with the case where the preimage $x \in T_{30}(\mathbb{F}_q)$ has an order l dividing $\Phi_{30}(q)$ so we know that z has order dividing $l(q^2 - q + 1)$, which we can use to reduce the exponent $\alpha\Phi_{30}(q)$. As noted before, the computation of y^{-1} is virtually free, so this method is clearly preferable to the first suggestion.

5.3 Arithmetic Costs

Let M, A, S and I represent the cost of multiplication, addition, squaring and inversion in $\mathbb{F}_q$, respectively. In Table 1 (cf. [9, Lemma 3]) we detail the relative costs for arithmetic in $\mathbb{F}_{q^5}$, and for both $T_6(\mathbb{F}_q)$ and $T_6(\mathbb{F}_{q^5})$. Compression and decompression are based on CEILIDH.

5.4 Exponentiation in T_{30}

In protocols, one is required to perform one of three operations involving exponentiation: a single exponentiation in $T_{30}(\mathbb{F}_q)$, a double exponentiation in $T_{30}(\mathbb{F}_q)$, or a single exponentiation in $T_6(\mathbb{F}_{q^5})$ (for the map $(\sigma')^{-1}$ described

	$\mathbb{F}_{q^5}$	$T_6(\mathbb{F}_q)$	$T_6(\mathbb{F}_{q^5})$
Multiply	$15M + 75A$	$18M + 53A$	$270M + 1615A$
Square		$6M + 21A$	$90M + 555A$
Inverse	$65M + 300A + I$	$2A$	$10A$
Frobenius	0	$1A$	$5A$
Compress		$15M + 31A + I$	$290M + 1580A + I$
Decompress		$27M + 52A + I$	$470M + 2585A + I$

Table 1. Arithmetic costs.

above). Since $T_{30}(\mathbb{F}_q) \subset T_6(\mathbb{F}_{q^5})$, we can perform all three of these in $T_6(\mathbb{F}_{q^5})$ using the methods developed in [28]; the main properties one can exploit are the degree two Frobenius automorphism and fast squaring.

In a subgroup of order l where $l|(q^{10} - q^5 + 1)$, we write an exponent m as $m \equiv m_1 + m_2 q^5 \mod l$, where m_1 and m_2 are approximately half the bit-length of l (as in [28]). One can find m_1 and m_2 very quickly having performed a one-time Gaussian two dimensional lattice basis reduction, and using this basis to find the closest vector to $(m, 0)^T$. Having split the exponent, to compute a^m for random a and m, we perform a double exponentiation $a^{m_1}(a^{q^5})^{m_2}$ using the Joint Sparse Form (JSF) of the integers m_1 and m_2 [27], which on average halves the number of pairs of non-zero bits in their paired binary expansion. The use of the JSF in the above is possible since we have virtually free inversion.

To perform a double exponentiation, we split both exponents as with the single exponentiation, and perform the necessary four-fold multi-exponentiation as a product of two double exponentiations, combining the required squarings.

In general one may also be able to exploit the additional structure of T_{30}, which possesses an automorphism of degree eight, namely, the Frobenius automorphism. One can in principle employ exactly the same method as above and perform an eight-fold multi-exponentiation. However for the parameter sizes we consider in this paper, we use a much simpler method based on the q-ary expansion of an exponent m. Specifically, since our value of q will be small we can write an exponent m as $m = \sum m_i q^i$.

For our implementation where q^{30} has size approximately 1024 bits, exponentiating by a 160 bit exponent consists of five terms in the q-ary expansion, and hence we perform a five-fold multi-exponentiation. To perform this one can use ideas of Proos [21], which extend the JSF to more than two exponents. However due to the amount of precomputation required, for exponents of cryptographic interest we use a naive combination of the JSF and the non-adjacent form (NAF). This results in an effective exponent length of around the same size as q, significantly reducing the number of squarings needed for exponentiation.

With regard to decompression, the exponent in this case is slightly longer than for a single exponentiation. Again we use a q-ary expansion, consisting of seven terms in this instance, and apply the JSF to three pairs of them and the NAF to the remaining one.

q	l
2229155309	931607823866669709267930039057677132828697751771
2527138379	963373263959318090938089232997832791220899903311
2559356147	922800311037389261880873570251585702571121590451
2925130259	899122187666688780457417063691715267976198516591
3020282723	734463532846449031549478184170595775906318188901
3734718203	789572131486790156853093352977895757720566978441

Table 2. Parameter examples with 32-bit primes q

q	l
9909125592335111369	1056384871088595423227115173568048528184621140903052910805301
10640772970658245433	3170119585137777422832938014760851013504258723575431018642871
11042402719715204339	1179345732085674283659621603717770735788409366766144466686061
11391285666382073129	1293678412210548537320558698939727346786705884728706067133651
11868436123416952031	1230352242796051691760643717809792393751225110105630495113071
17174393702711641469	1070675878645369998848869455205552403869773154208635001001721

Table 3. Parameter examples with 64-bit primes q

Note that for larger parameter choices, one can clearly construct more efficient multi-exponentiation methods than those we have optimised for 1024 bit fields. We omit the details.

5.5 Parameter Selection

Rubin and Silverberg discuss parameter selection for T_{30} in §3.10 of [25]. We followed their method, starting with primes $p \equiv 1 \bmod 30$ of about 30 (resp., 61) bits, finding 32-bit (resp., 64-bit) primes q of order 30 mod p such that $q \equiv 2 \bmod 9$ and $q \equiv 7 \bmod 11$, then using the Elliptic Curve Method to remove small prime factors from $\Phi_{30}(q)/p$, and checking to see if what remains is a prime of about 160 (resp., 200) bits. This results in parameters with q a 32-bit (resp., 64-bit) prime with the property that the order of $T_{30}(\mathbb{F}_q)$ is divisible by a 160-bit (resp., 200-bit) prime l. These choices give security equivalent to 960-bit (resp., 1920-bit) RSA security.

By suitably optimizing the Elliptic Curve Method parameter choices, we were able to generate parameters at a rate of about one example every minute or two for 32-bit primes q (with 160-bit primes l), using a Macintosh G5 dual 2.5GHz computer. For 64-bit primes q, we obtained examples with l a prime of between 198 and 202 bits at a rate of one every few hours. The parameters are like Diffie-Hellman parameters, in the sense that the same parameters can be used for all users, and for many applications do not need to be changed frequently. In Table 2 we list some examples with 32-bit primes q and 160-bit primes l. In Table 3 we list some examples with 64-bit primes q and 200-bit primes l.

	$\mathbb{F}_{q_L}$	$\mathbb{F}_{q_S}$	$\mathbb{F}_{q_S{}^5}$	$T_6(\mathbb{F}_{q_L})$	$T_{30}(\mathbb{F}_{q_S})$
Addition	$0.80\mu s$	$0.52\mu s$	$0.82\mu s$		
Frobenius			$0.48\mu s$	$1.64\mu s$	$1.10\mu s$
Square	$2.51\mu s$	$0.61\mu s$		$13.80\mu s$	$21.61\mu s$
Multiply	$2.58\mu s$	$0.62\mu s$	$3.78\mu s$	$32.30\mu s$	$65.92\mu s$
Inverse	$92.71\mu s$	$2.04\mu s$	$16.03\mu s$	$1.82\mu s$	$1.29\mu s$

Table 4. Timings of basic field and torus arithmetic.

	$T_6(\mathbb{F}_{q_L})$	$T_{30}(\mathbb{F}_{q_S})$
Compression		
Compress	$131.30\mu s$	$0.13ms$
Decompress	$188.61\mu s$	$4.92ms$
Exponentiation		
Binary	$5.21ms$	$9.12ms$
Sliding Window	$4.39ms$	$7.53ms$
q-ary		$3.11ms$
JSF Single	$2.79ms$	$4.57ms$

Table 5. Timings for exponentiations

6 Timings

In order to understand the real-world performance of our construction, we implemented the entire system and ran a number of timing experiments. Our main goal is to compare the performance of an implementation of $T_6(\mathbb{F}_{q_L})$ using CEILIDH against an implementation of $T_{30}(\mathbb{F}_{q_S})$ of similar cardinality using our construction. Here, we denote the special cases of large and small q as q_L and q_S. We used $\log_2(q_L) \approx 5 \cdot \log_2(q_S) \approx 176$ bits, so that in both cases, there is a subgroup of roughly $\log_2(l) \approx 160$ bits in size. These parameters heuristically provide the equivalent of 1024-bit RSA security.

We constructed our implementation entirely in C++, apart from small sequences of assembly language to accelerate arithmetic in $\mathbb{F}_q$, using the GCC 3.4.2 compiler suite. The timing experiments were carried out on a Linux based PC housing a 2.8 GHz Intel Pentium 4 processor and 1 GB of memory. We selected our system parameters as in §5.5. In all of our timing experiments we generated random operands and averaged the timings of many experiments to get a representative result. Note that exponents are reduced modulo l in all cases. Our sliding window had a maximum size of four.

Table 4 shows timings for basic field and torus arithmetic. Arithmetic in $\mathbb{F}_{q_L}$ is used in $T_6(\mathbb{F}_{q_L})$ and arithmetic in $\mathbb{F}_{q_S}$ and $\mathbb{F}_{q_S{}^5}$ is used in $T_{30}(\mathbb{F}_{q_S})$. One interesting feature of these results is the low cost ratio between inversion and multiplication in $\mathbb{F}_{q_S}$, which is potentially interesting to follow up. Table 5 details the cost of mapping between different representations (compress and decompress) and the cost of different exponentiation methods which might be used within an actual cryptosystem.

It is difficult to get an exact comparison with other work on ECC and XTR, partly because of differences in host processor and levels of optimisation used by different authors in producing benchmark timings. However, a comparison with the highly optimised ECC results of Avanzi [1], for example, gives some insight. For similar levels of security, direct comparison shows an exponentiation in $T_{30}(\mathbb{F}_{q_S})$ is only around twice as costly as an ECC point multiplication; correcting for the difference in processors still means that $T_{30}(\mathbb{F}_{q_S})$ is at least competitive. The case of XTR is easier to compare against since we essentially use the same experimental platform as that given in Granger, Page and Stam [9]. It turns out that XTR is marginally faster. Solely from the point of view of performance, we conclude that our construction is a competitive alternative to existing cryptosystems.

7 Conclusions

We construct an efficient "almost bijection" between $T_{30}(\mathbb{F}_q) \times \mathbb{A}^2(\mathbb{F}_q)$ and $\mathbb{A}^{10}(\mathbb{F}_q)$ which achieves better compression than XTR and CEILIDH for the compression of as few as two group elements. We give several applications, and obtain ElGamal ciphertexts that are 10% smaller than in previous schemes. We also develop an efficient implementation, using a variety of techniques for reducing the computational requirements and obtaining a scheme much more practical than that in [6]. From experimental results we conclude that our construction is a competitive alternative to the best existing public key cryptosystems.

References

[1] R.M. Avanzi. Aspects of Hyperelliptic Curves over Large Prime Fields in Software Implementations. *CHES'04*, LNCS **3156**, 148–162.

[2] W. Bosma, J. Hutton and E. R. Verheul. Looking beyond XTR. *Asiacrypt'02*, LNCS **2501**, 46–63.

[3] A. Bosselaers, R. Govaerts and J. Vandewalle. Comparison of Three Modular Reduction Functions. *Crypto'94*, LNCS **773**, 175–186.

[4] A.E. Brouwer, R. Pellikaan and E.R. Verheul. Doing More with Fewer Bits. *Asiacrypt'99*, LNCS **1716**, 321-332.

[5] I. Damgard and M. Jurik. A Length-Flexible Threshold Cryptosystem with Applications *ACISP'03*, LNCS **2727**, 350–364.

[6] M. van Dijk and D. Woodruff. Asymptotically Optimal Communication for Torus-Based Cryptography. *Crypto'04*, LNCS **3152**, 157–178.

[7] S. Goldwasser and S. Micali. Probabilistic Encryption. In *Comp. Sys. Sci*, **28**(1), 270–299, 1984.

[8] P. Golle and A. Juels. Parallel Mixing. In *Computer and Communications Security (CSS)*, ACM, 220–226, 2004.

[9] R. Granger, D. Page, and M. Stam. A comparison of CEILIDH and XTR. *ANTS-VI*, LNCS **3076**, 235–249, 2004.

[10] T. Itoh and S. Tsujii. A Fast Algorithm for Computing Multiplicative Inverses in $GF(2^m)$ Using Normal Bases. In *Info. and Comp.*, **78**(3), 171–177, 1988.

[11] A. Kiayias and M. Yung. Self-Tallying Elections and Perfect Ballot Secrecy. *PKC'02*, LNCS **2274**, 141–158.

[12] A.A. Klyachko. On the Rationality of Tori with Cyclic Splitting Field (Russian). In *Arithmetic and Geometry of Varieties*, Kuybyshev Univ. Press, 73–78, 1988.

[13] E. Kushilevitz and R. Ostrovsky. Replication is not needed: single database, computationally-private information retrieval. In *Foundations of Computer Science (FOCS)*, IEEE Press, 364–373, 1997.

[14] M.J.J. Lennon and P.J. Smith. LUC: A New Public Key System. In *IFIP TC11 Ninth International Conference on Information Security IFIP/Sec*, 103–117, 1993.

[15] A.K. Lenstra and E.R. Verheul. The XTR Public Key System. *Crypto'00*, LNCS **1880**, 1–19.

[16] A.K. Lenstra and E.R. Verheul. An Overview of the XTR Public Key System. In *Public-Key Cryptography and Computational Number Theory*, Verlages Walter de Gruyter, 151–180, 2001.

[17] P.L. Montgomery. Modular Multiplication Without Trial Division. In *Math. Comp.*, **44**, 519–521, 1985.

[18] M. Nöcker. Data structures for parallel exponentiation in finite fields. PhD Thesis, Universität Paderborn, 2001.

[19] P. Paillier. Public-key cryptosystems based on composite degree residuaosity classes. *Eurocrypt'99*, LNCS **1592**, 223–238.

[20] S.C. Pohlig and M.E. Hellman. An Improved Algorithm for Computing Logarithms over GF(p) and its Cryptographic Significance. In *IEEE Trans. on IT*, **24**, 106–110, 1978.

[21] J. Proos. Joint Sparse Forms and Generating Zero Columns when Combing. University of Waterloo, Technical Report CORR 2003-23.

[22] K. Rubin and A. Silverberg. Supersingular Abelian Varieties in Cryptology. *Crypto'02*, LNCS **2442**, 336–353.

[23] K. Rubin and A. Silverberg. Torus-Based Cryptography. *Crypto'03*, LNCS **2729**, 349–365.

[24] K. Rubin and A. Silverberg. Algebraic Tori in Cryptography. In *High Primes and Misdemeanours: Lectures in Honour of the 60th birthday of Hugh Cowie Williams*, Fields Institute Communications Series 41, American Mathematical Society, 317–326, 2004.

[25] K. Rubin and A. Silverberg. Using Primitive Subgroups to Do More with Fewer Bits. *ANTS-VI*, LNCS **3076**, 18–41, 2004.

[26] B. Schoenmakers and P. Tuyls, Practical Two-Party Computation Based on the Conditional Gate, *Asiacrypt'04*, LNCS 3329, 119–136.

[27] J.A. Solinas. Low-Weight Binary Representations for Pairs of Integers. University of Waterloo, Technical Report CORR 2001-41.

[28] M. Stam and A.K. Lenstra. Efficient Subgroup Exponentiation in Quadratic and Sixth Degree Extensions. *CHES'02*, LNCS **2523**, 318–332.

[29] M. Stam and A.K. Lenstra. Speeding Up XTR. *Asiacrypt'01*, LNCS **2248**, 125–143.

[30] V.E. Voskresenskiĭ. Algebraic Groups and Their Birational Invariants. *Translations of Mathematical Monographs* 179, American Mathematical Society, 1998.

[31] A. Yamamura and T. Saito. Private Information Retrieval Based on the Subgroup Membership Problem. *ACISP'01*, LNCS **2119**, 206–220.