GAUSS PERIODS: ORDERS
AND CRYPTOGRAPHICAL APPLICATIONS

SHUHONG GAO, JOACHIM VON ZUR GATHEN AND DANIEL PANARIO

Abstract. Experimental results on the multiplicative orders of Gauss periods in finite fields are presented. These results indicate that Gauss periods have high order and are often primitive (self-dual) normal elements in finite fields. It is shown that Gauss periods can be exponentiated in quadratic time. An application is an efficient pseudorandom bit generator.

1. Introduction

\( F_q \) denotes a finite field with \( q \) elements. Let \( n \) and \( k \) be positive integers such that \( r = nk + 1 \) is a prime, not dividing \( q \), and \( K \) the unique subgroup of order \( k \) of the multiplicative group of \( \mathbb{Z}_r = \mathbb{Z}/r\mathbb{Z} \). For any primitive \( r \)-th root \( \beta \) of unity in \( F_q^{*} \), the element

\[ \alpha = \sum_{a \in K} \beta^a \]

is a Gauss period of type \((n,k)\) over \( F_q \). It is easy to see that \( \alpha \in F_q^{*} \).

Adleman & Lenstra (1986) and Mullin et al. (1988) used Gauss periods to construct field extensions and normal bases with special properties over finite fields. A normal basis for \( F_q^n \) over \( F_q \) is a basis of the form \( \alpha, \alpha^q, \ldots, \alpha^{q^{n-1}} \) generated by some \( \alpha \in F_q^{*} \). Any such \( \alpha \) is called a normal element.

A Gauss period of type \((n,k)\) over \( F_q \) generates a normal basis for \( F_q^n \) over \( F_q \) if and only if gcd\((e,n) = 1\), where \( e \) denotes the index of \( q \) modulo \( r = nk + 1 \) (Wassermann 1990, 1993, Gao et al. 1995). Gao et al. (1995) present a method for fast multiplication and division under the normal bases generated by Gauss periods; thus exponentiation in finite fields can be sped up. We refer to that paper and the books by Jungnickel (1993) and Menezes et al. (1993) for a discussion of the literature.

Gauss periods of type \((n,2)\) over \( F_2 \) also have other remarkable properties. Gao & Vanstone (1995) proved that they can be exponentiated in

Date: November 11, 1997.
1991 Mathematics Subject Classification. Primary 11T30, 94A60; Secondary 11Y16, 12Y05, 68Q25.

Key words and phrases. Finite fields, primitive elements, normal bases, cryptography, pseudorandom bit generators.

This paper is in final form, no version of it will be submitted for publication elsewhere.
Fast exponentiation of Gauss periods

In this section we show that, for fixed \( k \) and \( q \), a Gauss period of type \((n,k)\) can be exponentiated in \( O(n^2) \) operations in \( \mathbb{F}_q \); see also Gao et al. (1995).

A pair \((n,k)\) is a Gauss pair over \( \mathbb{F}_q \) if \( r = nk + 1 \) is a prime not dividing \( q \) and \( \gcd(e,n) = 1 \), where \( e \) is the index of \( q \) modulo \( r \), i.e., \( e = nk / \text{ord}_r(q) \). When \( q \) is understood, we simply say that \((n,k)\) is a Gauss pair. It is always assumed that \((n,k)\) is a Gauss pair in the sequel.

In the notation of the introduction, \( K \) is the unique subgroup of order \( k \) of \( \mathbb{Z}_r^\times \), and

\[
a = \sum_{\alpha \in K} \beta^a,
\]

where \( \beta \) is a primitive \( r \)-th root of unity. Let

\[
K_i = q^i K = \{aq^i \mod r : a \in K \} \text{ for } 0 \leq i < n.
\]
Then \( \mathbb{Z}^* \) is the disjoint union of \( \mathcal{K}_0, \mathcal{K}_1, \cdots, \mathcal{K}_{n-1} \). We write \( \alpha_i = \alpha_i^\omega \) for \( 0 \leq i < n \). Then \( (\alpha_0, \alpha_1, \cdots, \alpha_{n-1}) \) is a normal basis for \( \mathbb{F}_q^n \) over \( \mathbb{F}_q \). It is shown in Gao et al. (1995) that

\[
a \cdot \alpha_i = \delta_i k + \sum_{0 \leq j < n} t_{ij} \alpha_j
\]

where \( t_{ij} = |(1 + \mathcal{K}_i) \cap \mathcal{K}_j|, \delta_i = 1 \) if \( i = i_0 \) and \( 0 \) otherwise, and \( i_0 \) is such that \(-1 \in \mathcal{K}_{i_0}\). Note that each \( a \cdot \alpha_i \) has at most \( k \) nonzero terms, and thus there are at most \( nk \) nonzero terms in total. We store all the nonzero \( t_{ij} \) in a table, called the multiplication table of Gauss periods of type \((n, k)\).

**Theorem 2.1.** Let \( \alpha \) be a Gauss period of type \((n, k)\) over \( \mathbb{F}_q \) and \( 0 \leq e < q^n \). Then \( \alpha^e \) can be computed in \( O(n^2 qk) \) operations in \( \mathbb{F}_q \).

**Proof.** We want to compute \( \alpha^e \) expressed in the normal basis \((\alpha_0, \alpha_1, \cdots, \alpha_{n-1})\). We use a redundant representation of \( \gamma \in \mathbb{F}_q^n \), writing

\[
\gamma = \left( \sum_{0 \leq i < n} a_i \alpha_i \right) + a_n
\]

where \( a_0, \cdots, a_{n-1}, a_n \in \mathbb{F}_q \). Thus \( \gamma \) is represented by an \((n + 1)\)-tuple \((a_0, \cdots, a_{n-1}, a_n)\). Of course, this representation is not unique; we just want to compute any one of the representations. For example, the unit \( 1 \) can either be represented as \((0, \cdots, 0, 1)\) or \((-1, \cdots, -1, 0)\), since \(-1 = \sum_{0 \leq i < n} a_i\). Our algorithm benefits from this flexibility.

Thus \( \gamma^e \) is the \((n + 1)\)-tuple obtained from that of \( \gamma \) by shifting cyclically the first \( n \) coordinates to the right by one position (the last coordinate remains fixed). So the cost for computing a \( q \)th power is negligible.

For any \( \gamma = \left( \sum_{0 \leq i < n} a_i \alpha_i \right) + a_n \in \mathbb{F}_q^n \) and \( 0 \leq j < n \),

\[
a_j \gamma = \left( \sum_{0 \leq i < n} a_i (\alpha a_{i-j})^{e_j} \right) + a_n \alpha_j.
\]

Since \( \alpha a_i \) is a sum of at most \( k \) terms and can be looked up from the multiplication table, \( \alpha_j \gamma \) can be computed in \( O(nk) \) operations in \( \mathbb{F}_q \).

Now to compute \( \alpha^e \), we use the \( q \)-ary representation \( e = \sum_{0 \leq j \leq \ell} e_j q^j \), with \( 0 \leq e_j < q \) for all \( j \) and \( e_\ell \neq 0 \). Then \( \ell < n \), and

\[
\alpha^e = \prod_{0 \leq j \leq \ell} (\alpha_j)^{e_j} = \prod_{0 \leq j \leq \ell} \alpha_j^{e_j}.
\]

This suggests that we compute \( \alpha^e \) iteratively. Initially, set \( \gamma := 1 \). For \( j \) from \( 0 \) to \( \ell \) set \( \gamma := \alpha_j^{e_j} \gamma \). Then, at the end, we have \( \gamma = \alpha^e \). We compute \( \alpha_j^{e_j} \gamma \) iteratively as \( \alpha_j^{e_j} \gamma \) for \( i \) from \( 1 \) to \( e_j \). This algorithm computes \( \alpha^e \) in

\[
O \left( \left( \sum_{0 \leq j < n} e_j \right) nk \right) = O(\sigma_q(e)nk)
\]
operations in \( \mathbb{F}_q \), where \( \sigma_q(e) \) is the sum of digits of \( e \) in \( q \)-ary representation. Now \( \sigma_q(e) \leq (q - 1)n < qn \) implies the claim. \( \square \)

Thus \( \alpha^e \) can be computed in \( O(n^2) \) operations in \( \mathbb{F}_q \), when the values of \( q \) and \( k \) are fixed. This is faster than any known algorithm for exponentiation of an arbitrary element. Furthermore, one needs only to store \( e, \gamma \) and the multiplication table, a total of \( O(nk) \) elements of \( \mathbb{F}_q \).

Exponentiation of an arbitrary element in \( \mathbb{F}_{q^n} \) (with \( q \) bounded) can be performed with \( O(n^2 \log \log n) \) operations in \( \mathbb{F}_q \) by the currently fastest algorithm, with storage for \( O(n/\log^2 n) \) elements in \( \mathbb{F}_q^n \) (Shoup 1994, Gao et al. 1995).

Recently, von zur Gathen & Pappalardi (1996) proved under ERH that, for any fixed \( k \) and \( q \), there are infinitely many values of \( n \) such that \( (n, k) \) is a Gauss pair over \( \mathbb{F}_q \). In fact, they determine a positive density for the primes \( nk + 1 \), where \( (n, k) \) is a Gauss pair, in the set of all primes. Thus, there are infinitely many fields \( \mathbb{F}_{q^n} \) in which Gauss periods can be exponentiated easily.

3. Experimental results

In this section, we present experimental results which indicate that Gauss periods almost always have high order. When \( (n, k) \) is a Gauss pair over \( \mathbb{F}_q \), Gauss periods of type \( (n, k) \) are algebraic conjugates of each other, hence they have the same multiplicative order. That is, the order of \( \alpha = \sum_{\beta \in \mathbb{F}_q} \beta^a \) does not depend on the choice of the primitive \( r \)-th root of unity. By the algorithm in the previous section, \( \alpha \) can exponentiated under the normal basis generated by \( \alpha \) itself without knowing \( \beta \), one just needs to precompute the multiplication table for Gauss periods of type \( (n, k) \). We computed the multiplicative orders of \( \alpha \) for all Gauss pairs \( (n, k) \) over \( \mathbb{F}_2 \) with \( 3 \leq k \leq 20 \) and \( 2 \leq n < 569 \), and also did the corresponding calculations for \( 569 \leq n \leq 1200 \) as far as current knowledge of the factors of \( 2^n - 1 \) permits. The results are tabulated in the table of the microfiche supplement, where “Ind” denotes index, which equals \( 2^n - 1 \) divided by the corresponding multiplicative order. An entry with a question mark “?" in the “Ind” column means that the corresponding index was computed from the partial factorization of \( 2^n - 1 \) known to the authors at the time of writing. Thus the true index is \( i \) times some of the unknown prime factors of \( 2^n - 1 \); we believe that these extra factors are unlikely to occur.

Our experiments show that Gauss periods have the expected multiplicative properties: they almost always have high multiplicative orders and are frequently primitive. More precisely, in the range \( 2 \leq n < 569 \) and \( 2 \leq k \leq 20 \) there are 1267 Gauss pairs \( (n, k) \), all the corresponding Gauss periods have order \( \geq (2^n - 1)/n \) except for 8 pairs, and 977 of them are primitive. In the range \( 569 \leq n \leq 1200 \) and \( 2 \leq k \leq 20 \), there are 1151 Gauss pairs \( (n, k) \), and the corresponding Gauss periods have order \( \geq (2^n - 1)/n \) except for 5 pairs, and 894 of them are primitive, provided that the corresponding index entries “?” are the true indices.
All the Gauss periods in the table generate normal bases over $\mathbb{F}_2$. When the index is 1, the corresponding basis is a primitive normal basis. Thus Gauss periods yield many primitive normal bases over $\mathbb{F}_2$. Also, when $k$ is even, the normal basis generated by a Gauss period of type $(n, k)$ over $\mathbb{F}_2$ is self-dual (Gao et al. 1995). We see that Gauss periods generate many primitive self-dual normal bases as well.

Gao & Vanstone (1995) observe that if $n$ and $2n + 1$ are both primes then Gauss periods of type $(n, 2)$ are primitive elements in $\mathbb{F}_{2^n}$ for $n \leq 1200$. Our experimental data show that their observation still holds for general Gauss periods. It is formulated as follows.

**Conjecture 3.1.** If $n$ and $nk + 1$ are both primes and $k < \log_2(n + 1)$, then Gauss periods of type $(n, k)$ form a primitive normal basis for $\mathbb{F}_{2^n}$ over $\mathbb{F}_2$.

We note that normality is not a problem here. Since the order $m$ of 2 modulo $nk + 1$ is at least $\log_2(nk + 1) \geq \log_2(n + 1) > k$, it follows that $n$ divides $m = nk/e$ and thus $\gcd(e, n) = 1$.

Wassermann (1993) proves that for a given $n$ there exists a Gauss pair $(n, k)$ over $\mathbb{F}_2$ if and only if $8 \nmid n$. There are 61 values of $n \leq 1200$ with $8 \nmid n$ for which there is no Gauss pair $(n, k)$ with $k \leq 20$. For each of these $n$, we list in the table the smallest Gauss pair $(n, k)$ and the corresponding index (or index?)

Our computations lead us to believe that for every $n$ not divisible by 8 there is a Gauss pair $(n, k)$ yielding a primitive Gauss period $\mathbb{F}_{2^n}$ over $\mathbb{F}_2$. As an experiment, we verified this for all $n < 569$. If there is no primitive Gauss period for $k \leq 20$, then the last entry of $k$ is the smallest $k$ whose Gauss period is primitive and normal in $\mathbb{F}_{2^n}$. The largest such $k$ occurs in (490, 69).

**Conjecture 3.2.** For any positive integer $n$ not divisible by 8, there exists an integer $k \geq 1$ such that the Gauss period of type $(n, k)$ is primitive normal in $\mathbb{F}_{2^n}$ over $\mathbb{F}_2$.

Motivated by this work and the previous work of Gao & Vanstone (1995), von zur Gathen & Shparlinski (1995) prove that Gauss periods of type $(n, 2)$ have order at least $2\sqrt{2n-2}$.

Next, we show how to construct primitive elements from elements of high order. The constructed primitive elements will still be essentially as easy to exponentiate as Gauss periods.

**Theorem 3.3.** Let $\alpha \in \mathbb{F}_{2^n}$ with index $e$. A primitive element can be constructed from a deterministically in time polynomial in $e$ and $n$.

**Proof.** The order of $\alpha$ is $m = (2^n - 1)/e$. Write $e = e_1 e_2$, where $\gcd(e_2, m) = 1$ and every prime divisor of $e_1$ divides $m$. Let $\beta_1 \in \mathbb{F}_{2^n}$ satisfy

$$\beta_1^{e_1} = \alpha,$$

and let $\beta_2$ be a primitive $e_2$th root of unity in $\mathbb{F}_{2^n}$. Then $\beta_1$ has order $me_1$, and $\beta_1 \beta_2$ has order $me_1 e_2 = 2^n - 1$, as $\gcd(me_1, e_2) = 1$. This means
that \( \beta_1 \beta_2 \) is primitive in \( \mathbb{F}_{2^n} \). Also, \( \beta_1 \) and \( \beta_2 \) can be constructed in time polynomial in \( e \) and \( n \).

Thus if the order of \( \alpha \) is at least \((2^n - 1)/n^c\) for a constant \( c \), then a primitive element in \( \mathbb{F}_{2^n} \) can be constructed in polynomial time \( n^{O(1)} \). In the special case \( e = 2^k - 1 \), the equation \( x^e - \alpha \) can be written as \( x^{2^k} = \alpha x \), which corresponds to a system of linear equations over \( \mathbb{F}_2 \), and can be solved by any efficient algorithm for linear equations. Our experimental data shows that many Gauss periods have indices \( \frac{n}{2^k}; \ldots; \frac{n}{1} \), etc., which are of the form \( 2^k - 1 \). Thus from our table of Gauss periods, it is easy to construct primitive elements if one really needs primitive elements instead of elements of high orders. In the table, we give for each \( n < 569 \) and \( 8 \nmid n \), the smallest \( k \) such that a Gauss period of type \((n, k)\) has index at most \( n \). One can see that for these \( n \) it is possible to find a reasonably small such \( k \).

Finally, we make some comments on the related work in the literature. As mentioned in the introduction, our work also contributes to the construction of primitive polynomials and primitive normal polynomials, since their irreducible polynomials are normal and primitive when Gauss periods are primitive. Hansen & Mullen (1992) and Morgan & Mullen (1994) give tables of primitive polynomials and primitive normal polynomials of degree \( m \) over \( \mathbb{F}_p \) for all prime powers \( p^m \leq 10^{50} \) with \( p \leq 97 \). Zivković (1994a, 1994b) gives a more extensive table of primitive polynomials of degree \( m \leq 1200 \) (and a few values of \( m \) between 1200 and 5000) over \( \mathbb{F}_2 \) when the factorization of \( 2^m - 1 \) is known. In their work, they search for sparse polynomials, i.e., those with the smallest number of nonzero terms. Such polynomials are useful in efficient implementation of feedback shift registers. Gao & Panario (1997) provide a construction of infinite families of sparse irreducible polynomials. In the extreme case, there is much interest in constructing irreducible trinomials over \( \mathbb{F}_2 \). Zierler & Brillhart (1968, 1969) give a table of irreducible trinomials of degree \( \leq 1000 \). Blake et al. (1994) extend this list to all irreducible trinomials of degree \( \leq 2000 \) over \( \mathbb{F}_2 \) (and a table for degree \( \leq 5000 \) is available from those authors).

It is, however, not clear how primitive elements from sparse polynomials can be exponentiated faster than an arbitrary primitive element. The primitive polynomials from Gauss periods may not in general be sparse, but they do provide computational advantage in fast exponentiation as shown by Theorem 2.1.

4. Cryptographical applications

Let \( \alpha \in \mathbb{F}_{2^n} \) be a primitive element (or an element of high order, say at least \((q^n - 1)/n^c\) for some constant \( c \)). Computing the exponentiation function that maps \( x \in \{0, \ldots, q^n - 1\} \) to \( \alpha^x \) is easy, but computing its inverse function, i.e., computing \( x \) given \( \alpha^x \), called the discrete logarithm problem, is believed to be hard in general. This one-wayness of exponentiation has found many applications in public-key cryptography: Diffie-Hellman key
ORDERS OF GAUSS PERIODS

exchange (Diffie & Hellman 1976), password schemes (Lamport 1981), ElGamal cryptosystem (ElGamal 1985), cryptosystems over \( \mathbb{F}_2 \) (Agnew et al. 1991, Agnew et al. 1993), smart cards (Beth 1988, Schnorr 1990, 1991), US Digital Signature Algorithm (NIST 1994), pseudorandom bit generators (Blum & Micali 1984, Long & Wigderson 1988). Pseudorandom bit generators based on discrete logarithms in finite fields are used by Zheng & Seberry (1992, 1993) and Lim & Lee (1993) to construct cryptosystems that leak no partial information and are secure against adaptively chosen ciphertext attacks. In these applications, one needs a fixed element of high order and computes \( \alpha^t \) for many random large integers \( t \). For example, in a signature scheme, for each signature one needs to generate a random integer \( t \) and compute \( \alpha^t \). Sometimes the computing power of the signature generating device is limited, e.g., in a smart card. So \( \alpha \) has to be chosen such that exponentiation of \( \alpha \) is easy. In practice, the currently popular choice for \( \alpha \) is from elements in \( \mathbb{F}_p \). If the prime \( p \) has \( n \) bits, then computing \( \alpha^t \) needs \( O(n^3) \) bit operations using repeated square and multiply method, or \( O(n^2 \log n \log \log n) \) bit operations using FFT-based fast multiplication algorithms. However, if we choose \( \alpha \) to be a Gauss period of type \( (n,k) \) in \( \mathbb{F}_2^m \) for a small \( k \) then the cost of exponentiating \( \alpha \) is reduced to \( O(n^2) \) bit operations, which is just the cost of one multiplication by the "classical" method. Our experimental results show that \( \alpha \) almost always has high order and is often primitive. Our exponentiation algorithm is also easy to implement. Gauss periods are therefore highly attractive in these applications.

In the following, we describe Blum & Micali’s pseudorandom bit generator based on exponentiation in \( \mathbb{F}_p \), then we adapt it to the fields \( \mathbb{F}_2^m \) and comment on its security and efficiency.

A pseudorandom bit generator produces sequences of bits (0 or 1) that cannot be distinguished from truly random sequences of bits of equal length by any (probabilistic) polynomial time algorithm. Blum & Micali (1984) presented the following pseudorandom bit generator. Let \( m \geq 1 \) be a fixed integer. Given \( n \geq 2 \), select an odd prime \( p \) of \( n \) bits, and a primitive root \( \alpha \) modulo \( p \). Pick a random integer \( a_0 \) (the seed) in the range \( 1 < a_0 < p - 1 \). Set

\[
a_{k+1} \equiv \alpha^a_{k} \mod p \text{ for } k \geq 0,
\]

and

\[
b_{k+1} = 1 \text{ if } a_{k+1} > (p-1)/2, \text{ and } b_{k+1} = 0 \text{ otherwise}.
\]

Then \( \{b_k : 1 \leq k \leq n^m + m\} \) is a sequence of \( n^m + m \) bits generated from the \( n \)-bit seed \( a_0 \). Blum & Micali (1984) proved that if the discrete logarithm problem in \( \mathbb{F}_p \) is hard then this much longer sequence of bits is pseudorandom. Blum & Micali’s generator outputs only one bit at each iteration, i.e., each bit costs one exponentiation \( \text{mod } p \). Long & Wigderson (1988) extended this to output about \( \log n \) bits at each iteration.

We now adapt the above generator to the fields \( \mathbb{F}_2^m \). Under a fixed basis for \( \mathbb{F}_2^m \) over \( \mathbb{F}_2 \), an element \( x \in \mathbb{F}_2^m \) is represented as a sequence of \( n \) bits (0 and 1). We use \( \mathfrak{f} \) to denote the integer whose binary representation is
the same as \( x \). Let \( \alpha \in \mathbb{F}_{2^n} \) be a primitive element (or an element of high order). Pick a random element \( x_0 \in \mathbb{F}_{2^n} \). Set
\[
x_{k+1} = \alpha^{x_k} \text{ for } k \geq 0,
\]
and let \( z_{k+1} \) be the least significant bit
\[
z_{k+1} = \overline{x_{k+1} \mod 2}.
\]
(1)
Then \( \{z_k : 1 \leq k \leq n^m + m\} \) is a sequence of bits generated from the seed \( x_0 \). We want to show that this sequence is pseudorandom.

Let \( f \) be a one-way function, i.e., it is easy to compute but hard to invert. A Boolean predicate \( B \) (i.e., \( B(x) = 0 \) or 1) is said to be hard for \( f \) if an oracle for \( B(f(x)) \) allows one to invert \( f \) easily. Blum & Micali (1984, Theorem 2) proved that if \( B \) is a hard predicate for a one-way function \( f \) then the following sequence is pseudorandom:
\[
B(f(a_0)), B(f^2(a_0)), B(f^3(a_0)), \ldots, B(f^k(a_0)), \ldots
\]
where \( a_0 \) is randomly chosen. To show that our sequence is pseudorandom, it is enough to prove Theorem 4.1 below. For any \( \beta \in \mathbb{F}_{2^n} \), the smallest positive integer \( x \) such that \( \beta = \alpha^x \) is called the discrete logarithm of \( \beta \) with respect to \( \alpha \), denoted by \( \log_{\alpha} \beta \). If no such \( x \) exists, we set (arbitrarily) \( \log_{\alpha} \beta = 0 \).

**Theorem 4.1.** Every bit of the discrete logarithm in \( \mathbb{F}_{2^n} \) is a hard predicate for the exponentiation function.

**Proof.** Let \( B(x) \) be the least significant bit of an integer \( x \). We show how to compute discrete logarithms via an oracle for \( B(\log_{\alpha} \beta) \), which returns the least significant bit of the discrete logarithm of any \( \beta \in \mathbb{F}_{2^n} \). Note that for any integer \( i \), \( \log_{\alpha} \beta 2^i \equiv 2^i \log_{\alpha} \beta \mod 2^n - 1 \). Suppose that
\[
\log_{\alpha} \beta = \sum_{k=0}^{n-1} a_k 2^k = (a_0, a_1, \ldots, a_{n-1})_2.
\]
Then for \( i \in \mathbb{N} \)
\[
2^i \log_{\alpha} \beta \equiv \sum_{k=0}^{n-1} a_k 2^{k+i} \equiv \sum_{k=0}^{n-1} a_{k-i} 2^k \mod 2^n - 1,
\]
where the subscripts of \( a \) are computed modulo \( n \), and
\[
\log_{\alpha} \beta 2^i = (a_{n-i}, \ldots, a_{n-1}, a_0, \ldots, a_{n-i-1})_2.
\]
Therefore
\[
B(\log_{\alpha} \beta 2^i) = a_{n-i} \text{ for } 0 \leq i \leq n - 1,
\]
and
\[
\log_{\alpha} \beta = (B(\log_{\alpha} \beta), B(\log_{\alpha} \beta 2^{n-1}), \ldots, B(\log_{\alpha} \beta 2))_2.
\]
This means that \( \log_{\alpha} \beta \) can be computed by \( n \) calls to the oracle. Since the bits can be shifted cyclically, \( \log_{\alpha} \beta \) can also be computed by \( n \) calls to an oracle for any bit. Therefore every bit is a hard predicate. \( \square \)
Corollary 4.2. If the discrete logarithm problem in \( \mathbb{F}_{2^n} \) is hard, then the bit sequence \( z_1, z_2, \ldots \) defined in (1) is pseudorandom.

We have assumed that the oracle is perfect, that is, it always gives correct answers. Blum & Micali (1984) dealt with the more general case of nonperfect oracles (an oracle that may give wrong answers, but it gives correct answers sufficiently more frequently than incorrect ones). It seems likely that Theorem 4.1 still holds for nonperfect oracles.

Our theorem says that every bit of the discrete logarithms is a hard predicate, that is, every bit is individually secure, provided the discrete logarithm is hard to compute. This is different from the discrete logarithms modulo an odd prime \( p \) where its least significant bit is not secure while the most significant bit is indeed secure (Peralta 1986). It seems also possible to modify the proof in Long & Wigderson (1988) to show that any \( O(\log n) \) consecutive bits of the discrete logarithms in \( \mathbb{F}_{2^n} \) are simultaneously secure. In this case, the modified pseudorandom bit generator could output \( O(\log n) \) bits per iteration. The discrete logarithm problem in \( \mathbb{F}_{2^n} \) seems easier than that in \( \mathbb{F}_p \), and one may need to pick a bigger field \( \mathbb{F}_{2^n} \) than for \( \mathbb{F}_p \) to maintain the same level of security.

In implementing our pseudorandom bit generator in \( \mathbb{F}_{2^n} \), one can choose \( \alpha \) to be a Gauss period of type \( (n, k) \) for a small \( k \). The cost of exponentiating \( \alpha \) at each iteration is only \( O(n^2) \) bit operations. This is advantageous compared to the generator in \( \mathbb{F}_p \), where exponentiation needs \( O(n^2 \log n \log \log n) \) bit operations by using fast multiplication algorithms.

Acknowledgement. We thank S.S. Wagstaff for making the updated factorization tables of the Cunningham project (Brillhart et al. 1988) available. We used Maple in our computations. Part of this work was done at the University of Toronto, where the first author was supported by an NSERC Postdoctoral Fellowship and the others by the second author’s NSERC operating grant.

References


Department of Mathematical Sciences, Clemson University, Clemson, SC 29634-1907, USA,

Fachbereich Mathematik-Informatik, Universität-GH Paderborn, D-33095 Paderborn, Germany,

Department of Computer Science, University of Toronto, Toronto, Ontario M5S 1A4, Canada,