Various Implementations of Blum Blum Shub Pseudo-Random Sequence Generator

Kaustubh Gawande
Department of Electrical Engineering
and Computer Science
Oregon State University
Corvallis, Oregon, 97330
Email: gawande@cs.orst.edu

Maithily Mundle
Department of Electrical Engineering
and Computer Science
Oregon State University
Corvallis, Oregon, 97330
Email: mundle@cs.orst.edu

***Abstract*— Two types of BBS Pseudo-random sequence generators are presented and their relative strengths in terms of their predictability is discussed. We survey the following implementations of x^2 mod N generator, presented by Blum, Blum and Shub:**

1) **Efficient and Secure Implementation of BBS Pseudo-Random Number Generator**
2) **Fast Blum-Blum-Shub Sequence Generation Using Montgomery Multiplication**
3) **A software implementation of BBS x^2 mod N generator in Java.**

I. Introduction

Ideally, a Pseudo Random Sequence generator should quickly produce long sequences of bits from short seeds. These sequences should appear to be generated by successive flips of coin. An important property of coin tosses is unpredictability. Pseudo-Random Sequence Generators should be unpredictable to computers with feasible resources. Pseudo-Random Sequence generator is said to be polynomial-time unpredictable (Unpredictable to the right and to the left) [1] if and only if for every finite initial segment of sequence that has been produced by such a generator, but with any element (The rightmost or leftmost element) deleted from the segment, a probabilistic Turing Machine can do no better in guessing in polynomial (Polynomial in the length of the seed) time what the missing element is than by flipping a fair coin.

A lot of research has been done in this field and people have come up with different Pseudo-Random Sequence Generators. Blum, Blum and Shub have presented two Pseudo-Random Sequence Generators [2]. The 1/P generator is completely predictable, whereas the x^2 mod N generator, under a certain intractability assumption, is unpredictable in a precise sense [2]. But, x^2 mod N generator is not very efficient. This paper surveys some of the efficient and secure implementations of x^2 mod N generator.

II. A Simple Unpredictable Pseudo-Random Number Generator

Blum, Blum and Shub have presented two pseudo-random number sequence generators and discussed their properties [2]. They are,

1) the 1/P generator
2) the x^2 mod N generator

There are significant similarities in the two generators. For e.g.: Each quickly generates long well-distributed sequences from short seeds. Also both generators contain hard problems at their core - the 1/P generator is based on the discrete logarithm problem and the x^2 mod N generator is based on the quadratic residuacity problem. What is most interesting is in spite of their similarities; only the second is unpredictable - assuming a certain intractability hypothesis.

The 1/P generator has been well studied in the history of number theory [3] and also as a random number generator [Knuth]. However [2] have come up with new and surprising results concerning its interference properties. The x^2 mod N generator, on the other hand, is an outgrowth of the coin-flipping protocol of [4]. It derives its strong security properties from complexity based number theoretic assumptions and arguments [4], [5], [6]. Blum, Blum and Shub have investigated and revealed additional useful properties of this generator like: From knowledge of the secret factorization of N, one can generate the sequence backwards; from additional information of N, one can even random access the sequence.

Both generators have applications. The 1/P generator has applications to the generations of generalized de Bruijn (i.e. maximum-length shift register) sequences. The $x^2 mod N$ generator has applications to public key cryptography (Quadratic Residue Cipher) as explained in Section [Fast BBS Sequence Generation using MM].

A. *The 1/P Generator:*

Fix an integer b > 1 and let $\sum = \{0, 1, ..., b-1\}$.

DEFINITION (1/P generator (base b)): To define the seed space, let **N** = {integers P>1 relatively prime to b} be the

parameter values, and let the seed domain X be the disjoint union $\bigcup_{P\in N} Z_P^*$. We can, and sometimes do, identify X with the dense subset {r / P|P $\in$ **N**, r $\in Z_P^*$} of the unit interval [0,1]. Let μ_n be the distribution on X^n given by $\mu_n(P,r) = u_n(P) * v_p(r)$, where u_n is the uniform probability distribution on $\{P \in N \parallel P \mid_b = n\}$ and v_p is the uniform distribution on Z_P^*. Then U = $\{\mu_n\}$ is an accessible probability distribution on X.

Recall that Z_N^* = { integers x | 0 < x < N and gcd(x,N) = 1} is a multiplicative group of order Ψ(N). If P is prime, then Z_P^* = {1, 2, , P-1} is cyclic. For each N, we consider $Z_N^* \subset \{0,1\}^n$ via the natural distribution.

Example: Let the base b=10, and let P=7 and r=1. The pseudo-random sequence generated by the 1/P generator (base 10) with input 1/7 is 142857142.Note that 10 is a primitive root mod 7 (i.e. a generator of the cyclic group Z_7^*) and that the period of this sequence is 7-1 = 6. (Please see [BBS - Theorem 1 for details])[2]. From the state space point of view, the orbit of 1/7 under T is: 1/7, 3/7, 2/7, 6/7, 4/7, 5/7, 1/7, and so, b_0=1 (since 1/7 $\in$ [1/10, 2/10]), b_1=4 (since 3/7 $\in$[4/10, 5/10]), $b_2 = 2, b_3 = 8, b_4 = 5$,

B. *The generator x^2 mod N:*

DEFINITION [x^2 mod N generator]: Let **N**={integers N|N = P*Q, such that P,Q are equal length (|P|=|Q|) distinct primes 3 mod 4} be the set of parameter values. For N $\in$ **N**, let $X_N = \{x^2 mod N \mid x \in Z_N^*\}$ be the quadratic residues mod N. Let X = disjoint $\bigcup_{N\in\mathbf{N}} X_N$ be the seed domain.

Example: Let N=7*19 = 133 and x_0=4.Then the sequence $x_0, x_1 = x_0^2$ mod 133, has period 6, where x_0, x_1 , , x_5, . = 4, 16, 123, 100, 25, 93, . So b_0 b_1 b_5 = 0 0 1 0 1 1 ..... The latter string of b's is the pseudo-random sequence generated by the x^2 mod N generator with input (133, 4). Here, λ(N) = 18 and $\lambda(\lambda(N))$ = 6.

C. *Predictability of BBS Generators:*

1) The 1/P generator is predictable: Let P and b be relatively prime integers > 1 and r_0 an integer in the range 0 < r_0 < P. Denote the expansion on r_0/P to base b by

$$r_0/P = .q_1q_2q_3$$

where 0 $\leq q_i \leq$ b. Since b is prime to P, the expansion is periodic. Then m = 0,

$$(b^m * r_0)/P = q_1q_m.q_{m+1}q_{m+2}... = (q_1....q_m) + r_m/P$$

where,

$$0 < r_m/P = .q_{m+1}q_{m+2}..... = (b^m * r_0/P) mod 1 < 1.$$

Here q_1, q_2,... are (quotient) digits base b and $q_1q_2q_3$... denotes their concatenation, whereas r_m, the mth remainder (of r_0 / P base b), is an integer whose length (base b) is less or equal to the length of P: $\mid r_m \mid_b \leq \mid P \mid$, where $\mid P \mid$ denotes $\mid P \mid_b$.

Then, the following problems are solvable in polynomial ($\mid P \mid$)-time [2]. (For proofs refer to [2]).

Problem 1:
Input: b, remainder r_m, positive integer k $\leq$ poly ($\mid P \mid$).
Output: $r_{m-1}, r_{m+k}; q_m q_{m+1} ... q_{m+k}$.

Problem 2:
Input: b, $\mid P \mid$ successive digits of quotient $q_{m+1}....q_{m+\mid P\mid}$
Output: r_m

Problem 3:
Input: b, $r_m, r_{m+1} such that r_m * b \neq r_{m+1}$
Output: P

Problem 4:
Input: b, k quotient digits, $q_m q_{m+1} ....q_{m+k}$, where k = $log_b(2P^2)$ and m is arbitrary
Output: P, r_m

Hence, the 1/P generator yields a poor pseudo-random sequence: from knowledge of P and any $\mid P \mid$-long segment of the expansion of r_0 / P base b, one can efficiently extend the segment backwards and forwards. Moreover, from the knowledge of 2$\mid P \mid$ + 1 successive elements of the sequence, but not P, one can efficiently reconstruct P, and hence efficiently continue the sequence in either direction. So, the 1/P generator is predictable.

2) The x^2 mod n generator is unpredictable: Blum, Blum and Shub proved that the x^2 mod n generator is polynomial-time unpredictable [2]. (For proofs refer to [2]).They investigated what properties can be inferred about sequences produced by the x^2 mod n generator, given varying amounts of information.

Let N = P * Q, where P, Q are distinct primes both congruent to 3 mod 4. Also x_i is a quadratic residue mod N, $x_{i+1} = x_i^2$ mod N and b_i = parity (x_i).

1. Knowledge of N is sufficient to efficiently generate sequences $x_0, x_1, x_2, ...$ and hence the sequences $b_0b_1b_2$... in the forward direction, starting from any given seed x_0. The number of steps per output is $\bigcirc(\mid N \mid^{1+\epsilon})$ using fast multiplication.

2. Given N, the factors of N are necessary and sufficient to efficiently generate the x^2 mod n sequences in the reverse direction, x_0, x_1, x_2,..., starting from the given seed x_0.

3. The factors of N are necessary- assuming they are necessary for deciding quadratic residuosity of an x in Z_N^*(+1) - to have even ϵ-advantage in guessing in polynomial time the parity of x_{-1}, given N and x_0 chosen at random from OR_N, it is sufficient to choose x at random from Z_N^* and square it mod N.

Blum, Blum and Shub proved that x^2 mod n generator is unpredictable [2].

D. *Advantages of BBS x^2 mod n generator:*

An interesting characteristic of this generator is that you can directly calculate any of the x values. In particular,

$$x_i = (x_0^{(2^i (mod((P-1)*(Q-1))))})(modN)$$

This means that in applications where many keys are generated in this fashion, it is not necessary to save them all. Each key can be effectively indexed and recovered from that small index and the initial x and N.

E. ***Disadvantages of BBS x^2 mod n generator:***

Its only disadvantage is that it is computationally intensive. It takes n^2 steps to generate one random bit of the bit-stream. This is not a serious draw back if it is used for moderately infrequent purposes, such as generating session keys.

III. Efficient and Secure Implementation of BBS Pseudo-Random Number Generator

There has been a lot of interest in provably good pseudo-random generators [2], [1], [6], [7]. In spite of the fact that all the generators proposed by these people pass all probabilistic polynomial time statistical tests, they are still inefficient. Vazirani and Vazirani proposed an efficient and secure pseudo-random number generator [8].

Vazirani and Vazirani used a variant of BBS generator to prove their theorems [8]. This variant of BBS generator outputs b_i = location (x_i), where location(x) = 0 if x $<$ (N-1)/2 and 1 if x $\geq$ (N-1)/2. The cryptographic security of this generator was also based on quadratic residuosity. However the generator which extracts parity as well as location at each stage may not be cryptographically secure, because revealing parity (x_i) may make location (x_i) predictable. Blum, Blum and Shub conjectured that this generator is also cryptographically secure and asked the open problem: how many bits can be extracted at each stage, maintaining cryptographic security? [2].

Vazirani and Vazirani proved the conjuncture and also answered the open problem by giving a simple condition, the XOR-Condition [8]. They proved that log n bits, where n = $\mid N \mid$, can be extracted at each stage from any generator satisfying this condition. (For proofs refer to [8]).

IV. Fast Blum-Blum-Shub Sequence Generation Using Montgomery Multiplication

Parker, Kemp and Shepherd have proposed VLSI modules for fast, efficient generation of high-throughput Blum-Blum-Shub (BBS) and BBS-like sequences using Montgomery Multiplication, where post-processing associated with Montgomery's algorithm can be eliminated [9].

A. ***Quadratic Residue Cipher and BBS:***

Public key cryptosystems ensure secrecy between communicating parties without the need to distribute secret keys. Quadratic Residue Cipher (QRC) is a lesser-known public key cryptosystem introduced by Blum, Blum, and Shub [2], which relies on the ease of squaring an integer, mod n, as compared to the intractability of finding the square root of a number, mod n when n is large.

B. ***QRC v/s RSA:***

Just like RSA [10], this scheme relies on the inability to factor n when n = p*q, and p and q are large strong primes. The advantages of QRC over RSA are that RSA is a deterministic cipher, whereas QRC is probabilistic because it starts from a randomly chosen seed. Moreover it is known that RSA can leak partial information about the message sent, whereas no such weakness is known for QRC. Also it is a bit easier to generate BBS for QRC, than successive exponents for RSA. QRC can also provide digital signature and resistance to a chosen cipher text attack, but at greater cost than with RSA [11]. Detailed comparisons of QRC and RSA are given in [11], [12], [13]. Although implementation complexity of QRC is slightly less than RSA, it is still costly. Successful implementation of QRC relies on efficient generation of BBS.

C. ***BBS using MM (Montgomery Multiplication):***

Parker, Kemp and Shepherd propose novel hardware to allow highly efficient generation of BBS using Montgomery Multipliers (MMs) [14], [15], [16], [17]. MMs are particularly suited to VLSI implementation of modular multiplication as they allow computation of modular reduction to begin before computation of the most-significant-bit has been completed. This speeds up successive modular arithmetic operations (such as squaring). However the drawback is the multiplicative offset associated with MM. In their paper, this offset is incorporated into BBS generation without cost, and further simplification is made possible by considering the generation of BBS-like sequences.

The authors conclude by saying that Montgomery Multiplication (MM) is appropriate for fast generation of BBS sequences in spite of the constant multiplicative offset inherent within successive squaring using MM. Post-processing associated with MM is taken out of the squaring loop to occur in parallel with the squaring. Further area/time savings are achieved by retaining the multiplicative offset to the BBS Sequence whilst ensuring each member of the sequence is less than the modulus. High-throughput sequence generation is obtained using interleaved squaring and fully parallel MM. The authors also suggest that one could eliminate post-processing completely by some other simple enhancements [9].

V. Software implementation of BBS x^2 mod n generator

We studied the software implementation of the BBS x^2 mod n generator by David Bishop [18]. It is implemented using J2SE V1.4.1. The java.math package [19] which provides classes for performing arbitrary-precision integer arithmetic (BigInteger) and arbitrary-precision decimal arithmetic (BigDecimal) is used. A java applet acts as the interface to this pseudo-random number generator.

VI. Conclusion

Generating a secure, fast and efficient Pseudo- random number generator (PRNG) has been a challenge to Cryptologists, Computer Scientists, Computer Engineers and Mathematicians for several decades. The Pseudo-Random Number Generator proposed by Blum, Blum and Shub is a cryptographically strong random number generator that has many interesting applications [BBS], especially in the field of Public Key Cryptography. By proving that the random number generator proposed by Blum, Blum and Shub is secure and efficient, Varizani and Vazirani opened the doors to implementing such a generator - a PRNG that can be successfully used in real life cryptography solutions [Vazirani]. Montgomery Multiplication has several interesting applications in Cryptography and building a Quadratic Residue Cipher, based on BBS that uses Montgomery Multiplication (MM), Parker, Kemp and Shepherd have proved that when put together BBS and MM can help us build a PRNG that is not only secure and efficient but also fast [9]. Our implementation of BBS in Java further establishes that it is very simple to implement a secure, fast and efficient PRNG.

References

[1] M. Blum and S. Micali, "How to generate cryptographically strong sequences of pseudo-random bits," 1982, pp. 112–117.

[2] L. Blum, M. Blum, and M.Shub, "A simple unpredictable pseudo-random number generator," *SIAM Journal on Computing*, vol. 15, no. 2, pp. 364–383, May 1986.

[3] L. Dickson, "History of the theory of numbers," *Chelsea Pub. Co.*, 1919.

[4] M. Blum, "Coin flipping by telephone," 1982, pp. 133–137.

[5] S. Goldwasser and S. Micali, "Probabilistic encryption and how to play mental poker keeping secret all partial information," 1982, pp. 365–377.

[6] A. Yao, "Theory and applications of trapdoor pseudo-random number generators," 1982, pp. 80–91.

[7] C. Schnorr and W. Alexi, "Rsa-bits are 0.5 + epsilon secure," *EUROCRYPT*, 1984.

[8] U. V. Vazirani and V. V. Vazirani, "Efficient and secure pseudo-random number generation," 1984, pp. 458–463.

[9] M.G.Parker, A.H.Kemp, and S.J.Shepherd, "Fast blum-blum-shub sequence generation using montgomery multiplication," vol. 147, 2000, pp. 252–254.

[10] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978.

[11] F. Rubin, "The quadratic residue and double quadratic residue ciphers," *Cryptologia*, no. 3, pp. 275–284, 1995.

[12] S. J. Shepherd, P. W. Sanders, and C. T. Stockel, "The quadratic residue cipher and some notes on implementation," *Cryptologia*, no. 3, pp. 264–282, 1993.

[13] S. J. Shepherd, A. H. Kemp, and S. K. Barton, "An efficient key exchange protocol for cryptographically secure cdma systems," *Globecom'96*, Nov. 1996.

[14] P. L. Montgomery, "Modular multiplication without trial divison," *Mathematical Computation*, vol. 44, pp. 519–521, 1985.

[15] C. D. Walter, "Systolic modular multiplication," vol. 42, 1993, pp. 376–378.

[16] P. Kornerup, "A systolic, linear array multiplier for a class of right-shift algorithms," vol. 43, 1994, pp. 892–898.

[17] T. Blum and C. Paar, "Montgomery modular exponentiation on reconfigurable hardware," 1999.

[18] D. Bishop, "Random bit-stream generator/blum-blum-shub method." [Online]. Available: http://computerscience.jbpub.com/cryptography/

[19] Sun Microsystems, "Java^{TM} 2 Platform, Standard Edition, v 1.4.1 API Specification." [Online]. Available: http://java.sun.com/j2se/1.4.1/docs/api/