Рахман П.А., МЭИ (ТУ), 2007.

Основы защиты данных от разрушения. Коды Рида-Соломона.

Одна из наиболее острых проблем в информационных технологиях – это защита данных от разрушения. Как говорится, «энтропия» слепа, но терпелива. «Энтропия» не имеет собственной воли и не разрушает целенаправленно конкретный порядок в конкретной структуре, но у нее есть единственная цель – глобальный хаос, и она случайным образом, вслепую, наносит свои удары по любым упорядоченным структурам, делая из них бесформенную бессмыслицу. «Энтропия» вечна, фундаментальна и непобедима. Сколько не восстанавливай порядок, она снова терпеливо стремится везде достичь беспорядка. Чтобы противостоять «энтропии» необходимо непрерывно затрачивать энергию, проявлять изобретательность и применять специальные технологии.

В течение большей части второй половины прошлого века математики и специалисты по аппаратным и программным средствам ЭВМ и проблемам передачи данных упорно бились над тем, чтобы выработать технологию, позволяющую кодировать информацию таким образом, чтобы при разрушении случайно выбранных ее блоков, эти блоки можно было восстановить. После того, как была заложена математическая основа для кодирования, разработаны эффективные алгоритмы кодирования и декодирования, технология ближе к концу прошлого века стала более или менее устоявшейся, и, честно говоря, вообще превратилась в ширпотреб, которым пользуются все повседневно и даже не подозревают о ее существовании. Технология применяется при приеме / передаче данных в сетях и при чтении / записи данных на большинстве носителей информации.

И здесь, если пользователям и домохозяйкам можно простить их невежественно-потребительское отношение к высоким технологиям, то уж уважающие себя специалисты по системным и сетевым технологиям обязаны быть знакомы с тонкостями и деталями технологии. Ну, а уж уважающие себя программисты, как в подтверждение своей высокой квалификации, должны иметь свои собственные корректно работающие программные реализации тех непростых алгоритмов, которые используются в этой технологии.

1. Кратко о конечных полях. Поля Галуа.

Основная трудность в понимании технологии кодирования заключается в том, что она базируется не на привычной еще со школьной скамьи алгебре в бесконечном поле вещественных чисел, а на специальной алгебре конечных полей [1], конкретно на полях Галуа с количеством элементов, равным 2^M. Поле Галуа обозначается $GF(2^M)$. Аббревиатура GF – это сокращение от Galois Field. Как известно, в привычной для нас алгебре операции сложения / вычитания и тем более уж умножения / деления могут дать результат, который выходит за пределы некоторого заданного диапазона. Например, если мы имеем диапазон чисел от 1 до 100, и работаем только с числами из этого диапазона, то, например, результаты операций: 45 + 90, 10 – 25, 50 * 30, 15 / 60 уже выйдут за этот диапазон. А вот в конечном поле, любые операции с любыми элементами этого поля в результате дают элемент, принадлежащий этому же полю. Соответственно, в конечных полях вообще отсутствует такие проблемы, как переполнение при умножении и потеря точности при делении, делая тем самым алгебру конечных полей наиболее естественной с точки зрения ЭВМ, которая имеет конечную разрядную сетку для представления данных и конечную емкость памяти.

В принципе алгебра конечных полей не сложнее, а местами гораздо даже проще, чем классическая алгебра. В ней можно выполнять любые арифметические операции, составлять и решать уравнения, и даже дифференцировать и интегрировать. Простыми словами, конечное поле – это некоторая конечная совокупность чисел, над которыми можно производить четыре арифметические операции, не выходя за пределы этой совокупности. В частности, поле Галуа $GF(2^8)$, используемое в технологии кодирования, состоит из $2^8 = 256$ целых чисел от 0 до $2^8 – 1$, расположенные не в порядке возрастания, а особым образом.

Согласно теории, i-й элемент поля Галуа – это результат возведения в i-ю степень некоторого примитивного элемента, в качестве которого обычно берется простое число 2, где i = 0…2^8 – 1. Но мы сразу видим, что, уже начиная i = 8, мы получим результат, который уже выходит за пределы [0, 2^8 – 1] и здесь используется особый подход.

Правило первоначальной генерации поля таково:

$$\begin{cases} GF[0]=1, GF[1]=2, GF[255]=0, \\ i=2...254 \\ GF[i]=\begin{cases} (GF[i-1]<<1), GF[i-1]<128 \\ (GF[i-1])<<1)\oplus 285, GF[i-1]\geq 128 \end{cases} \end{cases}$$

Простыми словами, 0-й элемент поля это 1, 1-й элемент – 2, а, начиная со 2-го элемента по 254-й элемент, элемент вычисляется как удвоенное значение предыдущего элемента, и если удвоение привело к числу, вышедшему за границы 8-разрядов, то на него делается XOR с числом 285_{10} ($11D_{16}$), наконец, последний 255-й элемент поля – 0. Число 285 – это десятичное представление (11D в шестнадцатеричном представлении) так называемого неприводимого полинома $x^8 \oplus x^4 \oplus x^3 \oplus x^2 \oplus 1$, с помощью которого и порождается первоначальное поле. Символом ⊕ обозначается операция XOR – побитовое сложение по модулю 2, а символом << обозначается логический сдвиг влево двоичного представления числа на указанное количество разрядов, при этом биты, «вылезшие слева» из 8-разрядного байта, пропадают, а разряды, «освобождающиеся справа», заполняются нулями. Сдвиг числа в двоичном представлении на один разряд влево – это эквивалентно удвоению числа.

Согласно теории, сгенерированное поле GF(2^8), содержит результаты возведения примитивного элемента «2» во все степени, начиная с 0, заканчивая 255.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	2	4	8	16	32	64	128	29	58	116	232	205	135	19	38
16	76	152	45	90	180	117	234	201	143	3	6	12	24	48	96	192
32	157	39	78	156	37	74	148	53	106	212	181	119	238	193	159	35
48	70	140	5	10	20	40	80	160	93	186	105	210	185	111	222	161
64	95	190	97	194	153	47	94	188	101	202	137	15	30	60	120	240
80	253	231	211	187	107	214	177	127	254	225	223	163	91	182	113	226
96	217	175	67	134	17	34	68	136	13	26	52	104	208	189	103	206
112	129	31	62	124	248	237	199	147	59	118	236	197	151	51	102	204
128	133	23	46	92	184	109	218	169	79	158	33	66	132	21	42	84
144	168	77	154	41	82	164	85	170	73	146	57	114	228	213	183	115
160	230	209	191	99	198	145	63	126	252	229	215	179	123	246	241	255
176	227	219	171	75	150	49	98	196	149	55	110	220	165	87	174	65
192	130	25	50	100	200	141	7	14	28	56	112	224	221	167	83	166
208	81	162	89	178	121	242	249	239	195	155	43	86	172	69	138	9
224	18	36	72	144	61	122	244	245	247	243	251	235	203	139	11	22
240	44	88	176	125	250	233	207	131	27	54	108	216	173	71	142	0

Помимо основного поля в технологии кодирование важно также иметь и так называемое обратное поле, позволяющее по заданному значению 2^k выяснить степень k, в которое было возведен примитивный элемент 2, иными словами иметь таблицу логарифмов по основанию 2. Обратное поле вычисляется достаточно просто:

$$\begin{cases} i=0...255 \\ GF^{-1}[GF[i]]=i \end{cases}$$

Согласно теории, сгенерированное обратное поле $GF^{-1}(2^8)$, содержит логарифмы всех элементов, начиная с 0, заканчивая 255, по основанию примитивного элемента 2.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	255	0	1	25	2	50	26	198	3	223	51	238	27	104	199	75
16	4	100	224	14	52	141	239	129	28	193	105	248	200	8	76	113
32	5	138	101	47	225	36	15	33	53	147	142	218	240	18	130	69
48	29	181	194	125	106	39	249	185	201	154	9	120	77	228	114	166
64	6	191	139	98	102	221	48	253	226	152	37	179	16	145	34	136
80	54	208	148	206	143	150	219	189	241	210	19	92	131	56	70	64
96	30	66	182	163	195	72	126	110	107	58	40	84	250	133	186	61
112	202	94	155	159	10	21	121	43	78	212	229	172	115	243	167	87
128	7	112	192	247	140	128	99	13	103	74	222	237	49	197	254	24
144	227	165	153	119	38	184	180	124	17	68	146	217	35	32	137	46
160	55	63	209	91	149	188	207	205	144	135	151	178	220	252	190	97
176	242	86	211	171	20	42	93	158	132	60	57	83	71	109	65	162
192	31	45	67	216	183	123	164	118	196	23	73	236	127	12	111	246
208	108	161	59	82	41	157	85	170	251	96	134	177	187	204	62	90
224	203	89	95	176	156	169	160	81	11	245	22	235	122	117	44	215
240	79	174	213	233	230	231	173	232	116	214	244	234	168	80	88	175

Таким образом, используя эти две таблицы, мы сводим операции вычисления 2^i и $\log_2(i)$ к простому считыванию соответствующего элемента из основного или обратного поля Галуа. В обычной алгебре вещественных чисел о такой скорости и простоте возведения в степень и расчета логарифма даже и мечтать нельзя было.

Однако, мы еще не определили основные арифметические операции: сложение, вычитание, умножение и деление. Со сложением и вычитанием элементов полей все просто – для случая поля Галуа $GF(2^8)$ обе операции считаются эквивалентными и заменяются обычной операцией XOR – побитовое сложение по модулю 2.

Что касается умножения и деления, если вспомнить школьную алгебру, то логарифм от произведения двух чисел равна сумме логарифмов от каждого числа в отдельности, а логарифм от отношения двух чисел равна разности логарифмов от каждого числа в отдельности. Тогда операции умножения / деления можно свести к вычислению логарифмов по основанию 2 от операндов (по обратному полю Галуа), сложению / вычитанию значений логарифмов, и возведение числа 2 в степень суммы / разности (по основному полю Галуа).

Примечание 1. Для сложения и вычитания же степеней используются обычные правила алгебры, операция XOR здесь недопустима. Если сумма степеней больше или равна 255, то из нее вычитается 255. Если разность степеней меньше 0, то к ней прибавляется 255.

Примечание 2. Иногда требуется вычислить произведение степеней, в этой ситуации также используют умножение из обычной алгебры, однако, за результат берется остаток произведения по модулю 255.

Примечание 3. При операции умножения, если один из операндов равен нулю, то сразу возвращается результат 0. Иначе вычисляются логарифмы от операндов, и вычисляется сумма. Если сумма больше либо равна 255, то из суммы вычитается 255. После этого число 2 возводится в результирующую степень.

Примечание 4. При операции деления, если делитель равен нулю, то сразу возвращается ошибка «деления на нуль», а если делимое равно нулю, но делитель не равен нулю, то сразу возвращается 0. Иначе вычисляются логарифмы от операндов, и вычисляется их разность. Если разность меньше нуля, то к разности добавляется 255. После этого число 2 возводится в результирующую степень.

Таким образом, определим 4 арифметические операции:

$$a+b=a\oplus b$$

$$a-b=a\oplus b$$

$$a\cdot b=\begin{cases} a=0\,|\,b=0\Rightarrow 0 \\ a\neq 0\ \&\ b\neq 0\Rightarrow \begin{cases} \log_2 a+\log_2 b<255\Rightarrow 2^{\left(\log_2 a+\log_2 b\right)} \\ \log_2 a+\log_2 b\geq 255\Rightarrow 2^{\left(\log_2 a+\log_2 b-255\right)} \end{cases} \end{cases}$$

$$a/b=\begin{cases} a=0\ \&\ b\neq 0\Rightarrow 0 \\ a\neq 0\ \&\ b\neq 0\Rightarrow \begin{cases} \log_2 a-\log_2 b\geq 0\Rightarrow 2^{\left(\log_2 a-\log_2 b\right)} \\ \log_2 a-\log_2 b<0\Rightarrow 2^{\left(\log_2 a-\log_2 b+255\right)} \end{cases} \\ b=0\Rightarrow Ошибка \end{cases}$$

$$2^x=GF[x]$$

$$\log_2 x=GF^{-1}[x]$$

$$a,b,x\in GF(2^8)$$

Примечание 1. В поле Галуа нет отрицательных элементов, операции сложения и вычитания сведены к одной и той же операции XOR. Если формально говорить о «замене знака», то, воспользовавшись правилом операции вычитания, и положив $a=0$, мы увидим, что $-b=b$. Положительный элемент тождественно равен себе отрицательному.

Примечание 2. Вычисление элемента b^{-1}, при условии $b\neq 0$ и $b\neq 1$, сводится к делению элемента $2^0=1$ на b. Воспользовавшись правилом деления, видим, что $\log_2 1=0$, $\log_2 b\neq 0$, их разность меньше нуля, и результат равен $2^{255-\log_2 b}$. При $b=1$, $b^{-1}=1$, при $b=0$ – ошибка деления на нуль.

Примечание 3. Вычисление $2^{-u}=1/2^u$, при условии $0<u<255$, учитывая, что $\log_2 1=0$ и $\log_2 2^u=u\neq 0$, их разность меньше нуля, сводится к вычислению 2^{255-u}. При $u=0$, $2^{-u}=1$. При $u\geq 255$ степень предварительно берется по модулю 255.

Достаточно легко убедится в непротиворечивости алгебры конечного поля. Например, если сложить 10 и 15, получим результат 10 XOR 15 = 5, теперь если из 5 вычесть 15, получим 5 XOR 15 = 10, так что условие a = (a + b) – b соблюдается. Или, например, умножим 7 на 3, получим 2 ^ ($\log_2$ 7 + $\log_2$ 3) = 2 ^ (198 + 25) = 2 ^ (223) = 9, теперь если разделить 9 на 3, получим 2 ^ ($\log_2$ 9 – $\log_2$ 3) = 2 ^ (223 – 25) = 2 ^ (198) = 7, так что соблюдается условие, a = (a / b) * b. Между прочим, в случае традиционной алгебры в поле вещественных чисел, на ЭВМ это условие не факт что выполнится точно, поскольку при делении может произойти потеря точности, и при последующем умножении частного на делитель получится не точно изначальное делимое, а в алгебре конечных полей все абсолютно точно. Вообще, для ЭВМ вещественные числа и операции над ними глубоко противоестественны, ЭВМ рождена для работы в конечных полях.

2. Кратко о полиномах и операциях над ними.

Также как и в традиционной алгебре, в алгебре конечных полей [1] можно говорить о функциях, представляемых в виде степенного ряда – иными словами, полиномах. В частности, мы будем рассматривать функции одной переменной, причем сама переменная носит формальный характер, и сама по себе не имеет какой-либо смысловой нагрузки. В полиноме важны лишь коэффициенты, стоящие при этой переменной в соответствующей степени и мы будем касаться только случая, когда ими являются элементы поля Галуа.

$$A(x) = A_{k-1}x^{k-1} \oplus \ldots \oplus A_1 x^1 \oplus A_0 = \sum_{i=0}^{k-1} A_i x^i$$

$$A_i \in GF(2^8), i = 0 \ldots k-1$$

Примечание 1. Особо обратим внимание на то, что сам символ суммы, подразумевает операцию сложения по правилам алгебры поля Галуа, иными словами, операцию XOR.

Примечание 2. При программной реализации полиномов, они представляются в виде массива коэффициентов, сама же формальная переменная, и ее степени нигде не хранятся. «Степень переменной», при которой стоит коэффициент однозначно определяется, как позиция (индекс) элемента массива, в котором хранится коэффициент.

Одна из наиболее важных характеристик полинома – это его степень, определяемая как наивысшая степень переменной, перед которой стоит ненулевой коэффициент.

$$\deg(A(x)) = \max_{i=0 \ldots k-1} \{i : A_i \neq 0\}$$

$$A_i \in GF(2^8), i = 0 \ldots k-1$$

Над полиномами можно производить арифметические операции, также на место формальной переменной можно подставлять конкретное значение, являющееся элементом поля Галуа, и вычислять значение функции. При пересчете коэффициентов полинома при выполнении операции или вычислении значении функции, строго соблюдаются правила арифметики для поля Галуа. Основные операции, которые нам потребуются в дальнейшем – это сложение / вычитание полиномов (в нашем случае обе операции сводятся к операции XOR), масштабирование полинома, сдвиг влево на одну позицию, умножение полиномов и вычисления остатка от деления одного полинома на другой полином.

Примечание 3. При перемножении полиномов, очевидно, что массив коэффициентов результирующего полинома имеет больший размер, равный сумме размеров массивов коэффициентов перемножаемых полиномов. При сдвиге полинома влево на одну позицию, массив коэффициентов результирующего полинома имеет размер на единицу больше, чем размер массива коэффициентов исходного полинома.

Сложение / вычитание двух полиномов выполняется путем выполнения операции XOR для коэффициентов, стоящих перед переменной в одной степени. Отсутствующие в полиномах коэффициенты, разумеется, считаются нулевыми.

$$A(x) \oplus B(x) = \sum_{i=0}^{k-1} \left(A_i \oplus B_i\right) x^i$$

$$A_i, B_i \in GF(2^8), i = 0 \ldots k-1$$

Масштабирование полинома – умножение полинома на константу (являющуюся элементом поля Галуа), которое сводится к умножению всех коэффициентов этого полинома на константу по правилам поля Галуа.

$$\lambda \cdot A(x) = \sum_{i=0}^{k-1} \left(\lambda \cdot A_i\right) x^i$$

$$A_i, \lambda \in GF(2^8), i = 0 \ldots k-1$$

«Сдвиг влево» полинома на одну позицию эквивалентно умножению полинома на x, соответственно, при каждом коэффициенте степени переменной увеличивается на единицу, с другой точки зрения – коэффициенты «сдвигаются» на одну позицию влево, а на место младшего коэффициента в результирующем полиноме записывается нуль.

$$x \cdot A(x) = \sum_{i=0}^{k-1} A_i x^{i+1} = \sum_{i=0}^{k} A_i^* x^i$$

$$A_{i+1}^* = A_i, A_0^* = 0, i = 0 \ldots k-1$$

Умножение двух полиномов осуществляется путем вычисления суммы результатов произведения одного полинома на каждый член другого полинома. Каждое произведение сводится к масштабированию одного полинома каждым коэффициентом другого полинома и сдвигу результирующего полинома влево на число позиций, равному степени переменной, при которой стоит коэффициент в другом полиноме, при этом освобождающиеся справа позиции заполняются нулями.

$$A(x) \cdot B(x) = \sum_{i=0}^{k-1} A_i x^i \cdot \left(\sum_{j=0}^{l-1} B_j x^j \right) = \sum_{r=0}^{k+l-2} x^r \left(\sum_{\substack{i=0 \ldots k-1 \\ j=0 \ldots l-1 \\ i+j=r}} A_i \cdot B_j \right)$$

$$A_i, B_j \in GF(2^8), i = 0 \ldots k-1, j = 0 \ldots l-1$$

Вычисление остатка от деления одного полинома на другой полином, или более коротко – вычисление остатка одного полинома по модулю другого. Сделаем сразу оговорку, что нам понадобится только частный случай, когда старший коэффициент полинома делителя равен единице, и это существенно упрощает процедуру деления.

Пусть задан полином-делимое $A(x)$ степени $k-1$, и полином-делитель $g(x)$ степени r, причем $r \le k-1$ и $g_r = 1$. Деление полиномов осуществляется по правилам, похожим на правила «деления в столбик» в традиционной арифметике. Однако, есть и различия.

Во-первых, в отличие от традиционного деления, на каждом шаге деления «в столбик» выполняется не вычитание, а операция XOR между соответствующими коэффициентами соответствующих полиномов, о них пойдет речь ниже. При этом на каждом шаге $s = 1 \ldots k-r$ получается остаток $R^{(s)}(x)$, который всегда состоит строго из r коэффициентов (столько же, сколько в делителе), старшие коэффициенты могут быть нулевыми, и, тем не менее, эти «нули слева» никогда не отбрасываются и являются частью остатка. Изначально, за «начальный остаток» принимается следующий полином:

$$R_j^{(0)} = \begin{cases} A_{k-r+j}, j = 0 \ldots r-1 \\ 0, j = r \end{cases}$$

Во-вторых, при традиционном делении, на очередном шаге s остаток может получаться разной длины (с разным количеством цифр), и в зависимости от ситуации, «сверху спускаются» необходимое количество цифр делимого так, чтобы сформированное таким образом число было больше либо равно делителя. Если «сверху спускается» количество цифр $w > 1$, то в частном справа дописывают $w-1$ нулей. При делении же полиномов «сверху спускается» всегда только один очередной коэффициент A_{k-r-s} делимого полинома $A(x)$, на соответствующем шаге $s \in [1, k-r]$.

В-третьих, в отличие от традиционного деления, «спущенный сверху» коэффициент A_{k-r-s} не дописывается справа, а делается по-другому: остаток $R^{(s)}(x)$, полученный на предыдущем шаге, сдвигается на одну позицию влево, а на место младшей позиции, записывается этот коэффициент.

$$R_j^*(s) = \begin{cases} R_{j-1}^{(s-1)}, j = 1 \ldots r \\ A_{k-r-s}, j = 0 \end{cases}$$

Очевидно, при этом старший коэффициент $R_r^{(s-1)}$ старого полинома-остатка пропадает, но в этом беды нет, так как на каждом шаге $s \in [1, k-r]$ деления получается остаток, у которого старший коэффициент равен нулю, $R_r^{(s)} = 0$, это будет показано ниже. Кроме того, как было показано ранее, в «начальном остатке» также $R_r^{(0)} = 0$.

В-четвертых, при традиционном делении, на каждом шаге после присоединения к очередному остатку необходимого количества цифр, «спущенных сверху» от делимого, подбирается очередная цифра частного так, чтобы делитель, умноженный на эту цифру, при его вычитании из сформированного выше числа давал неотрицательную разность, причем меньшую, чем сам делитель. В случае же деления полиномов, очередной коэффициент частного $Q^{(s)}$ выбирается так, чтобы старший коэффициент g_r полинома-делителя, умноженного на коэффициент частного (масштабированного коэффициентом частного), был равен старшему коэффициенту полинома $R_r^*(s)$, являющегося результатом сдвига влево на одну позицию остатка, полученного на предыдущем шаге. Иными словами, чтобы соблюдалось $Q^{(s)} \cdot g_r = R_r^*(s)$. Учитывая то, $g_r = 1$, нетрудно заметить, что обеспечить это условие достаточно просто, сделав «обратную связь», по которой передается старший коэффициент полинома, являющегося результатом сдвига влево на одну позицию остатка, полученного на предыдущем шаге, и именно он принимается за очередной коэффициент частного. Иными словами, $Q^{(s)} = R_r^*(s) = R_{r-1}^{(s-1)}$.

После этого, остаток на текущем шаге $s = 1 \ldots k-r$ вычисляется как:

$$R_j^{(s)} = R_j^*(s) \oplus g_j \cdot R_r^*(s), j = 0 \ldots r$$

Учитывая, что $g_r = 1$, заметим, что $R_r^{(s)} = R_r^*(s) \oplus R_r^*(s) = 0$. Тогда, окончательно получаем рекуррентное соотношение для вычисления остатка на шаге $s = 1 \ldots k-r$:

$$R_j^{(s)} = \begin{cases} R_{j-1}^{(s-1)} \oplus g_j \cdot R_{r-1}^{(s-1)}, j = 1 \ldots r \\ A_{k-r-s} \oplus g_0 \cdot R_{r-1}^{(s-1)}, j = 0 \end{cases}$$

Тогда после выполнения последнего шага $s = k-r$, мы будем иметь искомый остаток от деления полинома $A(x)$ на полином $g(x)$: $A(x) \bmod g(x) = R^{(k-r)}(x)$.

Заметим, что старший $r-й$ коэффициент полинома-остатка будет всегда равен нулю, и фактическая степень полинома-остатка $\deg R(x) \le r-1$, поэтому именно полином $R_{r-1}^{(k-r)} x^{r-1} \oplus \ldots \oplus R_1^{(k-r)} x \oplus R_0^{(k-r)}$ и принимается за остаток.

Пример 1. Вычислим остаток полинома $A(x) = 49 \cdot x^4 \oplus 50 \cdot x^3 \oplus 51 \cdot x^2 \oplus 0 \cdot x \oplus 0$ по модулю полинома $g(x) = x^2 \oplus 6 \cdot x \oplus 8$.

$$R^{(0)}(x) = 0 \cdot x^5 \oplus \overbrace{49}^{A_4} \cdot x^4 \oplus \overbrace{50}^{A_3} \cdot x^3$$

$$\leftarrow$$

$$\oplus\ 49 \cdot x^4 \oplus 50 \cdot x^3 \oplus \overbrace{51}^{A_2} \cdot x^2$$

$$\begin{array}{r|l} 49 \cdot x^4 \oplus 166 \cdot x^3 \oplus 149 \cdot x^2 & g(x) = x^2 \oplus 6 \cdot x \oplus 8 \\ \hline 0 \cdot x^4 \oplus 148 \cdot x^3 \oplus 166 \cdot x^2 & Q(x) = 49 \cdot x^2 \oplus 148 \cdot x \oplus 249 \end{array}$$

$$\leftarrow$$

$$\oplus 148 \cdot x^3 \oplus 166 \cdot x^2 + \overbrace{0}^{A_1} \cdot x$$

$$\begin{array}{r} 148 \cdot x^3 \oplus 95 \cdot x^2 + 212 \cdot x \\ \hline 0 \cdot x^3 \oplus 249 \cdot x^2 \oplus 212 \cdot x \end{array}$$

$$\leftarrow$$

$$\oplus 249 \cdot x^2 \oplus 212 \cdot x \oplus \overbrace{0}^{A_0}$$

$$\begin{array}{r} 249 \cdot x^2 \oplus 44 \cdot x \oplus 155 \\ \hline R(x) = 0 \cdot x^2 \oplus 248 \cdot x \oplus 155 \end{array}$$

В результате получаем остаток: $R(x) = 248 \cdot x \oplus 155$.

Пример 2. Проверим корректность операции деления в предыдущем примере, вычислив выражение $Q(x) \cdot g(x) \oplus R(x)$. Сложив произведение частного и делителя с остатком, мы должны снова получить исходное делимое.

$$Q(x) \cdot g(x) \oplus R(x) = \left(49 \cdot x^2 \oplus 148 \cdot x \oplus 249\right) \cdot \left(x^2 \oplus 6 \cdot x \oplus 8\right) \oplus \left(248 \cdot x \oplus 155\right) =$$

$$= \left(49 \cdot 1 \cdot x^4 \oplus \left(49 \cdot 6 \oplus 148 \cdot 1\right) \cdot x^3 \oplus \left(49 \cdot 8 \oplus 148 \cdot 6 \oplus 249 \cdot 1\right) \cdot x^2 \oplus \left(148 \cdot 8 \oplus 249 \cdot 6\right) \cdot x \oplus 249 \cdot 8\right)$$

$$\oplus \left(248 \cdot x \oplus 155\right) =$$

$$\left(49 \cdot x^4 \oplus 50 \cdot x^3 \oplus 51 \cdot x^2 \oplus 248 \cdot x \oplus 155\right) \oplus \left(248 \cdot x \oplus 155\right) = 49 \cdot x^4 \oplus 50 \cdot x^3 \oplus 51 \cdot x^2$$

Видим, что $Q(x) \cdot g(x) \oplus R(x) = 49 \cdot x^4 \oplus 50 \cdot x^3 \oplus 51 \cdot x^2$ не что иное, как исходный полином-делимое $A(x)$ в предыдущем примере. Это подтверждает корректность процедуры вычисления остатка одного полинома по модулю другого полинома.

Нетрудно заметить, что как в рекуррентном соотношении для вычисления остатка на шаге $s = 1 \ldots k - r$, так и в приведенном примере, явно видно, что на каждом шаге вычисляется старший коэффициент остатка $R_r^{(s)}$, который, во-первых, нигде потом не используется, а во вторых, после каждого шага становится нулевым. И здесь, если мы для большей наглядности операции деления «держали для своих нужд» этот коэффициент, то в практической реализации, очевидно, делать подобное совершенно избыточно.

Что касается очередного коэффициента частного, то его можно взять из остатка, вычисленного на предыдущем шаге $Q^{(s)} = R_{r-1}^{(s-1)}$, иными словами, взять предпоследний $(r-1)-й$ коэффициент остатка, полученного на предыдущем шаге, до того как мы выполним его сдвиг на текущем шаге. Тогда с учетом сказанного, окончательно получаем математическую модель процедуры вычисления остатка полинома $A(x)$ по модулю $g(x)$:

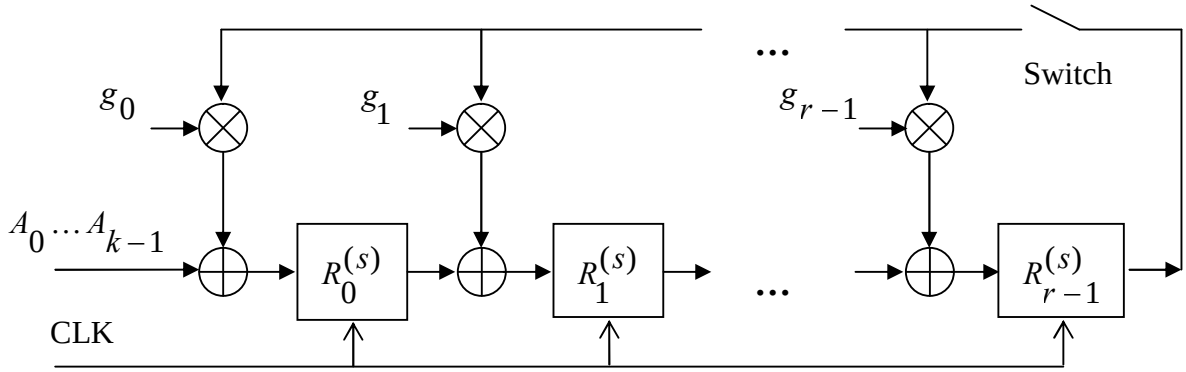
$$R_j^{(0)} = A_{k-r+j}, j = 0 \ldots r-1$$

$$R_j^{(s)} = \begin{cases} R_{j-1}^{(s-1)} \oplus g_j \cdot R_{r-1}^{(s-1)}, j = 1 \ldots r-1 \\ A_{k-r-s} \oplus g_0 \cdot R_{r-1}^{(s-1)}, j = 0 \end{cases}$$

$$s = 1 \ldots k - r$$

Следует отметить, что процедура фактически сводится к последовательности операций сдвига, скажем так, некоторого r – разрядного регистра (в нашем случае разряд – это не один бит, а целый байт – 8-битное число), хранящего остаток $R_{r-1}^{(s-1)} \ldots R_0^{(s-1)}$, полученный на предыдущем шаге, и сложения его с r – разрядным делителем (без старшего коэффициента $g_r = 1$), умноженного на старший коэффициент остатка $R_{r-1}^{(s-1)}$. При сдвиге регистра остатка в его младший $0-й$ разряд загружается очередной, коэффициент A_{k-r-s} полинома-делимого. Перед началом процедуры, в регистр остатка загружаются коэффициенты $A_{k-1} \ldots A_{k-r}$ исходного полинома. Подобная схема легко реализуется аппаратно в виде сдвигового регистра с обратной связью.

В приведенной ниже схеме, на первой фазе, ключ обратной связи Switch разомкнут и в течение r тактов в регистре остатка загрузятся коэффициенты $A_{k-1} \ldots A_{k-r}$, поступающие через младший разряд регистра. Поскольку ключ разомкнут, на схемах сложения они будут складываться с нулем, и поступят в регистр в исходном виде. Во второй фазе, ключ (Switch) замкнут, и в течение $k - r$ тактов будет вычислен остаток. Искомый остаток можно будет считать из регистра после последнего такта. CLK – вход для тактовых сигналов, по нему «разряды» регистра принимают информацию со своего входа.



3. Введение в теорию кодирования. Коды Рида-Соломона.

Мы не зря столь подробно рассматривали конечные поля, конкретно поле Галуа GF(2^8), и правила арифметики в них, а также полиномы и операции над ними в поле Галуа, поскольку технология кодирования и декодирования с применением так называемых кодов Рида-Соломона [2, 5, 6, 7] опирается именно на них.

Идея предельно проста, в ЭВМ основной единицей информации является байт, в котором может быть закодировано одно из 256 значений, именно это и обусловливает выбор не какого-либо конечного поля, а именно поля Галуа GF(2^8), которое содержит 256 различных элементов. Любое информационное сообщение или блок данных могут быть рассмотрены, как последовательности байтов. Более того, можно ввести формальную переменную x и тогда эти байты могут рассматриваться как коэффициенты полинома, причем, очевидно, что степень полинома на единицу меньше длины сообщения.

Иными словами, пусть имеется некоторое сообщение, состоящее из k байтов.

M_{k-1}	…	M_1	M_0

Тогда мы можем его также представить в виде полинома степени k – 1:

$$M(x) = M_{k-1}x^{k-1} \oplus \ldots \oplus M_1 x^1 \oplus M_0 = \sum_{i=0}^{k-1} M_i x^i$$

$$M_i \in GF(2^8), i = 0 \ldots k-1$$

С одной стороны каждый байт сообщения может содержать любое из 256 возможных значений, с другой стороны поле Галуа GF(2^8) также содержит всевозможные 256 различных элемента, так что никаких проблем не возникает. Просто в полиномиальном представлении «байты» сообщения уже являются не просто «байтами», они также трактуются как коэффициенты полинома, являющиеся элементами поля Галуа GF(2^8).

Важным понятием в теории кодирования [5] является, **кодовое расстояние**. Пусть заданы два сообщения A и B одинаковой длины k. Упрощенно, кодовое расстояние – это число байтов, которое у них отличаются значениями в соответствующих позициях. Например, кодовое расстояние между текстовыми сообщениями **ABCDE** и **AFGDE** равно 2. Кодовое расстояние между сообщениями **ABCDE** и **ACBDE** также будет равно 2, поскольку имеется несовпадение в двух позициях, несмотря на то, что в обоих сообщениях участвуют символы из одного и того же набора. Более строго дать определение кодового расстояния можно, если представить сообщения в виде полиномов.

Пусть заданы два сообщения A и B одинаковой длины k, которые могут быть представлены соответствующими полиномами $A(x) = A_{k-1}x^{k-1} \oplus \ldots \oplus A_1 x^1 \oplus A_0$ и $B(x) = B_{k-1}x^{k-1} \oplus \ldots \oplus B_1 x^1 \oplus B_0$. Тогда кодовое расстояние $D(A,B)$ между этими сообщения определяется как:

$$\begin{cases} D(A,B) = \sum_{j=0}^{k-1} \sigma_j \\ \sigma_j = \begin{cases} 0, A_j \oplus B_j = 0 \\ 1, A_j \oplus B_j \neq 0 \end{cases} \end{cases}$$

Всевозможные варианты различных сообщений длины k образуют, так называемое, k-мерное пространство сообщений. Очевидно, что различные сообщения пространства могут отличаться друг от друга в количестве позиций от 1 до k, иными словами кодовое расстояние между различными сообщениями пространства находится в пределах: $1 \le D \le k$.

В теории кодирования исключительно важной характеристикой пространства сообщений является **минимальное кодовое расстояние** d, иными словами, наименьшее количество позиций в которых могут различаться два разных сообщения из заданного пространства. В нашем случае эта величина равна единице, поскольку два разных сообщения отличаются, как минимум, в одной позиции, иначе они просто совпадают.

$$d = \min_{A,B \in Z^k} \{D(A,B)\} = 1$$

$$Z = \{0,\ldots,255\}$$

Что еще более важно – любое сообщение, принадлежащее k-мерному пространству сообщений, является, так сказать, «легальным», «допустимым», имеющим право на существование. Действительно, нам может понадобиться представить в сообщении самые разные комбинации значений байтов, для кодирования произвольных видов информации и мы не можем накладывать здесь никаких ограничений. Именно это препятствует нам каким-либо образом обнаруживать и, тем более уж, исправлять искаженные сообщения.

Чтобы появилась возможность обнаружения (а также, частично, и исправления) искажений в сообщениях, согласно теории кодирования требуется расширить пространство сообщений, путем увеличения длины сообщений за счет введения, так называемых, избыточных символов: в нашем случае – избыточных байтов. Но мало просто ввести избыточные байты, необходимо также расширенное пространство сообщений строго поделить на подпространство «допустимых» («легальных») сообщений и подпространство «недопустимых» («нелегальных») сообщений. Только в этом случае у нас будет возможность при приеме или считывании сообщения, выявить «допустимость» сообщения. Очевидно также, что значения избыточных байтов, должны как-то быть связаны со значениями информационных байтов, а не быть произвольными, поскольку в таком случае мы будем иметь всего лишь расширенное пространство, в котором все сообщения «допустимы».

Пусть задано исходное k-мерное пространство информационных сообщений (множество всевозможных вариантов информационных сообщений длины k). Расширим это пространство путем введения избыточных r байтов до $n = k + r$-мерного пространства, причем предъявим следующее ключевое требование: r избыточных байтов должны зависеть от k информационных байтов таким образом, чтобы минимальное кодовое расстояние подпространства «допустимых» сообщений было $d = r + 1$. Иными словами чтобы «допустимые» сообщения, вместе с корректно вычисленными для них значениями избыточных байтов, отличались друг от друга не менее чем в $d = r + 1$ позициях.

В этом случае согласно теории кодирования при приеме или считывании сообщения мы сможем обнаруживать искажение сообщения, вплоть до в r позициях. Более того, теория кодирования утверждает, что можно также обеспечить гарантированное исправление ошибок при искажениях вплоть до в $\lfloor r/2 \rfloor$ позициях. Мы не будем излагать здесь строгих доказательств этих утверждений. Интуитивно понятно, что мы можем гарантированно обнаружить искажение, только если сообщение M^* «отклонилось» от исходного «допустимого» сообщения M на кодовое расстояние менее чем d, т.е. ($D(M^*, M) < d$). При отклонении уже на расстояние d сообщение может попросту совпасть с другим «допустимым» сообщением и ошибка не будет обнаружена. Что же касается возможности исправления, мы гарантированно можем исправить ошибки, только если сообщение M^* «отклонилось» от исходного «допустимого» сообщения M менее чем половину минимального кодового расстояния $D(M^*, M) < d/2$, и в этом случае мы можем «притянуть» обратно искаженное сообщение к его исходному «допустимому» виду. В противном случае, если сообщение «отклонилось» на половину или большее расстояние от исходного сообщения, то оно уже «ближе» к другому «допустимому» сообщению и корректное исправление (восстановление исходного сообщения) невозможно.

Обозначим, через t - максимальную кратность исправляемой ошибки, иными словами, предельное число искажаемых байтов, при котором еще возможно их гарантированное исправление. Учитывая предыдущие рассуждения, несложно заметить, что этот параметр однозначно связан с требованием к минимальному кодовому расстоянию и требуемым количеством избыточных байтов. Для того, чтобы исправлять ошибки вплоть до t-кратных, минимальное кодовое расстояние подпространства «допустимых» сообщений должно быть, как минимум, быть больше чем удвоенная максимальная кратность исправляемых ошибок, иными словами, $d > 2t$. Как минимум, мы должны обеспечить минимальное кодовое расстояние, хоты бы на единицу больше, удвоенной максимальной кратности исправляемой ошибки, иными словами $d = 2t + 1$. Тогда, учитывая что $d = r + 1$, нам понадобятся $r = 2t$ избыточных байтов, чтобы обеспечить минимальное требование. Заметим также, из приведенных выше рассуждений следует, что при таком минимальном кодовом расстоянии возможно обнаружение ошибок вплоть до $d - 1 = r = 2t$ кратных.

Таким образом, пусть задано пространство сообщений длины k. Тогда, введя, $r = 2t$ избыточных байтов, то есть, расширив пространство до $n = k + r$-мерного, и наложив главное требование о том, что подпространство «допустимых» сообщений должно обладать минимальным кодовым расстоянием $d = r + 1 = 2t + 1$, мы обеспечим обнаружение ошибок кратности вплоть до $2t$, и исправление ошибок, вплоть до кратности t. Требование вполне логичное и выполнимое. Мы имели пространство сообщений длины k, для хранения которых требовалось k байт, которые отличались в худшем случае только в одном байте. Потом же, «пожертвовав» избыточными $r = 2t$ байтами, для хранения избыточной информации, которая должна однозначно вычисляться из исходной «полезной» информации, хотим получить подпространство «допустимых» сообщений длины $n = k + r$, которые будут отличаться в худшем случае не менее, чем $d = r + 1 = 2t + 1$ байтами.

Ключевой вопрос в том, как же именно вычислять избыточные байты, так чтобы с помощью них любое сообщение длины k исходного пространства с минимальным кодовым расстоянием $d = 1$, можно было преобразовать в сообщение длины $n = k + r$ подпространства с минимальным кодовым расстоянием $d = r + 1$.

Теория кодирования предлагает следующий подход с использованием, так называемых кодов Рида-Соломона, определенных в конечном поле Галуа GF(2^8). Вводится понятие, так называемого **порождающего полинома**:

$$g(x) = \prod_{i=1}^{2t} \left(x \oplus 2^i \right)$$
$$2 \in GF(2^8)$$

Порождающий полином согласно теории кодирования и обеспечивает «генерацию» $r = 2t$ избыточных байтов из k байтов исходного сообщения (причем любого), так чтобы результирующее «расширенное» сообщение, состоящее из $n = k + r$ байтов, получающееся в результате соединения байтов исходного сообщения с избыточными байтами, принадлежало $n = k + r$-мерному подпространству с минимальным кодовым расстоянием $d = r + 1$. Нетрудно заметить, что порождающий полином имеет степень $r = 2t$, и его корнями являются элементы поля Галуа GF(2^8), соответствующие результатам возведения примитивного элемента 2 в степени $1,2 \ldots 2t$, иными словами, элементы: $2^1, 2^2 \ldots, 2^{2t}$.

Примечание. Вычислим для примера, порождающий полином для случая построения кода с возможностью исправления одиночных ошибок и случая ошибок двойной кратности:

- $t = 1$: $g(x) = \prod_{i=1}^{2} \left(x \oplus 2^i \right) = x^2 \oplus 6x \oplus 8$
- $t = 2$: $g(x) = \prod_{i=1}^{4} \left(x \oplus 2^i \right) = x^4 \oplus 30x^3 \oplus 216x^2 \oplus 231x \oplus 116$

Пусть задан исходный полином $M(x) = M_{k-1}x^{k-1} \oplus \ldots \oplus M_1 x^1 \oplus M_0$ исходного информационного сообщения. Пусть задан полином $R(x)$, вычисленный как остаток от деления полинома $M(x)$, сдвинутого на r позиций влево (другими словами, умноженного на x^r), на порождающий полином $g(x)$. Иными словами, пусть задан полином:

$$R(x) = \left(x^r \cdot M(x)\right) \bmod (g(x)) = \sum_{i=0}^{r-1} R_i x^i$$

Тогда **кодом Рида-Соломона** называют коэффициенты информационного полинома $M(x)$ и полинома остатка $R(x) = \left(x^r \cdot M(x)\right) \bmod (g(x))$, представленные в следующем виде:

M_{k-1}	…	M_0	R_{r-1}	…	R_0

Следует отметить, что такое кодирование называется систематическим в силу того, что «информационные» коэффициенты можно легко отделить от коэффициентов остатка без какого-либо специального декодирования.

Говоря в терминологии криптографии, код Рида-Соломона – не что иное как, исходное сообщение и присоединенный к нему некий «хэш», вычисленный по сообщению.

Нетрудно заметить, что код Рида-Соломона представляет коэффициенты некоторого полинома $F(x)$ вычисляемого как сумма сдвинутого на r позиций влево информационного полинома $M(x)$ и полинома остатка $R(x)$:

$$F(x) = \left(x^r \cdot M(x)\right) \oplus R(x)$$

Относительно полинома $F(x)$, в теории кодирование делается два достаточно простых, но в то же время чрезвычайно важных утверждения:

- Полином $F(x)$ делится без остатка на порождающий полином $g(x)$.
- Полином $F(x)$ имеет те же корни, что и полином $g(x)$: элементы: $2^1, 2^2 \ldots, 2^r$.

Как первое, так и второе утверждение особенно важны тем, что именно на нем и базируется вся логика обнаружения ошибок. Если коэффициенты полинома $F(x)$ не искажены, то его деление на порождающий полином даст нулевой остаток, так же как подстановка в него любого из корней порождающего полинома обратит его в нуль. Если же коэффициенты полинома $F(x)$ были искажены (при передаче по сети или при хранении на носителе информации), то с очень высокой вероятностью, а при искажении не более r байтов со 100% гарантией, деление на порождающий полином даст ненулевой остаток, также как и подстановка корней порождающего полинома будут давать ненулевые результаты.

Здесь работает та техническая уловка, что «окружающая среда», конечно, враждебна к нашей информации и пытается случайным непредсказуемым образом ее искажать, но «она не настолько умна», чтобы искажать информацию так, чтобы искаженный полином, так же как и неискаженный полином делился без остатка на порождающий полином. Слишком уж «мудрые расчеты», и окружающая среда уж точно в них «не вникает» (в отличие, кстати, от человека-злоумышленника), она просто «портит» данные случайным образом и все. Вероятность того, что искаженный полином разделится без остатка на порождающий полином, весьма невелика при искажении более r байтов, и с увеличением числа избыточных байт она становится еще гораздо меньше, а при искажении не более r байтов, вероятность такого исхода вообще нулевая – это теоретически доказано и экспериментально подтверждено соответствующими учеными-специалистами.

Тогда с учетом вышесказанного можем представить несложную схему алгоритма кодирования (рис. 1) информационных сообщений с применением кодов Рида-Соломона, при заданной длине сообщения – k и максимальной кратности исправляемой ошибки – t.

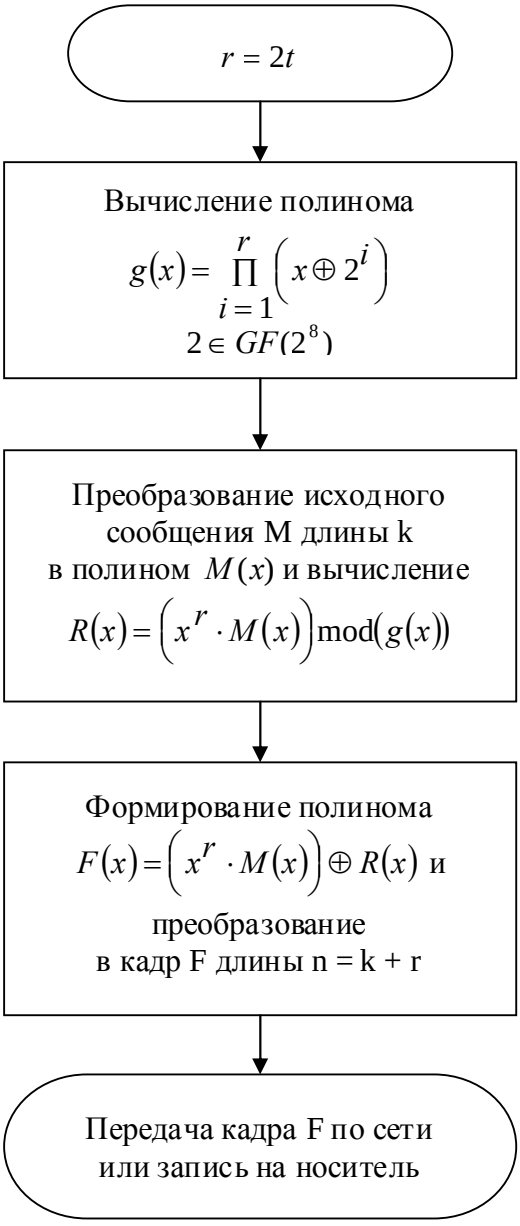


Рис. 1. Схема алгоритма кодирования информационного сообщения с применением кода Рида-Соломона

При передаче кадра по сети или при хранении кадра данных на носителе, информация может быть искажена в силу тех или иных физических причин: шумы в канале передаче данных или повреждение носителя данных. В таком случае можно говорить, что на кадр F будет наложено некоторое искажение E, или иными словами, полином кадра $F(x)$ будет складываться вместе с некоторым, так называемым, **полиномом искажения** $E(x)$ и в результате будем иметь **искаженный полином** $C(x) = F(x) \oplus E(x)$ на момент приема сообщения из сети или чтении данных с носителя. Искажаться могут любые коэффициенты полинома (байты кадра), как информационные, так и избыточные.

В итоге будем иметь, так называемый, **искаженный кадр** C.

C_{n-1}	…	C_1	C_0

4. Декодирование кодов Рида-Соломона

Пусть имеется кадр длины $n = k + r$, состоящий из k информационных и r избыточных байтов, принятый из сети или считанный с носителя, полином $C(x)$ которого является суммой полинома $F(x)$ исходного неискаженного кадра, отправленного по сети или записанного на носитель, и некоторого полинома искажения $E(x)$:

$$C(x) = F(x) \oplus E(x)$$

Тогда будем называть **синдромом ошибки** (синдром – совокупность симптомов, характеризующее заболевание) коэффициенты $S_j, j = 1 \ldots r$, вычисляемые подстановкой в полином $C(x)$ корней $2^1, 2^2 \ldots, 2^r$ порождающего полинома $g(x)$.

Тогда полином синдрома ошибок $S(x)$ может быть представлен как:

$$S(x) = \sum_{j=1}^{r} C(2^j) x^{j-1}$$

Заметим, что коэффициенты полинома синдрома нумеруются с 1 по r. Кроме того, учитывая то, что мы ранее установили, что полином $F(x)$ неискаженного кадра обращается в нуль при подстановке в него корней $2^1, 2^2 \ldots, 2^r$, то справедливо следующее равенство:

$$S_j = C(2^j) = E(2^j)$$
$$j = 1 \ldots r$$

Иными словами, синдром не зависит от самого неискаженного кадра *F* и полностью характеризуется только ошибкой *E*. Именно на этом и базируется вся технология расшифровки синдрома с целью нахождения местоположения ошибок и их значений.

Однако, несмотря на то, что коэффициенты синдрома так легко определяются по полиному искажения $E(x)$, обратное же вычисление полинома $E(x)$ по известным коэффициентам полинома синдрома ошибок $S(x)$ весьма сложная и нетривиальная задача. Ситуация ухудшается еще тем, что неизвестно сколько именно байтов было искажено.

Пусть, τ – предполагаемое количество искаженных байтов в кадре *C*.

Очевидно, что если полином синдрома равен нулю $S(x) = 0$ (все его коэффициенты нулевые), то можно говорить, что искажений либо не было, либо произошло искажение, не обнаруживаемое при заданном количестве избыточных байтов в силу того, что кратность произошедшей ошибки больше кратность обнаруживаемой ошибки, то есть $\tau > r = 2t$.

В противном случае, если синдром ненулевой, то можно декодировать его. Введем понятие, так называемого, **полинома локаторов ошибок** следующего вида:

$$\Lambda(x) = 1 \oplus \sum_{i=1}^{\tau} \Lambda_i x^i = \prod_{i=1}^{\tau} \left(1 \oplus x^i \cdot 2^{u_i} \right)$$

Очевидно, что корнями полинома локатора являются элементы $2^{-u_1}, \ldots, 2^{-u_\tau}$, и согласно теории кодирования сами степени $u_1, \ldots, u_\tau$, это не что иное, как локаторы местоположения ошибок в принятом из сети (считанном с носителя) кадре *C*.

Согласно теории кодирования, коэффициенты полинома синдрома $S(x)$ и полинома локаторов $\Lambda(x)$ связаны так называемой **ключевой системой уравнений декодирования**:

$$S_j = \sum_{i=1}^{\tau} \Lambda_i S_{j-i}$$
$$j = \tau + 1 \ldots 2t$$

Если переписать систему в развернутом виде, то эта система выглядит так:

$$\begin{cases} \Lambda_1 S_\tau \oplus \ldots \oplus \Lambda_\tau S_1 = S_{\tau+1} \\ \Lambda_1 S_{\tau+1} \oplus \ldots \oplus \Lambda_\tau S_2 = S_{\tau+2} \\ \vdots \\ \Lambda_1 S_{2t-1} \oplus \ldots \oplus \Lambda_\tau S_{2t-\tau} = S_{2t} \end{cases}$$

Как видим, это не просто система уравнений, а система из $2t-\tau$ уравнений с τ неизвестными, причем сам параметр τ неизвестен.

- При $\tau = 0$, случай отсутствия ошибок, система вырождается, и нечего решать.
- При $1 \le \tau \le t$, система имеет однозначное решение, поскольку число уравнений $(2t-\tau) = 2t-1 \ldots t$ больше либо равно (при $\tau = t$), чем соответствующее число неизвестных $\tau = 1 \ldots t$. Предельный случай, когда система все еще имеет однозначное решение – это $\tau = t$, при этом имеем $\tau = t$ уравнений и $\tau = t$ неизвестных.
- При $t+1 \le \tau \le 2t-1$, система не имеет однозначного решения, поскольку число уравнений $(2t-\tau) = t-1 \ldots 1$, оказывается меньше, чем число неизвестных $\tau = t+1 \ldots 2t-1$, и однозначное нахождение локаторов становится невозможным.
- При $\tau \ge 2t$, система опять же вырождается, и решение невозможно.

Задача нахождения полинома локаторов могла бы привести к необходимости полного перебора всевозможных решений при различных $\tau = 1 \ldots 2t-1$, однако, здесь снова приходит на помощь теория кодирования. Она утверждает, что либо однозначного решения не существует, либо существует однозначное решение при $1 \le \tau \le t$, и при этом подходит не первое попавшееся решение, а то решение, при котором степень полинома локаторов $\Lambda(x)$, равная τ, минимальна. Согласно теории именно в этом случае, полином локаторов наименьшей степени, коэффициенты которого удовлетворяют всей системе уравнений, и является **истинным полиномом локаторов ошибок**.

Тогда, окончательно, имеем следующую математическую модель задачи нахождения полинома локаторов – система $2t-\tau$ уравнений с τ неизвестными с минимизацией параметра τ – степени полинома-решения (она же и предполагаемая кратность ошибки).

$$\left\{ \begin{array}{c} S_j = \sum_{i=1}^{\tau} \Lambda_i S_{j-i} \\ \tau = \deg \Lambda(x) \to \min \\ j = \tau+1 \ldots 2t \end{array} \right\} \Rightarrow \left\{ \begin{array}{c} \tau \\ \Lambda(x) \end{array} \right\}$$

Коды Рида-Соломона и основанные на них методы кодирования, обнаружения и исправления ошибок не получили бы широкого распространение, если бы в свое время ученые Берлекэмп и Месси в 1968-1969 годах не предложили чрезвычайно эффективный алгоритм [3, 8, 9] решения этой задачи, который находит решение не более чем за $2t$ шагов. Именно этот момент можно считать началом массового применения кодов Рида-Соломона для обнаружения и исправления ошибок. До этого в силу достаточной медлительности и сложности в аппаратной реализации существующих на тот момент алгоритмов поиска полинома локаторов, в основном обходились только вычислением синдромов с целью только обнаружения ошибки. В случае обнаружения ошибки делалась попытка повторного запроса кадра, если это было возможно, в противном случае выдавалось сообщение об ошибке.

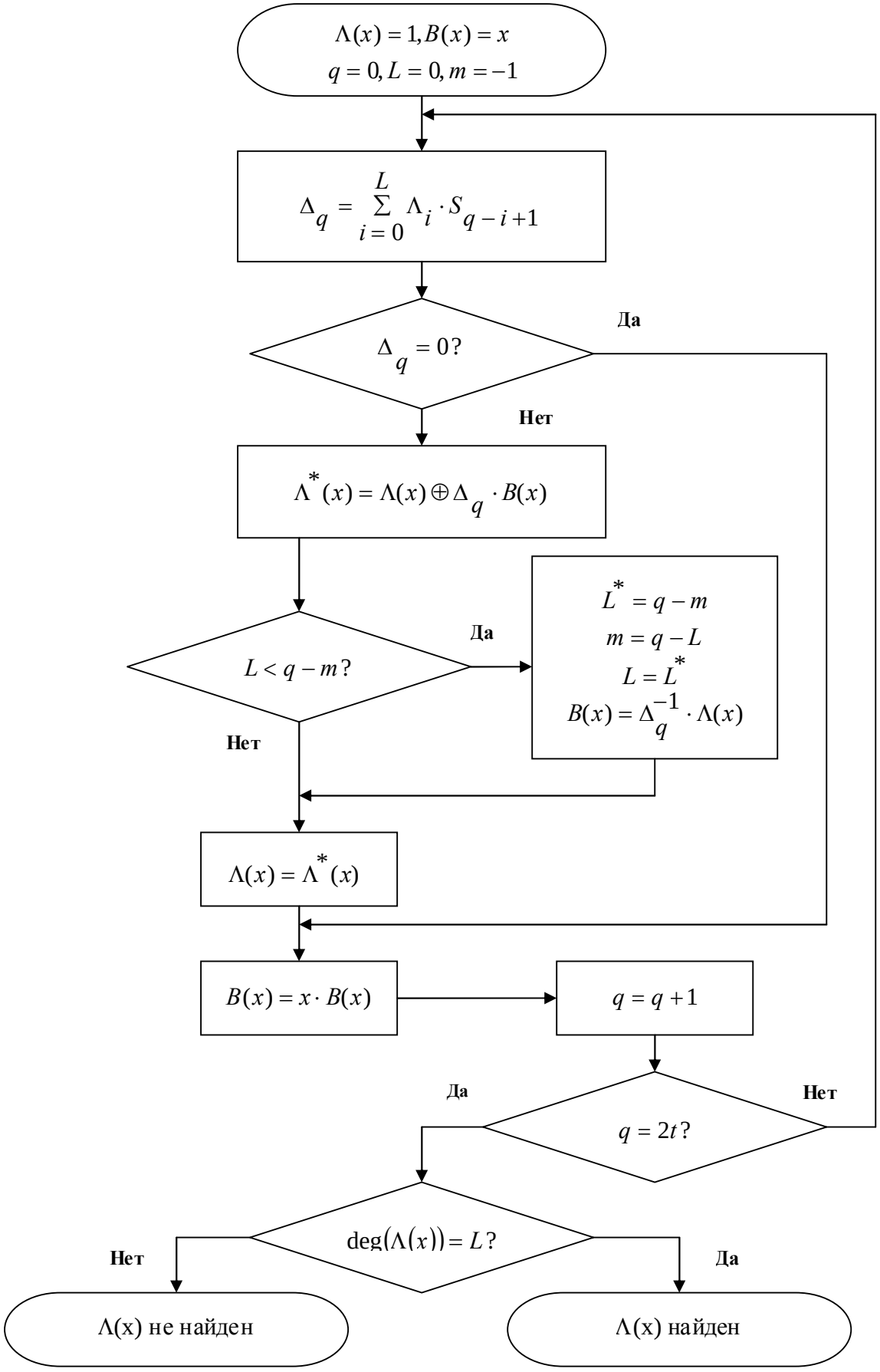


Рис. 2. Схема алгоритма Берлекэмпа-Месси для нахождения полинома локаторов

В алгоритме Берлекэмпа-Месси (рис. 2) используются следующие обозначения:

S_j – коэффициенты синдрома ошибки.

$\Lambda(x)$ – рассчитываемый полином локаторов.

q – номер итерации алгоритма.

Δ_q – значение невязки, это не что иное, как сумма левой и правой части $q-го$ уравнения, которая обращается в нуль, если полином локаторов удовлетворяет уравнению.

m – номер шага, на котором полином локатора последний раз модифицировался.

L – количество членов в левой части уравнений, фактически этот параметр отражает количество τ искаженных байтов, предполагаемое во время итераций алгоритма.

$B(x)$ – полином модификации, который используется как вспомогательный для модификации полинома локаторов для «подстройки» его под уравнения системы.

Алгоритм достаточно нетривиален. Начиная с полинома локаторов $\Lambda(x)=1$, вычисляется, так называемая, невязка (расхождение, несоответствие) этого полинома первому уравнению. Далее учитывая эту невязку, делается коррекция полинома локаторов, при необходимости наращивается его степень, а дальше все повторяется так, чтобы при условии удовлетворения всех предыдущих уравнений, он также стал удовлетворять и текущему уравнению, и так до тех пор, пока не будут удовлетворены все уравнения. Поскольку в алгоритме степень полинома наращивается только при необходимости, то решением и будет полином минимальной степени.

В результате работы алгоритма, в случае, если настоящее количество искаженных байт больше, чем максимальное количество исправляемых ошибок, возможно (но необязательно), что степень полинома локаторов не будет совпадать с предполагаемым количеством искаженных байт, в этом случае считается, что решение системы уравнений не найдено. Этот случай следует трактовать, как невозможность исправления ошибок.

В случае если полином локаторов $\Lambda(x)$ и соответствующая предполагаемая кратность ошибки τ найдены, то дальнейшее декодирование локаторов ошибок не представляет особой трудности. Для этого полином локаторов приравнивается нулю, и находят все корни $2^{-u_1},\ldots,2^{-u_\tau}$ уравнения. Делается это полным перебором элементов $2^1,\ldots,2^{255}$ поля Галуа и подстановкой каждого из них в полином локаторов, если полином локаторов обращается в нуль, то, значит, найден один из корней. Поскольку перебор составляет всего 255 итераций, такой подход вполне оправдан. Наконец, по найденным корням, выделяются их степени $u_1,\ldots,u_\tau$, и, тем самым вычисляются локаторы:

$$\Lambda(x)=0 \Rightarrow x^* = \left\{ 2^{-u_1},\ldots,2^{-u_\tau} \right\} \Rightarrow U = \left\{ u_1,\ldots,u_\tau \right\}$$

После вычисления локаторов, следует выполнить проверку каждого локатора на условие $u_l \in [0, n-1], l = 1\ldots\tau$, иными словами локатор не может быть «указывать за пределы кадра». Если хотя бы один из найденных локаторов «указывает за пределы кадров», то считается, что декодирование прошло некорректно и исправление ошибок невозможно – такое происходит только, если опять же настоящее количество ошибок, в действительности, больше, чем максимальное количество исправляемых ошибок.

Важное примечание. Конкретно для кодов Рида-Соломона в поле Галуа GF(2^8) локаторы ошибок могут принимать значения только в пределах от 0 до 254, поскольку сами являются, по сути, логарифмами от элементов поле Галуа GF(2^8), число которых конечно. Из этого следует очень важное ограничение для применения кодов Рида-Соломона в поле Галуа GF(2^8), длина кадра должна быть меньше либо равна 255, то есть $n = k + r \le 255$. В практике такое серьезное ограничение обходится путем разбиения данных на множество небольших кадров (блоков), длина которых меньше 255 и кодирования их по отдельности.

В случае успешного вычисления локаторов $u_1,\ldots,u_\tau$ ошибок, остается последняя задача: вычисления значений ошибок. Поскольку мы имеет дело не с битами, с байтами в каждой позиции кадра, то значения байта может исказиться 256 различными способами и величину искажения необходимо выяснить для каждого найденного локатора ошибок.

В теории кодирования для вычисления величин ошибок используется, так называемый, метод Форни [4, 8, 9].

Вводится, так называемый, **полином величин ошибок** – $\Omega(x)$, который связан, согласно теории, с полиномом синдрома $S(x)$ и полиномом локаторов $\Lambda(x)$ ошибок следующим уравнением:

$$\Omega(x)=\left(S(x)\cdot\Lambda(x)\right)\bmod\left(x^r\right)$$

Операция вычисления остатка по модулю x^r фактически это не что иное, как простое обнуление всех коэффициентов с индексами $i\geq r$, и использование процедуры деления полиномов здесь совсем излишне. После перемножения полинома синдрома и полинома локатора, коэффициенты с индексами $i\geq 2t$ результирующего полинома-произведения обнуляются, и результат переписывается в коэффициенты полинома $\Omega(x)$.

Вводится также, так называемая, формальная **производная полинома локаторов** – $\Lambda'(x)$, которая вычисляется следующим образом:

$$\Lambda'(x)=\frac{d}{dx}\Lambda(x)=\Lambda_1\oplus\Lambda_3x^2\oplus\ldots\oplus\begin{cases}\Lambda_\tau x^{\tau-1},\tau\bmod 2=1\\ \Lambda_{\tau-1}x^{\tau-2},\tau\bmod 2=0\end{cases}$$

Тогда величины ошибок $v_1,\ldots,v_\tau$ в позициях, указываемых локаторами ошибок, вычисляются, как отношение значения полинома $\Omega(x)$ величин ошибок к значению производной $\Lambda'(x)$полинома локатора в соответствующих корнях 2^{-u_l}, $l=1\ldots\tau$.

$$\begin{cases}v_l=\Omega\left(2^{-u_l}\right)\Big/\Lambda'\left(2^{-u_l}\right)\\ \qquad l=1\ldots\tau\\ V=\{v_1,\ldots,v_\tau\},v_l\in[1,255]\end{cases}$$

Таким образом, вычисляются величины $v_1,\ldots,v_\tau$ ошибок. После этого нетрудно сформировать полином искажений по известным локаторам и величинам ошибок.

$$E(x)=\sum_{l=1}^{\tau}v_l\cdot x^{u_l}$$

Наконец, последнее что остается – это сложить полином кадра (принятого из сети или считанного с носителя) с полиномом искажений и тем самым осуществить исправление ошибок и восстановление исходного полинома неискаженного кадра:

$$F(x)=C(x)\oplus E(x)$$

Ниже представлена схема алгоритма декодирования (рис. 3).

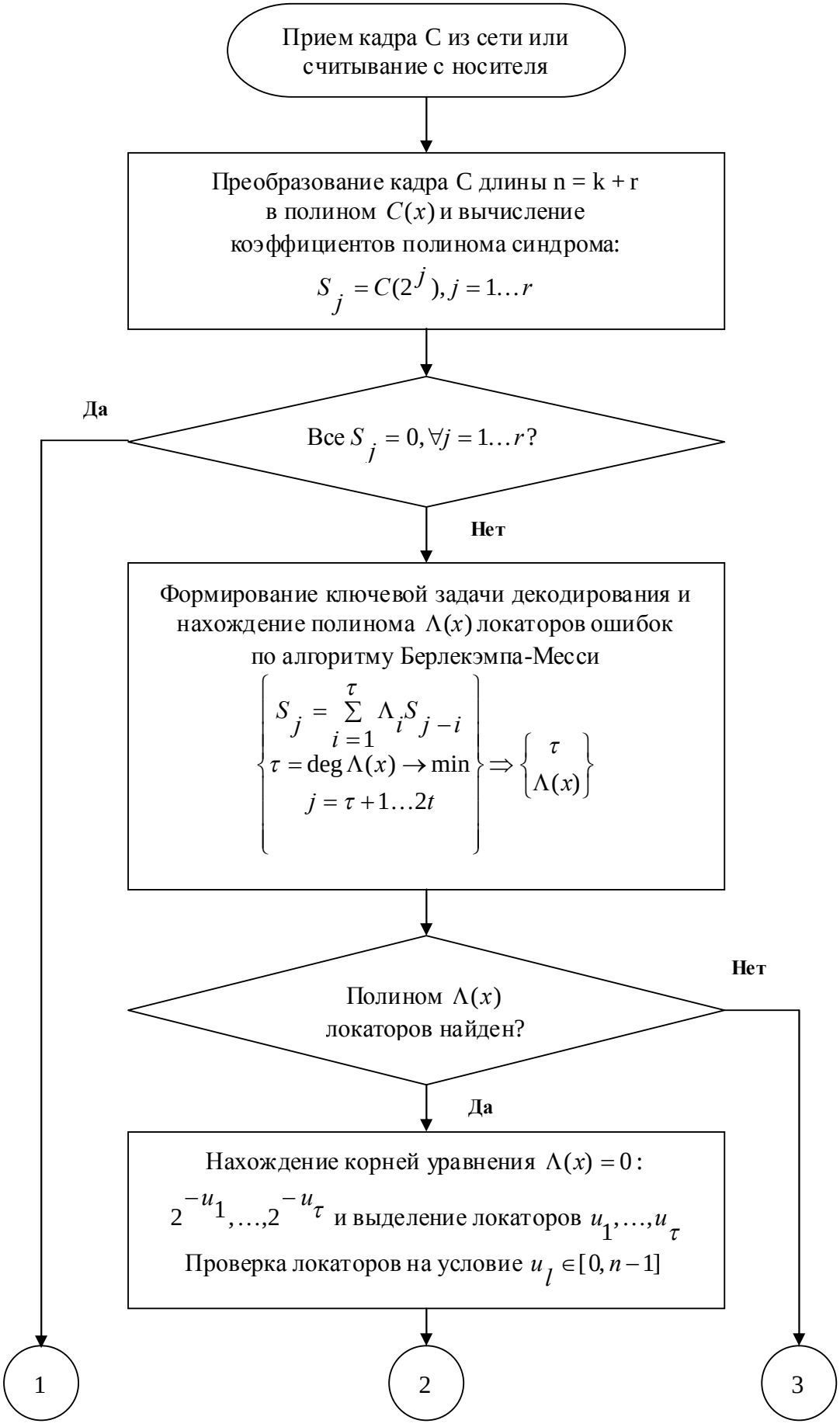


Рис. 3. Схема алгоритма декодирования кода Рида-Соломона (начало)

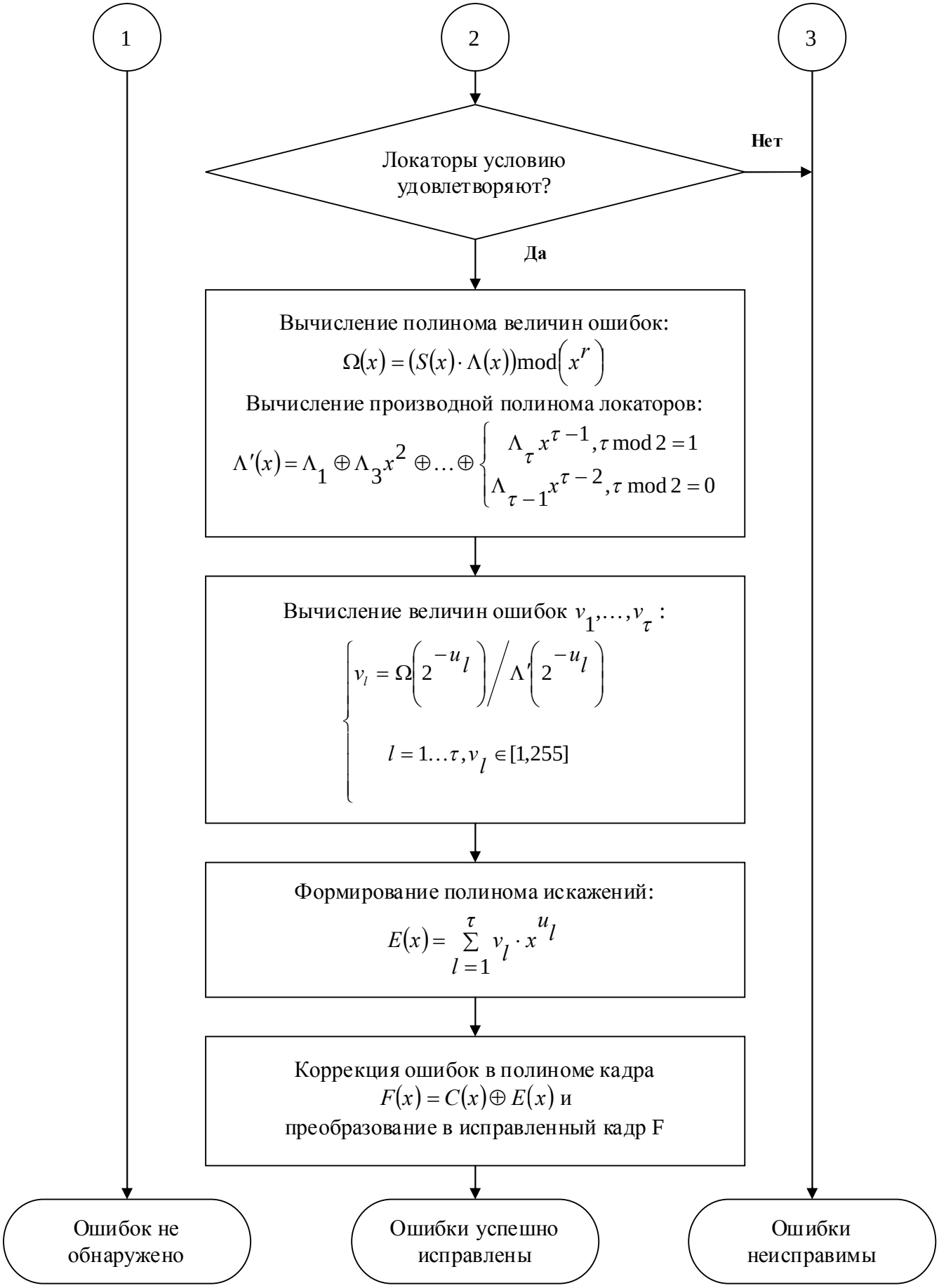


Рис. 3. Схема алгоритма декодирования кода Рида-Соломона (окончание)

Пример кодирования и декодирования

Пусть имеется исходное информационное сообщение **Хакер**, состоящее из $k = 5$ символов. Согласно 8-битной ASCII кодировки, каждому символу можно сопоставить соответствующий код в диапазоне от 0 до 255. В нашем случае исходное сообщение M будет в соответствующих символам кодах выглядеть так:

213	224	234	229	240

Соответственно, полином информационного сообщения будет выглядеть как:

$$M(x) = 213x^4 \oplus 224x^3 \oplus 234x^2 \oplus 229x \oplus 240$$

Допустим, мы хотим использовать коды Рида-Соломона для исправления до $t = 2$ искаженных байтов. В таком случае нам понадобится $r = 2t = 4$ избыточных байтов. Кроме того, порождающий полином будет следующим:

$$g(x) = \prod_{i=1}^{4} \left(x \oplus 2^i \right) = x^4 \oplus 30x^3 \oplus 216x^2 \oplus 231x \oplus 116$$

Тогда при кодировании мы должны вычислить остаток от деления полинома $M(x) \cdot x^4 = 213x^8 \oplus 224x^7 \oplus 234x^6 \oplus 229x^5 \oplus 240x^4 \oplus 0x^3 \oplus 0x^2 \oplus 0x \oplus 0$ на полином $g(x)$. В результате деления получим остаток: $R(x) = 5x^3 \oplus 61x^2 \oplus 81x \oplus 228$.

После этого, сложив полученный остаток с $M(x) \cdot x^4$, получаем полином кадра:

$$F(x) = 213x^8 \oplus 224x^7 \oplus 234x^6 \oplus 229x^5 \oplus 240x^4 \oplus 5x^3 \oplus 61x^2 \oplus 81x \oplus 228$$

И, наконец, тогда результирующий кадр F длины $n = k + r = 9$ будет следующим:

213	224	234	229	240	5	61	81	228

Теперь, допустим, при передаче кадра по каналу связи или при хранении на носителе в кадре исказились два байта и в результате (при приеме кадра из канала или считывании с носителя), мы получили следующий искаженный кадр C:

203	224	236	229	240	5	61	81	228

Если преобразуем информационные байты по таблице ASCII кодировки, то увидим, что получается слово **Ламер** – это, в общем-то, слово с противоположным значением, и тот, кто получит это сообщение, будет испытывать, совсем иные чувства, чем мы ожидали.

Что же, попробуем декодировать полученный кадр. Итак, в полиномиальном виде полученный кадр выглядит следующим образом:

$$C(x) = 203x^8 \oplus 224x^7 \oplus 236x^6 \oplus 229x^5 \oplus 240x^4 \oplus 5x^3 \oplus 61x^2 \oplus 81x \oplus 228$$

Вычислим коэффициенты синдрома:

$$\begin{cases} S_1 = C(2^1) = 203 \cdot 2^8 \oplus 224 \cdot 2^7 \oplus 236 \cdot 2^6 \oplus 229 \cdot 2^5 \oplus 240 \cdot 2^4 \oplus 5 \cdot 2^3 \oplus 61 \cdot 2^2 \oplus 81 \cdot 2^1 \oplus 228 = 246 \\ S_2 = C(2^2) = 203 \cdot 2^{16} \oplus 224 \cdot 2^{14} \oplus 236 \cdot 2^{12} \oplus 229 \cdot 2^{10} \oplus 240 \cdot 2^8 \oplus 5 \cdot 2^6 \oplus 61 \cdot 2^4 \oplus 81 \cdot 2^2 \oplus 228 = 207 \\ S_3 = C(2^3) = 203 \cdot 2^{24} \oplus 224 \cdot 2^{21} \oplus 236 \cdot 2^{18} \oplus 229 \cdot 2^{15} \oplus 240 \cdot 2^{12} \oplus 5 \cdot 2^9 \oplus 61 \cdot 2^6 \oplus 81 \cdot 2^3 \oplus 228 = 255 \\ S_4 = C(2^4) = 203 \cdot 2^{32} \oplus 224 \cdot 2^{28} \oplus 236 \cdot 2^{24} \oplus 229 \cdot 2^{20} \oplus 240 \cdot 2^{16} \oplus 5 \cdot 2^{12} \oplus 61 \cdot 2^8 \oplus 81 \cdot 2^4 \oplus 228 = 213 \end{cases}$$

Таким образом, $S_1 = 246, S_2 = 207, S_3 = 255, S_4 = 213$, коэффициенты синдрома ненулевые, и в том, что имело место быть искажение, сомневаться не приходится. Однако, сколько в действительности байтов искажено, на этапе декодирования никогда неизвестно.

Попытаемся найти полином локаторов $\Lambda(x)$ по алгоритму Берлекэмпа-Месси, и тем самым, также узнать предполагаемое количество τ искаженных байтов.

- Инициализация алгоритма: $\Lambda(x)=1$, $q=0$, $m=-1$, $L=0$, $B(x)=x$
- Итерация $q=0$: вычисляем невязку $\Delta_0 = \sum_{i=0}^{0} \Lambda_i \cdot S_{0-i+1} = \Lambda_0 S_1 = 1\cdot 246 = 246$. Невязка ненулевая, поэтому вычисляем $\Lambda^*(x) = \Lambda(x) \oplus \Delta_0 \cdot B(x) = 246\cdot x \oplus 1$. Далее, так как условие $L < q-m \Rightarrow 0 < 0-(-1) \Rightarrow 0 < 1$ выполняется, то проводятся следующие действия: $L^* = q-m = 0-(-1) = 1$, $m = q-L = 0-0 = 0$, $L = L^* = 1$, $B(x) = \Delta_0^{-1} \cdot \Lambda(x) = 246^{-1} \cdot 1 = 211$. После этого выполняется $\Lambda(x) = \Lambda^*(x) = 246\cdot x \oplus 1$ и $B(x) = x \cdot B(x) = 211\cdot x \oplus 0$. После этого, поскольку $q = q+1 = 1 < 2t = 4$, то переходим к следующей итерации.
- Итерация $q=1$: вычисляем невязку $\Delta_1 = \sum_{i=0}^{1} \Lambda_i \cdot S_{1-i+1} = \Lambda_1 S_1 \oplus \Lambda_0 S_2 = 246\cdot 246 \oplus 1\cdot 207 = 108$. Невязка ненулевая, поэтому вычисляем $\Lambda^*(x) = \Lambda(x) \oplus \Delta_1 \cdot B(x) = (246\cdot x \oplus 1) \oplus 108\cdot(211\cdot x \oplus 0) = 202\cdot x \oplus 1$. Далее, так как условие $L < q-m \Rightarrow 1 < 1-0 \Rightarrow 1 < 1$ не выполняется, то переходим к выполнению действий $\Lambda(x) = \Lambda^*(x) = 202\cdot x \oplus 1$ и $B(x) = x \cdot B(x) = 211\cdot x^2 \oplus 0\cdot x \oplus 0$. После этого, поскольку $q = q+1 = 2 < 2t = 4$, то переходим к следующей итерации.
- Итерация $q=2$: вычисляем невязку $\Delta_2 = \sum_{i=0}^{2} \Lambda_i \cdot S_{2-i+1} = \Lambda_2 S_1 \oplus \Lambda_1 S_2 \oplus \Lambda_0 S_3 = 0\cdot 246 \oplus 202\cdot 207 \oplus 1\cdot 255 = 160$. Невязка ненулевая, поэтому вычисляем $\Lambda^*(x) = \Lambda(x) \oplus \Delta_2 \cdot B(x) = (202\cdot x \oplus 1) \oplus 160\cdot(211x^2 \oplus 0\cdot x \oplus 0) = 158x^2 \oplus 202\cdot x \oplus 1$. Далее, так как условие $L < q-m \Rightarrow 1 < 2-0 \Rightarrow 1 < 2$ выполняется, то проводятся следующие действия: $L^* = q-m = 2-0 = 2$, $m = q-L = 2-1 = 1$, $L = L^* = 2$, $B(x) = \Delta_2^{-1} \cdot \Lambda(x) = 160^{-1} \cdot (202\cdot x \oplus 1) = 45\cdot x \oplus 28$. После этого выполняется $\Lambda(x) = \Lambda^*(x) = 158\cdot x^2 \oplus 202\cdot x \oplus 1$ и $B(x) = x \cdot B(x) = 45\cdot x^2 \oplus 28\cdot x \oplus 0$. После этого, поскольку $q = q+1 = 3 < 2t = 4$, то переходим к следующей итерации.
- Итерация $q=3$: невязка $\Delta_3 = \sum_{i=0}^{3} \Lambda_i \cdot S_{3-i+1} = \Lambda_3 S_1 \oplus \Lambda_2 S_2 \oplus \Lambda_1 S_3 \oplus \Lambda_0 S_4 = 0\cdot 246 \oplus 158\cdot 207 \oplus 202\cdot 255 \oplus 213 = 75$. Невязка ненулевая, поэтому вычисляем $\Lambda^*(x) = \Lambda(x) \oplus \Delta_3 \cdot B(x) = (158\cdot x^2 \oplus 202\cdot x \oplus 1) \oplus 75\cdot(45\cdot x^2 \oplus 28\cdot x \oplus 0) = 19\cdot x^2 \oplus 93\cdot x \oplus 1$. Далее, так как условие $L < q-m \Rightarrow 2 < 3-1 \Rightarrow 2 < 2$ не выполняется, то переходим к выполнению действий $\Lambda(x) = \Lambda^*(x) = 19\cdot x^2 \oplus 93\cdot x \oplus 1$ и $B(x) = x \cdot B(x) = 211\cdot x^2 \oplus 0\cdot x \oplus 0$. После этого, поскольку $(q = q+1 = 4) = (2t = 4)$, то переходим к следующей итерации, то выходим из цикла итераций.

Поскольку степень полинома локаторов $\deg(\Lambda(x) = 19\cdot x^2 \oplus 93\cdot x \oplus 1) = 2$, также как и $L = 2$, то считается, что полинома локаторов успешно найденным.

Таким образом, мы успешно нашли полинома локаторов $\Lambda(x) = 19 \cdot x^2 \oplus 93 \cdot x \oplus 1$ минимальной степени, и что более важно, его степень, равная 2, и есть предполагаемое количество искаженных байт, то есть, иными словами, $\tau = 2$.

Теперь, путем полного перебора значений $2^1,\ldots,2^{255}$ и, подставляя каждое из них в полином локаторов, находим корни, в которых полином локаторов обращается в нуль. Этими корнями, являются элементы $2^{-u_1} = 2^{247}$ и $2^{-u_2} = 2^{249}$, тогда учитывая, что $2^{-u} = 2^{255-u}$ при $0 < u < 255$, получаем, наконец, искомые локаторы ошибок:

$$u_1 = 8 \text{ и } u_2 = 6$$

Примечание. Мы легко можем проверить корни, подставив их в полином локаторов: $\Lambda(x) = 19 \cdot (2^{247})^2 \oplus 93 \cdot (2^{247}) \oplus 1 = 0$ и $\Lambda(x) = 19 \cdot (2^{249})^2 \oplus 93 \cdot (2^{249}) \oplus 1 = 0$

Теперь вычислим полином величин ошибок: $\Omega(x) = (S(x) \cdot \Lambda(x)) \bmod (x^r) =$

$$\left((213 \cdot x^3 \oplus 255 \cdot x^2 \oplus 207 \cdot x \oplus 246) \cdot (19 \cdot x^2 \oplus 93 \cdot x \oplus 1)\right) \bmod (x^4) =$$

$$\left(179 \cdot x^5 \oplus 165 \cdot x^4 \oplus 0 \cdot x^3 \oplus 0 \cdot x^2 \oplus 181 \cdot x \oplus 246\right) \bmod (x^4) = 181 \cdot x \oplus 246$$

Также вычислим формальную производную полинома локатора:

$$\Lambda'(x) = \Lambda_1 \oplus \Lambda_3 x^2 \oplus \ldots \oplus \begin{cases} \Lambda_\tau x^{\tau-1}, \tau \bmod 2 = 1 \\ \Lambda_{\tau-1} x^{\tau-2}, \tau \bmod 2 = 0 \end{cases} = \Lambda_1 = 93$$

Теперь уже можно вычислить сами величины ошибок:

$$v_1 = \Omega\left(2^{-u_1}\right) \Big/ \Lambda'\left(2^{-u_1}\right) = (181 \cdot 2^{247} \oplus 246) / 93 = 30$$

$$v_2 = \Omega\left(2^{-u_2}\right) \Big/ \Lambda'\left(2^{-u_2}\right) = (181 \cdot 2^{249} \oplus 246) / 93 = 6$$

Тогда, окончательно, получаем предполагаемый полином искажений:

$$E(x) = \sum_{l=1}^{2} v_l \cdot x^{u_l} = v_1 \cdot x^{u_1} \oplus v_2 \cdot x^{u_2} = 30 \cdot x^8 \oplus 6 \cdot x^6$$

Остается только сложить полином искаженного кадра с полиномом искажений:

$$\begin{array}{r} \oplus\ C(x) = 203x^8 \oplus 224x^7 \oplus 236x^6 \oplus 229x^5 \oplus 240x^4 \oplus 5x^3 \oplus 61x^2 \oplus 81x \oplus 228 \\ E(x) = 30 \cdot x^8 \oplus 0 \cdot x^7 \oplus 6 \cdot x^6 \oplus 0 \cdot x^5 \oplus 0 \cdot x^4 \oplus 0 \cdot x^3 \oplus 0 \cdot x^2 \oplus 0 \cdot x \oplus 0 \\ \hline F(x) = 213x^8 \oplus 224x^7 \oplus 234x^6 \oplus 229x^5 \oplus 240x^4 \oplus 5x^3 \oplus 61x^2 \oplus 81x \oplus 228 \end{array}$$

Наконец, после преобразования полинома в кадр имеем:

213	224	234	229	240	5	61	81	228

Что же, как видим исправленный кадр, полностью совпадает с кадром, который был передан по сети (записан на носитель). Таким образом, исправление прошло успешно.

Упрощенный способ декодирования при исправлении только одиночных ошибок

Как мы видим, процедура декодирования в общем случае выполняется достаточно сложным и нетривиальным способом. Аппаратная реализация алгоритмов кодирования и декодирования для общего случая требует, как минимум, микроЭВМ с прошитыми в ПЗУ всеми необходимыми алгоритмами, а также с прямым и обратным полями Галуа.

Однако, при реализации алгоритма декодирования частного случая $t=1$, когда закладывается возможность исправления только лишь одиночных ошибок, алгоритм поиска и исправления ошибки можно существенно упростить. Следует заметить, что алгоритм кодирования, и процедура вычисления синдрома ошибки на первом шаге декодирования не упрощаются, и остаются такими же, как и для общего случая.

При $t=1$, избыточность составляет всего $r=2$ байта. Порождающий полином при этом: $g(x)=\prod_{i=1}^{2}\left(x\oplus 2^i\right)=x^2\oplus 6x\oplus 8$. Кодирование некоторого сообщения M длиною k байтов, дает кадр длиною $n=k+2$ байтов:

M_{k-1}	…	M_0	R_1	R_0

Допустим, в результате искажения мы имеем некоторый принятый из сети (считанный с носителя) кадр C длиною $n=k+2$ байтов.

C_{n-1}	…	C_0

Поскольку $r=2$, то, подставляя в соответствующий ему полином $C(x)$ значения 2^1 и 2^2, мы получаем два коэффициента синдрома $S_1=C(2^1)$ и $S_2=C(2^2)$. Если оба коэффициента нулевые $S_1=S_2=0$, то считается, что ошибок нет.

Предположим теперь, что имело место быть искажение, причем количество ошибок неизвестно. Ключевая задача декодирования в общем случае выглядит:

$$\left\{\begin{array}{c} S_j=\sum_{i=1}^{\tau}\Lambda_i S_{j-i} \\ \tau=\deg\Lambda(x)\to\min \\ j=\tau+1\ldots 2t \end{array}\right\}\Rightarrow\left\{\begin{array}{c}\tau\\ \Lambda(x)\end{array}\right\}$$

Однако, с одной стороны $t=1$, с другой стороны минимальная кратность ошибки $\tau=1$, и тогда индекс заключен в пределы $j=2\ldots 2$. Тогда система упрощается до одного уравнения при индексе $j=2$: $S_2=\sum_{i=1}^{\tau}\Lambda_i S_{2-i}$. Кроме того, у нас всего два коэффициента синдрома S_1 и S_2, и при $j=2$, имеем допустимое слагаемое $\Lambda_1 S_1$, только при $i=1$. В итоге, ключевая система уравнений упрощается до вида $S_2=\Lambda_1 S_1$, откуда легко получаем, коэффициент полинома локатора $\Lambda_1=S_2/S_1$, и тогда поскольку в полиноме локатора младший коэффициент $\Lambda_0=1$ по определению, то в итоге имеем полином локаторов:

$$\Lambda(x)=\left(S_2/S_1\right)\cdot x\oplus 1$$

Таким образом, $t=1$, мы решаем ключевую задачу декодирования, не применяя сложного в реализации алгоритма Берлекэмпа-Месси.

Примечание. Важно отметить, что в таком случае мы автоматически предполагаем, что произошла только одиночная ошибка $\tau=1$, но, имея всего два коэффициента синдрома ничего большего и нельзя сделать. В случае, если действительное количество искажений будет больше одной $\tau>1$, декодирование, разумеется, будет работать некорректно.

Нетрудно видеть, что полином локаторов имеет единственный корень $2^{-u}=S_1/S_2$, откуда легко получаем сам локатор ошибки:

$$2^{-u}=S_1/S_2 \Rightarrow 2^{u}=S_2/S_1 \Rightarrow$$
$$u=\log_2\left(S_2/S_1\right)=\log_2 S_2-\log_2 S_1$$

Тогда, учитывая, что вычисление логарифма по основанию примитивного элемента 2 в поле Галуа $GF(2^8)$, можно выполнять, просто считывая соответствующий элемент из обратного поля Галуа $GF^{-1}(2^8)$. Таким образом, вычисление локатора ошибки сводится к простому извлечению элементов $GF^{-1}[S_2]$ и $GF^{-1}[S_1]$ обратного поля Галуа и вычислению разности между ними (по стандартным правилам традиционной алгебры). Напомним, что для вычисления разности степеней действует правило, что если разность степеней меньше нуля, то к ней прибавляется 255.

Полином локатора обязательно следует проверить на условие $u\in[0\ldots n-1]$, и если условие не соблюдается, то можно говорить о том, что декодирование прошло некорректно и, очевидно, имело место быть искажение большего числа байтов $\tau>1$.

Вычислим теперь полином величин ошибок: $\Omega(x)=\left(S(x)\cdot\Lambda(x)\right)\bmod(x^r)=$
$\left(\left(S_2\cdot x\oplus S_1\right)\cdot\left(\left(S_2/S_1\right)\cdot x\oplus 1\right)\right)\bmod(x^2)=\left(\left(S_2^2/S_1\right)\cdot x^2\oplus\left(S_2\oplus S_1\cdot\left(S_2/S_1\right)\right)\cdot x\oplus S_1\right)\bmod(x^2)=$
$\left(\left(S_2^2/S_1\right)\cdot x^2\oplus\left(S_2\oplus S_2\right)\cdot x\oplus S_1\right)\bmod(x^2)=\left(\left(S_2^2/S_1\right)\cdot x^2\oplus S_1\right)\bmod(x^2)=S_1$.

Вычислим теперь производную полинома локаторов, учитывая что $\tau=1$:

$$\Lambda'(x)=\Lambda_1\oplus\Lambda_3x^2\oplus\ldots\oplus\begin{cases}\Lambda_\tau x^{\tau-1},\tau\bmod 2=1\\ \Lambda_{\tau-1}x^{\tau-2},\tau\bmod 2=0\end{cases}=\Lambda_1=S_2/S_1$$

Наконец, можно вычислить величину ошибку:

$$v=\Omega\left(2^{-u}\right)/\Lambda'\left(2^{-u}\right)=S_1/\left(S_2/S_1\right)=S_1^2/S_2$$

Выполним следующие преобразования полученной формулы:

$$v=S_1^2/S_2=2^{\log_2\left(S_1^2/S_2\right)}=2^{\left(2\cdot\log_2 S_1-\log_2 S_2\right)}$$

Таким образом, вычисление величины ошибки сводится к простому извлечению элементов $GF^{-1}[S_1]$ и $GF^{-1}[S_2]$, и вычислению разности $2\cdot GF^{-1}[S_1]-GF^{-1}[S_2]$. Если при вычислении $2\cdot GF^{-1}[S_1]$ результат больше 255, то от результата вычисляется остаток по модулю 255. Кроме того, если разность меньше нуля, то к ней добавляется 255. После этого необходимо возвести примитивный элемент 2 в полученную разность, в поле Галуа $GF(2^8)$ – это делается извлечением соответствующего элемента из основного поля $GF(2^8)$.

5. Вероятностный анализ искажений. Выбор кратности исправляемых ошибок.

Для каналов передачи данных, как правило, задается оценка вероятности искажения бита, поскольку данные передаются в виде последовательного потока битов, и каждый из них может подвергнуться искажению. С точки зрения искажения все биты «равноправны», и не имеет значения то, к какому байту, слову и т.п. принадлежит бит. В случае искажения нескольких битов, последние могут различными способами располагаться внутри кадра, состоящего из n байтов, например, 8 искаженных битов могут «уместиться» внутри одного байта, а могут «распылиться» по 8 различным байтам – и это будут принципиально различные ситуации с точки зрения корректирующей способности кодов Рида-Соломона.

Чтобы оценивать вероятности искажения одного, нескольких или всех байтов в кадре на основе информации о базовой вероятности искажения одного бита мы должны обратиться к математическому аппарату теории вероятностей [10].

Пусть, p – базовая вероятность искажения одиночного бита (в канале передачи данных или в носителе информации).

Согласно теории вероятностей вероятность того, что исказится байт, равна величине, дополнительной (до единицы) к вероятности того, что ни один бит в байте не исказится:

$$P_{Byte} = 1 - (1-p)^8$$

Тогда, согласно биномиальному распределению Бернулли, получаем вероятность того, что исказится ровно θ байтов в кадре, состоящего из n байтов, при заданной вероятности p искажения одного бита:

$$P(\tau = \theta) = C_n^{\theta} \cdot \left(1 - (1-p)^8\right)^{\theta} \cdot \left((1-p)^8\right)^{(n-\theta)}$$

Искаженные биты могут «попадать» в какие-то байты, а в какие-то «не попадать». Формула определяет вероятность искажения θ байтов, при условии целостности остальных $n-\theta$ байтов, во всех подходящих вариантах искажения $\lambda = \theta \ldots 8\theta$ битов, при условии целостности остальных $8n-\lambda$ битов в кадре и условии, что в каждый из θ байтов «попадет» хотя бы один искаженный бит в каждом варианте, и также учитываются все сочетания искаженных θ байтов по n байтам в кадре.

Наконец, вероятность того, что исказится не более t байтов в кадре длиною n байтов, при заданной вероятности p искажения одного бита:

$$P(\tau \le t) = \sum_{\theta=0}^{t} C_n^{\theta} \cdot \left(1 - (1-p)^8\right)^{\theta} \cdot \left((1-p)^8\right)^{(n-\theta)}$$

Следует особо отметить, что может возникнуть ложная иллюзия, что вероятность ошибок большей кратности (искажения большего числа байтов), обязательно меньше вероятности ошибок меньшей кратности (искажения меньшего числа байтов). Например, иллюзия о том, что искажение двух байтов куда менее вероятно, чем искажение одного байта. В действительности же, все зависит от базовой вероятности искажения одного бита и количества байтов в кадре, и при большой базовой вероятности искажения бита и большой длине кадра, ошибка большей кратности может быть куда более вероятна, чем ошибка меньшей кратности. Это несложно пояснить: при большой вероятности искажения бита и большой длине кадра, среднее количество искаженных битов может быть настолько большим, что они просто «не умещаются», ни в одном, ни в двух, ни даже в большем числе байтов, и искажают большое количество байтов в кадре и именно это наиболее вероятно. Этот «эффект» наглядно виден на графиках (рис. 4) зависимостей вероятности искажения ровно θ байтов при различных значениях базовой вероятности p искажения бита.

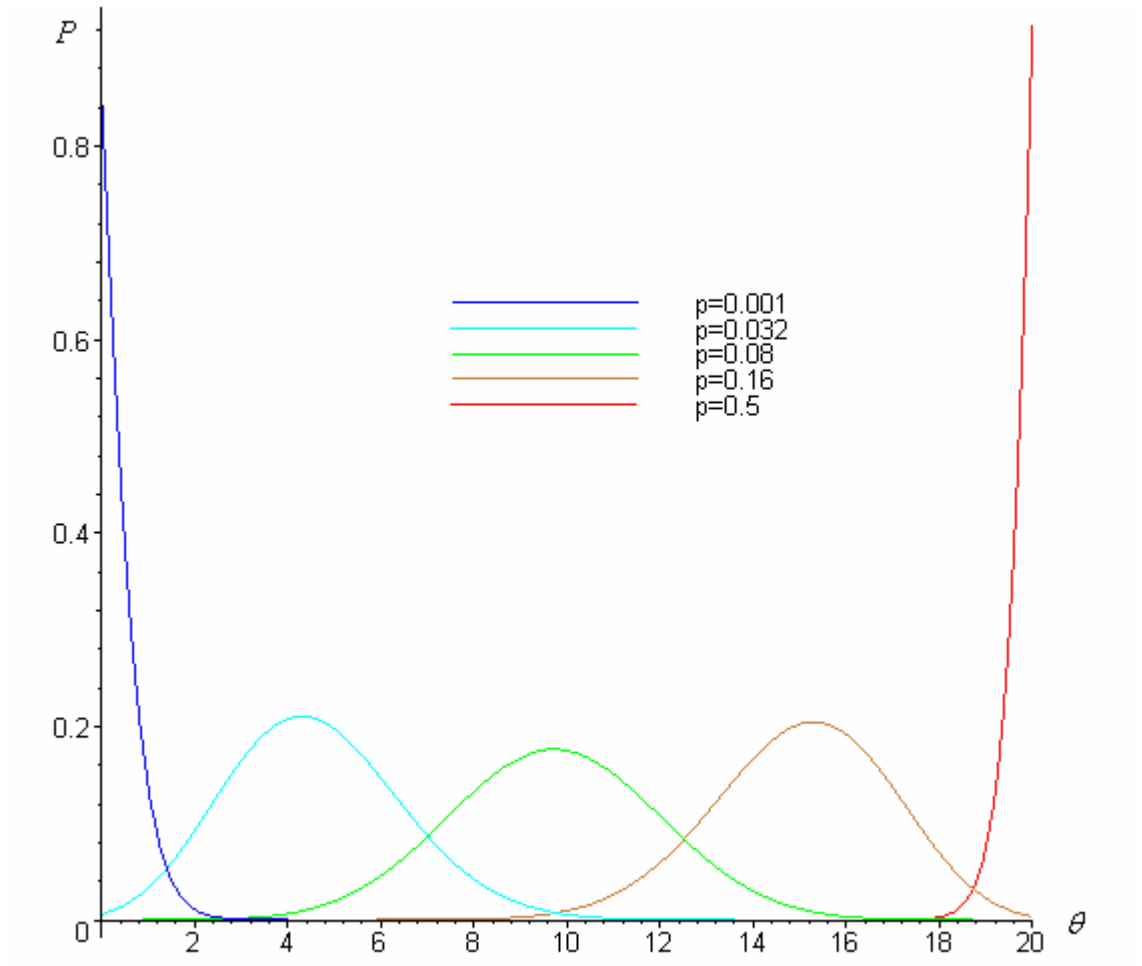


Рис. 4. Графики зависимостей вероятности искажения ровно θ байтов в кадре длиною n = 20 байтов при различной базовой вероятности p искажения бита.

Как видно из графиков, только при достаточно малой базовой вероятности искажения бита при заданной длине кадра можно говорить, что ошибки большей кратности менее вероятны, чем ошибки меньшей кратности. С ростом базовой вероятности искажения бита, ситуация меняется, и наблюдаются случаи, когда наиболее вероятны ошибки строго конкретной кратности, причем далеко не одиночной, и так далее, вплоть до крайнего случая, когда наиболее вероятно искажение всех байтов в кадре.

Оценим теперь вероятность искажения кадра длиною n байтов при условии применения кодов Рида-Соломона гарантированно исправляющих t искаженных байтов с помощью $r = 2t$ избыточных байтов. На долю полезной информации, очевидно, в этом случае остается лишь $k = n - r$ байтов. При возникновении ошибок кратности $\tau \le t$ декодер успешно исправит ошибки, и не «справится» он только при ошибках кратности $\tau > t$ (при $\tau \le 2t$ декодер также гарантированно распознает ошибку, но без возможности коррекции).

Тогда вероятность «неисправимого» искажения кадра длиною n байтов при условии применения кодов Рида-Соломона с $r = 2t$ избыточными байтами определяется как:

$$P(\tau > t) = \sum_{\theta = t+1}^{n} C_n^{\theta} \cdot \left(1 - (1-p)^8\right)^{\theta} \cdot \left((1-p)^8\right)^{(n-\theta)}$$

Ниже приведены графики (рис. 5) зависимостей вероятности искажения $\tau > t$ байтов в кадре длиною n байтов при различной базовой вероятности p искажения одиночного бита.

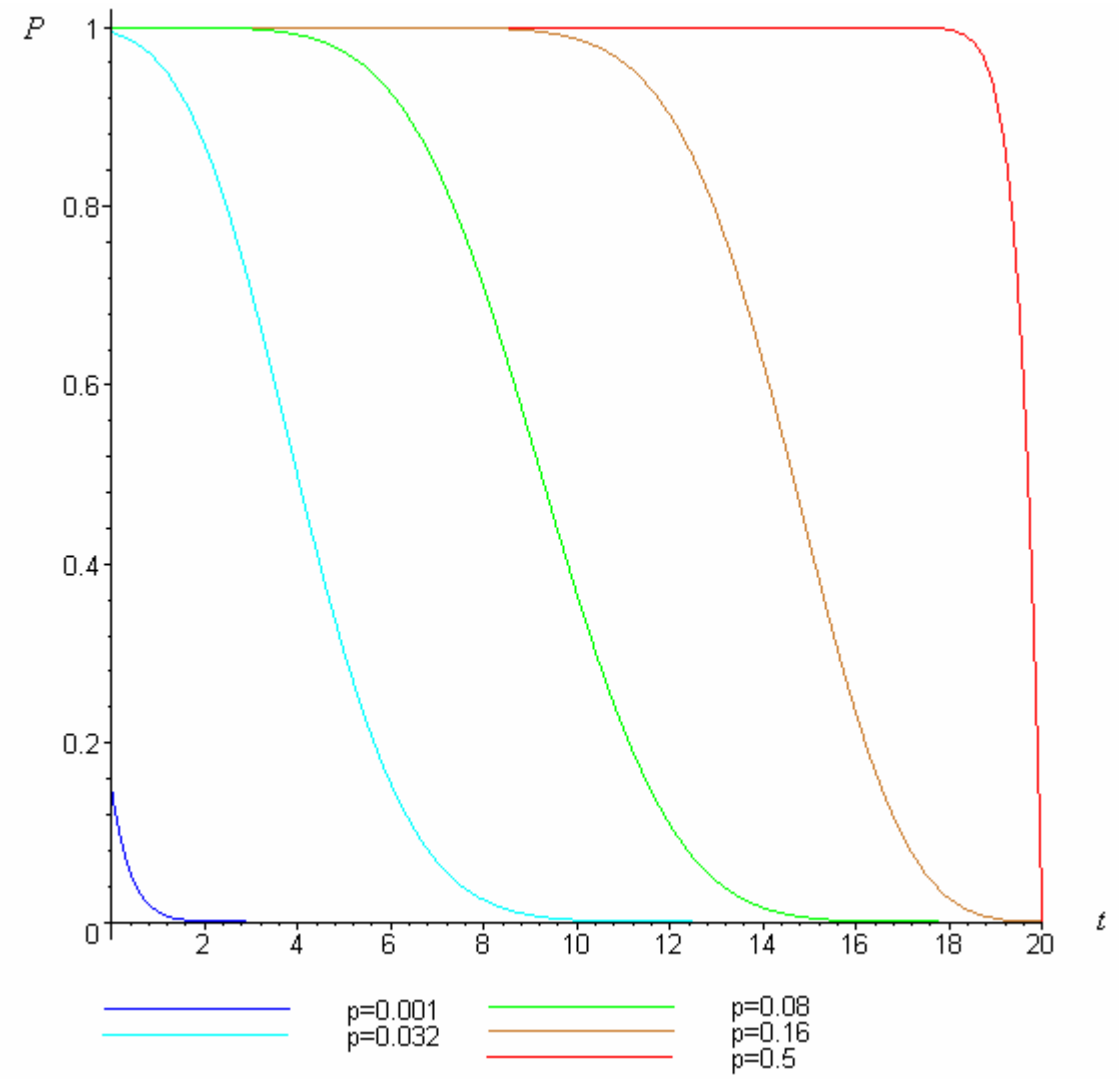


Рис. 5. Графики зависимостей вероятности искажения более t байтов в кадре длиною n = 20 байтов при различной базовой вероятности p искажения бита.

Как видно из графиков, вероятность «неисправимого» искажения кадра длиною n байтов при условии применения кодов Рида-Соломона с $r = 2t$ избыточными байтами сильно зависит опять же от базовой вероятности искажения одиночного бита. Причем, также не будем забывать, что при использовании кодов Рида-Соломона за возможность исправления t искаженных байтов приходится платить $r = 2t$ избыточными байтами, и в условиях, когда у нас задана фиксированная длина n кадра, на долю байтов полезной информации остается все меньшее $k = n - r$ число байтов. В пределе, если n четное число, коды Рида-Соломона могут обеспечить исправление вплоть до $t = n/2$ байтов, при этом все n кадров уйдут под избыточные байты, так как $r = 2t = 2 \cdot (n/2) = n$, при этом не останется ни одного байта для хранения полезной информации, так как $k = n - r = n - n = 0$.

Поэтому при постановке вопроса о целесообразности использования кодов Рида-Соломона и выбора числа избыточных байтов следует иметь в качестве исходных данных, как минимум, следующие данные:

n – фиксированная длина кадра в байтах.

p – базовая вероятность искажения одиночного бита.

$k_{\min}$ – требуемое минимальное число байтов полезной информации в кадре.

$P_{\max}$ – заданная предельно допустимая вероятность искажения кадра.

Далее, исходя из условия что, должно соблюдаться требование по допустимой вероятности искажения кадра и требование на минимальное число байтов полезной информации попытаться выбрать кратность t исправляемой ошибки:

$$\begin{cases} P(\tau > t) < P_{\max} \\ n - 2t \geq k_{\min} \end{cases} \Rightarrow t*$$

Если, невозможно найти параметр t, удовлетворяющий обоим условиям, то, очевидно, применение кодов Рида-Соломона нецелесообразно. Кроме того, если сама по себе вероятность искажений любой кратности меньше заданной допустимой границы, то есть $P(\tau > 0) < P_{\max}$, то применение кодов Рида-Соломона также нецелесообразно. Ниже (рис. 6) приведена схема алгоритма для оценки целесообразности применения кодов Рида-Соломона и выбора кратности исправляемой ошибки.

Пример 1. Задана длина кадра $n = 36$ байтов, минимальное число байтов полезной информации $k_{\min} = 32$, базовая вероятность искажения бита $p = 5 \cdot 10^{-4}$ и предельно допустимая вероятность искажения кадра $P_{\max} = 0{,}001$.

Рассчитаем сначала вероятность искажения кадра в исходных условиях без использования кодов Рида-Соломона, иными словами рассчитаем вероятность возникновения ошибок любой кратности:

$$P(\tau > 0) = 0{,}13414$$

Очевидно, что вероятность значительно выше допустимой границы $P_{\max} = 0{,}001$. Тогда попробуем выбрать кратность t исправляемой ошибки, исходя из условий:

$$\begin{cases} P(\tau > t) < 0{,}001 \\ 36 - 2t \geq 32 \end{cases}$$

Очевидно, для заданного требуемого числа байтов полезной информации, можно рассматривать только два варианта $t = 1$ и $t = 2$. В этих вариантах вероятности «неисправимого» искажения кадра равны: $P(\tau > 1) = 0{,}00918$ и $P(\tau > 2) = 0{,}000412$. Очевидно, что вариант $t = 2$, удовлетворяет обоим условием. Соответственно, применение кодов Рида-Соломона целесообразно, причем число избыточных байтов $r = 2t = 4$.

Пример 2. Задана длина кадра $n = 128$ байтов, минимальное число байтов полезной информации $k_{\min} = 120$, базовая вероятность искажения бита $p = 10^{-3}$ и предельно допустимая вероятность искажения кадра $P_{\max} = 0{,}001$.

Рассчитаем сначала вероятность искажения кадра в исходных условиях без использования кодов Рида-Соломона, иными словами рассчитаем вероятность возникновения ошибок любой кратности:

$$P(\tau > 0) = 0{,}641$$

Очевидно, вероятность гораздо выше заданной допустимой границы. Мы можем для сокращения перебора попробовать оценить вероятность «неисправимой» ошибки для «крайнего варианта» $t = 4$, при котором еще соблюдается условие $n - 2t = 128 - 2 \cdot 4 = 120 \geq k_{\min} = 120$. Имеем, $P(\tau > 4) = 0{,}0038$ и видим, что даже в «крайнем варианте», исправления ошибок вплоть до 4-х кратных, вероятность «неисправимого» искажения превышает заданную допустимую границу. Так, что применение кодов Рида-Соломона в данном случае нецелесообразно. Здесь требуется для начала либо уменьшить длину кадра, либо базовую вероятность искажения бита, либо и то и другое.

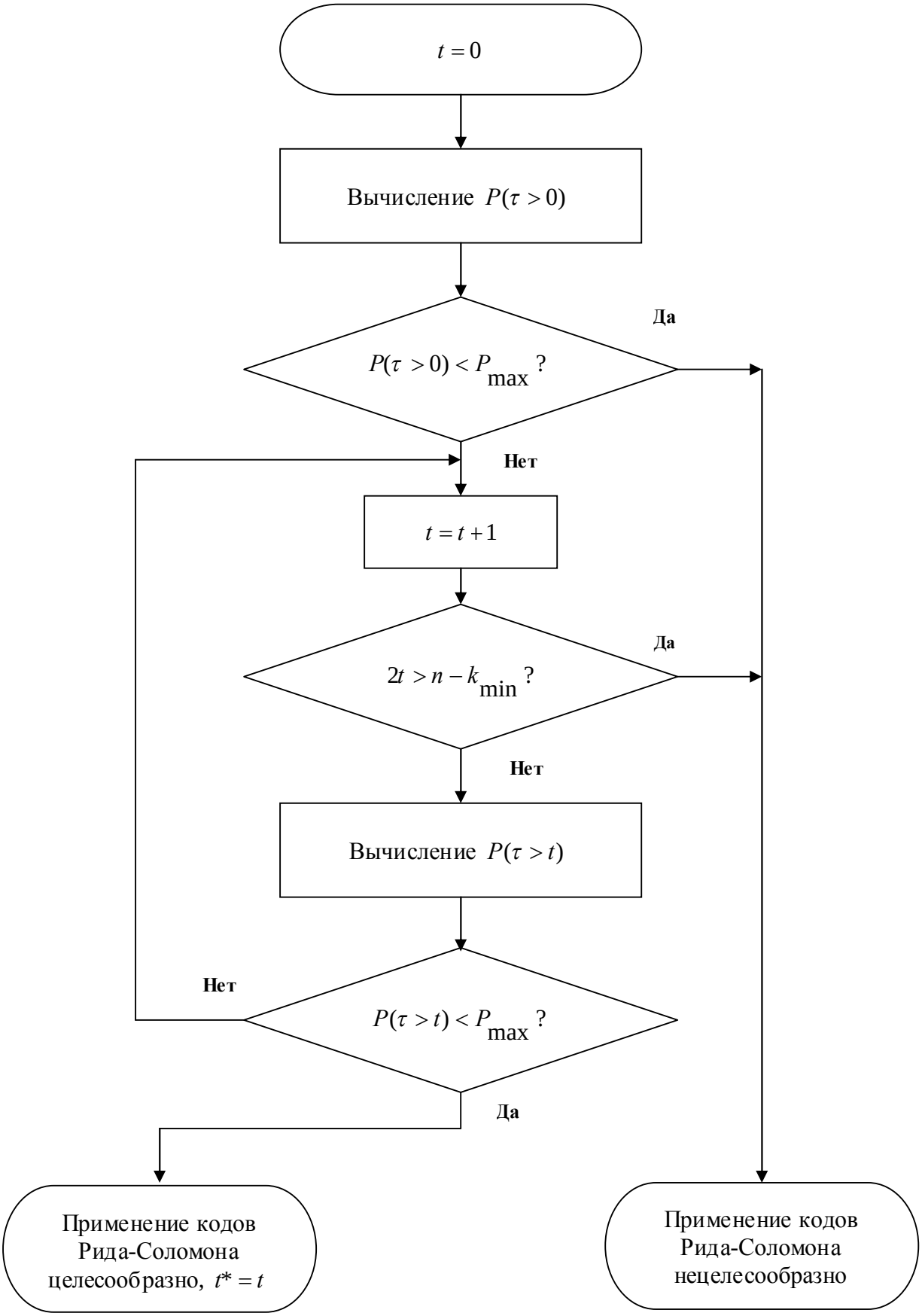


Рис. 6. Схема алгоритма для оценки целесообразности применения кодов Рида-Соломона и выбора кратности исправляемой ошибки.

Заключение

Практическую ценность кодов Рида-Соломона трудно переоценить. С помощью них можно не только обнаруживать, но и частично восстанавливать информацию практически «из пепла». К сожалению, коды Рида-Соломона у большинства специалистов ассоциируются только с помехоустойчивым кодированием в каналах передачи данных. В действительности, их можно применять везде, где необходимо предотвратить модификацию данных:

- Обнаружение и коррекция неумышленных ошибок (помех) при передаче данных по каналам связи, ошибок в данных на носителях информации при их сбое или отказе.
- Обнаружение и коррекция (при поддержке системы открытых и закрытых ключей) умышленной модификации информационных сообщений с целью дезинформации.
- Обнаружение и коррекция умышленной модификации информации об авторе или исполняемого кода с целью «взлома» программного обеспечения.
- Защита программного обеспечения или данных от копирования с лицензионного диска, при использовании специальных «настоящих» и «ложных» ошибок в секторах.
- Восстановление одного или нескольких томов многотомного архива, искаженных или вообще потерянных при загрузке из сети. Аналогично, восстановление данных одного или нескольких дисков в многодисковых системах RAID.
- Обнаружение и исправление ошибок в цепочках ДНК в генной инженерии.

Кроме того, особо следует отметить алгоритм декодирования синдрома ошибки. Алгоритм декодирования синдрома применительно к кодам Рида-Соломона – это один из самых элегантных примеров того, как на базе исходного синдрома, совокупности симптомов (признаков), характеризующих проблему, болезнь и т.п. можно установить «первопричины»: местоположения и характер ошибок, приведших к проблеме. В настоящее время, сплошь и рядом возникают проблемы в самых различных областях и ситуациях жизни, которые проявляются в виде симптомов, по которым редко когда можно с ходу установить «первопричины». Зачастую причины ищутся путем ограниченного перебора вероятных вариантов, которые чаще всего либо не дают каких-либо адекватных результатов, либо приводят к раскрытию только косвенных или вторичных (производных) причин. В лучшем случае «успешному декодированию» поддаются в основном случаи с одиночной причиной.

Литература

1. **Чеботарев Н.Г.** Основы теории Галуа. Часть 1. – М.: Едиториал УРСС, 2004.
2. **Блейхут Р.** Теория и практика кодов, контролирующих ошибки. – М.: Мир, 1986.
3. **Берлекэмп Э.** Алгебраическая теория кодирования. – М. Мир, 1971.
4. **Форни Д.** Каскадные коды. – М.: Мир, 1970.
5. **М. Вернер.** Основы кодирования. – М.: Техносфера, 2004.
6. **Д. Сэломон.** Практическое руководство по методам сжатия данных. – М.: Техносфера, 2004.
7. **Василенко О.Н.** Теоретико-числовые алгоритмы в криптографии. – М.: МЦНМО, 2003.
8. **Касперски К.** Техника защиты компакт-дисков от копирования. – СПб.: БХВ-Петербург, 2004.
9. **Кнут Д.** Искусство программирования. Том 2. – М.: Вильямс, 2005.
10. **Гнеденко Б.В.** Курс теории вероятностей. – М.: Едиториал УРСС, 2005.