Random Small Hamming Weight Products with Applications to Cryptography

Jeffrey Hoffstein, Joseph H. Silverman

NTRU Cryptosystems, Inc., 5 Burlington Woods, Burlington, MA 01803 USA,
jhoff@ntru.com, jhs@ntru.com

Abstract. There are many cryptographic constructions in which one uses a random power or multiple of an element in a group or a ring. We describe a fast method to compute random powers and multiples in certain important situations including powers in the Galois field $\mathbb{F}_{2^n}$, multiples on Koblitz elliptic curves, and multiples in NTRU convolution polynomial rings. The underlying idea is to form a random exponent or multiplier as a product of factors, each of which has low Hamming weight when expanded as a sum of powers of some fast operation.

Contents

Introduction

There are many cryptographic constructions in which one uses a random power or multiple of an element of a group or ring. A brief and far from complete list includes:

Diffie-Hellman Key Exchange One takes an element g in a finite field $\mathbb{F}$ and computes a random power g^k in $\mathbb{F}$. Here k is an integer.

Elliptic Curve DH Key Exchange One takes a point P in the group $E(\mathbb{F})$ of points on an elliptic curve over a finite field and computes a random multiple kP. Here k may be an integer or a more general endomorphism of the group $E(\mathbb{F})$.

DSS and ECDSS The Digital Signature Standard (using a finite field or an elliptic curve) requires a random power g^k or multiple kP in the signing portion of the algorithm. The verification process also require a power or multiple, but for specified values of k, not random values.

Classical ElGamal Public Key Cryptosystem ElGamal key generation requires computation of a power $\beta = \alpha^j$ with a fixed base α and a randomly chosen exponent j that forms the secret key. Encryption requires computation of two powers α^k and β^k to a randomly chosen exponent k. Decryption requires computation of a power γ^j.

Elliptic Curve ElGamal and Variants Key generation requires computation of a multiple $Q = jP$ with a fixed point P in $E(\mathbb{F})$ and a randomly chosen multiplier j that forms the secret key. Encryption requires computation of two multiples kP and kQ to a randomly chosen multiplier k. Decryption requires computation of a multiple jR. Again k may be an integer or a more general endomorphism of the group $E(\mathbb{F})$.

NTRU Public Key Cryptosystem The public key includes a random polynomial $f(X)$ in the ring $R_q = (\mathbb{Z}/q\mathbb{Z})[X]/(X^N - 1)$ of truncated polynomials modulo q. Encryption requires computation of a product $r(X)h(X)$ in the ring R, where $h(X)$ (the public key) is fixed and $r(X)$ is random. Decryption requires computation of a product $f(X)e(X)$ in the ring R, where $e(X)$ is the ciphertext.

In this note we describe a general method that in many situations allows random multiples to be computed more rapidly than previously described methods. Although not universally applicable, it can be used for many of the algorithms in the above list, including Diffie-Hellman over Galois fields $\mathbb{F}_{2^n}$, elliptic curve cryptography over Koblitz curves, and the NTRU cryptosystem.

Briefly, our idea is to write the random multiplier as a product of terms, each of which is a sum of terms that are relatively easily computed. We call these multipliers Small Hamming Weight Products (SHWP), because each term in the product has low Hamming weight relative to an easily computed operation.

The authors would like to thank Don Coppersmith, Dan Gordon, Victor Miller, and an anonymous referee for their suggestions and help with the existing literature.

1 Low Hamming Weight Exponents

In this section we describe some of the ways in which low Hamming weight exponents have been used in cryptography and, more generally, as a computational tool.

The use of low Hamming weight exponents has been studied in both RSA exponentiation [7] and in discrete logarithm algorithms [2, 13], but always in the context of taking a single exponent k of small Hamming weight. Our innovation is to use a product $k = k_1 k_2 \cdots k_r$ of very low Hamming weight exponents and take advantage of the fact that the sample space of the product k is more-or-less the *product* of the sample spaces for $k_1, \dots, k_r$, while the computational complexity (in certain situations) of computing α^k is the *sum* of the computational complexity of computing $\alpha_i^{k_i}$.

There is a vast literature on the evaluation of powers in general, see for example, [1, §1.2], [3], [6, §4.6.3], or [8, §14.6]. The usual binary method to compute x^k requires approximately $\lfloor \log_2 k \rfloor$ squarings and $\mathrm{HW}(k)$ multiplications, where

$$\mathrm{HW}(k) = \text{Hamming weight of } k$$

is the number of ones in the binary expansion of k. The use of addition chains for k will often yield an improvement, although for very large values of k it is difficult to find optimal chains.

We also mention an idea of Schnorr [10] to compute random powers by precomputing a list of powers, taking a product of a random subset, and gradually supplementing the list using intermediate calculations. Schnorr's method was broken by de Rooij [9] at the parameter levels suggested in [10].

Another method that is closer to the ideas in this paper is the *factor method*, which is briefly discussed in [6, §4.6.3, page 463 and exercise 3]. The idea is to write k as a product $k = uv$ and compute $z = x^k$ as $y = x^u$ and $z = y^v$. The process may be repeated and interspersed with the binary method or the use of other addition chains.

To illustrate the idea, let G be a group in which we want to compute the quantity x^k. Suppose that we write the exponent k as a sum of products

$$k = \sum_{i=1}^{d} k_i = \sum_{i=1}^{d} \prod_{n=1}^{N_i} K_{i,n}. \tag{1}$$

We compute x^k as the product $\prod_i x^{k_i}$, we compute each power x^{k_i} using the factor method with $k_i = \prod_n K_{i,n}$, and we compute each power $y^{K_{i,n}}$

by using (say) the binary method. This requires approximately $\log_2(k)$ squarings and approximately

$$d - 1 + \sum_{i=1}^{d} \sum_{n=1}^{N_i} \bigl(\mathrm{HW}(K_{i,n}) - 1\bigr) \quad \text{multiplications.} \tag{2}$$

For small values of k, one might ask for the decomposition (1) that minimizes (2). For larger values of k, one might ask for an algorithm that produces a reasonably small value of (2). These are both very interesting questions, but are not the focus of the present paper.

Both the goals and the analysis in this paper differ significantly from the material on exponentiation described in [6]. The goal in [6] is to describe efficient methods for computing x^k for a given exponent k. The subsequent analysis gives theoretical upper and lower bounds for the most efficient method and algorithms for taking a given k and finding a reasonably efficient way to evaluate x^k. Our goal in this paper is to find a collection of exponents k such that x^k is easy to compute and such that the collection is sufficiently "random" and sufficiently large. The seemingly minor change in perspective from specific exponents to random exponents actually represents a major shift in the underlying questions and in the methods that are used to study them.

There is a second important way in which our work differs from the factor method as described in [6]. We will concentrate on situations in which there is a "free" operation. We illustrate this point with an example. Let G be a group and suppose that we want to compute x^k using the factor method, where $k = uv$. The cost of computing x^k is approximately

$$(\log_2(k) \text{ squarings}) + (\mathrm{HW}(u) + \mathrm{HW}(v) \text{ multiplications}),$$

where we assume for simplicity that the two powers $y = x^u$ and $z = y^v$ are computed using the binary method. Now suppose that the (finite) group G has order N and suppose that we merely write k as a product modulo N, say $k \equiv uv \pmod{N}$. Then $y = x^u$ and $z = y^v$ will still give us the correct value $z = x^k$, but now the cost is approximately

$$(\log_2(uv) \text{ squarings}) + \mathrm{HW}(u) + \mathrm{HW}(v) \text{ multiplications.}$$

If squaring and multiplication take approximately the same amount of time, then this method will probably be very bad because the product uv will be very large.

On the other hand, if squaring is very fast, as it is for example in the Galois field $\mathbb{F}_{2^n}$, then large values of u and v may be advantageous as long as u and v have small (binary) Hamming weight. This is the other idea that we will develop in this paper in three situations of cryptographic interest, namely exponentiation in Galois fields $\mathbb{F}_{2^n}$, multiplication on Koblitz elliptic curves, and multiplication in NTRU convolution rings $\mathbb{F}_q[X]/(X^N-1)$. These specific situations are described in detail in Sections 2, 3, and 4. In Section 5 we discuss some of the issues surrounding the randomness of small Hamming weight products and describe some computations and experiments. Finally, in Section 6, we give a general formulation.

2 Random Powers in Galois Fields $\mathbb{F}_{2^n}$

In any group, the standard way to compute a power α^k is to use the binary expansion of k, see for example [1, §1.2] or [8, §14.6]. This reduces the computation of α^k to approximately $\log_2(k)$ squarings and $\mathrm{HW}(k)$ multiplies, where on average $\mathrm{HW}(k)$ equals approximately $\frac{1}{2}\log_2(k)$. (Using a signed binary expansion of k [8, §14.7.1] further reduces the number of multiplies, at the expense of an inversion.)

Binary powering algorithms apply to any group, but the feature we wish to exploit in $\mathbb{F}_{2^n}$ is the fact that squaring is essentially free compared to multiplication. Thus if k is randomly chosen in the interval from 1 to 2^n-1, then computation of α^k is dominated by the approximately $n/2$ multiplications that are required.

As indicated above, there are many cryptographic situations in which a person needs to compute α^k for a fixed base α and some randomly chosen exponent k. Generally a requirement is that k be chosen from a sufficiently large set that an exhaustive search (or more generally, a square root search such as Pollard's rho method) will be unable to determine k. Thus suppose that one chooses k to have the form

$$k=\sum_{i=0}^{n-1} k_i\cdot 2^i \quad \text{with } k_i\in\{0,1\}$$

with a fixed binary Hamming weight $d=\sum k_i$. Then the size of the search space of k is $\binom{n}{d}$. One typically wants the search space to have at least 2^{160} elements, since the running time will typically be proportional to the square root of the size of this space. More precisely, see [13] for a description of Coppersmith's baby-step giant-step algorithm to efficiently search this space in time proportional to $\sqrt{t}\binom{n/2}{d/2}$.

For cryptographic purposes, a typical value for n is $n\approx 1000$, which is dictated by the running time of sieve and index calculus methods for solving the discrete logarithm problem over $\mathbb{F}_{2^n}$. Then taking $d = 25$ gives a search space of size $\binom{1000}{25} \approx 2^{165}$, and computation of α^k requires 24 multiplications.

The new method we propose in this paper is to choose k to be a product of terms with very low binary Hamming weight. (More generally, one might use a sum of such products.) To illustrate with the above value $n \approx 1000$, suppose that we take k to have the form

$$k = k^{(1)}k^{(2)}k^{(3)},$$

where $k^{(1)}$ has binary Hamming weight 6 and $k^{(2)}$ and $k^{(3)}$ each have binary Hamming weight 7. Then the search space for k, which is the product of the search spaces for the three factors, has order $\binom{1000}{6}\binom{1000}{7}\binom{1000}{7} \approx 2^{165}$, while computation of

$$\alpha^k = ((\alpha^{k^{(1)}})^{k^{(2)}})^{k^{(3)}}$$

requires only

$$5 + 6 + 6 = 17 \text{ multiplications.}$$

This represents a savings of approximately 29%.

Remark 1. We have required a search space of order approximately 2^{160} since the standard square root search attacks [13] reduce the time to $O(2^{80})$. However, if k is a product of several low Hamming weight polynomials, it is not clear to us how to set up a square root attack on the full space. Thus if $k = k^{(1)}k^{(2)}k^{(3)}$, one can search (guess) the first two terms and then use a square root attack for the third term. A second approach to solving $\alpha^k = \beta$ for k is to transfer $k^{(3)}$ to the other side. Thus we let i run through the space of all products $k^{(1)}k^{(2)}$ and let j run through the space of all $k^{(3)}$ values and we make tables of the values of α^i and $\beta^{j^{-1}}$, where j^{-1} is the inverse of j modulo $2^n - 1$. Then the running time is proportional to the sum of the sizes of the two tables.

In the example given above, this yields a running time proportional to

$$\binom{1000}{6}\binom{1000}{7} + \binom{1000}{7} \approx 2^{107.7}.$$

However, in view of this search method, it makes more sense to take $k^{(3)}$ considerably larger than $k^{(1)}$ and $k^{(2)}$. Thus if we take $k^{(1)}$, $k^{(2)}$ and $k^{(3)}$ to have Hamming weights 2, 2, 11, respectively, then the first square root attack has time $O(2^{80.0})$ and the second square root attack has time $O(2^{84.3})$, and computation of α^k requires only 12 multiplications.

(We thank Don Coppersmith for showing us the second square root attack described in this remark.)

Remark 2. We have described how to use SHWP over fields with 2^n elements, but similar remarks apply to fields with p^n elements using multipliers of the form $\pm p^{e_1} \pm \cdots \pm p^{e_r}$.

3 Random Multiples on Koblitz Elliptic Curves

Let $E/\mathbb{F}_{2^m}$ be an elliptic curve defined over the field with 2^m elements, and let $P \in E(\mathbb{F}_{2^m})$ be a point on the curve. A number of cryptographic constructions require the computation of a multiple NP, where N has size comparable to 2^m. (See, e.g., [4, 8, 12].) Writing N in binary form as

$$N = N_0 + 2N_1 + 4N_2 + \cdots + 2^i N_i + \cdots + 2^m N_m \quad \text{with } N_0, \ldots, N_m \in \{0, 1\},$$

the computation of NP is reduced to approximately $N/2$ doublings and $N/2$ point additions. As already indicated, further savings may be obtained by choosing $N_0, \ldots, N_m$ in the set $\{-1, 0, 1\}$, reducing the number of additions to approximately $N/3$. Unfortunately, on elliptic curves, doubling a point is computationally more difficult than adding two different points.

For certain elliptic curves, it is possible to significantly reduce the necessary computation by replacing doubling with a Frobenius map that is essentially free. Let $E/\mathbb{F}_2$ be a "Koblitz curve", that is, an ordinary elliptic curve defined over the field with two elements. Thus E is one of the two curves

$$E : y^2 + xy = x^3 + ax^2 + 1 \qquad \text{with } a \in \mathbb{F}_2.$$

Let

$$\tau : E(\mathbb{F}_{2^m}) \to E(\mathbb{F}_{2^m}), \qquad \tau(x, y) = (x^2, y^2),$$

be the Frobenius map on E. The computation of $\tau(Q)$ takes very little time compared to point addition or doubling on E. It is possible to write any integer N as a linear combination

$$N = N_0 + \tau N_1 + \tau^2 N_2 + \cdots + \tau^i N_i + \cdots + \tau^m N_m \qquad \text{with } N_0, \ldots, N_m \in \{-1, 0, 1\},$$

and then the computation of NP is essentially reduced to $m/3$ additions in $E(\mathbb{F}_{2^m})$. (Approximately $m/3$ of the N_i's will be nonzero.) Further, for many cryptographic applications there is no real reason to use integer multiples of P; one can simply use multiples NP where N is a random linear combination of powers of τ as above. For example, Diffie-Hellman key exchange works perfectly well. See [4, 11] for basic material and computational methods on Koblitz curves.

To summarize, computation of a random signed τ-multiple of a point on a Koblitz curve over $\mathbb{F}_{2^m}$ requires approximately $m/3$ elliptic curve additions. We now describe a way to significantly reduce the number of elliptic curve additions. As indicated above, the basic idea is to choose the multiplier N to be a product of low Hamming weight linear combinations of τ.

For concreteness, we illustrate the idea with a particular field of cryptographic interest. We let $m = 163$, so we are working in the field $\mathbb{F}_{2^{163}}$. We choose N to have the form

$$N = N^{(1)}N^{(2)}N^{(3)} = \Bigl(1 + \sum_{u=1}^{6} \pm\tau^{i_u}\Bigr)\Bigl(1 + \sum_{u=1}^{6} \pm\tau^{j_u}\Bigr)\Bigl(1 + \sum_{u=1}^{6} \pm\tau^{k_u}\Bigr).$$

(We take each factor in the indicated form, since one can always pull off a power of τ from each factor. Using this form prevents overcounting.)

First we check how hard it is, given $Q = NP$, to perform a search for N or for some other integer N' satisfying $N'P = Q$. A square root search (e.g., Pollard rho) for N' takes on the order of $\sqrt{2^{163}}$ steps. A second search, which takes advantage of the special form of N, is to write the equation $Q = NP$ as

$$\bigl(N^{(3)}\bigr)^{-1}Q = N^{(1)}N^{(2)}P$$

and compare tables of values of the two sides. The time and space requirement for this search is the length of the longer of the two tables. For our example, each of the $N^{(i)}$'s is taken from a space of size $2^6\binom{162}{6} \approx 2^{40.4}$, so the table of values of $N^{(1)}N^{(2)}P$ has $O(2^{80})$ elements. Finally, one could try guessing the values of $N^{(1)}$ and $N^{(2)}$ and perform a square root search for $N^{(3)}$, but this gives an even larger search space.

The advantage of taking N in the above form is clear. Computation of the multiple

$$NP = N^{(1)}N^{(2)}N^{(3)}P$$

requires only $6 + 6 + 6 = 18$ elliptic curve additions. (Subtractions are essentially the same as additions.) It also requires many applications of powers of the Frobenius map τ, but these take very little time compared to point additions, so may be neglected in this rough analysis. We thus see that with this new method, a useful cryptographic multiple NP may be computed using 18 additions, rather than the approximately $163/3 \approx 54$ additions required by the earlier method. This yields a 3-fold speed increase.

Remark 3. We do not see a meet-in-the-middle attack on all of N, but even if such an attack exists, it suffices to replace the weights $(6,6,6)$ above with the weights $(8,9,9)$ to get a set of triples $\bigl(N^{(1)}, N^{(2)}, N^{(3)}\bigr)$ of order $2^{163.9}$. The computation of NP now requires 26 additions, yielding a speed increase by a factor of approximately 2.1.

Actually, in this situation it is even faster to use a product of four terms $N = N^{(1)}N^{(2)}N^{(3)}N^{(4)}$ with weights $(4,5,7,8)$. Then the total search space has size

$$2^4\binom{162}{4} \cdot 2^5\binom{162}{5} \cdot 2^7\binom{162}{7} \cdot 2^8\binom{162}{8} \approx 2^{160.48},$$

and the computation of NP requires only 24 additions for a speed increase by a factor of approximately 2.26.

Remark 4. One might instead take N to be a sum of products of small Hamming weight terms. For example, $N = N^{(1)}N^{(2)} + N^{(3)}N^{(4)}$ with the four terms having small Hamming weight. Of course, this allows a square root attack for the two halves of N by matching values of aP with values of $Q - bP$.

Remark 5. If one wants N to be an actual integer, rather than a polynomial in τ, one can include conjugate terms. For example, an expression of the form

$$\tau^i + \tau^{m-i}$$

represents an integer, and it is a simple matter to compute and store a table of values of $\tau^i + \tau^{m-i}$ for $1 \le i < m/2$.

Remark 6. This brings up the very interesting problem of whether one can efficiently find a small τ-Hamming weight multiplier system for a given integer N. We leave the precise formulation and study of this problem for a future paper.

4 The NTRU Public Key Cryptosystem

The NTRU public key cryptosystem uses truncated polynomials in the ring

$$R = \frac{(\mathbb{Z}/q\mathbb{Z})[X]}{(X^N - 1)}.$$

The encryption process includes computation of a product $r(X)h(X)$ for a fixed public key polynomial $h(X)$ and a randomly chosen polynomial $r(X)$ having small coefficients. The decryption process similarly includes computation of a product $f(X)e(X)$, where $e(X)$ is the ciphertext and the private key $f(X)$ is a polynomial with small coefficients. For further details, see [5].

In general, a computation $a(X)b(X)$ in the ring R is a convolution product of the vectors of coefficients of a and b. The naive algorithm to compute this convolution is N^2 steps, where each step is an addition and a multiplication. (If $a(X)$ has coefficients that are randomly distributed in $\{-1,0,1\}$, then the computation takes about $2N/3$ steps, where now a step is simply an addition or a subtraction.) Other methods such as Karatsuba multiplication or FFT techniques (if applicable) reduce this to $O(N\log N)$ steps, although the big-O constant may be moderately large.

Applying the ideas already described above, we can compute a small random multiple of $h(X)$ as a product

$$r^{(1)}(X)r^{(2)}(X)\cdots r^{(t)}(X)h(X),$$

where each $r^{(i)}(X)$ has only a few nonzero terms. Then the amount of computation needed is proportional to the sum of the number of nonzero terms, while the size of the sample space is approximately equal to the product of the sample spaces for the $r^{(i)}$.

We illustrate with a specific example. Let $N = 251$ and take

$$r(X) = r^{(1)}(X)r^{(2)}(X),$$

where $r^{(1)}$ and $r^{(2)}$ are polynomials with exactly eight nonzero coefficients, four 1's and four -1's. To avoid too much duplication, we also require that $r^{(i)}(0) = 1$, so only three of the 1's are randomly placed. Then the number of such $r(X)$ polynomials is approximately

$$\binom{250}{3}\binom{247}{4}\cdot\binom{250}{3}\binom{247}{4}\cdot\frac{1}{2}\approx 2^{95.94}.$$

If one tries to guess $r^{(1)}(X)$ and then use a square root search for $r^{(2)}(X)$, this leads to a search algorithm of length approximately

$$\binom{250}{3}\binom{247}{4}\cdot\sqrt{\binom{250}{3}\binom{247}{4}}\cdot\frac{1}{2}\approx 2^{71.1}.$$

The computation of the product $r(X)h(X)$ is reduced to approximately $16N$ additions and subtractions. Notice that $r(X)$ itself has about 64 nonzero coefficients, so a direct computation of $r(X)h(X)$ requires almost 4 times as many elementary operations.

A similar construction can be used for the NTRU private key $f(X)$, leading to a similar computational speedup for decryption.

5 Randomness of Small Hamming Weight Products

In this section we examine the question of the extent to which a product of small Hamming weight elements may be considered random. There are many ways of measuring randomness (see for example [6]). We will consider the products described in Section 4 and will study products of low Hamming weight polynomials. For concreteness, let

$$\mathcal{B}_N(D) = \{\text{binary polynomials of degree } N-1 \text{ with } D \text{ ones}\}.$$

That is, elements of $\mathcal{B}_N(D)$ are polynomials

$$a_0 + a_1X + a_2X^2 + \cdots + a_{N-1}X^{N-1}$$

with $a_i \in \{0,1\}$ and $\sum a_i = D$. As described in Section 4, we multiply polynomials using the convolution rule $X^N = 1$.

Products of polynomials are subject to a natural rotation of their coefficients by multiplying by powers of X. In other words, we can rewrite any product as

$$a(X) * b(X) = (X^k * a(X)) * (X^{N-k} * b(X)).$$

Such rotations are far from random, so it makes sense to discourage them in our sample spaces. We thus define

$$\mathcal{B}_N^*(D) = \{a(X) = a_0 + a_1X + \cdots + a_{N-1}X^{N-1} \in \mathcal{B}_N(D) : a_0 = 1\}$$

to be the subset of $\mathcal{B}_N(D)$ consisting of polynomials whose constant coefficient is nonzero.

We wish to compare the space of random binary polynomials $\mathcal{B}_N^*(D)$ with the space of products

$$\mathcal{P}_N^*(d_1,d_2) = \{c(X) = a(X) * b(X) : \\ a(X) \in \mathcal{B}_N^*(d_1),\ b(X) \in \mathcal{B}_N^*(d_2),\ c(X) \in \mathcal{B}_N^*(d_1d_2)\}.$$

Notice that we are only considering polynomials $a(X)$ and $b(X)$ whose product $a(X) * b(X)$ is binary. In practice, this may mean generating a number of pairs (a, b) at random, multiplying them, and discarding the product if it is not of the appropriate form. (This is what we did in our experiments.)

How might one compare the set of products $\mathcal{P}_N^*(d_1, d_2)$ with the truly random set $\mathcal{B}_N^*(d_1 d_2)$? In general, the former set will be much smaller than the latter set, so we cannot say that each element of $\mathcal{B}_N^*(d_1 d_2)$ is equally likely to be hit by an element of $\mathcal{P}_N^*(d_1, d_2)$. Experimentally, we can say that elements of $\mathcal{P}_N^*(d_1, d_2)$ generally have a unique representation as a product, but this is only a weak measure of randomness. So we will use Hamming weight differences to study the extent to which elements of $\mathcal{P}_N^*(d_1, d_2)$ are randomly distributed in the space $\mathcal{B}_N^*(d_1 d_2)$. For any two binary polynomials $a(X)$ and $b(X)$, we define their *Hamming weight difference* to be

$$\mathrm{HWD}(a, b) = \#\{i : a_i \neq b_i\}.$$

It is easy to compute the probability that a randomly chosen pair in $\mathcal{B}_N^*(D)$ will have a given Hamming weight difference. More precisely, for any fixed $a \in \mathcal{B}_N^*(D)$, if we ignore the known constant coefficient, there are $D-1$ ones and $N-D$ zeros. Suppose that $b \in \mathcal{B}_N(D)$ has k of its ones in common with the ones of a. Then $\mathrm{HWD}(a, b)$ gains $D-1-k$ from the ones in a that are hit by zeros of b and it gains $D-1-k$ from the ones of b that hit zeros of a, so $\mathrm{HWD}(a, b) = 2(D-1-k)$. Thus the Hamming weight difference is always even and it will equal $2*h$ when exactly $D-1-h$ of the ones of a and b coincide. Dividing the number of ways that this can happen by the total number of polynomials, we find that for a fixed $a \in \mathcal{B}_N^*(D)$, the probability that a randomly chosen $b \in \mathcal{B}_N^*(D)$ is Hamming weight distance $2*h$ from a is given by

$$\operatorname*{Prob}_{b \in \mathcal{B}_N^*(D)}(\mathrm{HWD}(a, b) = 2*h) = \frac{\binom{D-1}{D-1-h}\binom{N-D}{h}}{\binom{N-1}{D-1}}. \tag{3}$$

It seems more difficult to compute exactly the analogous probability for a randomly chosen $b \in \mathcal{P}_N^*(d_1, d_2)$, so we resort to computer simulation. We performed the following experiments. We randomly chose 10,000 polynomials from the sets

$$\mathcal{B} = \mathcal{B}_{251}^*(64) \qquad \text{and} \qquad \mathcal{P} = \mathcal{P}_{251}^*(8, 8).$$

We computed the distributions of Hamming weight differences $\mathrm{HWD}(a, b)$ for all 10^8 pairs (a, b) chosen from each of the sets $\mathcal{B} \times \mathcal{B}$, $\mathcal{B} \times \mathcal{P}$, and

$\mathcal{P} \times \mathcal{P}$. The results are listed in Table 1, together with the theoretical expected value from the formula (3). It seems clear from the table that there is no discernable difference in $\mathrm{HWD}(a, b)$ in the various situations studied.

Hamming Wt	Experimental			Theoretical
Difference	$\mathcal{B}$ vs $\mathcal{B}$	$\mathcal{B}$ vs $\mathcal{P}$	$\mathcal{P}$ vs $\mathcal{P}$	$\mathcal{B}$ vs $\mathcal{B}$
68	0.002%	0.002%	0.002%	0.002%
70	0.006%	0.006%	0.006%	0.006%
72	0.019%	0.019%	0.020%	0.019%
74	0.057%	0.057%	0.058%	0.057%
76	0.155%	0.155%	0.158%	0.155%
78	0.380%	0.380%	0.381%	0.379%
80	0.841%	0.843%	0.841%	0.841%
82	1.692%	1.691%	1.688%	1.691%
84	3.079%	3.082%	3.074%	3.080%
86	5.072%	5.071%	5.063%	5.072%
88	7.542%	7.539%	7.537%	7.545%
90	10.121%	10.118%	10.123%	10.124%
92	12.234%	12.228%	12.236%	12.229%
94	13.265%	13.271%	13.287%	13.270%
96	12.904%	12.901%	12.917%	12.901%
98	11.209%	11.208%	11.200%	11.203%
100	8.660%	8.657%	8.653%	8.658%
102	5.928%	5.929%	5.923%	5.928%
104	3.578%	3.581%	3.572%	3.578%
106	1.889%	1.891%	1.888%	1.892%
108	0.867%	0.869%	0.868%	0.869%
110	0.343%	0.343%	0.345%	0.344%
112	0.115%	0.115%	0.117%	0.116%
114	0.032%	0.033%	0.033%	0.033%
116	0.007%	0.008%	0.008%	0.008%
118	0.001%	0.001%	0.001%	0.001%

Table 1. Hamming weight difference probabilities

6 A General Formulation of Small Hamming Weight Products

All of the above constructions can be formulated quite generally in terms of a ring R, an R-module M, and a subset $S \subset R$ with the two properties:

(A) The set S is "sufficiently large."

(B) The computation of products $r \cdot m$ for $r \in S$ and $m \in M$ is "computationally easy."

These conditions are, to some extent, antagonistic to one another, since presumably the larger the set S, the harder on average it is to compute products $r \cdot m$ for $r \in S$.

One way to construct the set S is to choose a collection of smaller subsets $S_1, \ldots, S_t \subset R$ and let

$$S = \{r_1 \cdots r_t \,:\, r_1 \in S_1, \ldots, r_t \in S_t\}.$$

Under suitable hypotheses, the size of the set S is approximately the product of the sizes of $S_1, \ldots, S_t$. (See the remark below.) Obviously we want each S_i to satisfy condition (B).

Proceeding further, suppose that there is one particular element $\tau \in R$ such that the product $\tau \cdot m$ is easy to compute for every $m \in M$. Then a natural choice for the S_i are low Hamming weight polynomials in τ; that is, S_i might consist of all elements of r of the form

$$\tau^{j_1} + \tau^{j_2} + \cdots + \tau^{j_d}$$

for some fixed $d = d_i$ (or for some random $d \le D_i$ for a fixed D_i). Of course, if it is easy to compute inverses $-m$, then one can increase the size of S_i by using

$$\pm\tau^{j_1} \pm \tau^{j_2} \pm \cdots \pm \tau^{j_d}.$$

Similarly, if there are several easy-to-multiply elements $\tau_1, \ldots, \tau_u \in R$, then one can take low Hamming weight polynomials in the u "variables" $\tau_1, \ldots, \tau_u$, further increasing the size of the special sets S_i.

We can relate this general formulation to the earlier examples in this note.

1. **Powers in $\mathbb{F}_{2^n}$** The ring is $R = \mathbb{Z}$, the R-module is the multiplicative group $M = \mathbb{F}_{2^n}^*$, and the special map τ is the doubling (i.e., squaring) map $\tau(\alpha) = \alpha^2$.
2. **Multiples on Koblitz Curves** The ring is $R = \mathrm{End}(E(\mathbb{F}_{2^m}))$ (i.e., the ring of homomorphism from $E(\mathbb{F}_{2^m})$ to itself), the R-module is $M = E(\mathbb{F}_{2^m})$, and the special map τ is the Frobenius map $\tau(x, y) = (x^2, y^2)$.
3. **The NTRU Cryptosystem** The ring is $R = \mathbb{Z}[X]/(X^N - 1)$, the R-module is $M = R$ (i.e., R acts on itself via multiplication), and the special map τ is the multiplication-by-X map $\tau(f(X)) = Xf(X)$.

This makes clear how Small Hamming Weight Products apply to these particular situations and also gives some indication of the widespread applicability of the general idea.

Remark 7. We partially quantify our remark concerning the size of S, where the set S is the image of the map

$$S_1 \times S_2 \times \cdots \times S_t \longrightarrow R, \qquad (r_1, r_2, \ldots, r_t) \longmapsto r_1 r_2 \cdots r_t.$$

In practice, it is usually not hard to describe a natural set $T \subset R$ with the property that $S \subset T$ and with the property that a random t-tuple $(r_1, \ldots, r_t)$ of $S_1 \times \cdots \times S_t$ appears to have an equal chance of hitting each element of T. (Note our choice of words. It may be difficult to rigorously prove that T has this property, but usually one can at least obtain experimental evidence.) We write $N_i = |S_i|$ for the size of the set S_i and $M = |T|$ for the size of the set T. Then elementary probability theory allows us to estimate the expected number of distinct elements if we randomly choose $N_1 N_2 \cdots N_t$ elements of T with replacement.

References

1. H. Cohen, A Course in Computational Number Theory, GTM 138, Springer-Verlag, 1993.
2. D. Coppersmith, G. Seroussi, On the Minimum Distance of Some Quadratic Residue Codes, IEEE Transactions on Information Theory, vol. IT-30, No. 2, March 1984, 407–411.
3. D. Gordon, A survey of fast exponentiation methods, *Journal of Algorithms* **27** (1998), 129–146.
4. D. Hankerson, J.L. Hernandez, A. Menezes, Software implementation of elliptic curve cryptography over binary fields, in Cryptographic Hardware and Embedded Systems (CHES 2000), Ç. Koç and C. Paar (eds.), Lecture Notes in Computer Science, Springer-Verlag, to appear.
5. J. Hoffstein, J. Pipher, J.H. Silverman, NTRU: A new high speed public key cryptosystem, in Algorithmic Number Theory (ANTS III), Portland, OR, June 1998, Lecture Notes in Computer Science 1423 (J.P. Buhler, ed.), Springer-Verlag, Berlin, 1998, 267–288.
6. D. Knuth, The Art of Computer Programming, Volume 2, Seminumerical Algorithms, 3rd ed., Addison-Wesley, 1998.
7. C.H. Lim and P.J. Lee, Sparse RSA keys and their generation, preprint, 2000.
8. A.J. Menezes and P.C. van Oorschot and S.A. Vanstone. *Handbook of Applied Cryptography*, CRC Press, 1997.
9. P. de Rooij, On the security of the Schnorr scheme using preprocessing, in Advances in Cryptology (Eurocrypt 90), Aarhus, Denmark, May 1990, Lecture Notes in Computer Science 473 (I.B. Damgard, ed.), Springer-Verlag, Berlin, 1990, 71–80.

10. C.P. Schnorr, Efficient identification and signatures for smart cards, in Advances in Cryptology (Crypto 89), Santa Barbara, CA, August 1989, Lecture Notes in Computer Science 435, (G. Brassard, ed.), Springer-Verlag, Berlin, 1989, 239–252.
11. J. Solinas, Efficient arithmetic on Koblitz curves, *Designs, Codes, and Cryptography* **19** (2000), 195–249.
12. D. Stinson, *Cryptography: Theory and Practice.* CRC Press, 1997.
13. D.R. Stinson, Some baby-step giant-step algorithms for the low Hamming weight discrete logarithm problem, Mathematics of Computation, to appear.