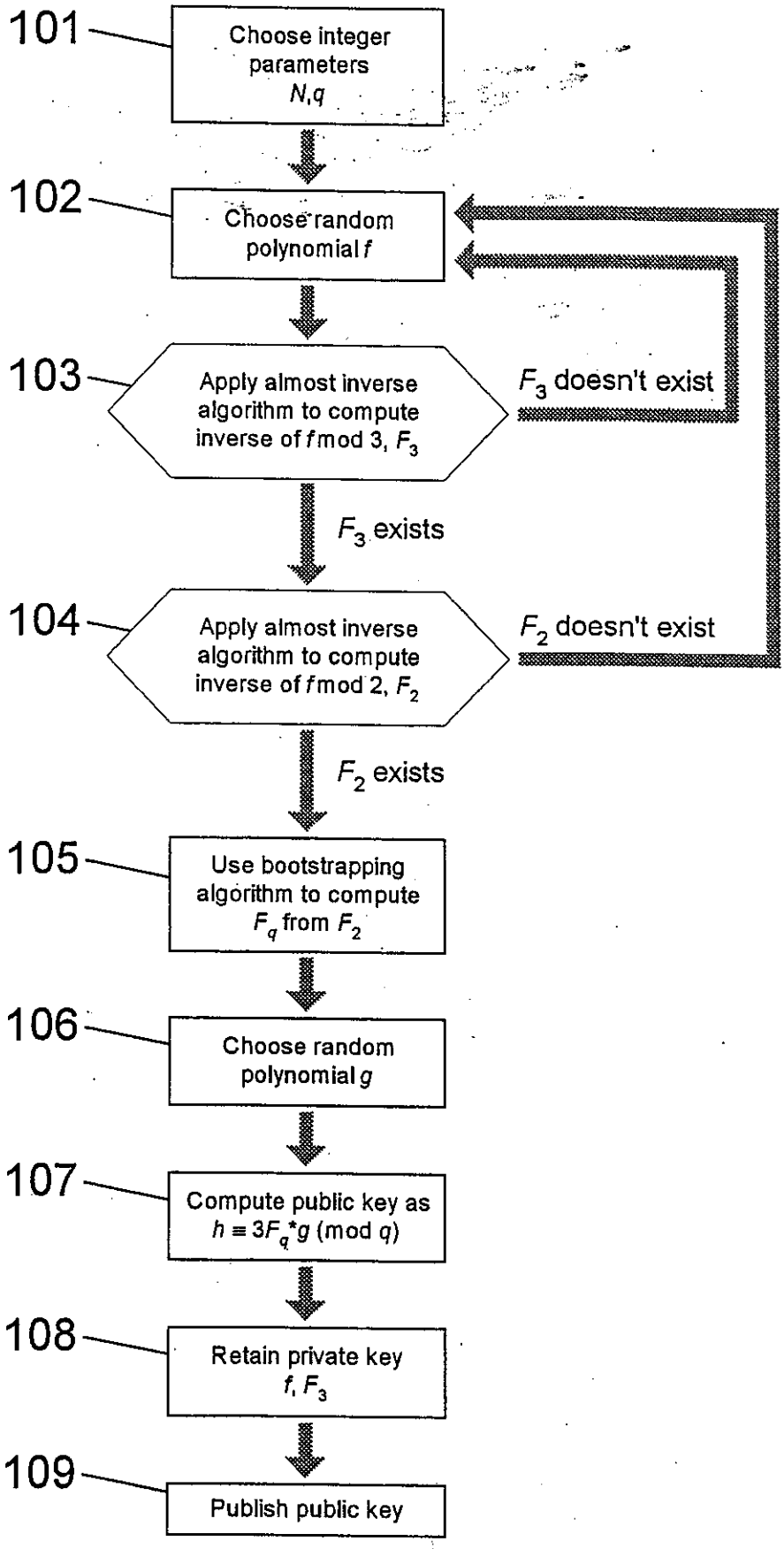

101
Choose integer parameters N,q
102
Choose random polynomial f
103
Apply almost inverse algorithm to compute inverse of f mod 3, F3
F3 doesn't exist
F3 exists
104
Apply almost inverse algorithm to compute inverse of f mod 2, F2
F2 doesn't exist
F2 exists
105
Use bootstrapping algorithm to compute Fq from F2
106
Choose random polynomial g
107
Compute public key as h ≡ 3Fq*g (mod q)
108
Retain private key f, F3
109
Publish public key

Figure 1

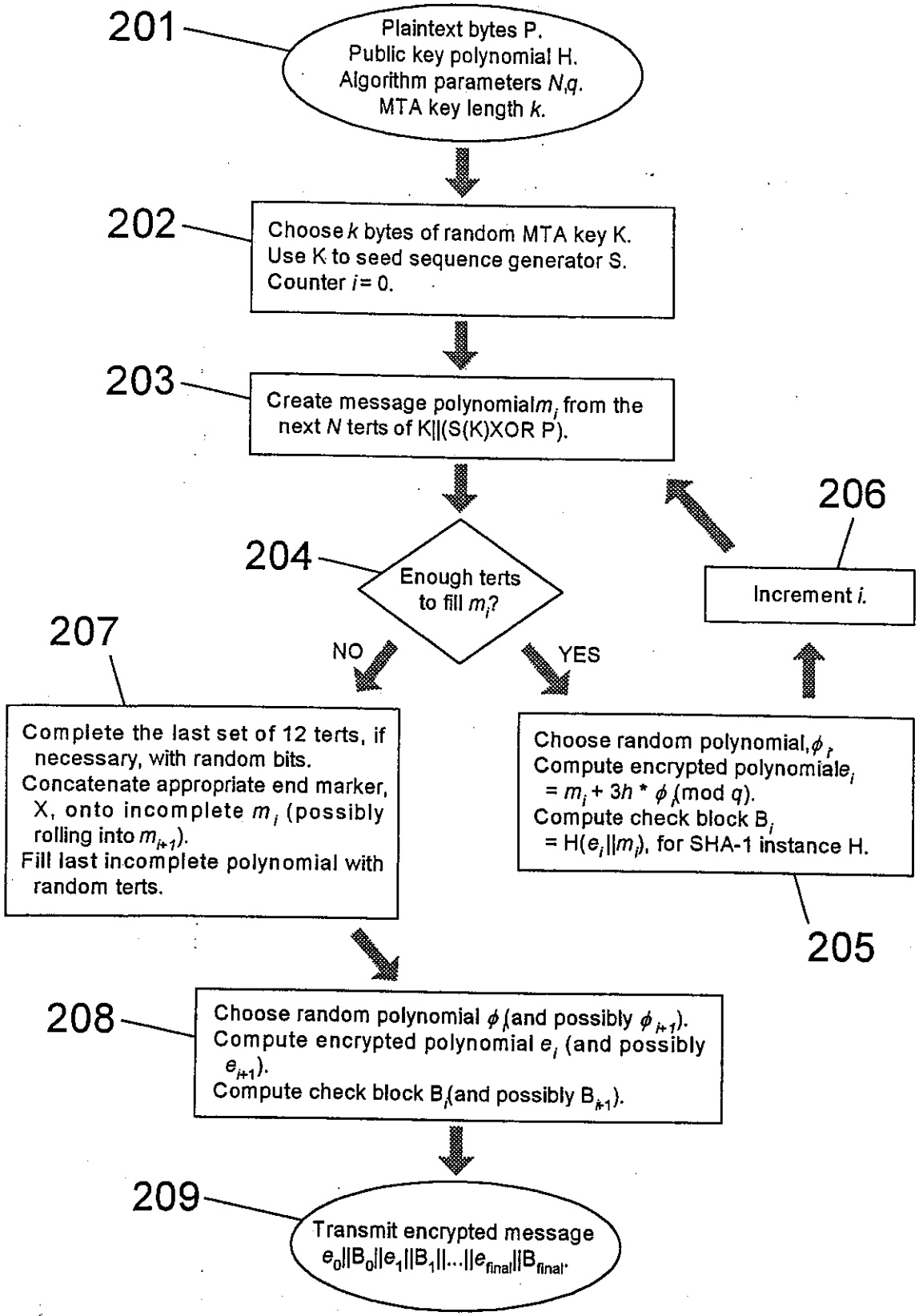

201
Plaintext bytes P.
Public key polynomial H.
Algorithm parameters N,q.
MTA key length k.
202
Choose k bytes of random MTA key K.
Use K to seed sequence generator S.
Counter i = 0.
203
Create message polynomial m_i from the next N terts of K||(S(K)XOR P).
204
Enough terts to fill m_i?
NO
YES
206
Increment i.
207
Complete the last set of 12 terts, if necessary, with random bits.
Concatenate appropriate end marker, X, onto incomplete m_i (possibly rolling into m_{i+1}).
Fill last incomplete polynomial with random terts.
Choose random polynomial, ϕ_i
Compute encrypted polynomial $e_i = m_i + 3h * \phi_i \pmod q$.
Compute check block $B_i = H(e_i||m_i)$, for SHA-1 instance H.
205
208
Choose random polynomial ϕ_i (and possibly ϕ_{i+1}).
Compute encrypted polynomial e_i (and possibly e_{i+1}).
Compute check block B_i (and possibly B_{i+1}).
209
Transmit encrypted message $e_0||B_0||e_1||B_1||...||e_{final}||B_{final}$.

Figure 2

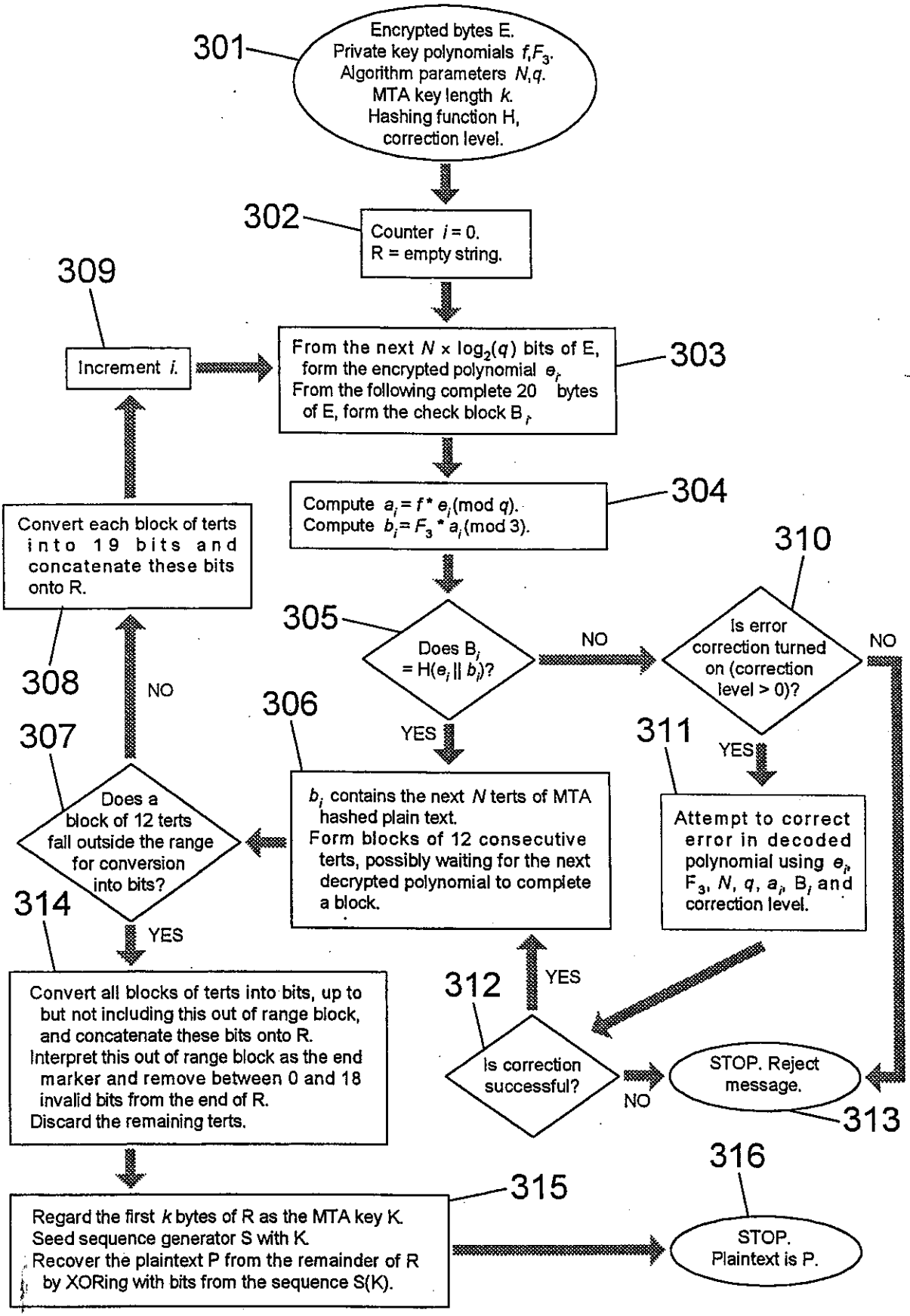

301
Encrypted bytes E.
Private key polynomials f, F_3.
Algorithm parameters N, q.
MTA key length k.
Hashing function H,
correction level.
302
Counter $i = 0$.
R = empty string.
303
From the next $N \times \log_2(q)$ bits of E, form the encrypted polynomial e_i.
From the following complete 20 bytes of E, form the check block B_i.
304
Compute $a_i = f * e_i \pmod q$.
Compute $b_i = F_3 * a_i \pmod 3$.
305
Does $B_i = H(e_i \| b_i)$?
NO
YES
306
b_i contains the next N terts of MTA hashed plain text.
Form blocks of 12 consecutive terts, possibly waiting for the next decrypted polynomial to complete a block.
307
Does a block of 12 terts fall outside the range for conversion into bits?
NO
YES
308
Convert each block of terts into 19 bits and concatenate these bits onto R.
309
Increment i.
310
Is error correction turned on (correction level > 0)?
NO
YES
311
Attempt to correct error in decoded polynomial using e_i, F_3, N, q, a_i, B_i and correction level.
312
Is correction successful?
YES
NO
313
STOP. Reject message.
314
Convert all blocks of terts into bits, up to but not including this out of range block, and concatenate these bits onto R.
Interpret this out of range block as the end marker and remove between 0 and 18 invalid bits from the end of R.
Discard the remaining terts.
315
Regard the first k bytes of R as the MTA key K.
Seed sequence generator S with K.
Recover the plaintext P from the remainder of R by XORing with bits from the sequence S(K).
316
STOP.
Plaintext is P.

Figure 3

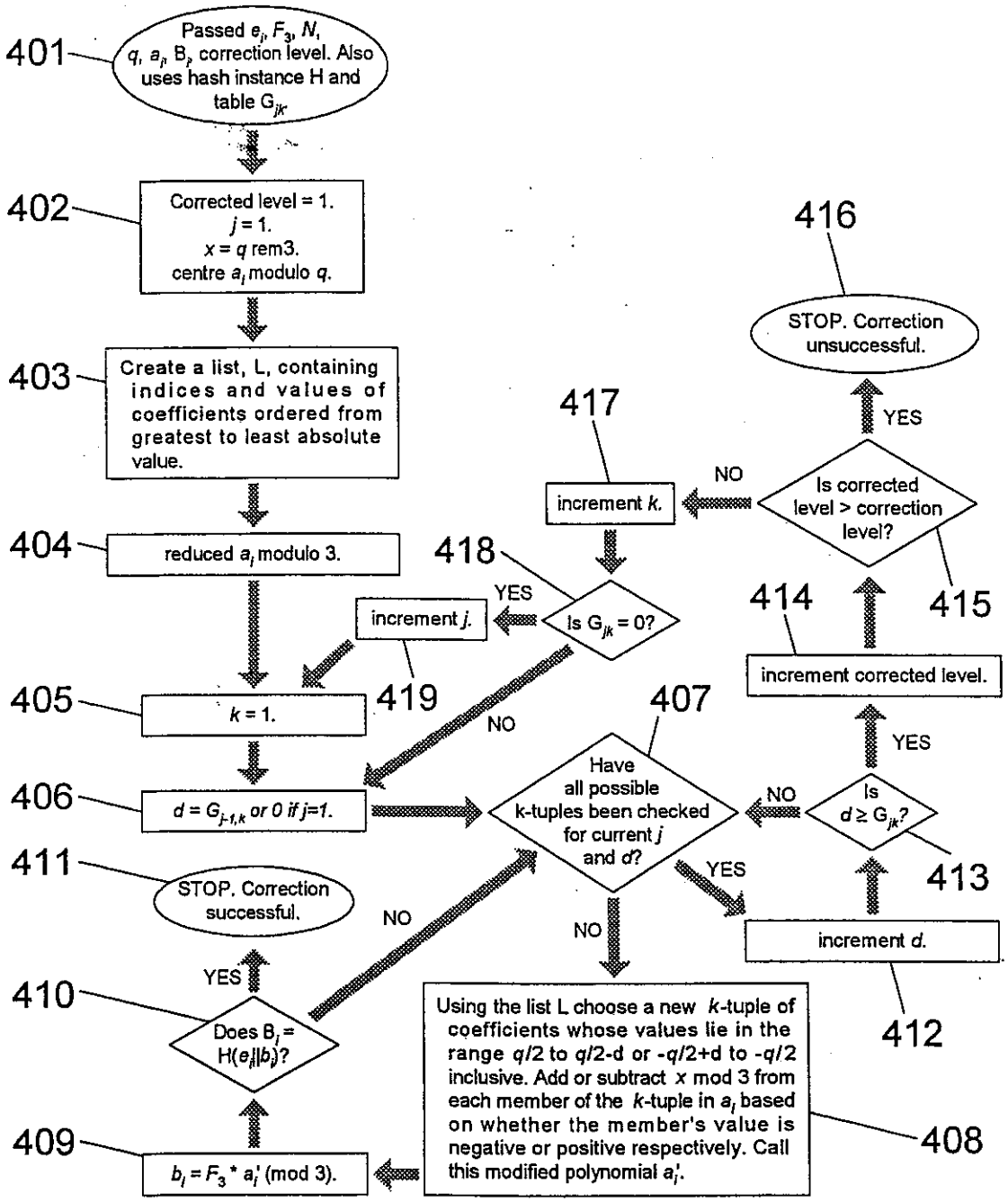

401
Passed e_i, F_3, N, q, a_i, B_i, correction level. Also uses hash instance H and table G_{jk}
402
Corrected level = 1.
j = 1.
x = q rem3.
centre a_i modulo q.
403
Create a list, L, containing indices and values of coefficients ordered from greatest to least absolute value.
404
reduced a_i modulo 3.
405
k = 1.
406
$d = G_{j-1,k}$ or 0 if j=1.
407
Have all possible k-tuples been checked for current j and d?
408
Using the list L choose a new k-tuple of coefficients whose values lie in the range $q/2$ to $q/2$-d or -$q/2$+d to -$q/2$ inclusive. Add or subtract x mod 3 from each member of the k-tuple in a_i based on whether the member's value is negative or positive respectively. Call this modified polynomial a_i'.
409
$b_i = F_3 * a_i'$ (mod 3).
410
Does B_i = $H(e_i||b_i)$?
411
STOP. Correction successful.
412
increment d.
413
Is $d \geq G_{jk}$?
414
increment corrected level.
415
Is corrected level > correction level?
416
STOP. Correction unsuccessful.
417
increment k.
418
Is G_{jk} = 0?
419
increment j.
YES
NO

Figure 4

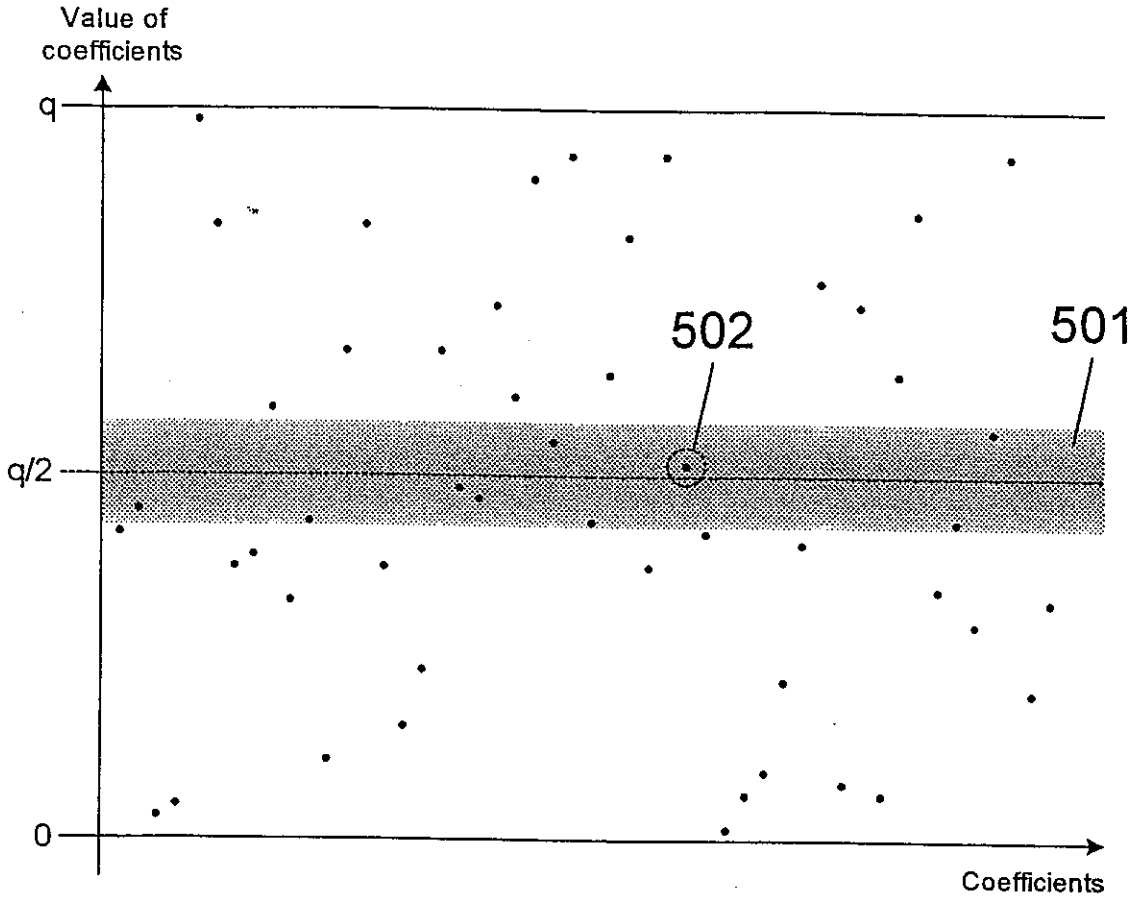


Figure 5

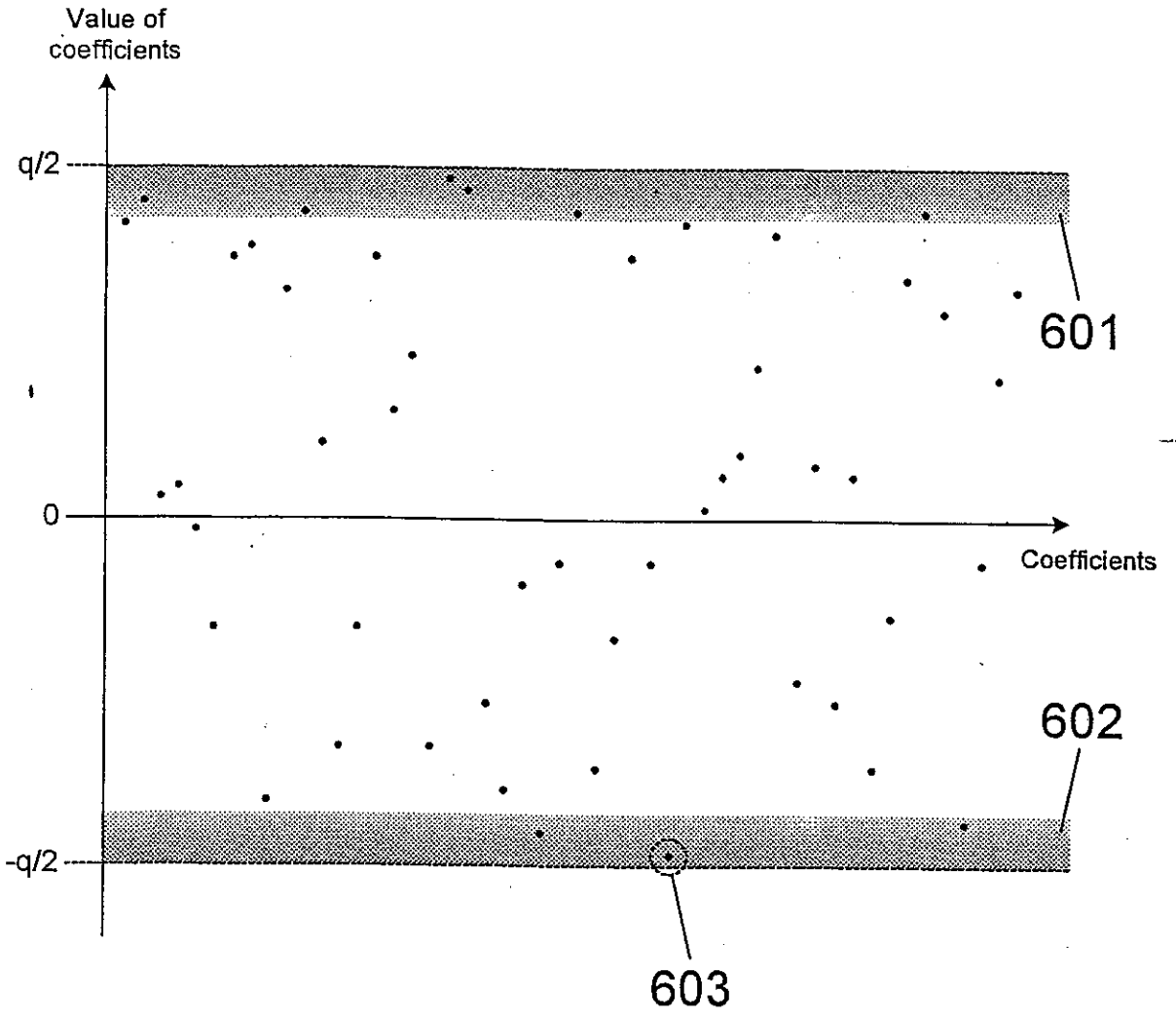


Figure 6

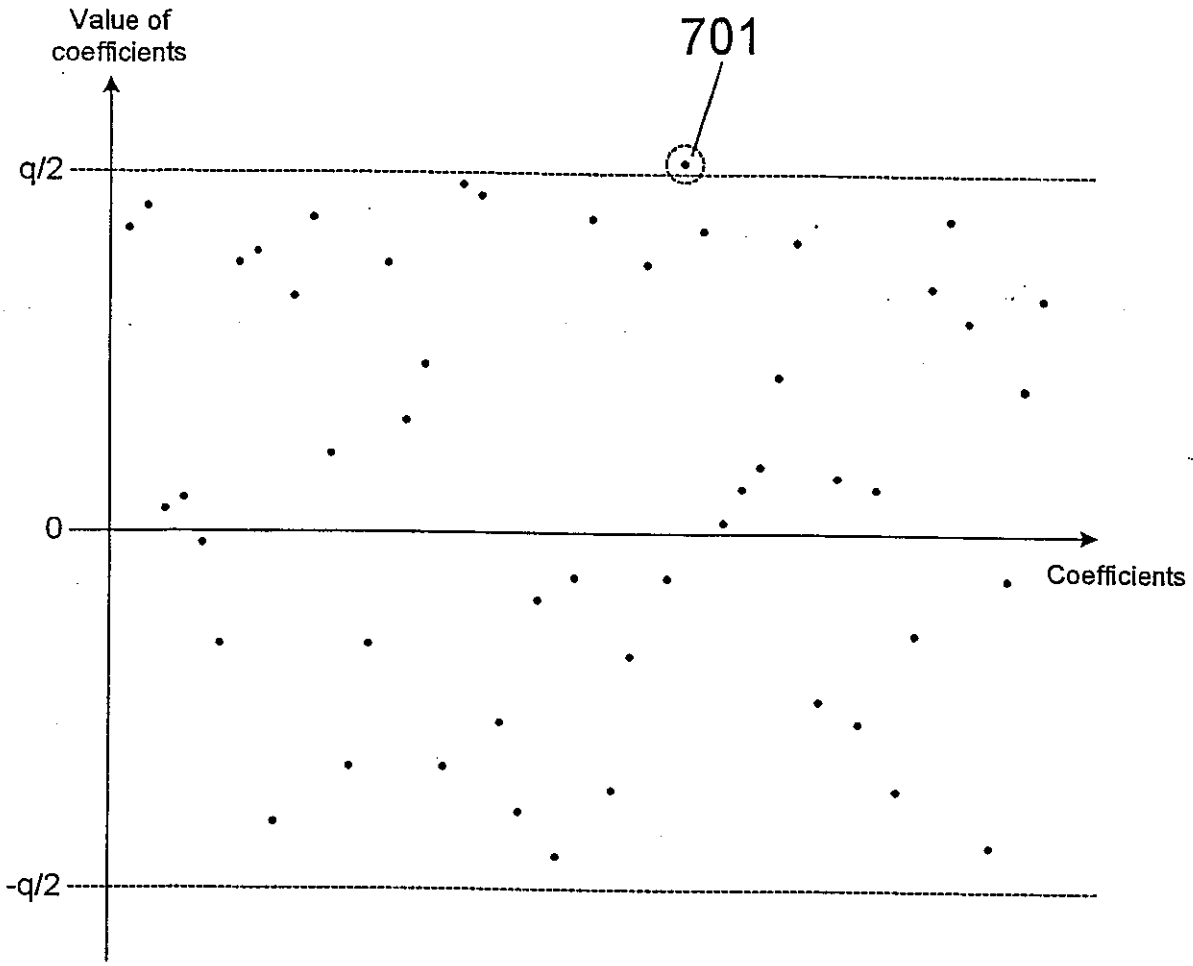


Figure 7

22 NOV 2004

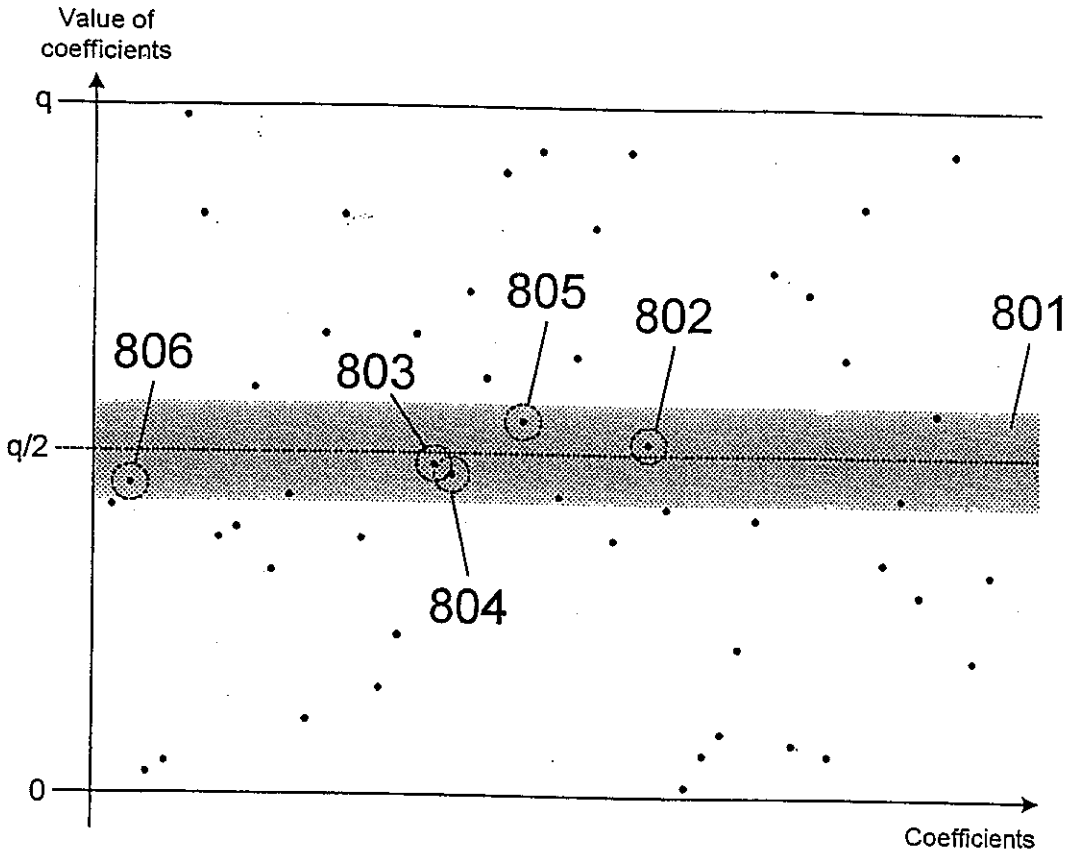


Figure 8

22 NOV 2004

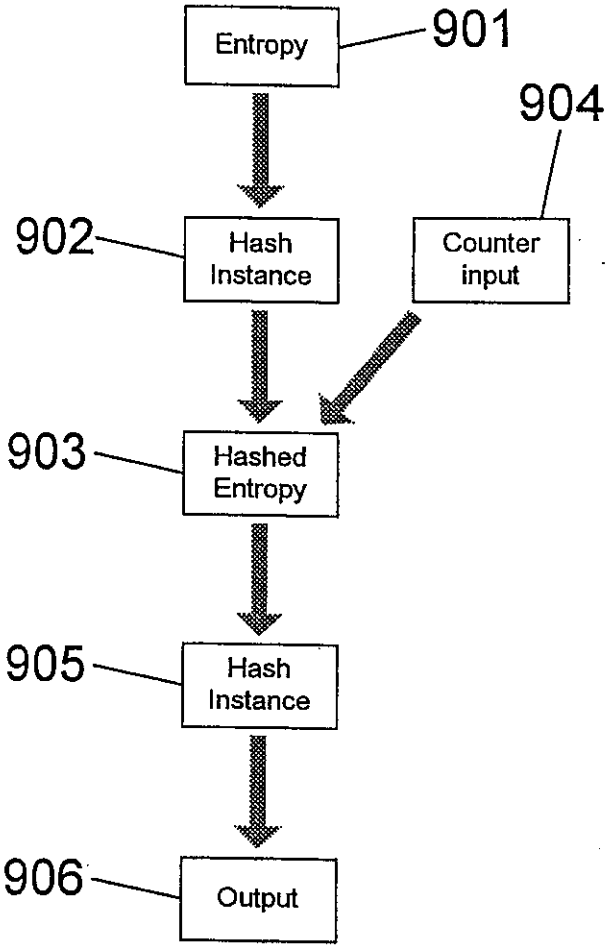


Figure 9

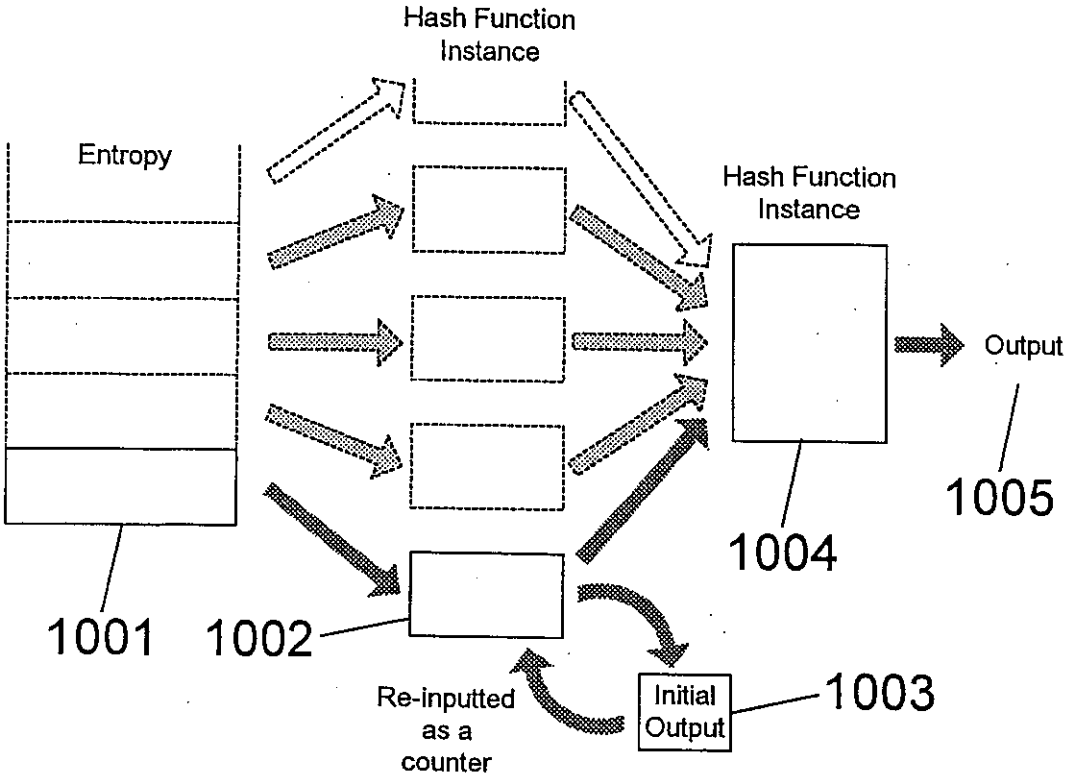

Hash Function Instance
Entropy
Hash Function Instance
Output
1001
1002
1004
1005
Re-inputted as a counter
Initial Output
1003

Figure 10

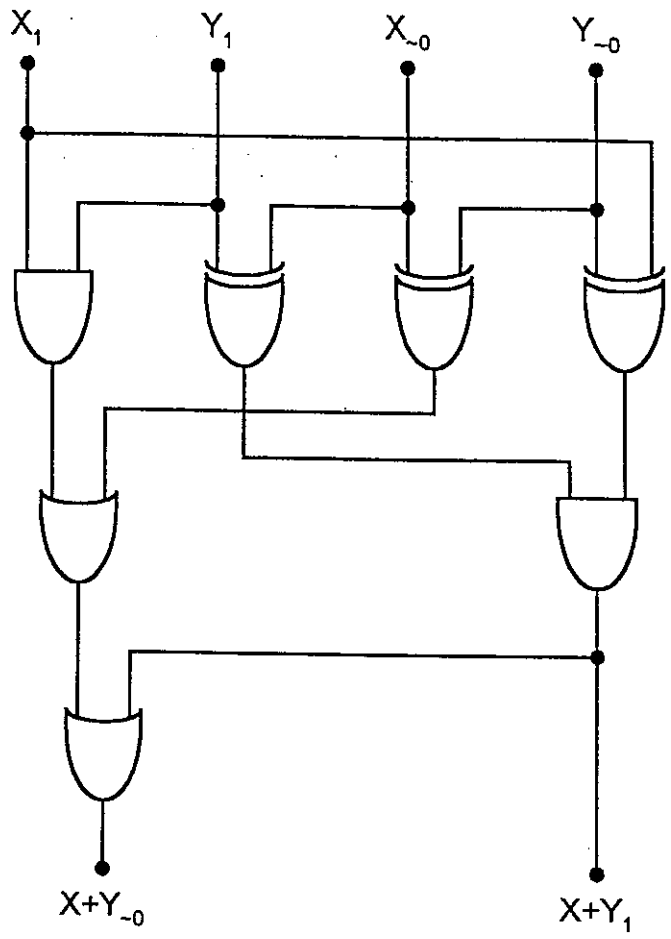


Figure 11

22 NOV 2004

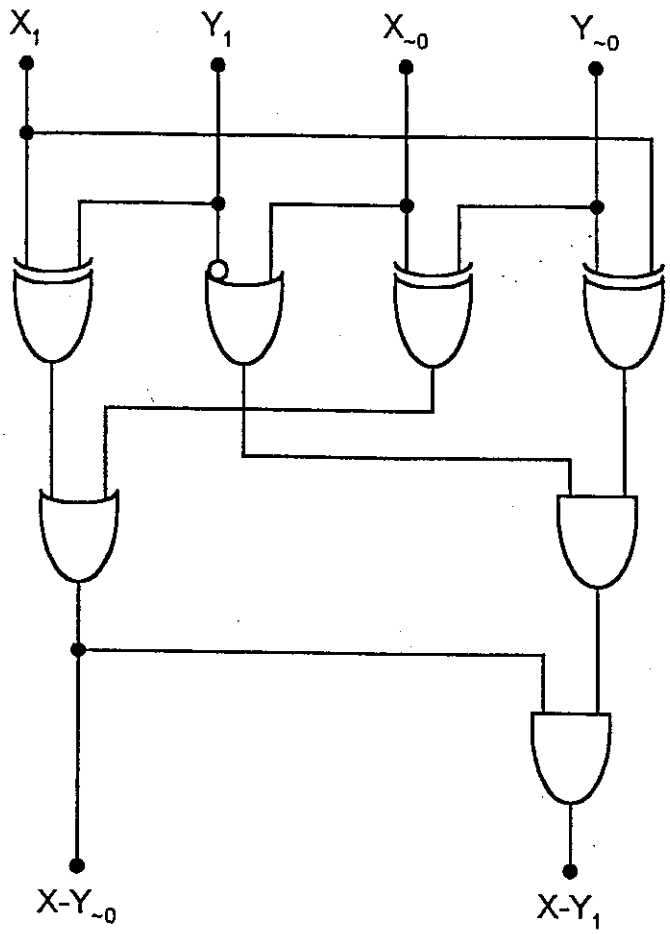


Figure 12

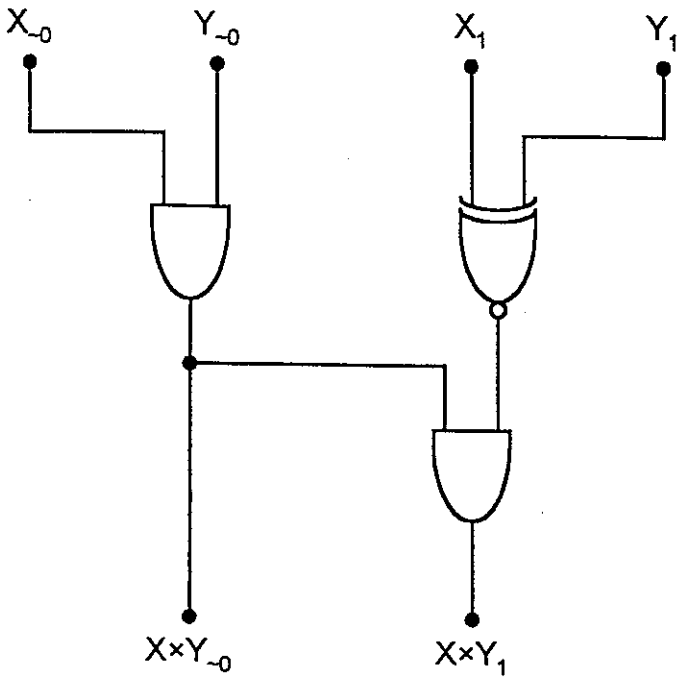


Figure 13