Кодирование информации с применением кодов Рида-Соломона.

П.А. Рахман

Введение

Одна из наиболее острых проблем в информационных технологиях – это защита данных от разрушения. Как каналы передачи данных, так и носители информации на сегодняшний день остаются далекими от совершенства, несмотря на все усилия производителей современных аппаратных средств. Кабельные и беспроводные линии передачи информации подвержены воздействию внешних помех, искажающих форму передаваемых сигналов и тем самым делающих невозможным однозначное распознавание информации на стороне приемника, магнитные и оптические носители информации чувствительны к физическим повреждениям, делающим невозможным чтение информации из отдельных участков на поверхности носителя. В такой ситуации особенно актуальным становится применение специальных технологий информационного резервирования, позволяющих, как минимум, обнаруживать искажения, а как максимум, не только обнаруживать, но и исправлять искажения.

В течение большей части второй половины XX века математики и специалисты по аппаратным и программным средствам ЭВМ и проблемам передачи данных упорно бились над тем, чтобы разработать 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 получила такое широкое распространение, что ее возможностями прямо или косвенно пользуется большинство людей мира, причем многие даже не подозревают о ее существовании. Технология применяется при приеме и передаче данных в сетях передачи данных, а также при чтении и записи данных в системах хранения данных, использующих различные виды носители информации.

1. Кратко о конечных полях и общей алгебре.

Основная трудность в понимании технологии кодирования заключается в том, что она базируется не на привычной со школы алгебре бесконечного поля действительных чисел, а на специальной алгебре конечного поля Галуа [1, 2, 3]. Более того, сама по себе технология кодирования базируется на, так называемой, алгебраической теории кодирования [6-14]. Соответственно, чтобы разобраться в технологии, необходимо сначала познакомиться с основами общей алгебры [1, 5] и алгебраическими структурами, рассматриваемыми в ней, в том числе с конечным полем Галуа.

Алгебра. Пусть задано некоторое множество элементов $\boldsymbol{G}$. Пусть задан набор операций $\{\varphi_1,\ldots,\varphi_m\}$ с некоторыми заданными арностями $\{n_1,\ldots,n_m\}$, каждая $i-я: 1 \le i \le m$ из которых любым n_i элементам из множества $\boldsymbol{G}$ ставит в соответствие некоторый элемент из этого же множества. Если $n_i = 1$, то операция φ_i называется унарной (одноместной), если $n_i = 2$, то операция φ_i называется бинарной (двуместной), и так далее. Тогда множество $\boldsymbol{G}$, с заданным набором операций $\{\varphi_1,\ldots,\varphi_m\}$ называется **алгебраической структурой** или просто **алгеброй**. При этом множество элементов $\boldsymbol{G}$ также называют **носителем** или **основой** алгебры, вектор арностей $\{n_1,\ldots,n_m\}$ называют **типом,** а набор операций $\{\varphi_1,\ldots,\varphi_m\}$ называют **сигнатурой**. Носитель $\boldsymbol{G}$ не обязательно конечен, но не пуст: $|G| \neq 0$. Арность любой из операций конечна, набор операций также конечен.

Особо отметим, что в определении алгебраической структуры неявно (выраженное в словесной форме) присутствует важнейшее **свойство замкнутости алгебраической структуры**. Любая операция, принадлежащая набору операций алгебраической структуры, заданной конечной арности, выполняемая над соответствующим числом элементов множества, являющегося основой алгебраической структуры, обязательно дает элемент, который также принадлежит этому множеству. Если это свойство не выполнено, то не может быть речи ни о какой-либо алгебраической структуре (алгебре).

Ярким примером алгебраической структуры, хорошо известным еще со школьной скамьи, является поле действительных чисел $<\boldsymbol{R}, \{+, \cdot\}>$, состоящее из бесконечного множества действительных чисел $\boldsymbol{R}$ и двух бинарных операций: сложения и умножения, ставящих любым двум элементам множества $\boldsymbol{R}$ в соответствие некоторый третий элемент из этого множества. Кроме того, обе операции обладают рядом свойств, которые мы обсудим подробнее ниже при рассмотрении понятия поля.

Самой простой алгебраической структурой является структура, состоящая из множества элементов G и одной единственной унарной операции, ставящей в соответствие любому элементу множества G некоторый другой элемент этого множества. Однако, мы не будем подробно останавливаться на простейших структурах, и перейдем к рассмотрению структур, состоящих из множества элементов, и одной или двух бинарных операций.

Полугруппа. Алгебраическая структура, состоящая из некоторого множества элементов $\boldsymbol{G}$ и одной бинарной операции, обозначим ее значком $\circ$, ставящей в соответствие любым двум элементам $\boldsymbol{a}$ и $\boldsymbol{b}$ множества $\boldsymbol{G}$ некоторый третий элемент $\boldsymbol{c}$ этого множества, называется **полугруппой**, если бинарная операция $\circ$ обладает единственным свойством:

- Свойство ассоциативности: $\forall a,b,c \in G : (a \circ b) \circ c = a \circ (b \circ c)$.

Простейшим примером полугруппы является множество непустых слов, составленных из букв некоторого алфавита, и заданной бинарной операции конкатенации двух слов: выписывания букв первого слова и дописывания к ним букв второго слова.

Моноид. Алгебраическая структура, состоящая из некоторого множества элементов $\boldsymbol{G}$ и одной бинарной операции, обозначим ее значком $\circ$, ставящей в соответствие любым двум элементам $\boldsymbol{a}$ и $\boldsymbol{b}$ множества $\boldsymbol{G}$ некоторый третий элемент $\boldsymbol{c}$ этого множества, называется **моноидом**, если бинарная операция $\circ$ обладает двумя свойствами:

- Свойство ассоциативности: $\forall a,b,c \in G : (a \circ b) \circ c = a \circ (b \circ c)$.
- Существование нейтрального элемента $\boldsymbol{e}$ относительно бинарной операции такого, что операция любого элемента a множества G с нейтральным элементом e снова дает элемент a: $\forall a \in G, \exists e \in G : e \circ a = a \circ e = a$.

Простейшим примером моноида является множество слов, включая пустое слово, составленных из букв некоторого алфавита, и заданной бинарной операции конкатенации двух слов. Заметим, что пустое слово – это и есть нейтральный элемент моноида. Очевидно, что конкатенация любого непустого слова с пустым словом дает исходное непустое слово.

Отметим также, что подмножество всех элементов множества G за исключением нейтрального элемента e, вместе с заданной бинарной операцией, образуют полугруппу. Иными словами, моноид по определению содержит в себе полугруппу.

Группа. Алгебраическая структура, состоящая из некоторого множества элементов $\boldsymbol{G}$ и одной бинарной операции, обозначим ее значком $\circ$, ставящей в соответствие любым двум элементам $\boldsymbol{a}$ и $\boldsymbol{b}$ множества $\boldsymbol{G}$ некоторый третий элемент $\boldsymbol{c}$ этого множества, называется **группой**, если бинарная операция $\circ$ обладает следующими тремя свойствами:

- Свойство ассоциативности: $\forall a,b,c \in G : (a \circ b) \circ c = a \circ (b \circ c)$.
- Существование нейтрального элемента: $\forall a \in G, \exists e \in G : e \circ a = a \circ e = a$.
- Существование для каждого элемента множества G такого обратного элемента относительно бинарной операции, что операция элемента с обратным ему элементом дает нейтральный элемент e: $\forall a \in G, \exists \overline{a} \in G : a \circ \overline{a} = \overline{a} \circ a = e$.

Определение 1. Если для бинарной операции дополнительно выполняется свойство коммутативности: $\forall a,b \in G : a \circ b = b \circ a$, то группа называется коммутативной (абелевой) по заданной бинарной операции, или более коротко: **коммутативной (абелевой) группой**.

Определение 2. Если под операцией $\circ$ понимается операция сложения, обозначаемая знаком +, то группа называется **аддитивной**, нейтральный элемент называется **нулем (нулевым элементом)**, обозначается **0**, а обратный элемент обозначается **–a**. Если под операцией $\circ$ понимается операция умножения, обозначаемая знаком ·, то группа называется **мультипликативной**, нейтральный элемент называется **единицей (единичным элементом)** и обозначается **1**, а обратный элемент обозначается $\mathbf{a^{-1}}$.

Определение 3. Число элементов множества G, составляющего группу, называют **порядком группы**. Если порядок группы бесконечен, то такая группа называется **бесконечной**. Если же порядок группы конечен, то группа называется **конечной**.

Определение 4. Конечная группа называется **циклической**, если множество ее элементов представляет собой последовательность из результатов u-кратной композиции некоторого, так называемого, **порождающего элемента ζ** группы с самим собой при помощи бинарной операции $\circ$, где $0 \le u \le q-1$, причем q – наименьшее целое, при котором начинаются «повторения», иными словами u-кратная композиция дает такой же результат, что и 0-кратная композиция. При этом за результат 0-кратной композиции принимается нейтральный элемент $\boldsymbol{e}$, который также по определению входит в группу. Иными словами циклическая группа состоит из q элементов: $\{e,\ e\circ\zeta,\ (e\circ\zeta)\circ\zeta,\ \ldots,\ (\ldots(e\circ\zeta)\circ\zeta\circ\ldots\circ\zeta)\}$ и бинарной операции $\circ$. Порядок циклической группы равен числу q элементов группы.

Таким образом, мы рассмотрели три ключевые алгебраические структуры с одной бинарной операцией: полугруппу, моноид и группу. Перейдем теперь к рассмотрению алгебраических структур с двумя бинарными операциями.

Кольцо. Пусть задано множество элементов $\boldsymbol{G}$ и две бинарные операции: сложения, обозначим ее + и умножения, обозначим ее ·, ставящие любым двум элементам множества G некоторый третий элемент этого множества. Пусть множество элементов G вместе с бинарной операцией сложения образует коммутативную аддитивную группу: соблюдается свойство ассоциативности и коммутативности относительно операции сложения, существует нейтральный элемент по сложению, и для каждого элемента существует обратный элемент по сложению. Также пусть множество G вместе с бинарной операцией умножения образует мультипликативную полугруппу: соблюдается свойство ассоциативности относительно операции умножения. Наконец, пусть дополнительно соблюдается свойство дистрибутивности операции умножения относительно операции сложения, причем как слева, так и справа: $\forall a,b,c \in G : a \cdot (b+c) = a \cdot b + a \cdot c$ и $\forall a,b,c \in G : (b+c) \cdot a = b \cdot a + c \cdot a$. В таком случае множество элементов $\boldsymbol{G}$ и две бинарные операции: сложения и умножения, обладающие всеми вышеперечисленными свойствами, образуют алгебраическую структуру, называемую **кольцом**. Таким образом, в кольце с множеством элементов G и двумя бинарными операциями по определению соблюдаются следующие шесть свойств:

- Ассоциативность по сложению: $\forall a,b,c \in G : (a+b)+c = a+(b+c)$.
- Существование нейтрального элемента по сложению: $\forall a \in G, \exists 0 \in G : 0 + a = a + 0 = a$.
- Существование для каждого элемента множества G обратного элемента по сложению $\forall a \in G, \exists -a \in G : a + (-a) = (-a) + a = 0$.
- Коммутативность по сложению: $\forall a,b \in G : a + b = b + a$.
- Ассоциативность по умножению: $\forall a,b,c \in G : (a \cdot b) \cdot c = a \cdot (b \cdot c)$.
- Дистрибутивность операции умножения (слева и справа) относительно операции сложения: $\forall a,b,c \in G : a \cdot (b+c) = a \cdot b + a \cdot c$ и $\forall a,b,c \in G : (b+c) \cdot a = b \cdot a + c \cdot a$.

Определение 1. Если операция умножения в кольце обладает дополнительным свойством коммутативности: $\forall a,b \in G : a \cdot b = b \cdot a$, то кольцо называется коммутативным по умножению, или более коротко: **коммутативным кольцом**.

Определение 2. Если во множестве элементов кольца существует нейтральный элемент по умножению, называемый единицей и обозначаемый 1, такой, что умножение любого элемента a множества G на нейтральный элемент по умножению снова дает элемент a, иными словами: $\forall a \in G, \exists 1 \in G : 1 \cdot a = a \cdot 1 = a$, то кольцо называют **кольцом с единицей**.

В качестве примера можно привести коммутативное кольцо с единицей, состоящее из множества целых чисел **Z** и двух заданных бинарных операций: сложения + и умножения ·.

Поле. Пусть задано множество элементов ***G*** и две бинарные операции: сложения, обозначим ее + и умножения, обозначим ее ·, ставящие любым двум элементам множества G некоторый третий элемент этого множества. Пусть множество элементов ***G*** и две бинарные операции сложения и умножения образуют коммутативное кольцо с единицей (выполняются шесть свойств кольца, а также два дополнительных свойства коммутативности по умножению и существования нейтрального элемента по умножению). Пусть дополнительно соблюдается еще одно свойство: для каждого ненулевого элемента (не являющегося нейтральным элементом по сложению, то есть нулем) существует обратный элемент по умножению, иными словами $\forall a \in G : a \neq 0; \exists a^{-1} \in G : a \cdot a^{-1} = a^{-1} \cdot a = 1$. Тогда множество G и две бинарные операции сложения и умножения образуют алгебраическую структуру, называемую **полем**. Таким образом, в поле с множеством элементов ***G*** и двумя бинарными операциями по определению соблюдаются следующие девять свойств:

- Ассоциативность по сложению: $\forall a, b, c \in G : (a + b) + c = a + (b + c)$.
- Существование нейтрального элемента по сложению: $\forall a \in G, \exists 0 \in G : 0 + a = a + 0 = a$.
- Существование обратного элемента по сложению для каждого элемента множества G: $\forall a \in G, \exists -a \in G : a + (-a) = (-a) + a = 0$.
- Коммутативность по сложению: $\forall a, b \in G : a + b = b + a$.
- Ассоциативность по умножению: $\forall a, b, c \in G : (a \cdot b) \cdot c = a \cdot (b \cdot c)$.
- Дистрибутивность операции умножения (слева и справа) относительно операции сложения: $\forall a, b, c \in G : a \cdot (b + c) = a \cdot b + a \cdot c$ и $\forall a, b, c \in G : (b + c) \cdot a = b \cdot a + c \cdot a$.
- Коммутативность по умножению: $\forall a, b \in G : a \cdot b = b \cdot a$.
- Существование нейтрального элемента по умножению: $\forall a \in G, \exists 1 \in G : 1 \cdot a = a \cdot 1 = a$.
- Существование обратного элемента по умножению для каждого ненулевого элемента множества G: $\forall a \in G : a \neq 0, \exists a^{-1} \in G : a \cdot a^{-1} = a^{-1} \cdot a = 1$.

Число элементов множества G, составляющего поле, называют **порядком поля**. Порядок поля с бесконечным числом элементов бесконечен, и такое поле называется **бесконечным**. Если же число элементов конечно, то поле называется **конечным**.

Ярким примером поля является бесконечное поле действительных чисел, состоящее из множества действительных чисел ***R*** и двух бинарных операций сложения + и умножения ·.

Поле Галуа. Если число элементов поля конечно, то мы имеем **конечное поле**, которое также называют **полем Галуа** по имени их первого исследователя, Эвариста Галуа. Поле Галуа обозначается ***GF(q)***. Аббревиатура ***GF*** – это сокращение от словосочетания **Galois Field**, а ***q*** – число элементов поля по определению является **порядком поля**.

Определение 1. Поле Галуа *GF(q)* по определению содержит единичный элемент 1 (нейтральный элемент по умножению). Тогда, наименьшее целое число $\rho > 0$ такое, что $\underbrace{1+1+\ldots+1+1}_{\rho \text{ слагаемых}} = 0$, называют **характеристикой поля**. Отметим также, что при помощи единичного элемента поля можно породить циклическую аддитивную группу порядка ρ.

Определение 2. Пусть задан ненулевой элемент a поля Галуа *GF(q)*. Наименьшее целое число $\sigma > 0$ такое, что $\underbrace{a \cdot a \cdot \ldots \cdot a \cdot a}_{\sigma \text{ множителей}} = 1$, называют **порядком элемента *a***. Отметим также, что при помощи элемента a, порядок которого равен σ, можно породить циклическую мультипликативную группу порядка σ.

Особо отметим, что поля Галуа существуют не для любого q. Согласно общей алгебре, поле Галуа можно построить только для числа q, являющегося либо простым числом p, либо некоторой степенью простого числа, то есть p^m, где m – целое число ≥ 2.

Определение 3. Поле порядка q, являющегося простым числом p, называют **простым полем Галуа** и обозначают ***GF(p)***, а поле порядка $q = p^m$, называют **расширенным полем Галуа** и обозначают ***GF(p^m)***, и оно является расширением простого поля *GF(p)*.

Согласно общей алгебре для заданного числа q, равного простому числу p или некоторой его степени p^m, существует одно и только одно поле Галуа *GF(q)*.

Простое поле Галуа. Ярким примером простого поля Галуа является поле $GF(2) = <\{0, 1\}, \{+, \cdot\}>$, состоящее всего из двух элементов: 0, являющегося нейтральным элементом по сложению, и 1, являющегося нейтральным элементом по умножению, а также двух операций, сложения и умножения, со следующими «таблицами сложения и умножения»:

$GF(2)$:

+	0	1
0	0	1
1	1	0

·	0	1
0	0	0
1	0	1

Заметим, что обратным элементом по сложению для 0 является сам элемент 0, а для элемента 1 – сам элемент 1. Обратным элементом по умножению для единственного ненулевого элемента 1 – сам элемент 1. Порядок поля равен 2, поскольку поле содержит всего два элемента, кроме того, характеристика поля также равна 2, поскольку $1+1=0$, т.е. две единицы в сумме уже дают нулевой элемент.

Отметим, что простое поле $GF(2)$ является наименьшим возможным простым полем.

Приведем еще один пример простого поля $GF(3) = <\{0, 1, 2\}, \{+, \cdot\}>$, с нейтральным элементом по сложению: 0, с нейтральным элементом по умножению: 1, и со следующими «таблицами сложения и умножения»:

$GF(3)$:

+	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

·	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

Заметим, что обратным элементом по сложению для 0 является сам элемент 0, для 1 – элемент 2, для 2 – элемент 1. Обратным элементом по умножению элемента 1 является сам элемент 1, а для элемента 2 – сам элемент 2. Порядок поля равен 3, поскольку поле содержит всего три элемента, кроме того, характеристика поля также равна 3, поскольку $1+1+1=(1+1)+1=2+1=0$, т.е. три единицы в сумме уже дают нулевой элемент.

Наконец, приведем пример простого поля $GF(5) = <\{0, 1, 2, 3, 4\}, \{+, \cdot\}>$, с нейтральным элементом по сложению: 0, с нейтральным элементом по умножению: 1, и со следующими «таблицами сложения и умножения»:

$GF(5)$:

+	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

·	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

Заметим, что обратным элементом по сложению для 0 является сам элемент 0, для 1 – элемент 4, для 2 – элемент 3, для 3 – элемент 2, для 4 – элемент 1. Обратным элементом по умножению для 1 является сам элемент 1, для 2 – элемент 3, для 3 – элемент 2, для 4 – сам элемент 4. Порядок поля равен 5, поскольку оно состоит из пяти элементов. Кроме того, характеристика поля равна 5, поскольку сумма из 5 единичных элементов уже дают нулевой элемент: $1+1+1+1+1=(((1+1)+(1+1))+1)=((2+2)+1)=(4+1)=0$.

Арифметика и свойства простого поля. На примере вышеприведенных трех простых полей, мы можем провести ряд аналогий с полем действительных чисел:

- Элементы простого поля *GF(p)* – это не что иное, как множество различных неотрицательных остатков по модулю простого числа *p*. Под «остатком по модулю простого числа *p*», понимается остаток, получаемый при выполнении операции деления по правилам поля действительных чисел <*R*, {+, ·}> некоторого целого числа, принадлежащего множеству действительных чисел *R*, на простое число *p*, также принадлежащего множеству *R*. Остаток по модулю *p* принято обозначать как ***mod p***.
- Операция сложения, заданная для простого поля *GF(p)* – это не что иное, как операция сложения по правилам поля действительных чисел <*R*, {+, ·}> с последующим вычислением остатка суммы по модулю *p*.
- Операция умножения, заданная для простого поля *GF(p)* – это не что иное, как операция умножения по правилам поля действительных чисел <*R*, {+, ·}> с последующим вычислением остатка произведения по модулю *p*.
- Обратный элемент по сложению для нулевого элемента 0 простого поля *GF(p)* – это сам нулевой элемент 0. Обратный элемент для ненулевого элемента *a* это элемент, который можно вычислить, как $p - a$, где вычитание выполняется по правилам поля действительных чисел <*R*, {+, ·}>. Заметим, что если к разности $p - a$ дополнительно применять вычисление остатка по модулю *p*, иными словами, использовать формулу $(p - a) \bmod p$ для вычисления обратного элемента простого поля *GF(p)* по сложению, то формула будет справедливой для всех элементов поля, включая нулевой элемент.
- Обратного элемента по умножению для нулевого элемента 0 не существует. Обратный элемент по умножению для ненулевого элемента *a* простого поля *GF(p)* является решением уравнения $(a \cdot b) \bmod p = 1$, заданного в поле действительных чисел <*R*, {+, ·}>, такое, что решение $b = a^{-1}$ также является элементом поля *GF(p)*.

Согласно общей алгебре, вышеприведенные «аналогии» с привычным для нас полем действительных чисел, справедливы для любого простого поля Галуа *GF(p)*.

Иными словами, для любого простого поля *GF(p)* справедливо следующее:

- $\forall p \in P : p \geq 2 \;\&\; \forall a \in R : a \in \{0,1,\ldots, p-1\} \Rightarrow a \in GF(p)$.
- $\forall p \in P : p \geq 2 \;\&\; \forall a,b \in GF(p) \Rightarrow \underbrace{a+b}_{GF(p)} = \underbrace{(a+b) \bmod p}_{<R,\{+,\cdot\}>}$.
- $\forall p \in P : p \geq 2 \;\&\; \forall a,b \in GF(p) \Rightarrow \underbrace{a \cdot b}_{GF(p)} = \underbrace{(a \cdot b) \bmod p}_{<R,\{+,\cdot\}>}$. (1.1)
- $\forall p \in P : p \geq 2 \;\&\; \forall a \in GF(p) \Rightarrow \underbrace{-a}_{GF(p)} = \underbrace{(p-a) \bmod p}_{<R,\{+,\cdot\}>}$.
- $\forall p \in P : p \geq 2 \;\&\; \forall a \in GF(p) : a \neq 0, \exists a^{-1} \in GF(p) : \underbrace{a \cdot a^{-1} = 1}_{GF(p)} \Leftrightarrow \underbrace{(a \cdot a^{-1}) \bmod p = 1}_{<R,\{+,\cdot\}>}$.

Таким образом, мы получили важные «связующие звенья» между операциями, выполняемыми в простом поле Галуа *GF(p)*, и привычными для нас операциями поля действительных чисел <*R*, {+, ·}>. Заметим также, что операция сложения элементов, умножения элементов и вычисления обратного элемента по сложению для простого поля выполняются с помощью не более двух соответствующих действий в поле действительных чисел. Что же касается операции вычисления обратного элемента по умножению, то она представляет собой некоторую процедуру поиска элемента, удовлетворяющего уравнению $(a \cdot x) \bmod p = 1$, и требует значительно большего времени для ее выполнения.

Отметим также, что рассмотренную выше «арифметику» простого поля *GF(p)* также называют модулярной арифметикой [4, 5], поскольку все операции с элементами поля выполняются по модулю *p* с точки зрения традиционной алгебры действительных чисел.

Теперь отдельно остановимся на вычислении обратного элемента по умножению в простом поле *GF(p)* для некоторого ненулевого элемента a. Очевидно, что для небольших простых полей, обратный элемент $b = a^{-1}$ можно найти простым перебором всех ненулевых элементов простого поля с проверкой каждого из них на условие $(a \cdot b) \bmod p = 1$. В принципе можно вообще избавиться от необходимости какого-либо перебора, если заранее сгенерировать «таблицу обратных элементов по умножению», и тогда поиск обратного элемента по умножению сведется к однократному обращению к этой таблице. Однако, в случае достаточно большого поля, полный перебор становится слишком медлительным по времени, а для хранения большой «таблицы обратных элементов» может потребоваться слишком много памяти. В таком случае можно применять алгоритм вычисления обратного элемента по умножению, базирующийся на расширенном алгоритме Евклида.

Идея алгоритма заключается в следующем. С одной стороны условие $(a \cdot b) \bmod p = 1$ эквивалентно условию $a \cdot b - p \cdot t = 1$, где t - частное от деления $a \cdot b$ на простое число p. С другой стороны расширенный алгоритм Евклида позволяет найти наибольший общий делитель $НОД(a, p)$ для чисел a и p, а также g и h такие, что $a \cdot g + p \cdot h = НОД(a, p)$. А теперь заметим, что p простое число, которое делится только на себя и единицу, а это значит, что наибольшим общим делителем для a и p может быть только 1: $НОД(a, p) = 1$. В такой ситуации имеет место быть равенство: $a \cdot g + p \cdot h = 1$. Следует особо отметить, что с учетом того, что p простое число, а число a, будучи ненулевым элементом поля *GF(p)*, находится в диапазоне $[1, p-1]$, расширенный алгоритм Евклида дает ненулевое число g, которое принадлежит одному из двух диапазонов: $[-(p-1)/2, -1]$ или $[1, (p-1)/2]$. В случае, если число $g > 0$, можно установить прямую связь между равенством $a \cdot g + p \cdot h = 1$ и условием $a \cdot b - p \cdot t = 1$ поиска обратного элемента по умножению: $b = g$ и $t = -h$. В случае, если число $g < 0$, то мы можем воспользоваться свойством остатков по модулю: $g \bmod p = (g + p) \bmod p$, и тогда равенство $a \cdot g + p \cdot h = 1$ мы можем преобразовать следующим образом $a \cdot g + p \cdot h = 1 \Rightarrow a \cdot (g + p) - a \cdot p + p \cdot h = 1 \Rightarrow a \cdot (g + p) + p \cdot (h - a) = 1$, и установить связь с условием $a \cdot b - p \cdot t = 1$ в виде: $b = g + p$ и $t = a - h$. Обобщая все вышесказанное, мы можем окончательно записать связь между b и g, а также между t и h:

$$b = \begin{cases} g, & g > 0 \\ g + p, & g < 0 \end{cases} ; \quad t = \begin{cases} -h, & g > 0 \\ a - h, & g < 0 \end{cases} ; \quad \begin{matrix} a \cdot g + p \cdot h = НОД(a, p) \\ НОД(a, p) = 1 \end{matrix} ; \quad \begin{matrix} p \in P \\ 1 \le a \le p - 1 \end{matrix}$$

Теперь кратко рассмотрим суть расширенного алгоритма Евклида, определяющий наибольший общий делитель для двух чисел a и p, и вычисляющий числа g и h. В математическом виде алгоритм можно записать следующим образом:

$$\left\{ \begin{array}{c} r^{(0)} = p \quad g^{(0)} = 0 \quad h^{(0)} = 1 \\ r^{(1)} = a \quad g^{(1)} = 1 \quad h^{(1)} = 0 \\ s = 2 \ldots n : r^{(n)} = 0 \\ r^{(s-2)} = q^{(s)} \cdot r^{(s-1)} + r^{(s)} \Rightarrow \left\{ \begin{matrix} q^{(s)} \\ r^{(s)} \end{matrix} \right. \\ g^{(s)} = g^{(s-2)} - g^{(s-1)} \cdot q^{(s)} \\ h^{(s)} = h^{(s-2)} - h^{(s-1)} \cdot q^{(s)} \end{array} \right\} \Rightarrow \left\{ \begin{array}{c} НОД(a, p) = r^{(n-1)} \\ g = g^{(n-1)} \\ h = h^{(n-1)} \end{array} \right\} \qquad (1.2)$$

Особо отметим, что при работе алгоритма все операции: сложения, вычитания и умножения, а также деления «в столбик» одного числа на другое число с целью нахождения частного и остатка от деления – выполняются по правилам арифметики поля действительных чисел, и все промежуточные результаты вычислений рассматриваются также как действительные числа. Интерпретация результатов как элементов простого поля *GF(p)* производится после вычисления чисел b и t по приведенным выше соотношениям.

«Ядром» алгоритма является ключевое рекуррентное соотношение $r^{(s-2)} = q^{(s)} \cdot r^{(s-1)} + r^{(s)}$, с помощью которого на каждой итерации, начиная с итерации $s = 2$, вычисляется текущий остаток $r^{(s)}$ и частное $q^{(s)}$ от деления остатков $r^{(s-2)}$ на $r^{(s-1)}$ двух «предыдущих» итераций. В качестве «начальных» условий для алгоритма принимаются: $r^{(0)} = p$ и $r^{(1)} = a$. В расширенном алгоритме Евклида на каждой итерации частное $q^{(s)}$ также используется для формирования чисел g и h при помощи двух дополнительных рекуррентных соотношений: $g^{(s)} = g^{(s-2)} - g^{(s-1)} \cdot q^{(s)}$ и $h^{(s)} = h^{(s-2)} - h^{(s-1)} \cdot q^{(s)}$, причем в качестве «начальных» условий для них принимаются: $g^{(0)} = 0$, $g^{(1)} = 1$, $h^{(0)} = 1$ и $h^{(1)} = 0$. Алгоритм выполняется до тех пор, пока на некотором шаге $s = n$ не будет получен остаток $r^{(n)} = 0$. В этом случае алгоритм завершается, а в качестве результатов принимаются результаты предпоследней итерации: $НОД(a, p) = r^{(n-1)}$, $g = g^{(n-1)}$ и $h = h^{(n-1)}$.

Отметим, что поскольку в нашем случае цель – поиск обратного элемента по умножению в простом поле *GF(p)*, то вычисление числа h нам фактически не требуется, для получения обратного элемента по умножению достаточно вычислить число g.

Рассмотрим пример вычисления обратного элемента по умножению в простом поле *GF(p)* при помощи вышеописанного алгоритма.

Пример. Вычислим в простом поле GF(97) обратный элемент по умножению для элемента $a = 10$. Сначала применим расширенный алгоритм Евклида для чисел $a = 10$ и $p = 97$, и определим число g. В качестве «начальных» условий алгоритма принимаются $r^{(0)} = p = 97$, $r^{(1)} = a = 10$, $g^{(0)} = 0$ и $g^{(1)} = 1$.

Начинаем с итерации $s = 2$. Делим число $r^{(0)} = 97$ на число $r^{(1)} = 10$, получаем частное $q^{(2)} = 9$ и остаток $r^{(2)} = 7$ от деления. Используя частное $q^{(2)}$, также вычисляем $g^{(2)} = g^{(0)} - g^{(1)} \cdot q^{(2)} = 0 - 1 \cdot 9 = -9$.

Переходим к итерации $s = 3$. Делим число $r^{(1)} = 10$ на число $r^{(2)} = 7$, получаем частное $q^{(3)} = 1$ и остаток $r^{(3)} = 3$ от деления. Используя частное $q^{(3)}$, также вычисляем $g^{(3)} = g^{(1)} - g^{(2)} \cdot q^{(3)} = 1 - (-9) \cdot 1 = 10$.

Переходим к итерации $s = 4$. Делим число $r^{(2)} = 7$ на число $r^{(3)} = 3$, получаем частное $q^{(4)} = 2$ и остаток $r^{(4)} = 1$ от деления. Используя частное $q^{(4)}$, также вычисляем $g^{(4)} = g^{(2)} - g^{(3)} \cdot q^{(4)} = -9 - 10 \cdot 2 = -29$.

Переходим к итерации $s = 5$. Делим число $r^{(3)} = 3$ на число $r^{(4)} = 1$, получаем частное $q^{(5)} = 3$ и остаток $r^{(5)} = 0$ от деления. Поскольку остаток от деления равен нулю, то работа алгоритма на этом завершается, а в качестве результатов принимаются результаты вычислений на предпоследней итерации: $НОД(a, p) = r^{(4)} = 1$ и $g = g^{(4)} = -29$.

Поскольку в результате работы расширенного алгоритма Евклида мы получили отрицательное число $g = -29$, то в качестве итогового результата, являющегося обратным элементом по умножению для элемента $a = 10$ в простом поле GF(97) принимается число $b = g + p = -29 + 97 = 68$.

Проверим полученный результат: $(a \cdot b) \bmod p = (10 \cdot 68) \bmod 97 = 680 \bmod 97 = 1$. Таким образом, мы убедились в том, что элемент $b = 68$ является обратным элементом по умножению для элемента $a = 10$ в простом поле GF(97).

Примечание. Следует отметить, что вышерассмотренный алгоритм также может применяться для поиска обратного элемента по умножению в коммутативном кольце с единицей, состоящем из множества целых неотрицательных чисел, являющихся остатками по модулю некоторого целого числа $n > 1$, не являющегося простым числом. Однако, в кольце по определению не для всякого ненулевого элемента существует обратный элемент по умножению, и для таких элементов наибольший общий делитель $НОД(a, n) \neq 1$.

Ниже на рисунке 1.1 приведена схема алгоритма для вычисления обратного элемента по умножению b для заданного ненулевого элемента a в простом поле *GF(p)*. В алгоритме отсутствует «итерационная» переменная s, поскольку для выполнения вычислений и отслеживания условия завершения алгоритмы можно обойтись без нее.

Отметим, что все операции с числами, включая вычисление частного и остатка от деления двух чисел, выполняются по обычным правилам арифметики действительных чисел.

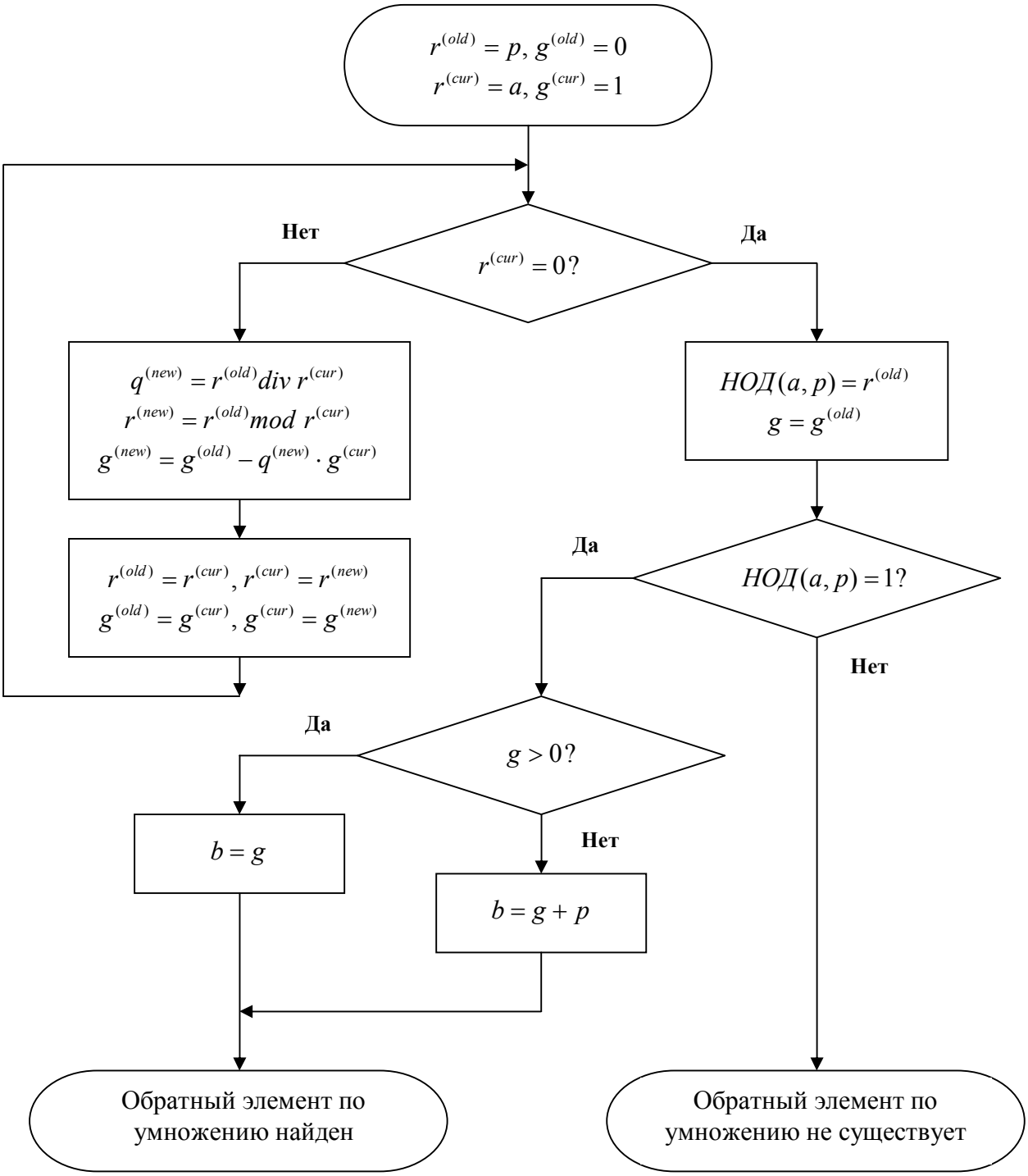

Рис. 1.1. Схема алгоритма вычисления обратного элемента по умножению для заданного ненулевого элемента в простом поле *GF(p)*.

Теперь отметим ряд важных свойств простых полей Галуа *GF(p)*:

Свойство 1. Характеристика любого простого поля Галуа *GF(p)* равна простому числу *p*. Иными словами, сумма из *p* единичных элементов простого поля *GF(p)* равна нулю.

Свойство 2. Простое поле Галуа *GF(p)* содержит в себе циклическую аддитивную группу порядка *p* с порождающим элементом 1. Иными словами, в простом поле Галуа *GF(p)*, используя единичный элемент 1 в качестве порождающего, операцию сложения и нулевой элемент 0 (нейтральный элемент по сложению) можно породить циклическую аддитивную группу порядка *p*. Действительно, в качестве 0-й композиции (операция композиции в нашем случае это бинарная операция сложения, заданная для простого поля) мы принимаем нейтральный элемент по сложению 0. Далее, складывая 0 с порождающим элементом 1, мы получаем элемент 1. Продолжая далее, получаем последовательно все элементы поля вплоть до $p-1$. При попытке сложить элемент $p-1$ с элементом 1 мы получаем снова 0 и это неудивительно, так как характеристика поля равна *p*. На этом цикл замыкается, и в итоге имеем циклическую аддитивную группу порядка p: $<\{0,1,\ldots,p-1\},\{+\}>$, порожденную при помощи элемента 1, в которой 0 является нейтральным элементом по сложению и для каждого элемента имеется обратный элемент по сложению. «Таблица сложения» для операции сложения точно такая же, как и у поля *GF(p)*.

Свойство 3. Для любого элемента *a* простого поля *GF(p)* и целого числа $n \geq 0$ сумма $\underbrace{a+\ldots+a}_{n\ слагаемых} = \overbrace{\lambda \cdot a}^{GF(p)}$, где $\lambda = \overbrace{n \bmod p}^{<R,\{+,\cdot\}>}$. Действительно, $\underbrace{a+\ldots+a}_{n\ слагаемых}^{GF(p)} = \underbrace{(a+\ldots+a) \bmod p}_{n\ слагаемых}^{<R,\{+,\cdot\}>} = \overbrace{(n \cdot a) \bmod p}^{<R,\{+,\cdot\}>} = \overbrace{((n \bmod p) \cdot a) \bmod p}^{<R,\{+,\cdot\}>} = \overbrace{(\lambda \cdot a) \bmod p}^{<R,\{+,\cdot\}>} = \overbrace{\lambda \cdot a}^{GF(p)}$, обозначая $\lambda = \overbrace{n \bmod p}^{<R,\{+,\cdot\}>}$. Заметим, что сам по себе множитель λ также является элементом простого поля *GF(p)*. Также особо отметим, что если *n* кратно *p*, то $\lambda = n \bmod p = 0$, и для любого *a* сумма равна нулю.

Свойство 4. Простое поле Галуа *GF(p)* содержит в себе циклическую мультипликативную группу порядка $p-1$, состоящую из всех ненулевых элементов поля, причем порожденную одним из этих элементов, имеющего порядок $p-1$ в поле *GF(p)*. Рассмотрим простое поле *GF(p)*. Очевидно, что если исключить нулевой элемент, то остается множество из $p-1$ ненулевых элементов. Выберем среди них элемент α такой, что произведение из $p-1$ сомножителей α дает единичный элемент 1, иными словами порядок элемента α в простом поле Галуа *GF(p)* равен $p-1$. Именно его мы и будем использовать в качестве порождающего элемента циклической мультипликативной группы. В качестве 0-й композиции (произведения) мы принимаем нейтральный элемент по умножению 1. Тогда умножая 1 на α, мы получаем следующий элемент. Далее, продолжая аналогичным образом, получаем все остальные ненулевые элементы, и так вплоть до произведения из $p-2$ сомножителей α. При попытке очередного умножения получаем произведение из $p-1$ сомножителей α, которое будет равным 1, на этом цикл замыкается. В итоге получаем циклическую мультипликативную группу: $<\{1,\alpha,\alpha\cdot\alpha,\ldots,\underbrace{\alpha\cdot\ldots\cdot\alpha}_{(p-2)\ сомножителей}\},\{\cdot\}>$, порожденную при помощи элемента α, в которой 1 является нейтральным элементом по умножению и для каждого элемента есть обратный элемент по умножению. «Таблица умножения» для операции умножения точно такая же, как и у поля *GF(p)*.

Определение 1. Элемент α, имеющий порядок $p-1$, при помощи которого порождаются все ненулевые элементы простого поля Галуа *GF(p)* называют **примитивным элементом поля**. В простом поле Галуа *GF(p)* могут быть несколько примитивных элементов. Так, например, в поле GF(5) имеются два примитивных элемента: 2 и 3, используя их можно получить все ненулевые элементы поля: {1, 1·2 = 2, 1·2·2 = 4, 1·2·2·2 = 3} и {1, 1·3 = 3, 1·3·3 = 4, 1·3·3·3 = 2}. В приложении 1 приведены списки примитивных элементов для некоторых простых полей *GF(p)*.

Определение 2. Пусть задано некоторое целое $u \geq 0$. Тогда, будем называть ***u*-й степенью элемента *a*** простого поля Галуа *GF(p)*, произведение следующего вида: $\underbrace{a \cdot \ldots \cdot a}_{u \text{ сомножителей}}$, обозначаемую a^u. Причем, в качестве 0-й степени любого элемента простого поля Галуа, включая нулевой элемент 0, будем считать единичный элемент 1. Иными словами: $\forall a \in GF(p), \forall u \in Z : u > 0 \Rightarrow a^u = \underbrace{a \cdot \ldots \cdot a}_{u \text{ сомножителей}}$, $\forall a \in GF(p) \Rightarrow a^0 = 1$.

Определение 3. Пусть задан некоторый примитивный элемент поля α, при помощи которого можно породить все ненулевые элементы простого поля Галуа *GF(p)*. Пусть задан некоторый ненулевой элемент *a* простого поля Галуа. Тогда будем называть **логарифмом элемента *a* по основанию примитивного элемента *α*** такое целое неотрицательное число *u*, для которого $\alpha^u = a$. Логарифм элемента *a* по основанию примитивного элемента α будем обозначать $\log_\alpha a$. Кроме того, будем считать недопустимой попытку применения логарифма по основанию примитивного элемента α к нулевому элементу. Иными словами: $\forall a \in GF(p) : a \neq 0, \exists u \in Z \,\&\, u \geq 0 : \alpha^u = a$ и $\log_\alpha 0 \Rightarrow$ *Ошибка*.

Свойство 5. Множество элементов простого поля Галуа *GF(p)* состоит из нулевого элемента 0 и множества *u*-х степеней примитивного элемента α, таких что $u = 0 \ldots p-2$. Иными словами, $GF(p) =< \{0, \alpha^0, \alpha^1, \ldots, \alpha^{p-2}\}, \{+,\cdot\} >$. Это свойство вытекает из свойства 3 и определения понятия «степени элемента поля». Убедимся в этом свойстве на примере простого поля *GF*(5), взяв в нем в качестве примитивного элемента $\alpha = 2$. Имеем: $GF(p) =< \{0,2^0,2^1,2^2,2^3\}, \{+,\cdot\} >=< \{0,1,2,2 \cdot 2, 2 \cdot 2 \cdot 2\}, \{+,\cdot\} >=< \{0,1,2,4,3\}, \{+,\cdot\} >$.

Кольцо логарифмов и его арифметика. Обратим особое внимание на то, что логарифмы всех ненулевых элементов простого поля *GF(p)* по основанию примитивного элемента α этого поля сами по себе представляют множество вида: {0, 1, 2,…, *p* – 2}. Это следует из свойства 4 для простого поля, согласно которому, любое простое поле $GF(p) =< \{0, \alpha^0, \alpha^1, \ldots, \alpha^{p-2}\}, \{+,\cdot\} >$. Тогда для всякого простого числа $p \geq 3$, множество логарифмов, состоящее из $p-1$ элементов, содержит нулевой элемент 0, и единичный элемент 1. Это следует из того, что среди элементов простого поля по определению имеется единичный элемент 1, логарифм которого равен 0, а также свойству 4 имеется примитивный элемент α порядка $p-1$, логарифм которого, очевидно, равен 1. Тогда, если для множества логарифмов определить операцию сложения, как эквивалент операции сложения по модулю $p-1$ в поле действительных чисел, а также операцию умножения, как эквивалент операции умножения по модулю $p-1$ в поле действительных чисел, то о множестве логарифмов ненулевых элементов поля *GF(p)* по основанию примитивного элемента α вместе с заданными операциями сложения и умножения, можно говорить, как об алгебраической структуре, являющейся коммутативном кольцом с единицей.

Таким образом, пусть имеется простое поле *GF(p)*, $p \geq 3$. Пусть L = {0, 1, 2,…, *p* – 2} множество логарифмов ненулевых элементов простого поля *GF(p)* по основанию некоторого его примитивного элемента α. Пусть задана операция сложения логарифмов, такая, что $\forall u, v \in L : u + v = \underbrace{(u+v) \bmod (p-1)}_{< R, \{+,\cdot\} >}$ и операция умножения логарифмов, такая, что $\forall u, v \in L : u \cdot v = \underbrace{(u \cdot v) \bmod (p-1)}_{< R, \{+,\cdot\} >}$. Тогда **множество логарифмов вместе двумя заданными операциями образует коммутативное кольцо с единицей**. Обозначим его ***LR*(*p* – 1)**.

Можно задаться вопросом о том, почему множество логарифмов вместе с заданными двумя операциями сложения и умножения, является только коммутативным кольцом с единицей, и «не дотягивает» до более «сильной» алгебраической структуры – поля. Ответ здесь прост: не для всякого ненулевого логарифма ненулевого элемента поля *GF(p)* можно найти обратный логарифмический элемент по умножению такой, что он также принадлежит множеству логарифмов *L*. Рассмотрим это на примере простого поля *GF*(5). Кольцо логарифмов *LR*(4) состоит из множества {0, 1, 2, 3} логарифмов ненулевых элементов простого поля GF(5) по основанию примитивного элемента $\alpha = 2$ поля. Иными словами: $\{2^0, 2^1, 2^2, 2^3\} = \{1,2,4,3\}$. Операции сложения и умножения логарифмов можем представить в табличном виде (их нетрудно получить, используя соответствующие операции сложения и умножения по модулю 4 в поле действительных чисел):

$LR(4)$:

+	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

·	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

Если посмотреть на таблицу сложения для логарифмов, нетрудно заметить, что обратный логарифмический элемент по сложению существует для всех логарифмических элементов: для 0 – это 0, для 1 – это 3, для 2 – это 2, для 3 – это 1. Заметим, что обратный логарифмический элемент по сложению нетрудно вычислить, используя соответствующую разность $((p-1)-u)$ по модулю $(p-1)$ в поле действительных чисел.

А вот что касается обратного элемента по умножению, то по таблице умножения нетрудно заметить, что для логарифмического элемента 2 не существует обратного логарифмического элемента по умножению. Таким образом, *LR*(4) «не дотягивает» до поля. Можно также задаться вопросом о том, почему в *LR*(4) не для всех логарифмических элементов можно найти обратный логарифмический элемент по умножению. Ответ также прост – число 4 не является простым. Тогда, учитывая, что кольцо логарифмов *LR*(p – 1), состоящее из p – 1 элементов, образуется для простого поля *GF(p)*, где $p \geq 3$ является простым числом, и очевидно, что нечетным, ясно, что число p – 1 всегда будет четным, а значит, как минимум, кратным числу 2. Известно только одно четное число, являющееся простым, это число 2. Иными словами, только для одного и единственного простого поля Галуа *GF*(3), соответствующее кольцо логарифмов *LR*(2) оказывается полем *GF*(2), для всех остальных простых полей *GF(p)*, $p \geq 5$, кольцо логарифмов *LR*(p – 1) не является полем.

Подведем итоги всему вышесказанному о кольце логарифмов *LR*(p – 1):

- $\forall p \in P : p \geq 3 \;\&\; \forall u \in R : u \in \{0,1,\ldots,p-2\} \Rightarrow u \in LR(p-1)$.
- $\forall p \in P : p \geq 3 \;\&\; \forall u,v \in LR(p-1) \Rightarrow \underset{LR(p-1)}{\underbrace{u+v}} = \underset{<R,\{+,\cdot\}>}{\underbrace{(u+v) \bmod (p-1)}}$.
- $\forall p \in P : p \geq 3 \;\&\; \forall u,v \in LR(p-1) \Rightarrow \underset{LR(p-1)}{\underbrace{u \cdot v}} = \underset{<R,\{+,\cdot\}>}{\underbrace{(u \cdot v) \bmod (p-1)}}$. (1.3)
- $\forall p \in P : p \geq 3 \;\&\; \forall u \in LR(p-1) \Rightarrow \underset{LR(p-1)}{\underbrace{-u}} = \underset{<R,\{+,\cdot\}>}{\underbrace{((p-1)-u) \bmod (p-1)}}$.

Теперь, когда мы ввели понятие логарифма, а также убедились, что из самого множества логарифмов и двух бинарных операций сложения и умножения можно образовать коммутативное кольцо с единицей, мы можем по-новому взглянуть на операцию умножения элементов простого поля *GF(p)*, выразив произведение двух элементов простого поля через соответствующие им логарифмы. Кроме того, с помощью логарифмов нам также удастся упростить задачу нахождения обратного элемента простого поля *GF(p)* по умножению, и, наконец, также найти простой способ деления элементов простого поля *GF(p)*.

Логарифмы в арифметике простого поля. Пусть заданы два ненулевых элемента a и b простого поля Галуа $GF(p)$, $p \geq 3$. Для них существуют логарифмы по основанию примитивного элемента α: $u = \log_\alpha a$ и $v = \log_\alpha b$, принадлежащие кольцу $LR(p-1)$. Тогда, произведение элементов a и b: $a \cdot b = \alpha^u \cdot \alpha^v = \underbrace{\alpha \cdot \ldots \cdot \alpha}_{u \text{ сомножителей}} \cdot \underbrace{\alpha \cdot \ldots \cdot \alpha}_{v \text{ сомножителей}} \Rightarrow$ $a \cdot b = \underbrace{\alpha \cdot \ldots \cdot \alpha}_{(u+v)\text{сомножителей}}$. Теперь заметим, что если суммарное количество сомножителей $u+v$ оказывается больше либо равно $p-1$, то произведение из $u+v$ можно сомножителей разбить на две группы: основную группу из $p-1$ сомножителей и оставшуюся группу из $(u+v)\bmod(p-1)$ сомножителей. Причем, мы неявно подразумеваем, что операции сложения, вычитания и вычисления остатка по модулю $p-1$ для логарифмов мы выполняем по правилам алгебры действительных чисел. Иными словами, мы можем записать: $a \cdot b = \left(\underbrace{\alpha \cdot \ldots \cdot \alpha}_{(p-1)\text{сомножителей}}\right) \cdot \left(\underbrace{\alpha \cdot \ldots \cdot \alpha}_{(u+v)\bmod(p-1)\text{сомножителей}}\right)$. Заметим, что если $u+v$ окажется равной $p-1$, то вторая группа будет отсутствовать, так как $(p-1)\bmod(p-1) = 0$. Теперь, вспомним, что примитивный элемент простого поля $GF(p)$ по определению имеет порядок $p-1$, иными словами: $\underbrace{\alpha \cdot \ldots \cdot \alpha}_{(p-1)\text{сомножителей}} = \alpha^{p-1} = 1$. Тогда получаем: $a \cdot b = \underbrace{\alpha \cdot \ldots \cdot \alpha}_{(u+v)\bmod(p-1)\text{сомножителей}}$. Заметим, что полученная формула справедлива также и в случае, если $u+v$ меньше $p-1$, поскольку операция вычисления остатка по модулю $p-1$ в этом случае даст исходную сумму $u+v$. Таким образом, окончательно: $a \cdot b = \alpha^w$, где $\underbrace{w = (u+v)\bmod(p-1)}_{<R,\{+,\cdot\}>}$. Особо отметим, что операции $(u+v)\bmod(p-1)$ для логарифмов, представленных в поле действительных чисел, соответствует операции сложения в кольце логарифмов $LR(p-1)$. Тогда математически вышеперечисленные рассуждения можно выразить таким образом: $\forall p \in P : p \geq 3 \;\&\; \forall a,b \in GF(p) : (a \neq 0 \;\&\; b \neq 0) \;\&\; \forall u,v \in LR(p-1) :$ $(u = \log_\alpha a) \;\&\; (v = \log_\alpha b), \exists w \in LR(p-1) : \underbrace{w = u+v}_{LR(p-1)} \Leftrightarrow \underbrace{w = (u+v)\bmod(p-1)}_{<R,\{+,\cdot\}>} \Leftrightarrow \underbrace{\alpha^w = a \cdot b}_{GF(p)}$. В случае же, если один из элементов, a или b, равен нулю, то, формулу нельзя применить, однако, очевидно, что произведение будет равным нулю в силу свойств самого поля.

Теперь попробуем выразить трудоемкую операцию нахождения обратного элемента по умножению для ненулевого элемента a простого поля $GF(p)$, $p \geq 3$. Для него по определению существует логарифм по основанию примитивного элемента α: $u = \log_\alpha a$. Тогда обратный элемент a^{-1} по определению самого поля это такой элемент, что $a \cdot a^{-1} = 1$. Обратный элемент a^{-1}, очевидно, также не равен нулю, и для него также существует логарифм $v = \log_\alpha(a^{-1})$. Теперь, учтем, что $\alpha^0 = 1$ и $\alpha^u = a$. Тогда условие $a \cdot a^{-1} = 1$ записывается, как $\alpha^u \cdot \alpha^v = \alpha^0$. Из него можно вывести условие для логарифмов в поле действительных чисел: $\underbrace{(u+v)\bmod(p-1) = 0}_{<R,\{+,\cdot\}>}$. Отсюда получаем: $\underbrace{v = ((p-1)-u)\bmod(p-1)}_{<R,\{+,\cdot\}>}$.

Таким образом, благодаря логарифмам, нахождение обратного элемента по умножению для ненулевого элемента *a* из поисковой процедуры превращается в простую операцию с логарифмом элемента *a*. Особо отметим, что операция $v = ((p-1)-u) \bmod (p-1)$ с логарифмом, представленным в поле действительных чисел, полностью соответствует операции вычисления обратного элемента по сложению в соответствующем кольце логарифмов *LR*(*p* − 1), которое мы определили выше. Математически полученный вывод выглядит так: $\forall p \in P : p \geq 3 \;\&\; \forall a \in GF(p) : a \neq 0 \;\&\; \forall u \in LR(p-1) : (u = \log_{\alpha} a), \exists v \in LR(p-1) :$

$$\underbrace{v = -u}_{LR(p-1)} \Leftrightarrow \underbrace{v = ((p-1)-u) \bmod (p-1)}_{< R, \{+,\cdot\} >} \Leftrightarrow \underbrace{\alpha^{v} = a^{-1}}_{GF(p)}.$$

Теперь, можем также выразить операцию деления одного ненулевого элемента *a* на другой ненулевой элемент *b* простого поля *GF(p)*, $p \geq 3$. Тогда по определению для них существуют логарифмы по основанию примитивного элемента α: $u = \log_{\alpha} a$ и $v = \log_{\alpha} b$. Тогда, учитывая все вышеприведенные рассуждения об умножении двух ненулевых элементов и вычислении обратного элемента по умножению для ненулевого элемента, отношение двух ненулевых элементов можно выразить как: $a / b = a \cdot b^{-1} = \alpha^{u} \cdot \left(\alpha^{v}\right)^{-1} = \alpha^{w}$, где $\underbrace{w = (u + ((p-1)-v) \bmod (p-1)) \bmod (p-1) = (u + ((p-1)-v)) \bmod (p-1)}_{< R, \{+,\cdot\} >}$. Заметим, что в кольце логарифмов $LR(p-1)$ мы имеем: $\underbrace{-v}_{LR(p-1)} = \underbrace{((p-1)-v) \bmod (p-1)}_{< R, \{+,\cdot\} >}$. Тогда, получаем: $\underbrace{w = u + (-v) = u - v}_{LR(p-1)}$, то есть операции деления ненулевых элементов в простом поле *GF(p)* соответствует операция вычитания логарифмов элементов в кольце $LR(p-1)$. Тогда: $\forall p \in P : p \geq 3 \;\&\; \forall a, b \in GF(p) : (a \neq 0 \;\&\; b \neq 0) \;\&\; \forall u, v \in LR(p-1) : (u = \log_{\alpha} a) \;\&\; (v = \log_{\alpha} b)$,

$$\exists w \in LR(p-1) : \underbrace{w = u - v}_{LR(p-1)} \Leftrightarrow \underbrace{w = (u + ((p-1)-v)) \bmod (p-1)}_{< R, \{+,\cdot\} >} \Leftrightarrow \underbrace{\alpha^{w} = a / b}_{GF(p)}.$$

В случае, если элемент $a = 0$, то формулу применять нельзя, однако, при условии, что $b \neq 0$, очевидно, что отношение элементов будет равным нулю в силу свойств самого поля.

Наконец, рассмотрим операцию возведения ненулевого элемента *a* простого поля *GF(p)*, $p \geq 3$, в степень *v*, являющуюся элементом соответствующего кольца логарифмов *LR*(*p* − 1). По определению $a^{v} = \underbrace{a \cdot \ldots \cdot a}_{v \text{ сомножителей}}$. Тогда, учитывая, что для любого $a \neq 0$ существует логарифм $u = \log_{\alpha} a$, имеем: $a^{v} = \underbrace{\alpha^{u} \cdot \ldots \cdot \alpha^{u}}_{v \text{ сомножителей}} = \underbrace{\alpha \cdot \ldots \cdot \alpha}_{(u \cdot v) \text{сомножителей}}$.

Заметим, что если в получившемся произведении, общее количество сомножителей α больше или равно $p-1$, то можно выделить одну или несколько групп ровно по $p-1$, и одну «остаточную» группу из $(u \cdot v) \bmod (p-1)$ сомножителей. Заметим, что группа из ровно $p-1$ сомножителей, являющихся примитивным элементом α, равна 1 по определению: $\alpha^{p-1} = 1$. Тогда имеем: $a^{v} = \underbrace{\alpha \cdot \ldots \cdot \alpha}_{(u \cdot v) \bmod (p-1) \text{сомножителей}} = \alpha^{w}$, где $\underbrace{w = (u \cdot v) \bmod (p-1)}_{< R, \{+,\cdot\} >}$.

Особо отметим, что операции $(u \cdot v) \bmod (p-1)$ для логарифмов, представленных в поле действительных чисел, соответствует операция умножения в кольце логарифмов *LR*(*p* − 1).

Таким образом, возведение ненулевого элемента a простого поля $GF(p)$, $p \geq 3$, в степень v, являющуюся элементом кольца логарифмов $LR(p-1)$, математически можно записать: $\forall p \in P : p \geq 3 \;\&\; \forall a \in GF(p) : (a \neq 0) \;\&\; \forall u, v \in LR(p-1) : u = \log_{\alpha} a, \exists w \in LR(p-1) :$

$\underbrace{w = u \cdot v}_{LR(p-1)} \Leftrightarrow \underbrace{w = (u \cdot v) \bmod (p-1)}_{< R, \{+,\cdot\} >} \Leftrightarrow \underbrace{\alpha^w = a^v}_{GF(p)}$. В случае если элемент $a = 0$, то формулу применять нельзя, однако, мы помним, что в силу свойств самого поля возведение нулевого элемента в целую положительную степень, дает снова нуль $0^v = 0, v > 0$, а $0^0 = 1$.

Отметим, что при выполнении операций умножения, вычисления обратного элемента, деления элементов простого поля $GF(p)$ с помощью соответствующих логарифмов, принадлежащих кольцу $LR(p-1$, требуется дополнительно вычислять логарифмы элементов поля и возводить примитивный элемент поля в требуемую степень, также принадлежащей кольцу $LR(p-1$. Чтобы вычисление логарифмов и возведение примитивного элемента в требуемую степень были наименее трудоемкими с вычислительной точки зрения, лучше всего хранить «логарифмы» элементов простого поля $GF(p)$ по основанию примитивного элемента α и «степени» примитивного элемента α в виде двух таблиц.

Пример. Рассмотрим операции умножения, вычисления обратного элемента по умножению и деления при помощи логарифмов на примере простого поля $GF(5)$. Имеем следующие таблицы сложения и умножения элементов этого поля:

$GF(5)$:

+	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

·	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

В качестве примитивного элемента выберем $\alpha = 2$, с помощью которого можно породить все ненулевые элементы простого поля. Тогда, используя таблицу умножения, нетрудно сформировать таблицу u-х степеней, $u \in \{0,1,2,3\}$, примитивного элемента $\alpha = 2$, а с помощью нее уже, в свою очередь, сформировать таблицу логарифмов по основанию примитивного элемента $\alpha = 2$ для всех ненулевых элементов поля $a \in \{1,2,3,4\}$:

$GF(5)$:

u	0	1	2	3
α^u	1	2	4	3

a	1	2	3	4
$\log_{\alpha} a$	0	1	3	2

Тогда, используя две таблицы, вычислим для примера произведение элементов $a = 3$ и $b = 4$, используя их логарифмы: $u = \log_{\alpha} a = \log_2 3 = 3$ и $v = \log_{\alpha} b = \log_2 4 = 2$, которые легко получаем из «таблицы логарифмов» для простого поля $GF(5)$. Тогда, $a \cdot b = \alpha^w \Rightarrow 3 \cdot 4 = 2^w$, где $\underbrace{w = (u+v) \bmod (p-1) = (3+2) \bmod 4 = 5 \bmod 4 = 1}_{< R, \{+,\cdot\} >}$. Тогда, используя «таблицу степеней», легко получаем $2^1 = 2$. Таким образом, $3 \cdot 4 = 2$. По таблице умножения элементов поля $GF(5)$ нетрудно заметить, что результат верный.

Вычислим также для примера обратный элемент по умножения для элемента $a = 2.$ Логарифм элемента a: $u = \log_2 2 = 1.$ Тогда, обратный элемент по умножению $a^{-1} = \alpha^v$, где $\underbrace{v = ((p-1) - u) \bmod (p-1) = (4-1) \bmod 4 = 3}_{< R, \{+,\cdot\} >}$. Тогда окончательно получаем $a^{-1} = 2^3 = 3$. По таблице умножения нетрудно заметить, что $a \cdot a^{-1} = 2 \cdot 3 = 1$.

Многочлены над простым полем Галуа. Пусть задано простое число p и целое число $m \geq 1$. Пусть задано простое поле Галуа *GF(p)*. Тогда, **многочленом над полем *GF(p)*** называется математическое выражение $a(x) = a_m \cdot x^m + \ldots + a_1 \cdot x + a_0$, где x – некоторая формальная переменная, а коэффициенты $a_m, \ldots, a_1, a_0$ являются элементами простого поля *GF(p)*. Многочлен $a(x) = 0$ называется **нулевым**. Многочлен, у которого старший коэффициент $a_m = 1$, называется **нормированным (приведенным)**.

Определение 1. **Степенью многочлена** $a(x)$ называется наивысший показатель степени переменной среди ненулевых слагаемых многочлена. Степень многочлена $a(x)$ обозначается, как $\deg(a(x))$. Например, степень многочлена $a(x) = x^2 + x + 1$, равна 2. Степень нулевого многочлена $a(x) = 0$ принимается за минус бесконечность: $\deg(0) = -\infty$.

Определение 2. **Суммой двух многочленов** $a(x)$ степени u и $b(x)$ степени v, заданных над полем *GF(p)*, называется многочлен $c(x) = \sum_{i=0}^{\max\{u,v\}} c_i \cdot x^i$, где коэффициенты $\overbrace{c_i = a_i + b_i}^{GF(p)}$ вычисляются при помощи операций сложения простого поля *GF(p)*. Степень многочлена $c(x)$ меньше либо равна наибольшей из степеней многочленов $a(x)$ и $b(x)$.

Определение 3. **Произведением двух многочленов** $a(x)$ степени u и $b(x)$ степени v, заданных над полем *GF(p)*, называется многочлен $c(x) = \sum_{k=0}^{u+v} c_k \cdot x^k$, где коэффициенты $\overbrace{c_k = \sum_{\substack{i+j=k \\ i=0\ldots u \quad j=0\ldots v}} \left(a_i \cdot b_j \right)}^{GF(p)}$ вычисляются при помощи операций сложения и умножения, заданных для простого поля *GF(p)*. Степень результирующего многочлена $c(x)$ равна сумме степеней многочленов $a(x)$ и $b(x)$.

Определение 4. **Умножением многочлена** $a(x)$ степени u **на скаляр** λ, являющийся элементом простого поля Галуа *GF(p)*, называется многочлен $(\overbrace{\lambda \cdot a_m}^{GF(p)}) \cdot x^m + \ldots + (\overbrace{\lambda \cdot a_1}^{GF(p)}) \cdot x + (\overbrace{\lambda \cdot a_0}^{GF(p)})$, получающийся путем умножения всех коэффициентов многочлена $a(x)$ на заданный скаляр $\lambda \in GF(p)$ с использованием операции умножения для простого поля *GF(p)*.

Аналогично можно определить также деление многочлена $a(x)$ на ненулевой скаляр $\lambda \in GF(p)$, которое сводится к умножению многочлена на скаляр $\mu \in GF(p)$, являющийся обратным элементом по умножению для элемента λ в простом поле $GF(p)$.

Определение 5. Пусть над простым полем *GF(p)* заданы некоторый многочлен $a(x)$ и некоторый ненулевой многочлен $g(x) \neq 0$ степени $m \geq 0$. Тогда существуют многочлены $q(x)$ и $r(x)$ над простым полем *GF(p)* такие, что $a(x) = q(x) \cdot g(x) + r(x)$, причем $\deg(r(x)) < \deg(g(x))$. Тогда, многочлен $q(x)$ называют **частным от деления**, и обозначают как $a(x)$ **div** $g(x)$, а многочлен $r(x)$ называют **остатком от деления** многочлена $a(x)$ на многочлен $g(x)$, и обозначают как $a(x)$ **mod** $g(x)$. Многочлен $r(x)$ также принято называть **остатком** многочлена $a(x)$ **по модулю** многочлена $g(x)$.

Примечание. Если многочлен $g(x)$ имеет нулевую степень, $\deg(g(x))=0$, иными словами $g(x)=g_0$, то многочлен $a(x)$ без остатка делится на многочлен $g(x)$, причем частное от деления $q(x)=a(x)\cdot g_0^{-1}$, где $g_0^{-1}\in GF(p)$ является обратным элементом по умножению для элемента g_0 в простом поле $GF(p)$, а остаток от деления $r(x)=0$.

Рассмотрим теперь некоторые свойства остатков по модулю для многочленов.

Свойство 1. Если степень многочлена $a(x)$ меньше степени многочлена $g(x)$, иными словами $\deg(a(x))<\deg(g(x))$, в том числе, если $a(x)=0$, то остатком многочлена $a(x)$ по модулю многочлена $g(x)$ является сам многочлен $a(x)$, иными словами, $\forall a(x):\deg(a(x))<\deg(g(x))\Rightarrow a(x)\bmod g(x)=a(x)$. В справедливости свойства нетрудно убедиться. При $\deg(a(x))<\deg(g(x))$ уравнение $a(x)=q(x)\cdot g(x)+r(x)$ имеет решение $q(x)=0$ и $r(x)=a(x)$. Таким образом, если $\deg(a(x))<\deg(g(x))$, то остаток многочлена $a(x)$ по модулю многочлена $g(x)$ равен самому многочлену $a(x)$, то есть $r(x)=a(x)$.

Свойство 2. Остаток суммы (разности) двух многочленов $a(x)$ и $b(x)$ по модулю многочлена $g(x)$ равен сумме (разности) остатков многочленов $a(x)$ и $b(x)$ по модулю многочлена $g(x)$. Иными словами: $(a(x)\pm b(x))\bmod g(x)=a(x)\bmod g(x)\pm b(x)\bmod g(x)$.

В справедливости свойства нетрудно убедиться. Поскольку $a(x)=q_a(x)\cdot g(x)+r_a(x)$ и $b(x)=q_b(x)\cdot g(x)+r_b(x)$, то $a(x)\pm b(x)=(q_a(x)\pm q_b(x))\cdot g(x)+(r_a(x)\pm r_b(x))$. Заметим, что $\deg(r_a(x)\pm r_b(x))<\deg(g(x))$, так как $\deg(r_a(x))<\deg(g(x))$ и $\deg(r_b(x))<\deg(g(x))$, а $\deg(r_a(x)\pm r_b(x))\le\max\{\deg(r_a(x)),\deg(r_b(x))\}$. Тогда, очевидно, $r_a(x)\pm r_b(x)$ и является искомым остатком $(a(x)\pm b(x))$ по модулю многочлена $g(x)$.

Свойство 3. Остаток произведения двух многочленов $a(x)$ и $b(x)$ по модулю многочлена $g(x)$ равен остатку произведения остатков многочленов $a(x)$ и $b(x)$ по модулю $g(x)$. Иными словами: $(a(x)\cdot b(x))\bmod g(x)=(a(x)\bmod g(x))\cdot(b(x)\bmod g(x))\bmod g(x)$.

В справедливости свойства нетрудно убедиться. Поскольку $a(x)=q_a(x)\cdot g(x)+r_a(x)$ и $b(x)=q_b(x)\cdot g(x)+r_b(x)$, то произведение многочленов $a(x)$ и $b(x)$ дает следующее: $a(x)\cdot b(x)=q_a(x)\cdot q_b(x)\cdot g(x)\cdot g(x)+q_a(x)\cdot r_b(x)\cdot g(x)+q_b(x)\cdot r_a(x)\cdot g(x)+(r_a(x)\cdot r_b(x))=$ $=(q_a(x)\cdot q_b(x)\cdot g(x)+q_a(x)\cdot r_b(x)+q_b(x)\cdot r_a(x))\cdot g(x)+(r_a(x)\cdot r_b(x))$. Теперь заметим, что степень произведения остатков $r_a(x)\cdot r_b(x)$ может быть как меньше, так и больше либо равна степени $g(x)$. В то же время само произведение $r_a(x)\cdot r_b(x)$ мы также можем представить в виде $r_a(x)\cdot r_b(x)=q_{ab}(x)\cdot g(x)+r_{ab}(x)$, где $r_{ab}(x)=(r_a(x)\cdot r_b(x))\bmod g(x)$, тогда $a(x)\cdot b(x)=(q_a(x)\cdot q_b(x)\cdot g(x)+q_a(x)\cdot r_b(x)+q_b(x)\cdot r_a(x)+q_{ab}(x))\cdot g(x)+r_{ab}(x)$. Тогда, поскольку $\deg(r_{ab}(x))<\deg(g(x))$, то остаток $r_{ab}(x)=(r_a(x)\cdot r_b(x))\bmod g(x)$ и является искомым остатком $(a(x)\cdot b(x))$ по модулю многочлена $g(x)$.

Свойство 4. Остаток многочлена $a(x)$, умноженного на скаляр λ, по модулю многочлена $g(x)$ равен остатку по модулю многочлена $g(x)$, умноженному на скаляр λ. Иными словами, $(a(x)\cdot\lambda)\bmod g(x)=(a(x)\bmod g(x))\cdot\lambda$. В справедливости свойства нетрудно убедиться: $a(x)=q_a(x)\cdot g(x)+r_a(x)\Rightarrow\lambda\cdot a(x)=(\lambda\cdot q_a(x))\cdot g(x)+\lambda\cdot r_a(x)$. Так как $\deg(\lambda\cdot r_a(x))=\deg(r_a(x))<\deg(g(x))$, то $\lambda\cdot r_a(x)$ и является искомым остатком $(a(x)\cdot\lambda)$ по модулю многочлена $g(x)$.

Следует отметить, что хотя деление многочленов, заданных над простым полем *GF(p)*, во многом похоже на деление многочленов, заданных над полем действительных чисел, однако, здесь имеется один существенный момент. При делении многочленов «в столбик» используются операции умножения многочлена-делителя на одночлен (в том числе на одночлен нулевой степени, то есть скаляр), а также операция вычитания многочленов. Эти операции, в свою очередь, приводят к операциям умножения и вычитания коэффициентов, которые в рассматриваемом нами случае по определению являются элементами простого поля *GF(p)*. Соответственно, все операции с коэффициентами должны выполняться по правилам арифметики простого поля *GF(p)*, которые мы подробно рассматривали выше.

Пример 1. Выполним деление многочлена $a(x) = x^4 + 2 \cdot x^3 + 3 \cdot x^2 + 4 \cdot x + 1$ на многочлен $g(x) = x^3 + 2 \cdot x + 1$, заданных над простым полем *GF*(5):

$$\begin{array}{r|l} 1 \cdot x^4 + 2 \cdot x^3 + 3 \cdot x^2 + 4 \cdot x + 1 & g(x) = x^3 + 2 \cdot x + 1 \\ \underline{1 \cdot x^4 + 0 \cdot x^3 + 2 \cdot x^2 + 1 \cdot x + 0} & \underline{\quad\quad\quad\quad\quad\quad} \\ 2 \cdot x^3 + 1 \cdot x^2 + 3 \cdot x + 1 & q(x) = x + 2 \\ \underline{2 \cdot x^3 + 0 \cdot x^2 + 4 \cdot x + 2} & \\ r(x) = x^2 + 4 \cdot x + 4 & \end{array}$$

Таким образом, частное от деления: $q(x) = x + 2$, а остаток: $r(x) = x^2 + 4 \cdot x + 4$.

Пример 2. Выполним деление многочлена $a(x) = x^5 + x^4 + x^2 + 1$ на многочлен $g(x) = x^3 + 1$, заданных над простым полем *GF*(2):

$$\begin{array}{r|l} 1 \cdot x^5 + 1 \cdot x^4 + 0 \cdot x^3 + 1 \cdot x^2 + 0 \cdot x + 1 & g(x) = x^3 + 1 \\ \underline{1 \cdot x^5 + 0 \cdot x^4 + 0 \cdot x^3 + 1 \cdot x^2 + 0 \cdot x + 0} & \underline{\quad\quad\quad\quad\quad\quad} \\ 1 \cdot x^4 + 0 \cdot x^3 + 0 \cdot x^2 + 0 \cdot x + 1 & q(x) = x^2 + x \\ \underline{1 \cdot x^4 + 0 \cdot x^3 + 0 \cdot x^2 + 1 \cdot x + 0} & \\ r(x) = x + 1 & \end{array}$$

Таким образом, частное от деления: $q(x) = x^2 + x$, а остаток: $r(x) = x + 1$.

Примечание 1. Отметим, что если многочлен $g(x)$ имеет нулевую степень $m = 0$, то есть является скаляром g_0, то, уравнение $a(x) = q(x) \cdot g(x) + r(x)$ имеет решение при остатке $r(x) = 0$, при этом степень нулевого многочлена-остатка считается минус бесконечностью, и частном $q(x) = a(x) / g_0$, и деление многочленов «в столбик» в этом случае излишне.

Примечание 2. Отметим, что деление выполнять удобнее, когда многочлен $g(x)$ степени m является нормированным, то есть $g_m = 1$. Если многочлен $g(x)$ не нормирован, $g_m > 1$, его всегда можно привести к нормированному виду, разделив все его коэффициенты на коэффициент g_m. При этом следует учесть, что поскольку $a(x) = q(x) \cdot g(x) + r(x)$, то $a(x) = (q(x) \cdot g_m) \cdot (g(x) / g_m) + r(x)$. Таким образом, при делении на нормированный многочлен $g(x) / g_m$, мы получаем тот же самый остаток $r(x)$, а частное равно $q(x) \cdot g_m$, и его необходимо разделить на скаляр g_m, чтобы получить $q(x)$.

Теперь формализуем вычисление частного и остатка многочлена $a(x)$ по модулю нормированного многочлена $g(x)$ степени $m \geq 1$ в виде некоторой итерационной схемы вычисления с использованием рекуррентных соотношений.

В качестве «начального» остатка принимается $r^{(0)}(x) = a(x)$. Далее на каждой итерации, начиная с итерации $s = 1$, проверяется степень остатка «предыдущей» итерации, $\mu = \deg(r^{(s-1)}(x))$, и если она больше либо равна степени многочлена $g(x)$, то из остатка $r^{(s-1)}(x)$ вычитается многочлен $g(x)$, умноженный на одночлен $x^{(\mu-m)}$ и на старший коэффициент $r_{\mu}^{(s-1)}$ остатка $r^{(s-1)}(x)$, таким образом вычисляется новый остаток $r^{(s)}(x)$ на текущей итерации *s*. Процедура продолжается до тех пор, пока на некоторой итерации $s = s*$, степень остатка предыдущей итерации не окажется меньше степени многочлена $g(x)$, то есть $\deg(r^{(s*-1)}(x)) < \deg(g(x))$. Тогда, остаток $r^{(s*-1)}(x)$ принимается в качестве остатка $r(x)$ многочлена $a(x)$ по модулю нормированного многочлена $g(x)$.

Если помимо остатка также требуется найти частное $q(x)$ от деления, то его также нетрудно сформировать. В качестве начального частного принимается $q^{(0)}(x) = 0$. Далее на каждой итерации, начиная с итерации $s = 1$, «очередное» частное вычисляется путем добавления к нему одночлена $x^{(\mu-m)}$, умноженного на старший коэффициент $r_{\mu}^{(s-1)}$ остатка $r^{(s-1)}(x)$: $q^{(s)}(x) = q^{(s-1)}(x) + r_{\mu}^{(s-1)} \cdot x^{(\mu-m)}$. Тогда, частное $q^{(s*-1)}(x)$ принимается в качестве частного $q(x)$ от деления многочлена $a(x)$ на многочлен $g(x)$.

Тогда с учетом всего сказанного итерационная схема вычисления частного и остатка многочлена $a(x)$ по модулю нормированного многочлена $g(x)$ выглядит так:

$$
\begin{gathered}
a(x) \bmod g(x) = r(x) = r^{(s*-1)}(x); \quad q(x) = q^{(s*-1)}(x); \\
\begin{cases}
s = 1 \ldots s* : \left(\deg(r^{(s*-1)}(x)) < m \right); \quad r^{(0)}(x) = a(x); \quad q^{(0)}(x) = 0; \\
r^{(s)}(x) = \overbrace{r^{(s-1)}(x) - r_{\mu}^{(s-1)} \cdot g(x) \cdot x^{(\mu-m)}}^{GF(p)}; \quad \mu = \deg(r^{(s-1)}(x)) \\
q^{(s)}(x) = \overbrace{q^{(s-1)}(x) + r_{\mu}^{(s-1)} \cdot x^{(\mu-m)}}^{GF(p)}; \\
m = \deg(g(x)) \geq 1; \quad g_m = 1
\end{cases}
\end{gathered}
\qquad (1.4.1)
$$

Отметим, что в худшем случае на каждой итерации степень остатка $r^{(s)}(x)$ меньше степени остатка $r^{(s-1)}(x)$ на единицу, как минимум, за счет того, что при выполнении операции вычитания коэффициенты при x^{μ} всегда взаимно уничтожаются, поскольку $g_m = 1$. Тогда, в худшем случае для вычисления остатка многочлена $a(x)$ по модулю нормированного многочлена $g(x)$ потребуется $\deg(a(x)) - \deg(g(x)) + 1$ итераций.

Следует отметить, что приведенная выше итерационная процедура вычисления остатка $r(x) = a(x) \bmod g(x)$, а также частного $q(x)$, является достаточно наглядной и тривиальной, но не единственной. Основным ее недостатком является то, что на каждой итерации приходится прибегать к умножению на одночлен $x^{(\mu-m)}$, степень которого от итерации к итерации меняется, что вносит дополнительную сложность при программной или аппаратной реализации итерационной схемы. Поэтому мы выведем альтернативную «сдвиговую» итерационную схему вычисления остатка $r(x)$ и частного $q(x)$.

Пусть задан многочлен $a(x) = a_{u-1} \cdot x^{u-1} + \ldots + a_1 \cdot x + a_0$ степени $u-1$.

Многочлен $a(x)$ мы всегда можем привести к следующему «рекуррентному» виду: $a(x) = (\ldots((0 + a_{u-1}) \cdot x + a_{u-2}) \cdot x + \ldots + a_1) \cdot x + a_0$. Теперь учтем свойства остатков суммы и произведения многочленов: $(a(x) + b(x)) \bmod g(x) = a(x) \bmod g(x) + b(x) \bmod g(x)$ и $(a(x) \cdot b(x)) \bmod g(x) = (a(x) \bmod g(x)) \cdot (b(x) \bmod g(x)) \bmod g(x)$, а также свойство остатка многочлена, умноженного на скаляр: $(\lambda \cdot a(x)) \bmod g(x) = \lambda \cdot (a(x) \bmod g(x))$.

Тогда, мы можем записать $a(x) \bmod g(x) = (a_{u-1} \cdot x^{u-1} + \ldots + a_1 \cdot x + a_0) \bmod g(x) =$

$= ((\ldots((((0 \bmod g(x) + a_{u-1}) \cdot x) \bmod g(x) + a_{u-2}) \cdot x) \bmod g(x) + \ldots + a_1) \cdot x) \bmod g(x) + a_0$.

Тогда, при таком рекуррентном представлении формулы для вычисления $a(x) \bmod g(x)$ мы принимаем в качестве начального остатка $r^{(0)}(x) = 0$. Далее на каждой итерации, начиная с итерации $s = 1$, мы вычисляем очередной остаток следующим образом: $\tilde{r}^{(s)}(x) = (x \cdot \tilde{r}^{(s-1)}(x)) \bmod g(x) + a_{u-s}$. Тогда, на итерации $s = u$, мы получим искомый остаток $r(x) = \tilde{r}^{(u)}(x)$. Теперь отметим, что, степень остатка $\tilde{r}^{(s)}(x)$ всегда $\le m-1$, поскольку он является суммой многочлена-остатка $(x \cdot \tilde{r}^{(s-1)}(x)) \bmod g(x)$, степень которого $\le m-1$ и скаляра a_{u-s} (многочлена нулевой степени). Тогда, многочлен $x \cdot \tilde{r}^{(s-1)}(x) = \tilde{r}_{m-1}^{(s-1)} \cdot x^m + \ldots + \tilde{r}_0^{(s-1)} \cdot x$ в худшем случае имеет степень *m*, и, вычисление остатка $(x \cdot \tilde{r}^{(s-1)}(x)) \bmod g(x)$ сводится к вычислению $x \cdot \tilde{r}^{(s-1)}(x) - \tilde{r}_{m-1}^{(s-1)} \cdot g(x)$, поскольку $g(x) = x^{m-1} + g_{m-1} \cdot x^{m-1} + \ldots + g_1 \cdot x + g_0$.

Если помимо остатка также требуется найти частное $q(x)$ от деления, то его также нетрудно сформировать. В качестве начального частного принимается $\tilde{q}^{(0)}(x) = 0$. Далее на каждой итерации, начиная с итерации $s = 1$, «очередное» частное также вычисляется рекуррентным образом: $\tilde{q}^{(s)}(x) = x \cdot \tilde{q}^{(s-1)}(x) + \tilde{r}_{m-1}^{(s-1)}$.

Тогда, с учетом всего сказанного получаем следующую итерационную схему:

$$\begin{cases} a(x) \bmod g(x) = r(x) = \tilde{r}^{(u)}(x); \quad q(x) = \tilde{q}^{(u)}(x) \\ \quad s = 1 \ldots u; \quad \tilde{r}^{(0)}(x) = 0; \quad \tilde{q}^{(0)}(x) = 0; \\ \tilde{r}^{(s)}(x) = \overbrace{x \cdot \tilde{r}^{(s-1)}(x) - \tilde{r}_{m-1}^{(s-1)} \cdot g(x) + a_{u-s}}^{GF(p)} \\ \tilde{q}^{(s)}(x) = \overbrace{x \cdot \tilde{q}^{(s-1)}(x) + \tilde{r}_{m-1}^{(s-1)}}^{GF(p)} \\ u = \deg(a(x)) + 1; \quad m = \deg(g(x)) \ge 1; \quad g_m = 1 \end{cases} \qquad (1.4.2)$$

Теперь отметим, что в случае, когда степень $u-1$ многочлена $a(x)$ меньше либо равна $m-1$, очевидно, остатком многочлена $a(x)$ по модулю многочлена $g(x)$ степени *m* является сам многочлен $a(x)$, иными словами $\deg(a(x)) < \deg(g(x)) \Rightarrow a(x) \bmod g(x) = a(x)$, и нет необходимости в итерационном вычислении остатка. Что же касается частного $q(x)$, то оно, очевидно, в этом случае равно нулю, иными словами $\deg(a(x)) < \deg(g(x)) \Rightarrow q(x) = 0$.

В случае же $u-1>m-1\Rightarrow u>m$, то, как минимум, на первых m итерациях $s=1\ldots m$ степень многочлена-остатка $\tilde{r}^{(s)}(x)$ меньше либо равна $m-1$, так как $\tilde{r}^{(1)}(x)=a_{u-1}$, $\tilde{r}^{(2)}(x)=a_{u-1}\cdot x+a_{u-2}$, и так далее, $\tilde{r}^{(m)}(x)=a_{u-1}\cdot x^{m-1}+\ldots+a_{u-m+1}\cdot x+a_{u-m}$, и, только начиная с итерации $s=m+1$, коэффициент $\tilde{r}_{m-1}^{(s-1)}$ может оказаться отличным от нуля, и, соответственно, $\tilde{r}_{m-1}^{(s-1)}\cdot g(x)$ будет участвовать в вычислении очередного остатка. Аналогично, по этой же причине, на первых m итерациях $s=1\ldots m$ частное $\tilde{q}^{(s)}(x)=0$. Тогда, очевидно, если принять остаток $\tilde{r}^{(m)}(x)=a_{u-1}\cdot x^{m-1}+\ldots+a_{u-m+1}\cdot x+a_{u-m}$ и частное $\tilde{q}^{(m)}(x)=0$, то мы можем начать вычисления сразу с итерации $s=m+1$, и за оставшиеся $u-m$ итераций вычислить остаток $\tilde{r}^{(u)}(x)$ и частное $\tilde{q}^{(u)}(x)$, и, тем самым, сократить количество итераций на m. Заметим, что, в этой ситуации удобнее «сместить на m» нумерацию итераций: $s=m+1\ldots u\rightarrow s=1\ldots u-m$, при этом учтем, что $\tilde{r}^{(m)}(x)\rightarrow\tilde{r}^{(0)}(x)$, $\tilde{r}^{(u)}(x)\rightarrow\tilde{r}^{(u-m)}(x)$, $\tilde{q}^{(m)}(x)\rightarrow\tilde{q}^{(0)}(x)$, $\tilde{q}^{(u)}(x)\rightarrow\tilde{q}^{(u-m)}(x)$ и $a_{u-s}\rightarrow a_{u-m-s}$.

Тогда, наконец, получаем «оптимизированную» итерационную схему вычисления частного и остатка многочлена $a(x)$ по модулю нормированного многочлена $g(x)$:

$$a(x)\bmod(g(x))=r(x);\quad u=\deg(a(x))+1;\quad m=\deg(g(x))\geq 1;\quad g_m=1$$

$$\begin{pmatrix}r(x)\\q(x)\end{pmatrix}=\begin{cases}\begin{pmatrix}a(x)\\0\end{pmatrix}; & u\leq m\\ \begin{pmatrix}\tilde{r}^{(u-m)}(x)\\\tilde{q}^{(u-m)}(x)\end{pmatrix}; & u>m;\end{cases}\quad\begin{cases}\tilde{r}^{(0)}(x)=\sum\limits_{j=0}^{m-1}a_{u-m+j}\cdot x^j;\quad \tilde{q}^{(0)}(x)=0\\ s=1\ldots u-m\\ \tilde{r}^{(s)}(x)=\overbrace{x\cdot\tilde{r}^{(s-1)}(x)-\tilde{r}_{m-1}^{(s-1)}\cdot g(x)+a_{u-m-s}}^{GF(p)}\\ \tilde{q}^{(s)}(x)=\overbrace{x\cdot\tilde{q}^{(s-1)}(x)+\tilde{r}_{m-1}^{(s-1)}}^{GF(p)}\end{cases}\qquad(1.4.3)$$

Пример 1. Найдем частное и остаток многочлена $a(x)=x^4+2\cdot x^3+3\cdot x^2+4\cdot x+1$ по модулю нормированного многочлена $g(x)=x^3+2\cdot x+1$, заданных над полем $GF(5)$.

Поскольку $u=\deg(a(x))+1=5>m=\deg(g(x))=3$, то в качестве начального остатка принимаем: $\tilde{r}^{(0)}(x)=a_4\cdot x^2+a_3\cdot x+a_2=x^2+2\cdot x+3$. Начальное частное: $\tilde{q}^{(0)}(x)=0$.

На итерации $s=1$ вычисляем остаток и частное:

$$\overbrace{\tilde{r}^{(1)}(x)=x\cdot\tilde{r}^{(0)}(x)-\tilde{r}_2^{(0)}\cdot g(x)+a_1=x\cdot(x^2+2\cdot x+3)-1\cdot(x^3+2\cdot x+1)+4=2\cdot x^2+x+3}^{GF(5)}$$

$$\tilde{q}^{(1)}(x)=x\cdot\tilde{q}^{(0)}(x)+\tilde{r}_2^{(0)}=1$$

На итерации $s = 2$ вычисляем остаток и частное:

$$\overbrace{\tilde{r}^{(2)}(x) = x \cdot \tilde{r}^{(1)}(x) - \tilde{r}_2^{(1)} \cdot g(x) + a_0 = x \cdot (2 \cdot x^2 + x + 3) - 2 \cdot (x^3 + 2 \cdot x + 1) + 1 = x^2 + 4 \cdot x + 4}^{GF(5)}$$

$$\tilde{q}^{(2)}(x) = x \cdot \tilde{q}^{(1)}(x) + \tilde{r}_2^{(1)} = x \cdot 1 + 2 = x + 2$$

Поскольку мы достигли итерации $s = u - m = 2$, то многочлен $\tilde{q}^{(2)}(x) = x + 2$ принимается в качестве частного, а многочлен $\tilde{r}^{(2)}(x) = x^2 + 4 \cdot x + 4$ в качестве остатка многочлена $a(x) = x^4 + 2 \cdot x^3 + 3 \cdot x^2 + 4 \cdot x + 1$ по модулю многочлена $g(x) = x^3 + 2 \cdot x + 1$.

Пример 2. Найдем частное и остаток многочлена $a(x) = x^5 + x^4 + x^2 + 1$ по модулю нормированного многочлена $g(x) = x^3 + 1$, заданных над полем *GF*(2).

Поскольку $u = \deg(a(x)) + 1 = 6 > m = \deg(g(x)) = 3$, то в качестве начального остатка принимаем: $\tilde{r}^{(0)}(x) = a_5 \cdot x^2 + a_4 \cdot x + a_3 = x^2 + x$. Начальное частное: $\tilde{q}^{(0)}(x) = 0$.

На итерации $s = 1$ вычисляем остаток и частное:

$$\overbrace{\tilde{r}^{(1)}(x) = x \cdot \tilde{r}^{(0)}(x) - \tilde{r}_2^{(0)} \cdot g(x) + a_2 = x \cdot (x^2 + x) - 1 \cdot (x^3 + 1) + 1 = x^2}^{GF(2)}$$

$$\tilde{q}^{(1)}(x) = x \cdot \tilde{q}^{(0)}(x) + \tilde{r}_2^{(0)} = 1$$

На итерации $s = 2$ вычисляем остаток и частное:

$$\overbrace{\tilde{r}^{(2)}(x) = x \cdot \tilde{r}^{(1)}(x) - \tilde{r}_2^{(1)} \cdot g(x) + a_1 = x \cdot (x^2) - 1 \cdot (x^3 + 1) + 0 = 1}^{GF(2)}$$

$$\tilde{q}^{(2)}(x) = x \cdot \tilde{q}^{(1)}(x) + \tilde{r}_2^{(1)} = x \cdot 1 + 1 = x + 1$$

На итерации $s = 3$ вычисляем остаток и частное:

$$\overbrace{\tilde{r}^{(3)}(x) = x \cdot \tilde{r}^{(2)}(x) - \tilde{r}_2^{(2)} \cdot g(x) + a_0 = x \cdot (1) - 0 \cdot (x^3 + 1) + 1 = x + 1}^{GF(2)}$$

$$\tilde{q}^{(3)}(x) = x \cdot \tilde{q}^{(2)}(x) + \tilde{r}_2^{(2)} = x \cdot (x + 1) + 0 = x^2 + x$$

Поскольку мы достигли итерации $s = u - m = 3$, то многочлен $\tilde{q}^{(3)}(x) = x^2 + x$ принимается в качестве частного, а многочлен $\tilde{r}^{(3)}(x) = x + 1$ принимается в качестве остатка многочлена $a(x) = x^5 + x^4 + x^2 + 1$ по модулю $g(x) = x^3 + 1$.

Ниже на рисунке 1.2 приведена схема алгоритма вычисления частного и остатка от деления многочлена $a(x)$ на ненулевой многочлен $g(x) \neq 0$, заданных над простым полем *GF(p)*. Все операции с коэффициентами многочленов выполняются по правилам арифметики простого поля *GF(p)*. Отметим, что алгоритм учитывает все особые случаи, в том числе случай, когда многочлен $g(x) = 0$, в этом случае деление невозможно, и случай, когда многочлен $g(x)$ имеет нулевую степень, $g(x) = g_0$, в этом случае остаток равен нулю, а частное равно многочлену $a(x)$, деленному на скаляр g_0. Также в алгоритме учитывается случай ненормированного многочлена $g(x)$, $g_m > 1$, в этом случае деление осуществляется на нормированный многочлен $\tilde{g}(x) = g(x) / g_m$, а вычисленное частное также делится на g_m.

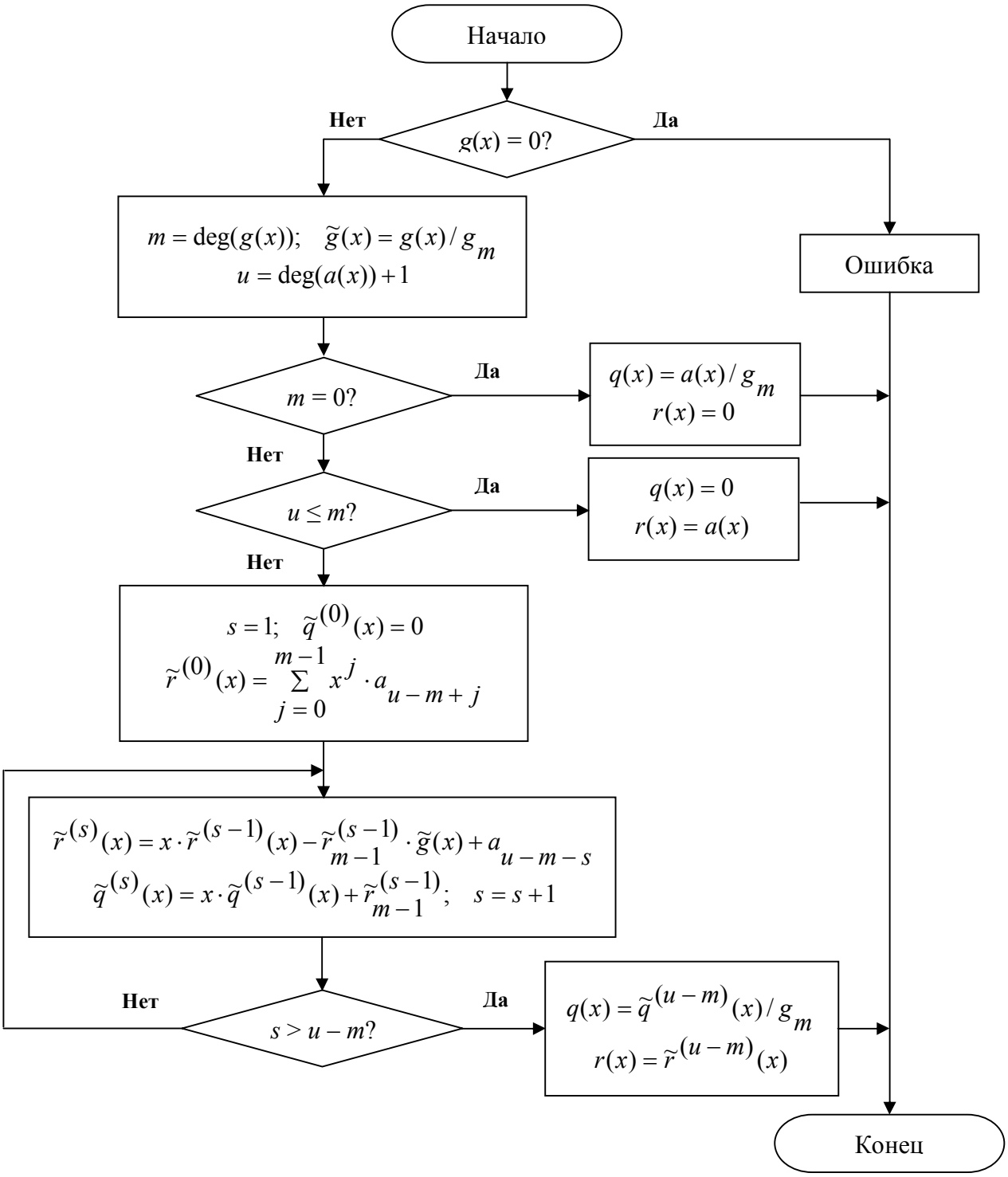

Рис. 1.2. Схема алгоритма «сдвигового» вычисления частного и остатка от деления многочлена $a(x)$ на многочлен $g(x)$, заданных над простым полем *GF(p)*.

Определение 6. Многочлен $p(x)$ степени $m \ge 2$, заданный над простым полем *GF(p)* такой, что его невозможно представить как произведение многочленов (разложить на многочлены-множители) ненулевой степени меньшей *m*, также являющихся многочленами над простым полем *GF(p)*, будем называть **неприводимым многочленом**.

Неприводимый многочлен $p(x)$ является аналогом простого числа *p*, и также как и простые числа, неприводимые многочлены для заданного простого поля *GF(p)* и степени *m* в общем случае можно найти только методом перебора. В то же время в общей алгебре имеется теорема, которая позволяет упростить нахождение, по крайней мере, некоторых нормированных (старший коэффициент равен 1) неприводимых многочленов.

Теорема 1. Двучлен $x^q - x$, где $q = p^m$, равен произведению всех нормированных неприводимых многочленов над простым полем *GF(p)*, степени которых делят *m*.

Следует особо отметить, что знак «–» перед *x* в двучлене $x^q - x$ следует понимать, как коэффициент $a_1 = -1$ при *x*, и поскольку этот коэффициент по определению должен являться элементом простого поля *GF(p)*, соответственно, под «–1» понимается обратный элемент по сложению для единичного элемента простого поля *GF(p)*. В различных простых полях обратные элементы по сложению для единичного элемента выглядят по-разному, для поля $GF(2): -1 = 1$, $GF(3): -1 = 2$, и, так далее, для поля $GF(p): -1 = p - 1$. Тогда, фактически двучлен $x^q - x$, следующим образом выглядит в поле $GF(2): x^q + x$, в поле $GF(3): x^q + 2 \cdot x$, и так далее, в поле $GF(p): x^q + (p-1) \cdot x$, где $q = p^m$.

Таким образом, нахождение нормированных неприводимых многочленов над простым полем *GF(p)*, степени которых делят *m*, сводится к задаче разложения двучлена $x^q + (p-1) \cdot x$, где $q = p^m$ на множители, являющиеся многочленами над полем *GF(p)*.

Рассмотрим примеры разложения двучлена $x^q + (p-1) \cdot x$ на нормированные неприводимые многочлены над простым полем *GF(p)*, для некоторых значений *p* и *m*:

- $p = 2, m = 2 : x^4 + x = x \cdot (x+1) \cdot (x^2 + x + 1)$.
- $p = 2, m = 3 : x^8 + x = x \cdot (x+1) \cdot (x^3 + x + 1) \cdot (x^3 + x^2 + 1)$.
- $p = 2, m = 4 : x^{16} + x = x \cdot (x+1) \cdot (x^2 + x + 1) \cdot (x^4 + x + 1) \cdot \ldots \cdot (x^4 + x^3 + x^2 + x + 1)$.
- $p = 3, m = 2 : x^9 + 2 \cdot x = x \cdot (x+1) \cdot (x+2) \cdot (x^2 + 1) \cdot (x^2 + x + 2) \cdot (x^2 + 2 \cdot x + 2)$.
- $p = 3, m = 3 : x^{27} + 2 \cdot x = x \cdot (x+1) \cdot (x+2) \cdot (x^3 + 2 \cdot x + 1) \cdot \ldots \cdot (x^3 + 2 \cdot x^2 + 2 \cdot x + 2)$.
- $p = 5, m = 2 : x^{25} + 4 \cdot x = x \cdot (x+1) \cdot (x+2) \cdot (x+3) \cdot (x+4) \cdot (x^2 + 2) \cdot \ldots \cdot (x^2 + 4 \cdot x + 2)$.

Согласно общей алгебре при помощи неприводимых многочленов *m*-й степени можно задать, так называемое, поле многочленов $GF(p^m)$, которое по сути является алгебраическим расширением простого поля *GF(p)*. В свою очередь, именно поля многочленов $GF(p^m)$ имеют большое прикладное значение в области кодирования информации. О них мы будем подробно говорить в следующем подразделе. Пока лишь отметим то, что в поле многочленов $GF(p^m)$, также как и в простом поле *GF(p)*, существуют примитивные элементы (многочлены) такие, что при помощи них можно получить все ненулевые многочлены поля $GF(p^m)$, путем возведения примитивного элемента во все степени $0 \ldots q-2$, где $q = p^m$. То, какие именно элементы (многочлены) будут являться примитивными в поле многочленов $GF(p^m)$, это напрямую зависит от неприводимого многочлена *m*-й степени, с помощью которого задается поле многочленов. В общей алгебре и теории кодирования особое значение имеют так называемые **примитивные неприводимые многочлены**, при помощи которых можно задать алгебраическое расширение простого поля *GF(p)* такое, что одночлен $\alpha(x) = x$ в поле многочленов $GF(p^m)$ будет являться примитивным элементом.

К сожалению, нахождение примитивных неприводимых многочленов, является еще более трудной задачей, чем нахождение самих неприводимых многочленов. В тоже время современное математическое программное обеспечение и вычислительные машины позволяют достаточно быстро находить примитивные неприводимые многочлены для относительно небольших значений *p* и *m* таких, что $q = p^m < 10^{100}$. Кроме того, ученые в области математики и теории кодирования регулярно составляют, обновляют и публикуют списки примитивных неприводимых многочленов в научных статьях и книгах [2, 10].

В приложении 2 приведено по одному примитивному неприводимому многочлену для некоторых простых полей *GF(p)* и некоторых степеней *m*. Наконец, отметим, что по умолчанию неприводимые многочлены принято задавать в нормированном виде.

Также особо следует отметить, что в общей алгебре имеется теорема, которая позволяет осуществлять проверку неприводимого многочлена на «примитивность».

Теорема 2. Неприводимый многочлен $p(x)$ степени *m*, заданный над простым полем *GF(p)*, примитивен тогда и только тогда, когда для любого простого делителя d числа $p^m - 1$ остаток одночлена $x^{((p^m-1)/d)}$ по модулю $p(x)$ не равен 1. Иными словами: $p(x) \in primitive \Leftrightarrow \left(\forall d \in P : ((p^m - 1) \bmod d = 0) \Rightarrow (x^{((p^m-1)/d)} \bmod p(x) \neq 1) \right)$.

Следствие. Если число $p^m - 1$ простое, то любой неприводимый многочлен $p(x)$ степени *m*, заданный над простым полем *GF(p)*, является примитивным.

Пример 1. Для примера протестируем на «примитивность» неприводимый многочлен $p(x) = x^4 + x + 1$, заданный над простым полем *GF*(2). Имеем $p = 2$ и $m = 4$, тогда $p^m - 1 = 15$. Простыми делителями числа 15 являются $d_1 = 3$ и $d_2 = 5$. Далее вычисляем остатки по модулю $p(x)$ от одночленов $x^{((p^m-1)/d_1)} = x^5$ и $x^{((p^m-1)/d_2)} = x^3$, получаем: $x^5 \bmod(x^4 + x + 1) = x^2 + x$ и $x^3 \bmod(x^4 + x + 1) = x^3$. Видим, что оба остатка не равны 1, а значит, неприводимый многочлен $p(x) = x^4 + x + 1$ является примитивным.

Если при тех же $p = 2$ и $m = 4$ протестировать неприводимый многочлен $p(x) = x^4 + x^3 + 1$, то получим следующие остатки: $x^5 \bmod(x^4 + x^3 + 1) = x^3 + x + 1$ и $x^3 \bmod(x^4 + x^3 + 1) = x^3$. Видим, что оба остатка не равны 1, а значит, неприводимый многочлен $p(x) = x^4 + x^3 + 1$ также является примитивным.

Если при тех же $p = 2$ и $m = 4$ протестировать неприводимый многочлен $p(x) = x^4 + x^3 + x^2 + x + 1$, то получим следующие остатки: $x^5 \bmod(x^4 + x^3 + x^2 + x + 1) = 1$ и $x^3 \bmod(x^4 + x^3 + x^2 + x + 1) = x^3$. Видим, что один из остатков равен 1, а значит, неприводимый многочлен $p(x) = x^4 + x^3 + x^2 + x + 1$ не является примитивным.

Пример 2. Протестируем на «примитивность» неприводимый многочлен $p(x) = x^2 + x + 2$, заданный над полем *GF*(3). Имеем $p = 3$ и $m = 2$, тогда $p^m - 1 = 8$. Простым делителем числа 8 является число $d_1 = 2$. Далее вычисляем остаток по модулю $p(x)$ от одночлена $x^{((p^m-1)/d_1)} = x^4$ и получаем $x^4 \bmod(x^2 + x + 2) = 2$. Видим, что остаток не равен 1, а значит, многочлен $p(x) = x^2 + x + 2$ является примитивным.

Если при тех же $p = 3$ и $m = 2$ протестировать неприводимый многочлен $p(x) = x^2 + 2x + 2$, то получим остаток $x^4 \bmod(x^2 + 2x + 2) = 2$, а значит, неприводимый многочлен $p(x) = x^2 + 2x + 2$ также является примитивным.

Если при тех же $p = 3$ и $m = 2$ протестировать неприводимый многочлен $p(x) = x^2 + 1$, то получим остаток $x^4 \bmod(x^2 + 1) = 1$, а значит, неприводимый многочлен $p(x) = x^2 + 1$ не является примитивным.

Определение 7. Пусть заданы два многочлена $a(x)$ степени u и $b(x)$ степени v, а также многочлен $g(x)$ степени $m \geq 2$ над полем *GF(p)*. Тогда **сверткой многочленов** $a(x)$ и $b(x)$ над полем *GF(p)* будем называть выражение вида $(a(x) \cdot b(x)) \bmod g(x)$. По сути свертка является остатком произведения многочленов $a(x)$ и $b(x)$ по модулю многочлена $g(x)$.

Примечание. Многочлен $g(x)$ должен иметь степень не ниже 2, поскольку при степени 1 свертки вырождаются в скаляры (многочлены нулевой степени), а при степени 0, свертка вообще всегда равна 0, и такие свертки не представляют особого интереса.

Теперь отметим, что различают несколько разновидностей сверток многочленов. Если степень многочлена $g(x)$ больше, чем сумма степеней многочленов $a(x)$ и $b(x)$, иными словами, если $\deg(g(x)) > \deg(a(x)) + \deg(b(x))$, то поскольку степень произведения многочленов равна $\deg(a(x)) + \deg(b(x))$ и она меньше степени многочлена $g(x)$, то, очевидно, остатком произведения $a(x) \cdot b(x)$ по модулю многочлена $g(x)$ будем само произведение $a(x) \cdot b(x)$. Такую свертку будем называть **линейной сверткой** (или **полной линейной сверткой**), и она по сути совпадает с произведением $a(x) \cdot b(x)$, и вычисляет как:

$$(a(x) \cdot b(x)) \bmod (g(x)) = a(x) \cdot b(x) = \sum_{k=0}^{u+v} x^k \cdot \left(\overbrace{\sum_{\substack{i+j=k \\ i=0\ldots u \quad j=0\ldots v}} \left(a_i \cdot b_j \right)}^{GF(p)} \right) \qquad (1.5.1)$$

$$u = \deg(a(x)); \quad v = \deg(b(x)); \quad \deg(g(x)) > \deg(a(x)) + \deg(b(x))$$

Особо отметим, что если мы будем подразумевать, что любые «отсутствующие» коэффициенты в многочлене, в том числе коэффициенты с индексами больше степени многочлена, равны нулю: $\forall i > \deg(a(x)) : a_i = 0$ и $\forall j > \deg(b(x)) : b_j = 0$, то мы внутреннее суммирование по двум индексам можем преобразовать суммирование по одному индексу. Для этого мы изменим верхние границы для обоих индексов: $i \leq k$ и $j \leq k$, при этом мы ничего не теряем, поскольку $i \geq 0$, $j \geq 0$ и $i + j = k$. Случаи, когда $i > k$ (при $u > k$), или $j > k$ (при $v > k$), все равно никогда не пройдут условие $i + j = k$, так как $i \geq 0$ и $j \geq 0$. А при $i > u$ (при $k > u$) коэффициенты $a_i = 0$, а при $j > v$ (при $k > v$) коэффициенты $b_j = 0$. Тогда в итоге мы получаем три условия: $i + j = k$, $0 \leq i \leq k$ и $0 \leq j \leq k$. Заметим, что оба индекса неотрицательны и ограничены в одним и тем же верхним пределом, причем сумма индексов также равна верхнему пределу. Тогда, очевидно, можно избавиться от одного из индексов, попросту выразив его через другой: $j = k - i$, и мы ничего не теряем, поскольку $0 \leq k - i \leq k$, так как $0 \leq i \leq k$. В итоге мы получаем следующую формулу полной свертки:

$$(a(x) \cdot b(x)) \bmod (g(x)) = a(x) \cdot b(x) = \sum_{k=0}^{u+v} x^k \cdot \overbrace{\left(\sum_{i=0}^{k} \left(a_i \cdot b_{k-i} \right) \right)}^{GF(p)} \qquad (1.5.2)$$

$$\forall i > u = \deg(a(x)) : a_i = 0; \quad \forall j > v = \deg(b(x)) : b_j = 0; \quad \deg(g(x)) > u + v$$

Следует отметить, что полные линейные свертки, являющиеся по сути произведением двух многочленов, не представляют особого интереса в практике кодирования информации. На практике гораздо большее распространение имеют свертки, в которых многочлен $g(x)$ имеет некоторую фиксированную степень $m \geq 2$, а степень произведения $a(x) \cdot b(x)$ может быть как меньше, так и больше либо равна степени многочлена $g(x)$. При этом степень результирующего многочлена-свертки всегда меньше степени многочлена $g(x)$ в силу свойств операции вычисления остатка одного многочлена по модулю другого.

Пример. Вычислим свертку многочленов $a(x)=4\cdot x^3+x^2+2$ и $b(x)=2\cdot x^2+x+3$ заданных над полем $GF(5)$ при заданном многочлене $g(x)=x^8+2\cdot x^4+3$. Заметим, что степень многочлена $g(x)$ равна 8, и она больше суммы степеней многочленов $a(x)$ и $b(x)$, равной 5, а значит, мы имеем дело с линейной сверткой. Вычислим сначала произведение $a(x)\cdot b(x)=(4\cdot x^3+x^2+2)\cdot(2\cdot x^2+x+3)=\underbrace{(4\cdot 2)}_{GF(5)}\cdot x^5+\underbrace{(4\cdot 1+1\cdot 2)}_{GF(5)}\cdot x^4+\underbrace{(4\cdot 3+1\cdot 1)}_{GF(5)}\cdot x^3+$ $\underbrace{(1\cdot 3+2\cdot 2)}_{GF(5)}\cdot x^2+\underbrace{(2\cdot 1)}_{GF(5)}\cdot x+\underbrace{(2\cdot 3)}_{GF(5)}=3\cdot x^5+x^4+3\cdot x^3+2\cdot x^2+2\cdot x+1$. Попробуем теперь вычислить остаток от полученного многочлена-произведения по модулю многочлена $g(x)$:

$$\begin{array}{l} 3\cdot x^5+1\cdot x^4+3\cdot x^3+2\cdot x^2+2\cdot x+1 \\ \underline{0\cdot x^5+0\cdot x^4+0\cdot x^3+0\cdot x^2+0\cdot x+0} \\ 3\cdot x^5+1\cdot x^4+3\cdot x^3+2\cdot x^2+2\cdot x+1 \end{array}\Bigg| \begin{array}{l} g(x)=x^8+2\cdot x^4+3 \\ \hline 0 \end{array}$$

. Легко видим, что остатком произведения по модулю многочлена $g(x)$ является само произведение, поскольку степень многочлена-произведения меньше степени многочлена $g(x)$, и таким образом получаем линейную свертку $(a(x)\cdot b(x))\bmod g(x)=a(x)\cdot b(x)=3\cdot x^5+x^4+3\cdot x^3+2\cdot x^2+2\cdot x+1$.

Определение 8. Если многочлен $g(x)=x^m$, $m\ge 2$, то свертку $(a(x)\cdot b(x))\bmod(x^m)$, мы будем называть **усеченной линейной сверткой**. Нетрудно заметить, что усеченная линейная свертка легко получается из полной линейной свертки, то есть произведения многочленов $a(x)\cdot b(x)$, путем «отбрасывания» всех слагаемых, в которых степень переменной x больше либо равна m. Тогда, очевидно, степень усеченной линейной свертки всегда меньше либо равна $m-1$.

Примечание. Особо отметим, что мы не налагаем никаких условий на сумму степеней многочленов $a(x)$ и $b(x)$. То есть в любом случае свертку вида $(a(x)\cdot b(x))\bmod(x^m)$ мы называем усеченной линейной сверткой. В случае $m>\deg(a(x))+\deg(b(x))$ усеченная линейная свертка будет совпадать с полной линейной сверткой.

Очевидно, что усеченная линейная свертка может быть вычислена как:

$$\begin{gathered}(a(x)\cdot b(x))\bmod(x^m)=\sum_{k=0}^{m-1}x^k\cdot\overbrace{\left(\sum_{i=0}^{k}\left(a_i\cdot b_{k-i}\right)\right)}^{GF(p)} \\ \forall i>\deg(a(x)):a_i=0;\quad \forall j>\deg(b(x)):b_j=0;\quad m\ge 2\end{gathered} \tag{1.6.1}$$

Перепишем полученную формулу в развернутом виде для наглядности:

$$\begin{cases} (a(x)\cdot b(x))\bmod(x^m)=\sum\limits_{k=0}^{m-1}c_k\cdot x^k;\quad m\ge 2 \\ GF(p)\begin{cases} c_0=a_0\cdot b_0 \\ c_1=a_0\cdot b_1+a_1\cdot b_0 \\ \vdots \\ c_{m-1}=a_0\cdot b_{m-1}+a_1\cdot b_{m-2}+\ldots+a_{m-1}\cdot b_0 \end{cases} \\ \forall i>\deg(a(x)):a_i=0;\quad \forall j>\deg(b(x)):b_j=0 \end{cases} \tag{1.6.2}$$

Пример. Вычислим усеченную линейную свертку $(a(x)\cdot b(x))\bmod(x^4)$ для многочленов $a(x)=4\cdot x^3+x^2+2$ и $b(x)=2\cdot x^2+x+3$ заданных над полем $GF(5)$. Для начала вычислим усеченную линейную свертку, используя определение, то есть вычислим произведение $a(x)\cdot b(x)$, а затем вычислим остаток по модулю многочлена x^4. Итак, имеем произведение $a(x)\cdot b(x)=(4\cdot x^3+x^2+2)\cdot(2\cdot x^2+x+3)=3\cdot x^5+x^4+3\cdot x^3+2\cdot x^2+2\cdot x+1$.

Теперь вычислим остаток по модулю x^4:
$$\begin{array}{r|l} 3\cdot x^5+1\cdot x^4+3\cdot x^3+2\cdot x^2+2\cdot x+1 & g(x)=x^4 \\ \underline{3\cdot x^5+0\cdot x^4+0\cdot x^3+0\cdot x^2+0\cdot x+0} & \overline{3\cdot x+1} \\ 1\cdot x^4+3\cdot x^3+2\cdot x^2+2\cdot x+1 & \\ \underline{1\cdot x^4+0\cdot x^3+0\cdot x^2+0\cdot x+0} & \\ 3\cdot x^3+2\cdot x^2+2\cdot x+1 & \end{array}.$$

Таким образом, $(a(x)\cdot b(x))\bmod(x^4)=3\cdot x^3+2\cdot x^2+2\cdot x+1$. Заметим, что операция вычисления остатка по модулю фактически приводит к «отбрасыванию» (обнулению) всех слагаемых многочлена-произведения, степень которых больше либо равна $m=4$.

Теперь вычислим эту же свертку $(a(x)\cdot b(x))\bmod(x^4)$, но используя полученную выше оптимизированную формулу. Заметим, что $(\deg(b(x))=2)<(m-1=3)$, поэтому при суммировании у нас будет участвовать «отсутствующий» в многочлене $b(x)$ коэффициент b_3, который равен нулю. Тогда имеем: $(a(x)\cdot b(x))\bmod(x^4)=\sum\limits_{k=0}^{3}x^k\cdot\overbrace{\sum\limits_{i=0}^{k}\left(a_i\cdot b_{k-i}\right)}^{GF(5)}=$

$$=\overbrace{(a_0b_0)}^{GF(5)}+\overbrace{(a_0b_1+a_1b_0)}^{GF(5)}\cdot x+\overbrace{(a_0b_2+a_1b_1+a_2b_0)}^{GF(5)}\cdot x^2+\overbrace{(a_0b_3+a_1b_2+a_2b_1+a_3b_0)}^{GF(5)}\cdot x^3=$$

$$\overbrace{(2\cdot 3)}^{GF(5)}+\overbrace{(2\cdot 1+0\cdot 3)}^{GF(5)}\cdot x+\overbrace{(2\cdot 2+0\cdot 1+1\cdot 3)}^{GF(5)}\cdot x^2+\overbrace{(2\cdot 0+0\cdot 2+1\cdot 1+4\cdot 3)}^{GF(5)}\cdot x^3=1+2\cdot x+2\cdot x^2+3\cdot x^3.$$

Тогда окончательно имеем: $(a(x)\cdot b(x))\bmod(x^4)=3\cdot x^3+2\cdot x^2+2\cdot x+1$.

Таким образом, при помощи оптимизированной формулы мы получили тот же самый результат. Однако, благодаря использованию оптимизированной формулы, мы вместо вычисления всех коэффициентов многочлена-произведения $a(x)\cdot b(x)$, мы вычисляем только младшие m коэффициента, что же касается операции вычисления остатка по модулю многочлена x^m, то она фактически вообще не требуется в оптимизированной формуле.

Определение 9. Если $g(x)=x^m-1$, $m\geq 2$, то свертку вида $(a(x)\cdot b(x))\bmod(x^m-1)$ мы будем называть **циклической сверткой**. Отметим, что циклические свертки играют важнейшую роль в практике кодирования информации.

Особо отметим, что в двучлене x^m-1, младший коэффициент равен «–1», и так записано ради общности – применимости для любого простого поля *GF(p)*, но по сути этот коэффициент следует понимать как обратный элемент по сложению для элемента «1» в заданном простом поле *GF(p)*. Таким образом, в каждом конкретном поле *GF(p)* двучлен x^m-1 будет выглядеть по-разному: например, в поле *GF*(2) как x^m+1, в поле *GF*(3) как x^m+2, в поле *GF*(5) как x^m+4, и так далее, в поле *GF(p)* как $x^m+(p-1)$.

Перейдем теперь к выводу «оптимизированной» формулы для вычисления циклической свертки $(a(x)\cdot b(x))\bmod(x^m-1)$. Заметим, что произведение многочленов $a(x)$ степени u и $b(x)$ степени v может быть вычислено по формуле, которую мы рассматривали выше: $a(x)\cdot b(x)=\sum_{k=0}^{u+v} x^k\cdot\overbrace{\left(\sum_{i=0}^{k}(a_i\cdot b_{k-i})\right)}^{GF(p)}$, подразумевая, что

$$\forall i>u=\deg(a(x)):a_i=0;\quad \forall j>v=\deg(b(x)):b_j=0;$$

«отсутствующие» коэффициенты многочленов $a(x)$ и $b(x)$ равны нулю.

Теперь сделаем следующее обозначение: $u+v=\varphi\cdot m+\theta$, где $\varphi\ge 0$ и $0\le\theta\le\rho-1$ являются целыми числами. Тогда произведение многочленов можно переписать следующим образом: $a(x)\cdot b(x)=\sum_{k=0}^{\varphi\cdot m+\theta} x^k\cdot\sum_{i=0}^{k}(a_i\cdot b_{k-i})$. Заметим, что мы можем расширить пределы для внешнего суммирования: $0\le k\le\varphi\cdot m+(m-1)$, поскольку при $k>\varphi\cdot m+\theta=u+v$ внутренняя сумма $\sum_{i=0}^{k}(a_i\cdot b_{k-i})$ равна нулю, так как при индексах $0\le i\le u$, при которых могут быть ненулевые коэффициенты a_i, индексы $k-i>v$, поскольку $k>u+v$, а при таких индексах, соответственно, коэффициенты $b_{k-i}=0$. Таким образом, имеем: $a(x)\cdot b(x)=\sum_{k=0}^{\varphi\cdot m+(m-1)} x^k\cdot\sum_{i=0}^{k}(a_i\cdot b_{k-i})$. Теперь разложим внешнюю сумму на слагаемые следующим образом: $a(x)\cdot b(x)=\sum_{k=0}^{\varphi\cdot m+(m-1)} x^k\cdot\sum_{i=0}^{k}(a_i\cdot b_{k-i})=$

$$\sum_{k=\varphi\cdot m}^{\varphi\cdot m+(m-1)} x^k\sum_{i=0}^{k}(a_i\cdot b_{k-i})+\ldots+\sum_{k=m}^{m+(m-1)} x^k\sum_{i=0}^{k}(a_i\cdot b_{k-i})+\sum_{k=0}^{m-1} x^k\sum_{i=0}^{k}(a_i\cdot b_{k-i}).$$

Теперь выполним преобразование (смещение) индекса во внешнем суммировании каждого слагаемого – в младшем слагаемом $k\to k$, в следующем слагаемом $k\to m+k$, и так далее, в старшем слагаемом $k\to\varphi\cdot m+k$, и получим: $\sum_{k=0}^{m-1} x^{\varphi\cdot m+k}\cdot\left(\sum_{i=0}^{\varphi\cdot m+k}(a_i\cdot b_{\varphi\cdot m+k-i})\right)+\ldots$

$$\ldots+\sum_{k=0}^{m-1} x^{m+k}\cdot\left(\sum_{i=0}^{m+k}(a_i\cdot b_{m+k-i})\right)+\sum_{k=0}^{m-1} x^k\cdot\left(\sum_{i=0}^{k}(a_i\cdot b_{k-i})\right).$$

Заметим, что полученную сумму нетрудно «свернуть» в одну тройную сумму: $\sum_{\mu=0}^{\varphi}\left(\sum_{k=0}^{m-1} x^{\mu\cdot m+k}\cdot\left(\sum_{i=0}^{\mu\cdot m+k}(a_i\cdot b_{\mu\cdot m+k-i})\right)\right)$. Преобразуем выражение $x^{\mu\cdot m+k}$ в выражение $(x^{\mu\cdot m}-1+1)\cdot x^k$, и тогда получаем следующее выражение: $a(x)\cdot b(x)=$

$$\sum_{\mu=0}^{\varphi}(x^{\mu\cdot m}-1)\sum_{k=0}^{m-1} x^k\cdot\left(\sum_{i=0}^{\mu\cdot m+k}(a_i b_{\mu\cdot m+k-i})\right)+\sum_{\mu=0}^{\varphi}\sum_{k=0}^{m-1} x^k\cdot\left(\sum_{i=0}^{\mu\cdot m+k}(a_i b_{\mu\cdot m+k-i})\right).$$

Теперь, наконец, можем вычислить $(a(x)\cdot b(x))\bmod(x^m-1)$. Учтем, что левая часть полученной выше суммы содержит множитель $(x^{\mu\cdot m}-1)$, который при $\mu=0$ равен нулю, а при $\mu\ge 1$, выражение $(x^{\mu\cdot m}-1)$ по модулю многочлена (x^m-1) также равно нулю, поскольку составной многочлен $(x^{\mu\cdot m}-1)$ содержит среди своих сомножителей (x^m-1). Таким образом, $\left(\sum_{\mu=0}^{\varphi}(x^{\mu\cdot m}-1)\cdot\sum_{k=0}^{m-1}x^k\cdot\left(\sum_{i=0}^{\mu\cdot m+k}(a_i b_{\mu\cdot m+k-i})\right)\right)\bmod(x^m-1)=0$. Что же касается правой части полученной выше суммы, то нетрудно заметить, что степень переменной x содержащейся в сумме $\sum_{\mu=0}^{\varphi}\sum_{k=0}^{m-1}x^k\cdot\left(\sum_{i=0}^{\mu\cdot m+k}(a_i b_{\mu\cdot m+k-i})\right)$ не выше $m-1$, и, соответственно, остатком от суммы по модулю многочлена (x^m-1) будет сама сумма.

Тогда, с учетом всего вышесказанного, а также учитывая, что $u+v=\varphi\cdot m+\theta$, где $\varphi\ge 0$ и $0\le\theta\le m-1$, $u=\deg(a(x))$, $v=\deg(b(x))$, откуда нетрудно выразить параметр $\varphi=\lfloor(u+v)/m\rfloor=\lfloor(\deg(a(x))+\deg(b(x)))/m\rfloor$, можем окончательно записать формулу для вычисления циклической свертки двух многочленов $a(x)$ и $b(x)$:

$$(a(x)\cdot b(x))\bmod(x^m-1)=\sum_{k=0}^{m-1}x^k\cdot\overbrace{\left(\sum_{\mu=0}^{\varphi}\left(\sum_{i=0}^{\mu\cdot m+k}\left(a_i\cdot b_{\mu\cdot m+k-i}\right)\right)\right)}^{GF(p)}$$
$$\varphi=\lfloor(\deg(a(x))+\deg(b(x)))/m\rfloor;\quad m\ge 2; \qquad (1.7.1)$$
$$\forall i>\deg(a(x)):a_i=0;\quad\forall j>\deg(b(x)):b_j=0;$$

Наконец, если обозначить $j=\mu\cdot m+k-i$, то нетрудно заметить, что $i+j=\mu\cdot m+k$. Тогда с одной стороны, мы можем применить операцию вычисления остатка по модулю m к левой и правой части условия $i+j=\mu\cdot m+k$, и, учитывая, что $\mu\ge 0$ целое число, а $0\le k\le m-1$, а значит $(\mu\cdot m+k)\bmod m=k$, то получим новое условие: $(i+j)\bmod m=k$, и таким образом мы можем избавиться от суммирования по $0\le\mu\le\varphi$. С другой стороны, чтобы «не упустить из внимания» ни один из коэффициентов многочленов $a(x)$ и $b(x)$, пределы для индексов должны быть, соответственно, $0\le i\le\deg(a(x))$ и $0\le j\le\deg(b(x))$. Тогда в итоге мы можем записать более компактную формулу для циклической свертки:

$$(a(x)\cdot b(x))\bmod(x^m-1)=\sum_{k=0}^{m-1}x^k\cdot\left(\overbrace{\sum_{\substack{(i+j)\bmod m=k\\ i=0\ldots\deg(a(x))\quad j=0\ldots\deg(b(x))}}\left(a_i\cdot b_j\right)}^{GF(p)}\right) \qquad (1.7.2)$$

Примечание. Нетрудно заметить, что если сумма степеней многочленов $a(x)$ и $b(x)$ меньше либо равна $m-1$, иными словами $\deg(a(x))+\deg(b(x))\le m-1$, то сумма индексов $0\le i+j\le m-1$, так как $0\le i\le\deg(a(x))$ и $0\le j\le\deg(b(x))$. Тогда, $(i+j)\bmod m=i+j$, и условие $(i+j)\bmod m=k$ упрощается: $i+j=k$. Кроме того, $0\le k\le\deg(a(x))+\deg(b(x))$, так как при $k>\deg(a(x))+\deg(b(x))$, условие $i+j=k$ все равно не выполняется. В таком случае, циклическая свертка превращается в полную линейную свертку: $\sum_{k=0}^{\deg(a(x))+\deg(b(x))}x^k\cdot\left(\sum_{i+j=k}\left(a_i\cdot b_j\right);\ \begin{matrix}i=0\ldots\deg(a(x))\\ j=0\ldots\deg(b(x))\end{matrix}\right);\quad\deg(a(x))+\deg(b(x))\le m-1.$

На практике большую важность имеет частный случай циклической свертки $(a(x)\cdot b(x))\bmod(x^m-1)$, когда степени многочленов $a(x)$ и $b(x)$ меньше либо равны $m-1$, иными словами $\deg(a(x))\le m-1$ и $\deg(b(x))\le m-1$. В таком случае, очевидно, $0\le i\le m-1$ и $0\le j\le m-1$. Тогда, в вышеприведенной формуле циклической свертки условие $(i+j)\bmod m=k$ можно преобразовать следующим образом: $(i\bmod m+j\bmod m)\bmod m=k\Rightarrow$ $\Rightarrow j\bmod m=(k-i\bmod m)\bmod m\Rightarrow j=(k-i)\bmod m$, так как $0\le i\le m-1$ и $0\le j\le m-1$. Однако, заметим, что при $i>k$ имеем $k-i<0$, и чтобы избегать вычисления остатка отрицательных чисел по модулю m, равенство $j=(k-i)\bmod m$ можно эквивалентно преобразовать: $j=(m+k-i)\bmod m$. Таким образом, мы выражаем один индекс через другой, и суммирование по двум индексам, связанным равенством, сводим к суммированию по одному индексу. Тогда, получаем окончательную формулу для вычисления циклической свертки $(a(x)\cdot b(x))\bmod(x^m-1)$ для случая, когда $\deg(a(x))\le m-1$ и $\deg(b(x))\le m-1$:

$$\begin{gathered}(a(x)\cdot b(x))\bmod(x^m-1)=\sum_{k=0}^{m-1}x^k\cdot\overbrace{\left(\sum_{i=0}^{m-1}\left(a_i\cdot b_{(m+k-i)\bmod m}\right)\right)}^{GF(p)}\\ \forall i>\deg(a(x)):a_i=0;\quad \forall j>\deg(b(x)):b_j=0;\quad m\ge 2\\ \deg(a(x))\le m-1;\quad \deg(b(x))\le m-1\end{gathered}\tag{1.7.3}$$

Таким образом, вычисление циклической свертки в частном случае $\deg(a(x))\le m-1$ и $\deg(b(x))\le m-1$, заметно проще, чем для случая произвольных многочленов $a(x)$ и $b(x)$. Характерной особенностью полученной формулы является использование операции вычисления остатка по модулю m в индексах коэффициентов одного из многочленов.

Перепишем полученную формулу в развернутом виде для наглядности:

$$\begin{cases}(a(x)\cdot b(x))\bmod(x^m-1)=\sum_{k=0}^{m-1}c_k\cdot x^k;\quad m\ge 2\\ GF(p)\begin{cases}c_0=a_0\cdot b_0+a_1\cdot b_{m-1}+\ldots+a_{m-2}\cdot b_2+a_{m-1}\cdot b_1\\ c_1=a_0\cdot b_1+a_1\cdot b_0+a_2\cdot b_{m-1}+\ldots+a_{m-1}\cdot b_2\\ \vdots\\ c_{m-2}=a_0\cdot b_{m-2}+\ldots+a_{m-3}\cdot b_1+a_{m-2}\cdot b_0+a_{m-1}\cdot b_{m-1}\\ c_{m-1}=a_0\cdot b_{m-1}+a_1\cdot b_{m-2}+\ldots+a_{m-2}\cdot b_1+a_{m-1}\cdot b_0\end{cases}\\ \forall i>\deg(a(x)):a_i=0;\quad \forall j>\deg(b(x)):b_j=0\\ \deg(a(x))\le m-1;\quad \deg(b(x))\le m-1\end{cases}\tag{1.7.4}$$

Заметим, что степень многочлена $c(x)=(a(x)\cdot b(x))\bmod(x^m-1)$, являющегося циклической сверткой двух многочленов $a(x)$ и $b(x)$, причем $\deg(a(x))\le m-1$ и $\deg(b(x))\le m-1$, также не превышает $m-1$, иными словами $\deg(c(x))\le m-1$. Также заметим, что каждый коэффициент c_k вычисляется как сумма из m слагаемых вида $a_i\cdot b_j$, где $0\le i\le m-1$, $0\le j\le m-1$, причем нетрудно заметить, что $(i+j)\bmod m=k$. Также отметим, что в развернутом виде формулы хорошо видно, что при переходе от одного коэффициента c_k к другому в соответствующих суммах коэффициенты a_i «остаются на одном месте», а коэффициенты b_j «циклически переставляются».

Пример 1. Вычислим циклическую свертку $(a(x)\cdot b(x))\bmod(x^4-1)$ для многочленов $a(x)=4\cdot x^3+x^2+2$ и $b(x)=2\cdot x^2+x+3$ заданных над полем $GF(5)$.

Сначала вычислим циклическую свертку, используя определение, то есть, вычислив произведение многочленов, а затем вычислив остаток по модулю многочлена x^4-1. Также заметим, что многочлен x^4-1 над полем $GF(5)$ эквивалентен многочлену x^4+4, поскольку «4» является обратным элементом по сложению для «1» в простом поле $GF(5)$. Имеем: $a(x)\cdot b(x)=(4\cdot x^3+x^2+2)\cdot(2\cdot x^2+x+3)=3\cdot x^5+x^4+3\cdot x^3+2\cdot x^2+2\cdot x+1$. Тогда остаток $a(x)\cdot b(x)$ по модулю многочлена x^4+4:

$$\begin{array}{r|l} 3\cdot x^5+1\cdot x^4+3\cdot x^3+2\cdot x^2+2\cdot x+1 & x^4+4 \\ \underline{3\cdot x^5+0\cdot x^4+0\cdot x^3+0\cdot x^2+2\cdot x+0} & \overline{3\cdot x+1} \\ 1\cdot x^4+3\cdot x^3+2\cdot x^2+0\cdot x+1 & \\ \underline{1\cdot x^4+0\cdot x^3+0\cdot x^2+0\cdot x+4} & \\ 3\cdot x^3+2\cdot x^2+0\cdot x+2 & \end{array}.$$

Таким образом, $((4\cdot x^3+x^2+2)\cdot(2\cdot x^2+x+3))\bmod(x^4-1)=3\cdot x^3+2\cdot x^2+2$.

Теперь, отметим, что $\deg(a(x))=3\le m-1=3$ и $\deg(b(x))=2\le m-1=3$, и вычислим циклическую свертку, используя полученную выше «оптимизированную» формулу.

Сначала для наглядности выпишем все коэффициенты многочленов $a(x)$ и $b(x)$: $a_0=2;a_1=0;a_2=1;a_3=4$ и $b_0=3;b_1=1;b_2=2;b_3=0$. Тогда циклическая свертка:

$$(a(x)\cdot b(x))\bmod(x^4-1)=\sum_{k=0}^{3}x^k\cdot\overbrace{\left(\sum_{i=0}^{3}\left(a_i\cdot b_{(4+k-i)\bmod 4}\right)\right)}^{GF(5)}=\overbrace{(a_0b_0+a_1b_3+a_2b_2+a_3b_1)}^{GF(5)}+$$

$$\overbrace{(a_0b_1+a_1b_0+a_2b_3+a_3b_2)}^{GF(5)}x+\overbrace{(a_0b_2+a_1b_1+a_2b_0+a_3b_3)}^{GF(5)}x^2+\overbrace{(a_0b_3+a_1b_2+a_2b_1+a_3b_0)}^{GF(5)}x^3=$$

$$=\overbrace{(2\cdot 3+0\cdot 0+1\cdot 2+4\cdot 1)}^{GF(5)}+\overbrace{(2\cdot 1+0\cdot 3+1\cdot 0+4\cdot 2)}^{GF(5)}\cdot x+\overbrace{(2\cdot 2+0\cdot 1+1\cdot 3+4\cdot 0)}^{GF(5)}\cdot x^2+$$

$$+\overbrace{(2\cdot 0+0\cdot 2+1\cdot 1+4\cdot 3)}^{GF(5)}\cdot x^3=2+0\cdot x+2\cdot x^2+3\cdot x^3=3\cdot x^3+2\cdot x^2+2.$$

Таким образом, по «оптимизированной» формуле мы получили тот же результат: $((4\cdot x^3+x^2+2)\cdot(2\cdot x^2+x+3))\bmod(x^4-1)=3\cdot x^3+2\cdot x^2+2$.

Пример 2. Вычислим циклическую свертку $(a(x)\cdot b(x))\bmod(x^4-1)$ для многочленов $a(x)=x^2$ и $b(x)=x^2+x$ заданных над полем $GF(2)$.

Заметим, что в многочлене $a(x)$ только коэффициент $a_2=1$, остальные равны нулю. Тогда, используя формулу для вычисления циклической свертки, получаем следующее:

$$(a(x)\cdot b(x))\bmod(x^4-1)=\sum_{k=0}^{3}x^k\cdot\overbrace{\left(\sum_{i=0}^{3}\left(a_i\cdot b_{(4+k-i)\bmod 4}\right)\right)}^{GF(2)}=\sum_{k=0}^{3}x^k\cdot\overbrace{a_2\cdot b_{(k+2)\bmod 4}}^{GF(2)}$$

$$=b_2+b_3\cdot x+b_0\cdot x^2+b_1\cdot x^3=1+x^3.$$

Таким образом $((x^2)\cdot(x^2+x))\bmod(x^4-1)=x^3+1$.

Определение 10. Пусть заданы два многочлена $a(x)$ степени $\deg(a(x)) \le m-1$ и $b(x)$ степени $\deg(b(x)) \le m-1$, а также нормированный многочлен $g(x)$ степени $m \ge 2$ над полем *GF(p)*. Тогда свертку $(a(x)\cdot b(x)) \bmod g(x)$ над полем *GF(p)* будем называть **кольцевой сверткой**. Если многочлен $g(x)$ неприводим над полем *GF(p)*, то кольцевую свертку $(a(x)\cdot b(x)) \bmod g(x)$ будем называть **полевой сверткой**.

Ключевым свойством полевой свертки является то, что для любого ненулевого многочлена $a(x)$ степени $\le m-1$ существует «обратный по умножению» многочлен $a^{-1}(x)$ степени $\le m-1$ такой, что полевая свертка $(a(x)\cdot a^{-1}(x)) \bmod g(x) = 1$. В случае же кольцевой свертки для многочлена $a(x)$ существует «обратный по умножению» многочлен $a^{-1}(x)$, только если наибольший общий делитель многочленов $a(x)$ и $g(x)$ является скаляром (многочленом нулевой степени), иными словами $\deg(НОД(a(x), g(x))) = 0$.

Кольцевые и полевые свертки имеют исключительную важность, как с теоретической, так и с практической точки зрения. Используя сложение многочленов в качестве бинарной операции сложения, а также кольцевую свертку в качестве бинарной операции умножения можно построить, так называемое, кольцо многочленов над полем *GF(p)*, состоящее из множества всевозможных многочленов-остатков по модулю многочлена $g(x)$, и замкнутое относительно обеих бинарных операций. Если к тому же многочлен $g(x)$ неприводим над полем *GF(p)*, то кольцевая свертка также является полевой, а соответствующее кольцо многочленов является полем многочленов над полем *GF(p)*.

В общем случае для заданного многочлена $g(x)$ «оптимизированная» формула для «прямого» вычисления коэффициентов многочлена, являющегося кольцевой сверткой $c(x) = (a(x)\cdot b(x)) \bmod g(x)$, по коэффициентам многочленов $a(x)$, $b(x)$ и $g(x)$, оказывается достаточно громоздкой и то, насколько ее можно упростить, зависит от конкретного вида многочлена $g(x)$. В то же время можно вывести достаточно компактную «рекуррентную» формулу для вычисления кольцевой свертки. Остановимся на этом подробнее.

Заметим, что многочлен $b(x) = b_{m-1}\cdot x^{m-1} + \ldots + b_1 \cdot x + b_0$ мы можем представить в виде: $b(x) = (\ldots((0 + b_{m-1})\cdot x + b_{m-2})\cdot x + \ldots + b_1)\cdot x + b_0$. Также учтем свойства остатков суммы и произведения многочленов: $(a(x)+b(x)) \bmod g(x) = a(x) \bmod g(x) + b(x) \bmod g(x)$ и $(a(x)\cdot b(x)) \bmod g(x) = (a(x) \bmod g(x))\cdot(b(x) \bmod g(x)) \bmod g(x)$, а также свойство остатка многочлена, умноженного на скаляр: $(\lambda \cdot a(x)) \bmod g(x) = \lambda \cdot (a(x) \bmod g(x))$.

Тогда, $(a(x)\cdot b(x)) \bmod g(x) = (a(x)\cdot b_{m-1}\cdot x^{m-1} + \ldots + a(x)\cdot b_1 \cdot x + a(x)\cdot b_0) \bmod g(x)$

$= b_{m-1}\cdot(a(x)\cdot x^{m-1}) \bmod g(x) + \ldots + b_1\cdot(a(x)\cdot x) \bmod g(x) + b_0\cdot(a(x)) \bmod g(x)) =$

$= ((\ldots((((0 \bmod g(x) + a(x)\cdot b_{m-1})\cdot x) \bmod g(x) + a(x)\cdot b_{m-2})\cdot x) \bmod g(x) + \ldots$

$\ldots + a(x)\cdot b_1)\cdot x) \bmod g(x) + a(x)\cdot b_0$, учитывая, что $\deg(a(x)) \le m-1 \Rightarrow a(x) \bmod (g(x) = a(x)$.

Тогда, при таком рекуррентном представлении формулы для вычисления кольцевой свертки $(a(x)\cdot b(x)) \bmod g(x)$, можем записать следующую итерационную схему вычисления:

$$\begin{cases} s = 1\ldots m; \quad m \ge 2; \quad c^{(0)}(x) = 0; & (a(x)\cdot b(x)) \bmod (g(x)) = c^{(m)}(x) \\ c^{(s)}(x) = \overbrace{(x\cdot c^{(s-1)}(x)) \bmod g(x) + a(x)\cdot b_{m-s}}^{GF(p)} & \deg(a(x)), \deg(b(x)) \le m-1 \\ & m = \deg(g(x)) \end{cases}$$

Результат вычислений на последней итерации $s = m$ и будет являться искомой кольцевой сверткой. Иными словами, $(a(x)\cdot b(x)) \bmod g(x) = c^{(m)}(x)$.

Теперь рассмотрим подробнее выражение $(x\cdot c^{(s-1)}(x)) \bmod g(x)$. Учтем, что поскольку $\deg(a(x)) \le m-1$ и $\deg(x\cdot c^{(s-1)}(x)) \bmod g(x)) \le m-1$, то $\deg(c^{(s)}(x)) \le m-1$. Тогда, в худшем случае $\deg(x\cdot c^{(s-1)}(x)) = m$, и многочлен $x\cdot c^{(s-1)}(x)$ имеет следующий вид: $c_{m-1}^{(s-1)}\cdot x^m + \ldots + c_1^{(s-1)}\cdot x^2 + c_0^{(s-1)}\cdot x$. Учитывая, что многочлен $g(x)$ является нормированным и имеет вид: $g(x) = x^m + g_{m-1}\cdot x^{m-1} + \ldots + g_1\cdot x + g_0$, то деление многочлена $x\cdot c^{(s-1)}(x)$ на многочлен $g(x)$ фактически выполняется «за один шаг»:

$$\begin{array}{l} c_{m-1}^{(s-1)}\cdot x^m + c_{m-2}^{(s-1)}\cdot x^{m-1} + \ldots + c_0^{(s-1)}\cdot x + 0 \\ \underline{c_{m-1}^{(s-1)}\cdot x^m + c_{m-1}^{(s-1)}\cdot g_{m-1}\cdot x^{m-1} + \ldots + c_{m-1}^{(s-1)}\cdot g_1\cdot x + c_{m-1}^{(s-1)}\cdot g_0} \\ 0\cdot x^m + (c_{m-2}^{(s-1)} - c_{m-1}^{(s-1)}\cdot g_{m-1})\cdot x^{m-1} + \ldots + (c_0^{(s-1)} - c_{m-1}^{(s-1)}\cdot g_1)\cdot x - c_{m-1}^{(s-1)}\cdot g_0 \end{array} \Bigg| \begin{array}{l} g(x) \\ \hline c_{m-1}^{(s-1)} \end{array}$$

Нетрудно заметить, что остаток многочлена $x\cdot c^{(s-1)}(x)$ по модулю многочлена $g(x)$ фактически получается путем вычисления разности $(x\cdot c^{(s-1)}(x) - c_{m-1}^{(s-1)}\cdot g(x))$. Особо отметим, что при вычислении разности, коэффициенты при x^m всегда взаимно уничтожаются, поэтому степень многочлена-разности всегда $\le m-1$.

Тогда мы можем переписать рекуррентную схему вычисления кольцевую свертки, используя полученное равенство $(x\cdot c^{(s-1)}(x)) \bmod g(x) = x\cdot c^{(s-1)}(x) - c_{m-1}^{(s-1)}\cdot g(x)$:

$$\begin{cases} s=1\ldots m; \quad m\ge 2; \quad c^{(0)}(x)=0; & (a(x)\cdot b(x)) \bmod (g(x)) = c^{(m)}(x) \\ c^{(s)}(x) = \overbrace{x\cdot c^{(s-1)}(x) - c_{m-1}^{(s-1)}\cdot g(x) + a(x)\cdot b_{m-s}}^{GF(p)} & \deg(a(x)),\deg(b(x)) \le m-1 \\ & m = \deg(g(x)) \end{cases} \quad (1.8.1)$$

В развернутом виде рекуррентная схема выглядит следующим образом:

$$\begin{cases} (a(x)\cdot b(x)) \bmod (g(x)) = \sum\limits_{k=0}^{m-1} c_k^{(m)}\cdot x^k \\ s=1\ldots m; \quad m\ge 2; \quad c_0^{(0)} = \ldots = c_{m-1}^{(0)} = 0; \\ GF(p)\begin{cases} c_0^{(s)} = & -c_{m-1}^{(s-1)}\cdot g_0 + a_0\cdot b_{m-s} \\ c_1^{(s)} = c_0^{(s-1)} & -c_{m-1}^{(s-1)}\cdot g_1 + a_1\cdot b_{m-s} \\ \vdots & \\ c_{m-1}^{(s)} = c_{m-2}^{(s-1)} & -c_{m-1}^{(s-1)}\cdot g_{m-1} + a_{m-1}\cdot b_{m-s} \end{cases} \\ \deg(a(x)),\deg(b(x)) \le m-1; \quad m=\deg(g(x)); \quad g_m = 1 \end{cases} \quad (1.8.2)$$

Таким образом, используя полученную рекуррентную формулу, за m итераций мы вычисляем искомую кольцевую свертку: $c(x) = (a(x)\cdot b(x)) \bmod g(x)$, в качестве которой принимается результат вычислений на последней итерации $s=m$. Иными словами, кольцевая свертка: $(a(x)\cdot b(x)) \bmod g(x) = \sum\limits_{k=0}^{m-1} c_k^{(m)}\cdot x^k$.

Полученная выше рекуррентная формула дает нам универсальную последовательную (итерационную) схему вычисления кольцевой свертки $(a(x)\cdot b(x)) \bmod g(x)$ для любых многочленов $a(x)$ степени $\deg(a(x)) \le m-1$ и $b(x)$ степени $\deg(b(x)) \le m-1$, и любого нормированного многочлена $g(x)$ степени $m \ge 2$, заданных над полем *GF(p)*.

Следует особо отметить, что на практике, особенно при аппаратной реализации схемы вычисления свертки, многочлен $g(x)$ задается еще на этапе проектирования и является «многочленом-константой». В такой ситуации для построения высокопроизводительной аппаратной реализации вычисления кольцевой свертки чаще всего рекуррентную формулу на этапе проектирования итерационно «просчитывают» для заданного многочлена $g(x)$, и получают «прямую» формулу вычисления кольцевой свертки, и уже полученную прямую формулу используют для аппаратной реализации. Рассмотрим это на примерах.

Пример 1. Пусть задан многочлен $g(x) = x^4 + x^2 + 2\cdot x + 2$ степени $m = 4$ над полем *GF*(5). Подставим его в рекуррентную формулу вычисления кольцевой свертки, и получим:

$$\begin{cases} (a(x)\cdot b(x)) \bmod (x^4 + x^2 + 2\cdot x + 2) = \sum\limits_{k=0}^{3} c_k^{(4)} \cdot x^k \\ s = 1\ldots 4; \quad c_0^{(0)} = \ldots = c_3^{(0)} = 0; \quad \deg(a(x)), \deg(b(x)) \le 3 \\ GF(5)\begin{cases} c_0^{(s)} = \quad -c_3^{(s-1)} \cdot 2 + a_0 \cdot b_{4-s} \\ c_1^{(s)} = c_0^{(s-1)} - c_3^{(s-1)} \cdot 2 + a_1 \cdot b_{4-s} \\ c_2^{(s)} = c_1^{(s-1)} - c_3^{(s-1)} \cdot 1 + a_2 \cdot b_{4-s} \\ c_3^{(s)} = c_2^{(s-1)} - c_3^{(s-1)} \cdot 0 + a_3 \cdot b_{4-s} \end{cases} \end{cases}$$

Просчитаем теперь «вручную» рекуррентную формулу по итерациям.

Тогда, на итерации $s = 1$ имеем:

$$GF(5)\begin{cases} c_0^{(1)} = \quad -c_3^{(0)} \cdot 2 + a_0 \cdot b_3 = a_0 \cdot b_3 \\ c_1^{(1)} = c_0^{(0)} - c_3^{(0)} \cdot 2 + a_1 \cdot b_3 = a_1 \cdot b_3 \\ c_2^{(1)} = c_1^{(0)} - c_3^{(0)} \cdot 1 + a_2 \cdot b_3 = a_2 \cdot b_3 \\ c_3^{(1)} = c_2^{(0)} - c_3^{(0)} \cdot 0 + a_3 \cdot b_3 = a_3 \cdot b_3 \end{cases}$$

Далее, на итерации $s = 2$ имеем:

$$GF(5)\begin{cases} c_0^{(2)} = \quad -c_3^{(1)} \cdot 2 + a_0 \cdot b_2 = 3\cdot a_3 \cdot b_3 + a_0 \cdot b_2 \\ c_1^{(2)} = c_0^{(1)} - c_3^{(1)} \cdot 2 + a_1 \cdot b_2 = a_0 \cdot b_3 + 3\cdot a_3 \cdot b_3 + a_1 \cdot b_2 \\ c_2^{(2)} = c_1^{(1)} - c_3^{(1)} \cdot 1 + a_2 \cdot b_2 = a_1 \cdot b_3 + 4\cdot a_3 \cdot b_3 + a_2 \cdot b_2 \\ c_3^{(2)} = c_2^{(1)} - c_3^{(1)} \cdot 0 + a_3 \cdot b_2 = a_2 \cdot b_3 + a_3 \cdot b_2 \end{cases}$$

Далее, на итерации $s = 3$ имеем:

$$GF(5)\begin{cases} c_0^{(3)} = \quad -c_3^{(2)} \cdot 2 + a_0 \cdot b_1 = 3 \cdot a_2 \cdot b_3 + 3 \cdot a_3 \cdot b_2 + a_0 \cdot b_1 \\ c_1^{(3)} = c_0^{(2)} - c_3^{(2)} \cdot 2 + a_1 \cdot b_1 = 3 \cdot a_3 \cdot b_3 + a_0 \cdot b_2 + 3 \cdot a_2 \cdot b_3 + 3 \cdot a_3 \cdot b_2 + a_1 \cdot b_1 \\ c_2^{(3)} = c_1^{(2)} - c_3^{(2)} \cdot 1 + a_2 \cdot b_1 = a_0 \cdot b_3 + 3 \cdot a_3 \cdot b_3 + a_1 \cdot b_2 + 4 \cdot a_2 \cdot b_3 + 4 \cdot a_3 \cdot b_2 + a_2 \cdot b_1 \\ c_3^{(3)} = c_2^{(2)} - c_3^{(2)} \cdot 0 + a_3 \cdot b_1 = a_1 \cdot b_3 + 4 \cdot a_3 \cdot b_3 + a_2 \cdot b_2 + a_3 \cdot b_1 \end{cases}$$

Наконец, на итерации $s = 4$ имеем:

$$GF(5)\begin{cases} c_0^{(4)} = \quad -c_3^{(3)} \cdot 2 + a_0 \cdot b_0 = 3 \cdot a_1 \cdot b_3 + 2 \cdot a_3 \cdot b_3 + 3 \cdot a_2 \cdot b_2 + 3 \cdot a_3 \cdot b_1 + a_0 \cdot b_0 \\ c_1^{(4)} = c_0^{(3)} - c_3^{(3)} \cdot 2 + a_1 \cdot b_0 = 3 \cdot a_2 \cdot b_3 + 3 \cdot a_3 \cdot b_2 + a_0 \cdot b_1 + 3 \cdot a_1 \cdot b_3 + 2 \cdot a_3 \cdot b_3 + \\ \qquad + 3 \cdot a_2 \cdot b_2 + 3 \cdot a_3 \cdot b_1 + a_1 \cdot b_0 \\ c_2^{(4)} = c_1^{(3)} - c_3^{(3)} \cdot 1 + a_2 \cdot b_0 = 4 \cdot a_3 \cdot b_3 + a_0 \cdot b_2 + 3 \cdot a_2 \cdot b_3 + 3 \cdot a_3 \cdot b_2 + a_1 \cdot b_1 + \\ \qquad + 4 \cdot a_1 \cdot b_3 + 4 \cdot a_2 \cdot b_2 + 4 \cdot a_3 \cdot b_1 + a_2 \cdot b_0 \\ c_3^{(4)} = c_2^{(3)} - c_3^{(3)} \cdot 0 + a_3 \cdot b_0 = a_0 \cdot b_3 + 3 \cdot a_3 \cdot b_3 + a_1 \cdot b_2 + 4 \cdot a_2 \cdot b_3 + 4 \cdot a_3 \cdot b_2 \\ \qquad + a_2 \cdot b_1 + a_3 \cdot b_0 \end{cases}$$

Тогда, получаем окончательную «прямую» формулу для вычисления кольцевой свертки $(a(x) \cdot b(x)) \bmod (x^4 + x^2 + 2 \cdot x + 2)$, которая также является полевой сверткой, поскольку многочлен $x^4 + x^2 + 2 \cdot x + 2$ неприводим над полем *GF*(5):

$$\begin{cases} (a(x) \cdot b(x)) \bmod (x^4 + x^2 + 2 \cdot x + 2) = \sum_{k=0}^{3} c_k \cdot x^k \\ \deg(a(x)), \deg(b(x)) \le 3 \end{cases}$$

$$GF(5)\begin{cases} c_0 = a_0 \cdot b_0 + 3 \cdot a_1 \cdot b_3 + 3 \cdot a_2 \cdot b_2 + 3 \cdot a_3 \cdot b_1 + 2 \cdot a_3 \cdot b_3 \\ c_1 = a_0 \cdot b_1 + a_1 \cdot b_0 + 3 \cdot a_1 \cdot b_3 + 3 \cdot a_2 \cdot b_2 + 3 \cdot a_2 \cdot b_3 + \\ \quad + 3 \cdot a_3 \cdot b_1 + 3 \cdot a_3 \cdot b_2 + 2 \cdot a_3 \cdot b_3 \\ c_2 = a_0 \cdot b_2 + a_1 \cdot b_1 + 4 \cdot a_1 \cdot b_3 + a_2 \cdot b_0 + 4 \cdot a_2 \cdot b_2 + \\ \quad + 3 \cdot a_2 \cdot b_3 + 4 \cdot a_3 \cdot b_1 + 3 \cdot a_3 \cdot b_2 + 4 \cdot a_3 \cdot b_3 \\ c_3 = a_0 \cdot b_3 + a_1 \cdot b_2 + a_2 \cdot b_1 + 4 \cdot a_2 \cdot b_3 + a_3 \cdot b_0 + \\ \quad + 4 \cdot a_3 \cdot b_2 + 3 \cdot a_3 \cdot b_3 \end{cases}$$

Забегая вперед, отметим, что полученная формула фактически позволяет выполнять «быстрое» умножение двух заданных элементов поля многочленов $GF(5^4)$, заданного над простым полем *GF*(5) при помощи неприводимого многочлена $x^4 + x^2 + 2 \cdot x + 2$.

Теперь, располагая, «прямой» формулой для вычисления полевой свертки $(a(x)\cdot b(x))\bmod(x^4+x^2+2\cdot x+2)$, вычислим для примера полевую свертку многочленов $a(x)=4\cdot x^3+x^2+2$ и $b(x)=2\cdot x^2+x+3$ над полем *GF*(5).

Сначала для наглядности выпишем все коэффициенты многочленов $a(x)$ и $b(x)$: $a_0=2; a_1=0; a_2=1; a_3=4$ и $b_0=3; b_1=1; b_2=2; b_3=0$. Тогда подставляя их в «прямую» формулу получаем коэффициенты полевой свертки $\sum_{k=0}^{3} c_k\cdot x^k$:

$$GF(5)\begin{cases} c_0=2\cdot 3+3\cdot 0\cdot 0+3\cdot 1\cdot 2+3\cdot 4\cdot 1+2\cdot 4\cdot 0=4 \\ c_1=2\cdot 1+0\cdot 3+3\cdot 0\cdot 0+3\cdot 1\cdot 2+3\cdot 1\cdot 0+3\cdot 4\cdot 1+3\cdot 4\cdot 2+2\cdot 4\cdot 0=4 \\ c_2=2\cdot 2+0\cdot 1+4\cdot 0\cdot 0+1\cdot 3+4\cdot 1\cdot 2+3\cdot 1\cdot 0+4\cdot 4\cdot 1+3\cdot 4\cdot 2+4\cdot 4\cdot 0=0 \\ c_3=2\cdot 0+0\cdot 2+1\cdot 1+4\cdot 1\cdot 0+4\cdot 3+4\cdot 4\cdot 2+3\cdot 4\cdot 0=0 \end{cases}$$

Таким образом, полевая свертка: $(a(x)\cdot b(x))\bmod(x^4+x^2+2\cdot x+2)=4\cdot x+4$.

Проверим полученный результат. Для этого вычислим свертку традиционным способом, перемножив многочлены $a(x)$ и $b(x)$, и вычислив остаток произведения по модулю многочлена $x^4+x^2+2\cdot x+2$ путем деления «в столбик». Имеем произведение: $a(x)\cdot b(x)=(4\cdot x^3+x^2+2)\cdot(2\cdot x^2+x+3)=3\cdot x^5+x^4+3\cdot x^3+2\cdot x^2+2\cdot x+1$.

Тогда, остаток произведения по модулю многочлена $x^4+x^2+2\cdot x+2$:

$$\begin{array}{r|l} 3\cdot x^5+1\cdot x^4+3\cdot x^3+2\cdot x^2+2\cdot x+1 & x^4+x^2+2\cdot x+2 \\ \underline{3\cdot x^5+0\cdot x^4+3\cdot x^3+1\cdot x^2+1\cdot x+0} & \hline 3\cdot x+1 \\ 1\cdot x^4+0\cdot x^3+1\cdot x^2+1\cdot x+1 & \\ \underline{1\cdot x^4+0\cdot x^3+1\cdot x^2+2\cdot x+2} & \\ 4\cdot x+4 & \end{array}$$

Мы получили тот же самый результат: $(a(x)\cdot b(x))\bmod(x^4+x^2+2\cdot x+2)=4\cdot x+4$.

Пример 2. Пусть задан многочлен $g(x)=x^4+x+1$ степени $m=4$ над полем *GF*(2). Подставим его в рекуррентную формулу вычисления кольцевой свертки, и получим:

$$\begin{cases} (a(x)\cdot b(x))\bmod(x^4+x+1)=\sum_{k=0}^{3} c_k^{(4)}\cdot x^k \\ s=1\ldots 4;\quad c_0^{(0)}=\ldots=c_3^{(0)}=0;\quad \deg(a(x)),\deg(b(x))\le 3 \\ GF(2)\begin{cases} c_0^{(s)}= & -c_3^{(s-1)}\cdot 1+a_0\cdot b_{4-s} \\ c_1^{(s)}=c_0^{(s-1)} & -c_3^{(s-1)}\cdot 1+a_1\cdot b_{4-s} \\ c_2^{(s)}=c_1^{(s-1)} & -c_3^{(s-1)}\cdot 0+a_2\cdot b_{4-s} \\ c_3^{(s)}=c_2^{(s-1)} & -c_3^{(s-1)}\cdot 0+a_3\cdot b_{4-s} \end{cases} \end{cases}$$

Просчитаем теперь «вручную» рекуррентную формулу по итерациям.

Тогда, на итерации $s=1$ имеем:

$$GF(2)\begin{cases} c_0^{(1)} = \quad -c_3^{(0)}\cdot 1 + a_0\cdot b_3 = a_0\cdot b_3 \\ c_1^{(1)} = c_0^{(0)} - c_3^{(0)}\cdot 1 + a_1\cdot b_3 = a_1\cdot b_3 \\ c_2^{(1)} = c_1^{(0)} - c_3^{(0)}\cdot 0 + a_2\cdot b_3 = a_2\cdot b_3 \\ c_3^{(1)} = c_2^{(0)} - c_3^{(0)}\cdot 0 + a_3\cdot b_3 = a_3\cdot b_3 \end{cases}$$

Далее, на итерации $s=2$ имеем:

$$GF(2)\begin{cases} c_0^{(2)} = \quad -c_3^{(1)}\cdot 1 + a_0\cdot b_2 = a_3\cdot b_3 + a_0\cdot b_2 \\ c_1^{(2)} = c_0^{(1)} - c_3^{(1)}\cdot 1 + a_1\cdot b_2 = a_0\cdot b_3 + a_3\cdot b_3 + a_1\cdot b_2 \\ c_2^{(2)} = c_1^{(1)} - c_3^{(1)}\cdot 0 + a_2\cdot b_2 = a_1\cdot b_3 + a_2\cdot b_2 \\ c_3^{(2)} = c_2^{(1)} - c_3^{(1)}\cdot 0 + a_3\cdot b_2 = a_2\cdot b_3 + a_3\cdot b_2 \end{cases}$$

Далее, на итерации $s=3$ имеем:

$$GF(2)\begin{cases} c_0^{(3)} = \quad -c_3^{(2)}\cdot 1 + a_0\cdot b_1 = a_2\cdot b_3 + a_3\cdot b_2 + a_0\cdot b_1 \\ c_1^{(3)} = c_0^{(2)} - c_3^{(2)}\cdot 1 + a_1\cdot b_1 = a_3\cdot b_3 + a_0\cdot b_2 + a_2\cdot b_3 + a_3\cdot b_2 + a_1\cdot b_1 \\ c_2^{(3)} = c_1^{(2)} - c_3^{(2)}\cdot 0 + a_2\cdot b_1 = a_0\cdot b_3 + a_3\cdot b_3 + a_1\cdot b_2 + a_2\cdot b_1 \\ c_3^{(3)} = c_2^{(2)} - c_3^{(2)}\cdot 0 + a_3\cdot b_1 = a_1\cdot b_3 + a_2\cdot b_2 + a_3\cdot b_1 \end{cases}$$

Наконец, на итерации $s=4$ имеем:

$$GF(2)\begin{cases} c_0^{(4)} = \quad -c_3^{(3)}\cdot 1 + a_0\cdot b_0 = a_1\cdot b_3 + a_2\cdot b_2 + a_3\cdot b_1 + a_0\cdot b_0 \\ c_1^{(4)} = c_0^{(3)} - c_3^{(3)}\cdot 1 + a_1\cdot b_0 = a_2\cdot b_3 + a_3\cdot b_2 + a_0\cdot b_1 + a_1\cdot b_3 + a_2\cdot b_2 + a_3\cdot b_1 + a_1\cdot b_0 \\ c_2^{(4)} = c_1^{(3)} - c_3^{(3)}\cdot 0 + a_2\cdot b_0 = a_3\cdot b_3 + a_0\cdot b_2 + a_2\cdot b_3 + a_3\cdot b_2 + a_1\cdot b_1 + a_2\cdot b_0 \\ c_3^{(4)} = c_2^{(3)} - c_3^{(3)}\cdot 0 + a_3\cdot b_0 = a_0\cdot b_3 + a_3\cdot b_3 + a_1\cdot b_2 + a_2\cdot b_1 + a_3\cdot b_0 \end{cases}$$

Тогда получаем окончательную «прямую» формулу для вычисления кольцевой свертки $(a(x)\cdot b(x))\bmod(x^4+x+1)$, которая также является полевой сверткой, поскольку многочлен x^4+x+1 неприводим над полем $GF(2)$.

$$\begin{cases} (a(x)\cdot b(x))\bmod(x^4+x+1) = \sum\limits_{k=0}^{3} c_k\cdot x^k; \quad \deg(a(x)),\deg(b(x)) \le 3 \\ GF(2)\begin{cases} c_0 = a_0\cdot b_0 + a_1\cdot b_3 + a_2\cdot b_2 + a_3\cdot b_1 \\ c_1 = a_0\cdot b_1 + a_1\cdot b_0 + a_1\cdot b_3 + a_2\cdot b_2 + a_2\cdot b_3 + a_3\cdot b_1 + a_3\cdot b_2 \\ c_2 = a_0\cdot b_2 + a_1\cdot b_1 + a_2\cdot b_0 + a_2\cdot b_3 + a_3\cdot b_2 + a_3\cdot b_3 \\ c_3 = a_0\cdot b_3 + a_1\cdot b_2 + a_2\cdot b_1 + a_3\cdot b_0 + a_3\cdot b_3 \end{cases} \end{cases}$$

Забегая вперед, отметим, что полученная формула фактически позволяет выполнять «быстрое» умножение двух заданных элементов поля многочленов $GF(2^4)$, заданного над простым полем $GF(2)$ при помощи неприводимого многочлена $x^4 + x + 1$.

Теперь, располагая, «прямой» формулой для вычисления полевой свертки $(a(x) \cdot b(x)) \bmod (x^4 + x + 1)$, вычислим для примера полевую свертку многочленов $a(x) = x^3 + x^2 + 1$ и $b(x) = x^3 + x + 1$ над полем $GF(2)$.

Сначала для наглядности выпишем все коэффициенты многочленов $a(x)$ и $b(x)$: $a_0 = 1; a_1 = 0; a_2 = 1; a_3 = 1$ и $b_0 = 1; b_1 = 1; b_2 = 0; b_3 = 1$. Тогда подставляя их в «прямую» формулу получаем коэффициенты полевой свертки $\sum_{k=0}^{3} c_k \cdot x^k$:

$$GF(2)\begin{cases} c_0 = 1 \cdot 1 + 0 \cdot 1 + 1 \cdot 0 + 1 \cdot 1 = 0 \\ c_1 = 1 \cdot 1 + 0 \cdot 1 + 0 \cdot 1 + 1 \cdot 0 + 1 \cdot 1 + 1 \cdot 1 + 1 \cdot 0 = 1 \\ c_2 = 1 \cdot 0 + 0 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 + 1 \cdot 0 + 1 \cdot 1 = 1 \\ c_3 = 1 \cdot 1 + 0 \cdot 0 + 1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 = 0 \end{cases}$$

Таким образом, полевая свертка: $(a(x) \cdot b(x)) \bmod (x^4 + x + 1) = x^2 + x$.

Проверим полученный результат. Для этого вычислим свертку традиционным способом, перемножив многочлены $a(x)$ и $b(x)$, и вычислив остаток произведения по модулю многочлена $x^4 + x + 1$ путем деления «в столбик». Имеем произведение многочленов $a(x) \cdot b(x) = (x^3 + x^2 + 1) \cdot (x^3 + x + 1) = x^6 + x^5 + x^4 + x^3 + x^2 + x + 1$. Тогда остаток произведения по модулю многочлена $x^4 + x + 1$:

$$\begin{array}{r|l} 1 \cdot x^6 + 1 \cdot x^5 + 1 \cdot x^4 + 1 \cdot x^3 + 1 \cdot x^2 + 1 \cdot x + 1 & x^4 + x + 1 \\ \underline{1 \cdot x^6 + 0 \cdot x^5 + 0 \cdot x^4 + 1 \cdot x^3 + 1 \cdot x^2 + 0 \cdot x + 0} & \overline{x^2 + x + 1} \\ 1 \cdot x^5 + 1 \cdot x^4 + 0 \cdot x^3 + 0 \cdot x^2 + 1 \cdot x + 1 & \\ \underline{1 \cdot x^5 + 0 \cdot x^4 + 0 \cdot x^3 + 1 \cdot x^2 + 1 \cdot x + 0} & \\ 1 \cdot x^4 + 0 \cdot x^3 + 1 \cdot x^2 + 0 \cdot x + 1 & \\ \underline{1 \cdot x^4 + 0 \cdot x^3 + 0 \cdot x^2 + 1 \cdot x + 1} & \\ x^2 + x & \end{array}$$

Мы получили тот же самый результат: $(a(x) \cdot b(x)) \bmod (x^4 + x + 1) = x^2 + x$.

Поле многочленов над простым полем Галуа. Пусть задано простое поле Галуа *GF(p)*. Пусть задано некоторое целое число $m \geq 2$. Пусть задан некоторый нормированный неприводимый многочлен *m*-й степени *p(x)* над простым полем *GF(p)*. Пусть задано множество всевозможных многочленов-остатков по модулю неприводимого многочлена *p(x)* над простым полем *GF(p)*. Пусть задана бинарная операция сложения + многочленов, вычисляющая сумму двух многочленов с последующим взятием остатка суммы по модулю неприводимого многочлена *p(x)*. Пусть задана операция умножения · многочленов, вычисляющая произведение двух многочленов с последующим взятием остатка произведения по модулю неприводимого многочлена *p(x)*. Тогда, согласно общей алгебре, множество всевозможных многочленов-остатков по модулю неприводимого многочлена *p(x)* над простым полем *GF(p)* и заданные две бинарные операции сложения и умножения многочленов образуют **поле многочленов *GF(p^m)* над простым полем Галуа *GF(p)***. Согласно общей алгебре поле многочленов *GF(p^m)* также является алгебраическим расширением простого поля *GF(p)*, поэтому поле многочленов для краткости будем называть **расширенным полем Галуа *GF(p^m)***.

Примечание. В литературе иногда расширенное поле Галуа *GF(p^m)* так же, как и простое поле Галуа *GF(p)* обозначают в виде *GF(q)*. Например, расширенное поле *GF*(2^2) может быть обозначено, как *GF*(4), или, например, поле *GF*(3^2), как *GF*(9). Ошибки в таком обозначении нет, но это может ввести в заблуждении неопытного читателя, который по ошибке может решить, что речь идет о простом поле. Поэтому, если поле обозначено в виде *GF(q)*, всегда следует выяснять то, является *q* простым числом или некоторой целой степенью $m \geq 2$ некоторого простого числа *p*. В первом случае речь идет о простом поле, а во втором случае – о расширенном поле. Наконец, если *q* не является ни простым числом, ни какой-либо целой степенью $m \geq 2$ какого-либо простого числа *p*, то такого поля вообще не может существовать. Например, *GF*(17) – простое поле, поскольку 17 – простое число, *GF*(16) – расширенное поле, поскольку $16 = 2^4$, причем 2 – простое число. А, к примеру, поля *GF*(15) – вообще не может существовать по определению, поскольку 15 не является ни простым числом, ни какой-либо целой степенью $m \geq 2$ какого-либо простого числа.

Простейший пример расширенного поля – *GF*(2^2). Поле образовано на базе простого поля Галуа *GF*(2) и неприводимого многочлена $p(x) = x^2 + x + 1$. Множество элементов расширенного поля содержит $4 = 2^2$ многочлена: $\{0, 1, x, x+1\}$. Элементы расширенного поля являются всевозможными многочленами-остатками по модулю неприводимого многочлена $p(x) = x^2 + x + 1$. Нейтральным элементом по сложению (нулевым элементом) является многочлен $a(x) = 0$, а нейтральным элементом по умножению (единичным элементом) является многочлен $a(x) = 1$. Ниже приведены таблицы сложения и умножения для элементов расширенного поля *GF*(2^2), которые формируются путем вычисления сумм и произведений двух многочленов, принадлежащих полю *GF*(2^2), с последующим взятием остатка по модулю неприводимого многочлена $p(x)$.

$GF(2^2)$:

+	0	1	x	$x+1$
0	0	1	x	$x+1$
1	1	0	$x+1$	x
x	x	$x+1$	0	1
$x+1$	$x+1$	x	1	0

·	0	1	x	$x+1$
0	0	0	0	0
1	0	1	x	$x+1$
x	0	x	$x+1$	1
$x+1$	0	$x+1$	1	x

Из таблицы сложения нетрудно заметить, что для каждого элемента имеется обратный элемент по сложению – это сам элемент. Из таблицы умножения нетрудно заметить, что для каждого ненулевого элемента поля имеется обратный элемент по умножению: для элемента 1 – это элемент 1, для элемента x – это элемент $x+1$, для элемента $x+1$ – это элемент x.

Порядок поля равен 4, поскольку поле содержит 4 элемента, а характеристика поля равна 2, поскольку $1+1=0$, то есть уже два единичных элемента в сумме дают нуль.

Еще один пример расширенного поля – $GF(3^2)$. Поле образовано на базе простого поля Галуа $GF(3)$ и неприводимого многочлена $p(x) = x^2 + x + 2$. Множество элементов расширенного поля содержит $9 = 3^2$ многочленов: $\{0,1,2,x,x+1,x+2,2x,2x+1,2x+2\}$. Элементы расширенного поля являются всевозможными многочленами-остатками по модулю неприводимого многочлена $p(x) = x^2 + x + 2$. Нейтральным элементом по сложению (нулевым элементом) является многочлен $a(x) = 0$, а нейтральным элементом по умножению (единичным элементом) является многочлен $a(x) = 1$. Ниже приведены таблицы сложения и умножения для элементов расширенного поля $GF(3^2)$, которые формируются путем вычисления сумм и произведений двух многочленов, принадлежащих полю $GF(3^2)$, с последующим взятием остатка по модулю неприводимого многочлена $p(x)$.

$GF(3^2)$:

+	0	1	2	x	$x+1$	$x+2$	$2x$	$2x+1$	$2x+2$
0	0	1	2	x	$x+1$	$x+2$	$2x$	$2x+1$	$2x+2$
1	1	2	0	$x+1$	$x+2$	x	$2x+1$	$2x+2$	$2x$
2	2	0	1	$x+2$	x	$x+1$	$2x+2$	$2x$	$2x+1$
x	x	$x+1$	$x+2$	$2x$	$2x+1$	$2x+2$	0	1	2
$x+1$	$x+1$	$x+2$	x	$2x+1$	$2x+2$	$2x$	1	2	0
$x+2$	$x+2$	x	$x+1$	$2x+2$	$2x$	$2x+1$	2	0	1
$2x$	$2x$	$2x+1$	$2x+2$	0	1	2	x	$x+1$	$x+2$
$2x+1$	$2x+1$	$2x+2$	$2x$	1	2	0	$x+1$	$x+2$	x
$2x+2$	$2x+2$	$2x$	$2x+1$	2	0	1	$x+2$	x	$x+1$

$GF(3^2)$:

·	0	1	2	x	$x+1$	$x+2$	$2x$	$2x+1$	$2x+2$
0	0	0	0	0	0	0	0	0	0
1	0	1	2	x	$x+1$	$x+2$	$2x$	$2x+1$	$2x+2$
2	0	2	1	$2x$	$2x+2$	$2x+1$	x	$x+2$	$x+1$
x	0	x	$2x$	$2x+1$	1	$x+1$	$x+2$	$2x+2$	2
$x+1$	0	$x+1$	$2x+2$	1	$x+2$	$2x$	2	x	$2x+1$
$x+2$	0	$x+2$	$2x+1$	$x+1$	$2x$	2	$2x+2$	1	x
$2x$	0	$2x$	x	$x+2$	2	$2x+2$	$2x+1$	$x+1$	1
$2x+1$	0	$2x+1$	$x+2$	$2x+2$	x	1	$x+1$	2	$2x$
$2x+2$	0	$2x+2$	$x+1$	2	$2x+1$	x	1	$2x$	$x+2$

Из таблицы сложения нетрудно заметить, что для каждого элемента поля имеется обратный элемент по сложению: для элемента 0 – это сам элемент 0, для элемента 1 – элемент 2, для элемента 2 – элемент 1, для элемента x – элемент $2x$, и так далее, для элемента $2x+2$ – это элемент $x+1$. Из таблицы умножения нетрудно заметить, что для каждого ненулевого элемента поля имеется обратный элемент по умножению: для элемента 1 – это сам элемент 1, для элемента 2 – это сам элемент 2, для элемента x – элемент $x+1$ и так далее, для элемента $2x+2$ – элемент $2x$.

Порядок поля равен 9, поскольку поле содержит 9 элементов, а характеристика поля равна 3, поскольку $1+1+1=0$, то есть уже три единичных элемента в сумме дают нуль.

Арифметика и свойства расширенного поля. В расширенном поле $GF(p^m)$ по определению имеется взаимосвязь между операциями с элементами расширенного поля (сложения, умножения, а также вычисление обратного элемента по сложению и по умножению), и операциями с соответствующими многочленами над простым полем $GF(p)$:

- $\forall a,b \in GF(p^m) \Rightarrow \underbrace{a+b}_{GF(p^m)} = \underbrace{(a(x)+b(x)) \bmod p(x)}_{GF(p)}$.
- $\forall a,b \in GF(p^m) \Rightarrow \underbrace{a \cdot b}_{GF(p^m)} = \underbrace{(a(x) \cdot b(x)) \bmod p(x)}_{GF(p)}$. (1.9)
- $\forall a \in GF(p^m), \exists -a \in GF(p^m) : \underbrace{a+(-a)=0}_{GF(p^m)} \Leftrightarrow \underbrace{(a(x)+(-a(x))) \bmod p(x)=0}_{GF(p)}$.
- $\forall a \in GF(p^m) : a \neq 0, \exists a^{-1} \in GF(p^m) : \underbrace{a \cdot a^{-1}=1}_{GF(p^m)} \Leftrightarrow \underbrace{(a(x) \cdot a^{-1}(x)) \bmod p(x)=1}_{GF(p)}$.

Сложение элементов a и b расширенного поля $GF(p^m)$ сводится к сложению соответствующих многочленов $a(x)$ и $b(x)$, заданных над простым полем $GF(p)$. Заметим, что по определению расширенного поля в качестве результата сложения принимается остаток по модулю неприводимого многочлена $p(x)$ от суммы двух многочленов. Однако, учитывая то, что степень многочлена-суммы $a(x)+b(x)$ так же, как и степени многочленов-слагаемых всегда меньше степени $p(x)$, то $(a(x)+b(x)) \bmod p(x) = a(x)+b(x)$. Таким образом, при сложении элементов a и b расширенного поля, достаточно сложить соответствующие им многочлены $a(x)$ и $b(x)$, а вычисление остатка по модулю $p(x)$ от многочлена-суммы $a(x)+b(x)$ уже излишне. Вычислим, например, в расширенном поле $GF(2^2)$ сумму элементов x и $x+1$: $\underbrace{(1 \cdot x+0)+(1 \cdot x+1)}_{GF(2)} = \underbrace{(1+1)}_{GF(2)} \cdot x + \underbrace{(0+1)}_{GF(2)} = 0 \cdot x+1=1$.

Вычислим также в расширенном поле $GF(3^2)$ сумму элементов $2x+1$ и $2x+2$: $\underbrace{(2 \cdot x+1)+(2 \cdot x+2)}_{GF(3)} = \underbrace{(2+2)}_{GF(3)} \cdot x + \underbrace{(1+2)}_{GF(3)} = 1 \cdot x+0=x$.

Умножение элементов a и b в расширенном поле $GF(p^m)$ является гораздо более трудоемкой операцией, требующей вычисление произведения двух многочленов $a(x)$ и $b(x)$ над простым полем $GF(p)$. Кроме того, в большинстве случаях степень многочлена-произведения $a(x) \cdot b(x)$ оказывается большей либо равной степени неприводимого многочлена $p(x)$, и здесь уже не избежать вычисления остатка $a(x) \cdot b(x)$ по модулю $p(x)$, что само по себе является достаточно трудоемкой операцией. Вычислим в расширенном поле $GF(2^2)$ произведение элементов x и $x+1$: $\underbrace{((1 \cdot x+0) \cdot (1 \cdot x+1)) \bmod (x^2+x+1)}_{GF(2)} =$

$$\left(\underbrace{(1 \cdot 1)}_{GF(2)} \cdot x^2 + \underbrace{(1 \cdot 1+0 \cdot 1)}_{GF(2)} \cdot x + \underbrace{(0 \cdot 1)}_{GF(2)} \right) \bmod (x^2+x+1) = \left(\begin{array}{r|l} -1 \cdot x^2+1 \cdot x+0 & x^2+x+1 \\ \hline 1 \cdot x^2+1 \cdot x+1 & q(x)=1 \\ \hline r(x)=1 & \end{array} \right) = 1.$$ Найдем

в поле $GF(3^2)$ произведение элементов $2x+1$ и $2x+2$: $\underbrace{((2 \cdot x+1) \cdot (2 \cdot x+2)) \bmod (x^2+x+2)}_{GF(3)} =$

$$\left(\underbrace{(2 \cdot 2)}_{GF(3)} \cdot x^2 + \underbrace{(2 \cdot 2+1 \cdot 2)}_{GF(3)} \cdot x + \underbrace{(1 \cdot 2)}_{GF(3)} \right) \bmod (x^2+x+2) = \left(\begin{array}{r|l} -1 \cdot x^2+0 \cdot x+2 & x^2+x+2 \\ \hline 1 \cdot x^2+1 \cdot x+2 & q(x)=1 \\ \hline r(x)=2 \cdot x & \end{array} \right) = 2x.$$

Вычисление обратного элемента по сложению $-a$ расширенного поля $GF(p^m)$ сводится к нахождению соответствующего многочлена $-a(x)$ над простым полем $GF(p)$ такого, что $a(x)+(-a(x))=0$. Однако, поиск обратного элемента по сложению фактически сводится к вычислению обратных элементов по сложению для коэффициентов многочлена $a(x)$, то есть: $\underbrace{-a}_{GF(p^m)} \Leftrightarrow \underbrace{-a(x)}_{GF(p)} = (\underbrace{-a_{m-1}}_{GF(p)})\cdot x^{m-1}+\ldots+(\underbrace{-a_1}_{GF(p)})\cdot x+(\underbrace{-a_0}_{GF(p)})$. Отметим также, что поскольку в любом простом поле $GF(p)$ обратным элементом по сложению для нулевого элемента является сам нулевой элемент, то достаточно вычислять обратные элементы по сложению только для ненулевых коэффициентов многочлена $a(x)$. Вычислим, например, в расширенном поле $GF(2^2)$ обратный элемент по сложению для элемента $x+1$. Имеем, $\underbrace{-(1\cdot x+1)}_{GF(2)} = \underbrace{(-1)}_{GF(2)}\cdot x+\underbrace{(-1)}_{GF(2)} = 1\cdot x+1 = x+1$. То есть, обратным элементом по сложению для элемента $x+1$ является сам элемент $x+1$. Вычислим в расширенном поле $GF(3^2)$ обратный элемент по сложению для элемента $2x+1$: $\underbrace{-(2\cdot x+1)}_{GF(3)} = \underbrace{(-2)}_{GF(3)}\cdot x+\underbrace{(-1)}_{GF(3)} = 1\cdot x+2 = x+2$.

Примечание. Вообще говоря, в любом расширенном поле $GF(2^m)$, имеющем характеристику равную 2, для любого его элемента обратным элементом по сложению является сам элемент. Это следует из того, что коэффициенты соответствующих многочленов над простым полем $GF(2)$, являются элементами простого поля $GF(2)$, и обратными элементами по сложению для них являются сами элементы.

Наиболее трудоемкой операцией в расширенном поле $GF(p^m)$ является вычисление обратного элемента по умножению a^{-1}, поскольку это сводится к нахождению такого многочлена $b(x)=a^{-1}(x)$ над простым полем $GF(p)$, что $\underbrace{(a(x)\cdot b(x)) \bmod p(x)=1}_{GF(p)}$. В первом приближении подходящий обратный элемент a^{-1} можно найти полным перебором всех ненулевых элементов расширенного поля $GF(p^m)$. Однако, нетрудно заметить, что даже при небольших p и m, объем перебора будет слишком большим. Например, при $p = 5$ и $m = 10$, придется перебрать $5^{10}-1 \approx 10^7$ элементов. В то же время существует весьма эффективный алгоритм для нахождения обратного элемента по умножению в расширенном поле $GF(p^m)$, который базируется на расширенном алгоритме Евклида нахождения наибольшего общего делителя для двух многочленов. Остановимся на этом подробнее.

Итак, с одной стороны нам требуется найти $b(x)$ из условия $\underbrace{(a(x)\cdot b(x)) \bmod p(x)=1}_{GF(p)}$.

Мы можем это условие переписать в виде $\underbrace{a(x)\cdot b(x)-p(x)\cdot t(x)=1}_{GF(p)}$, где $t(x)$ - многочлен, являющийся частным от деления $a(x)\cdot b(x)$ на неприводимый многочлен $p(x)$. С другой стороны, мы можем применить расширенный алгоритм Евклида для многочленов $a(x)$ и $p(x)$, и найти такие многочлены $g(x)$ и $h(x)$, что $\underbrace{a(x)\cdot g(x)+p(x)\cdot h(x)=НОД(a(x),p(x))}_{GF(p)}$.

Теперь заметим, что многочлен $p(x)$ по определению не разлагается на многочлены меньшей степени (но не ниже 1) над полем $GF(p)$. Это означает, что наибольшим общим делителем для многочленов $a(x)$ и $p(x)$, может быть только некоторый ненулевой скаляр λ, являющийся элементом поля $GF(p)$. Тогда имеем $\underbrace{a(x)\cdot g(x)+p(x)\cdot h(x)=\lambda}_{GF(p)}$. Тогда нетрудно установить связь между этим уравнением и условием поиска обратного элемента:

$$\begin{cases} b(x)=\dfrac{1}{\lambda}\cdot g(x) \\ t(x)=-\dfrac{1}{\lambda}\cdot h(x) \end{cases};\quad \begin{matrix} a(x)\cdot g(x)+p(x)\cdot h(x)=\lambda \\ НОД(a(x),p(x))=\lambda \end{matrix};\quad \begin{matrix} a(x)\neq 0, a(x)\in GF(p^m) \\ 0\le \deg(a(x))\le m-1 \end{matrix};\quad \begin{matrix} \lambda\neq 0,\lambda\in GF(p) \\ \deg(p(x))=m \end{matrix}$$

Таким образом, используя расширенный алгоритм Евклида для многочленов $a(x)$ и $p(x)$, мы можем найти многочлены $g(x)$, $h(x)$ и наибольший общий делитель-скаляр λ, а затем уже легко получить искомый многочлен $b(x)$, разделив многочлен $g(x)$ на скаляр λ.

Рассмотрим теперь суть расширенного алгоритма Евклида для многочленов, который представляет собой итерационную процедуру с использованием рекуррентных соотношений:

$$\left.\begin{array}{c} r^{(0)}(x)=p(x)\quad g^{(0)}(x)=0\quad h^{(0)}(x)=1 \\ r^{(1)}(x)=a(x)\quad g^{(1)}(x)=1\quad h^{(1)}(x)=0 \\ s=2\ldots n: r^{(n)}(x)=0 \\ r^{(s-2)}(x)=q^{(s)}(x)\cdot r^{(s-1)}(x)+r^{(s)}(x)\Rightarrow \left\{\begin{matrix} q^{(s)}(x) \\ r^{(s)}(x) \end{matrix}\right\} \\ g^{(s)}(x)=g^{(s-2)}(x)-g^{(s-1)}(x)\cdot q^{(s)}(x) \\ h^{(s)}(x)=h^{(s-2)}(x)-h^{(s-1)}(x)\cdot q^{(s)}(x) \end{array}\right\} \Rightarrow \left\{\begin{array}{c} НОД(a(x),p(x))=r^{(n-1)}(x) \\ g(x)=g^{(n-1)}(x) \\ h(x)=h^{(n-1)}(x) \end{array}\right\} \qquad (1.10)$$

Ключевым рекуррентным соотношением является $r^{(s-2)}(x)=q^{(s)}(x)\cdot r^{(s-1)}(x)+r^{(s)}(x)$, с помощью которого при заданных начальных «остаточных» многочленах $r^{(0)}(x)=p(x)$ и $r^{(1)}(x)=a(x)$, на каждой итерации, начиная с итерации $s=2$, вычисляется очередной «остаточный» многочлен $r^{(s)}(x)$ и многочлен-частное $q^{(s)}(x)$ путем деления многочлена $r^{(s-2)}(x)$ на многочлен $r^{(s-1)}(x)$ над простым полем *GF(p)*. Рекуррентное вычисление «остаточного» многочлена $r^{(s)}(x)$ при помощи «остаточных» многочленов двух предыдущих итераций $r^{(s-1)}(x)$ и $r^{(s-2)}(x)$ является основой алгоритма Евклида, и именно оно используется для получения в конечном итоге многочлена, являющегося наибольшим общим делителем для двух многочленов $a(x)$ и $p(x)$. Кроме того, в расширенном алгоритме Евклида (в отличие от стандартного) также используется многочлен $q^{(s)}(x)$ для формирования многочленов $g(x)$ и $h(x)$ при помощи двух дополнительных рекуррентных соотношений: $g^{(s)}(x)=g^{(s-2)}(x)-g^{(s-1)}(x)\cdot q^{(s)}(x)$ и $h^{(s)}(x)=h^{(s-2)}(x)-h^{(s-1)}(x)\cdot q^{(s)}(x)$, причем $g^{(0)}(x)=0$, $g^{(1)}(x)=1$, $h^{(0)}(x)=1$ и $h^{(1)}(x)=0$. Алгоритм выполняется до тех пор, пока на некоторой итерации $s=n$ не будет получен нулевой «остаточный» многочлен $r^{(n)}(x)=0$. Тогда за наибольший общий делитель принимается «остаточный» многочлен предпоследней итерации: $НОД(a(x),p(x))=r^{(n-1)}(x)$, соответственно, в качестве многочленов $g(x)$ и $h(x)$ принимаются $g^{(n-1)}(x)$ и $h^{(n-1)}(x)$, также получаемые на предпоследней итерации.

Отметим, что в нашем случае для вычисления многочлена $b(x)=(1/\lambda)\cdot g(x)$, являющегося обратным по умножению для многочлена $a(x)$ в расширенном поле *GF(p^m)*, достаточно вычислить многочлен $g(x)$ и скаляр $\lambda=НОД(a(x),p(x))$, а вычисление многочлена $h(x)$ необязательно. Также заметим, что для того, чтобы $НОД(a(x),p(x))$ был скаляром (многочленом нулевой степени) необходимо и достаточно, чтобы многочлен $p(x)$ был неприводимым над простым полем *GF(p)*. Поскольку мы имеем дело с полем *GF(p^m)*, то многочлен $p(x)$ по определению поля является неприводимым, и для любого ненулевого многочлена $a(x)$, представляющего элемент поля *GF(p^m)*, существует многочлен $b(x)$, также представляющий элемент поля *GF(p^m)*, такой что $\underbrace{(a(x)\cdot b(x))\bmod p(x)=1}_{GF(p)}$.

Особо отметим, что в алгоритме вычисления обратного элемента по умножению в расширенном поле $GF(p^m)$, все операции с многочленами выполняются как с многочленами, заданными над простым полем $GF(p)$. Соответственно, все операции с коэффициентами многочленов при операциях с многочленами (в том числе делении многочлена на скаляр λ) выполняются по правилам арифметики простого поля $GF(p)$.

Рассмотрим теперь пару примеров вычисления обратных элементов по умножению в некоторых расширенных полях $GF(p^m)$.

Пример 1. Вычислим в поле $GF(2^4)$, образованного при помощи неприводимого многочлена $p(x)=x^4+x+1$, обратный элемент по умножению для элемента $a(x)=x^2+1$. Воспользуемся расширенным алгоритмом Евклида для нахождения $НОД(a(x),p(x))$ и $g(x)$.

Согласно алгоритму: $r^{(0)}(x)=x^4+x+1$, $r^{(1)}(x)=x^2+1$, $g^{(0)}(x)=0$ и $g^{(1)}(x)=1$.

Начинаем с итерации $s=2$, выполняем деление многочленов $r^{(0)}(x)=x^4+x+1$ и $r^{(1)}(x)=x^2+1$: $\begin{array}{r|l} 1\cdot x^4+0\cdot x^3+0\cdot x^2+1\cdot x+1 & x^2+1 \\ \underline{1\cdot x^4+0\cdot x^3+1\cdot x^2+0\cdot x+0} & \overline{q^{(2)}(x)=x^2+1} \\ 1\cdot x^2+1\cdot x+1 & \\ \underline{1\cdot x^2+0\cdot x+1} & \\ r^{(2)}(x)=x & \end{array}$, получаем $q^{(2)}(x)=x^2+1$ и $r^{(2)}(x)=x$. Кроме того, используя многочлен $q^{(2)}(x)$, также вычисляем $g^{(2)}(x)=g^{(0)}(x)-g^{(1)}(x)\cdot q^{(2)}(x)=0-1\cdot(x^2+1)=x^2+1$.

Переходим к итерации $s=3$, выполняем деление многочленов $r^{(1)}(x)=x^2+1$ и $r^{(2)}(x)=x$: $\begin{array}{r|l} 1\cdot x^2+0\cdot x+1 & x \\ \underline{1\cdot x^2+0\cdot x+0} & \overline{q^{(3)}(x)=x} \\ r^{(3)}(x)=1 & \end{array}$, получаем $q^{(3)}(x)=x$ и $r^{(3)}(x)=1$. Используя $q^{(3)}(x)$, также вычисляем $g^{(3)}(x)=g^{(1)}(x)-g^{(2)}(x)\cdot q^{(3)}(x)=1-(x^2+1)\cdot x=x^3+x+1$.

Переходим к итерации $s=4$, выполняем деление многочленов $r^{(2)}(x)=x$ и $r^{(3)}(x)=1$: $\begin{array}{r|l} 1\cdot x+0 & 1 \\ \underline{1\cdot x+0} & \overline{q^{(4)}(x)=x} \\ r^{(4)}(x)=0 & \end{array}$. Поскольку $r^{(4)}(x)=0$, то работа алгоритма на этом завершается. Тогда за результат работы алгоритма принимаются результаты вычислений на предпоследней итерации $s=3$: $g(x)=g^{(3)}(x)=x^3+x+1$, $\lambda=НОД(a(x),p(x))=r^{(3)}(x)=1$.

Поскольку $\lambda=1$, то обратный элемент по умножению $b(x)=(1/\lambda)\cdot g(x)=x^3+x+1$. Проверим полученный обратный элемент по умножению $b(x)=x^3+x+1$: $(a(x)\cdot b(x))\bmod p(x)=((x^2+1)\cdot(x^3+x+1))\bmod(x^4+x+1)=(x^5+x^2+x+1)\bmod(x^4+x+1)=$ $\left(\begin{array}{r|l} x^5+x^2+x+1 & x^4+x+1 \\ \underline{x^5+x^2+x+0} & \overline{x} \\ 1 & \end{array}\right)=1$. Таким образом $(a(x)\cdot b(x))\bmod p(x)=1$, а значит многочлен $b(x)=x^3+x+1$, является обратным элементом по умножению для элемента $a(x)=x^2+1$ в расширенном поле $GF(2^4)$.

Пример 2. Вычислим в поле $GF(3^4)$, образованного при помощи неприводимого многочлена $p(x)=x^4+x+2$, обратный элемент по умножению для элемента $a(x)=x^2+2\cdot x+1$. Воспользуемся расширенным алгоритмом Евклида для нахождения $НОД(a(x),p(x))$ и многочлена $g(x)$.

Согласно алгоритму: $r^{(0)}(x)=x^4+x+2$, $r^{(1)}(x)=x^2+2\cdot x+1$, $g^{(0)}(x)=0$ и $g^{(1)}(x)=1$.

Начинаем с итерации $s=2$, выполняем деление многочленов $r^{(0)}(x)=x^4+x+2$ и $r^{(1)}(x)=x^2+2\cdot x+1$:

$$\begin{array}{r|l} 1\cdot x^4+0\cdot x^3+0\cdot x^2+1\cdot x+2 & x^2+2\cdot x+1 \\ \underline{1\cdot x^4+2\cdot x^3+1\cdot x^2+0\cdot x+0} & \overline{q^{(2)}(x)=x^2+x} \\ 1\cdot x^3+2\cdot x^2+1\cdot x+2 & \\ \underline{1\cdot x^3+2\cdot x^2+1\cdot x+0} & \\ r^{(2)}(x)=2 & \end{array}$$

, получаем $q^{(2)}(x)=x^2+x$ и $r^{(2)}(x)=2$. Кроме того, используя многочлен $q^{(2)}(x)=x^2+x$, также вычисляем $g^{(2)}(x)=g^{(0)}(x)-g^{(1)}(x)\cdot q^{(2)}(x)=0-1\cdot(x^2+x)=2\cdot x^2+2\cdot x$.

Переходим к итерации $s=3$, выполняем деление многочленов $r^{(1)}(x)=x^2+2\cdot x+1$ и $r^{(2)}(x)=2$. Поскольку многочлен-делитель $r^{(2)}(x)=2$ является скаляром (многочленом нулевой степени), то, он нацело делит многочлен $r^{(1)}(x)=x^2+2\cdot x+1$, и остаток от деления будет равным нулю $r^{(3)}(x)=0$. На этом работа алгоритма завершается. В качестве результатов принимаются: $g(x)=g^{(2)}(x)=2\cdot x^2+2\cdot x$ и $\lambda=НОД(a(x),p(x))=r^{(2)}(x)=2$.

Тогда, поскольку, $\lambda=2$, то $b(x)=(1/\lambda)\cdot g(x)=(1/2)\cdot(2\cdot x^2+2\cdot x)=x^2+x$. Проверим полученный обратный элемент по умножению $b(x)=x^2+x$. Имеем, $(a(x)\cdot b(x))\bmod p(x)=$

$$((x^2+2\cdot x+1)\cdot(x^2+x))\bmod(x^4+x+2)=(x^4+x)\bmod(x^4+x+2)=\left(\begin{array}{r|l} x^4+x+0 & x^4+x+2 \\ \underline{x^4+x+2} & \overline{1} \\ 1 & \end{array}\right)=1.$$

Таким образом $(a(x)\cdot b(x))\bmod p(x)=1$, и многочлен $b(x)=x^2+x$, является обратным элементом по умножению для элемента $a(x)=x^2+2\cdot x+1$ в расширенном поле $GF(3^4)$.

Примечание 1. Заметим, что в полях $GF(2^m)$, являющихся расширением базового простого поля $GF(2)$ скаляр $\lambda=НОД(a(x),p(x))$ всегда равен 1 для любого ненулевого многочлена $a(x)$, поскольку скаляр сам по себе является ненулевым, а в поле $GF(2)$ существует только один ненулевой элемент – это 1.

Примечание 2. В случае если $a(x)$ является ненулевым скаляром a_0 (многочленом нулевой степени), алгоритм вычисления обратного элемента по умножению для элемента $a(x)$ в поле $GF(p^m)$ фактически сводится к вычислению обратного элемента по умножению для элемента a_0 в базовом простом поле $GF(p)$. В этом нетрудно убедиться, поскольку в этом случае алгоритм начинает работу при $r^{(0)}(x)=p(x)$, $r^{(1)}(x)=a(x)=a_0$, $g^{(0)}(x)=0$ и $g^{(1)}(x)=1$, и уже на начальной итерации $s=2$, остаток от деления $r^{(0)}(x)=p(x)$ на скаляр $r^{(1)}(x)=a_0$ даст нулевой многочлен $r^{(2)}(x)=0$. Тогда, согласно алгоритму, в качестве результатов будут приняты: $g(x)=g^{(1)}(x)=1$ и $\lambda=НОД(a(x),p(x))=r^{(1)}(x)=a_0$. Тогда, обратным элементом по умножению в поле $GF(p^m)$ будет элемент $b(x)=(1/\lambda)\cdot g(x)=(1/a_0)$.

Ниже на рисунке 1.3 приведена схема алгоритма для вычисления обратного элемента по умножению b для заданного ненулевого элемента a в расширенном поле $GF(p^m)$.

Отметим, что все операции многочленами выполняются как с многочленами, заданными над простым полем $GF(p)$. Соответственно, все операции с коэффициентами многочленов выполняются по правилам арифметики простого поля $GF(p)$.

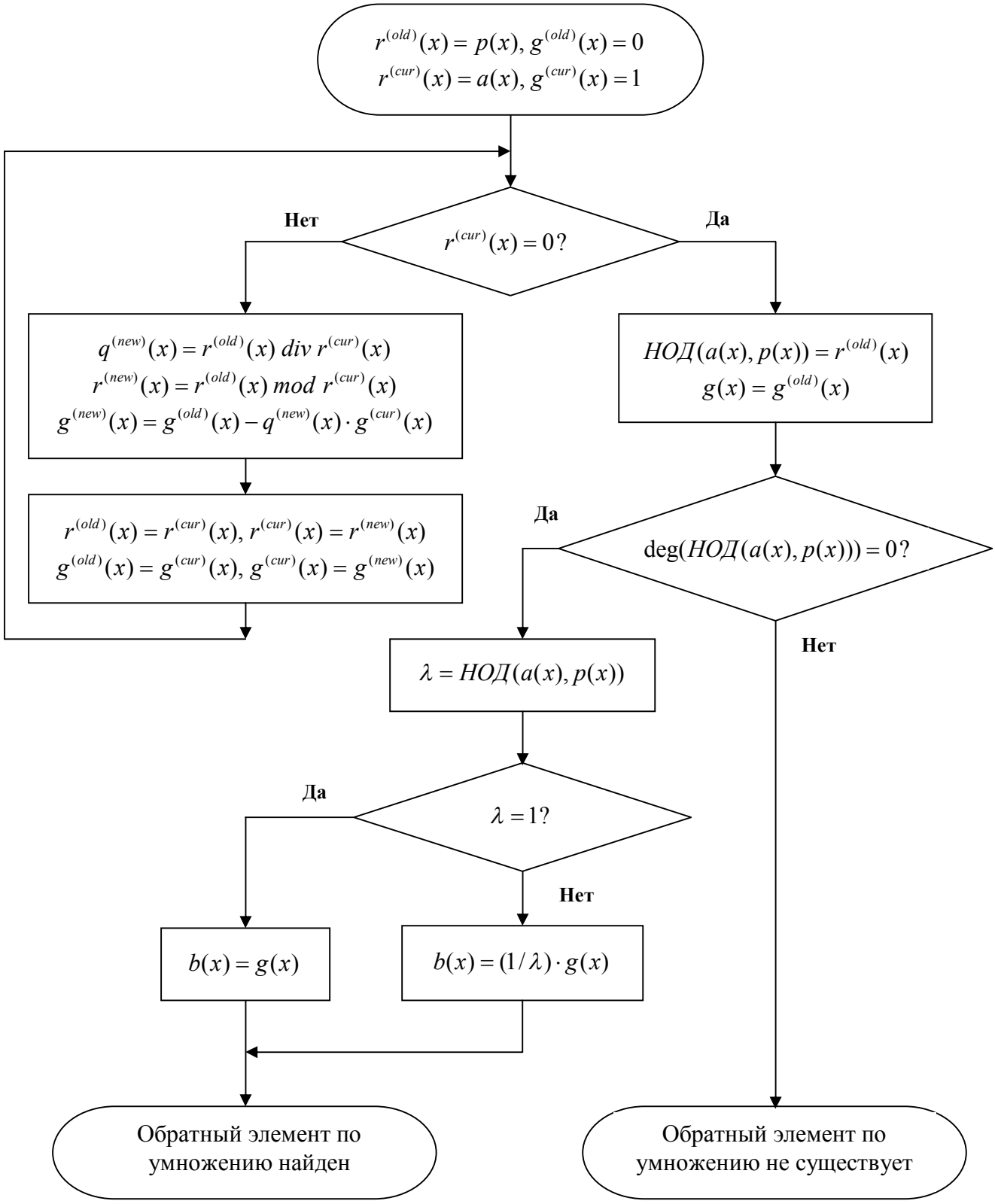

Рис. 1.3. Схема алгоритма вычисления обратного элемента по умножению для заданного ненулевого элемента в расширенном поле $GF(p^m)$.

Отметим теперь несколько важных свойств расширенного поля $GF(p^m)$.

Свойство 1. Порядок (число элементов) расширенного поля $GF(p^m)$ равен $q = p^m$. Характеристика расширенного поля $GF(p^m)$ равна характеристике соответствующего базового простого поля $GF(p)$, и равна простому числу p.

Свойство 2. Расширенное поле $GF(p^m)$ содержит в себе циклическую аддитивную группу порядка p, образованную при помощи порождающего элемента «1» и содержащую все элементы базового простого поля $GF(p)$.

Это свойство является прямым следствием того, что расширенное поле содержит в себе простое поле $GF(p)$, которое, в свою очередь, содержит циклическую аддитивную группу порядка p, образованную при помощи порождающего элемента «1».

Также особо отметим, что в расширенном поле $GF(p^m)$ не существует элемента, при помощи которого можно было бы породить циклическую аддитивную группу порядка $q = p^m$, содержащую все элементы поля $GF(p^m)$. Это обусловлено тем, что характеристика расширенного поля $GF(p^m)$ равна характеристике базового простого поля $GF(p)$ и равна p.

Свойство 3. Для любого элемента a расширенного поля $GF(p^m)$ и целого числа $n \geq 0$ сумма: $\overbrace{\underbrace{a+\ldots+a}_{n\ слагаемых}}^{GF(p^m)} = \overbrace{\lambda \cdot a}^{GF(p^m)}$, где $\lambda = \overbrace{n \bmod p}^{<R,\{+,\cdot\}>}$. В этом несложно убедиться. Согласно арифметике расширенного поля $GF(p^m)$, имеем: $\overbrace{\underbrace{a+\ldots+a}_{n\ слагаемых}}^{GF(p^m)} = \overbrace{(\underbrace{a(x)+\ldots+a(x)}_{n\ слагаемых}) \bmod p(x)}^{GF(p)}$

$$= \overbrace{\left(\underbrace{(a_{m-1}+\ldots+a_{m-1})}_{n\ слагаемых} \cdot x^{m-1} + \ldots + \underbrace{(a_1+\ldots+a_1)}_{n\ слагаемых} \cdot x + \underbrace{(a_0+\ldots+a_0)}_{n\ слагаемых} \right)}^{GF(p)} \bmod p(x).$$

Тогда, учитывая свойство простого поля $GF(p)$: $\underbrace{(a_j+\ldots+a_j)}_{n\ слагаемых} = \overbrace{\lambda \cdot a_j}^{GF(p)}$, где $\lambda = \overbrace{n \bmod p}^{<R,\{+,\cdot\}>}$, получаем:

$$= \overbrace{\left(\overbrace{(\lambda \cdot a_{m-1})}^{GF(p)} \cdot x^{m-1} + \ldots + \overbrace{(\lambda \cdot a_1)}^{GF(p)} \cdot x + \overbrace{\lambda \cdot a_0}^{GF(p)} \right)}^{GF(p)} \bmod p(x) = \overbrace{(\lambda \cdot (a_{m-1} \cdot x^{m-1} + \ldots + a_1 \cdot x + a_0)) \bmod p(x)}^{GF(p)}$$

$= \overbrace{\lambda \cdot a}^{GF(p^m)}$, где $\lambda = \overbrace{n \bmod p}^{<R,\{+,\cdot\}>}$. Заметим, что сам по себе множитель λ по сути является остатком по модулю простого p, и является элементом простого поля $GF(p)$. Его также можно интерпретировать как элемент расширенного поля $GF(p^m)$, поскольку с точки зрения многочленов он является скаляром (многочленом нулевой степени) и принадлежит полю многочленов $GF(p^m)$. Таким образом, сумма из n слагаемых a, являющихся элементом поля $GF(p^m)$ равна произведению элемента a на множитель λ, также интерпретируемый, как элемент поля $GF(p^m)$ и численно равный остатку числа слагаемых по модулю p.

Примечание. Данное свойство является прямым следствием того, что поле $GF(p^m)$ является конечным и имеет конечную характеристику p. Особо отметим, что если n кратно p, то $\lambda = n \bmod p = 0$, и тогда сумма из n слагаемых равна нулю для любого a. В частности, в полях $GF(2^m)$ характеристики $p = 2$, сумма для любого четного количества слагаемых равна нулю, а для нечетного количества сумма слагаемых равна самому слагаемому.

Свойство 4. Расширенное поле $GF(p^m)$ содержит в себе базовое простое поле $GF(p)$.

На примере таблиц сложения и таблиц умножения для расширенных полей $GF(2^2)$ и $GF(3^2)$ хорошо видно, что они содержат в себе «подтаблицы», являющиеся таблицами сложения и умножения для соответствующих простых полей $GF(2)$ и $GF(3)$.

Свойство 5. В расширенном поле $GF(p^m)$ содержится циклическая мультипликативная группа порядка $p^m - 1$, состоящая из всех ненулевых элементов поля, которая может быть образована при помощи примитивного элемента α, принадлежащего расширенного полю.

Ненулевые элементы расширенного поля являются соответствующими $0,1,\ldots,q-2$ степенями примитивного элемента α: $\{\alpha^0,\alpha^1,\ldots,\alpha^{q-2}\}$, где $q = p^m$. В расширенном поле может быть несколько примитивных элементов. Согласно общей алгебре любое поле Галуа, как расширенное, так и простое поле содержит хотя бы один примитивный элемент.

Следует особо отметить то, что то, какие именно элементы расширенного поля являются примитивными, напрямую зависит от неприводимого многочлена $p(x)$, с помощью которого задается расширенное поле $GF(p^m)$ над простым полем $GF(p)$. Если $p(x)$ является примитивным неприводимым многочленом, то одним из примитивных элементов расширенного поля является многочлен первой степени x, который является минимально возможным многочленом первой степени. Таким образом, использование примитивных неприводимых многочленов для формирования расширенного поля, избавляет впоследствии от необходимости поиска примитивного элемента. В дальнейшем мы будем говорить о расширенных полях, формируемых именно при помощи примитивных неприводимых многочленов. Соответственно, в качестве примитивного элемента мы по умолчанию всегда будем использовать одночлен $\alpha(x) = x$.

Рассмотрим в качестве примера расширенное поле $GF(2^2)$ порядка $q = p^m = 2^2 = 4$, сформированное на базе простого поля $GF(2)$ и примитивного неприводимого многочлена $p(x) = x^2 + x + 1$. Тогда, все ненулевые элементы поля можно представить, как степени примитивного элемента $\alpha(x) = x$: $\{\alpha^0,\alpha^1,\ldots,\alpha^{q-2}\} = \{\alpha^0,\alpha^1,\alpha^2\} = \{1, x, x\cdot x\}$. Тогда, используя таблицу умножения для расширенного поля $GF(2^2)$ окончательно получаем: $\{\alpha^0,\alpha^1,\alpha^2\} = \{1, x, x+1\}$. Особо отметим, что если продолжить дальше и вычислить 3-ю степень примитивного элемента, то получим $\alpha^3(x) = \alpha^2(x)\cdot\alpha(x) = (x+1)\cdot x$. Согласно таблице умножения имеем $(x+1)\cdot x = 1$. Отсюда получаем, что $\alpha^3(x) = \alpha^0(x) = 1$. Продолжая аналогично, можно получить, что $\alpha^4(x) = \alpha^1(x) = x$, $\alpha^5(x) = \alpha^2(x) = x+1$ и так далее. В итоге можно показать, что $\forall u, v \in Z: u \ge 0 \,\&\, v \ge 0 \,\&\, \underbrace{u = v \bmod 3}_{<R,+,\cdot>} \Leftrightarrow \underbrace{\alpha^u = \alpha^v}_{GF(2^2)}$:

$$GF(2^2): \begin{cases} \alpha^0(x) = \alpha^3(x) = \ldots = \alpha^{3k}(x) = 1 \\ \alpha^1(x) = \alpha^4(x) = \ldots = \alpha^{3k+1}(x) = x \\ \alpha^2(x) = \alpha^5(x) = \ldots = \alpha^{3k+2}(x) = x+1 \end{cases} ; \forall k \ge 0, k \in Z$$

Рассмотрим также расширенное поле $GF(3^2)$ порядка $q = p^m = 3^2 = 9$, сформированное на базе простого поля $GF(3)$ и примитивного неприводимого многочлена $p(x) = x^2 + x + 2$. Тогда, все ненулевые элементы поля можно представить, как степени примитивного элемента $\alpha(x) = x$: $\{\alpha^0,\alpha^1,\ldots,\alpha^{q-2}\} = \{\alpha^0,\alpha^1,\ldots,\alpha^7\} = \{1, x, x\cdot x, \ldots, x\cdot x\cdot x\cdot x\cdot x\cdot x\cdot x\}$. Используя таблицу умножения для расширенного поля $GF(3^2)$, получаем $x\cdot x = 2x+1$, $x\cdot x\cdot x = 2x+2$, $x\cdot x\cdot x\cdot x = 2$, $x\cdot x\cdot x\cdot x\cdot x = 2x$, $x\cdot x\cdot x\cdot x\cdot x\cdot x = x+2$, $x\cdot x\cdot x\cdot x\cdot x\cdot x\cdot x = x+1$. Тогда, можем записать: $\{\alpha^0,\alpha^1,\alpha^2,\alpha^3,\alpha^4,\alpha^5,\alpha^6,\alpha^7\} = \{1, x, 2x+1, 2x+2, 2, 2x, x+2, x+1\}$.

Особо отметим, что если продолжить последовательность и вычислить следующую, восьмую степень примитивного элемента, то получим $\alpha^8(x) = \alpha^7(x) \cdot \alpha(x) = (x+1) \cdot x$. Согласно таблице умножения имеем $(x+1) \cdot x = 1$. Отсюда получаем, что $\alpha^8(x) = \alpha^0(x) = 1$. Продолжая аналогично, можно получить, что $\alpha^9(x) = \alpha^1(x) = x$, $\alpha^{10}(x) = \alpha^2(x) = 2x+1$ и так далее. В итоге можно показать, что $\forall u, v \in Z : u \geq 0 \,\&\, v \geq 0 \,\&\, \underbrace{u = v \bmod 8}_{< R,+,\cdot >} \Leftrightarrow \underbrace{\alpha^u = \alpha^v}_{GF(3^2)}$:

$$GF(3^2): \begin{cases} \alpha^0(x) = \alpha^8(x) = \ldots = \alpha^{8k}(x) = 1 \\ \alpha^1(x) = \alpha^9(x) = \ldots = \alpha^{8k+1}(x) = x \\ \alpha^2(x) = \alpha^{10}(x) = \ldots = \alpha^{8k+2}(x) = 2x+1 \\ \alpha^3(x) = \alpha^{11}(x) = \ldots = \alpha^{8k+3}(x) = 2x+2 \\ \alpha^4(x) = \alpha^{12}(x) = \ldots = \alpha^{8k+4}(x) = 2 \\ \alpha^5(x) = \alpha^{13}(x) = \ldots = \alpha^{8k+5}(x) = 2x \\ \alpha^6(x) = \alpha^{14}(x) = \ldots = \alpha^{8k+6}(x) = x+2 \\ \alpha^7(x) = \alpha^{15}(x) = \ldots = \alpha^{8k+7}(x) = x+1 \end{cases} ; \forall k \geq 0, k \in Z$$

Таким образом, на примере рассмотренных выше двух небольших расширенных полей, видим, что расширенное поле $GF(p^m)$ содержит циклическую мультипликативную группу порядка $q - 1$, где $q = p^m$, состоящую из всех ненулевых элементов поля, причем их можно получить при помощи соответствующего примитивного элемента поля. Если расширенное поле задано при помощи примитивного неприводимого многочлена $p(x)$ над простым полем $GF(p)$, то одним из примитивных элементов является многочлен $\alpha(x) = x$. Кроме того, $\forall u, v \in Z : u \geq 0 \,\&\, v \geq 0 \,\&\, \underbrace{u = v \bmod (q-1)}_{< R,+,\cdot >} \Leftrightarrow \underbrace{\alpha^u = \alpha^v}_{GF(q)}$, где $q = p^m$:

$$GF(p^m): \begin{cases} \alpha^0 = \alpha^{(q-1)} = \ldots = \alpha^{((q-1)\cdot k)} \\ \alpha^1 = \alpha^{((q-1)+1)} = \ldots = \alpha^{((q-1)\cdot k+1)} \\ \vdots \\ \alpha^{q-2} = \alpha^{((q-1)+(q-2))} = \ldots = \alpha^{((q-1)\cdot k+(q-2))} \end{cases} ; \forall k \geq 0, k \in Z; \quad q = p^m$$

Таким образом, в любом расширенном поле $GF(p^m)$ существует примитивный элемент α, при помощи которого можно представить все ненулевые элементы расширенного поля как степени примитивного элемента: $\{\alpha^0, \alpha^1, \ldots, \alpha^{q-2}\}$, где $q = p^m$.

Логарифмы в арифметике расширенного поля. В расширенном поле $GF(p^m)$ так же, как и в простом поле, мы можем говорить о логарифмах по основанию примитивного элемента α для всех ненулевых элементов поля. Под логарифмом $\log_\alpha a$ от ненулевого элемента a по основанию примитивного поля α мы понимаем целое неотрицательное число u такое, что $\alpha^u = a$. Кроме того, мы считаем недопустимой попытку применения логарифма по основанию примитивного элемента α к нулевому элементу расширенного поля $GF(p^m)$: $\forall a \in GF(p^m) : a \neq 0, \exists u \in Z \,\&\, u \geq 0 : \alpha^u = a$ и $\log_\alpha 0 \Rightarrow$ *Ошибка*.

Согласно общей алгебре в расширенном поле $GF(p^m)$ так же, как и в простом поле, множество всевозможных логарифмов $\{0,1,\ldots,p^m-2\}$, с двумя бинарными операциями сложения и умножения по модулю p^m-1 (с точки зрения алгебры действительных чисел), образуют коммутативное кольцо с единицей, обозначаемое как $LR(p^m-1)$. Так же как и в случае кольца логарифмов для простого поля, арифметика кольца логарифмов для расширенного поля является «модулярной» с точки зрения алгебры действительных чисел:

- $\forall u \in R : u \in \{0,1,\ldots,p^m-2\} \Rightarrow u \in LR(p^m-1)$.
- $\forall u,v \in LR(p^m-1) \Rightarrow \underset{LR(p^m-1)}{\underbrace{u+v}} = \underset{<R,\{+,\cdot\}>}{\underbrace{(u+v)\bmod(p^m-1)}}$.
- $\forall u \in LR(p^m-1) \Rightarrow \underset{LR(p^m-1)}{\underbrace{-u}} = \underset{<R,\{+,\cdot\}>}{\underbrace{((p^m-1)-u)\bmod(p^m-1)}}$. (1.11)
- $\forall u,v \in LR(p^m-1) \Rightarrow \underset{LR(p^m-1)}{\underbrace{u\cdot v}} = \underset{<R,\{+,\cdot\}>}{\underbrace{(u\cdot v)\bmod(p^m-1)}}$.

Если требуется вычислить разность логарифмов u и v, то операцию вычитания нетрудно выразить через операцию сложения логарифма u с логарифмом $-v$, являющимся обратным элементом по сложению для логарифма v в кольце логарифмов $LR(p^m-1)$:

- $\forall u,v \in LR(p^m-1) \Rightarrow \underset{LR(p^m-1)}{\underbrace{w = u - v = u + (-v)}} = \underset{<R,\{+,\cdot\}>}{\underbrace{(u + ((p^m-1)-v))\bmod(p^m-1)}}$.

Как и в случае простых полей Галуа, в расширенных полях $GF(p^m)$ значимость логарифмов также огромна, поскольку они позволяют значительно упростить умножение элементов, вычисление обратных элементов по умножению, и, соответственно, деление элементов расширенного поля. Для расширенных полей применение логарифмов особенно актуально, поскольку операция умножения элементов связана с трудоемкой операцией перемножения многочленов с последующим вычислением многочлена-остатка по модулю неприводимого многочлена, а вычисление обратного элемента по умножению требует использования расширенного алгоритма Евклида для многочленов.

Тогда так же, как и в случае простых полей, операцию умножения двух ненулевых элементов a и b расширенного поля $GF(p^m)$ можно свести к сложению соответствующих логарифмов $u = \log_\alpha a$ и $v = \log_\alpha b$, являющихся элементами кольца $LR(p^m-1)$:

$$\left\{\begin{array}{c}\forall a,b \in GF(p^m) : a \neq 0 \,\&\, b \neq 0 \\ \exists \alpha^w \in GF(p^m) : \underset{GF(p^m)}{\underbrace{\alpha^w = a \cdot b}}\end{array}\right\} \Leftrightarrow \left\{\begin{array}{c}\forall u,v \in LR(p^m-1) : (u = \log_\alpha a) \,\&\, (v = \log_\alpha b) \\ \exists w \in LR(p^m-1) : \underset{LR(p^m-1)}{\underbrace{w = u + v}}\end{array}\right\}$$

В случае если один из элементов, a или b, равен нулю, то логарифм для нулевого элемента определить невозможно. Однако, в такой ситуации по определению самого поля произведение равно нулю. Иными словами: $\forall a,b \in GF(p^m) : a = 0 \vee b = 0 \Rightarrow a \cdot b = 0$.

Операцию вычисления обратного элемента по умножению для ненулевого элемента a расширенного поля $GF(p^m)$ можно свести к вычислению обратного элемента по сложению для соответствующего логарифма $u = \log_\alpha a$, являющегося элементом кольца $LR(p^m - 1)$:

$$\left\{\begin{array}{c} \forall a \in GF(p^m) : a \neq 0 \\ \exists \alpha^v \in GF(p^m) : \underbrace{\alpha^v = a^{-1}}_{GF(p^m)} \end{array}\right\} \Leftrightarrow \left\{\begin{array}{l} \forall u \in LR(p^m - 1) : (u = \log_\alpha a) \\ \exists v \in LR(p^m - 1) : \quad \underbrace{v = -u}_{LR(p^m - 1)} \end{array}\right\}$$

В случае если $a = 0$, то по определению самого поля не существует обратного элемента по умножению для нулевого элемента.

Операцию деления двух ненулевых элементов a и b расширенного поля $GF(p^m)$ можно выразить через операцию умножения элемента a на элемент b^{-1}, являющийся обратным по умножению для элемента b в расширенном поле $GF(p^m)$. Тогда операцию деления двух ненулевых элементов a и b можно свести к операции вычитания логарифмов $u = \log_\alpha a$ и $v = \log_\alpha b$, являющихся элементами кольца $LR(p^m - 1)$:

$$\left\{\begin{array}{c} \forall a,b \in GF(p^m) : a \neq 0 \,\&\, b \neq 0 \\ \exists \alpha^w \in GF(p^m) : \underbrace{\alpha^w = a/b}_{GF(p^m)} \end{array}\right\} \Leftrightarrow \left\{\begin{array}{c} \forall u,v \in LR(p^m - 1) : (u = \log_\alpha a) \,\&\, (v = \log_\alpha b) \\ \exists w \in LR(p^m - 1) : \quad \underbrace{w = u - v}_{LR(p^m - 1)} \end{array}\right\}$$

В случае если $b = 0$, то деление невозможно, поскольку по определению самого поля не существует обратного элемента по умножению для нулевого элемента. В случае же если $b \neq 0$, но при этом $a = 0$, тогда, хотя и логарифм для элемента a определить невозможно, но по определению самого поля произведение $a \cdot b^{-1}$ равно нулю, а значит $a/b = 0$.

Наконец, если ненулевой элемент a расширенного поля $GF(p^m)$ требуется возвести в степень v, являющуюся элементом кольца логарифмов $LR(p^m - 1)$, то такую операцию можно свести к умножению элемента $u = \log_\alpha a$ на элемент v в кольце логарифмов:

$$\left\{\begin{array}{c} \forall a \in GF(p^m) : a \neq 0 \\ \exists \alpha^w \in GF(p^m) : \underbrace{\alpha^w = a^v}_{GF(p^m)} \end{array}\right\} \Leftrightarrow \left\{\begin{array}{l} \forall u,v \in LR(p^m - 1) : u = \log_\alpha a \\ \exists w \in LR(p^m - 1) : \quad \underbrace{w = u \cdot v}_{LR(p^m - 1)} \end{array}\right\}$$

В случае если элемент $a = 0$, то логарифм для нулевого элемента определить невозможно. Однако в силу свойств самого поля возведение нулевого элемента в целую положительную степень, дает снова нуль, $0^v = 0, v > 0$, а возведение нулевого элемента в нулевую степень, дает единицу, $0^0 = 1$, по определению самого понятия степени.

Обобщая все вышесказанное, получаем: $\forall a \neq 0, b \neq 0 \in GF(p^m) \,\&\, \forall v \in LR(p^m - 1) \Rightarrow$

- $\underbrace{GF(p^m)}_{a \cdot b} = \alpha^{\overbrace{\log_\alpha a + \log_\alpha b}^{LR(p^m - 1)}} = \alpha^{\overbrace{(\log_\alpha a + \log_\alpha b) \bmod (p^m - 1)}^{< R, \{+,\cdot\} >}}$.
- $\underbrace{GF(p^m)}_{a^{-1}} = \alpha^{\overbrace{-\log_\alpha a}^{LR(p^m - 1)}} = \alpha^{\overbrace{((p^m - 1) - \log_\alpha a) \bmod (p^m - 1)}^{< R, \{+,\cdot\} >}}$. (1.12)
- $\underbrace{GF(p^m)}_{a/b} = \alpha^{\overbrace{\log_\alpha a - \log_\alpha b}^{LR(p^m - 1)}} = \alpha^{\overbrace{(\log_\alpha a + ((p^m - 1) - \log_\alpha b)) \bmod (p^m - 1)}^{< R, \{+,\cdot\} >}}$.
- $\underbrace{GF(p^m)}_{a^v} = \alpha^{\overbrace{v \cdot \log_\alpha a}^{LR(p^m - 1)}} = \alpha^{\overbrace{(v \cdot \log_\alpha a) \bmod (p^m - 1)}^{< R, \{+,\cdot\} >}}$.

Следует отметить, что «вычислительный эффект» от использования логарифмов в операциях умножения, вычисления обратного элемента по умножению, деления и возведения в заданную степень элементов расширенного поля можно получить только в том случае, если операция вычисления заданной степени примитивного элемента, то есть α^w, и операция вычисления логарифма от ненулевого элемента расширенного поля по основанию примитивного элемента, то есть $u = \log_\alpha a$, сами по себе выполняются достаточно быстро.

Максимальное быстродействие можно получить предварительно «подготовив» две таблицы – «таблицу степеней», в которой степени примитивного элемента упорядочены в порядке возрастания показателей степени, и «таблицу логарифмов», в которой логарифмы упорядочены в порядке возрастания элементов расширенного поля $GF(p^m)$. Тогда, вычисление заданной степени примитивного элемента сводится к простому извлечению результирующего элемента из «таблицы степеней», причем показатель заданной степени используется в качестве «индекса», указывающего на искомый элемент в таблице. Вычисление логарифма ненулевого элемента по основанию примитивного элемента сводится к простому извлечению результирующего логарифма из «таблицы логарифмов», причем сам элемент используется в качестве «индекса», указывающего на искомый логарифм в таблице.

Однако, здесь возникает некоторая трудность. Дело в том, с одной стороны в «таблице степеней» требуется хранить все ненулевые элементы расширенного поля, которые мы до сих пор представляли в виде соответствующих многочленов. Соответственно, требуется некоторый способ «компактного» представления многочленов в таблице, причем лучше в виде некоторого числового эквивалента многочлена. С другой стороны в «таблице логарифмов» ситуация еще более сложная, поскольку для извлечения требуемого логарифма сам элемент, который мы представляем в виде многочлена, должен использоваться в качестве «индекса» в «таблице логарифмов». Использовать многочлены в качестве индекса в таблице, очевидно, достаточно затруднительно, и здесь опять же было бы лучше, если каждому многочлену бы сопоставлен некоторый числовой эквивалент.

Таким образом, мы вплотную подошли к тому, что для удобства работы с «таблицей степеней» и «таблицей логарифмов» каждому ненулевому элементу расширенного поля $GF(p^m)$ необходимо поставить в соответствие некоторое число. Сделать это не так уж сложно, как кажется на первый взгляд.

Числовое представление элементов расширенного поля Галуа. Расширенное поле $GF(p^m)$ по определению состоит из множества всевозможных многочленов вида $a(x) = a_{m-1} \cdot x^{m-1} + \ldots + a_1 \cdot x + a_0$, являющихся множеством всевозможных многочленов-остатков по модулю некоторого неприводимого многочлена $p(x)$, причем коэффициенты $a_{m-1},\ldots,a_1,a_0$ являются элементами простого поля $GF(p)$.

Тогда возникает достаточно простая и очевидная идея рассматривать коэффициенты $a_{m-1},\ldots,a_1,a_0$ как «цифры» некоторого *m*-разрядного *p*-ичного числа. Кроме того, *p*-ичное число, в свою очередь, при необходимости мы всегда также можем перевести в десятичную систему счисления и обратно. На сегодняшний день существуют эффективные алгоритмы перевода чисел из одной системы счисления в другую, и они описаны в литературе [4].

Рассмотрим на примере расширенного поля $GF(2^2)$, образованного на базе простого поля $GF(2)$ и примитивного неприводимого многочлена $p(x) = x^2 + x + 1$, числовое представление элементов поля в двоичной и десятичной системе счисления:

$GF(2^2)$:

Элемент – многочлен $a(x)$:	0	1	x	$x+1$
Двоичный эквивалент $(a)_2$:	00	01	10	11
Десятичный эквивалент $(a)_{10}$:	0	1	2	3

Приведем таблицы сложения и умножения поля $GF(2^2)$ в десятичном представлении:

$GF(2^2)$:

+	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

·	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	3	1
3	0	3	1	2

Особо отметим, что приведенные таблицы сложения и умножения не имеют ничего общего с арифметикой по модулю 4. Более того, простого поля Галуа с арифметикой по модулю 4 вообще не существует по определению, поскольку число 4 не является простым. Здесь мы имеем с расширением второй степени простого поля $GF(2)$, и за каждым числом уже «скрывается» двучлен с коэффициентами, являющимися элементами поля $GF(2)$.

Рассмотрим также на примере расширенного поля $GF(3^2)$, образованного на базе простого поля $GF(3)$ и примитивного неприводимого многочлена $p(x) = x^2 + x + 2$, числовое представление элементов поля в троичной и десятичной системе счисления:

$GF(3^2)$:

Элемент – многочлен $a(x)$:	0	1	2	x	$x+1$	$x+2$	$2x$	$2x+1$	$2x+2$
Троичный эквивалент $(a)_3$:	00	01	02	10	11	12	20	21	22
Десятичный эквивалент $(a)_{10}$:	0	1	2	3	4	5	6	7	8

Приведем таблицы сложения и умножения поля $GF(3^2)$ в десятичном представлении:

$GF(3^2)$:

+	0	1	2	3	4	5	6	7	8
0	0	1	2	3	4	5	6	7	8
1	1	2	0	4	5	3	7	8	6
2	2	0	1	5	3	4	8	6	7
3	3	4	5	6	7	8	0	1	2
4	4	5	3	7	8	6	1	2	0
5	5	3	4	8	6	7	2	0	1
6	6	7	8	0	1	2	3	4	5
7	7	8	6	1	2	0	4	5	3
8	8	6	7	2	0	1	5	3	4

·	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8
2	0	2	1	6	8	7	3	5	4
3	0	3	6	7	1	4	5	8	2
4	0	4	8	1	5	6	2	3	7
5	0	5	7	4	6	2	8	1	3
6	0	6	3	5	2	8	7	4	1
7	0	7	5	8	3	1	4	2	6
8	0	8	4	2	7	3	1	6	5

Отметим также, что приведенные таблицы сложения и умножения не имеют ничего общего с арифметикой по модулю 9. Более того, простого поля Галуа с арифметикой по модулю 9 вообще не существует по определению, поскольку число 9 не является простым. Здесь мы имеем с расширением второй степени простого поля $GF(3)$, и за каждым числом уже «скрывается» двучлен с коэффициентами, являющимися элементами поля $GF(3)$.

Формирование таблиц степеней и логарифмов. При формировании «таблицы степеней» не обойтись без возведения примитивного элемента α последовательно во все степени $\{0,1,\ldots,p^m-2\}$. Используя соотношение $\alpha^k = \alpha \cdot \alpha^{k-1}; k = 1 \ldots p^m - 2; \alpha^0 = 1$, можно свести формирование «таблицы степеней» к последовательности из операций умножения на примитивный элемент. Однако, само по себе умножение требует перемножение соответствующих многочленов над полем $GF(p)$ с последующим вычислением остатка по модулю неприводимого многочлена $p(x)$.

Вычислительная трудоемкость формирования таблицы степеней во многом определяется многочленом, используемым в качестве примитивного элемента α расширенного поля $GF(p^m)$, который, в свою очередь, зависит от выбора неприводимого многочлена $p(x)$. Лучше всего выбирать примитивный неприводимый многочлен таким, чтобы в расширенном поле $GF(p^m)$ примитивным элементом являлся одночлен $\alpha(x) = x$.

Нетрудно заметить, что выполнять умножение заданного ненулевого многочлена $a(x) = a_{m-1} \cdot x^{m-1} + \ldots + a_1 \cdot x + a_0$ на одночлен $\alpha(x) = x$ гораздо легче, чем на какой-либо другой многочлен, поскольку в этом случае результат может быть легко получен: $x \cdot a(x) = a_{m-1} \cdot x^m + \ldots + a_1 \cdot x^2 + a_0 \cdot x$. Далее, вычисление остатка $x \cdot a(x)$ по модулю неприводимого многочлена $p(x)$ фактически сводится к вычитанию из многочлена $x \cdot a(x)$ неприводимого многочлена $p(x)$, умноженного на старший коэффициент a_{m-1}:

$$\left(\begin{array}{l} a_{m-1} \cdot x^m + a_{m-2} \cdot x^{m-1} + \ldots + a_0 \cdot x + 0 \\ \underline{-\; a_{m-1} \cdot x^m + a_{m-1} \cdot p_{m-1} \cdot x^{m-1} + \ldots + a_{m-1} \cdot p_1 \cdot x + a_{m-1} \cdot p_0} \\ 0 \cdot x^m + (a_{m-2} - a_{m-1} \cdot p_{m-1}) \cdot x^{m-1} + \ldots + (a_0 - a_{m-1} \cdot p_1) \cdot x + (-a_{m-1} \cdot p_0) \end{array} \middle| \begin{array}{l} p(x) \\ \hline a_{m-1} \end{array} \right)$$

Иными словами $(x \cdot a(x)) \bmod p(x) = x \cdot a(x) - a_{m-1} \cdot p(x)$. Также отметим, что в случаях, когда коэффициент a_{m-1} равен нулю, остатком $x \cdot a(x)$ по модулю многочлена $p(x)$ является сам многочлен $x \cdot a(x)$. Тогда, обобщая все вышесказанное, можно записать следующую процедуру формирования последовательности из степеней примитивного элемента α, являющегося одночленом x в расширенном поле $GF(p^m)$, заданного при помощи примитивного нормированного неприводимого многочлена $p(x)$:

$$\begin{cases} k = 1 \ldots p^m - 2; \quad \alpha^0(x) = 1 \\ \alpha^k(x) = \underbrace{x \cdot \alpha^{k-1}(x) - \alpha_{m-1}^{(k-1)} \cdot p(x)}_{GF(p)} \end{cases} \tag{1.13.1}$$

Выражая вычисление коэффициентов многочлена $\alpha^k(x)$ в явном виде, также имеем:

$$\begin{cases} k = 1 \ldots p^m - 2; \quad \alpha_0^{(0)} = 1; \quad \alpha_1^{(0)} = \ldots = \alpha_{m-1}^{(0)} = 0; \\ \alpha_i^{(k)} = \begin{cases} \underbrace{-\alpha_{m-1}^{(k-1)} \cdot p_0}_{GF(p)}; & i = 0 \\ \underbrace{\alpha_{i-1}^{(k-1)} - \alpha_{m-1}^{(k-1)} \cdot p_i}_{GF(p)}; & i = 1 \ldots m-1 \end{cases} \end{cases} \tag{1.13.2}$$

Нетрудно заметить, что на каждом шаге процедуры вычисление «очередной» степени $\alpha^k(x)$ фактически сводится к «сдвигу на одну позицию влево (в сторону старшего разряда)» коэффициентов «предыдущей» степени $\alpha^{k-1}(x)$ с дописыванием нуля в младший коэффициент, с последующим «поразрядным» вычитанием коэффициентов неприводимого многочлена $p(x)$, умноженных на старший коэффициент «предыдущей» степени $\alpha^{k-1}(x)$:

$$\begin{array}{lllll} & \alpha_{m-1}^{(k-1)} & \alpha_{m-2}^{(k-1)} & \ldots & \alpha_1^{(k-1)} & \alpha_0^{(k-1)} \\ & \leftarrow & \leftarrow & \leftarrow & \leftarrow & \leftarrow \\ \hline \alpha_{m-1}^{(k-1)} & \alpha_{m-2}^{(k-1)} & \alpha_{m-3}^{(k-1)} & \ldots & \alpha_0^{(k-1)} & 0 \\ \underline{-\; \alpha_{m-1}^{(k-1)} \cdot 1} & \underline{\alpha_{m-1}^{(k-1)} \cdot p_{m-1}} & \underline{\alpha_{m-1}^{(k-1)} \cdot p_{m-2}} & \ldots & \underline{\alpha_{m-1}^{(k-1)} \cdot p_1} & \underline{\alpha_{m-1}^{(k-1)} \cdot p_0} \\ 0 & \alpha_{m-1}^{(k)} & \alpha_{m-2}^{(k)} & \ldots & \alpha_1^{(k)} & \alpha_0^{(k)} \end{array} \tag{1.13.3}$$

Ниже на рис. 1.4 представлена схема алгоритма формирования таблицы степеней примитивного элемента $\alpha(x) = x$ в десятичном представлении и таблицы соответствующих логарифмов по основанию примитивного элемента для расширенного поля $GF(p^m)$.

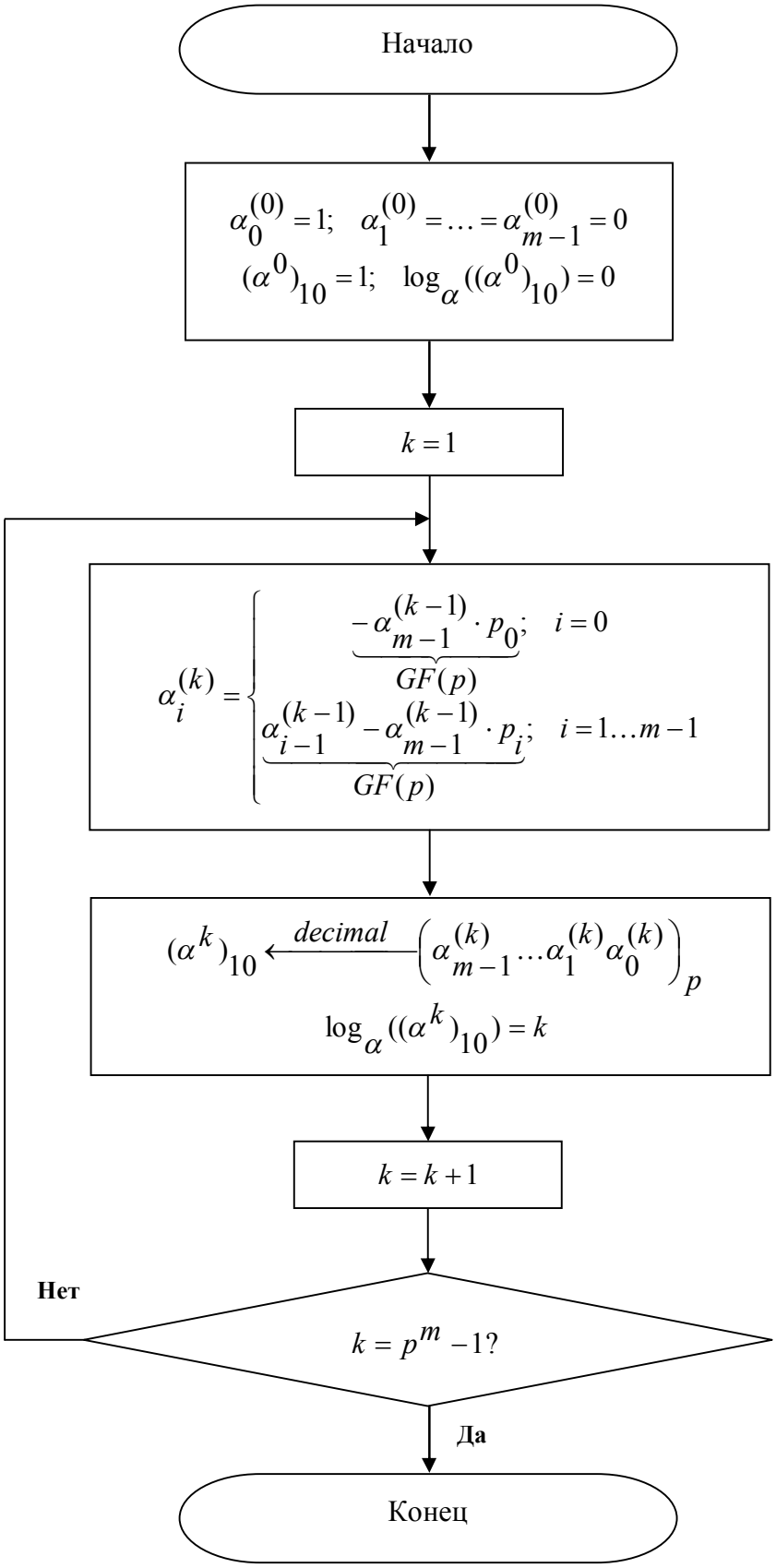

Рис. 1.4. Схема алгоритма формирования таблицы степеней примитивного элемента $\alpha(x) = x$ и таблицы соответствующих логарифмов для расширенного поля $GF(p^m)$.

Пример 1. Рассмотрим пример формирования «таблицы степеней» для расширенного поля $GF(2^2)$ заданного с помощью неприводимого многочлена $p(x)=x^2+x+1$, при котором примитивным элементом поля $GF(2^2)$ является элемент $\alpha(x)=x$. Тогда имеем:

$$\begin{cases} p=2;\quad m=2;\quad k=1\ldots 2;\quad p(x)=x^2+x+1;\quad \alpha^0(x)=1 \\ \alpha^1(x)=x\cdot\alpha^0(x)-\alpha_1^{(0)}\cdot p(x)=x\cdot 1-0\cdot(x^2+x+1)=x \\ \alpha^2(x)=x\cdot\alpha^1(x)-\alpha_1^{(1)}\cdot p(x)=x\cdot x-1\cdot(x^2+x+1)=x+1 \end{cases}$$

Тогда используя полученные степени примитивного элемента, а также вычислив десятичные эквиваленты для них (путем интерпретации коэффициентов многочленов как соответствующих цифр двоичных чисел, и перевода двоичных чисел в десятичную систему счисления), несложно сформировать «таблицу степеней» примитивного элемента.

В, свою очередь, благодаря наличию десятичных эквивалентов элементов поля, из «таблицы степеней» нетрудно сформировать «таблицу логарифмов» по основанию примитивного элемента. Для этого достаточно поменять местами первую и последнюю строки в «таблице степеней», после чего произвести перестановку столбцов так, чтобы элементы первой строки («десятичные эквиваленты») располагались в порядке возрастания.

$GF(2^2)$:

u	0	1	2
$\alpha^u(x)$	1	x	$x+1$
$(\alpha^u)_{10}$	1	2	3

$GF(2^2)$:

$(a)_{10}$	1	2	3
$a(x)$	1	x	$x+1$
$\log_\alpha a$	0	1	2

Пример 2. Рассмотрим пример формирования «таблицы степеней» для расширенного поля $GF(3^2)$ заданного с помощью неприводимого многочлена $p(x)=x^2+x+2$, при котором примитивным элементом поля $GF(3^2)$ является элемент $\alpha(x)=x$. Тогда имеем:

$$\begin{cases} p=3;\quad m=2;\quad k=1\ldots 7;\quad p(x)=x^2+x+2;\quad \alpha^0(x)=1 \\ \alpha^1(x)=x\cdot\alpha^0(x)-\alpha_1^{(0)}\cdot p(x)=x\cdot 1-0\cdot(x^2+x+1)=x \\ \alpha^2(x)=x\cdot\alpha^1(x)-\alpha_1^{(1)}\cdot p(x)=x\cdot x-1\cdot(x^2+x+2)=2x+1 \\ \alpha^3(x)=x\cdot\alpha^2(x)-\alpha_1^{(2)}\cdot p(x)=x\cdot(2x+1)-2\cdot(x^2+x+2)=2x+2 \\ \alpha^4(x)=x\cdot\alpha^3(x)-\alpha_1^{(3)}\cdot p(x)=x\cdot(2x+2)-2\cdot(x^2+x+2)=2 \\ \alpha^5(x)=x\cdot\alpha^4(x)-\alpha_1^{(4)}\cdot p(x)=x\cdot 2-0\cdot(x^2+x+2)=2x \\ \alpha^6(x)=x\cdot\alpha^5(x)-\alpha_1^{(5)}\cdot p(x)=x\cdot(2x)-2\cdot(x^2+x+2)=x+2 \\ \alpha^7(x)=x\cdot\alpha^6(x)-\alpha_1^{(6)}\cdot p(x)=x\cdot(x+2)-1\cdot(x^2+x+2)=x+1 \end{cases}$$

Тогда используя полученные степени примитивного элемента, а также вычислив для них десятичные эквиваленты, получаем «таблицу степеней примитивного элемента:

$GF(3^2)$:

u	0	1	2	3	4	5	6	7
$\alpha^u(x)$	1	x	$2x+1$	$2x+2$	2	$2x$	$x+2$	$x+1$
$(\alpha^u)_{10}$	1	3	7	8	2	6	5	4

Соответственно, из «таблицы степеней» также получаем «таблицу логарифмов» по основанию примитивного элемента:

$GF(3^2)$:

$(a)_{10}$	1	2	3	4	5	6	7	8
$a(x)$	1	2	x	$x+1$	$x+2$	$2x$	$2x+1$	$2x+2$
$\log_\alpha a$	0	4	1	7	6	5	2	3

Арифметика расширенного поля Галуа при числовом представлении элементов. При числовом представлении элементов расширенного поля для компактности хранения и наглядности отображения элементов обычно используется представление в десятичной системе счисления. Это к тому же делает простым и удобным работу с «таблицей логарифмов», поскольку само десятичное число может использоваться в качестве «индекса» для выборки требуемого «логарифма» из таблицы. В свою очередь, в «таблице степеней» вместо массивов коэффициентов многочленов хранится компактное десятичное число.

Однако, при десятичном представлении элементов расширенного поля $GF(p^m)$, очевидно, возникает небольшой нюанс при сложении и вычитании элементов. По определению сложение и вычитание многочленов сводится к сложению и вычитанию соответствующих коэффициентов многочленов по правилам арифметики базового простого поля $GF(p)$, поскольку коэффициенты являются элементами базового простого поля:

$$\underbrace{a \pm b}_{GF(p^m)} = \underbrace{(a(x) \pm b(x)) \bmod p(x)}_{GF(p)} = \underbrace{(a_{m-1} \pm b_{m-1})}_{GF(p)} \cdot x^{m-1} + \ldots + \underbrace{(a_1 \pm b_1)}_{GF(p)} \cdot x + \underbrace{(a_0 \pm b_0)}_{GF(p)}$$

В то же время коэффициенты многочленов мы всегда можем также интерпретировать как соответствующие цифры некоторого *m*-разрядного числа в *p*-ичной системы счисления. Также отметим, что мы всегда может перевести число из *p*-ичной системы счисления в десятичную систему и обратно. Тогда сложение/вычитание элементов расширенного поля $GF(p^m)$ по сути сводится к «поразрядному» сложению/вычитанию соответствующих чисел в *p*-ичной системы счисления. Слово «поразрядное» здесь играет ключевую роль, в каждом *k*-м разряде операция сложения/вычитания соответствующих цифр, являющихся по сути *k*-ми коэффициентами складываемых/вычитаемых многочленов, выполняется независимо от других разрядов, никаких переносов/займов здесь не может быть. Сложение/вычитание цифр выполняется по правилам арифметики базового простого поля $GF(p)$, поскольку они фактически представляют собой коэффициенты, которые являются элементами поля $GF(p)$.

$$\underbrace{a \pm b}_{GF(p^m)} = c \Leftrightarrow \left\{ \begin{array}{c|c|c|c} \left.\begin{matrix} a_{m-1} \\ \pm \\ b_{m-1} \end{matrix}\right\} GF(p) & \ldots & \left.\begin{matrix} a_1 \\ \pm \\ b_1 \end{matrix}\right\} GF(p) & \left.\begin{matrix} a_0 \\ \pm \\ b_0 \end{matrix}\right\} GF(p) \\ \hline c_{m-1} & \ldots & c_1 & c_0 \end{array} \right\}$$

Очевидно, что если элементы хранятся в виде чисел десятичной системы счисления, то для сложения/вычитания двух заданных элементов их предварительно необходимо перевести в соответствующую *p*-ичную систему счисления, затем выполнить операцию поразрядного сложения/вычитания, а затем результат перевести в десятичную систему.

Пример 1. Найдем сумму элементов расширенного поля $GF(3^2)$, представленных в виде соответствующих чисел «5» и «6» в десятичной системе счисления. Имеем: $\underbrace{(5)_{10} + (6)_{10}}_{GF(3^2)} = \underbrace{(12)_3 + (20)_3}_{GF(3^2)} = \left\{ \begin{array}{c|c} \left.\begin{matrix} 1 \\ + \\ 2 \end{matrix}\right\} GF(3) & \left.\begin{matrix} 2 \\ + \\ 0 \end{matrix}\right\} GF(3) \\ \hline 0 & 2 \end{array} \right\} = (02)_3 = (2)_{10}$. По таблице сложения для расширенного поля $GF(3^2)$ в десятичном представлении, видим, что результат верный.

Пример 2. Найдем разность элементов расширенного поля $GF(3^2)$, представленных в виде соответствующих чисел «4» и «7» в десятичной системе счисления. Имеем: $\underbrace{(4)_{10} - (7)_{10}}_{GF(3^2)} = \underbrace{(11)_3 - (21)_3}_{GF(3^2)} = \left\{ \begin{array}{c|c} \left.\begin{matrix} 1 \\ - \\ 2 \end{matrix}\right\} GF(3) & \left.\begin{matrix} 1 \\ - \\ 1 \end{matrix}\right\} GF(3) \\ \hline 2 & 0 \end{array} \right\} = (20)_3 = (6)_{10}$. По таблице сложения для расширенного поля $GF(3^2)$ в десятичном представлении видим, что $\underbrace{(6)_{10} + (7)_{10}}_{GF(3^2)} = (4)_{10}$.

Умножение/деление ненулевых элементов расширенного поля $GF(p^m)$ быстрее всего можно выполнять, используя сложение/вычитание соответствующих логарифмов по основанию примитивного элемента α при условии, что заранее сформирована «таблица логарифмов» и «таблица степеней». Сложение/вычитание логарифмов выполняется согласно арифметике кольца логарифмов $LR(p^m-1)$, которое с точки зрения традиционного алгебры действительных чисел фактически сводится к сложению/вычитания по модулю p^m-1:

$$\underbrace{GF(p^m)}_{a\cdot b} = \alpha^{\overbrace{\log_\alpha a+\log_\alpha b}^{LR(p^m-1)}} = \alpha^{\overbrace{(\log_\alpha a+\log_\alpha b)\bmod(p^m-1)}^{<R,\{+,\cdot\}>}}$$

$$\underbrace{GF(p^m)}_{a/b} = \alpha^{\overbrace{\log_\alpha a-\log_\alpha b}^{LR(p^m-1)}} = \alpha^{\overbrace{(\log_\alpha a+((p^m-1)-\log_\alpha b))\bmod(p^m-1)}^{<R,\{+,\cdot\}>}}$$

Аналогично, вычисление обратного элемента по умножению в расширенном поле $GF(p^m)$ для ненулевого элемента сводится к вычислению обратного элемента по сложению для соответствующего логарифма в кольце логарифмов $LR(p^m-1)$.

$$\underbrace{GF(p^m)}_{a^{-1}} = \alpha^{\overbrace{-\log_\alpha a}^{LR(p^m-1)}} = \alpha^{\overbrace{((p^m-1)-\log_\alpha a)\bmod(p^m-1)}^{<R,\{+,\cdot\}>}}$$

Наконец, возведение ненулевого элемента расширенного поля $GF(p^m)$ в заданную степень v сводится к умножению соответствующего логарифма на заданную степень v в кольце логарифмов $LR(p^m-1)$.

$$\underbrace{GF(p^m)}_{a^v} = \alpha^{\overbrace{v\cdot\log_\alpha a}^{LR(p^m-1)}} = \alpha^{\overbrace{(v\cdot\log_\alpha a)\bmod(p^m-1)}^{<R,\{+,\cdot\}>}}$$

Пример 3. Найдем произведение элементов поля $GF(3^2)$, представленных в виде соответствующих чисел «7» и «8» в десятичной системе счисления. По «таблице логарифмов» для поля $GF(3^2)$ имеем логарифмы элементов: $\log_\alpha(7)_{10}=2$ и $\log_\alpha(8)_{10}=3$.

Тогда получаем: $\overbrace{(7)_{10}\cdot(8)_{10}}^{GF(3^2)} = \alpha^{\overbrace{(2+3)\bmod(8)}^{<R,\{+,\cdot\}>}} = \alpha^5$. По «таблице степеней» для поля $GF(3^2)$ имеем $\alpha^5=(6)_{10}$. В итоге получаем: $\overbrace{(7)_{10}\cdot(8)_{10}}^{GF(3^2)}=(6)_{10}$. По таблице умножения для поля $GF(3^2)$ в десятичном представлении нетрудно проверить, что результат верный.

Пример 4. Найдем отношение элементов поля $GF(3^2)$, представленных в виде соответствующих чисел «3» и «4» в десятичной системе счисления. По «таблице логарифмов» для поля $GF(3^2)$ имеем логарифмы элементов: $\log_\alpha(3)_{10}=1$ и $\log_\alpha(4)_{10}=7$.

Тогда получаем: $\overbrace{(3)_{10}/(4)_{10}}^{GF(3^2)} = \alpha^{\overbrace{(1+(8-7))\bmod(8)}^{<R,\{+,\cdot\}>}} = \alpha^2$. По «таблице степеней» для поля $GF(3^2)$ имеем $\alpha^2=(7)_{10}$. В итоге получаем: $\overbrace{(3)_{10}/(4)_{10}}^{GF(3^2)}=(7)_{10}$. По таблице умножения для поля $GF(3^2)$ в десятичном представлении нетрудно проверить: $\overbrace{(7)_{10}\cdot(4)_{10}}^{GF(3^2)}=(3)_{10}$.

Пример 5. Найдем обратный элемент по умножению в поле $GF(3^2)$ для элемента, представленного в виде числа «7» в десятичной системе счисления. По «таблице логарифмов» для поля $GF(3^2)$ имеем логарифм элемента: $\log_\alpha (7)_{10} = 2$. Тогда получаем $\overbrace{((7)_{10})^{-1}}^{GF(3^2)} = \alpha^{\overbrace{(8-2)\bmod(8)}^{<R,\{+,\cdot\}>}} = \alpha^6$. В итоге по «таблице степеней» для поля $GF(3^2)$ имеем $\alpha^6 = (5)_{10}$. По таблице умножения для поля $GF(3^2)$ видим, что $\overbrace{(7)_{10} \cdot (5)_{10}}^{GF(3^2)} = (1)_{10}$.

Пример 6. Возведем элемент расширенного поля $GF(3^2)$, представленный в виде числа «4» в десятичной системе счисления в степень v = 3. По «таблице логарифмов» для поля $GF(3^2)$ имеем логарифм элемента: $\log_\alpha (4)_{10} = 7$. Тогда получаем следующее: $\overbrace{((4)_{10})^3}^{GF(3^2)} = \alpha^{\overbrace{(3\cdot 7)\bmod(8)}^{<R,\{+,\cdot\}>}} = \alpha^5$. В итоге по «таблице степеней» для поля $GF(3^2)$ имеем $\alpha^5 = (6)_{10}$. По таблице умножения для поля $GF(3^2)$ видим, что $\overbrace{(4)_{10} \cdot (4)_{10} \cdot (4)_{10}}^{GF(3^2)} = (6)_{10}$.

Обобщая все вышесказанное, в итоге имеем следующую арифметику расширенного поля $GF(p^m)$, выраженную через логарифмы по основанию примитивного элемента, степени примитивного элемента и арифметику поля действительных чисел:

- $\overbrace{a+b}^{GF(p^m)} = \overbrace{(a_{m-1}\ldots a_0)_p + (b_{m-1}\ldots b_0)_p}^{GF(p^m)} = (\overbrace{(a_{m-1}+b_{m-1})\bmod p}^{<R,\{+,\cdot\}>}\ldots\overbrace{(a_0+b_0)\bmod p}^{<R,\{+,\cdot\}>})_p$
- $\overbrace{a-b}^{GF(p^m)} = \overbrace{(a_{m-1}\ldots a_0)_p - (b_{m-1}\ldots b_0)_p}^{GF(p^m)} = (\overbrace{(a_{m-1}+p-b_{m-1})\bmod p}^{<R,\{+,\cdot\}>}\ldots\overbrace{(a_0+p-b_0)\bmod p}^{<R,\{+,\cdot\}>})_p$
- $\overbrace{a\cdot b}^{GF(p^m)} = \begin{cases} \alpha^{\overbrace{(\log_\alpha a + \log_\alpha b)\bmod(p^m-1)}^{<R,\{+,\cdot\}>}} & a\neq 0 \,\&\, b\neq 0 \\ 0 & a=0 \vee b=0 \end{cases}$
- $\overbrace{a/b}^{GF(p^m)} = \begin{cases} \alpha^{\overbrace{(\log_\alpha a + ((p^m-1)-\log_\alpha b))\bmod(p^m-1)}^{<R,\{+,\cdot\}>}} & a\neq 0 \,\&\, b\neq 0 \\ 0 & a=0 \,\&\, b\neq 0 \\ \textit{Ошибка} & b=0 \end{cases}$ (1.14)
- $\overbrace{a^{-1}}^{GF(p^m)} = \begin{cases} \alpha^{\overbrace{((p^m-1)-\log_\alpha a)\bmod(p^m-1)}^{<R,\{+,\cdot\}>}} & a\neq 0 \\ \textit{Ошибка} & a=0 \end{cases}$
- $\overbrace{a^v}^{GF(p^m)} = \begin{cases} \alpha^{\overbrace{(v\cdot\log_\alpha a)\bmod(p^m-1)}^{<R,\{+,\cdot\}>}} & a\neq 0 \,\&\, v\geq 0 \\ 0 & a=0 \,\&\, v>0 \\ 1 & a=0 \,\&\, v=0 \end{cases}$

Примечание. Особо отметим, что при m = 1, арифметика расширенного поля $GF(p^m)$ полностью соответствует арифметике простого поля $GF(p)$.

Контрольные вопросы

1. Что такое алгебраическая структура?

2. Какое ключевое свойство отличает поле от кольца?

3. Какие поля называют полями Галуа *GF(p)*? Что такое характеристика поля?

4. Определите порядки элементов 12, 13, 14, 15 в поле *GF*(23).

5. Найдите мультипликативный обратный элемент для элемента 13 в поле *GF*(23).

6. Какие элементы поля называют примитивными?

7. Найдите наименьший примитивный элемент в поле *GF*(109).

8. Каким образом в арифметике поля используется кольцо логарифмов элементов поля по основанию примитивного элемента, и какие преимущества это дает?

9. Сформируйте таблицу степеней и таблицу логарифмов для поля *GF*(23), используя элемент 5 в качестве примитивного элемента.

10. Вычислите значение выражения $(13^6 * 15^5 + 12^3 / 14^7)$ в поле *GF*(23).

11. Что такое многочлен над полем *GF*(*p*)? Как определяется степень многочлена?

12. Выполните деление многочлена $x^8 + 1$ на многочлен $x^4 + x + 1$, заданного над полем *GF*(5), используя сдвиговую схему деления многочленов.

13. Найти усеченную линейную, циклическую и полевую свертки многочленов $x^3 + x + 3$ и $x^3 + x^2 + 2$, заданных над полем *GF*(5), используя соответствующие многочлены-модули: x^4, $x^4 + 4$ и $x^4 + x + 4$.

14. Определите, над каким из полей: *GF*(2), *GF*(3), *GF*(5), *GF*(7), *GF*(11), *GF*(13), многочлен $x^4 + x + 1$ является неприводимым.

15. Найдите мультипликативный обратный элемент для элемента $2x + 1$ в поле $GF(5^2)$, заданного при помощи неприводимого многочлена $x^2 + x + 2$.

16. Найдите наименьший примитивный элемент в поле $GF(7^2)$, заданного при помощи неприводимого многочлена $x^2 + 2x + 2$.

17. Что такое числовое представление элементов поля $GF(p^m)$ и как оно используется в арифметике поля совместно с арифметикой кольца логарифмов элементов поля?

18. Определите, какие многочлены соответствуют элементам поля $GF(5^4)$, заданным в числовом представлении в десятичной системе счисления, как: 91, 154, 407.

19. Вычислите значение выражения $(13^6 * 15^5 + 12^3 / 14^7)$ в поле $GF(5^2)$, где элементы поля заданы в числовом представлении в десятичной системе счисления, а само поле задано при помощи неприводимого многочлена $x^2 + x + 2$.

2. Конечные поля GF(2^m). Аппаратная реализация арифметики поля.

Расширенные конечные поля Галуа $GF(2^m)$ характеристики $p=2$ являются важнейшим частным случаем расширенных конечных полей $GF(p^m)$ и имеют огромное практическое значение в современных цифровых системах.

Отметим, что в базовом простом поле $GF(2)$ для элемента 0 обратным элементом по сложению является сам элемент 0, также как и для элемента 1 обратным элементом по сложению является сам элемент 1. Иными словами: $\forall a \in GF(2) \Rightarrow a = -a$. Кроме того, фактически сложение элементов простого поля $GF(2)$ эквивалентно операции **исключающего ИЛИ**, т.е. операции **XOR**. А учитывая то, что обратный элемент по сложению простого поля $GF(2)$ равен самому элементу, то справедливо следующее: $\forall a,b \in GF(2) \Rightarrow a-b=a+(-b)=a+b=a \oplus b$. Символом $\oplus$ мы обозначаем операцию XOR.

Тогда, при сложении / вычитании элементов расширенного поля $GF(2^m)$ мы имеем:

$$\underbrace{a \pm b}_{GF(2^m)} = \underbrace{(a(x) \pm b(x)) \bmod p(x)}_{GF(2)} = \underbrace{(a_{m-1} \pm b_{m-1})}_{GF(2)} x^{m-1} + \ldots + \underbrace{(a_1 \pm b_1)}_{GF(2)} x + \underbrace{(a_0 \pm b_0)}_{GF(2)} =$$
$$= (a_{m-1} \oplus b_{m-1}) x^{m-1} + \ldots + (a_1 \oplus b_1) x + (a_0 \oplus b_0) = \left(\left(a_{m-1} \oplus b_{m-1} \right) \ldots \left(a_0 \oplus b_0 \right) \right)_2 \quad (2.1)$$

Нетрудно заметить, что применительно к элементам расширенного поля (поля многочленов) $GF(2^m)$ как операция сложения, так и операция вычитания элементов сводится к операциям XOR между коэффициентами соответствующих многочленов. Упрощенно говоря, сложение / вычитание элементов сводится к операции «побитового» XOR между элементами, иными словами, $\forall a,b \in GF(2^m) \Rightarrow a-b=a+(-b)=a+b=a \oplus b$.

Также отметим, что из эквивалентности операций сложения и вычитания элементов расширенного поля $GF(2^m)$ следует также то, что обратный элемент поля по сложению равен самому элементу. Иными словами, $\forall a \in GF(2^m) \Rightarrow a = -a$.

Следующий важный момент – это то, что расширенное поле $GF(2^m)$ по определению состоит из множества всевозможных многочленов вида $a(x) = a_{m-1} \cdot x^{m-1} + \ldots + a_1 \cdot x + a_0$, являющихся множеством всевозможных многочленов-остатков по модулю некоторого неприводимого многочлена $p(x)$, причем коэффициенты $a_{m-1},\ldots,a_1,a_0$ являются элементами простого поля $GF(2)$. Тогда, очевидно коэффициенты $a_{m-1},\ldots,a_1,a_0$ можно также рассматривать как «цифры» некоторого *m*-разрядного двоичного числа. Кроме того, двоичное число, в свою очередь, при необходимости мы всегда также можем перевести в десятичную систему счисления и обратно. Иными словами, любому элементу поля $GF(2^m)$ мы всегда можем поставить в соответствие *m*-разрядное двоичное число, а также перевести его для «компактности» отображения информации в десятичную систему счисления. Например, элементы расширенного поля $GF(2^4)$ можно представить в следующем виде:

$GF(2^4)$:	$a(x)$:	0	1	x	$x+1$	x^2	x^2+1	x^2+x	x^2+x+1
	$(a)_2$:	0000	0001	0010	0011	0100	0101	0110	0111
	$(a)_{10}$:	0	1	2	3	4	5	6	7

$a(x)$:	x^3	x^3+1	x^3+x	x^3+x+1	x^3+x^2	x^3+x^2+1	x^3+x^2+x	x^3+x^2+x+1
$(a)_2$:	1000	1001	1010	1011	1100	1101	1110	1111
$(a)_{10}$:	8	9	10	11	12	13	14	15

Расширенное поле $GF(2^4)$ образовано на базе простого поля $GF(2)$ при помощи некоторого неприводимого многочлена $p(x)$ четвертой степени, например, x^4+x+1. Если неприводимый многочлен к тому же является еще и примитивным (многочлен x^4+x+1 является именно таким), то по определению элемент $\alpha(x)=x$ поля является примитивным и при помощи него можно получить все ненулевые элементы поля. Особо отметим, что в двоичном представлении примитивный элемент поля $GF(2^4)$ выглядит как $(\alpha)_2=10$, а в десятичном представлении, соответственно, как $(\alpha)_{10}=2$. Вообще говоря, это справедливо для любого расширенного поля Галуа $GF(2^m)$, имеющего характеристику $p=2$.

С учетом всего вышесказанного нетрудно заметить, что сложение и вычитание элементов расширенного поля $GF(2^4)$ фактически можно свести к операции «побитового» XOR двух соответствующих двоичных эквивалентов элементов поля. Если имеем дело с десятичным представлением, то предварительно необходимо выполнить преобразование в двоичную систему, выполнить операцию «побитового» XOR, а затем перевести результат обратно в десятичную систему счисления. Рассмотрим пример.

Пример. Найдем сумму элементов расширенного поля $GF(2^4)$, представленных в виде соответствующих чисел «12» и «6» в десятичной системе счисления. Имеем, $\underbrace{(12)_{10}+(6)_{10}}_{GF(2^4)}=\underbrace{(1100)_2+(0110)_2}_{GF(2^4)}=\left\{\begin{array}{cccc}\oplus\begin{matrix}1\\0\end{matrix} & \oplus\begin{matrix}1\\1\end{matrix} & \oplus\begin{matrix}0\\1\end{matrix} & \oplus\begin{matrix}0\\0\end{matrix}\\ \hline 1 & 0 & 1 & 0\end{array}\right\}=(1010)_2=(10)_{10}$. Заметим, что разность элементов дала бы точно такой же результат, поскольку для двоичных расширенных полей сложение и вычитание сводится к операции «побитового» XOR. Особо отметим, что результирующее число «10» не имеет ничего общего с обычной суммой (равной 18) или разностью (равной 6) с точки зрения алгебры действительных чисел. Также отметим, что при выполнении побитовой операции XOR никогда не возникает ни перенос, ни заем, и разрядность результата в двоичном представлении такая же, как и у операндов.

Для умножения и деления, как и в общем случае, выгоднее использовать таблицы степеней и логарифмов. Заметим, что при использовании примитивного неприводимого многочлена $p(x)$ в качестве примитивного элемента всегда можно использовать одночлен $\alpha(x)=x$ для формирования таблицы степеней примитивного элемента. Тогда процедура формирования последовательности из степеней примитивного элемента α, являющегося одночленом x в расширенном поле $GF(2^m)$, заданного при помощи примитивного нормированного неприводимого многочлена $p(x)$, будет выглядеть следующим образом:

$$\left\{\begin{array}{c}k=1\ldots 2^m-2;\quad \alpha^0(x)=1\\ \alpha^k(x)=\underbrace{x\cdot\alpha^{k-1}(x)-\alpha_{m-1}^{(k-1)}\cdot p(x)}_{GF(2)}\end{array}\right. \qquad (2.2.1)$$

Заметим, что умножение многочлена $\alpha^{k-1}(x)$ на одночлен x при двоичном представлении элементов поля это не что иное, как «сдвиг влево» соответствующего двоичного числа на один разряд влево. Также учтем, что в поле $GF(2^m)$ операция вычитания сводится к операции «побитового» XOR. Иными словами, каждая следующая степень α^k примитивного элемента α, получается из предыдущей путем ее сдвига влево на один разряд, и если «старший бит» предыдущей степени α^{k-1} был ненулевым, то выполняется операция «побитового» XOR результата сдвига с двоичным эквивалентом примитивного неприводимого многочлена $p(x)$.

Резюмируя все вышесказанное, можно записать процедуру формирования таблицы степеней в следующем виде, используя двоичное представление элементов поля $GF(2^m)$, а также формирования таблицы логарифмов параллельно с формированием таблицы степеней, используя степени примитивного элемента в качестве индекса таблицы логарифмов:

$$\begin{cases} k = 1...2^m - 2; \quad \alpha^0 = 1; \quad \log_\alpha(\alpha^0) = 0 \\ \alpha^k = \begin{cases} \alpha^{k-1} << 1, & \alpha_{m-1}^{(k-1)} = 0 \\ (\alpha^{k-1} << 1) \oplus p, & \alpha_{m-1}^{(k-1)} = 1 \end{cases} \\ \log_\alpha(\alpha^k) = k \end{cases} \tag{2.2.2}$$

Под выражением $\alpha^{k-1} << 1$ понимается сдвиг двоичного числа влево на один разряд. Под выражением $(\alpha^{k-1} << 1) \oplus p$ понимается сдвиг двоичного числа влево на один разряд с последующей операцией «побитового» XOR результата сдвига с двоичным эквивалентом примитивного неприводимого многочлена $p(x)$.

Пример. Сформируем таблицу степеней для поля $GF(2^4)$, используя примитивный неприводимый многочлен $p(x) = x^4 + x + 1$, двоичный эквивалент: $(p)_2 = (10011)_2$.

$$\begin{cases} p = 2; \quad m = 4; \quad k = 1\ldots14; \quad p(x) = x^4 + x + 1; \quad \alpha^0 = (0001)_2 = (1)_{10} \\ \alpha^1 = (0001)_2 << 1 = (0010)_2 = (2)_{10} \\ \alpha^2 = (0010)_2 << 1 = (0100)_2 = (4)_{10} \\ \alpha^3 = (0100)_2 << 1 = (1000)_2 = (8)_{10} \\ \alpha^4 = ((1000)_2 << 1) \oplus (10011)_2 = (10000)_2 \oplus (10011)_2 = (0011)_2 = (3)_{10} \\ \alpha^5 = (0011)_2 << 1 = (0110)_2 = (6)_{10} \\ \alpha^6 = (0110)_2 << 1 = (1100)_2 = (12)_{10} \\ \alpha^7 = ((1100)_2 << 1) \oplus (10011)_2 = (11000)_2 \oplus (10011)_2 = (1011)_2 = (11)_{10} \\ \alpha^8 = ((1011)_2 << 1) \oplus (10011)_2 = (10110)_2 \oplus (10011)_2 = (0101)_2 = (5)_{10} \\ \alpha^9 = (0101)_2 << 1 = (1010)_2 = (10)_{10} \\ \alpha^{10} = ((1010)_2 << 1) \oplus (10011)_2 = (10100)_2 \oplus (10011)_2 = (0111)_2 = (7)_{10} \\ \alpha^{11} = (0111)_2 << 1 = (1110)_2 = (14)_{10} \\ \alpha^{12} = ((1110)_2 << 1) \oplus (10011)_2 = (11100)_2 \oplus (10011)_2 = (1111)_2 = (15)_{10} \\ \alpha^{13} = ((1111)_2 << 1) \oplus (10011)_2 = (11110)_2 \oplus (10011)_2 = (1101)_2 = (13)_{10} \\ \alpha^{14} = ((1101)_2 << 1) \oplus (10011)_2 = (11010)_2 \oplus (10011)_2 = (1001)_2 = (9)_{10} \end{cases}$$

Таким образом, в итоге имеем следующую таблицу степеней примитивного элемента, а также таблицу логарифмов по основанию примитивного элемента для поля $GF(2^4)$:

u	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$(\alpha^u)_2$	0001	0010	0100	1000	0011	0110	1100	1011	0101	1010	0111	1110	1111	1101	1001
$(\alpha^u)_{10}$	1	2	4	8	3	6	12	11	5	10	7	14	15	13	9

$(a)_{10}$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$(a)_2$	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
$\log_\alpha(a)$	0	1	4	2	8	5	10	3	14	9	7	6	13	11	12

Формирование таблиц степеней и логарифмов легко реализуется аппаратно при помощи сдвигового регистра с обратной связью и двоичного счетчика. Ниже на рисунке 2.1 приведена функциональная схема «конечного автомата», последовательно генерирующего степени примитивного элемента и логарифмы по основанию примитивного элемента.

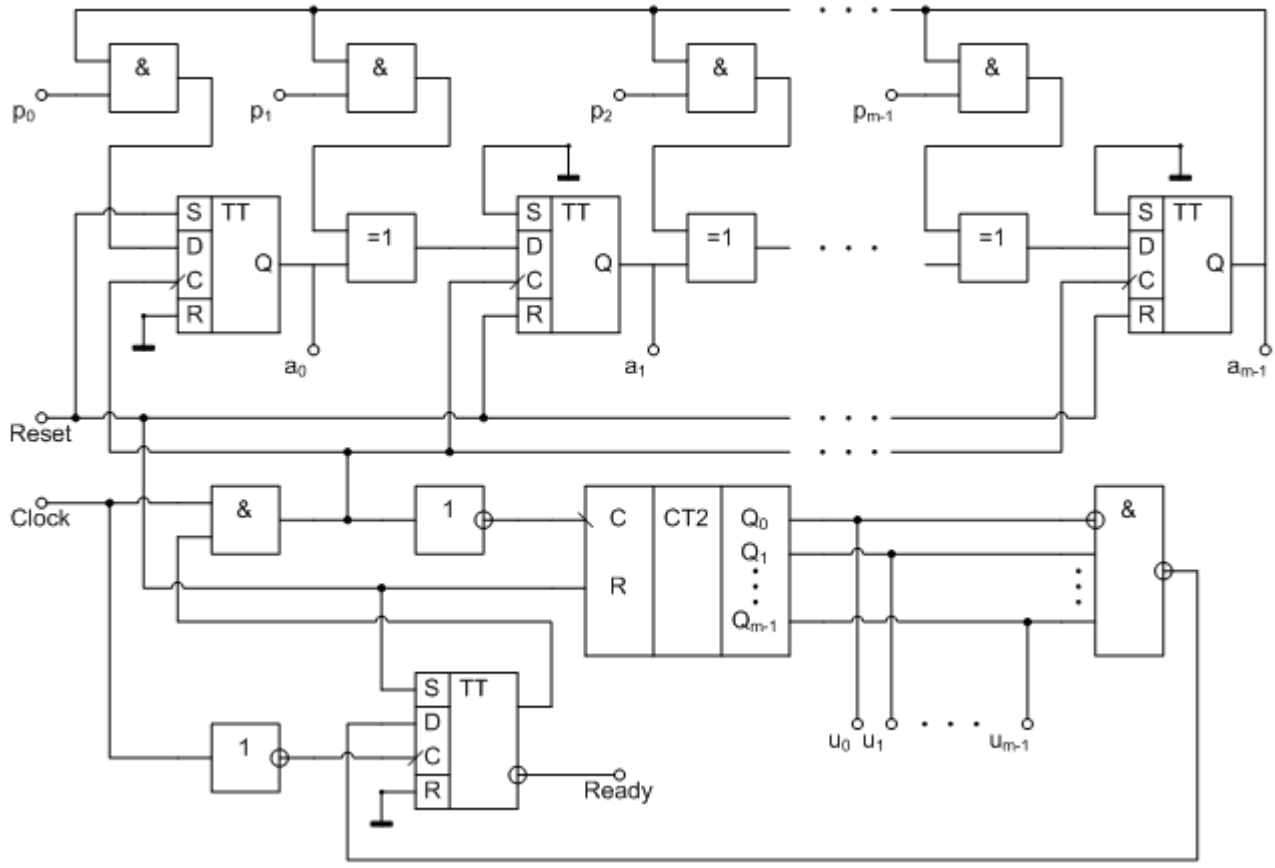

Рис. 2.1. Функциональная схема аппаратного «генератора степеней и логарифмов».

Схема устанавливается в начальное состояние при помощи сигнала *Reset*. При этом на выходах $a_{m-1},\ldots,a_1,a_0$ сдвигового регистра, построенного на базе D-триггеров, устанавливается значение $(0\ldots01)_2$, соответствующее 0-й степени примитивного элемента, то есть, $\alpha^0=(0001)_2=(1)_{10}$. Выходы $u_{m-1},\ldots,u_1,u_0$ двоичного счетчика сбрасываются в нулевое значение $(0\ldots00)_2$, соответствующее 0-му показателю степени (логарифму). Далее при каждом тактовом импульсе, подаваемом на вход *Clock*, счетчик просто увеличивает свое значение на единицу (логарифм – показатель степени, последовательно растет), а в регистре происходит сдвиг на один разряд влево. Если при этом старший бит a_{m-1} был ненулевым, то помимо сдвига также происходит операция «побитового» XOR с двоичным эквивалентом $p_{m-1},\ldots,p_1,p_0$ примитивного неприводимого многочлена. Обратим особое внимание на то, что в схеме отсутствует цепь для вычисления разряда $a_m^{(k)}$, также отсутствует цепь для ввода значения разряда p_m, который по определению используемого нормированного неприводимого многочлена $p(x)$ всегда равен 1. Нетрудно заметить, что мы всегда имеем $a_m^{(k)}=a_{m-1}^{(k-1)}\oplus a_{m-1}^{(k-1)}\cdot p_m=0$, так как $p_m=1$, поэтому мы можем не вычислять m-й разряд, и, тем самым, экономить на аппаратных средствах.

При достижении на двоичном счетчике значения $(1\ldots10)_2$, соответствующего логарифму (показателю степени), равного 2^m-2, срабатывает схема управления (нижний D-триггер), блокирующая прохождение тактовых импульсов на сдвиговый регистр и на счетчик, и устанавливающая сигнал *Ready*. На выходах сдвигового регистра при этом значение, соответствующее (2^m-2)-й степени примитивного элемента поля $GF(2^m)$. Схема остается в таком состоянии до повторного подачи сигнала сброса на вход *Reset*.

Таким образом, «конечный автомат» одновременно генерирует последовательность логарифмов $u=0\ldots2^m-2$ и соответствующих степеней $\alpha^u=\alpha^0\ldots\alpha^{(2^m-2)}$ примитивного поля. Используя дополнительные схемы двоичного сравнения, можно автомат превратить в узел вычисления степени по заданному u или вычисления логарифма по заданному a, и использовать его для выполнения операций умножения и деления элементов поля. Однако, следует отметить то, что автомат является последовательным и время, затрачиваемое на поиск требуемого логарифма или степени в худшем случае пропорционально 2^m-2, что, конечно же, при больших m делает автомат малопригодным из-за низкого быстродействия. Поэтому автомат может использоваться только при небольших m, или же как часть аппаратного «программатора» ПЗУ для записи в них таблиц степеней и логарифмов.

Располагая таблицами степеней и логарифмов, как и в общем случае, мы сводим операции умножения и деления элементов поля $GF(2^m)$ к сложению и вычитанию по модулю 2^m-1 соответствующих логарифмов элементов поля.

- $$\overbrace{a\cdot b}^{GF(2^m)}=\begin{cases}\alpha^{\overbrace{(\log_\alpha a+\log_\alpha b)\bmod(2^m-1)}^{<R,\{+,\cdot\}>}} & a\neq0\ \&\ b\neq0\\ 0 & a=0\vee b=0\end{cases} \qquad (2.3)$$
- $$\overbrace{a/b}^{GF(2^m)}=\begin{cases}\alpha^{\overbrace{(\log_\alpha a+((2^m-1)-\log_\alpha b))\bmod(2^m-1)}^{<R,\{+,\cdot\}>}} & a\neq0\ \&\ b\neq0\\ 0 & a=0\ \&\ b\neq0\\ \textit{Ошибка} & b=0\end{cases}$$

Пример. Найдем произведение элементов поля $GF(2^4)$, представленных в виде соответствующих чисел «5» и «7» в десятичной системе счисления. По «таблице логарифмов» для поля $GF(2^4)$ имеем логарифмы элементов: $\log_\alpha(5)_{10}=8$ и $\log_\alpha(7)_{10}=10$.

Тогда получаем: $\overbrace{(5)_{10}\cdot(7)_{10}}^{GF(2^4)}=\alpha^{\overbrace{(8+10)\bmod(15)}^{<R,\{+,\cdot\}>}}=\alpha^3$. По «таблице степеней» для поля $GF(2^4)$ имеем $\alpha^3=(8)_{10}$. В итоге получаем: $\overbrace{(5)_{10}\cdot(7)_{10}}^{GF(2^4)}=(8)_{10}$.

Пример. Найдем отношение элементов «12» на «9» в поле $GF(2^4)$. По «таблице логарифмов» для поля $GF(2^4)$ имеем логарифмы элементов: $\log_\alpha(12)_{10}=6$ и $\log_\alpha(9)_{10}=14$. Тогда получаем: $\overbrace{(12)_{10}/(9)_{10}}^{GF(2^4)}=\alpha^{\overbrace{(6+(15-14))\bmod(15)}^{<R,\{+,\cdot\}>}}=\alpha^7$. По «таблице степеней» для поля $GF(2^4)$ имеем $\alpha^7=(11)_{10}$. В итоге получаем: $\overbrace{(12)_{10}/(9)_{10}}^{GF(2^4)}=(11)_{10}$.

Теперь отметим то, что для аппаратной реализации операций умножения и деления элементов поля $GF(2^m)$ с использованием таблиц степеней и логарифмов, очевидно, что помимо ПЗУ, хранящих эти таблицы, потребуется функциональный узел сложения и вычитания логарифмов элементов поля. Учтем, что логарифмы с точки зрения алгебры вещественных чисел являются целыми неотрицательными числами в диапазоне $0\ldots 2^m-2$ и их сложение / вычитание производится по модулю 2^m-1.

Среди цифровых интегральных микросхем присутствуют, так называемые полные сумматоры, позволяющие складывать целые неотрицательные числа в диапазоне $0\ldots 2^m-1$, представленные в виде *m*-разрядных двоичных чисел. Полные сумматоры также формируют выходной сигнал «переноса» в случае, когда сумма чисел больше 2^m-1 и не умещается в *m* разрядов. Также полные сумматоры имеют специальный вход для учета сигнала переноса от другого сумматора. То есть фактически полный сумматор складывает три двоичных числа: два *m*-разрядных и одно 1-разрядное двоичное число, и формирует $m+1$-разрядную двоичную сумму. Особо отметим, что без разряда переноса, *m*-разрядная сумма, формируемая полным сумматором при сложении *m*-разрядных двоичных чисел *u* и *v*, это не что иное, как остаток $(u+v)$ по модулю 2^m, иными словами $(u+v)\bmod(2^m)$, а разряд переноса можно интерпретировать как целую часть от деления $(u+v)$ на число 2^m, иными словами $(u+v)div(2^m)$. Теперь заметим, что мы имеет место быть тождество $(u+v)=((u+v)div(2^m))\cdot 2^m+(u+v)\bmod(2^m)$. Преобразуем его следующим образом: $(u+v)=((u+v)div(2^m))\cdot(2^m-1)+((u+v)\bmod(2^m)+(u+v)div(2^m))$. Теперь применим операцию вычисления остатка по модулю 2^m-1 к левой и правой части тождества, и тогда в итоге получим: $(u+v)\bmod(2^m-1)=((u+v)\bmod(2^m)+(u+v)div(2^m))\bmod(2^m-1)$.

Тогда, очевидно, что для сложения *m*-разрядных двоичных чисел *u* и *v* по модулю 2^m-1 можно воспользоваться двумя полными сумматорами: первый будет осуществлять сложение чисел, вычисляя *m*-разрядную двоичную сумму $(u+v)\bmod(2^m)$ и перенос $(u+v)div(2^m)$, а второй – складывать $(u+v)\bmod(2^m)$ с переносом $(u+v)div(2^m)$, как с одноразрядным двоичным числом, и на втором сумматоре будет получаться $((u+v)\bmod(2^m)+(u+v)div(2^m))$. Однако, для получения итогового результата нужно еще вычислить остаток по модулю 2^m-1, но мы заметим, что в большинстве случаях $((u+v)\bmod(2^m)+(u+v)div(2^m))$ меньше 2^m-1, и вычисления остатка по модулю 2^m-1 излишне, кроме двух особых случаев, когда $(u+v)\bmod(2^m)=2^m-1$, а $(u+v)div(2^m)=0$, и когда $(u+v)\bmod(2^m)=2^m-2$, а $(u+v)div(2^m)=1$, в обоих случаях получается, что $((u+v)\bmod(2^m)+(u+v)div(2^m))=2^m-1$, и итоговым результатом должен быть 0. Заметим, что в *m*-разрядной двоичной сетке, число 2^m-1 несложно превратить в 0, прибавив к нему «лишнюю» 1 при возникновении особых случаев на втором сумматоре, используя его возможность складывать два *m*-разрядных и одно 1-разрядное число.

Наконец, последний момент – вычитание логарифмов. Здесь мы воспользуемся тождеством $(u-v)\bmod(2^m-1)=(u+((2^m-1)-v))\bmod(2^m-1)$, а также заметим, что $((2^m-1)-v)$ с точки зрения двоичной арифметики – это не что иное, как «побитовая» инверсия разрядов *m*-разрядного двоичного числа *v*.

Тогда с учетом всего вышесказанного, можно использовать следующую схему сложения / вычитания логарифмов *u* и *v* по модулю $2^m - 1$, представленную на рисунке 2.2.

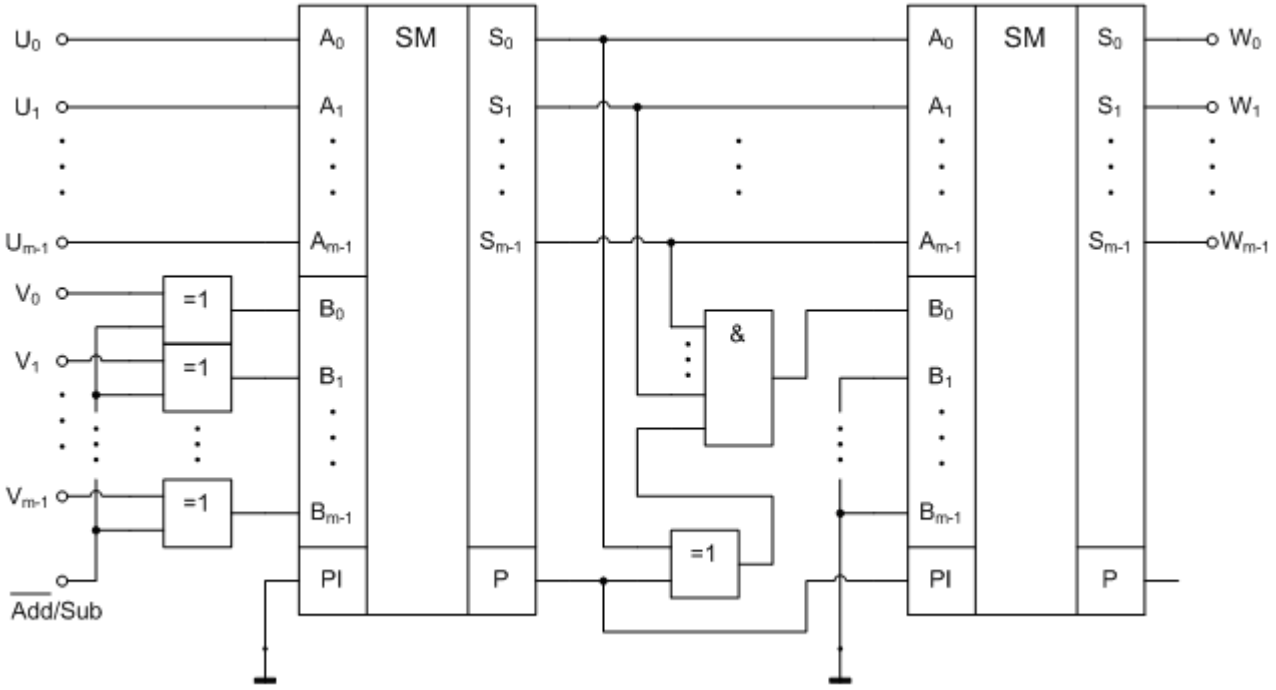

Рис. 2.2. Функциональная схема узла сложения / вычитания по модулю $2^m - 1$.

Двоичное число *u* поступает непосредственно как первое слагаемое на первый сумматор. Двоичное число *v* поступает как второе слагаемое, предварительно пройдя элементы XOR, которые либо сохраняют его в неизменном виде в случае операции сложения (вход выбора операции Add/Sub = 0), либо осуществляют побитовую инверсию в случае операции вычитания (вход выбора операции Add/Sub = 1). Первый сумматор вычисляет *m*-разрядную сумму и одноразрядный перенос, которые поступают на второй сумматор, где *m*-разрядная сумма складывается с переносом, как с одноразрядным двоичным числом. Кроме того, если либо сумма равна $2^m - 1$, а перенос равен 0, либо сумма равна $2^m - 2$, а перенос равен 1, то при помощи многовходового элемента «И» и двухвходового элемента XOR формируется «лишняя» 1, которая также подается на второй сумматор, как слагаемое. В итоге на выходе второго сумматора получаем сумму или разность *u* и *v* по модулю $2^m - 1$.

Теперь располагая узлом сложения / вычитания по модулю $2^m - 1$, и добавив к нему двухпортовое ПЗУ с таблицей логарифмов, а также ПЗУ с таблицей степеней, и некоторые вспомогательные узлы, можем рассмотреть арифметический процессор, выполняющий четыре основные операции с элементами поля Галуа $GF(2^m)$.

Ниже на рисунке 2.3 представлена функциональная схема арифметического процессора. Ключевым элементом является *m*-разрядный выходной мультиплексор $2 \to 1$, который при управляющем сигнале S1 = 0 соединяет свои выходы с выходами элементов XOR, осуществляющих фактически сложение (оно же эквивалентно вычитанию) путем операции «побитового» XOR между входными операндами *a* и *b* – элементами поля $GF(2^m)$. При управляющем сигнале S1 = 1, мультиплексор подключает свои выходы к выходам ПЗУ с таблицей степеней, на вход которого, в свою очередь, поступает результат сложения (при управляющем сигнале S0 = 0) или вычитания (при S0 = 1) по модулю $2^m - 1$ логарифмов, извлекаемых из ПЗУ с таблицей логарифмов, одновременно по двум независимым шинам для обоих входных операндов *a* и *b* – элементов поля $GF(2^m)$.

Таким образом, при S1 = 0, и S0 = 0 или S0 = 1, выполняется операция сложения / вычитания элементов поля путем одной и той же операции «побитового» XOR. При S1 = 1 и S0 = 0 выполняется операция умножения, а при S1 = 1 и S0 = 1 выполняется деление. Отметим также, что в схеме присутствует логическая схема, проверяющая равенство второго операнда *b* нулю, и при *b* = 0 в случае выбора операции деления, формируется специальный выходной сигнал ошибки деления на нуль – «divide by zero error». Управляющий вход CS (chip select) служит для активизации выходов микросхем ПЗУ и мультиплексора.

Несомненным преимуществом рассмотренного процессора является высокое быстродействие, все операции с элементами поля выполняются фактически «за один шаг». Однако, следует отметить, что суммарная емкость двух ПЗУ составляет $2 \cdot 2^m \cdot m$ бит, и при больших *m* аппаратные затраты на ПЗУ могут оказаться существенными.

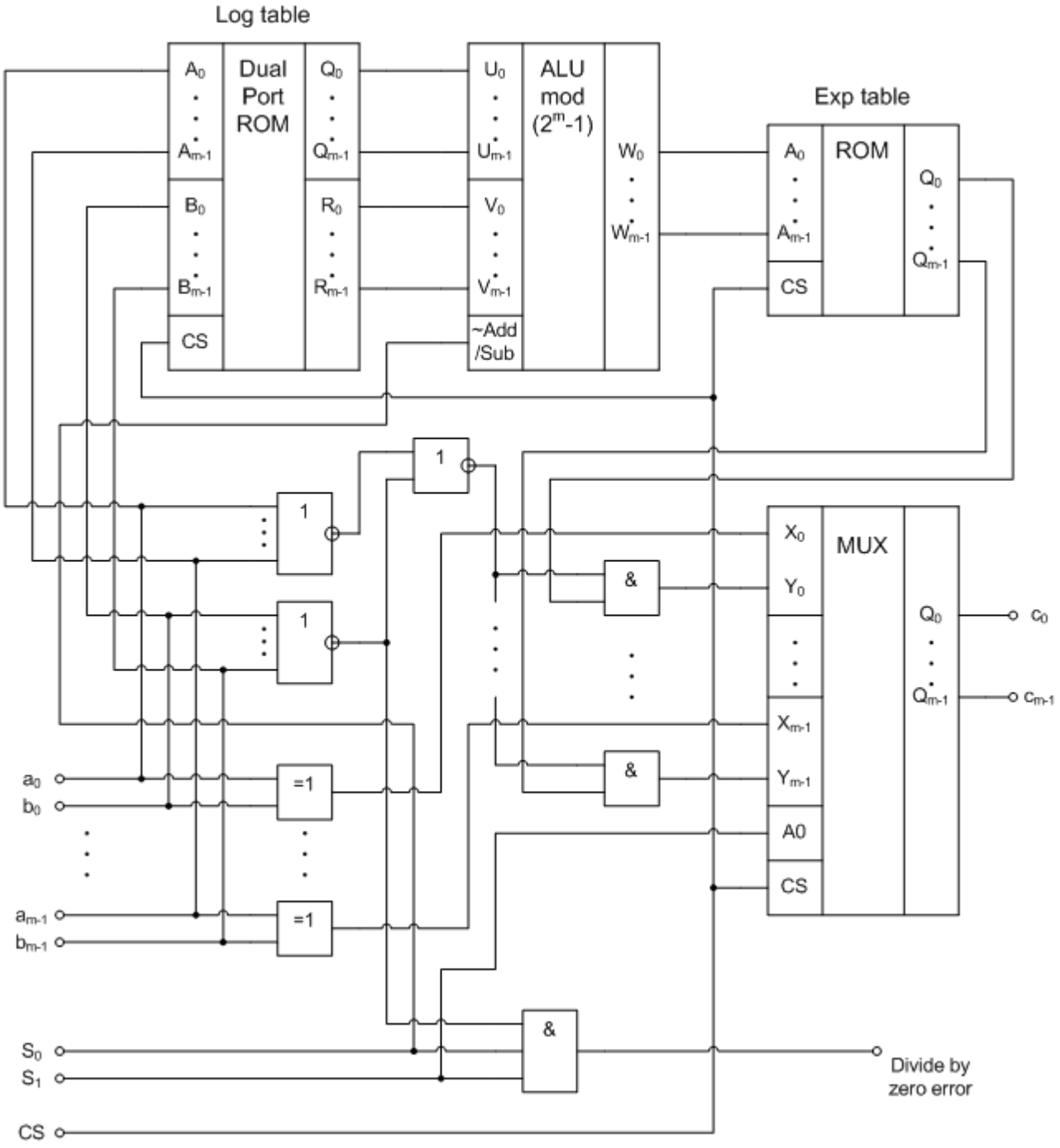

Рис. 2.3. Функциональная схема арифметического процессора для поля Галуа *GF*(2^m) с использованием таблиц логарифмов по основанию и степеней примитивного элемента.

Вышеприведенный арифметический процессор позволяет с высокой скоростью («за один шаг») выполнять 4 основные операции с любыми двумя заданными элементами поля $GF(2^m)$. Кроме того процессор является «программируемым» с точки зрения используемого в операциях умножения и деления примитивного неприводимого многочлена. Замена примитивного неприводимого многочлена требует лишь перезаписи таблиц степеней и логарифмов, соответствующих требуемому примитивному неприводимому многочлену, в ПЗУ, а с аппаратной точки зрения функциональная схема процессора остается неизменной.

Однако, следует отметить, что на практике, в частности в аппаратных реализациях алгоритмов кодирования и декодирования информации, как правило, примитивный неприводимый многочлен является заданным «раз и навсегда», и, соответственно, перепрограммирование таблиц степеней и логарифмов практически никогда не требуется. Кроме того, в алгоритмах кодирования чаще требуется, в первую очередь, сложение и умножение элементов Галуа, причем для достижения наибольшего быстродействия требуется выполнять одновременно нескольких операций, например, умножить несколько элементов поля $GF(2^m)$ на некоторый другой элемент. В такой ситуации использовать арифметический процессор в качестве умножителя становится накладным с точки зрения быстродействия в силу необходимости многократного обращения к нему. Особенно это становится накладным при умножении элементов поля на некоторый фиксированный элемент поля, который задается как «константа» в том или ином алгоритме.

В такой ситуации для построения быстрых и компактных умножителей выгоднее использовать классическое определение операции умножения элементов поля Галуа $GF(2^m)$, представленных в виде многочленов с коэффициентами из простого поля $GF(2): \forall a,b \in GF(2^m) \Rightarrow \underbrace{a \cdot b}_{GF(2^m)} = \underbrace{(a(x) \cdot b(x)) \bmod p(x)}_{GF(2)}$.

Рассмотрим умножение элементов на примере поля Галуа $GF(2^4)$, образованного при помощи примитивного неприводимого многочлена $p(x) = x^4 + x + 1$. Имеем следующее: $\underbrace{a \cdot b}_{GF(2^4)} = \underbrace{((a_3 \cdot x^3 + a_2 \cdot x^2 + a_1 \cdot x + a_0) \cdot (b_3 \cdot x^3 + b_2 \cdot x^2 + b_1 \cdot x + b_0)) \bmod (x^4 + x + 1)}_{GF(2)}$. После перемножения многочленов и вычисления остатка по модулю $p(x) = x^4 + x + 1$ получаем:

$$\underbrace{a \cdot b}_{GF(2^4)} = \underbrace{(a(x) \cdot b(x)) \bmod p(x)}_{GF(2)} = c_3 \cdot x^3 + c_2 \cdot x^2 + c_1 \cdot x + c_0$$

$$\begin{cases} c_3 = a_0 \cdot b_3 \oplus a_1 \cdot b_2 \oplus a_2 \cdot b_1 \oplus a_3 \cdot b_0 \oplus a_3 \cdot b_3 \\ c_2 = a_0 \cdot b_2 \oplus a_1 \cdot b_1 \oplus a_2 \cdot b_0 \oplus a_3 \cdot b_3 \oplus a_2 \cdot b_3 \oplus a_3 \cdot b_2 \\ c_1 = a_0 \cdot b_1 \oplus a_1 \cdot b_0 \oplus a_2 \cdot b_3 \oplus a_3 \cdot b_2 \oplus a_1 \cdot b_3 \oplus a_2 \cdot b_2 \oplus a_3 \cdot b_1 \\ c_0 = a_0 \cdot b_0 \oplus a_1 \cdot b_3 \oplus a_2 \cdot b_2 \oplus a_3 \cdot b_1 \end{cases} \quad (2.4.1)$$

Таким образом, мы имеем *m* аддитивных функций для вычисления коэффициентов $c_{m-1} \ldots c_0$. Функции содержат слагаемые в виде произведений коэффициентов $a_i \cdot b_j$, где $i, j = 0 \ldots m-1$. Поскольку мы имеем дело с полями $GF(2^m)$, являющиеся расширением базового простого поля $GF(2)$, то произведение коэффициентов эквивалентно логическому умножению (конъюнкции). Для аппаратной реализации таких функции удобнее использовать специализированные программируемые логические матрицы (ПЛМ), содержащие в себе логические элементы «И» с двумя входами и многовходовые сумматоры по модулю 2.

Ниже на рис. 2.4 приведена функциональная схема умножителя элементов поля $GF(2^4)$, образованного на базе примитивного неприводимого многочлена $p(x) = x^4 + x + 1$.

Входы сумматоров в соответствии с аддитивными функциями подключаются к выходам логических элементов «И», формирующих соответствующие произведения коэффициентов $a_i \cdot b_j$. Незадействованные входы сумматоров подключаются к «земле».

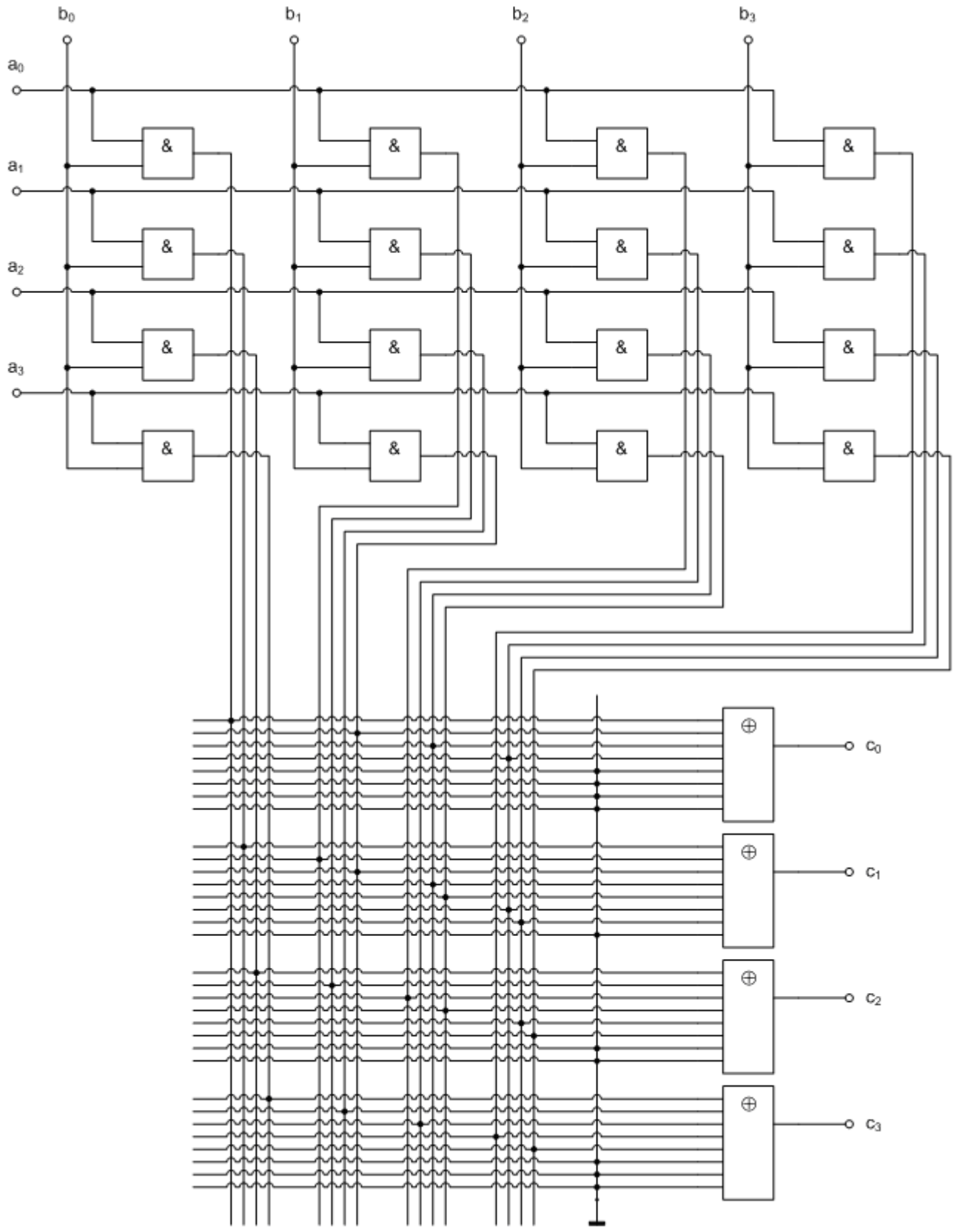

Рис. 2.4. Функциональная схема умножителя элементов поля Галуа $GF(2^4)$ на базе специализированной логической матрицы.

Приведенный пример умножителя элементов поля Галуа $GF(2^4)$ нетрудно обобщить для общего случая умножения элементов полей Галуа $GF(2^m)$ образованного на базе некоторого заданного примитивного неприводимого многочлена $p(x)$.

$$\underbrace{a \cdot b}_{GF(2^m)} = \underbrace{(a(x) \cdot b(x)) \bmod p(x)}_{GF(2)} = c_{m-1} \cdot x^{m-1} + \ldots + c_1 \cdot x + c_0$$

$$\begin{cases} c_{m-1} = \Sigma_{m-1}\left(a_i \cdot b_j\right) \\ \vdots \\ c_1 = \Sigma_1\left(a_i \cdot b_j\right) \\ c_0 = \Sigma_0\left(a_i \cdot b_j\right) \end{cases} \qquad (2.4.2)$$

Ниже на рис. 2.5. приведена общая функциональная схема умножителя элементов поля Галуа $GF(2^m)$ на базе специализированной программируемой логической матрицы.

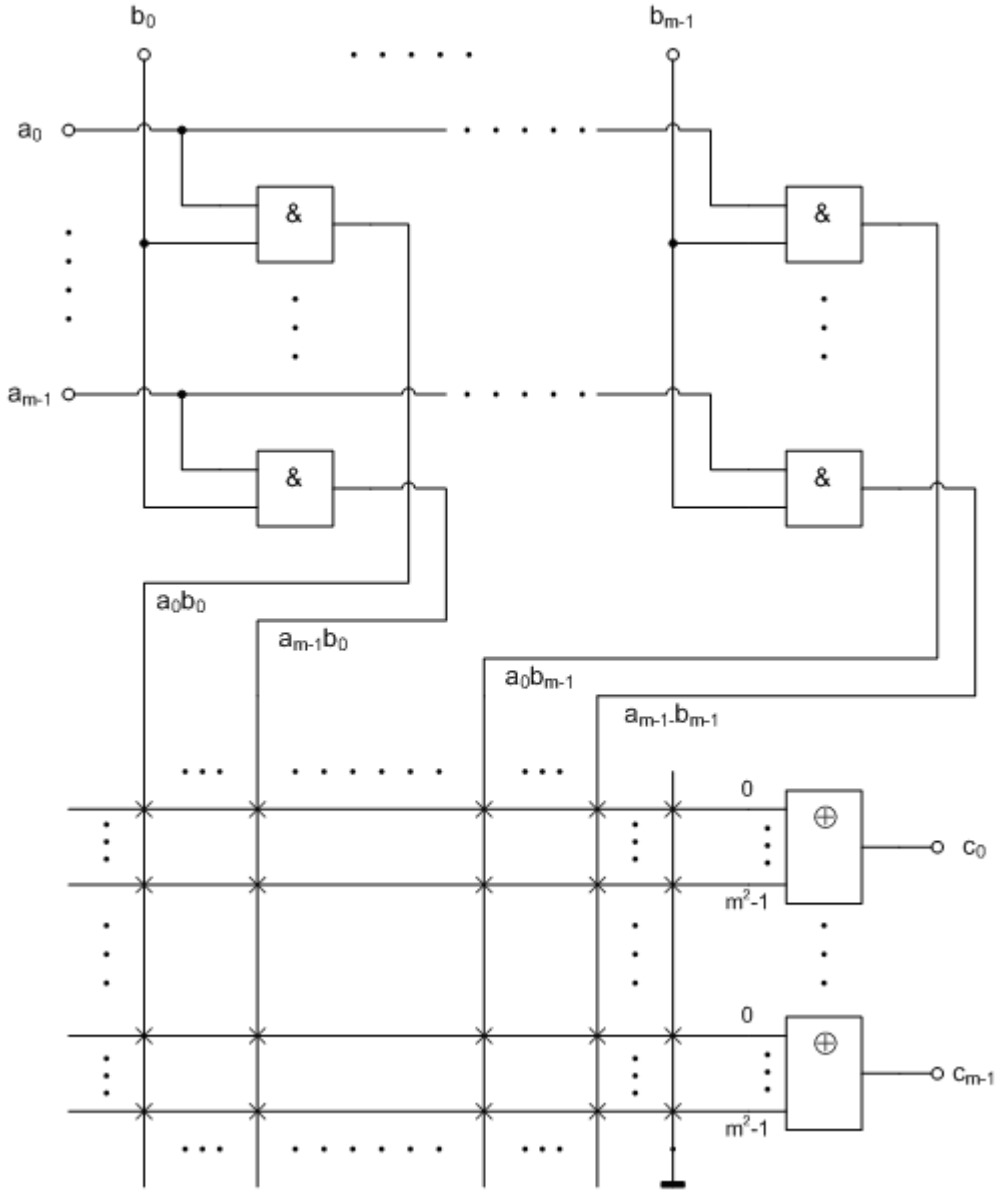

Рис. 2.5. Общая функциональная схема умножителя элементов для полей Галуа $GF(2^m)$ на базе специализированной программируемой логической матрицы.

Теперь рассмотрим умножение произвольного элемента a поля Галуа $GF(2^m)$ на фиксированный элемент $\tilde{b}$ этого поля, который является константой в том или ином алгоритме, и определяется еще на этапе проектирования аппаратной реализации алгоритма.

Рассмотрим на примере поля Галуа $GF(2^4)$, образованного при помощи примитивного неприводимого многочлена $p(x) = x^4 + x + 1$ умножение элемента поля на некоторый фиксированный (константный) элемент $\tilde{b}$, например, $\tilde{b} = (1111)_2$. Тогда аддитивные функции для вычисления коэффициентов многочлена-произведения $c(x)$ значительно упрощаются:

$$\underbrace{a \cdot \tilde{b}}_{GF(2^4)} = c_3 \cdot x^3 + c_2 \cdot x^2 + c_1 \cdot x + c_0$$

$$\begin{cases} c_3 = a_0 \oplus a_1 \oplus a_2 \\ c_2 = a_0 \oplus a_1 \\ c_1 = a_0 \\ c_0 = a_0 \oplus a_1 \oplus a_2 \oplus a_3 \end{cases} \qquad (2.5.1)$$

Для аппаратной реализации также можно использовать специализированные программируемые логические матрицы, содержащие в себе многовходовые сумматоры по модулю 2. Ниже на рис. 2.6 приведена функциональная схема умножителя элементов поля $GF(2^4)$, образованного на базе примитивного неприводимого многочлена $p(x) = x^4 + x + 1$, на фиксированный элемент $\tilde{b} = (1111)_2$ с использованием ПЛМ. Входы сумматоров в соответствии с аддитивными функциями подключаются к соответствующим линиям коэффициентов a_i. Незадействованные входы сумматоров подключаются к «земле».

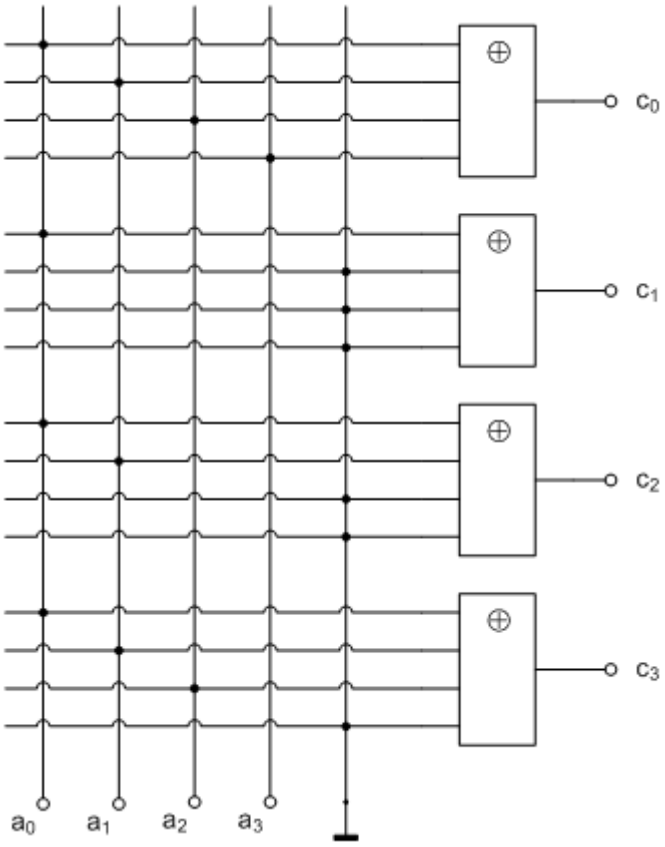

Рис. 2.6. Функциональная схема умножителя элемента a поля Галуа $GF(2^4)$ на константный элемент $\tilde{b} = (1111)_2$ на базе специализированной логической матрицы.

Следует отметить, что вышеприведенный умножитель можно было бы реализовать в «экономичном» варианте (рис. 2.7), используя двухвходовые логические элементы XOR. Однако, у такого варианта есть существенный недостаток – различные задержки формирования выходных сигналов, ведущие к проблеме «гонок» в цифровых схемах: сигнал на выходе c_1 формируется практически сразу же, так как он просто соединен с входной линией a_0, остальные же сигналы формируются позже из-за задержек на одном (для c_2), двух (для c_3) и на трех (для c_0) последовательно соединенных логических элементах XOR. Поэтому ПЛМ хотя и более затратные, зато обеспечивают более или менее равные задержки.

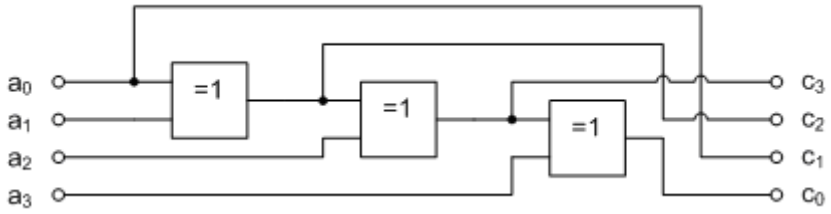

Рис. 2.7. Экономичный вариант функциональной схемы умножения элемента a поля Галуа $GF(2^4)$ на константный элемент $\tilde{b} = (1111)_2$.

Приведенный пример умножения элемента поля Галуа $GF(2^4)$ на фиксированный элемент $\tilde{b} = (1111)_2$ нетрудно обобщить для общего случая умножения элемента поля Галуа $GF(2^m)$ образованного на базе заданного примитивного неприводимого многочлена $p(x)$ на заданный фиксированный элемент $\tilde{b}$.

$$\underbrace{a \cdot \tilde{b}}_{GF(2^m)} == c_{m-1} \cdot x^{m-1} + \ldots + c_1 \cdot x + c_0$$

$$\begin{cases} c_{m-1} = \Sigma_{m-1}\left(a_i\right) \\ \vdots \\ c_1 = \Sigma_1\left(a_i\right) \\ c_0 = \Sigma_0\left(a_i\right) \end{cases} \qquad (2.5.2)$$

Ниже на рис. 2.8 также приведена общая функциональная схема «умножителя на константу» для поля Галуа $GF(2^m)$ на базе специализированной ПЛМ.

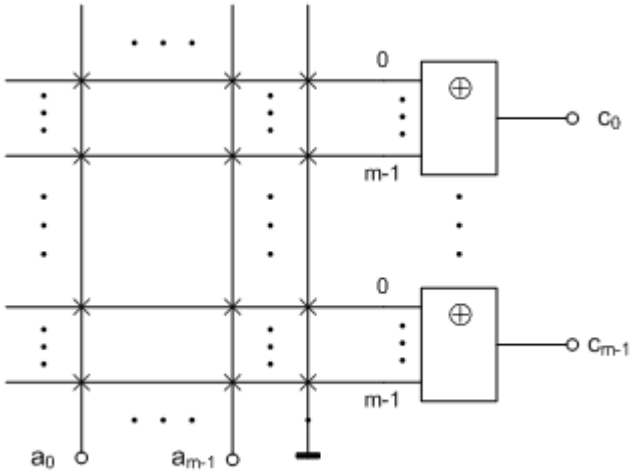

Рис. 2.8. Общая функциональная схема «умножителя на константу» для полей Галуа $GF(2^m)$ на базе специализированной программируемой логической матрицы.

Наконец, заметим, что используя умножитель элементов поля Галуа $GF(2^m)$ на базе специализированной программируемой логической матрицы также можно построить арифметический процессор для поля Галуа $GF(2^m)$, сведя операцию деления элемента *a* на ненулевой элемент *b* поля к умножению на обратный элемент b^{-1} по умножению.

$$\forall a,b \in GF(2^m): b \neq 0 \Rightarrow \underbrace{a/b}_{GF(2^m)} = \underbrace{a \cdot b^{-1}}_{GF(2^m)}$$

Вычисление обратного элемента по умножению для максимального быстродействия, очевидно, лучше всего опять же осуществлять табличным способом, используя ПЗУ емкостью $2^m \cdot m$ бит. Что касается формирования самой таблицы обратных элементов на этапе проектирования, то это можно осуществлять программным способом, используя расширенный алгоритм Евклида для многочленов, который мы подробно рассматривали выше, который сводит решение уравнения $\underbrace{(b(x) \cdot b^{-1}(x)) \bmod p(x) = 1}_{GF(2)}$, где $b^{-1}(x)$ разыскиваемый обратный многочлен по умножению, к нахождению многочленов $g(x)$ и $h(x)$, а также наибольшего общего делителя $НОД(b(x), p(x))$ для многочленов $b(x)$ и $p(x)$ таких, что $\underbrace{b(x) \cdot g(x) + p(x) \cdot h(x) = НОД(b(x), p(x))}_{GF(2)}$. Поскольку мы имеем дело с полем, то для любого ненулевого многочлена $b(x)$ алгоритм в качестве $НОД(b(x), p(x)$ дает скаляр $\lambda \in GF(2)$ (многочлен нулевой степени), и в нашем «двоичном» случае скаляр будет равен строго $\lambda = 1$, так как в базовом простом поле $GF(2)$ существует только один ненулевой элемент – это единица. Соответственно, многочлен $g(x)$, находимый алгоритмом, и является искомым обратным многочленом по умножению, то есть $b^{-1}(x) = g(x)$.

Приведем пример таблицы обратных элементов для элементов поля $GF(2^4)$, представленных для компактности в виде десятичных и двоичных эквивалентов элементов.

$(b)_{10}$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$(b)_2$	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
$(b^{-1})_{10}$	1	9	14	13	11	7	6	15	2	12	5	10	4	3	8
$(b^{-1})_2$	0001	1001	1110	1101	1011	0111	0110	1111	0010	1100	0101	1010	0100	0011	1000

Ниже на рисунке 2.9 представлена функциональная схема арифметического процессора для поля Галуа $GF(2^m)$, использующего таблицу обратных элементов по умножению, хранящуюся в ПЗУ и умножитель элементов, который, как было рассмотрено выше, реализуется на базе специализированной программируемой логической матрицы.

В схеме процессора используются два *m*-битных мультиплексора $2 \to 1$. Нижний мультиплексор коммутирует свои входы с выходами логических элементов XOR при управляющем сигнале S1 = 0 (режим операций сложения / вычитания), и с выходами умножителя при S1 = 1 (режим операций умножения / деления). При S1 = 0, управляющий сигнал S0 не играет никакой роли (сложение и вычитание сводится к одной и той же операции «побитового» XOR). При S1 = 1, сигнал S0 управляет верхним мультиплексором, который при S0 = 0 подключает линии $b_{m-1} \ldots b_0$ напрямую к умножителю элементов, что соответствует операции умножения на операнд *b*, а при S0 = 1 подключает выходы ПЗУ, преобразующего операнд *b* в его обратный элемент по умножению, что соответствует операции деления на операнд *b*. В схеме процессора также предусмотрена цепь обнаружения нулевого делителя (*b* = 0) в режиме операции деления (S1 = 1 и S0 = 1).

Очевидно, что по сравнению с арифметическим процессором на базе таблиц логарифмов и степеней, требующего два ПЗУ общей емкостью $2 \cdot 2^m \cdot m$ бит, а также узла сложения / вычитания логарифмов по модулю $2^m - 1$, арифметический процессор на базе умножителя и таблицы обратных элементов более экономичен по аппаратным затратам. Он требует только одного ПЗУ емкостью $2^m \cdot m$ бит, и умножителя, состоящего из m^2 двухвходовых логических элементов «И», m многовходовых сумматоров по модулю 2, и коммутационной матрицы размером $(m^2+1) \cdot m^2 \cdot m \sim m^5$. Если коммутационную матрицу тоже рассматривать как «постоянную память», то, очевидно, что ее размер m^5 с ростом m значительно медленнее растет по сравнению с емкостью ПЗУ $2^m \cdot m$.

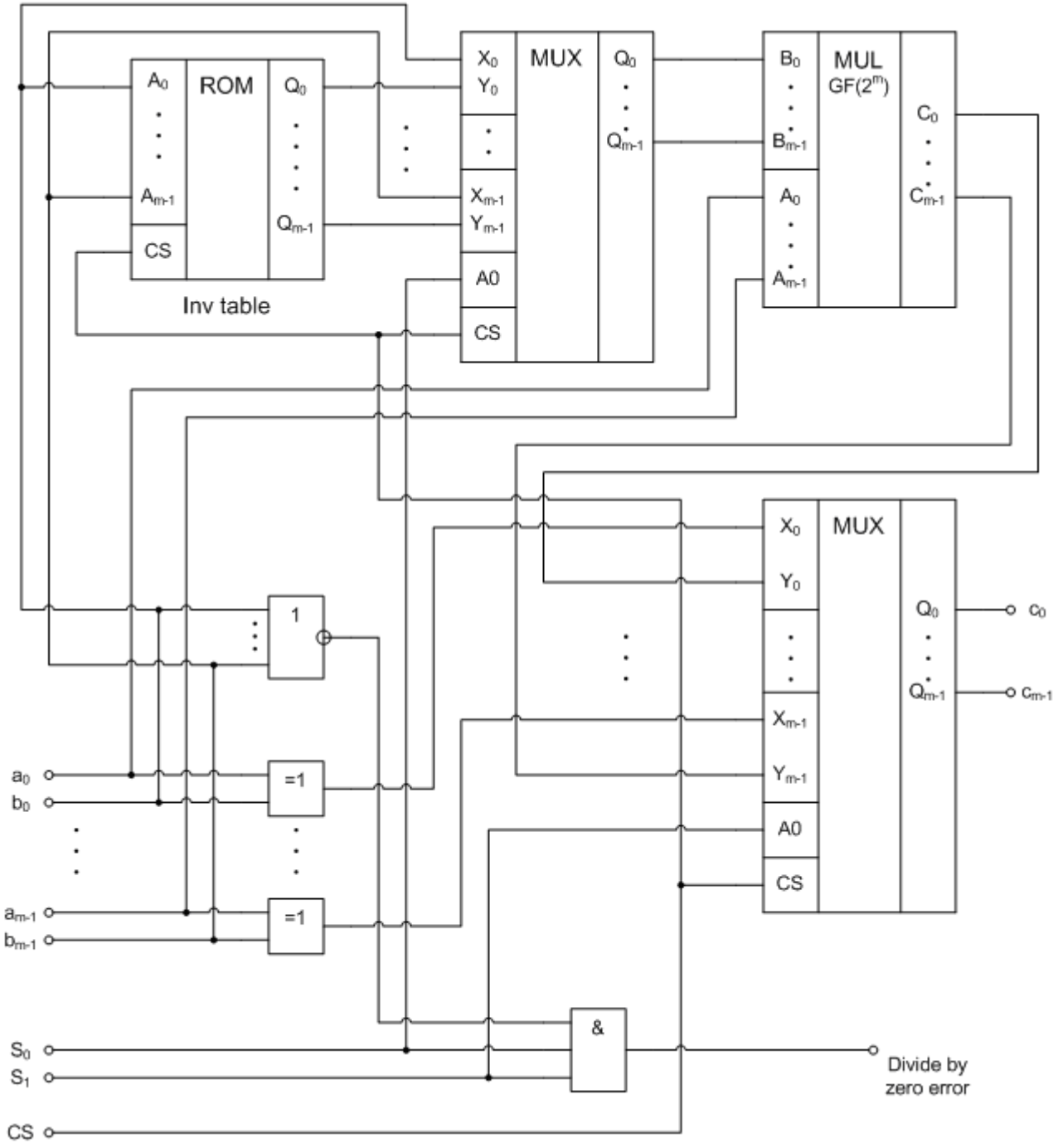

Рис. 2.9. Функциональная схема арифметического процессора для поля Галуа *GF*(2^m) с использованием умножителя элементов и таблицы обратных элементов по умножению.

Конечное поле Галуа $GF(2^8)$ в технологии помехоустойчивого кодирования. Расширенное конечное поле Галуа $GF(2^8)$ является частным случаем расширенных конечных полей $GF(2^m)$ характеристики 2 и имеет широкое применение в технологиях помехоустойчивой передачи и хранения информации благодаря тому, что основной единицей информации в вычислительной технике является байт. Байт состоит 8 битов и с помощью него можно представить 256 различных символов, и поле $GF(2^8)$ также содержит 256 элементов, которые можно представить в виде 8-разрядных двоичных чисел.

Яркие примеры использования арифметики поля Галуа $GF(2^8)$: помехоустойчивое кодирование с применением кодов Рида-Соломона для хранения информации на оптических дисках с данными – Data CD-ROM и при передаче информации в стандарте цифрового телевещания DVB – Digital Video Broadcasting.

Поле Галуа $GF(2^8)$ содержит 256 элементов:

	$a(x)$:	0	1	x	…	$x^7+x^6+x^5+x^4+x^3+x^2+x+1$
$GF(2^8)$:	$(a)_2$:	00000000	00000001	00000010	…	11111111
	$(a)_{10}$:	0	1	2	…	255

Поле Галуа $GF(2^8)$, по определению являющееся полем многочленов вида $a(x)=a_7\cdot x^7+\ldots+a_1\cdot x+a_0$, образуется на базе простого поля Галуа $GF(2)$ и примитивного неприводимого многочлена 8-й степени. В технологии помехоустойчивого кодирования обычно используется примитивный неприводимый многочлен следующего вида:

$$p(x)=x^8+x^4+x^3+x^2+1 \qquad (2.6.1)$$

В двоичном представлении примитивный неприводимый многочлен выглядит как: $(p)_2=100011101$, а в десятичном представлении: $(p)_{10}=285$. Примитивным элементом поля $GF(2^8)$ является элемент $\alpha(x)=x$, и при помощи него можно получить все ненулевые элементы поля. В двоичном представлении примитивный элемент поля выглядит как $(\alpha)_2=10$, а в десятичном представлении, соответственно, как $(\alpha)_{10}=2$.

Для формирования таблицы степеней примитивного элемента $(\alpha)_{10}=2$ используется процедура, являющаяся упрощением процедуры для поля $GF(2^m)$ при m = 8. Таблицу логарифмов можно формировать параллельно с формированием таблицы степеней, используя двоичное или десятичное представление степеней примитивного элемента в качестве индексов таблицы логарифмов:

$$\begin{cases} k=1\ldots 2^8-2;\quad \alpha^0=1;\quad \log_\alpha(\alpha^0)=0 \\ \alpha^k=\begin{cases} \alpha^{k-1}<<1,\quad \alpha_7^{(k-1)}=0 \\ (\alpha^{k-1}<<1)\oplus(100011101)_2,\quad \alpha_7^{(k-1)}=1 \end{cases} \\ \log_\alpha(\alpha^k)=k \end{cases} \qquad (2.6.2)$$

Под выражением $\alpha^{k-1}<<1$ понимается сдвиг двоичного числа влево на один разряд. Под выражением $(\alpha^{k-1}<<1)\oplus(100011101)_2$ понимается сдвиг двоичного числа влево на один разряд с последующей операцией «побитового» XOR результата сдвига с двоичным эквивалентом примитивного неприводимого многочлена.

Примечание. Поскольку в десятичном виде примитивный элемент поля $GF(2^8)$ выглядит как $(\alpha)_{10}=2$, то будем также обозначать *k*-ую степень примитивного элемента как 2^k, а логарифм от элемента *a* по основанию примитивного элемента как $\log_2 a$.

Ниже в таблице 1.1 (в виде матрицы 16 х 16) приведены степени примитивного элемента $(\alpha)_{10} = 2$ поля $GF(2^8)$, образованного при помощи примитивного неприводимого многочлена $p(x) = x^8 + x^4 + x^3 + x^2 + 1$. Степени примитивного элемента для компактности приведены в десятичном представлении, и расположены построчно (по 16 в строке), начиная с 0-й степени, и заканчивая 255-й. Заметим, что 255-я степень эквивалентна 0-й степени и равна 1 в силу свойств конечных полей Галуа $GF(p^m)$: $\alpha^{(p^m-1)} = \alpha^0 \Rightarrow \alpha^{(2^8-1)} = \alpha^0$.

Примечание 1. Для того, чтобы выбрать в таблице требуемую степень 2^k, необходимо выбрать строку и столбец таким образом, чтобы сумма индексов строки и столбца (индексы строк приведены в левом заголовочном столбце серого цвета, индексы столбцов – в верхней заголовочной строке серого цвета) была равна показателю степени *k*.

Таблица 1.1. Таблица степеней 2^k для поля Галуа $GF(2^8)$.

	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	1

Также ниже в таблице 1.2 (в виде матрицы 16 х 16) приведены логарифмы по основанию примитивного элемента $(\alpha)_{10} = 2$ поля $GF(2^8)$. Логарифмы расположены построчно (по 16 в строке) для всех элементов поля, начиная с $(a)_{10} = 0$, заканчивая $(a)_{10} = 255$. Заметим, что логарифм от 0 не существует (соответствующая ячейка «N/A»).

Таблица 1.2. Таблица логарифмов $\log_2 a$ для поля Галуа $GF(2^8)$.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	N/A	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

Примечание 2. Для того, чтобы выбрать в таблице логарифм $\log_2 a$ заданного элемента a поля $GF(2^8)$, необходимо выбрать строку и столбец таким образом, чтобы сумма индексов строки и столбца (индексы строк приведены в левом заголовочном столбце серого цвета, индексы столбцов – в верхней заголовочной строке серого цвета) была равна десятичному представлению (эквиваленту) элемента a.

Тогда с учетом всего вышесказанного имеем арифметику поля Галуа $GF(2^8)$, образованного с помощью неприводимого многочлена $p(x) = x^8 + x^4 + x^3 + x^2 + 1$, при котором примитивным элементом поля является элемент $(\alpha)_{10} = 2$:

- $\underbrace{a \pm b}_{GF(2^8)} = \left(\left(a_7 \oplus b_7\right) \ldots \left(a_0 \oplus b_0\right)\right)_2 = a \oplus b$
- $$\overbrace{a \cdot b}^{GF(2^8)} = \begin{cases} 2^{\overbrace{(\log_2 a + \log_2 b) \bmod (2^8 - 1)}^{< R, \{+,\cdot\} >}} & a \neq 0 \,\&\, b \neq 0 \\ 0 & a = 0 \vee b = 0 \end{cases} \qquad (2.6.3)$$
- $$\overbrace{a / b}^{GF(2^8)} = \begin{cases} 2^{\overbrace{(\log_2 a + ((2^8 - 1) - \log_2 b)) \bmod (2^8 - 1)}^{< R, \{+,\cdot\} >}} & a \neq 0 \,\&\, b \neq 0 \\ 0 & a = 0 \,\&\, b \neq 0 \\ \textit{Ошибка} & b = 0 \end{cases}$$

Кроме того, если необходимо найти обратный элемент по умножению, то можно воспользоваться упрощенным вариантом формулы деления элементов:

- $$\overbrace{a^{-1}}^{GF(2^8)} = \begin{cases} 2^{\overbrace{((2^8 - 1) - \log_2 a) \bmod (2^8 - 1)}^{< R, \{+,\cdot\} >}} & a \neq 0 \\ \textit{Ошибка} & a = 0 \end{cases}. \qquad (2.6.4)$$

Наконец, для возведения в степень v заданного элемента a поля $GF(2^8)$ можно использовать формулу, используя операцию умножения для кольца логарифмов $\forall u, v \in LR(p^m - 1) \Rightarrow \underbrace{u \cdot v}_{LR(p^m - 1)} = \underbrace{(u \cdot v) \bmod (p^m - 1)}_{< R, \{+,\cdot\} >}$, и полагая $u = \log_2 a$:

- $$\overbrace{a^v}^{GF(2^8)} = \begin{cases} 2^{\overbrace{(v \cdot \log_2 a) \bmod (2^8 - 1)}^{< R, \{+,\cdot\} >}} & a \neq 0 \,\&\, v \geq 0 \\ 0 & a = 0 \,\&\, v > 0 \\ 1 & a = 0 \,\&\, v = 0 \end{cases} \qquad (2.6.5)$$

Примечание 3. Отметим, что возведение в степень v заданного элемента a поля $GF(2^8)$ по вышеприведенной формуле с аппаратной точки зрения требует сложного в реализации узла умножения логарифмов (умножения целых неотрицательных чисел по модулю $2^8 - 1$). В действительности, на практике операцию возведения в степень удается избежать при кодировании / декодировании информации за счет использование специальных подходов, например, использование схемы Горнера [4] для вычисления значений полиномов при подстановке в них в качестве аргумента элемента поля $GF(2^8)$.

Примечание 4. Особо отметим, что при использовании десятичного представления элементов поля $GF(2^8)$, и элементы и логарифмы выглядят как десятичные числа, и следует соблюдать особую осторожность, и всегда знать, где логарифмы, а где элементы поля.

Пример 1. Найдем сумму элементов расширенного поля $GF(2^8)$, представленных в виде соответствующих чисел «123» и «231» в десятичной системе счисления. Имеем,

$$\underbrace{(123)_{10}+(231)_{10}}_{GF(2^8)}=\underbrace{(01111011)_2+(11100111)_2}_{GF(2^8)}=\left\{\begin{array}{c|c|c|c|c|c|c|c}\oplus\begin{array}{c}0\\1\end{array} & \oplus\begin{array}{c}1\\1\end{array} & \oplus\begin{array}{c}1\\1\end{array} & \oplus\begin{array}{c}1\\0\end{array} & \oplus\begin{array}{c}1\\0\end{array} & \oplus\begin{array}{c}0\\1\end{array} & \oplus\begin{array}{c}1\\1\end{array} & \oplus\begin{array}{c}1\\1\end{array}\\ \hline 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0\end{array}\right\}=$$

$=(10011100)_2=(156)_{10}$. Таким образом, $\underbrace{(123)_{10}+(231)_{10}}_{GF(2^8)}=(156)_{10}$.

Пример 2. Найдем произведение элементов поля $GF(2^8)$, представленных в виде соответствующих чисел «20» и «11» в десятичной системе счисления. По «таблице логарифмов» для поля $GF(2^8)$ имеем логарифмы элементов: $\log_2(20)_{10}=52$ и $\log_2(11)_{10}=238$. Тогда получаем: $\overbrace{(20)_{10}\cdot(11)_{10}}^{GF(2^8)}=2^{\overbrace{(52+238)\bmod(255)}^{<R,\{+,\cdot\}>}}=2^{35}$. По «таблице степеней» для поля $GF(2^8)$ имеем $2^{35}=(156)_{10}$. Таким образом: $\overbrace{(20)_{10}\cdot(11)_{10}}^{GF(2^8)}=(156)_{10}$.

Пример 3. Найдем отношение элементов поля $GF(2^8)$, представленных в виде соответствующих чисел «220» и «127» в десятичной системе счисления. По «таблице логарифмов» для поля $GF(2^8)$ имеем логарифмы элементов: $\log_2(220)_{10}=187$ и $\log_2(127)_{10}=87$. Тогда получаем: $\overbrace{(220)_{10}/(127)_{10}}^{GF(2^8)}=2^{\overbrace{(187+(255-87))\bmod(255)}^{<R,\{+,\cdot\}>}}=2^{100}$. По «таблице степеней» имеем $2^{100}=(17)_{10}$. Таким образом: $\overbrace{(220)_{10}/(127)_{10}}^{GF(2^8)}=(17)_{10}$.

Пример 4. Найдем обратный элемент по умножению для элемента «111», представленного в десятичном виде. По «таблице логарифмов» имеем логарифм элемента: $\log_2(111)_{10}=61$. Тогда обратный элемент: $\overbrace{((111)_{10})^{-1}}^{GF(2^8)}=2^{\overbrace{(255-61)\bmod(255)}^{<R,\{+,\cdot\}>}}=2^{194}$. По «таблице степеней» имеем $2^{194}=(50)_{10}$. Таким образом: $\overbrace{((111)_{10})^{-1}}^{GF(2^8)}=(50)_{10}$.

Пример 5. Возведем элемент «13», представленный в десятичном виде, в заданную степень v = 17. По «таблице логарифмов» для поля $GF(2^8)$ имеем логарифм элемента: $\log_2(13)_{10}=104$. Тогда по формуле получаем: $\overbrace{((13)_{10})^{17}}^{GF(2^8)}=2^{\overbrace{(17\cdot104)\bmod(255)}^{<R,\{+,\cdot\}>}}=2^{238}$. По «таблице степеней» имеем $2^{238}=(11)_{10}$. Таким образом, $\overbrace{((13)_{10})^{17}}^{GF(2^8)}=(11)_{10}$.

Наконец, отметим, что также как и в общем случае полей Галуа $GF(2^m)$ в случае необходимости применения быстрого и компактного аппаратного умножителя элементов поля Галуа $GF(2^8)$, образованного на базе примитивного неприводимого многочлена $p(x)=x^8+x^4+x^3+x^2+1$, для построения такого умножителя можно использовать базовое определение для операции умножения элементов полей многочленов:

$$\underbrace{a \cdot b}_{GF(2^8)} = \underbrace{(a(x) \cdot b(x)) \bmod p(x)}_{GF(2)} = c_7 \cdot x^7 + c_6 \cdot x^6 + c_5 \cdot x^5 + c_4 \cdot x^4 + c_3 \cdot x^3 + c_2 \cdot x^2 + c_1 \cdot x + c_0$$

$$\begin{cases}
c_0 = a_0 \cdot b_0 \oplus a_1 \cdot b_7 \oplus a_2 \cdot b_6 \oplus a_3 \cdot b_5 \oplus a_4 \cdot b_4 \oplus a_5 \cdot b_3 \oplus a_5 \cdot b_7 \oplus a_6 \cdot b_2 \oplus a_6 \cdot b_6 \oplus a_6 \cdot b_7 \oplus a_7 \cdot b_1 \\
\quad \oplus a_7 \cdot b_5 \oplus a_7 \cdot b_6 \oplus a_7 \cdot b_7 \\
c_1 = a_0 \cdot b_1 \oplus a_1 \cdot b_0 \oplus a_2 \cdot b_7 \oplus a_3 \cdot b_6 \oplus a_4 \cdot b_5 \oplus a_5 \cdot b_4 \oplus a_6 \cdot b_3 \oplus a_6 \cdot b_7 \oplus a_7 \cdot b_2 \oplus a_7 \cdot b_6 \oplus a_7 \cdot b_7 \\
c_2 = a_0 \cdot b_2 \oplus a_1 \cdot b_1 \oplus a_1 \cdot b_7 \oplus a_2 \cdot b_0 \oplus a_2 \cdot b_6 \oplus a_3 \cdot b_5 \oplus a_3 \cdot b_7 \oplus a_4 \cdot b_4 \oplus a_4 \cdot b_6 \oplus a_5 \cdot b_3 \oplus a_5 \cdot b_5 \\
\quad \oplus a_5 \cdot b_7 \oplus a_6 \cdot b_2 \oplus a_6 \cdot b_4 \oplus a_6 \cdot b_6 \oplus a_6 \cdot b_7 \oplus a_7 \cdot b_1 \oplus a_7 \cdot b_3 \oplus a_7 \cdot b_5 \oplus a_7 \cdot b_6 \\
c_3 = a_0 \cdot b_3 \oplus a_1 \cdot b_2 \oplus a_1 \cdot b_7 \oplus a_2 \cdot b_1 \oplus a_2 \cdot b_6 \oplus a_2 \cdot b_7 \oplus a_3 \cdot b_0 \oplus a_3 \cdot b_5 \oplus a_3 \cdot b_6 \oplus a_4 \cdot b_4 \oplus a_4 \cdot b_5 \\
\quad \oplus a_4 \cdot b_7 \oplus a_5 \cdot b_3 \oplus a_5 \cdot b_4 \oplus a_5 \cdot b_6 \oplus a_5 \cdot b_7 \oplus a_6 \cdot b_2 \oplus a_6 \cdot b_3 \oplus a_6 \cdot b_5 \oplus a_6 \cdot b_6 \oplus a_7 \cdot b_1 \oplus a_7 \cdot b_2 \\
\quad \oplus a_7 \cdot b_4 \oplus a_7 \cdot b_5 \\
c_4 = a_0 \cdot b_4 \oplus a_1 \cdot b_3 \oplus a_1 \cdot b_7 \oplus a_2 \cdot b_2 \oplus a_2 \cdot b_6 \oplus a_2 \cdot b_7 \oplus a_3 \cdot b_1 \oplus a_3 \cdot b_5 \oplus a_3 \cdot b_6 \oplus a_3 \cdot b_7 \oplus a_4 \cdot b_0 \\
\quad \oplus a_4 \cdot b_4 \oplus a_4 \cdot b_5 \oplus a_4 \cdot b_6 \oplus a_5 \cdot b_3 \oplus a_5 \cdot b_4 \oplus a_5 \cdot b_5 \oplus a_6 \cdot b_2 \oplus a_6 \cdot b_3 \oplus a_6 \cdot b_4 \oplus a_7 \cdot b_1 \oplus a_7 \cdot b_2 \\
\quad \oplus a_7 \cdot b_3 \oplus a_7 \cdot b_7 \\
c_5 = a_0 \cdot b_5 \oplus a_1 \cdot b_4 \oplus a_2 \cdot b_3 \oplus a_2 \cdot b_7 \oplus a_3 \cdot b_2 \oplus a_3 \cdot b_6 \oplus a_3 \cdot b_7 \oplus a_4 \cdot b_1 \oplus a_4 \cdot b_5 \oplus a_4 \cdot b_6 \oplus a_4 \cdot b_7 \\
\quad \oplus a_5 \cdot b_0 \oplus a_5 \cdot b_4 \oplus a_5 \cdot b_5 \oplus a_5 \cdot b_6 \oplus a_6 \cdot b_3 \oplus a_6 \cdot b_4 \oplus a_6 \cdot b_5 \oplus a_7 \cdot b_2 \oplus a_7 \cdot b_3 \oplus a_7 \cdot b_4 \\
c_6 = a_0 \cdot b_6 \oplus a_1 \cdot b_5 \oplus a_2 \cdot b_4 \oplus a_3 \cdot b_3 \oplus a_3 \cdot b_7 \oplus a_4 \cdot b_2 \oplus a_4 \cdot b_6 \oplus a_4 \cdot b_7 \oplus a_5 \cdot b_1 \oplus a_5 \cdot b_5 \oplus a_5 \cdot b_6 \\
\quad \oplus a_5 \cdot b_7 \oplus a_6 \cdot b_0 \oplus a_6 \cdot b_4 \oplus a_6 \cdot b_5 \oplus a_6 \cdot b_6 \oplus a_7 \cdot b_3 \oplus a_7 \cdot b_4 \oplus a_7 \cdot b_5 \\
c_7 = a_0 \cdot b_7 \oplus a_1 \cdot b_6 \oplus a_2 \cdot b_5 \oplus a_3 \cdot b_4 \oplus a_4 \cdot b_3 \oplus a_4 \cdot b_7 \oplus a_5 \cdot b_2 \oplus a_5 \cdot b_6 \oplus a_5 \cdot b_7 \oplus a_6 \cdot b_1 \oplus a_6 \cdot b_5 \\
\quad \oplus a_6 \cdot b_6 \oplus a_6 \cdot b_7 \oplus a_7 \cdot b_0 \oplus a_7 \cdot b_4 \oplus a_7 \cdot b_5 \oplus a_7 \cdot b_6
\end{cases}$$

Функциональную схему умножителя мы не будем приводить в силу ее громоздкости. Отметим лишь, что ее несложно построить на базе логической матрицы, содержащей 64 двухвходовых логических элементов «И» и 8 многовходовых сумматоров по модулю 2.

Также в случае необходимости применения быстрого аппаратного вычислителя обратного элемента по умножению, можно использовать ПЗУ, хранящее таблицу обратных элементов по умножению. Ниже в таблице 1.3 (в виде матрицы 16 x 16) приведены обратные элементы по умножению для элементов поля Галуа $GF(2^8)$, представленные в десятичном виде. Обратные элементы по умножению расположены построчно (по 16 в строке) для всех элементов поля, начиная с $(a)_{10} = 0$, заканчивая $(a)_{10} = 255$. Заметим, что обратного элемента по умножению для нуля не существует (соответствующая ячейка «N/A»).

Таблица 1.3. Таблица обратных элементов по умножению для поля Галуа $GF(2^8)$.

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

Поле Галуа $GF(2^8)$ в криптографическом алгоритме AES. Конечные поля Галуа находят свое применение не только в помехоустойчивом кодировании информации, но и криптографии для защиты информации. В частности поле Галуа $GF(2^8)$ используется в алгоритме шифрования AES – Advanced Encryption Standard [15].

Поле Галуа $GF(2^8)$ содержит 256 элементов:

$GF(2^8)$:	$a(x)$:	0	1	x	…	$x^7+x^6+x^5+x^4+x^3+x^2+x+1$
	$(a)_2$:	00000000	00000001	00000010	…	11111111
	$(a)_{10}$:	0	1	2	…	255

Поле Галуа $GF(2^8)$, по определению являющееся полем многочленов вида $a(x)=a_7\cdot x^7+\ldots+a_1\cdot x+a_0$, образуется на базе простого поля Галуа $GF(2)$ и неприводимого многочлена 8-й степени. В алгоритме шифрования AES используется неприводимый многочлен следующего вида:

$$p(x)=x^8+x^4+x^3+x+1 \qquad (2.7.1)$$

В двоичном представлении неприводимый многочлен выглядит как: $(p)_2=100011011$, а в десятичном представлении: $(p)_{10}=283$. Наименьшим примитивным элементом поля $GF(2^8)$ является элемент $\alpha(x)=x+1$, и при помощи него можно получить все ненулевые элементы поля. В двоичном представлении примитивный элемент поля выглядит как $(\alpha)_2=11$, а в десятичном представлении, соответственно, как $(\alpha)_{10}=3$.

Для формирования таблицы степеней примитивного элемента $\alpha(x)=x+1$ используется процедура, являющаяся модификацией процедуры, используемой для случая поля Галуа $GF(2^8)$, образуемого при помощи другого неприводимого многочлена $p(x)=x^8+x^4+x^3+x^2+1$, при котором наименьшим примитивным элементом поля является элемент $\alpha(x)=x$. Модификация заключается в том, что на каждой итерации «предыдущую» степень $\alpha^{k-1}(x)$ необходимо умножать на двучлен $\alpha(x)=x+1$. При двоичном представлении элементов поля операцию умножения на $(\alpha)_2=11$ в поле Галуа $GF(2^8)$ можно свести к операции сдвига влево на один разряд и операции сложения («побитового» XOR), причем в случае если старший бит «предыдущей» степени был ненулевым, то еще выполняется «побитовый» XOR с двоичным эквивалентом неприводимого многочлена. Таблицу логарифмов можно формировать параллельно с формированием таблицы степеней, используя двоичное или десятичное представление степеней примитивного элемента в качестве индексов таблицы логарифмов:

$$\begin{cases} k=1\ldots 2^8-2;\quad \alpha^0=1;\quad \log_\alpha(\alpha^0)=0 \\ \alpha^k=\begin{cases}((\alpha^{k-1}<<1)\oplus\alpha^{k-1}),\quad \alpha_7^{(k-1)}=0\\((\alpha^{k-1}<<1)\oplus\alpha^{k-1})\oplus(100011011)_2,\quad \alpha_7^{(k-1)}=1\end{cases} \\ \log_\alpha(\alpha^k)=k \end{cases} \qquad (2.7.2)$$

Под выражением $((\alpha^{k-1}<<1)\oplus\alpha^{k-1})$ понимается сдвиг двоичного числа влево на один разряд с последующей операцией «побитового» XOR результата сдвига с самим числом. Под выражением $((\alpha^{k-1}<<1)\oplus\alpha^{k-1})\oplus(100011011)_2$ понимается сдвиг двоичного числа влево на один разряд с последующей операцией «побитового» XOR результата сдвига с самим числом, с последующей операцией «побитового» XOR результата с двоичным эквивалентом неприводимого многочлена.

Примечание. Поскольку в десятичном виде примитивный элемент поля $GF(2^8)$ выглядит как $(\alpha)_{10} = 3$, то будем также обозначать *k*-ую степень примитивного элемента как 3^k, а логарифм от элемента *a* по основанию примитивного элемента как $\log_3 a$.

Ниже в таблице 2.1 (в виде матрицы 16 x 16) приведены степени примитивного элемента $(\alpha)_{10} = 3$ поля $GF(2^8)$, образованного при помощи неприводимого многочлена $p(x) = x^8 + x^4 + x^3 + x + 1$. Степени примитивного элемента для компактности приведены в десятичном представлении, и расположены построчно (по 16 в строке), начиная с 0-й степени, и заканчивая 255-й. Заметим, что 255-я степень эквивалентна 0-й степени и равна 1 в силу свойств конечных полей Галуа $GF(p^m)$: $\alpha^{(p^m-1)} = \alpha^0 \Rightarrow \alpha^{(2^8-1)} = \alpha^0$.

Таблица 2.1. AES-таблица степеней 3^k для поля Галуа $GF(2^8)$.

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

Также ниже в таблице 2.2 (в виде матрицы 16 x 16) приведены логарифмы по основанию примитивного элемента $(\alpha)_{10} = 3$ поля $GF(2^8)$. Логарифмы расположены построчно (по 16 в строке) для всех элементов поля, начиная с $(a)_{10} = 0$, заканчивая $(a)_{10} = 255$. Заметим, что логарифм от 0 не существует (соответствующая ячейка «N/A»).

Таблица 2.2. AES-таблица логарифмов $\log_3 a$ для поля Галуа $GF(2^8)$.

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

Тогда с учетом всего вышесказанного имеем арифметику поля Галуа $GF(2^8)$, образованного с помощью неприводимого многочлена $p(x) = x^8 + x^4 + x^3 + x + 1$, при котором примитивным элементом поля является элемент $(\alpha)_{10} = 3$:

- $\underbrace{a \pm b}_{GF(2^8)} = \left(\left(a_7 \oplus b_7 \right) \ldots \left(a_0 \oplus b_0 \right) \right)_2 = a \oplus b$
- $\underbrace{a \cdot b}_{GF(2^8)} = \begin{cases} 3^{\overbrace{(\log_3 a + \log_3 b) \bmod (2^8 - 1)}^{< R, \{+,\cdot\} >}} & a \neq 0 \,\&\, b \neq 0 \\ 0 & a = 0 \vee b = 0 \end{cases}$ (2.7.3)
- $\underbrace{a / b}_{GF(2^8)} = \begin{cases} 3^{\overbrace{(\log_3 a + ((2^8 - 1) - \log_3 b)) \bmod (2^8 - 1)}^{< R, \{+,\cdot\} >}} & a \neq 0 \,\&\, b \neq 0 \\ 0 & a = 0 \,\&\, b \neq 0 \\ \textit{Ошибка} & b = 0 \end{cases}$

Кроме того, если необходимо найти обратный элемент по умножению, то можно воспользоваться упрощенным вариантом формулы деления элементов:

- $\underbrace{a^{-1}}_{GF(2^8)} = \begin{cases} 3^{\overbrace{((2^8 - 1) - \log_3 a) \bmod (2^8 - 1)}^{< R, \{+,\cdot\} >}} & a \neq 0 \\ \textit{Ошибка} & a = 0 \end{cases}$. (2.7.4)

Наконец, для возведения в степень v заданного элемента a поля $GF(2^8)$ можно использовать формулу, используя операцию умножения для кольца логарифмов $\forall u, v \in LR(p^m - 1) \Rightarrow \underbrace{u \cdot v}_{LR(p^m - 1)} = \underbrace{(u \cdot v) \bmod (p^m - 1)}_{< R, \{+,\cdot\} >}$, и полагая $u = \log_3 a$:

- $\underbrace{a^v}_{GF(2^8)} = \begin{cases} 3^{\overbrace{(v \cdot \log_3 a) \bmod (2^8 - 1)}^{< R, \{+,\cdot\} >}} & a \neq 0 \,\&\, v \geq 0 \\ 0 & a = 0 \,\&\, v > 0 \\ 1 & a = 0 \,\&\, v = 0 \end{cases}$ (2.7.5)

Примечание 3. Отметим, что возведение в степень v заданного элемента a поля $GF(2^8)$ по вышеприведенной формуле с аппаратной точки зрения требует сложного в реализации узла умножения логарифмов (умножения целых неотрицательных чисел по модулю $2^8 - 1$). В действительности, на практике операцию возведения в степень удается избежать при кодировании / декодировании информации за счет использование специальных подходов, например, использование схемы Горнера для вычисления значений полиномов при подстановке в них в качестве аргумента элемента поля $GF(2^8)$.

Примечание 4. Особо отметим, что при использовании десятичного представления элементов поля $GF(2^8)$, и элементы и логарифмы выглядят как десятичные числа, и следует соблюдать особую осторожность, и всегда знать, где логарифмы, а где элементы поля.

Пример 1. Найдем сумму элементов расширенного поля $GF(2^8)$, представленных в виде соответствующих чисел «87» и «131» в десятичной системе счисления. Имеем,

$$\underbrace{(87)_{10}+(131)_{10}}_{GF(2^8)}=\underbrace{(01010111)_2+(10000011)_2}_{GF(2^8)}=\left\{\begin{array}{c|c|c|c|c|c|c|c}\oplus\begin{array}{c}0\\1\end{array} & \oplus\begin{array}{c}1\\0\end{array} & \oplus\begin{array}{c}0\\0\end{array} & \oplus\begin{array}{c}1\\0\end{array} & \oplus\begin{array}{c}0\\0\end{array} & \oplus\begin{array}{c}1\\0\end{array} & \oplus\begin{array}{c}1\\1\end{array} & \oplus\begin{array}{c}1\\1\end{array}\\ \hline 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0\end{array}\right\}=$$

$=(11010100)_2=(212)_{10}$. Таким образом, $\underbrace{(87)_{10}+(131)_{10}}_{GF(2^8)}=(212)_{10}$.

Пример 2. Найдем произведение элементов поля $GF(2^8)$, представленных в виде соответствующих чисел «87» и «131» в десятичной системе счисления. По «таблице логарифмов» для поля $GF(2^8)$ имеем логарифмы элементов: $\log_3(87)_{10}=98$ и $\log_3(131)_{10}=80$. Тогда получаем: $\overbrace{(87)_{10}\cdot(131)_{10}}^{GF(2^8)}=3^{\overbrace{(98+80)\bmod(255)}^{<R,\{+,\cdot\}>}}=3^{178}$. По «таблице степеней» для поля $GF(2^8)$ имеем $3^{178}=(193)_{10}$. Таким образом: $\overbrace{(87)_{10}\cdot(131)_{10}}^{GF(2^8)}=(193)_{10}$.

Пример 3. Найдем отношение элементов поля $GF(2^8)$, представленных в виде соответствующих чисел «131» и «193» в десятичной системе счисления. По «таблице логарифмов» для поля $GF(2^8)$ имеем логарифмы элементов: $\log_3(131)_{10}=80$ и $\log_3(193)_{10}=178$. Тогда получаем: $\overbrace{(131)_{10}/(193)_{10}}^{GF(2^8)}=3^{\overbrace{(80+(255-178))\bmod(255)}^{<R,\{+,\cdot\}>}}=3^{157}$. По «таблице степеней» имеем $3^{157}=(191)_{10}$. Таким образом: $\overbrace{(131)_{10}/(193)_{10}}^{GF(2^8)}=(191)_{10}$.

Пример 4. Найдем обратный элемент по умножению для элемента «191», представленного в десятичном виде. По «таблице логарифмов» имеем логарифм элемента: $\log_3(191)_{10}=157$. Тогда обратный элемент: $\overbrace{((191)_{10})^{-1}}^{GF(2^8)}=3^{\overbrace{(255-157)\bmod(255)}^{<R,\{+,\cdot\}>}}=3^{98}$. По «таблице степеней» имеем $3^{98}=(87)_{10}$. Таким образом: $\overbrace{((191)_{10})^{-1}}^{GF(2^8)}=(87)_{10}$.

Пример 5. Возведем элемент «13», представленный в десятичном виде, в заданную степень v = 17. По «таблице логарифмов» для поля $GF(2^8)$ имеем логарифм элемента: $\log_3(13)_{10}=238$. Тогда по формуле получаем: $\overbrace{((13)_{10})^{17}}^{GF(2^8)}=3^{\overbrace{(17\cdot 238)\bmod(255)}^{<R,\{+,\cdot\}>}}=3^{221}$. По «таблице степеней» имеем $3^{221}=(81)_{10}$. Таким образом, $\overbrace{((13)_{10})^{17}}^{GF(2^8)}=(81)_{10}$.

Наконец, отметим, что также как и в общем случае полей Галуа $GF(2^m)$ в случае необходимости применения быстрого и компактного аппаратного умножителя элементов поля Галуа $GF(2^8)$, образованного на базе примитивного неприводимого многочлена $p(x)=x^8+x^4+x^3+x+1$, для построения такого умножителя можно использовать базовое определение для операции умножения элементов полей многочленов:

$$\underbrace{a \cdot b}_{GF(2^8)} = \underbrace{(a(x) \cdot b(x)) \bmod p(x)}_{GF(2)} = c_7 \cdot x^7 + c_6 \cdot x^6 + c_5 \cdot x^5 + c_4 \cdot x^4 + c_3 \cdot x^3 + c_2 \cdot x^2 + c_1 \cdot x + c_0$$

$$\begin{cases}
c_0 = a_0 \cdot b_0 \oplus a_1 \cdot b_7 \oplus a_2 \cdot b_6 \oplus a_3 \cdot b_5 \oplus a_4 \cdot b_4 \oplus a_5 \cdot b_3 \oplus a_5 \cdot b_7 \oplus a_6 \cdot b_2 \oplus a_6 \cdot b_6 \oplus a_6 \cdot b_7 \oplus a_7 \cdot b_1 \\
\quad \oplus a_7 \cdot b_5 \oplus a_7 \cdot b_6 \\
c_1 = a_0 \cdot b_1 \oplus a_1 \cdot b_0 \oplus a_1 \cdot b_7 \oplus a_2 \cdot b_6 \oplus a_2 \cdot b_7 \oplus a_3 \cdot b_5 \oplus a_3 \cdot b_6 \oplus a_4 \cdot b_4 \oplus a_4 \cdot b_5 \oplus a_5 \cdot b_3 \oplus a_5 \cdot b_4 \\
\quad \oplus a_5 \cdot b_7 \oplus a_6 \cdot b_2 \oplus a_6 \cdot b_3 \oplus a_6 \cdot b_6 \oplus a_7 \cdot b_1 \oplus a_7 \cdot b_2 \oplus a_7 \cdot b_5 \oplus a_7 \cdot b_7 \\
c_2 = a_0 \cdot b_2 \oplus a_1 \cdot b_1 \oplus a_2 \cdot b_0 \oplus a_2 \cdot b_7 \oplus a_3 \cdot b_6 \oplus a_3 \cdot b_7 \oplus a_4 \cdot b_5 \oplus a_4 \cdot b_6 \oplus a_5 \cdot b_4 \oplus a_5 \cdot b_5 \oplus a_6 \cdot b_3 \\
\quad \oplus a_6 \cdot b_4 \oplus a_6 \cdot b_7 \oplus a_7 \cdot b_2 \oplus a_7 \cdot b_3 \oplus a_7 \cdot b_6 \\
c_3 = a_0 \cdot b_3 \oplus a_1 \cdot b_2 \oplus a_1 \cdot b_7 \oplus a_2 \cdot b_1 \oplus a_2 \cdot b_6 \oplus a_3 \cdot b_0 \oplus a_3 \cdot b_5 \oplus a_3 \cdot b_7 \oplus a_4 \cdot b_4 \oplus a_4 \cdot b_6 \oplus a_4 \cdot b_7 \\
\quad \oplus a_5 \cdot b_3 \oplus a_5 \cdot b_5 \oplus a_5 \cdot b_6 \oplus a_5 \cdot b_7 \oplus a_6 \cdot b_2 \oplus a_6 \cdot b_4 \oplus a_6 \cdot b_5 \oplus a_6 \cdot b_6 \oplus a_6 \cdot b_7 \oplus a_7 \cdot b_1 \oplus a_7 \cdot b_3 \\
\quad \oplus a_7 \cdot b_4 \oplus a_7 \cdot b_5 \oplus a_7 \cdot b_6 \oplus a_7 \cdot b_7 \\
c_4 = a_0 \cdot b_4 \oplus a_1 \cdot b_3 \oplus a_1 \cdot b_7 \oplus a_2 \cdot b_2 \oplus a_2 \cdot b_6 \oplus a_2 \cdot b_7 \oplus a_3 \cdot b_1 \oplus a_3 \cdot b_5 \oplus a_3 \cdot b_6 \oplus a_4 \cdot b_0 \oplus a_4 \cdot b_4 \\
\quad \oplus a_4 \cdot b_5 \oplus a_4 \cdot b_7 \oplus a_5 \cdot b_3 \oplus a_5 \cdot b_4 \oplus a_5 \cdot b_6 \oplus a_6 \cdot b_2 \oplus a_6 \cdot b_3 \oplus a_6 \cdot b_5 \oplus a_7 \cdot b_1 \oplus a_7 \cdot b_2 \oplus a_7 \cdot b_4 \\
\quad \oplus a_7 \cdot b_7 \\
c_5 = a_0 \cdot b_5 \oplus a_1 \cdot b_4 \oplus a_2 \cdot b_3 \oplus a_2 \cdot b_7 \oplus a_3 \cdot b_2 \oplus a_3 \cdot b_6 \oplus a_3 \cdot b_7 \oplus a_4 \cdot b_1 \oplus a_4 \cdot b_5 \oplus a_4 \cdot b_6 \oplus a_5 \cdot b_0 \\
\quad \oplus a_5 \cdot b_4 \oplus a_5 \cdot b_5 \oplus a_5 \cdot b_7 \oplus a_6 \cdot b_3 \oplus a_6 \cdot b_4 \oplus a_6 \cdot b_6 \oplus a_7 \cdot b_2 \oplus a_7 \cdot b_3 \oplus a_7 \cdot b_5 \\
c_6 = a_0 \cdot b_6 \oplus a_1 \cdot b_5 \oplus a_2 \cdot b_4 \oplus a_3 \cdot b_3 \oplus a_3 \cdot b_7 \oplus a_4 \cdot b_2 \oplus a_4 \cdot b_6 \oplus a_4 \cdot b_7 \oplus a_5 \cdot b_1 \oplus a_5 \cdot b_5 \oplus a_5 \cdot b_6 \\
\quad \oplus a_6 \cdot b_0 \oplus a_6 \cdot b_4 \oplus a_6 \cdot b_5 \oplus a_6 \cdot b_7 \oplus a_7 \cdot b_3 \oplus a_7 \cdot b_4 \oplus a_7 \cdot b_6 \\
c_7 = a_0 \cdot b_7 \oplus a_1 \cdot b_6 \oplus a_2 \cdot b_5 \oplus a_3 \cdot b_4 \oplus a_4 \cdot b_3 \oplus a_4 \cdot b_7 \oplus a_5 \cdot b_2 \oplus a_5 \cdot b_6 \oplus a_5 \cdot b_7 \oplus a_6 \cdot b_1 \oplus a_6 \cdot b_5 \\
\quad \oplus a_6 \cdot b_6 \oplus a_7 \cdot b_0 \oplus a_7 \cdot b_4 \oplus a_7 \cdot b_5 \oplus a_7 \cdot b_7
\end{cases}$$

Функциональную схему умножителя мы не будем приводить в силу ее громоздкости. Отметим лишь, что ее несложно построить на базе логической матрицы, содержащей 64 двухвходовых логических элементов «И» и 8 многовходовых сумматоров по модулю 2.

Также в случае необходимости применения быстрого аппаратного вычислителя обратного элемента по умножению, можно использовать ПЗУ, хранящее таблицу обратных элементов по умножению. Ниже в таблице 2.3 (в виде матрицы 16 x 16) приведены обратные элементы по умножению для элементов поля Галуа $GF(2^8)$, представленные в десятичном виде. Обратные элементы по умножению расположены построчно (по 16 в строке) для всех элементов поля, начиная с $(a)_{10} = 0$, заканчивая $(a)_{10} = 255$. Заметим, что обратного элемента по умножению для нуля не существует (соответствующая ячейка «N/A»).

Таблица 2.3. AES-таблица обратных элементов по умножению для поля Галуа $GF(2^8)$.

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

Контрольные вопросы

1. Какое преимущество использования примитивных неприводимых многочленов для построения полей $GF(2^m)$ над простым полем $GF(2)$ с точки зрения программной и аппаратной реализации их арифметики?

2. Какой математический смысл у цепи обратной связи в схеме сдвигового регистра, используемого в качестве генератора степеней примитивного элемента поля $GF(2^m)$?

3. Для чего выполняется добавление переноса в младший разряд в схеме сложения двух логарифмов элементов $GF(2^m)$ поля по основанию примитивного элемента?

4. В чем преимущество аппаратных умножителей на базе программируемых логических матриц по сравнению со схемой умножения, использующей логарифмы и степени?

5. Сформируйте таблицу логарифмов и степеней для поля $GF(2^5)$, заданного при помощи примитивного неприводимого многочлена $p(x) = x^5 + x^2 + 1$.

6. Составьте схему быстрого аппаратного умножителя элементов поля $GF(2^5)$, заданного при помощи примитивного неприводимого многочлена $p(x) = x^5 + x^2 + 1$.

7. Вычислите таблицу обратных элементов для всех ненулевых элементов поля $GF(2^5)$, заданного при помощи примитивного неприводимого многочлена $p(x) = x^5 + x^2 + 1$.

8. Вычислите значение выражения $14^7 * 15^6 + 16^3 / 12^8$ в поле $GF(2^5)$, заданного при помощи примитивного неприводимого многочлена $p(x) = x^5 + x^2 + 1$. Элементы поля представлены в виде числовых эквивалентов в десятичной системе счисления.

9. Найдите наибольший (в десятичном представлении) примитивный элемент поля $GF(2^8)$, заданного при помощи неприводимого многочлена $p(x) = x^8 + x^4 + x^3 + x^2 + 1$.

10. Вычислите значение выражения $13^{12} * 5^6 + 56^{23} / 17^3$ в поле $GF(2^8)$, заданного при помощи неприводимого многочлена $p(x) = x^8 + x^4 + x^3 + x^2 + 1$. Элементы поля представлены в виде числовых эквивалентов в десятичной системе счисления.

3. Полиномы над конечными полями GF(2^m) и операции с ними.

Над расширенными полями $GF(p^m)$ можно задавать полиномы заданной степени, так же как и над простыми полями Галуа $GF(p)$ можно задавать многочлены. Мы специально будем использовать термин **полином**, чтобы не путать полиномы над полем $GF(p^m)$ с многочленами над простым полем $GF(p)$, которые мы рассматривали ранее. Кроме того, мы будем рассматривать полиномы, задаваемые конкретно над конечными полями Галуа $GF(2^m)$ характеристики $p = 2$, поскольку именно с такими полиномами в основном имеет дело алгебраическая теория кодирования. А учитывая, что для полей Галуа $GF(2^m)$ операции сложения и вычитания эквивалентны и сводятся к одной и той же операции «побитового» XOR, то в дальнейшем будем использовать знак $\oplus$ вместо знаков сложения и вычитания. Тогда полином, заданный над полем $GF(2^m)$, имеет следующий вид:

$$\Psi(x) = \Psi_{k-1} \cdot x^{k-1} \oplus \ldots \oplus \Psi_1 \cdot x \oplus \Psi_0 = \sum_{i=0}^{k-1} \Psi_i \cdot x^i$$
$$\Psi_i \in GF(2^m); \quad i = 0 \ldots k-1; \tag{3.1}$$

Одна из наиболее важных характеристик полинома – это его степень, определяемая как наивысший показатель степени переменной среди ненулевых слагаемых полинома:

$$\deg(\Psi(x)) = \max_{i \geq 0} \left\{ i : \left(\Psi_i \cdot x^i \neq 0 \right) \right\}$$
$$\Psi_i \in GF(2^m); i \geq 0 \tag{3.2}$$

Над полиномами можно производить алгебраические операции, также на место формальной переменной можно подставлять конкретное значение, являющееся элементом поля Галуа $GF(2^m)$, и вычислять значение функции. При пересчете коэффициентов полинома при выполнении операции или вычислении значении функции, строго соблюдаются правила арифметики для поля $GF(2^m)$. Основные операции, которые нам потребуются в дальнейшем – это сложение полиномов, умножение полиномов, вычисление остатка от деления одного полинома на другой полином и вычисление формальной производной полинома.

Сложение полиномов выполняется путем сложения по правилам арифметики поля Галуа $GF(2^m)$ соответствующих коэффициентов, стоящих перед переменной в одной степени. Отсутствующие в полиномах коэффициенты, считаются нулевыми.

$$\Psi(x) \oplus \xi(x) = \sum_{q=0}^{\max(k-1, l-1)} \left(\Psi_q \oplus \xi_q \right) \cdot x^q$$
$$\Psi_i, \xi_j \in GF(2^m); \quad \deg(\Psi(x)) = k-1; \quad \deg(\xi(x)) = l-1; \tag{3.3}$$

Пример. Сложим полиномы $\Psi(x) = 49 \cdot x^2 \oplus 50 \cdot x \oplus 51$ и $\xi(x) = 19 \cdot x^2 \oplus 93 \cdot x \oplus 1$, заданные над полем $GF(2^8)$, заданного с помощью неприводимого многочлена $p(x) = x^8 + x^4 + x^3 + x^2 + 1$. Используя арифметику поля $GF(2^8)$ для сложения коэффициентов, имеем: $(49 \oplus 19) \cdot x^2 \oplus (50 \oplus 93) \cdot x \oplus (51 \oplus 1) = 34 \cdot x^2 \oplus 111 \cdot x \oplus 50$.

Умножение полиномов осуществляется путем вычисления суммы результатов умножения одного полинома на каждый член другого полинома. Степень результирующего полинома равна сумме степеней перемножаемых полиномов:

$$\Psi(x) \cdot \xi(x) = \sum_{i=0}^{k-1} \Psi_i \cdot x^i \cdot \left(\sum_{j=0}^{l-1} \xi_j \cdot x^j \right) = \sum_{q=0}^{k+l-2} x^q \cdot \sum_{\substack{i+j=q \\ i=0 \ldots k-1 \quad j=0 \ldots l-1}} \left(\Psi_i \cdot \xi_j \right)$$
$$\Psi_i, \xi_j \in GF(2^m); \quad \deg(\Psi(x)) = k-1; \quad \deg(\xi(x)) = l-1; \tag{3.4.1}$$

Пример. Умножим полиномы $\Psi(x) = 49 \cdot x^2 \oplus 50 \cdot x \oplus 51$ и $\xi(x) = 19 \cdot x^2 \oplus 93 \cdot x \oplus 1$, заданные над полем $GF(2^8)$, заданного с помощью неприводимого многочлена $p(x) = x^8 + x^4 + x^3 + x^2 + 1$. Используя арифметику поля $GF(2^8)$ для коэффициентов, имеем:

$(49 \cdot 19) \cdot x^4 \oplus (49 \cdot 93 \oplus 50 \cdot 19) \cdot x^3 \oplus (49 \cdot 1 \oplus 50 \cdot 93 \oplus 51 \cdot 19) \cdot x^2 \oplus (50 \cdot 1 \oplus 51 \cdot 93) \cdot x \oplus 51 \cdot 1 =$

$100 \cdot x^4 \oplus 218 \cdot x^3 \oplus 31 \cdot x^2 \oplus 3 \cdot x \oplus 51$.

Особо отметим, что если мы будем подразумевать, что любые «отсутствующие» коэффициенты в полиномах, в том числе коэффициенты с индексами больше степени полиномов, равны нулю: $\forall i > k-1 : \Psi_i = 0$ и $\forall j > l-1 : \xi_j = 0$, то мы внутреннее суммирование по двум индексам можем преобразовать суммирование по одному индексу. Для этого мы изменим верхние границы для обоих индексов: $i \le q$ и $j \le q$, при этом мы ничего не теряем, поскольку $i \ge 0$, $j \ge 0$ и $i + j = q$. Случаи, когда $i > q$ (при $k-1 > q$), или $j > q$ (при $l-1 > q$), все равно никогда не пройдут условие $i + j = q$, так как $i \ge 0$ и $j \ge 0$. А при $i > k-1$ (при $q > k-1$) коэффициенты $\Psi_i = 0$, а при $j > l-1$ (при $q > l-1$) коэффициенты $\xi_j = 0$. Тогда в итоге мы получаем три условия: $i + j = q$, $0 \le i \le q$ и $0 \le j \le q$. Заметим, что оба индекса неотрицательны и ограничены в одним и тем же верхним пределом, причем сумма индексов также равна верхнему пределу. Тогда, очевидно, можно избавиться от одного из индексов, попросту выразив его через другой: $j = q - i$, и мы ничего не теряем, поскольку $0 \le q - i \le q$, так как $0 \le i \le q$.

В итоге мы получаем следующую формулу произведения:

$$\Psi(x) \cdot \xi(x) = \sum_{q=0}^{k+l-2} x^q \cdot \left(\sum_{i=0}^{q} \left(\Psi_i \cdot \xi_{q-i} \right) \right) \qquad (3.4.2)$$

$$\Psi_i, \xi_j \in GF(2^m); \quad \forall i > \deg(\Psi(x)) = k-1 : \Psi_i = 0; \quad \forall j > \deg(\xi(x)) = l-1 : \xi_j = 0;$$

Умножение полинома на скаляр – умножение полинома на скаляр λ (полином нулевой степени) является частным случаем умножения полиномов, которое сводится к умножению всех коэффициентов этого полинома на этот скаляр – константу, являющуюся элементом поля Галуа $GF(2^m)$:

$$\lambda \cdot \Psi(x) = \sum_{i=0}^{k-1} \left(\lambda \cdot \Psi_i \right) \cdot x^i \qquad (3.5)$$

$$\lambda, \Psi_i \in GF(2^m); \quad \deg(\Psi(x)) = k-1;$$

Умножение полинома на одночлен *x* – умножение полинома на одночлен *x* является частным случаем умножения полиномов. В результате при каждом коэффициенте полинома степень переменной *x* увеличивается на единицу.

$$x \cdot \Psi(x) = \sum_{i=0}^{k-1} \Psi_i \cdot x^{i+1} \qquad (3.6)$$

$$\Psi_i \in GF(2^m); \quad \deg(\Psi(x)) = k-1;$$

Вычисление остатка от деления одного полинома на другой полином, или более коротко – вычисление остатка одного полинома по модулю другого. Пусть над полем $GF(2^m)$ заданы некоторый полином $\Psi(x)$ степени $k-1$ и некоторый ненулевой полином $g(x)$ степени *r*. Тогда существуют полиномы $Q(x)$ и $R(x)$ над полем $GF(2^m)$ такие, что $\Psi(x) = Q(x) \cdot g(x) \oplus R(x)$. Полином $Q(x)$ называется **частным**, а многочлен $R(x)$ называется **остатком от деления** полинома $\Psi(x)$ на полином $g(x)$. Более того, полином $R(x)$ также называют **остатком** $\Psi(x)$ **по модулю** $g(x)$, и обозначают это как: $\Psi(x) \bmod g(x)$.

Отметим, что мы ограничимся специальным частным случаем, используемым в алгебраической теории кодирования, когда старший коэффициент полинома делителя $g(x)$ равен единице, то есть $g_r = 1$, иным словами полином-делитель является нормированным. Также заметим, что фактическое деление требуется только тогда, когда степень полинома $\Psi(x)$ больше либо равна степени полинома $g(x)$, то есть $k-1 \geq r$, а в случае же $k-1 < r$, очевидно, сам полином $\Psi(x)$ является искомым полиномом-остатком $R(x)$.

Следует отметить, что хотя деление полиномов, заданных над простым полем $GF(2^m)$, во многом похоже на деление полиномов, заданных над полем действительных чисел, однако, здесь имеется один существенный момент. При делении полиномов «в столбик» используются операции умножения полинома-делителя на одночлен (в том числе на одночлен нулевой степени, то есть скаляр), а также операция вычитания полиномов. Эти операции, в свою очередь, приводят к операциям умножения и вычитания коэффициентов, которые в рассматриваемом нами случае по определению являются элементами поля $GF(2^m)$. Соответственно, все операции с коэффициентами должны выполняться по правилам арифметики поля $GF(2^m)$, которые мы подробно рассматривали выше. Также отметим, что в арифметике поля $GF(2^m)$ сложение и вычитание сводятся к одной и той же операции «побитового» XOR: $\forall a,b \in GF(2^m) \Rightarrow a-b = a+(-b) = a+b = a \oplus b$.

Пример. Вычислим остаток полинома $\Psi(x) = 49 \cdot x^4 \oplus 50 \cdot x^3 \oplus 51 \cdot x^2 \oplus 0 \cdot x \oplus 0$ по модулю полинома $g(x) = x^2 \oplus 6 \cdot x \oplus 8$, заданных над полем Галуа $GF(2^8)$, заданного с помощью неприводимого многочлена $p(x) = x^8 + x^4 + x^3 + x^2 + 1$.

$$
\begin{array}{l|l}
49 \cdot x^4 \oplus 50 \cdot x^3 \oplus 51 \cdot x^2 \oplus 0 \cdot x \oplus 0 & g(x) = x^2 \oplus 6 \cdot x \oplus 8 \\
\underline{49 \cdot x^4 \oplus 166 \cdot x^3 \oplus 149 \cdot x^2 \oplus 0 \cdot x \oplus 0} & \overline{Q(x) = 49 \cdot x^2 \oplus 148 \cdot x + 249} \\
0 \cdot x^4 \oplus 148 \cdot x^3 \oplus 166 \cdot x^2 \oplus 0 \cdot x \oplus 0 & \\
\underline{0 \cdot x^4 \oplus 148 \cdot x^3 \oplus 95 \cdot x^2 \oplus 212 \cdot x \oplus 0} & \\
0 \cdot x^4 \oplus 0 \cdot x^3 \oplus 249 \cdot x^2 \oplus 212 \cdot x \oplus 0 & \\
\underline{0 \cdot x^4 \oplus 0 \cdot x^3 \oplus 249 \cdot x^2 \oplus 44 \cdot x \oplus 155} & \\
R(x) = 248 \cdot x \oplus 155 &
\end{array}
$$

Таким образом, искомый полином-остаток $R(x) = \Psi(x) \bmod g(x) = 248 \cdot x \oplus 155$.

Проверим корректность операции деления в предыдущем примере. Вычислив выражение $Q(x) \cdot g(x) \oplus R(x)$, очевидно, мы должны снова получить исходное делимое:

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

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

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

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 $\Psi(x)$ в предыдущем примере.

Теперь остановимся на особенностях программной и аппаратной реализации вычисления остатка от полинома $\Psi(x)$ степени $k-1$ по модулю нормированного полинома $g(x)$ степени r, заданных над полем $GF(2^m)$. Перепишем процедуру деления в вышеприведенном примере немного по-другому:

$$R^{(0)}(x) = 0 \cdot x^5 \oplus \overset{\Psi_4}{\overbrace{49}} \cdot x^4 \oplus \overset{\Psi_3}{\overbrace{50}} \cdot x^3$$

$$\leftarrow$$

$$\oplus \begin{array}{r} \overset{\Psi_2}{\overbrace{49 \cdot x^4 \oplus 50 \cdot x^3 \oplus 51}} \cdot x^2 \\ 49 \cdot x^4 \oplus 166 \cdot x^3 \oplus 149 \cdot x^2 \end{array} \quad \left| \begin{array}{r} g(x) = x^2 \oplus 6 \cdot x \oplus 8 \\ Q(x) = 49 \cdot x^2 \oplus 148 \cdot x \oplus 249 \end{array} \right.$$

$$R^{(1)}(x) = 0 \cdot x^4 \oplus 148 \cdot x^3 \oplus 166 \cdot x^2$$

$$\leftarrow$$

$$\oplus \begin{array}{r} \overset{\Psi_1}{\overbrace{148 \cdot x^3 \oplus 166 \cdot x^2 + 0}} \cdot x \\ 148 \cdot x^3 \oplus 95 \cdot x^2 + 212 \cdot x \end{array}$$

$$R^{(2)}(x) = 0 \cdot x^3 \oplus 249 \cdot x^2 \oplus 212 \cdot x$$

$$\leftarrow$$

$$\oplus \begin{array}{r} \overset{\Psi_0}{\overbrace{249 \cdot x^2 \oplus 212 \cdot x \oplus 0}} \\ 249 \cdot x^2 \oplus 44 \cdot x \oplus 155 \end{array}$$

$$R^{(3)}(x) = 0 \cdot x^2 \oplus 248 \cdot x \oplus 155$$

Заметим, что вычисление полинома-остатка $R(x) = \Psi(x) \bmod g(x)$ сводится к итерационной процедуре из $k - r$ шагов, на каждом шаге $s = 1 \ldots k - r$ вычисляется некоторый «остаточный» полином $R^{(s)}(x)$. В качестве начального «остаточного» полинома принимается полином, состоящий из старших *r* слагаемых полинома $\Psi(x)$. Иными словами:

$$R^{(0)}(x) = \Psi_{k-1} \cdot x^{k-1} \oplus \ldots \oplus \Psi_{k-r} \cdot x^{k-r} \tag{3.7.1}$$

Далее, на каждом шаге $s = 1 \ldots k - r$ вычисляется «остаточный» полином $R^{(s)}(x)$ путем выполнения следующий операций:

- К «остаточному» полиному $R^{(s-1)}(x)$, вычисленному на предыдущем шаге, добавляется очередное слагаемое $\Psi_{k-r-s} \cdot x^{k-r-s}$ полинома-делимого $\Psi(x)$. Тем самым, формируется полином $(R^{(s-1)}(x) \oplus \Psi_{k-r-s} \cdot x^{k-r-s})$.
- Выбирается такой очередной коэффициент Q_{k-r-s} для полинома-частного $Q(x)$, чтобы старший коэффициент $R_{k-s}^{(s-1)}$ полинома $(R^{(s-1)}(x) \oplus \Psi_{k-r-s} \cdot x^{k-r-s})$ был равен $Q_{k-r-s} \cdot g_r$. Такой выбор является необходимым условием сходимости процедуры деления. Учитывая то, что полином-делитель $g(x)$ является нормированным, то есть $g_r = 1$, то выбор коэффициента Q_{k-r-s} предельно упрощается, он попросту будет равен $R_{k-s}^{(s-1)}$. Иными словами: $Q_{k-r-s} = R_{k-s}^{(s-1)}$.

- Полином $(R^{(s-1)}(x) \oplus \Psi_{k-r-s} \cdot x^{k-r-s})$ складывается с полиномом $g(x) \cdot x^{k-r-s}$, умноженным на скаляр $Q_{k-r-s} = R_{k-s}^{(s-1)}$, и тем самым вычисляется «остаточный» полином $R^{(s)}(x)$.

Резюмируя все вышесказанного можно записать:

$$R^{(s)}(x) = R^{(s-1)}(x) \oplus (\Psi_{k-r-s} \oplus g(x) \cdot R_{k-s}^{(s-1)}) \cdot x^{k-r-s} \tag{3.7.2}$$

Тогда, остаточный полином, вычисленный на последней итерации $k-r$, и будет искомым полиномом-остатком, иными словами: $R(x) = R^{(k-r)}(x)$.

Таким образом, процедура вычисления полинома-остатка $R(x) = R^{(k-r)}(x)$:

$$\begin{cases} s = 1 \ldots k-r; \quad R^{(0)}(x) = \Psi_{k-1} \cdot x^{k-1} \oplus \ldots \oplus \Psi_{k-r} \cdot x^{k-r} \\ R^{(s)}(x) = R^{(s-1)}(x) \oplus (\Psi_{k-r-s} \oplus g(x) \cdot R_{k-s}^{(s-1)}) \cdot x^{k-r-s} \end{cases} \tag{3.7.3}$$

Особо отметим, что начальный «остаточный» полином $R^{(0)}(x)$ имеет степень $k-1$. Далее на каждом шаге $s = 1 \ldots k-r$ степень полинома «остаточного» полинома понижается, как минимум, за счет того, что старший коэффициент $R_{k-s}^{(s-1)}$ полинома $(R^{(s-1)}(x) \oplus \Psi_{k-r-s} \cdot x^{k-r-s})$, складываясь со старшим коэффициентом $g_r = 1$ полинома $g(x) \cdot x^{k-r-s}$, умноженного на очередной коэффициент $Q_{k-r-s} = R_{k-s}^{(s-1)}$ полинома-частного, обращается в нуль.

Также отметим, что «остаточный» полином как начальный $R^{(0)}(x)$, так и очередной $R^{(s)}(x)$ на любом из шагов $s = 1 \ldots k-r$, фактически состоит не более чем из *r* ненулевых коэффициентов. Поэтому с вычислительной точки зрения выгоднее хранить и обрабатывать не сам «остаточный» полином, а некоторый, так сказать, «смещенный остаточный» полином $\widetilde{R}^{(s)}(x) = \widetilde{R}_{r-1}^{(s)} \cdot x^{r-1} \oplus \ldots \oplus \widetilde{R}_1^{(s)} \cdot x \oplus \widetilde{R}_0^{(s)}$, для хранения и обработки которого достаточно *r* ячеек памяти, и который связан с самим «остаточным» полиномом соотношением:

$$R^{(s)}(x) = x^{k-r-s} \cdot \widetilde{R}^{(s)}(x); \quad s = 0 \ldots k-r \tag{3.8.1}$$

Тогда с учетом вышесказанного начальный «смещенный остаточный» полином имеет вид $\widetilde{R}^{(0)}(x) = R^{(0)}(x) \big/ x^{k-r} = \Psi_{k-1} \cdot x^{r-1} \oplus \ldots \oplus \Psi_{k-r}$, а на последней итерации $s = k-r$, «смещенный остаточный» полином совпадает с самим «остаточным» полиномом $R^{(k-r)}(x) = \widetilde{R}^{(k-r)}(x)$. Теперь выполним преобразование рекуррентного соотношения, используемого для вычисления на шаге $s = 1 \ldots k-r$ «остаточного» полинома $R^{(s)}(x)$, подставив вместо него «смещенный остаточный» полином $\widetilde{R}^{(s)}(x)$:

$$R^{(s)}(x) = R^{(s-1)}(x) \oplus (\Psi_{k-r-s} \oplus g(x) \cdot R_{k-s}^{(s-1)}) \cdot x^{k-r-s} \Rightarrow$$

$$\Rightarrow x^{k-r-s} \cdot \widetilde{R}^{(s)}(x) = x^{k-r-(s-1)} \cdot \widetilde{R}^{(s-1)}(x) \oplus (\Psi_{k-r-s} \oplus g(x) \cdot \widetilde{R}_{r-1}^{(s-1)}) \cdot x^{k-r-s} \Rightarrow$$

$$\Rightarrow \widetilde{R}^{(s)}(x) = x \cdot \widetilde{R}^{(s-1)}(x) \oplus \Psi_{k-r-s} \oplus g(x) \cdot \widetilde{R}_{r-1}^{(s-1)}.$$

Тогда можем окончательно записать процедуру вычисления полинома-остатка $R(x) = \widetilde{R}^{(k-r)}(x)$ с использованием «смещенного остаточного» полинома:

$$\begin{cases} s = 1 \ldots k-r; \quad \widetilde{R}^{(0)}(x) = \Psi_{k-1} \cdot x^{r-1} \oplus \ldots \oplus \Psi_{k-r+1} \cdot x \oplus \Psi_{k-r} \\ \widetilde{R}^{(s)}(x) = x \cdot \widetilde{R}^{(s-1)}(x) \oplus \Psi_{k-r-s} \oplus g(x) \cdot \widetilde{R}_{r-1}^{(s-1)} \end{cases} \quad (3.8.2)$$

Заметим, что в рекуррентном соотношении складываются полиномы $x \cdot \widetilde{R}^{(s-1)}(x)$ и $(\Psi_{k-r-s} \oplus g(x) \cdot \widetilde{R}_{r-1}^{(s-1)})$, коэффициенты при x^r у которых равны $\widetilde{R}_{r-1}^{(s-1)}$, так как $g_r = 1$, и при сложении коэффициенты при x^r всегда в сумме дают нуль. Это позволяет сэкономить на одной ячейке памяти и на одной операции сложения коэффициентов при x^r.

Тогда с учетом всего вышесказанного перепишем процедуру вычисления полинома-остатка $R(x) = \widetilde{R}^{(k-r)}(x)$ в развернутом виде, с детализацией вычисления отдельных коэффициентов «смещенного остаточного» полинома $\widetilde{R}^{(s)}(x)$:

$$\begin{cases} \widetilde{R}_{r-1}^{(0)} = \Psi_{k-1} \quad \ldots \quad \widetilde{R}_0^{(0)} = \Psi_{k-r} \\ \\ s = 1 \ldots k-r \\ \\ \widetilde{R}_j^{(s)} = \begin{cases} \widetilde{R}_{j-1}^{(s-1)} \oplus g_j \cdot \widetilde{R}_{r-1}^{(s-1)}; j = 1 \ldots r-1 \\ \Psi_{k-r-s} \oplus g_0 \cdot \widetilde{R}_{r-1}^{(s-1)}; j = 0 \end{cases} \end{cases} \quad (3.8.3)$$

Следует отметить, что процедура фактически сводится к последовательности операций сдвига $r-$ячейного регистра (в каждой ячейке сдвигового регистра мы имеем m двоичных разрядов), хранящего коэффициенты $\widetilde{R}_{r-1}^{(s-1)} \ldots \widetilde{R}_0^{(s-1)}$ полинома-остатка $\widetilde{R}^{(s-1)}(x)$, полученного на предыдущем шаге, и сложения его с коэффициентами $g_{r-1} \ldots g_0$ полинома-делителя $g(x)$ (без старшего коэффициента $g_r = 1$), умноженного на коэффициент $\widetilde{R}_{r-1}^{(s-1)}$ полинома-остатка $\widetilde{R}^{(s-1)}(x)$. При сдвиге регистра в младшую 0-ую ячейку загружается очередной коэффициент Ψ_{k-r-s} полинома-делимого $\Psi(x)$. Перед началом процедуры, в регистр остатка загружаются коэффициенты $\Psi_{k-1} \ldots \Psi_{k-r}$.

Вычисление остатка полинома $\Psi(x)$ по модулю полинома $g(x)$ схема на практике аппаратно чаще всего реализуется в виде $r-$ячейного сдвигового регистра с линейной обратной связью, причем на начальном этапе (перед непосредственным делением) загрузка коэффициентов $\Psi_{k-1} \ldots \Psi_{k-r}$ осуществляется не параллельным, а последовательным (сдвиговым) способом через младшую 0-ю ячейку сдвигового регистра. Соответственно, цикл работы схемы состоит из двух фаз: фаза последовательной загрузки «начального» остатка $\widetilde{R}^{(0)}(x) = \Psi_{k-1} \cdot x^{r-1} \oplus \ldots \oplus \Psi_{k-r+1} \cdot x \oplus \Psi_{k-r}$, выполняемой в течение r тактов, и фазы итерационного деления, выполняемой в течение $k - r$ тактов. В итоге суммарно требуется k тактов для вычисления остатка. Ниже на рис. 3.1. приведена функциональная схема «вычислителя» остатка полинома $\Psi(x)$ по модулю полинома $g(x)$.

В приведенной схеме, на первой фазе, обратная связь разомкнута (сигнал *Control* = 0) на коммутационной схеме, обозначенной знаком $\triangleright$, и в течение первых *r* тактов, $q = 1 \ldots r$, в регистр загружаются коэффициенты $\Psi_{k-1} \ldots \Psi_{k-r}$, поступающие через вход 0-й ячейки регистра. Поскольку обратная связь разомкнута, то на схемах сложения они складываются с нулем, и поступают в регистр в неизменном виде. Во второй фазе, коммутатор замыкает обратную связь (сигнал *Control* = 1), и в течение $k - r$ тактов, $q = r + 1 \ldots k$, на вход 0-й ячейки регистра поступают коэффициенты $\Psi_{k-r-1} \ldots \Psi_0$, и итерационно вычисляется остаток. Результат будет содержаться в *r* ячейках сдвигового регистра после завершения такта $q = k$.

Clock – вход для тактового сигнала, по которому осуществляется запись информации в ячейки регистра, поступающей на вход ячеек. *Control* – вход управления *m*-разрядной коммутационной схемой ($\triangleright$), которая легко реализуется с помощью *m* двухвходовых логических элементов «И». Знаком $\oplus$ обозначено сложение двух элементов поля Галуа $GF(2^m)$. Сложение легко реализуется с помощью *m* двухвходовых логических элементов XOR. Знаком $\otimes$ обозначено умножение элементов поля Галуа $GF(2^m)$. Заметим, что на практике при аппаратной реализации схем вычисления остатка, коэффициенты $g_{r-1} \ldots g_0$ полинома-делителя обычно являются константами, задаваемыми на этапе проектирования, поэтому в качестве схем умножения можно использовать быстрые и компактные «умножители на константу», которые мы рассматривали выше. Каждый из умножителей выполняет умножение на конкретную фиксированную константу $g_j, j = 0 \ldots r-1$.

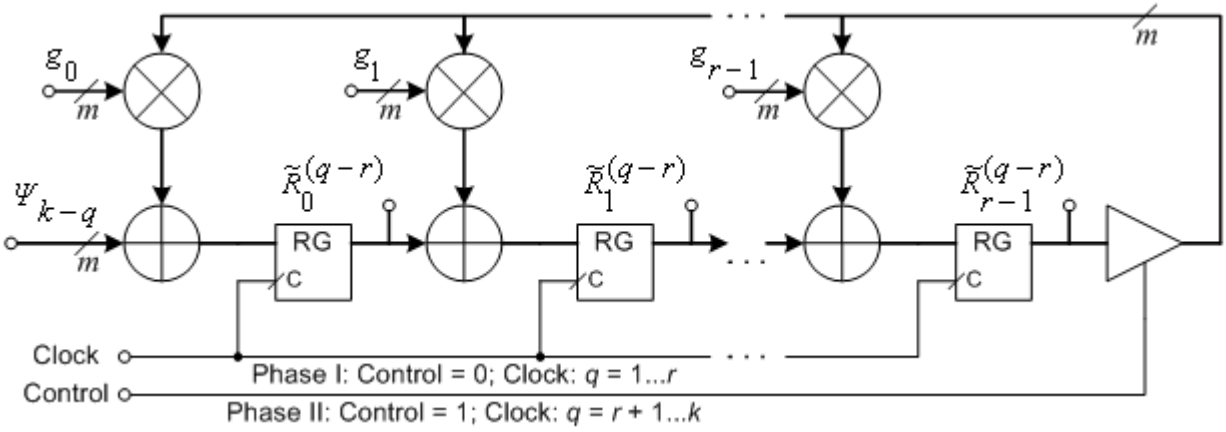

Рис. 3.1. Функциональная схема последовательного вычислителя остатка полинома $\Psi(x)$ по модулю полинома $g(x)$ на базе сдвигового регистра с обратной связью.

Усеченной линейной сверткой полиномов будем называть остаток от произведения этих полиномов по модулю полинома x^r. По сути, это произведение полиномов, в котором «отброшены» члены, степень переменной *x* в которых больше либо равна *r*:

$$\Psi(x) \circ \xi(x) = (\Psi(x) \cdot \xi(x)) \bmod x^r = \sum_{q=0}^{r-1} x^q \cdot \left(\sum_{i=0}^{q} \Psi_i \cdot \xi_{q-i} \right); \quad r \geq 2$$
$$\Psi_i, \xi_j \in GF(2^m); \quad \forall i > \deg(\Psi(x)) = k-1 : \Psi_i = 0; \quad \forall j > \deg(\xi(x)) = l-1 : \xi_j = 0; \tag{3.9}$$

Пример. Найдем усеченную свертку полиномов $\Psi(x) = 49 \cdot x^2 \oplus 50 \cdot x \oplus 51$ и $\xi(x) = 19 \cdot x^2 \oplus 93 \cdot x \oplus 1$, заданных над полем $GF(2^8)$, по модулю полинома x^2. Получаем:

$$\sum_{q=0}^{1} x^q \cdot \left(\sum_{i=0}^{q} \Psi_i \cdot \xi_{q-i} \right) = \Psi_0 \xi_0 \oplus (\Psi_0 \xi_1 \oplus \Psi_1 \xi_0) \cdot x = 51 \cdot 1 \oplus (51 \cdot 93 \oplus 50 \cdot 1) \cdot x = 3 \cdot x \oplus 51.$$

Циклической сверткой полиномов будем называть остаток от произведения этих полиномов по модулю полинома $x^r \oplus 1$.

В общем случае циклическая свертка вычисляется по следующей формуле (эта формула выводится так же, как в случае циклической свертки для многочленов заданных над полем *GF*(*p*), которые мы рассматривали выше):

$$\Psi(x) \circ \xi(x) = (\Psi(x) \cdot \xi(x)) \bmod (x^r \oplus 1) = \sum_{q=0}^{r-1} x^q \cdot \sum_{\substack{(i+j) \bmod r = q \\ i=0\ldots k-1 \quad j=0\ldots l-1}} \left(\Psi_i \cdot \xi_j \right) \tag{3.10.1}$$

$$\Psi_i, \xi_j \in GF(2^m); \quad \deg(\Psi(x)) = k-1; \quad \deg(\xi(x)) = l-1;$$

На практике чаще всего применяется циклическая свертка для полиномов степени не выше $r-1$. В этом случае формула для вычисления циклической свертки может быть упрощена следующим образом (это делается так же, как в случае циклической свертки для многочленов заданных над полем *GF*(*p*), которые мы рассматривали выше):

$$\Psi(x) \circ \xi(x) = (\Psi(x) \cdot \xi(x)) \bmod (x^r \oplus 1) = \sum_{q=0}^{r-1} x^q \cdot \left(\sum_{i=0}^{r-1} \left(\Psi_i \cdot \xi_{(r+q-i) \bmod r} \right) \right) \tag{3.10.2}$$

$$\Psi_i, \xi_j \in GF(2^m); \quad \deg(\Psi(x)) = k-1 \le r-1; \quad \deg(\xi(x)) = l-1 \le r-1;$$

$$\forall i > k-1 : \Psi_i = 0; \quad \forall j > l-1 : \xi_j = 0; \quad r \ge 2;$$

Пример. Найдем циклическую свертку полиномов $\Psi(x) = 49 \cdot x^2 \oplus 50 \cdot x \oplus 51$ и $\xi(x) = 19 \cdot x^2 \oplus 93 \cdot x \oplus 1$, заданных над полем *GF*(2^8), по модулю полинома $x^3 \oplus 1$. Имеем:

$$\sum_{q=0}^{2} x^q \cdot \left(\sum_{i=0}^{2} \left(\Psi_i \cdot \xi_{(3+q-i) \bmod 3} \right) \right) = (\Psi_0\xi_0 \oplus \Psi_1\xi_2 \oplus \Psi_2\xi_1) \oplus (\Psi_0\xi_1 \oplus \Psi_1\xi_0 \oplus \Psi_2\xi_2) \cdot x$$

$$\oplus (\Psi_0\xi_2 \oplus \Psi_1\xi_1 \oplus \Psi_2\xi_0) \cdot x^2 = (51 \cdot 1 \oplus 50 \cdot 19 \oplus 49 \cdot 93) \oplus (51 \cdot 93 \oplus 50 \cdot 1 \oplus 49 \cdot 19) \cdot x \oplus$$

$$\oplus (51 \cdot 19 \oplus 50 \cdot 93 \oplus 49 \cdot 1) \cdot x^2 = 31 \cdot x^2 \oplus 103 \cdot x \oplus 233.$$

Вычисление формальной производной полинома. Достаточно очевидно, что вычисление производной от полинома, заданного в поле Галуа, требует особого подхода, хотя бы потому, что здесь мы, как в случае традиционной алгебры, не можем говорить о бесконечно малых величинах – в поле Галуа нет таких элементов, которые бы мы могли использовать в качестве бесконечно малого. Однако, никто нам не запрещает говорить о некоторой формальной бесконечно малой величине, обозначим ее ε, которая будет нами использоваться исключительно для выполнения математических преобразований для получения формальной производной от заданной функции $y = f(x)$.

Сделаем ряд оговорок относительно величины ε:

$$0 + \varepsilon^k = \varepsilon^k; \quad \varepsilon^k - \varepsilon^k = 0; \quad 0 \cdot \varepsilon^k = 0; \quad 0 / \varepsilon^k = 0;$$

$$1 \cdot \varepsilon^k = \varepsilon^k; \quad \varepsilon^k / \varepsilon^k = 1; \quad \varepsilon^k \cdot \varepsilon = \varepsilon^{k+1}; \quad \varepsilon^{k+1} / \varepsilon = \varepsilon^k;$$

$$\lim_{\varepsilon \to 0} \varepsilon^k = 0, \quad k \ge 1$$

Кроме того, ради общности рассуждений мы рассмотрим формальную производную полинома, заданного над полем Галуа *GF*(p^m), а затем рассмотрим частный случай производной над полем Галуа *GF*(2^m).

Тогда, мы можем дать определение формальной производной функции $f(x)$:

$$\frac{d}{dx} f(x) = \lim_{\varepsilon \to 0} \frac{f(x+\varepsilon) - f(x)}{\varepsilon}$$

Особо отметим, что не следует «в лоб» пытаться искать численное значение предела при конкретных значениях аргумента *x*. Формула задана исключительно для аналитических преобразований, подразумевающих исключение величины ε из итогового аналитического выражения для производной функции. Заметим, что так же, как и в традиционной алгебре, здесь справедливы следующие свойства формальной производной функции:

$$\frac{d(\beta\cdot f(x))}{dx}=\beta\cdot\frac{df(x)}{dx};\quad \frac{d(f(x)\pm g(x))}{dx}=\frac{df(x)}{dx}\pm\frac{dg(x)}{dx};\quad \frac{d(f(x)\cdot g(x))}{dx}=\frac{df(x)}{dx}g(x)+\frac{dg(x)}{dx}f(x);$$

Теперь же выведем формальные производные для простейших функций:

- Константная функция $f(x)=\beta\Rightarrow \frac{d}{dx}f(x)=\lim\limits_{\varepsilon\to 0}\frac{\beta-\beta}{\varepsilon}=\lim\limits_{\varepsilon\to 0}\frac{0}{\varepsilon}=0.$
- Функция $f(x)=\beta\cdot x\Rightarrow \frac{d}{dx}f(x)=\beta\cdot\lim\limits_{\varepsilon\to 0}\frac{x+\varepsilon-x}{\varepsilon}=\beta\cdot\lim\limits_{\varepsilon\to 0}\frac{\varepsilon}{\varepsilon}=\beta\cdot 1=\beta.$

Однако, прежде чем продолжить выводить производные степенных функций более высокого порядка, рассмотрим подробнее выражение $(x+\varepsilon)^k, k\ge 1$. В традиционной алгебре для возведения в степень *k* выражения $x+\varepsilon$ существует простая общая формула: $(x+\varepsilon)^k=\sum\limits_{i=0}^{k}C_k^i\cdot x^i\cdot\varepsilon^{k-i}$. Биномиальный коэффициент $C_k^i=\frac{k!}{i!(k-i)!}$ представляет собою количество одинаковых (повторяющихся) слагаемых $x^i\cdot\varepsilon^{k-i}$ (для каждого $i=0\ldots k$), образуемых и затем складывающихся при возведении в степень *k* выражения $x+\varepsilon$. Однако, здесь следует учесть арифметическое свойство полей Галуа *GF*(p^m): $\underbrace{\underbrace{GF(p^m)}_{a+\ldots+a}}_{n\ слагаемых}=\underbrace{GF(p^m)}_{\lambda\cdot a}$, где $\lambda=\underbrace{<R,\{+,\cdot\}>}_{n\bmod p}$. В нашем случае, $a=x^i\cdot\varepsilon^{k-i}$ и $n=C_k^i$, и тогда:

$$\underbrace{\underbrace{GF(p^m)}_{x^i\cdot\varepsilon^{k-i}+\ldots+x^i\cdot\varepsilon^{k-i}}}_{n\ слагаемых}=\overbrace{\left(C_k^i\bmod p\right)\cdot x^i\cdot\varepsilon^{k-i}}^{GF(p^m)}$$

Тогда, учитывая все сказанное, можно вывести общую формулу для $(x+\varepsilon)^k, k\ge 1$: $(x+\varepsilon)^k=\sum\limits_{i=0}^{k}x^i\cdot\varepsilon^{k-i}\cdot\left(C_k^i\bmod p\right)$.

Теперь, наконец, вычислим производную от функции $f(x)=\beta\cdot x^k, k\ge 1\Rightarrow$

$$\frac{df(x)}{dx}=\beta\cdot\lim_{\varepsilon\to 0}\frac{(x+\varepsilon)^k-x^k}{\varepsilon}=\beta\cdot\lim_{\varepsilon\to 0}\frac{\left(\sum\limits_{i=0}^{k}x^i\cdot\varepsilon^{k-i}\cdot\left(C_k^i\bmod p\right)\right)-x^k}{\varepsilon}.$$

Выделим из суммирования слагаемое x^k при $i=k$, и учтем, что $C_k^k=1$ и $\varepsilon^0=1$. Тогда в итоге имеем:

$$\frac{df(x)}{dx}=\beta\cdot\lim_{\varepsilon\to 0}\frac{\left(x^k+\sum\limits_{i=0}^{k-1}x^i\cdot\varepsilon^{k-i}\cdot\left(C_k^i\bmod p\right)\right)-x^k}{\varepsilon}=\beta\cdot\lim_{\varepsilon\to 0}\frac{\left(\sum\limits_{i=0}^{k-1}x^i\cdot\varepsilon^{k-i}\cdot\left(C_k^i\bmod p\right)\right)}{\varepsilon}.$$

Заметим, что в числителе среди оставшихся слагаемых суммирования, только при $i=k-1$, слагаемое $x^{k-1}\cdot\varepsilon$ содержит ε в первой степени, которое может сократиться с ε в знаменателе, а при $i\le k-2$, слагаемые содержат ε в степени большей, чем 1, и оно не уничтожится вместе с ε в знаменателе, и после взятия предела слагаемые превратятся в нуль, поскольку будут содержать ε в ненулевой степени.

$$\frac{df(x)}{dx} = \beta \cdot \lim_{\varepsilon \to 0} \frac{x^{k-1} \cdot \varepsilon \cdot \left(C_k^{k-1} \bmod p \right) + \left(\sum_{i=0}^{k-2} x^i \cdot \varepsilon^{k-i} \cdot \left(C_k^i \bmod p \right) \right)}{\varepsilon} = \beta \cdot (k \bmod p) \cdot x^{k-1}.$$

Таким образом, окончательно: $\frac{d}{dx}\left(\beta \cdot x^k \right) = \beta \cdot (k \bmod p) \cdot x^{k-1}$ при $k \geq 1$.

Теперь мы можем, вывести формулу для вычисления формальной производной от полинома $\Psi(x) = \Psi_{k-1} \cdot x^{k-1} + \ldots + \Psi_1 \cdot x + \Psi_0$, заданного над полем Галуа $GF(p^m)$:

$$\frac{d}{dx}\Psi(x) = \frac{d}{dx}\left(\sum_{i=0}^{k-1} \Psi_i \cdot x^i \right) = \frac{d}{dx}\left(\Psi_0 \right) + \sum_{i=1}^{k-1} \Psi_i \cdot \frac{d}{dx}\left(x^i \right) = \sum_{i=1}^{k-1} \left(\left(\underbrace{\Psi_i \cdot \overbrace{(i \bmod p)}^{< R, \{+,\cdot\} >}}_{GF(p^m)} \right) \cdot x^{i-1} \right) \quad (3.11.1)$$

$$\Psi_i \in GF(p^m); \quad i = 0 \ldots k-1;$$

Особо отметим, что выражение $\Psi_i \cdot (i \bmod p)$ вычисляется как произведение элементов Ψ_i и λ в поле $GF(p^m)$, где элемент λ численно равен $i \bmod p$ в традиционной арифметике действительных чисел, но интерпретируется именно как элемент поля $GF(p^m)$.

В частном случае, если мы вычисляем формальную производную от полинома $\Psi(x) = \Psi_{k-1} \cdot x^{k-1} \oplus \ldots \oplus \Psi_1 \cdot x \oplus \Psi_0$, заданного над полем Галуа $GF(2^m)$, характеристики $p = 2$, мы получаем следующую формулу формальной производной:

$$\frac{d}{dx}\Psi(x) = \sum_{i=1}^{k-1} \left(\left(\underbrace{\Psi_i \cdot \overbrace{(i \bmod 2)}^{< R, \{+,\cdot\} >}}_{GF(2^m)} \right) \cdot x^{i-1} \right) \quad (3.11.2)$$

$$\Psi_i \in GF(2^m); \quad i = 0 \ldots k-1;$$

Заметим, что выражение $(i \bmod 2)$ приводит к тому, что в полях $GF(2^m)$, коэффициенты Ψ_i с четными индексами, стоящие при переменной x^{i-1} у полинома-производной, умножаются на нуль, и, соответственно, в полиноме-производной всегда отсутствуют слагаемые с нечетными степенями переменной x.

Пример. Формальная производная $\Psi(x) = 100 \cdot x^4 \oplus 218 \cdot x^3 \oplus 31 \cdot x^2 \oplus 3 \cdot x \oplus 51$, заданного над полем $GF(2^8)$: $\frac{d\Psi(x)}{dx} = \sum_{i=1}^{4} \left(\Psi_i \cdot (i \bmod 2) \cdot x^{i-1} \right) = \Psi_3 \cdot x^2 \oplus \Psi_1 = 218 \cdot x^2 \oplus 3$.

Вычисление значения полинома при заданном аргументе. Помимо операций с полиномами в технологии кодирования информации часто требуется вычисление значения полинома при заданном аргументе, и здесь имеется свой небольшой нюанс.

При разработке программных или аппаратных реализаций, вычисление значения полинома $\Psi(x)$ путем прямой подстановки заданного значения аргумента $x*$ в формулу $\Psi(x) = \Psi_{k-1} \cdot x^{k-1} \oplus \ldots \oplus \Psi_1 \cdot x \oplus \Psi_0$ не очень разумно, поскольку требует, по меньшей мере, $k-1$ операций сложения и $k-1$ операций умножения по правилам алгебры для поля Галуа, а также $k-2$ операций возведения заданного элемента поля Галуа в соответствующую степень по формуле: $(x*)^i = \alpha^{\left(i \cdot \log_\alpha (x*) \right) \bmod (2^m - 1)}$, $x* \neq 0$.

Заметим, что каждая операция возведения в степень требует извлечения логарифма для заданного $x*$ из поля логарифмов, умножение на степень i, вычисления остатка по модулю 2^m-1, и, наконец, извлечения элемента поля Галуа, соответствующего степени $\left(i\cdot\log_{\alpha}(x*)\right)\bmod(2^m-1)$ примитивного элемента α. Поэтому вместо столь излишне затратной с точки зрения вычислительной сложности процедуры, рекомендуется использовать хорошо известную схему Горнера для вычисления значения полинома $\Psi(x)$ при заданном аргументе $x*$.

Идея предельно проста, полином $\Psi(x)$ с использованием простых тождественных преобразований, можно всегда привести к виду:

$$\Psi(x)=\left(\ldots\left(\Psi_{k-1}\cdot x\oplus\Psi_{k-2}\right)\cdot x\oplus\ldots\oplus\Psi_1\right)\cdot x\oplus\Psi_0 \qquad (3.12.1)$$

После этого мы можем применить достаточно простую рекуррентную схему, предложенную Горнером, для итерационного вычисления значения полинома (в качестве номера итерации будем использовать символ *s*):

$$\begin{cases}\Psi^{(s)}(x*)=(x*)\cdot\left(\Psi^{(s-1)}(x*)\right)\oplus\Psi_{k-s}\\ \Psi^{(0)}(x*)=0;\quad s=1\ldots k;\quad k\geq 0\end{cases} \qquad (3.12.2)$$

После k итераций, результат вычислений на последней итерации, $\Psi^{(k)}(x*)$, очевидно, будет являться искомым значением полинома $\Psi(x)$ при заданном аргументе $x*$. Заметим, что если же $k=0$, то в качестве результата возвращается нуль, а если $k=1$, то в качестве результата возвращается Ψ_0.

С точки зрения вычислительной сложности такая схема вычислений гораздо более выгодна, поскольку здесь уже для вычисления значения полинома потребуется только *k* операций сложения и *k* операций умножения по правилам алгебры для поля Галуа, а необходимость трудоемкого возведения $x*$ в степень i вообще отпадает.

Пример. Вычислим значение полинома $\Psi(x)=x^4\oplus 30\cdot x^3\oplus 216\cdot x^2\oplus 231\cdot x\oplus 116$ заданного над $GF(2^8)$ при заданном аргументе *x* = 77. Подставляя *x* = 77 в полином, имеем: $77^4\oplus 30\cdot 77^3\oplus 216\cdot 77^2\oplus 231\cdot 77\oplus 116=94\oplus 30\cdot 150\oplus 216\cdot 156\oplus 231\cdot 77\oplus 116=160$. Тот же самый результат можно получить, предварительно приведя полином к вышеупомянутому виду $(((x\oplus 30)\cdot x\oplus 216)\cdot x\oplus 231)\cdot x\oplus 116$, а потом уж, подставив $x=77$, достаточно быстро выполнить вычисления, не прибегая к трудоемкой операции возведения в степень: $(((77\oplus 30)\cdot 77\oplus 216)\cdot 77\oplus 231)\cdot 77\oplus 116=160$.

Отметим, что вычисление значения полинома при заданном аргументе по схеме Горнера легко реализуется аппаратно с помощью сумматора, умножителя элементов поля Галуа $GF(2^m)$ и одного *m*-битного регистра. Ниже на рис 3.2 приведена функциональная схема последовательного вычислителя $\Psi(x*)$. Перед началом работы вычислителя регистр сбрасывается в нуль по сигналу, поступающему на вход *Reset*. Далее, с приходом первого тактового сигнала, *s* = 1, на вход *Clock* в регистр через сумматор загружается старший коэффициент полинома $\Psi(x)$ в чистом виде (от умножителя поступает нуль), иными словами $\Psi^{(1)}(x*)=\Psi_{k-1}$. Далее с каждым следующим тактовым сигналом $s=2\ldots k$ на вход сумматора поступают остальные коэффициенты Ψ_{k-s}, которые складываются с содержимым регистра, умноженным на $x*$, и результат заносится в регистр. После поступления последнего тактового сигнала $s=k$ в регистре получаем результат $\Psi(x*)$.

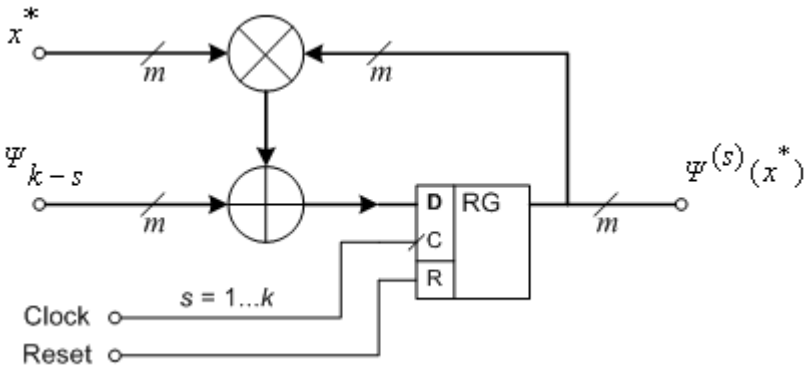

Рис. 3.2. Функциональная схема последовательного вычислителя значения полинома $\Psi(x)$ при заданном аргументе $x*$.

Вычисление значения формальной производной полинома при заданном аргументе. В практике помехоустойчивого кодирования также иногда требуется значение формальной производной полинома при заданном аргументе, при этом сам по себе полином-производная нигде не требуется. В такой ситуации схема Горнера для вычисления значения полинома $\Psi(x)$ при заданном аргументе $x*$ также может быть адаптирована для вычисления значения формальной производной этого полинома при заданном аргументе. Ранее мы выяснили, что формальная производная полинома $\Psi(x)$ над полем $GF(2^m)$ определяется как:

$$\frac{d\Psi(x)}{dx} = \sum_{i=1}^{k-1}\left(\Psi_i \cdot (i \bmod 2)\cdot x^{i-1}\right) = \Psi_1 \oplus \Psi_3 \cdot x^2 \oplus \ldots \oplus \begin{cases} \Psi_{k-1}\cdot x^{k-2}, (k-1)\bmod 2 = 1 \\ \Psi_{k-2}\cdot x^{k-3}, (k-1)\bmod 2 = 0 \end{cases}.$$

Тогда формальную производную можно переписать в следующем виде:

$$\left(\ldots\left(\Psi_{k-1}\cdot((k-1)\bmod 2)\cdot x \oplus \Psi_{k-2}\cdot((k-2)\bmod 2)\right)\cdot x \oplus \ldots \oplus \Psi_2 \cdot 0\right)\cdot x \oplus \Psi_1 \cdot 1 \quad (3.13.1)$$

После этого мы можем применить адаптированную схему Горнера для итерационного вычисления значения формальной производной полинома $\Psi(x)$ при заданном аргументе $x*$:

$$\begin{cases} d\Psi^{(s)}(x*)\Big/dx = (x*)\cdot\left(d\Psi^{(s-1)}(x*)\Big/dx\right) \oplus \Psi_{k-s}\cdot((k-s)\bmod 2) \\ d\Psi^{(0)}(x*)\Big/dx = 0; \quad s = 1\ldots k-1; \quad k \ge 0 \end{cases} \quad (3.13.2)$$

После $k-1$ итераций, результат вычислений на последней итерации $d\Psi^{(k-1)}(x*)\Big/dx$ и будет являться искомым значением формальной производной $d\Psi(x)/dx$ при заданном аргументе $x*$. Заметим, что если $k=0$ или $k=1$, то в качестве результата возвращается нуль, а если $k=2$ или $k=3$, то в качестве результата возвращается Ψ_1.

Пример. Вычислим значение формальной производной полинома $\Psi(x) = 222\cdot x^5 \oplus 29\cdot x^4 \oplus 34\cdot x^3 \oplus 183\cdot x^2 \oplus 232\cdot x \oplus 1$ заданного над полем $GF(2^8)$ при $x = 64$. С одной стороны, формальная производная полинома имеет следующий вид: $d\Psi(x)/dx = 222\cdot x^4 \oplus 34\cdot x^2 \oplus 232$. Подставляя в нее $x = 64$, получаем значение формальной производной $222\cdot 64^4 \oplus 34\cdot 64^2 \oplus 232 = 70$. С другой стороны, можно получить тот же результат, не прибегая к расчету полинома-производной $d\Psi(x)/dx$, используя лишь только исходный полином $\Psi(x)$ и применив к нему адаптированную схему Горнера: $((((222\cdot 1)\cdot 64 \oplus 29\cdot 0)\cdot 64 \oplus 34\cdot 1)\cdot 64 \oplus 183\cdot 0)\cdot 64 \oplus 232\cdot 1 = 70$.

Вычисление значения формальной производной $d\Psi(x)/dx$ при заданном аргументе $x*$ также несложно реализуется аппаратно. Ниже на рис. 3.3 приведена функциональная схема последовательного вычислителя $d\Psi(x*)/dx$. Особенность схемы – в ней присутствует счётный D-триггер, который перед началом вычислений сбрасывается, а затем с приходом каждого тактового сигнала меняет состояние на противоположное. Выход триггера подключен к логическому элементу XOR, ко второму входу которого подается 1, если степень полинома $\Psi(x)$ является нечетной, или 0, если степень является четной. В итоге на управляющий вход коммутационной схемы $\triangleright$, поступает последовательность 101…101, при нечетной степени полинома $\Psi(x)$, или 01…101, при четной степени полинома. Таким образом, коммутационная схема всегда блокирует коэффициенты с четными индексами, коэффициенты с нечетными индексами, наоборот, всегда пропускает.

Изначально схема сбрасывается сигналом, подаваемым на вход *Reset*, тем самым и регистр и триггер сбрасываются в нулевое значение. При поступлении очередного тактового сигнала $s = 1…k – 1$ на один вход сумматора поступает результат умножения содержимого регистра на аргумент $x*$, а на второй вход сумматора поступает очередной коэффициент Ψ_{k-s}, если $k – s$ является нечетным, и он складывается с результатом умножения, и сумма записывается в регистр, если же $k – s$ является четным, то на второй вход сумматора поступает нуль, и в итоге в регистр записывается результат умножения без изменений.

В итоге после $k – 1$ тактов на выходе регистра мы получаем искомое $d\Psi(x*)/dx$.

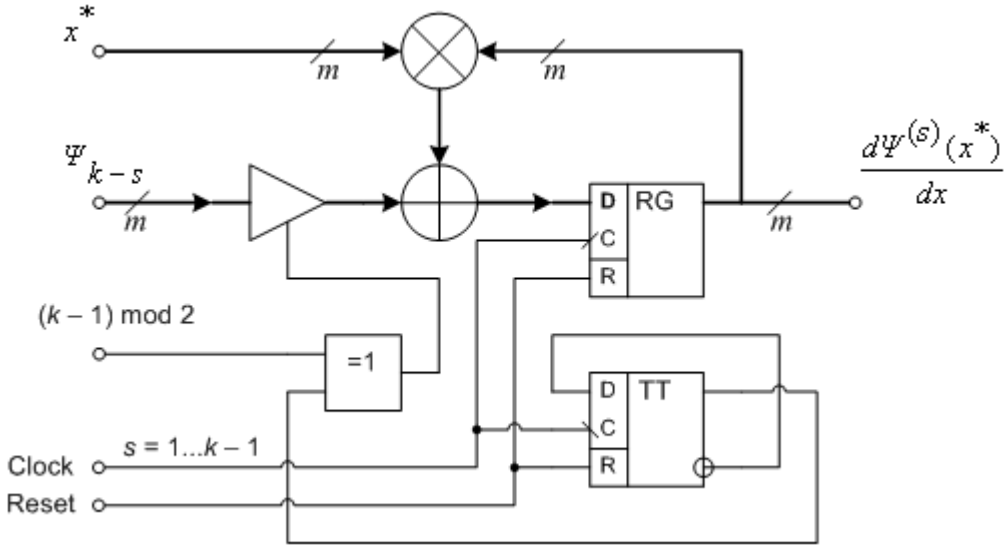

Рис. 3.3. Функциональная схема последовательного вычислителя значения формальной производной $d\Psi(x)/dx$ при заданном аргументе $x*$.

Вычисление значения полинома-свертки при заданном аргументе. В практике помехоустойчивого кодирования также иногда требуется значение полинома, являющегося усеченной сверткой двух заданных полиномов, при заданном аргументе, при этом сам по себе полином-свертка нигде не требуется. В такой ситуации схема Горнера для вычисления значения полинома $\Psi(x)$ при заданном аргументе $x*$ также может быть адаптирована для вычисления значения полинома-свертки при заданном аргументе.

Ранее мы определили, что усеченная свертка $\Omega(x)$ двух полиномов $\Psi(x)$ и $\xi(x)$, а также заданного r, при условии, что степени полиномов не ниже $r – 1$, определяется как:

$$\Omega(x) = \left(\Psi(x)\cdot\xi(x)\right)\bmod x^r = \sum_{q=0}^{r-1} x^q \cdot \left(\sum_{i=0}^{q} \Psi_i \cdot \xi_{q-i}\right)$$

$$\Psi_i, \xi_j \in GF(2^m); \quad i = 0\ldots k-1; \quad j = 0\ldots l-1; \quad k \geq r; \quad l \geq r$$

Нетрудно заметить, что вычисление значения полинома-свертки при заданном аргументе по вышеприведенной формуле в общей сложности требует $r\cdot(r+1)/2 \sim r^2/2$ итераций для вычисления двойной суммы. Выполним ряд преобразований формулы:

$$\sum_{q=0}^{r-1} x^q \cdot \left(\sum_{i=0}^{q} \Psi_i \cdot \xi_{q-i} \right) = \Psi_0\xi_0 \oplus (\Psi_0\xi_1 \oplus \Psi_1\xi_0)\cdot x \oplus \ldots \oplus (\Psi_0\xi_{r-1} \oplus \ldots \oplus \Psi_{r-1}\xi_0)\cdot x^{r-1}$$

$$= \Psi_0(\xi_0 \oplus \xi_1 x \oplus \ldots \oplus \xi_{r-1}x^{r-1}) \oplus \Psi_1 x\cdot(\xi_0 \oplus \xi_1 x \oplus \ldots \oplus \xi_{r-2}x^{r-2}) \oplus \ldots \oplus \Psi_{r-1}x^{r-1}\cdot\xi_0$$

$$= \sum_{i=0}^{r-1} \Psi_i \cdot x^i \cdot \left(\sum_{j=0}^{r-1-i} \xi_j \cdot x^j \right).$$ Преобразуем формулу для использования схемы Горнера:

$$\Omega(x) = \left(\ldots \left(\Psi_{r-1}\left(\xi_0\right)\cdot x \oplus \Psi_{r-2}\left(\xi_0 \oplus \xi_1 x\right)\right)\cdot x \oplus \ldots \oplus \Psi_1\left(\sum_{j=0}^{r-2}\xi_j x^j\right)\right)\cdot x \oplus \Psi_0\left(\sum_{j=0}^{r-1}\xi_j x^j\right)$$

Тогда, в первом приближении имеем следующую итерационную схему вычисления значения полинома-свертки $\Omega(x)$ при заданном аргументе x^*:

$$\begin{cases} \Omega^{(s)}(x^*) = (x^*)\cdot\left(\Omega^{(s-1)}(x^*)\right) \oplus \Psi_{r-s}\cdot\left(\sum_{j=0}^{s-1}\xi_j\cdot(x^*)^j\right) \\ \Omega^{(0)}(x^*) = 0; \quad s = 1\ldots r; \quad r \ge 0 \end{cases} \qquad (3.14.1)$$

Заметим, что хотя и итерационная процедура вычисляет $\Omega(x^*)$ за r итераций, но на каждой итерации, требуется вычисление суммы $\sum_{j=0}^{s-1}\xi_j\cdot(x^*)^j$, количество слагаемых в которой с каждой итерацией $s = 1\ldots r$ только растет. Однако заметим, что количество слагаемых в сумме в точности совпадает с номером итерации, и можно не заново пересчитывать сумму на каждой итерации s, а лишь добавлять очередное слагаемое к сумме, вычисленной на предыдущей итерации $s-1$. Обозначив $\xi^{(s)}(x^*) = \sum_{j=0}^{s-1}\xi_j\cdot(x^*)^j$, имеем рекуррентное соотношение: $\xi^{(s)}(x^*) = \xi^{(s-1)}(x^*) \oplus \xi_{s-1}\cdot(x^*)^{s-1}$, причем $\xi^{(0)}(x^*) = 0$. Очевидно, можно совместить обе итерационные схемы, на каждой итерации вычисляя сначала очередную $\xi^{(s)}(x^*)$, используя $\xi^{(s-1)}(x^*)$, а затем и $\Omega^{(s)}(x^*)$, используя $\Omega^{(s-1)}(x^*)$ и $\xi^{(s)}(x^*)$. Наконец, отметим, что для вычисления $(x^*)^s$ также необязательно на каждой итерации возводить аргумент x^* в степень s, можно использовать соотношение $(x^*)^s = (x^*)^{s-1}\cdot x^*$, причем $(x^*)^0 = 1$. Тогда, имеем «оптимизированную» итерационную схему вычисления значения полинома-свертки $\Omega(x)$ при заданном аргументе x^*:

$$\begin{cases} \xi^{(s)}(x^*) = \xi^{(s-1)}(x^*) \oplus \xi_{s-1}\cdot(x^*)^{s-1} \\ \Omega^{(s)}(x^*) = (x^*)\cdot\left(\Omega^{(s-1)}(x^*)\right) \oplus \Psi_{r-s}\cdot\left(\xi^{(s)}(x^*)\right) \\ (x^*)^s = (x^*)^{s-1}\cdot x^* \\ \Omega^{(0)}(x^*) = 0; \quad \xi^{(0)}(x^*) = 0; \quad (x^*)^0 = 1; \quad s = 1\ldots r; \quad r \ge 0 \end{cases} \qquad (3.14.2)$$

Полученная рекуррентная схема вычисления значения полинома-свертки $\Omega(x)$ при заданном аргументе $x*$ нетрудно реализовать аппаратно при помощи трех m-битных регистров, а также двух сумматоров и четырех умножителей элементов поля $GF(2^m)$. Ниже на рис. 3.4 представлена функциональная схема последовательная вычислителя значения полинома-свертки $\Omega(x)$ при заданном аргументе $x*$.

Регистр $RG1$ используется для хранения вычисленного на последней итерации степени аргумента $(x*)^{s-1}$, регистр $RG2$, соответственно, $\xi^{(s-1)}(x*)$, наконец, в регистре $RG3$ хранится $\Omega^{(s-1)}(x*)$. Перед началом работы по сигналу сброса, поступающего на вход *Reset*, регистр $RG1$ устанавливается в значение (0…01), соответствующее элементу «1» поля $GF(2^m)$, а регистры $RG2$ и $RG3$ сбрасываются в нулевое значение. Далее на вход *Clock* поступают тактовые сигналы в течение итераций $s = 1…r$. Особо отметим, что вход синхронизации (C) регистра $RG2$ подключен к *Clock* напрямую, и, соответственно, регистр принимает информацию со своего информационного входа (D) по фронту тактового сигнала. Что же касается регистров $RG1$ и $RG2$ то они принимают информации по спаду тактового сигнала, так как и входы синхронизации подключены к *Clock* через инвертор. Такая «двухтактная» синхронизации регистров позволяет на каждом такте $s=1…r$ сначала вычислять и записывать в регистр $RG2$ значение $\xi^{(s)}(x*)=\xi^{(s-1)}(x*)\oplus\xi_{s-1}\cdot(x*)^{s-1}$, и только потом вычислять и записывать $\Omega^{(s)}(x*)=(x*)\cdot\left(\Omega^{(s-1)}(x*)\right)\oplus\Psi_{r-s}\cdot\left(\xi^{(s)}(x*)\right)$ в регистр $RG1$, и одновременно с этим также вычислять и записывать очередную степень аргумента $(x*)^s=(x*)^{s-1}\cdot x*$ в регистр $RG3$, и таким образом, соблюдать правильную «очередность» вычислений на каждом такте. В результате за r тактов схема вычисляет искомое значение полинома-свертки $\Omega(x)=(\Psi(x)\cdot\xi(x))\bmod(x^r)$ при аргументе $x*$.

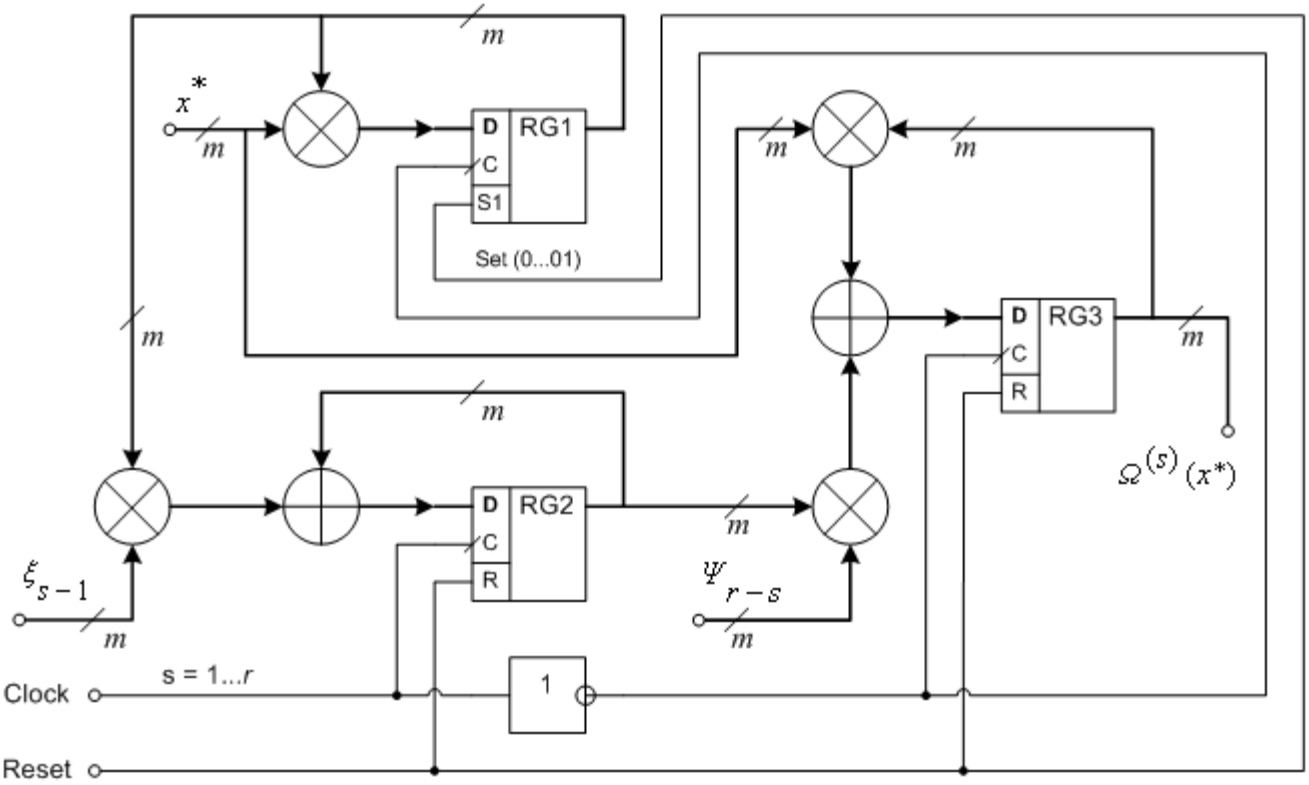

Рис. 3.4. Функциональная схема последовательного вычислителя значения полинома-свертки $\Omega(x)=(\Psi(x)\cdot\xi(x))\bmod(x^r)$ при заданном аргументе $x*$.

Контрольные вопросы

1. Найдите произведение полиномов $L(x) = 148 \cdot x^4 \oplus 68 \cdot x^3 \oplus 116 \cdot x^2 \oplus 187 \cdot x \oplus 1$ и $S(x) = 192 \cdot x^7 \oplus 109 \cdot x^6 \oplus 129 \cdot x^5 \oplus 51 \cdot x^4 \oplus 140 \cdot x^3 \oplus 135 \cdot x^2 \oplus 23 \cdot x \oplus 59$, заданных над полем Галуа $GF(2^8)$, заданного с помощью неприводимого многочлена $p(x) = x^8 + x^4 + x^3 + x^2 + 1$.
2. Найдите остаток полинома $L(x) = 148 \cdot x^4 \oplus 68 \cdot x^3 \oplus 116 \cdot x^2 \oplus 187 \cdot x \oplus 1$ по модулю полинома $g(x) = x^2 \oplus 6 \cdot x \oplus 8$, заданного над полем Галуа $GF(2^8)$, заданного с помощью неприводимого многочлена $p(x) = x^8 + x^4 + x^3 + x^2 + 1$.
3. Найдите усеченную линейную свертку вышеприведенных полиномов $L(x)$ и $S(x)$ по модулю полинома x^8, заданных над полем Галуа $GF(2^8)$, заданного с помощью неприводимого многочлена $p(x) = x^8 + x^4 + x^3 + x^2 + 1$.
4. Найдите циклическую свертку вышеприведенных полиномов $L(x)$ и $S(x)$ по модулю полинома $x^8 \oplus 1$, заданных над полем Галуа $GF(2^8)$, заданного с помощью неприводимого многочлена $p(x) = x^8 + x^4 + x^3 + x^2 + 1$.
5. Найдите формальную производную для вышеприведенного полинома $S(x)$, заданного над полем Галуа $GF(2^8)$, заданного с помощью неприводимого многочлена $p(x) = x^8 + x^4 + x^3 + x^2 + 1$.
6. Вычислите значение вышеприведенного полинома $L(x)$ и формальной производной полинома $L(x)$, заданного над полем Галуа $GF(2^8)$, заданного с помощью неприводимого многочлена $p(x) = x^8 + x^4 + x^3 + x^2 + 1$, при заданном аргументе 77.
7. Вычислите значение полинома-свертки вышеприведенных полиномов $L(x)$ и $S(x)$ по модулю x^8, заданных над полем Галуа $GF(2^8)$, заданного с помощью неприводимого многочлена $p(x) = x^8 + x^4 + x^3 + x^2 + 1$, при заданном аргументе 77.

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

Мы не зря столь подробно рассматривали конечные поля Галуа $GF(2^m)$, и правила арифметики в них, а также полиномы, заданные над полями Галуа $GF(2^m)$, поскольку технология кодирования информации в современных цифровых системах опирается именно на них. Мы будем рассматривать кодирование информации конкретно на примере полей $GF(2^8)$, образованных при помощи заданного неприводимого многочлена $p(x)$ и с заданным примитивным элементом α. На практике чаще всего используется неприводимый многочлен $p(x) = x^8 \oplus x^4 \oplus x^3 \oplus x^2 \oplus 1$ и примитивный элемент $\alpha = 2$.

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:

$$M(x) = M_{k-1} \cdot x^{k-1} \oplus \ldots \oplus M_1 \cdot x \oplus M_0 = \sum_{i=0}^{k-1} M_i \cdot x^i$$
$$M_i \in GF(2^8), i = 0 \ldots k-1 \qquad (4.1)$$

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 интерпретируются, как коэффициенты некоторого полинома, и являются элементами поля Галуа $GF(2^8)$.

Важным понятием в теории кодирования [6-14] является, **кодовое расстояние** (также известное как расстояние по метрике Хэмминга). Упрощенно, кодовое расстояние – это число байтов, которое у них отличаются значениями в соответствующих позициях. Например, кодовое расстояние между 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 M и N одинаковой длины k, которые могут быть представлены соответствующими полиномами $M(x) = M_{k-1} \cdot x^{k-1} \oplus \ldots \oplus M_1 \cdot x \oplus M_0$ и $N(x) = N_{k-1} \cdot x^{k-1} \oplus \ldots \oplus N_1 \cdot x \oplus N_0$. Тогда кодовое расстояние $D(M,N)$, согласно метрике Хэмминга, между этими сообщениями определяется как:

$$D(M,N) = \sum_{j=0}^{k-1} \sigma_j; \quad \sigma_j = \begin{cases} 0, M_j \oplus N_j = 0 \\ 1, M_j \oplus N_j \neq 0 \end{cases} \qquad (4.2)$$

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 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.

$$d = \min_{\forall M,N \in \Gamma^k} \{D(M,N)\} = 1 \qquad (4.3.1)$$

$$\Gamma = GF(2^8)$$

Что еще более важно – любое сообщение, принадлежащее k-мерному пространству сообщений, является, так сказать, «легальным», «допустимым». 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. Очевидно, что общее количество всевозможных информационных сообщений длины k равно 256^k:

$$|\Gamma^k| = 256^k \qquad (4.3.2)$$

Чтобы появилась возможность обнаружения, и, тем более, исправления искажений, согласно теории кодирования требуется расширить пространство за счет введения, так называемых, **контрольных байтов** (избыточных байтов), и в итоге получится n-мерное пространство кадров. Под термином **кадр** будем понимать последовательность, состоящую из k байтов информационного сообщения и r контрольных байтов, причем, $n = k + r$.

Однако недостаточно просто ввести контрольные байты, необходимо также пространство кадров структурировать и выделить подмножество «допустимых» кадров $\Re$ и подмножество «недопустимых» кадров $\Im$. Только в этом случае у нас будет возможность при приеме или считывании кадра, выявить «допустимость» кадра. 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 расширенное пространство, в котором все кадры «допустимы».

Как один из вариантов структурирования пространства кадров, предлагаемых в теории кодирования – это такое n-мерное пространство кадров, состоящих из k информационных байтов и r контрольных байтов, в котором минимальное кодовое расстояние между любыми двумя «допустимыми» кадрами X и Y равно $d = r + 1$:

$$d = \min_{\forall X,Y \in \Re \subset \Gamma^n} \{D(X,Y)\} = r + 1 \qquad (4.4.1)$$

$$\Gamma = GF(2^8)$$

В теории кодирования [6-14] такое структурированное пространство называют пространством «максимально разнесенных кодов», а то, что мы ради наглядности назвали **«допустимым кадром»**, в литературе обычно это называют **«кодовым словом»**.

Обратимся к наглядной геометрической аналогии, представленной ниже на рис. 4.1. Однако, мы здесь должны сделать особую оговорку, что недопустимо проводить прямую аналогию между «евклидовым» расстоянием, с которым имеет дело обычная геометрия, и «хэмминговым» расстоянием, с которым мы имеем дело в n-мерном пространстве кадров. Дело еще осложняется тем, что согласно определению кодового расстояния по метрике Хэмминга, не важно, какое именно из 255 различных вариантов отличий реализовано в том или ином байте кадра Y, которое отличает его от значения в байте, стоящем в той же позиции внутри кадра X. То есть важен лишь сам факт отличия, а то, каким из 255 способов это отличие реализовано – не важно, от этого кодовое расстояние не меняется. В классической же геометрии расстояние зависит не только от самого факта отличия в той или иной координате между двумя точками евклидова пространства, но и величины отличия. Именно, поэтому при проведении аналогии мы должны соблюдать особую осторожность.

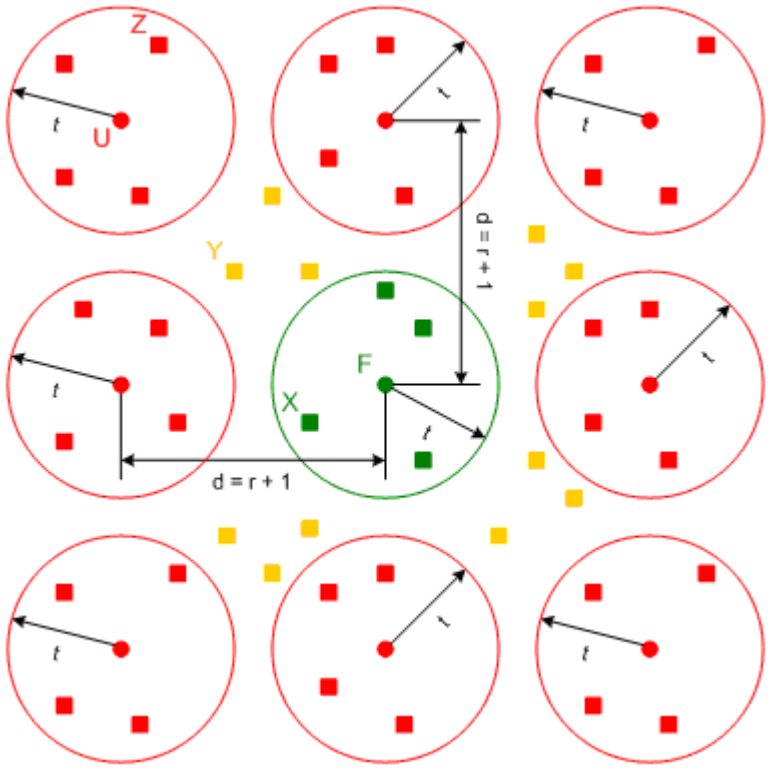

Рис. 4.1. «Геометрическая картина» небольшого участка пространства кадров.

В n-мерном пространстве кадров имеется подмножество «допустимых» кадров, на рисунке они обозначены маленькими кружочками и подмножество всех остальных «недопустимых» кадров, на рисунке они обозначены маленькими квадратиками. Расстояние между любыми двумя «допустимыми» кадрами, как минимум, $d = r + 1$. Теперь попробуем построить вокруг каждого «допустимого» кадра «шар» радиуса t так, чтобы этот радиус был максимально возможным, но при этом чтобы «шар» не пересекался (и даже не соприкасался) ни в одном кадре с «шарами», построенных вокруг ближайших соседних «допустимых» кадров. Очевидно, что сами «допустимые кадры» в этом случае будут являться центрами своего шара. Теперь, учитывая, что расстояние между ближайшими «допустимыми» кадрами $d = r + 1$, и то, что между шарами необходимо оставить «минимальный зазор», равный хотя бы 1, то очевидно, что радиус $t = \lfloor r/2 \rfloor$ - целая часть от результата деления r на 2. Отметим, что невыгодно иметь дело с нечетным количеством контрольных байтов r, поскольку в таком случае радиус t такой же, как и при $r - 1$, а «минимальный зазор» становится равным 2 (происходит жертвование единичным расстоянием в пользу не радиуса, а «зазора»). Например, при $r = 3$, расстояние между центрами («допустимыми» кадрами) ближайших «шаров» составляет $d = r + 1 = 4$, и вокруг них можно построить шары только радиуса $t = 1$ (при $t = 2$, шары будут уже соприкасаться), и под «минимальный зазор» уйдет уже две единицы расстояния. Точно такой же радиус мы бы имели и при $r = 2$, а под «зазор» в этом случае потратили бы только одну единицу расстояния (на рис. 4.2 это хорошо видно):

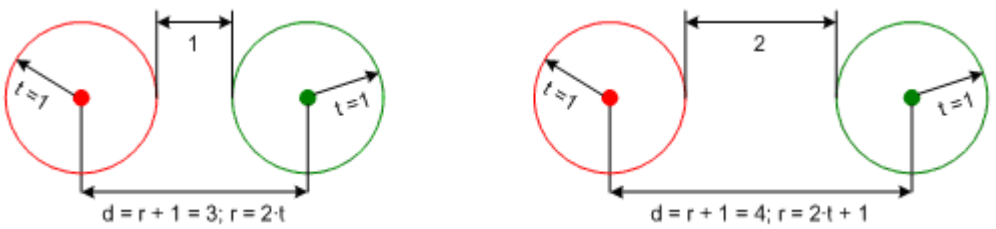

Рис. 4.2. Случаи чётного и нечётного количества контрольных байтов.

Тогда с учетом всего вышесказанного, мы можем сделать следующий важный вывод: для заданного радиуса $t \geq 1$, имеется принципиальная возможность преобразования «недопустимого» кадра, находящегося на расстоянии не более t от некоторого «допустимого» кадра, в этот самый «допустимый» вид. Иными словами, если «недопустимый» кадр принадлежит к какому-либо из «шаров» радиуса t с центром в некотором «допустимом» кадре, то имеется принципиальная возможность «притянуть» его к центру этого «шара». Однако, не стоит забывать, что в пространстве кадров, существует также и такие кадры, которые находятся за пределами всех «шаров», находятся, так сказать, в «промежутках» между «шарами». Однозначное преобразование подобных кадров в какой-либо «допустимый» вид, очевидно, принципиально невозможно.

Тогда при такой трактовке, очевидно, что t – это, не что иное, как **кратность исправляемых ошибок** – максимальное количество искаженных байтов, при котором принципиально возможно восстановление искаженного кадра в исходный неискаженный вид (причем исходный кадр, разумеется, должен быть «допустимым» кадром).

Следует особо отметить, что **восстановление и исправление – это не одно и то же**. **Восстановление** – это преобразование искаженного кадра в исходный вид – такой вид, который кадр имел до искажения, а **исправление** – это преобразование искаженного кадра в ближайший «допустимый» вид, если это вообще возможно (кадр находится на расстоянии не более t от некоторого «допустимого» кадра, не являющегося исходным).

Итак, пусть F – это исходный «допустимый» кадр. Тогда всего возможно 5 ситуаций:

1) Кадр F остается в исходном неискаженном виде, то есть **искажение отсутствует**.
2) Искажается не более t байтов, и получается искаженный кадр X, который находится в пределах «шара» радиуса t и возможно его «притяжение к центру», которым является исходный кадр F. То есть **искажение восстанавливаемое**.
3) Искажается более t байтов (от $t + 1$ до n), причем искаженный кадр Y не принадлежит ни одному «шару» (находится, так сказать, в «промежутке между шарами»). В таком случае невозможно не только восстановление исходного вида, но и вообще какое-либо исправление искаженного кадра, то есть **искажение неисправимое вообще**.
4) Искажается более t байтов (от $t + 1$ до n, если r четно, $r = 2 \cdot t$) или более $t + 1$ байтов (от $t + 2$ до n, если r нечетно, $r = 2 \cdot t + 1$), причем искаженный кадр Z принадлежит одному из «шаров» радиуса t, в центре которого находится другой «допустимый» кадр U. «Шар», в центре которого находится кадр U, является смежным (необязательно ближайшим соседним) по отношению к «шару», в центре которого находится исходный кадр F. Очевидно, что в этой ситуации принципиально возможно исправить искаженный кадр Z в «допустимый» кадр U, поскольку расстояние между ними $D(Z,U) \leq t$. Очевидно, после исправления мы получим «допустимый» кадр U, а не исходный кадр F. Это мы будем называть **смежно-исправимым искажением**.
5) Искажается более $2 \cdot t$ (от $2 \cdot t + 1$ до n, если r четно, $r = 2 \cdot t$) или более $2 \cdot t + 1$ байтов (от $2 \cdot t + 2$ до n, если r нечетно, $r = 2 \cdot t + 1$), причем искаженный кадр не просто принадлежит смежном «шару», радиуса t, в центре которого находится другой «допустимый» кадр U, а вообще совпадает с кадром U. Очевидно, в такой ситуации невозможно не то, что какое-либо восстановление или исправление, но и даже обнаружение самого факта искажения. Такое явление, хотя и довольно редкое, и все же вполне возможное. Это мы будем называть **маскируемым искажением**.

Примечание. Попытку исправления кадра со смежно-исправимым или маскируемым искажением в литературе [6, 14] иногда называют **ошибочным декодированием**. Однако, такое название опасно тем, что его можно спутать со случаями, когда декодирование работает неверно в силу ошибок, допущенных разработчиком программных или аппаратных реализаций алгоритма декодирования. Поэтому, чтобы не путать субъективные ошибки разработчиков с объективными (математически обоснованными) явлениями смежно-исправимого и маскируемого искажения, мы это будем называть **обманом декодера**. К тому же такой термин лучше отражает суть, когда речь заходит об умышленных искажениях.

Таким образом, в случае искажения не более t байтов, причем минимальное кодовое расстояние в этом случае либо $d = 2 \cdot t + 1$ (при $r = 2 \cdot t$), либо $d = 2 \cdot t + 2$ (при $r = 2 \cdot t + 1$), принципиально возможно однозначное и гарантированное восстановление искаженного кадра в исходный неискаженный вид. При искажении более t байтов, восстановление заведомо невозможно, и в зависимости от ситуации (описанных выше) возможно либо установление факта неисправимости кадра, либо смежное исправление, либо маскирование искажения. Заметим также, что смежное исправление возможно только при искажении более t байтов (при $r = 2 \cdot t$) или более $t + 1$ байтов (при $r = 2 \cdot t + 1$), а маскирование только при искажении более $2 \cdot t$ байтов (при $r = 2 \cdot t$) или более $2 \cdot t + 1$ байтов (при $r = 2 \cdot t + 1$).

Отметим, что с одной стороны нечетное количество контрольных байтов $r = 2 \cdot t + 1$ (при этом минимальное кодовое расстояние $d = 2 \cdot t + 2$) может показаться выгодным тем, что на 1 поднимаются соответствующие «нижние пороги» для маскируемого искажения и для смежно-исправимого искажения. Однако, «верхний порог» восстанавливаемого искажения остается прежним, равным t. Поэтому на практике, большее распространение имеет кодирование с минимальным кодовым расстоянием $d = 2 \cdot t + 1$, и, соответственно, с четным числом контрольных байтов $r = 2 \cdot t$. Далее мы будем рассматривать только случай $d = 2 \cdot t + 1$.

Таким образом, мы получили структурированное n-мерное пространство кадров, с минимальным кодовым расстоянием $d = 2 \cdot t + 1$ для подмножества «допустимых» кадров. Очевидно, что общее количество всевозможных кадров в этом пространстве равно 256^n:

$$| \Gamma^n | = 256^n \tag{4.4.2}$$

Распределение «допустимых» кадров

Следующий немаловажный вопрос заключается в том, какое именно количество «допустимых» кадров присутствует среди общего числа всевозможных кадров. Кроме того, с точки зрения анализа структуры n-мерного пространства кадров, еще более важно то, какое количество «допустимое» кадров присутствует на том или ином расстоянии $0 \le \theta \le n$ от некоторого заданного «допустимого» кадра. Иными словами, нас интересует функция количественного распределения «допустимых» кадров в n-мерном пространстве кадров.

Согласно теории кодирования [6-14], «нулевой» кадр, все байты которого равны нулю – является «допустимым». Мы можем использовать «нулевой» кадр, как некую «точку отсчета» в пространстве кадров, для того, чтобы вывести функцию количественного распределения. При этом выведенная функция также будет справедлива и для любого другого «допустимого» кадра, который точно также мог бы быть выбранным в качестве «точки» отсчета. Просто, относительно «нулевого» кадра вести рассуждения будет немного проще, и они будут более наглядными, чем относительно другого «допустимого» кадра.

Итак, пусть имеется «нулевой» кадр Θ, состоящий из n байтов, содержащих нуль, и этот кадр является «допустимым». Нетрудно подсчитать, что для заданного расстояния $0 \le \theta \le n$ общее количество кадров, у которых ровно θ ненулевых байтов (остальные $n - \theta$ байты содержат нуль), равно $C_n^\theta \cdot 255^\theta$, поскольку можно C_n^θ способами выбрать θ позиций среди n позиций, и каждую позицию независимо заменить на одно из 255 ненулевых значений. С геометрической точки зрения – это «поверхность сферы радиуса θ», и все кадры, у которых ровно θ ненулевых байтов (и $n - \theta$ нулевых байтов), лежат на поверхности этой «сферы». Таким образом, полученный нами результат в математической форме можно выразить следующим образом:

$$\begin{gathered} \left| \{X\} \right| = C_n^\theta \cdot 255^\theta \\ \forall X \in \Gamma^n : D(X, \Theta) = \theta; \quad \Gamma = GF(2^8) \end{gathered} \tag{4.5.1}$$

Отметим также, что совокупность всех «сфер» с радиусами от *0* до *w* образуют «шар радиуса *w*», который содержит все кадры, у которых не более *w* ненулевых байтов (то есть, они находятся на расстоянии не более *w* от «нулевого» кадра Θ). Очевидно, что можно подсчитать общее количество кадров в «шаре радиуса *w*», просуммировав количества кадров в соответствующих «сферах» с радиусами от *0* до *w*. Иными словами, общее количество кадров, находящихся на расстоянии не более чем *w* от «нулевого» кадра:

$$\left|\{X\}\right| = \sum_{\theta=0}^{w} C_n^{\theta} \cdot 255^{\theta}$$
$$\forall X \in \Gamma^n : 0 \le D(X,\Theta) \le w; \quad \Gamma = GF(2^8) \tag{4.5.2}$$

Особо отметим, что если положить $w = n$, то получаем «шар радиуса *n*», который, очевидно, содержит все кадры пространства, количество которых $\sum_{\theta=0}^{n} C_n^{\theta} \cdot 255^{\theta} = 256^n$.

Теперь же обозначим через A_{θ} количество «допустимых» кадров, находящихся на расстоянии ровно θ от «нулевого» кадра. Очевидно, что это количество не может превышать общего количества всевозможных кадров на расстоянии θ от «нулевого» кадра.

$$A_{\theta} = \left|\{X\}\right| \le C_n^{\theta} \cdot 255^{\theta}$$
$$\forall X \in \Re \subset \Gamma^n : D(X,\Theta) = \theta; \quad \Gamma = GF(2^8) \tag{4.5.3}$$

Где, $\Re$ - подмножество «допустимых» кадров в *n*-мерном пространстве кадров.

Для начала отметим, что при $\theta = 0$, существует только один единственный кадр – это сам «нулевой» кадр, и он является допустимым. Таким образом, $A_0 = 1$. Также отметим, что при $1 \le \theta \le 2 \cdot t$, допустимых кадров не может быть по определению, поскольку минимальное кодовое расстояние равно $d = 2 \cdot t + 1$ для подмножества «допустимых» кадров, а значит ближайшие «допустимые» кадры находятся на расстоянии не менее, чем $d = 2 \cdot t + 1$ от «нулевого» кадра. Иными словами, количество «допустимых» кадров при $1 \le \theta \le 2 \cdot t$ равно нулю: $A_1 = \ldots = A_{2t} = 0$. Анализ $A_{2t+1} \ldots A_n$ представляет собой нетривиальную задачу.

Рассчитаем самый простой случай: A_{2t+1}. В этом случае ровно $\theta = 2 \cdot t + 1$ байтов должны быть ненулевыми с одной стороны, и в то же время все «допустимые» наборы из $2 \cdot t + 1$ ненулевых байтов должны отличатся друг от друга всеми $2 \cdot t + 1$ байтами. Тогда если один из $2 \cdot t + 1$ байтов считать независимым, могущим принимать любое из 255 значений, а остальные $2 \cdot t$ - зависимыми от значения в независимом байте, то всего наборов, очевидно, будет ровно 255. Кроме того, учитывая, что $2 \cdot t + 1$ позиций из *n* позиций можно выбрать C_n^{2t+1} способами, то в итоге получаем, что $A_{2t+1} = C_n^{2t+1} \cdot 255$. Это формулу можно также переписать в следующем эквивалентном виде $A_{2t+1} = C_n^{2t+1} \cdot (256-1) \cdot C_{2t+1}^0$ (ниже будет понятно, зачем понадобилось такое преобразование).

Используя аналогичные, но более сложные приемы комбинаторного анализа можно вывести, что $A_{2t+2} = C_n^{2t+2} \cdot \left(255^2 - (2t) \cdot 255\right) = C_n^{2t+2} \cdot \left((256^2 - 1) - (2t+2) \cdot 255\right) =$

$$= C_n^{2t+2} \cdot \left((256^2 - 1) \cdot C_{2t+2}^0 - (256-1) \cdot C_{2t+2}^1\right).$$

Продолжая далее, найдем все остальные компоненты A_{θ} вплоть до последнего A_n:

$$A_n = C_n^n \cdot \left((256^{n-2t} - 1) \cdot C_n^0 - (256^{n-2t-1} - 1) \cdot C_n^1 + \ldots + (-1)^{n-2t-1} \cdot (256-1) \cdot C_n^{n-2t-1}\right)$$

Тогда, подводя итог всему вышеизложенному, можем записать окончательную формулу для функции количественного распределения «допустимых» кадров:

$$\begin{cases} A_0 = 1; \quad A_1 = \ldots = A_{2t} = 0 \\ A_\theta = C_n^\theta \cdot \left(\sum\limits_{i=0}^{\theta-2t-1} (-1)^i \cdot C_\theta^i \cdot (256^{\theta-2t-i} - 1) \right); \quad 2t+1 \le \theta \le n \end{cases} \tag{4.5.4}$$

Получив функцию количественного распределения «допустимых кадров», теперь мы можем оценить общее количество «допустимых» кадров в n-мерном пространстве кадров.

Для этого требуется вычислить сумму функции распределения по всем $0 \le \theta \le n$ (сосчитать все «допустимые» кадры, находящиеся на всех возможных расстояниях $0 \le \theta \le n$ от «нулевого» кадра): $\sum\limits_{\theta=0}^{n} A_\theta$. Заметим, что $A_0 = 1$, а $A_1 = \ldots = A_{2t} = 0$, поэтому, очевидно, что частичная сумма $\sum\limits_{\theta=0}^{2t} A_\theta = 1$, и нам нужно только вычислить сумму $\sum\limits_{\theta=2t+1}^{n} A_\theta$.

Итак, $\sum\limits_{\theta=2t+1}^{n} A_\theta = \sum\limits_{\theta=2t+1}^{n} C_n^\theta \cdot \left(\sum\limits_{i=0}^{\theta-2t-1} (-1)^i \cdot C_\theta^i \cdot (256^{\theta-2t-i} - 1) \right)$. Обозначим, $q = \theta - 2t - i$. Тогда получим: $\sum\limits_{\theta=2t+1}^{n} C_n^\theta \cdot \left(\sum\limits_{q=1}^{\theta-2t} (-1)^{\theta-2t-q} \cdot C_\theta^{\theta-2t-q} \cdot (256^q - 1) \right)$.

Заметим, что биномиальный коэффициент $C_\theta^{\theta-2t-q}$ не равен нулю, только при $\theta - 2t - q \ge 0 \Rightarrow q \le \theta - 2t$. Поэтому мы можем расширить диапазон суммирования по q вплоть до n, то так как при $\theta - 2t + 1 \le q \le n$ коэффициент $C_\theta^{\theta-2t-q}$ равен нулю. Тогда имеем: $\sum\limits_{\theta=2t+1}^{n} C_n^\theta \cdot \left(\sum\limits_{q=1}^{n} (-1)^{\theta-2t-q} \cdot C_\theta^{\theta-2t-q} \cdot (256^q - 1) \right)$. Теперь мы можем легко переставить порядок суммирования, поскольку пределы внутреннего суммирования не содержат θ, и получим: $\sum\limits_{q=1}^{n} (256^q - 1) \left(\sum\limits_{\theta=2t+1}^{n} C_n^\theta \cdot (-1)^{\theta-2t-q} \cdot C_\theta^{\theta-2t-q} \right)$. Теперь учтем, что коэффициент $C_\theta^{\theta-2t-q}$ равен нулю, при $\theta < 2t + q$, поэтому мы можем нижний предел внутреннего суммирования приподнять до $2t + q$. Кроме того, произведение $C_n^\theta \cdot C_\theta^{\theta-2t-q}$ можно преобразовать следующим образом $\frac{n!}{\theta!(n-\theta)!} \cdot \frac{\theta!}{(2t+q)!(\theta-(2t+q))!} =$

$= \frac{n!}{(n-\theta)!} \cdot \frac{1}{(2t+q)!(\theta-(2t+q))!} \cdot \frac{(n-(2t+q))!}{(n-(2t+q))!} = C_n^{2t+q} \cdot C_{n-(2t+q)}^{n-\theta}$. Теперь один из биномиальных коэффициентов не содержит θ, его можно вынести за пределы внутреннего суммирования: $\sum\limits_{q=1}^{n} (256^q - 1) \cdot C_n^{2t+q} \cdot \left(\sum\limits_{\theta=2t+q}^{n} \cdot C_{n-(2t+q)}^{n-\theta} \cdot (-1)^{\theta-(2t+q)} \right)$. Теперь обозначим, $j = \theta - (2t + q)$, при этом пределы суммирования будут: $j = 0 \ldots n - (2t + q)$.

Тогда получим: $\sum\limits_{q=1}^{n} (256^q - 1) \cdot C_n^{2t+q} \cdot \left(\sum\limits_{j=0}^{n-(2t+q)} \cdot C_{n-(2t+q)}^{n-(2t+q)-j} \cdot (-1)^j \right)$.

Легко заметить, что внутренняя сумма это не что иное, как разложение в ряд выражения $(1-1)^{n-(2t+q)}$, которое обращается в нуль во всех случаях, кроме случая, когда $n-(2t+q)=0 \Rightarrow q=n-2t$, и в этом случае $(1-1)^0 = 1$. Таким образом, внутренняя сумма для всех $q \neq n-2t$ равна нулю, и только при $q=n-2t$ она равна 1. Тогда, очевидно, что для внешнего суммирования будет единственное ненулевое слагаемое при $q=n-2t$ и, оно будет следующим: $(256^{n-2t}-1)\cdot C_n^n \cdot 1 = 256^{n-2t}-1$. Тогда окончательно получаем:

$$\sum_{\theta=2t+1}^{n} A_\theta = 256^{n-2t}-1 \qquad (4.5.5)$$

Наконец, учитывая, что $A_0=1$, $A_1=\ldots=A_{2t}=0$, а также учитывая, $n-2t=n-r=k$, получаем общее количество «допустимых» кадров в *n*-мерном пространстве кадров:

$$\sum_{\theta=0}^{n} A_\theta = 256^{n-2t} = 256^k \qquad (4.5.6)$$

Мы получили очень важный результат. Общее количество «допустимых» кадров в *n*-мерном пространстве кадров с минимальным кодовым расстоянием $d = 2\cdot t + 1$ в точности равно общему количеству всевозможных информационных сообщений в *k*-мерном пространстве информационных сообщений, причем размерность пространства: $k = n - 2\cdot t$.

Коды Рида-Соломона

Следующий вопрос заключается в том, каким именно образом вычислять контрольные байты так, чтобы с помощью них любые информационные сообщения длины k пространства информационных сообщений с минимальным кодовым расстоянием $d = 1$, можно было преобразовывать в «допустимые» кадры длины $n=k+r$, минимальное кодовое расстояние для которых равно $d = r+1$ в *n*-мерном пространстве кадров, причем $r = 2\cdot t$.

Как один из вариантов теория кодирования [6-14] предлагает следующий подход с использованием, так называемых кодов Рида-Соломона, определенных над конечным полем Галуа $GF(p^m)$. Вводится понятие, так называемого **порождающего полинома**, который в общем случае для поля Галуа $GF(p^m)$, примитивного элемента $\alpha \in GF(p^m)$ этого поля, а также минимального кодового расстояния d задается следующим образом:

$$g(x) = \sum_{i=0}^{d-1} g_i \cdot x^i = \prod_{s=b}^{b+d-2} \left(x-\alpha^s\right) = \prod_{s=1}^{d-1} \left(x-\alpha^{s+b-1}\right); \quad \alpha, g_i \in GF(p^m) \qquad (4.6.1)$$

Где, b – некоторая константа, принадлежащая соответствующему кольцу логарифмов $LR(p^m-1)$. По сути b – неотрицательное целое число в диапазоне от 0 до p^m-2. В технологии кодирования обычно принимается $b = 1$, хотя на практике также достаточно часто используется случай $b = 0$.

Особо отметим, что в силу свойства примитивного элемента $\alpha^{p^m-1}=1$, а это значит, что $\alpha^s = \alpha^{s \bmod (p^m-1)}$ так же, как и $\alpha^{s+b-1} = \alpha^{(s+b-1) \bmod (p^m-1)}$. Поэтому более корректно формула для порождающего полинома выглядит как:

$$g(x) = \sum_{i=0}^{d-1} g_i \cdot x^i = \prod_{s=1}^{d-1} \left(x-\alpha^{(s+b-1) \bmod (p^m-1)}\right); \quad \alpha, g_i \in GF(p^m) \qquad (4.6.2)$$

Таким образом, если показатель $s+b+1$ степени, в которую возводится примитивный элемент α, больше либо равен p^m-1, то он «по кольцу заводится» в диапазон $0\ldots p^m-2$ кольца логарифмов $LR(p^m-1)$, путем вычисления остатка по модулю p^m-1.

Теперь отметим, что в рамках технологии кодирования мы ограничиваемся рассмотрением только полей $GF(2^m)$ характеристики 2. В таких полях операция сложения и вычитания элементов поля сводятся к операции «побитового» XOR, обозначаемой как $\oplus$. Кроме того, мы рассматриваем случай четного количества контрольных байтов, при котором кодовое расстояние равно: $d = r + 1 = 2 \cdot t + 1$. Наконец, отметим, что мы рассматриваем коды Рида-Соломона конкретно на примере поля Галуа $GF(2^8)$ образованного при помощи заданного неприводимого многочлена $p(x)$ и с заданным примитивным элементом α. В этом случае порождающий полином принимает вид:

$$g(x) = \sum_{i=0}^{r} g_i \cdot x^i = \prod_{s=1}^{r} \left(x \oplus \alpha^{(s+b-1) \bmod (2^8-1)} \right); \quad g_i \in GF(2^8); \quad r = 2 \cdot t \quad (4.6.3)$$

Порождающий полином согласно теории кодирования обеспечивает «генерацию» $r = 2 \cdot t$ избыточных байтов для любого информационного сообщения, состоящего из k байтов, так что результирующий кадр, состоящий из $n = k + r$ байтов, получающийся в результате соединения k информационных байтов с r контрольными байтами, является «допустимым» кадром в n-мерном пространстве кадров.

Особо отметим, что порождающий полином $g(x)$ имеет степень $r = 2 \cdot t$, и корнями уравнения $g(x) = 0$ являются следующие элементы поля Галуа $GF(2^8)$: $\alpha^{b \bmod (2^8-1)}, \alpha^{(b+1) \bmod (2^8-1)} \ldots, \alpha^{(r+b-1) \bmod (2^8-1)}$, причем для кодов Рида-Соломона важно, чтобы все корни были различными (некратными). Заметим, что, начиная с $t = 128$, имеем $r = 256 \Rightarrow \alpha^{(r+b-1) \bmod (2^8-1)} = \alpha^{(256+b-1) \bmod (2^8-1)} = \alpha^{b \bmod (2^8-1)}$, иными словами корни начинают повторяться. Из этого следует, что максимально возможная кратность исправляемой ошибки конкретно для поля Галуа $GF(2^8)$, при котором $r < 256$:

$$t_{\max} = (2^8 - 2)/2 = 127 \quad (4.6.4)$$

Таким образом, мы ограничиваем сверху кратность исправляемой ошибки, $t \le 127$, и соответственно, число контрольных байтов $r \le 254$. В такой ситуации в формуле для порождающего полинома также имеем $1 \le s \le 254$, так как $1 \le s \le r$. Также имеем $\alpha^{(s+b-1) \bmod (2^8-1)} = \alpha^{s \bmod (2^8-1)} \cdot \alpha^{(b-1) \bmod (2^8-1)}$, и, учитывая, что $1 \le s \le 254$, имеем $s \bmod (2^8 - 1) = s$, а тогда $\alpha^{(s+b-1) \bmod (2^8-1)} = \alpha^s \cdot \alpha^{(b-1) \bmod (2^8-1)}$.

Введем следующее обозначение:

$$\beta = \alpha^{(b-1) \bmod (2^8-1)} = \begin{cases} \alpha^{b-1}, & b = 1 \ldots 2^8 - 2 \\ 1/\alpha, & b = 0 \end{cases} \quad (4.6.5)$$

Особо отметим, что если в качестве константы b принимается 1, то $\beta = \alpha^0 = 1$.

Тогда формула для порождающего полинома принимает следующий вид:

$$g(x) = \sum_{i=0}^{r} g_i \cdot x^i = \prod_{s=1}^{r} \left(x \oplus \beta \cdot \alpha^s \right); \quad g_i \in GF(2^8); \quad r = 2 \cdot t \quad (4.6.6)$$

<u>Формирование порождающего полинома</u>. Формирование полинома $g(x)$ связано с перемножением полиномов первой степени вида $(x \oplus \beta \cdot \alpha^s); s = 1 \ldots r$. Очевидно, что процесс формирования $g(x)$ можно записать в виде рекуррентной итерационной процедуры:

$$\begin{cases} g^{(s)}(x) = g^{(s-1)}(x) \cdot (x \oplus \beta \cdot \alpha^s) \\ s = 1 \ldots r; \quad g^{(0)}(x) = 1 \end{cases} \quad (4.6.7)$$

Заметим, что поскольку мы начинаем с полинома нулевой степени $g^{(0)}(x)=1$, а далее на каждой итерации $s=1\ldots r$, выполняем умножение на полином первой степени, то очевидно, что степень $g^{(s)}(x)$ будет равна *s*. Кроме того, учтем, что полином-множитель $(x\oplus\beta\cdot\alpha^s)$ при *x* содержит коэффициент равный 1. Тогда стоит рассмотреть более детально вычисления на итерации s с целью их оптимизации: $g^{(s)}(x)=g^{(s-1)}(x)\cdot(x\oplus\beta\cdot\alpha^s)=$ $=\left(\sum_{i=0}^{s-1} g_i^{(s-1)}\cdot x^i\right)\cdot\left(x\oplus\beta\cdot\alpha^s\right)=\sum_{i=0}^{s-1} g_i^{(s-1)}\cdot x^{i+1}\oplus\sum_{i=0}^{s-1} g_i^{(s-1)}\cdot\beta\cdot\alpha^s\cdot x^i$. Заменим в первом суммировании индекс $j=i+1$ и получим: $\sum_{j=1}^{s} g_{j-1}^{(s-1)}\cdot x^j\oplus\sum_{i=0}^{s-1} g_i^{(s-1)}\cdot\beta\cdot\alpha^s\cdot x^i$. Вынесем из первой суммы слагаемое при $j=s$, а из второй – слагаемое при $i=0$ и получим: $g_{s-1}^{(s-1)}\cdot x^s\oplus\left(\sum_{j=1}^{s-1} g_{j-1}^{(s-1)}\cdot x^j\oplus\sum_{i=1}^{s-1} g_i^{(s-1)}\cdot\beta\cdot\alpha^s\cdot x^i\right)\oplus g_0^{(s-1)}\cdot\beta\cdot\alpha^s$. Объединим обе суммы, и получим: $g_{s-1}^{(s-1)}\cdot x^s\oplus\left(\sum_{j=1}^{s-1}\left(g_{j-1}^{(s-1)}\oplus g_j^{(s-1)}\cdot\beta\cdot\alpha^s\right)\cdot x^j\right)\oplus g_0^{(s-1)}\cdot\beta\cdot\alpha^s$.

Тогда, анализируя полученную формулу, выводим рекуррентную итерационную процедуру для вычисления коэффициентов полинома $g^{(s)}(x)$ на итерациях $s=1\ldots r$:

$$g_j^{(s)}=\begin{cases} g_{s-1}^{(s-1)}; & j=s \\ g_{j-1}^{(s-1)}\oplus g_j^{(s-1)}\cdot\beta\cdot\alpha^s; & j=1\ldots s-1 \\ g_0^{(s-1)}\cdot\beta\cdot\alpha^s; & j=0 \end{cases} \qquad (4.6.8)$$

$$s=1\ldots r;\quad g_0^{(0)}=1;\quad g_1^{(0)}=\ldots=g_r^{(0)}=0;$$

На последней $r-й$ итерации мы получаем искомый порождающий полином $g(x)$.

Пример. Сформируем порождающий полином $g(x)$ для случая исправления одиночных ошибок (t = 1) и для случая исправления ошибок двойной кратности (t = 2) при параметре b = 0 и при параметре b = 1, для поля Галуа $GF(2^8)$ с неприводимым многочленом $p(x)=x^8\oplus x^4\oplus x^3\oplus x^2\oplus 1$ и примитивным элементом $\alpha=2$.

- При b = 0 имеем $\beta=\alpha^{-1}=2^{-1}$.
  - При t = 1, имеем r = 2 и $g(x)=\prod_{s=1}^{2}\left(x\oplus 2^{-1}\cdot 2^s\right)=x^2\oplus 3\cdot x\oplus 2$.
  - При t = 2, имеем r = 4 и $g(x)=\prod_{s=1}^{4}\left(x\oplus 2^{-1}\cdot 2^s\right)=x^4\oplus 15\cdot x^3\oplus 54\cdot x^2\oplus 120\cdot x\oplus 64$.
- При b = 1 имеем $\beta=\alpha^0=1$.
  - При t = 1, имеем r = 2 и $g(x)=\prod_{s=1}^{2}\left(x\oplus 1\cdot 2^s\right)=x^2\oplus 6\cdot x\oplus 8$.
  - При t = 2, имеем r = 4 и $g(x)=\prod_{s=1}^{4}\left(x\oplus 1\cdot 2^s\right)=x^4\oplus 30\cdot x^3\oplus 216\cdot x^2\oplus 231\cdot x\oplus 116$.

Тогда, пусть задан 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 (умноженного на x^r), на порождающий полином $g(x)$. Иными словами, пусть задан полином:

$$R(x) = (x^r \cdot M(x)) \bmod (g(x)) = \sum_{i=0}^{r-1} R_i \cdot x^i \qquad (4.7)$$

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

M_{k-1}	…	M_0	R_{r-1}	…	R_0

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

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 $R(x)$:

$$F(x) = (x^r \cdot M(x)) \oplus R(x) \qquad (4.8)$$

Следует особо отметить (это будет показано далее в разделе декодирования) при декодировании кодов Рида-Соломона вычисляются локаторы (позиции внутри кадра *F*) ошибок, которые принадлежат кольцу логарифмов $LR(2^8 - 1)$ и по сути являются целыми числами в диапазоне от 0 до 254. Соответственно, локаторы могут указывать только в пределах от 0-й до 254-й позиции внутри кадра *F*. Из этого следует ограничение, что максимальная длина кадра *F*, при котором возможно корректное декодирование, равна 255:

$$n_{\max} = 255 \qquad (4.9)$$

Таким образом, конкретно для поля Галуа $GF(2^8)$, сумма количества информационных байтов и количества контрольных байтов должна быть меньше либо равна 255. Это, конечно, очень маленький размер, поэтому на практике большие массивы данных предварительно разбивают на блоки размером, не превышающим $255 - r$, и кодируют их по отдельности.

Кроме того, относительно полинома $F(x)$, в теории кодирования делается два достаточно простых, но в тоже время чрезвычайно важных утверждения:

- Полином $F(x)$ делится без остатка на порождающий полином $g(x)$.
- Уравнение $F(x) = 0$ имеет те же корни, что и уравнение $g(x) = 0$. Корнями являются элементы поля Галуа: $x^* = \beta \cdot \alpha^1, \ldots, \beta \cdot \alpha^r$, то есть $F(\beta \cdot \alpha^S) = g(\beta \cdot \alpha^S) = 0$, $s = 1 \ldots r$.

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 и подстановка в него любого из корней $x^* = \beta \cdot \alpha^1, \ldots, \beta \cdot \alpha^r$ 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 $r = 2t$ байтов – всегда) деление на порождающий полином даст ненулевой остаток, также как и подстановка корней $x^* = \beta \cdot \alpha^1, \ldots, \beta \cdot \alpha^r$ в полином $F(x)$ даст ненулевые значения.

Тогда с учетом вышесказанного можем представить несложную схему алгоритма кодирования (рис. 4.3) информационных сообщений с применением кодов Рида-Соломона, при заданной длине сообщения k и кратности исправляемой ошибки t.

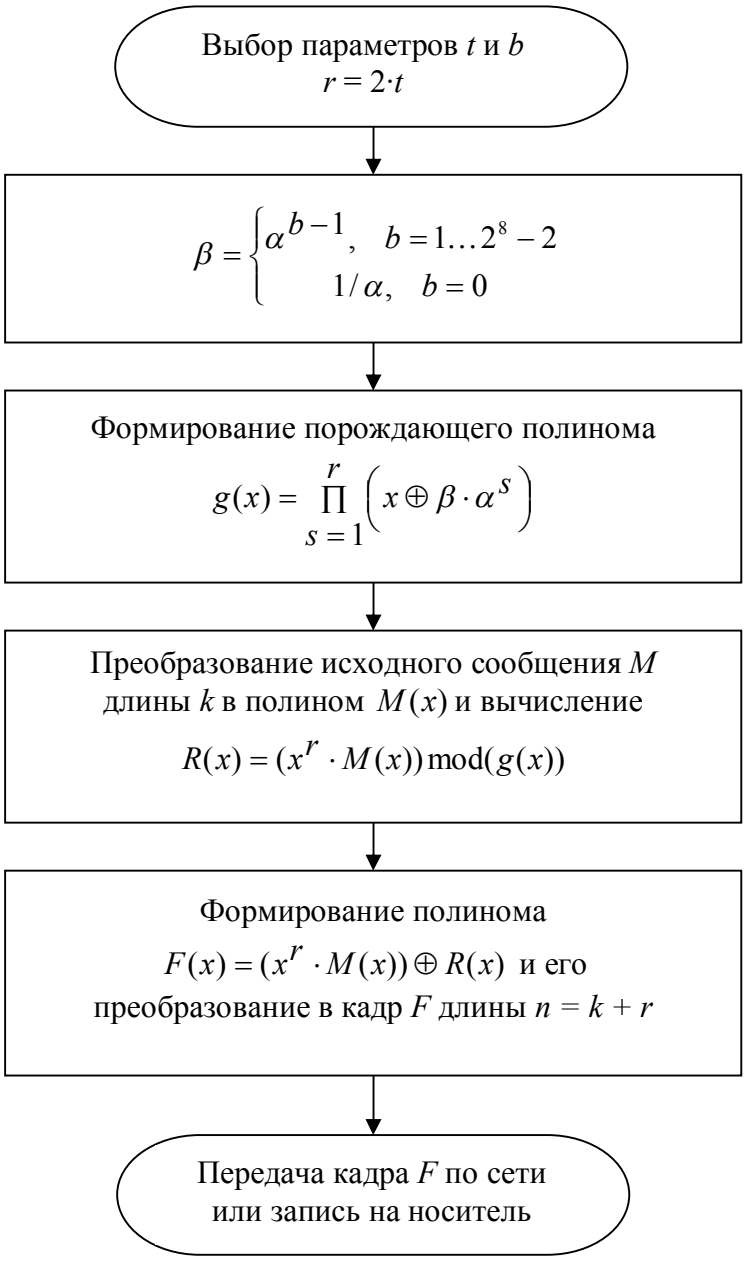

Рис. 4.3. Схема алгоритма кодирования информационного сообщения с применением кодов Рида-Соломона.

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 искажений** $E(x)$ и в результате будем иметь **полином искаженного кадра** $C(x) = F(x) \oplus E(x)$ на момент приема сообщения из сети или чтении данных с носителя. Заметим, что 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

Контрольные вопросы

1. Что такое кодовое расстояние и как оно определяется?

2. Какая связь между минимальным кодовым расстоянием в заданном пространстве и кратностью исправляемых ошибок?

3. В чем суть смежно-исправимых искажений? В каком случае подобные искажения можно считать признаком попытки дезинформации?

4. Чем обусловлено ограничение на размер кадра и максимальную возможную кратность исправляемых ошибок при кодировании с использованием кодов Рида-Соломона?

5. Найдите порождающий полином $g(x)$, над полем $GF(2^8)$ с неприводимым многочленом $p(x) = x^8 \oplus x^4 \oplus x^3 \oplus x^2 \oplus 1$ и примитивным элементом $\alpha = 2$, для кратности исправляемых ошибок $t = 8$ и константы $b = 1$.

6. Вычислите полином кадра $F(x)$ для заданного информационного полинома $M(x) = 72 \cdot x^4 \oplus 101 \cdot x^3 \oplus 108 \cdot x^2 \oplus 108 \cdot x \oplus 111$, используя порождающий полином $g(x) = x^4 \oplus 30 \cdot x^3 \oplus 216 \cdot x^2 \oplus 231 \cdot x \oplus 116$, над полем $GF(2^8)$ с неприводимым многочленом $p(x) = x^8 \oplus x^4 \oplus x^3 \oplus x^2 \oplus 1$ и примитивным элементом $\alpha = 2$.

7. В каких случаях остаток полинома кадра по модулю порождающего полинома равен нулю, несмотря на то, что кадр был заведомо искажен?

5. Декодирование кодов Рида-Соломона

Пусть имеется кадр C длины $n = k + r$, состоящий из k информационных и r избыточных байтов, принятый из сети или считанный с носителя. Тогда полином $C(x)$ кадра можно представить как сумму полинома $F(x)$ исходного неискаженного кадра F, отправленного по сети или записанного на носитель, и некоторого полинома ошибок $E(x)$:

$$C(x) = F(x) \oplus E(x) \tag{5.1}$$

Тогда будем называть **синдромом ошибки** (синдром – совокупность симптомов, характеризующее заболевание) совокупность компонент $S_j, j = 1 \ldots r$, вычисляемых путем подстановки в полином $C(x)$ корней $\beta \cdot \alpha^1, \ldots, \beta \cdot \alpha^r$ порождающего полинома $g(x)$. Учитывая то, что мы ранее установили, что полином $F(x)$ неискаженного кадра обращается в нуль при подстановке в него корней $\beta \cdot \alpha^1, \ldots, \beta \cdot \alpha^r$, то справедливо следующее равенство:

$$S_j = C(\beta \cdot \alpha^j) = E(\beta \cdot \alpha^j), j = 1 \ldots r \tag{5.2.1}$$

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

Кроме того, дополнительно вводится понятие полинома синдрома ошибок $S(x)$, который может быть представлен как:

$$S(x) = S_1 \oplus S_2 \cdot x \oplus \ldots \oplus S_r \cdot x^{r-1} = \sum_{j=0}^{r-1} S_{j+1} \cdot x^j \tag{5.2.2}$$

Особо отметим, что в качестве коэффициентов полинома синдрома используются компоненты синдрома, которые нумеруются не с 0 по *r* – 1 как коэффициенты, а с 1 по *r*.

Очевидно, что если все компоненты синдрома равны нулю, то можно говорить, что искажений либо не было, либо произошел редкий случай, когда искажение было таким, что кадр превратился в другой «допустимый» кадр в заданном n-мерном пространстве кадров, для которого синдром также равен нулю – это мы называем случаем маскируемого искажения. В противном случае, если синдром *S* ненулевой, то можно попытаться декодировать и исправить ошибки, произошедшие в кадре.

Однако, несмотря на то, что компоненты синдрома *S* легко определяются по полиному ошибок $E(x)$, обратное же вычисление полинома $E(x)$ по известным компонентам синдрома ошибок 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*, причем $\tau \le t$.

Мы можем представить полином ошибок в следующем виде:

$$E(x) = \sum_{l=1}^{\tau} v_l \cdot x^{u_l} \tag{5.3}$$

Где, u_l - это локаторы ошибок, а v_l - величины ошибок, $l = 1 \ldots \tau$.

Согласно теории кодирования [6-14], декодирование истинных локаторов и величин ошибок, и восстановление исходного кадра *F* возможно только тогда, когда истинное количество искаженных байтов не превышает *t* – кратности исправляемых ошибок. При искажении более *t* байтов возможно либо установление факта неисправимости кадра по косвенным признакам – не удается найти ровно τ локаторов, причем таких, которые бы указывали на позиции байтов в пределах границ кадра длины *n*, либо смежное исправление. Смежное исправление всегда сопровождается расхождением между предполагаемым количеством искаженных байтов $\tau \le t$, и истинным количеством, которое больше *t*.

Тогда, учитывая, что $S_j = E(\beta \cdot \alpha^j), j = 1 \ldots r$, а также учитывая, что $(\beta \cdot \alpha^j)^u = \beta^u \cdot (\alpha^u)^j$, мы можем выразить компоненты синдрома в следующем виде:

$$S_j = \sum_{l=1}^{\tau} v_l \cdot (\beta \cdot \alpha^j)^{u_l} = \sum_{l=1}^{\tau} v_l \cdot \beta^{u_l} \cdot (\alpha^{u_l})^j, j = 1 \ldots r \qquad (5.4.1)$$

Для наглядности перепишем систему уравнений в развернутом виде:

$$\begin{cases} v_1 \cdot \beta^{u_1} \cdot (\alpha^{u_1})^1 \oplus \ldots \oplus v_\tau \cdot \beta^{u_\tau} \cdot (\alpha^{u_\tau})^1 = S_1 \\ \vdots \\ v_1 \cdot \beta^{u_1} \cdot (\alpha^{u_1})^r \oplus \ldots \oplus v_\tau \cdot \beta^{u_\tau} \cdot (\alpha^{u_\tau})^r = S_r \end{cases} \qquad (5.4.2)$$

Очевидно, что система состоит из r уравнений с 2τ неизвестными (τ локаторов и τ величин ошибок), к тому же еще сами уравнения нелинейные, поэтому поиск решения затруднен. Простой полный перебор решений (всевозможные комбинации значений для локаторов и величин) малоперспективен, поскольку, например, при длине кадре n = 255 и кратности предполагаемой ошибки всего $\tau = 2$, это уже составит $(255 \cdot 254/2) \cdot 255^2 > 2$ млрд. комбинаций. Поэтому в теории кодирования предлагается более элегантный и эффективный метод поиска локаторов и величин ошибок, хотя и непростой для понимания.

Введем понятие, так называемого, **полинома локаторов ошибок** следующего вида:

$$\Lambda(x) = 1 \oplus \sum_{i=1}^{\tau} \Lambda_i x^i = \prod_{l=1}^{\tau} \left(1 \oplus x \cdot \alpha^{u_l} \right); \quad \Lambda_0 = 1 \qquad (5.5)$$

Полином локаторов ошибок специально задается таким образом, чтобы элементы $1/\alpha^{u_1}, \ldots, 1/\alpha^{u_\tau}$ были корнями уравнения $\Lambda(x) = 0$, а сами степени $u_1, \ldots, u_\tau$, являлись предполагаемыми локаторами местоположения ошибок в кадре *C*.

Тогда мы можем записать равенство следующего вида, умножив полином локаторов ошибок на l-ое слагаемое суммы, определяющей j-ую компоненту синдрома:

$$\Lambda(x) \cdot v_l \beta^{u_l} (\alpha^{u_l})^j = v_l \beta^{u_l} (\alpha^{u_l})^j \cdot 1 \oplus v_l \beta^{u_l} (\alpha^{u_l})^j \cdot \Lambda_1 \cdot x \oplus \ldots \oplus v_l \beta^{u_l} (\alpha^{u_l})^j \cdot \Lambda_\tau \cdot x^\tau$$

Теперь мы можем последовательно для каждого $l = 1 \ldots \tau$ в получившееся равенство в качестве аргумента x подставлять корни уравнения $\Lambda(x) = 0$. В итоге получим l равенств, у которых левая часть будет равна нулю, поскольку при вышеуказанном значении аргумента полином локаторов $\Lambda(x)$ обращается в нуль согласно его определению. После этого, мы почленно сложим все равенства вместе и в итоге получим итоговое равенство:

$$0 = 1 \cdot \sum_{l=1}^{\tau} \left(v_l \beta^{u_l} (\alpha^{u_l})^j \right) \oplus \Lambda_1 \cdot \sum_{l=1}^{\tau} \left(v_l \beta^{u_l} (\alpha^{u_l})^{j-1} \right) \oplus \ldots \oplus \Lambda_\tau \cdot \sum_{l=1}^{\tau} \left(v_l \beta^{u_l} (\alpha^{u_l})^{j-\tau} \right)$$

А теперь, заметим, что в суммированиях по $l = 1 \ldots \tau$, мы имеем не что иное, как компоненты синдрома: $S_j, S_{j-1} \ldots S_{j-\tau}$. Тогда, для всех $j = \tau + 1 \ldots r$ имеем равенства: $0 = S_j \oplus \Lambda_1 \cdot S_{j-1} \oplus \ldots \oplus \Lambda_\tau \cdot S_{j-\tau}$. Наконец, перенеся S_j на левую сторону равенства, получаем, так называемую, **ключевую систему уравнений декодирования**, связывающую компоненты синдрома *S* и коэффициенты полинома локаторов ошибок $\Lambda(x)$:

$$S_j = \sum_{i=1}^{\tau} \Lambda_i \cdot S_{j-i}; \quad j = \tau + 1 \ldots r \qquad (5.6.1)$$

Если переписать систему в развернутом виде, учитывая что $r = 2 \cdot t$, то получим:

$$\begin{cases} \Lambda_1 \cdot S_\tau \oplus \ldots \oplus \Lambda_\tau \cdot S_1 = S_{\tau+1} \\ \Lambda_1 \cdot S_{\tau+1} \oplus \ldots \oplus \Lambda_\tau \cdot S_2 = S_{\tau+2} \\ \vdots \\ \Lambda_1 \cdot S_{2t-1} \oplus \ldots \oplus \Lambda_\tau \cdot S_{2t-\tau} = S_{2t} \end{cases} \quad (5.6.2)$$

Как видим, это не просто система уравнений, а система из $2t - \tau$ уравнений с τ неизвестными, причем сам параметр τ неизвестен. Но, по крайней мере, сами уравнения системы являются линейными.

- При $\tau = 0$, в случае предположения отсутствия ошибок, система превращается в $2t$ простых уравнений вида $\Lambda_0 S_j = S_j, j = 1 \ldots 2t$, из которых очевидно, что $\Lambda_0 = 1$ (это также очевидно и из самого определения полинома локаторов).
- При $1 \le \tau \le t$, система имеет однозначное решение, поскольку число уравнений $(2t - \tau) \in [t, 2t-1]$ больше либо равно (при $\tau = t$), чем соответствующее число неизвестных $\tau \in [1, t]$. 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 $(2t - \tau) \in [1, t-1]$, оказывается меньше, чем число неизвестных $\tau \in [t+1, 2t-1]$, и нахождение однозначного решения становится невозможным.
- При $\tau \ge 2t$, система 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 [6-14]. 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 искомым 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).

$$\left\{ \begin{array}{c} S_j = \sum_{i=1}^{\tau} \Lambda_i \cdot S_{j-i} \\ \tau = \deg \Lambda(x) \to \min \\ j = \tau + 1 \ldots 2t \end{array} \right\} \Rightarrow \left\{ \begin{array}{c} \tau \\ \Lambda(x) \end{array} \right\} \quad (5.6.3)$$

Коды Рида-Соломона и основанные на них методы кодирования, обнаружения и исправления ошибок не получили бы широкого распространения, если бы в свое время ученые Берлекэмп и Месси в 1968-1969 годах не предложили чрезвычайно эффективный алгоритм [6-14] решения этой задачи с вычислительной сложностью $\sim \tau^2$.

Особо отметим, что алгоритм всегда работает из того допущения, что предполагаемое количество ошибок $\tau \le t$. Если истинное количество искаженных байтов также ≤ t, то оно гарантированно совпадает с τ, и алгоритм дает истинный полином локаторов. В случае же, если истинное количество искаженных байтов > t, то в большинстве случаев алгоритм также находит некоторый полином локаторов минимальной степени, удовлетворяющий системе уравнений, однако, фактически такой полином в лучшем случае приводит к установлению факта неисправимости кадра, а в худшем случае – к смежному исправлению.

Ниже представлена схема алгоритма нахождения полинома локаторов (рис. 5.1).

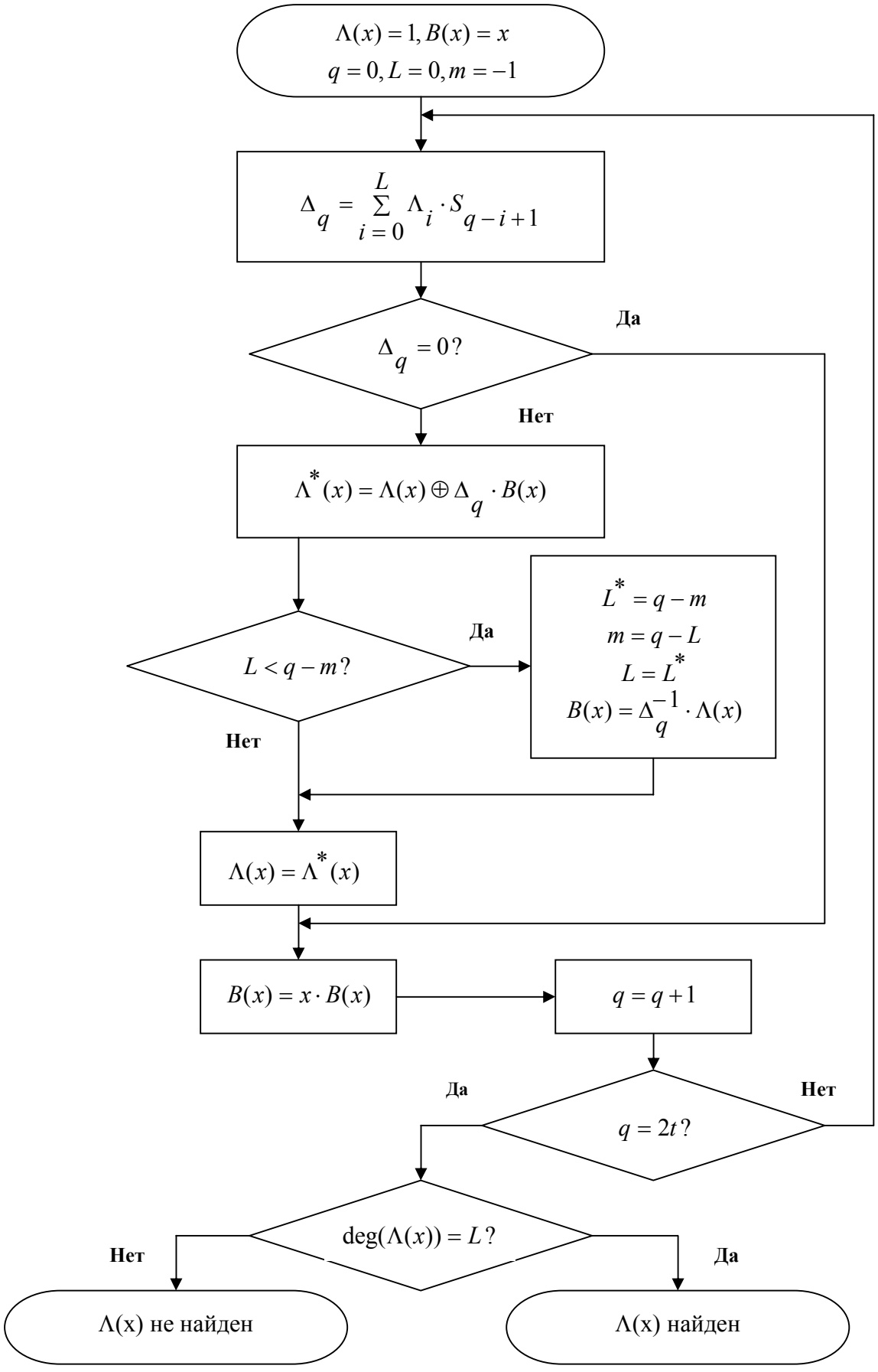

Рис. 5.1. Схема алгоритма Берлекэмпа-Месси для нахождения полинома локаторов.

В алгоритме Берлекэмпа-Месси используются следующие обозначения:

S_j – компоненты 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 $q+1-го$ уравнения, которая обращается в нуль, если полином локаторов удовлетворяет уравнению.

m – номер шага, на котором полином локаторов 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 истинное количество искаженных байт больше, чем кратность исправляемых ошибок, возможно (но необязательно), что степень полинома локаторов не будет совпадать с предполагаемым количеством искаженных байт, то есть $L \neq \deg(\Lambda(x))$, в этом случае считается, что решение системы уравнений не найдено. Этот случай следует трактовать, как невозможность исправления ошибок. Однако, если даже $L = \deg(\Lambda(x))$, это, как мы увидим далее, все равно не является стопроцентной гарантией того, что найден корректный полином локаторов.

В случае если полином локаторов $\Lambda(x)$ успешно найден, то тогда полином локаторов приравнивается нулю, и находят все корни $x*_1,\ldots,x*_\rho$ уравнения $\Lambda(x)=0$, где ρ - количество найденных корней. Делается это полным перебором всех ненулевых элементов поля Галуа. Поскольку конкретно в рассматриваемом нами поле Галуа $GF(2^8)$ объем перебора составляет всего $2^8 - 1$, то такой подход вполне оправдан.

Следует особо отметить, что количество найденных корней, в которых полином локаторов обращается в нуль, должно получиться строго равным степени полинома локаторов. В противном случае полином локаторов опять же следует считать некорректным, и считать, что истинное количество ошибок, очевидно, больше *t*.

Примечание 1. В случаях, когда истинное количество ошибок больше чем *t*, количество корней неверного полинома локаторов в большинстве случаев оказывается меньше, чем степень полинома локаторов, либо вообще не удается найти ни одного корня. Однако, следует иметь ввиду, что при небольших *t* (особенно при *t* = 1), велика вероятность того (чем меньше *t*, тем больше вероятность), что количество корней будет совпадать со степенью полинома локаторов, и полином локаторов будет считаться корректным с точки зрения декодирования, даже если на самом деле истинное количество ошибок больше *t*.

Иными словами, для возможности корректного декодирования, должно строго соблюдаться условие $\rho = \deg(\Lambda(x))$. Если это условие выполнено, тогда по определению полинома локаторов ошибок сами локаторы связаны с найденными корнями $x*_1,\ldots,x*_\tau$ простым соотношением $x*_l = 1/\alpha^{u_l}, l = 1\ldots\tau$, где $\tau = \deg(\Lambda(x))$.

По найденным корням легко вычислить локаторы ошибок:

$$x^*_l = 1/\alpha^{u_l} \Rightarrow u_l = \log_\alpha\left(1/x^*_l\right) \qquad (5.7)$$
$$l = 1\ldots\tau \quad \tau = \deg(\Lambda(x))$$

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

Примечание 2. Следует отметить, что локаторы ошибок вообще могут принимать значения только в пределах от 0 до 254, поскольку сами являются, по сути, логарифмами от элементов поля Галуа $GF(2^8)$. Следует также отметить, что если используется кадр максимально возможной длины равной 255, то проверка локаторов на допустимость становится бессмысленной, поскольку любой любое значение от 0 до 254 для локатора является допустимым при длине кадра 255. Впрочем, при длине кадра n < 255, но достаточно близкой к 255, велика вероятность того, что некорректный локатор «не попадет» на достаточно узкий «запрещенный» диапазон n…255. Поэтому, несмотря на выгодность кадров максимальной длины 255 с точки зрения скорости передачи данных, стоит учитывать потерю дополнительной возможности проверки локаторов на корректность.

В случае успешного вычисления локаторов $u_1,\ldots,u_\tau$ ошибок, остается последняя задача – вычисление значений ошибок. Поскольку мы имеет дело не с битами, с байтами в каждой позиции кадра, то значения байта может исказиться 255 различными способами и величину искажения необходимо выяснить для каждого найденного локатора ошибок.

В теории кодирования для вычисления величин ошибок используется, так называемый, метод Форни [6-14].

Введем понятие, так называемого, **полинома величин ошибок**, связав его с полиномом синдрома $S(x)$ и полиномом локаторов ошибок $\Lambda(x)$ следующим образом:

$$\Omega(x) = (S(x)\cdot\Lambda(x)) \bmod x^r \qquad (5.8.1)$$

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

Теперь проанализируем полином величин ошибок более детально, используя мультипликативную форму представления полинома локаторов $\Lambda(x) = \prod_{i=1}^{\tau}\left(1 \oplus x\cdot\alpha^{u_i}\right)$:

$$\Omega(x) = (S(x)\cdot\Lambda(x)) \bmod x^r = \left(\left(\sum_{j=0}^{r-1} x^j \cdot \left(\sum_{q=1}^{\tau} v_q \cdot \beta^{u_q} \cdot \left(\alpha^{u_q}\right)^{j+1}\right)\right) \cdot \prod_{i=1}^{\tau}\left(1 \oplus x\cdot\alpha^{u_i}\right)\right) \bmod x^r$$

$$= \left(\left(\sum_{q=1}^{\tau} v_q \cdot \beta^{u_q} \cdot \alpha^{u_q} \cdot \left(\sum_{j=0}^{r-1} x^j \cdot \left(\alpha^{u_q}\right)^{j}\right)\right) \cdot \prod_{i=1}^{\tau}\left(1 \oplus x\cdot\alpha^{u_i}\right)\right) \bmod x^r .$$

Теперь прибегнем к следующей уловке: из произведений по $i=1\ldots\tau$, вытащим отдельно множитель при $i=q$ и внесем его внутрь суммирования по $q=1\ldots\tau$.

$$\Omega(x)=\left(\left(\sum_{q=1}^{\tau} v_q\cdot\beta^{u_q}\cdot\alpha^{u_q}\cdot\left(\left(1\oplus x\cdot\alpha^{u_q}\right)\cdot\sum_{j=0}^{r-1}\left(x\cdot\alpha^{u_q}\right)^j\right)\right)\cdot\prod_{i=1,i\neq q}^{\tau}\left(1\oplus x\cdot\alpha^{u_i}\right)\right)\bmod x^r$$

. Теперь воспользуемся тем, что $(1\oplus z)\cdot\sum_{j=0}^{r-1} z^j=(1\oplus z)\cdot\left(1\oplus z\oplus z^2\oplus\ldots\oplus z^{r-1}\right)=1\oplus z^r$.

Тогда: $\Omega(x)=\left(\left(\sum_{q=1}^{\tau} v_q\cdot\beta^{u_q}\cdot\alpha^{u_q}\cdot\left(1\oplus\left(x\cdot\alpha^{u_q}\right)^r\right)\right)\cdot\prod_{i=1,i\neq q}^{\tau}\left(1\oplus x\cdot\alpha^{u_i}\right)\right)\bmod x^r$. Далее учитывая, что $\left(1\oplus\left(x\cdot\alpha^{u_q}\right)^r\right)\bmod x^r=1\oplus\alpha^{u_q}\cdot\left(x^r\bmod x^r\right)=1$, то в итоге получим:

$$\Omega(x)=\sum_{q=1}^{\tau} v_q\cdot\beta^{u_q}\cdot\alpha^{u_q}\cdot\prod_{i=1,i\neq q}^{\tau}\left(1\oplus x\cdot\alpha^{u_i}\right) \qquad (5.8.2)$$

Теперь определим значение полинома $\Omega(x)$ при $x=1/\alpha^{u_l}$. Нетрудно заметить, что при суммировании по $q=1\ldots\tau$, внутреннее произведение не будет содержать нулевого множителя (при $i=l$), и произведение не будет обращаться в нуль только тогда, когда $q=l$, поскольку $i\neq q=l$, и в произведении мы избегаем нулевого множителя. Таким образом, только слагаемое при $q=l$ в суммировании будет ненулевым. В итоге имеем:

$$\Omega\left(1/\alpha^{u_l}\right)=v_l\cdot\beta^{u_l}\cdot\alpha^{u_l}\cdot\prod_{i=1,i\neq l}^{\tau}\left(1\oplus x\cdot\alpha^{u_i}\right) \qquad (5.8.3)$$

Таким образом, нам, наконец, удалось выделить в явном виде величину ошибку v_l. Попробуем теперь понять суть множителя, стоящего рядом с величиной ошибки.

Найдем **формальную производную полинома локаторов ошибок** – $\Lambda'(x)$. При этом учтем, что формальная производная от произведения функций вычисляется как $\frac{d}{dx}\left(\prod_{i=1}^{\tau} f_i(x)\right)=\sum_{q=1}^{\tau}\frac{df_q(x)}{dx}\cdot\left(\prod_{i=1,i\neq q}^{\tau} f_i(x)\right)$. Тогда: $\frac{d}{dx}\Lambda(x)=\frac{d}{dx}\prod_{i=1}^{\tau}\left(1\oplus x\cdot\alpha^{u_i}\right)\Rightarrow$

$$\Lambda'(x)=\sum_{q=1}^{\tau}\alpha^{u_q}\cdot\left(\prod_{i=1,i\neq q}^{\tau}\left(1\oplus x\cdot\alpha^{u_i}\right)\right) \qquad (5.9.1)$$

Теперь определим значение формальной производной $\Lambda'(x)$ при $x=1/\alpha^{u_l}$. Анализируя формулу производной, видим, что при $x=1/\alpha^{u_l}$, как и в случае с полиномом величин ошибок, только при $q=l$ слагаемое в суммировании будет ненулевым. В итоге получим следующее выражение:

$$\Lambda'\left(1/\alpha^{u_l}\right)=\alpha^{u_l}\cdot\prod_{i=1,i\neq l}^{\tau}\left(1\oplus x\cdot\alpha^{u_i}\right) \qquad (5.9.2)$$

Сравнивая полученное выражение с аналогичным выражением, полученным выше для вычисления полинома величин ошибок $\Omega(x)$ при $x = 1/\alpha^{u_l}$, легко замечаем, что $\Omega\left(1/\alpha^{u_l}\right) = v_l \cdot \beta^{u_l} \cdot \Lambda'\left(1/\alpha^{u_l}\right)$. Тогда, очевидно, что величины ошибок $v_1,\ldots,v_\tau$ в позициях, указываемых локаторами $u_1,\ldots,u_\tau$, вычисляются, как отношение значения полинома $\Omega(x)$ величин ошибок к значению формальной производной $\Lambda'(x)$ полинома локаторов ошибок в соответствующих корнях $x^*_l = 1/\alpha^{u_l}, l = 1\ldots\tau$ уравнения $\Lambda(x) = 0$:

$$v_l = \Omega\left(1/\alpha^{u_l}\right) \Big/ \left(\beta^{u_l} \cdot \Lambda'\left(1/\alpha^{u_l}\right)\right) \qquad (5.10)$$
$$l = 1\ldots\tau \qquad \tau = \deg(\Lambda(x))$$

После этого нетрудно сформировать предполагаемый полином ошибок по найденным локаторам ошибок $u_1,\ldots,u_\tau$ и вычисленным величинам ошибок $v_1,\ldots,v_\tau$:

$$\widetilde{E}(x) = \sum_{l=1}^{\tau} v_l \cdot x^{u_l} \qquad (5.11)$$

Наконец, остается только сложить полином кадра (принятого из сети или считанного с носителя) с полиномом ошибок и осуществить исправление искаженного полинома кадра:

$$\widetilde{F}(x) = C(x) \oplus \widetilde{E}(x) \qquad (5.12)$$

Наконец, в качестве дополнительной «страховки», можно проверить полученный исправленный полином кадра $\widetilde{F}(x)$, вычислив для него синдром, так же, как мы это делали для полинома кадра (принятого из сети или считанного с носителя):

$$\widetilde{S}_j = \widetilde{F}(\beta \cdot \alpha^j), j = 1\ldots r \qquad (5.13)$$

Если «проверочный синдром» равен нулю, то декодирование прошло корректно, а исправленный полином кадра $\widetilde{F}(x)$, в случае если истинная кратность ошибки не превышает *t*, однозначно является исходным полиномом кадра $F(x)$. В случае если истинная кратность больше *t*, то $\widetilde{F}(x)$, по крайней мере, будет являться если уж не исходным, то хотя бы другим «допустимым» кадром, в $k + r$-мерном пространстве кадров.

Ниже представлена схема алгоритма декодирования (рис. 5.2).

Примечание 3. Отметим, что при выводе формул для полинома величин ошибок $\Omega(x)$ и формальной производной $\Lambda'(x)$ полинома локаторов ошибок мы использовали мультипликативную форму полинома локаторов ошибок $\Lambda(x) = \prod_{i=1}^{\tau}\left(1 \oplus x \cdot \alpha^{u_i}\right)$, и это делалось специально, чтобы в явном виде выделить величины ошибок $v_1,\ldots,v_\tau$. Однако, при программной реализации выведенные формулы усложняют вычислительные процедуры. Поэтому выведем формулы (эквивалентные ранее выведенным) для полиномов $\Omega(x)$ и $\Lambda'(x)$, используя аддитивную форму полинома локаторов $\Lambda(x) = 1 \oplus \sum_{i=1}^{\tau} \Lambda_i x^i$, которая эквивалентна мультипликативной форме полинома локаторов $\Lambda(x)$ по определению при условии, что полином локаторов является корректным с точки зрения декодирования.

Выведем формулу для полинома величин ошибок: $\Omega(x) = (S(x) \cdot \Lambda(x)) \bmod x^r =$

$$= \left(\left(\sum_{j=0}^{r-1} S_{j+1} \cdot x^j \right) \cdot \left(\sum_{i=0}^{\tau} \Lambda_i \cdot x^i \right) \right) \bmod x^r = \left(\sum_{q=0}^{\tau+r-1} x^q \left(\sum_{\substack{i+j=q \\ i=0\ldots\tau \quad j=0\ldots r-1}} S_{j+1} \cdot \Lambda_i \right) \right) \bmod x^r .$$

Заметим, что при индексах $q = \tau \ldots r-1$, мы во внутреннем суммировании получаем выражения вида $\Lambda_0 S_{q+1} \oplus \Lambda_1 S_q \oplus \ldots \oplus \Lambda_\tau S_{q-\tau+1}$. Если, учесть, что $\Lambda_0 = 1$, то нетрудно заметить, что при $q = \tau \ldots r-1$, эти выражения являются не чем иным, как суммой левой и правой части соответствующих уравнений ключевой системы уравнений декодирования. Однако, коэффициенты полинома локаторов $\Lambda(x)$ сами являются решением этой системы и удовлетворяет всем уравнениям, а значит суммы левой и правой части уравнений, разумеется, дадут нуль. Таким образом, очевидно, что коэффициенты с индексами при $q = \tau \ldots r-1$ полинома-произведения $S(x) \cdot \Lambda(x)$ окажутся равными нулю. Далее, поскольку от полинома-произведения $S(x) \cdot \Lambda(x)$ берется остаток по модулю x^r, то все коэффициенты с индексами $q \geq r$ результирующего полинома обнуляются, и тогда в итоге остается полином величин ошибок $\Omega(x)$ только лишь с коэффициентами с индексами $q = 0 \ldots \tau - 1$:

$$\Omega(x) = \sum_{q=0}^{\tau-1} x^q \cdot \sum_{\substack{i+j=q \\ i=0\ldots\tau \quad j=0\ldots r-1}} S_{j+1} \cdot \Lambda_i \qquad (5.14.1)$$

Обратим внимание на то, что $q = 0 \ldots \tau - 1$, а индексы i и j связаны жестким уравнением $i + j = q$. Тогда легко заметить, что даже при максимальном значении индекса $q = \tau - 1$, индекс $i \leq \tau - 1$, так как $j \geq 0$, и наоборот $j \leq \tau - 1$, так как $i \geq 0$. Таким образом, границы для индексов сужаются: $i = 0 \ldots \tau - 1$ и $j = 0 \ldots \tau - 1$. Но в таком случае, так как сумма индексов $i + j = q$ заключена в те же границы $q = 0 \ldots \tau - 1$, мы можем избавиться от индекса j, выразив его через i: $j = q - i$, и заключив индекс i в границы $i = 0 \ldots q$. В итоге:

$$\Omega(x) = \sum_{q=0}^{\tau-1} x^q \cdot \sum_{i=0}^{q} \Lambda_i \cdot S_{q-i+1} = \Lambda_0 \cdot S_1 \oplus \ldots \oplus (\Lambda_0 \cdot S_\tau \oplus \ldots \oplus \Lambda_{\tau-1} \cdot S_1) \cdot x^{\tau-1} \qquad (5.14.2)$$

Нетрудно заметить, что степень полинома величин ошибок $\Omega(x)$ равна $\tau - 1$:

$$\deg(\Omega(x)) = \tau - 1 \qquad (5.14.3)$$

Аналогично выведем формулу для формальной производной полинома локаторов ошибок – $\Lambda'(x)$, используя аддитивную форму полинома локаторов ошибок.

$$\Lambda'(x) = \frac{d}{dx}\Lambda(x) = \frac{d}{dx}\left(\sum_{i=0}^{\tau} \Lambda_i \cdot x^i \right) = \sum_{i=1}^{\tau} ((\Lambda_i \cdot (i \bmod 2)) \cdot x^{i-1}) \qquad (5.15.1)$$

Формальную производную полинома локаторов можно переписать в более удобном виде, используя другую индексную переменную $j = i - 1$:

$$\Lambda'(x) = \sum_{j=0}^{\tau-1} x^j \cdot (\Lambda_{j+1} \cdot ((j+1) \bmod 2)) = \Lambda_1 \oplus \Lambda_3 x^2 \oplus \ldots \oplus \begin{cases} \Lambda_\tau x^{\tau-1}, \tau \bmod 2 = 1 \\ \Lambda_{\tau-1} x^{\tau-2}, \tau \bmod 2 = 0 \end{cases} \qquad (5.15.2)$$

Заметим, что степень формальной производной полинома локаторов ошибок равна $\tau - 1$, если τ нечетно, и $\tau - 2$, когда τ четно. Одной формулой это можно выразить так:

$$\deg(\Lambda'(x)) = \tau - 1 - ((\tau - 1) \bmod 2) \qquad (5.15.3)$$

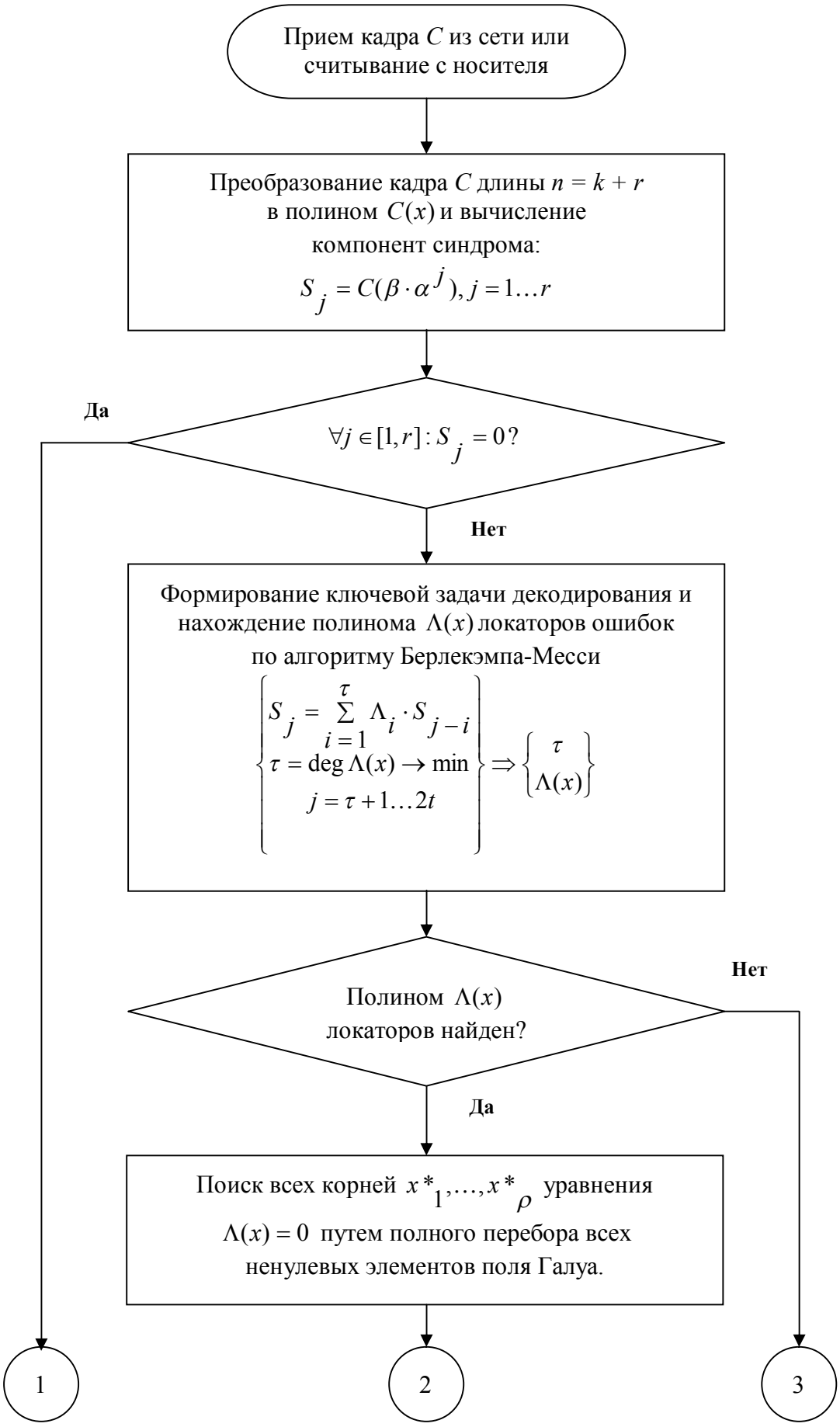

Рис. 5.2. Схема алгоритма декодирования кода Рида-Соломона (начало).

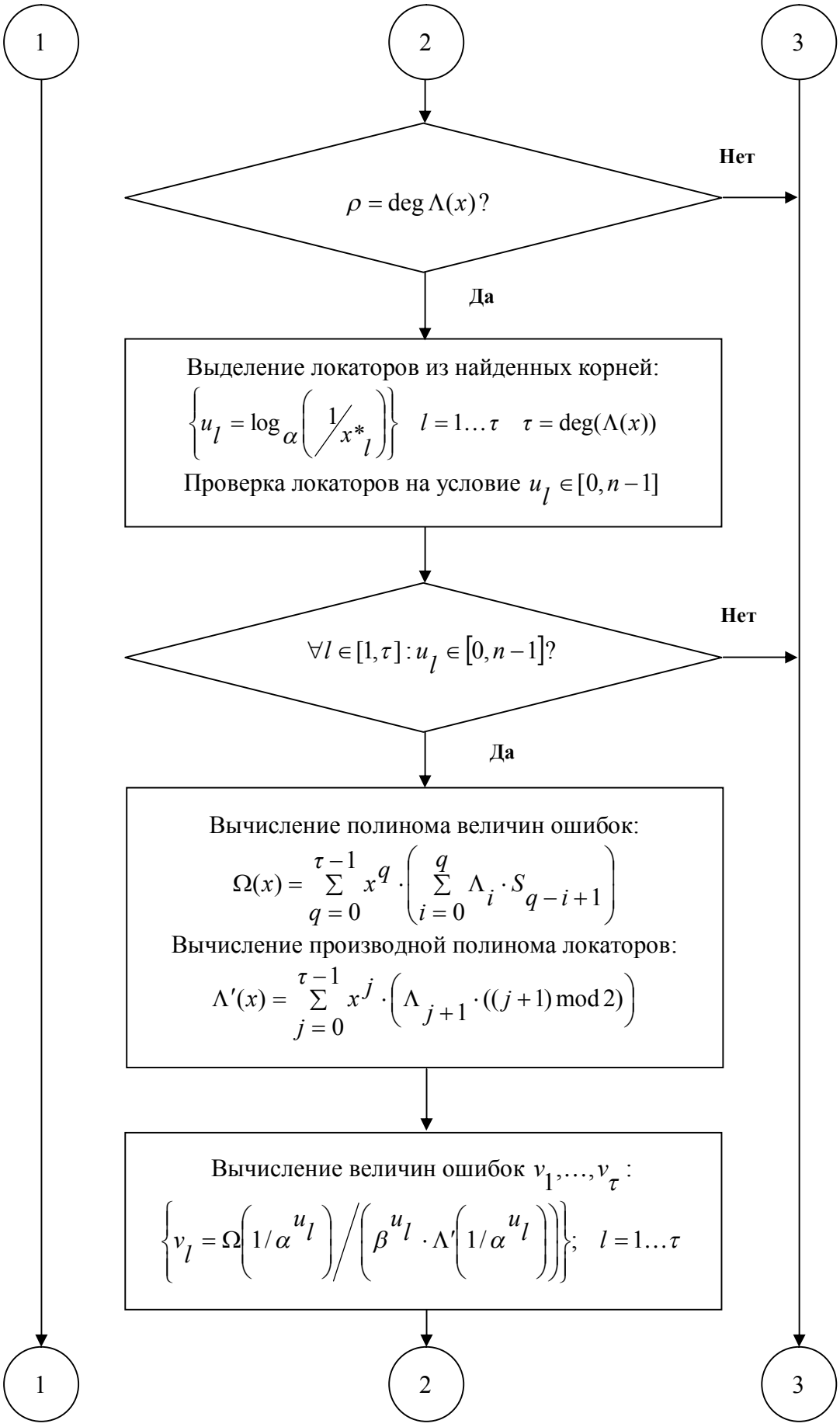

Рис. 5.2. Схема алгоритма декодирования кода Рида-Соломона (продолжение).

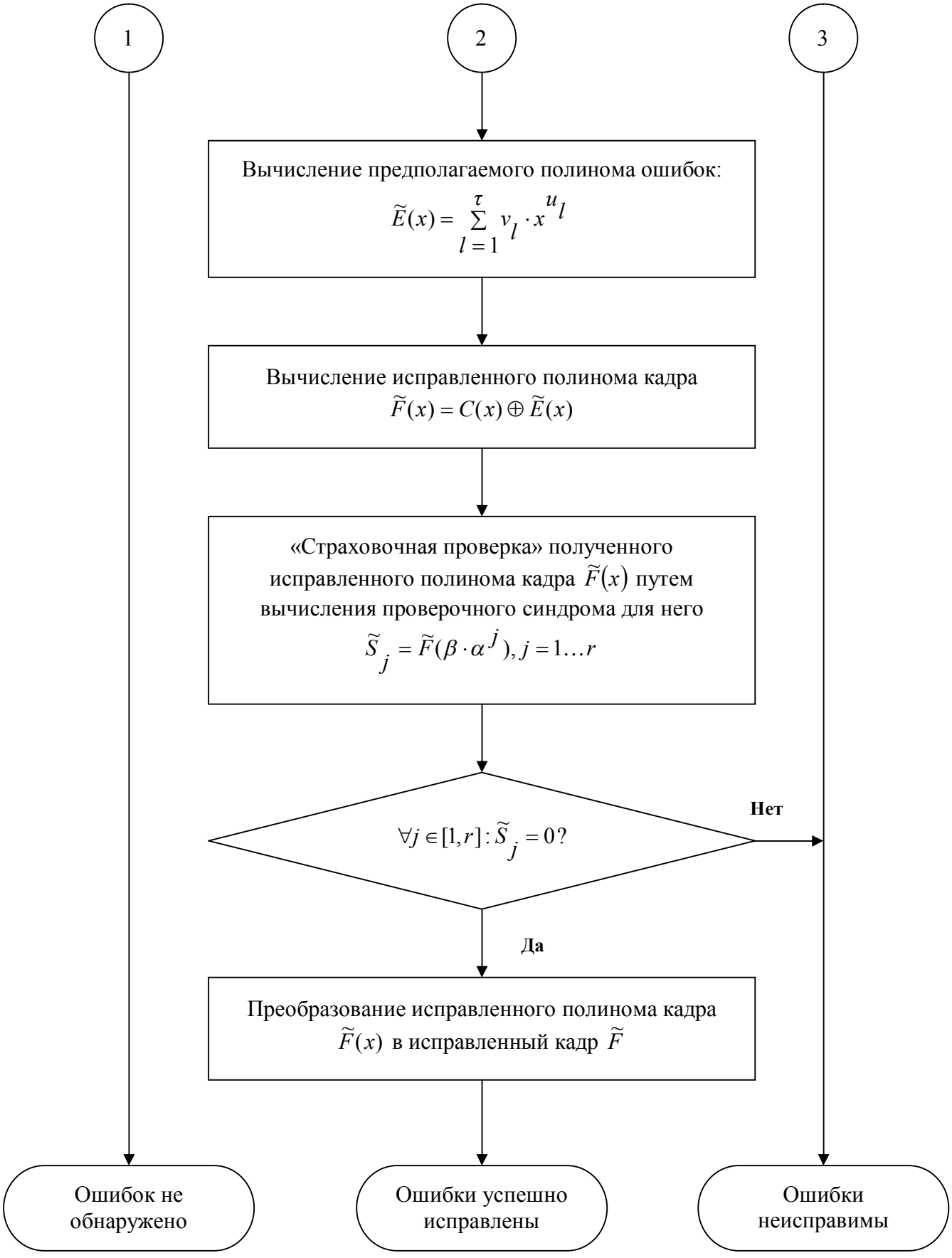

Рис. 5.2. Схема алгоритма декодирования кода Рида-Соломона (окончание).

Проанализируем «основные варианты развития событий» при декодировании кадра для 5 случаев, которые мы рассматривали в разделе кодирования, учитывая, что $r = 2 \cdot t$:

1) Искажения вообще отсутствует, и кадр C находится в исходном неискаженном виде, иными словами $C = F$. В этом случае, очевидно, декодер получит нулевой синдром для кадра C, и примет **верное решение об отсутствии искажений** в кадре C.
2) Искажено не более t байтов, и получен искаженный кадр $C = X$, который находится на расстоянии не более t от исходного кадра F. Синдром для кадра C, очевидно, будет ненулевым, и декодер примет верное решение о наличии искажений в кадре, и о необходимости его исправления. При исправлении допущение декодера о том, что искажено не более t байтов также будет совершенно верным. Соответственно, декодер найдет истинный полином локаторов, найдет его корни, с помощью которых вычислит соответствующие локаторы и величины ошибок, и со стопроцентной гарантией произведет **восстановление исходного кадра** $\widetilde{F} = F$.
3) Искажено более t байтов (от $t + 1$ до n), причем искаженный кадр $C = Y$ находится на расстоянии более t до любого «допустимого» кадра в n-мерном пространстве кадров. Синдром для кадра C, очевидно, будет ненулевым, и декодер примет верное решение о наличии ошибок в кадре, и о необходимости его исправления. Поскольку кадр C находится на расстоянии более t до любого «допустимого» кадра, то возможны три ключевые ситуации (они же – косвенные признаки «неисправимости» кадра):
   - Не удается найти полином локаторов по алгоритму Берлекэмпа-Месси.
   - Число найденных корней для полинома локаторов не совпадает с его степенью.
   - Вычисленные локаторы ошибок указывают за пределы границ кадра: $[0, n - 1]$.

   В таких случаях декодер принимает **верное решение о неисправимых искажениях**.
4) Искажено более t байтов (от $t + 1$ до n), причем искаженный кадр $C = Z$ находится на расстоянии не более t от некоторого «допустимого» кадра U, который, очевидно, не является исходным кадром F. Синдром для кадра C, очевидно, будет ненулевым, и декодер примет верное решение о наличии ошибок в кадре, и о необходимости его исправления. Допущение декодера о том, что искажено не более t байтов окажется неверным, но в силу того, что кадр C находится на расстоянии не более t от ближайшего «допустимого» кадра U, декодер успешно найдет «ложный» полином локаторов (ложный по сути, но корректный с точки зрения алгоритма декодирования). Далее декодер найдет корни для полинома локаторов, причем их количество совпадет с его степенью. Далее декодер вычислит локаторы ошибок (которые также будут ложными по сути, но корректными с точки зрения алгоритма декодирования, и будут указывать на позиции в пределах границ кадра), а также величины ошибок, и осуществит исправление. В результате получится кадр $\widetilde{F} = U$, который является «допустимым», но не исходным. Таким образом, будет осуществлено **смежное исправление**. Это также можно назвать **частичным обманом декодера**, поскольку факт искажения каналу передачи данных, носителю информации, или злоумышленнику скрыть не удалось, но, по крайней мере, удалось «увести» декодер в сторону «ложного» допустимого кадра U.
5) Искажено более $2 \cdot t$ байтов (от $2 \cdot t + 1$ до n), причем искаженный кадр C совпадает с некоторым «допустимым» кадром U, не являющимся исходным кадром F. Очевидно, что синдром для кадра $C = U$, будет равным нулю, поскольку кадр U является «допустимым». Соответственно, декодер примет неверное решение об отсутствии ошибок в кадре C. Таким образом, будет осуществлено **маскирование искажения**. Это также можно назвать **полным обманом декодера**, поскольку каналу передачи данных, носителю информации, или злоумышленнику удалось скрыть сам факт искажения и ввести декодер в полное заблуждение.

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:

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

Допустим, мы хотим использовать коды Рида-Соломона для исправления до $t = 2$ искаженных байтов. В таком случае нам понадобится $r = 2 \cdot t = 4$ избыточных байтов. Для кодирования будем использовать поле Галуа $GF(2^8)$ с неприводимым многочленом $p(x) = x^8 \oplus x^4 \oplus x^3 \oplus x^2 \oplus 1$ и примитивным элементом $\alpha = 2$. Параметр b выберем = 1, тогда и $\beta = 1$. Тогда порождающий полином будет следующим:

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

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

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

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

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 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** – и это слово вообще-то имеет противоположный смысл.

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

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

Вычислим компоненты синдрома:

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

Таким образом, $S_1 = 246, S_2 = 207, S_3 = 255, S_4 = 213$, очевидно, что синдром ненулевой, 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 $B(x) = x \cdot B(x) = 45 \cdot x^3 \oplus 28 \cdot x^2 \oplus 0 \cdot x \oplus 0$. После этого, поскольку $(q = q + 1 = 4) = (2t = 4)$, 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 ненулевых элементов поля Галуа 1,…,255 и, подставляя каждое из них в полином локаторов, находим корни, в которых полином локаторов обращается в нуль. Этими корнями, являются элементы $x_1^* = 131$ и $x_2^* = 54$. Заметим, что количество корней, равное 2, совпадает со степенью полинома локаторов ошибок, равного 2, так что можно продолжать декодирование.

Примечание. Мы легко можем перепроверить корни, подставив их в полином локаторов: $\Lambda(x) = 19 \cdot (131)^2 \oplus 93 \cdot (131) \oplus 1 = 0$ и $\Lambda(x) = 19 \cdot (54)^2 \oplus 93 \cdot (54) \oplus 1 = 0$.

После этого легко получаем искомые локаторы ошибок, используя логарифм по основанию примитивного элемента $\alpha = 2$:

$$\begin{cases} u_1 = \log_2(1/131) = 8 \\ u_2 = \log_2(1/54) = 6 \end{cases}$$

Локаторы, равные 8 и 6, не выходят за пределы [0, 8] кадра длиной n = 9, так что локаторы считаются корректными, и можно продолжать декодирование.

Теперь вычислим полином величин ошибок: $\Omega(x) = \sum_{q=0}^{1} x^q \cdot \left(\sum_{i=0}^{q} \Lambda_i \cdot S_{q-i+1} \right) =$

$$= \left(\Lambda_0 \cdot S_1 \right) \oplus x \cdot \left(\Lambda_0 \cdot S_2 \oplus \Lambda_1 \cdot S_1 \right) = 1 \cdot 246 \oplus x \cdot (1 \cdot 207 \oplus 93 \cdot 246) = 181 \cdot x \oplus 246.$$

Также вычислим формальную производную полинома локатора:

$$\Lambda'(x) = \sum_{j=0}^{1} x^j \cdot \left(\Lambda_{j+1} \cdot ((j+1) \bmod 2) \right) = \Lambda_1 \cdot (1 \bmod 2) \oplus x \cdot \Lambda_2 \cdot (2 \bmod 2) = \Lambda_1 = 93.$$

Наконец, можно вычислить величины ошибок, используя соответствующие локаторы (учтем также, что $\beta = 1$, а значит $\beta^{u_1} = \beta^{u_2} = 1$):

$$\begin{cases} v_1 = \Omega(1/2^8) \big/ \Lambda'(1/2^8) = (181 \cdot (131) \oplus 246) / (1 \cdot 93) = 30 \\ v_2 = \Omega(1/2^6) \big/ \Lambda'(1/2^6) = (181 \cdot (54) \oplus 246) / (1 \cdot 93) = 6 \end{cases}$$

Тогда, окончательно, получаем предполагаемый полином ошибок:

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

Остается только сложить полином искаженного кадра с полиномом ошибок:

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

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, имело место быть успешное восстановление исходного кадра.

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 полем Галуа и полем логарифмов.

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.

При $t=1$, избыточность составляет всего $r=2$ байта. Ограничимся рассмотрением случая поля Галуа $GF(2^8)$ с неприводимым многочленом $p(x)=x^8\oplus x^4\oplus x^3\oplus x^2\oplus 1$ и примитивным элементом $\alpha=2$. Параметр b принимается равным 1, соответственно $\beta=1$. Тогда порождающий полином получается следующим: $g(x)=\prod_{s=1}^{2}\left(x\oplus 2^s\right)=x^2\oplus 6\cdot x\oplus 8$.

Кодирование сообщения M длиной k байтов, дает кадр длиной $n=k+2$ 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 компонента синдрома $S_1=C(2^1)$ и $S_2=C(2^2)$. Если оба компонента 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:

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

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 $j=2$: $S_2=\sum_{i=1}^{\tau}\Lambda_i\cdot S_{2-i}$. Кроме того, у нас всего два компонента синдрома S_1 и S_2, и при $j=2$, имеем допустимое слагаемое $\Lambda_1\cdot S_1$, только при $i=1$. В итоге, ключевая система уравнений упрощается до вида $S_2=\Lambda_1\cdot S_1$, 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:

$$\Lambda(x)=\left(S_2/S_1\right)\cdot x\oplus 1 \qquad (5.16)$$

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

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

Нетрудно видеть, что полином локаторов имеет единственный корень $x^*=S_1/S_2$, откуда легко получаем сам локатор ошибки, используя логарифм по основанию примитивного элемента $\alpha=2$:

$$u=\log_2\left(1/x^*\right)=\log_2(S_2/S_1) \tag{5.17}$$

Выполним следующие преобразования полученной формулы:

$$u=\log_2\left(2^{\left(\log_2 S_2+(255-\log_2 S_1)\right)\bmod 255}\right)=\left(\log_2 S_2-\log_2 S_1+255\right)\bmod 255.$$

Учтем, что вычисление логарифма по основанию примитивного элемента 2 можно выполнять, просто считывая соответствующий логарифм из таблицы логарифмов. Тогда вычисление локатора ошибки можно свести к считыванию логарифмов $\log_2 S_2$ и $\log_2 S_1$ из таблицы логарифмов, вычислению разности между ними, добавления 255 к разности и, наконец, вычисления остатка по модулю 255. В итоге получаем локатор ошибки.

Полученный локатор u 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, учитывая что $\tau=1$:

$$\Omega(x)=\sum_{q=0}^{0}x^q\cdot\left(\sum_{i=0}^{q}\Lambda_i\cdot S_{q-i+1}\right)=\Lambda_0\cdot S_1=S_1 \tag{5.18}$$

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

$$\Lambda'(x)=\sum_{j=0}^{0}x^j\cdot\left(\Lambda_{j+1}\cdot((j+1)\bmod 2)\right)=\Lambda_1\cdot(1\bmod 2)=\Lambda_1=S_2/S_1 \tag{5.19}$$

Наконец, можно вычислить величину ошибки, учитывая, что полином величин и формальная производная полинома локаторов – константные функции (не зависят от x), а также учитывая, что $\beta=1$:

$$v=\Omega(x^*)/(1\cdot\Lambda'(x^*))=S_1/(S_2/S_1)=S_1^2/S_2 \tag{5.20}$$

Выполним следующие преобразования полученной формулы:

$$v=S_1^2/S_2=(S_1\cdot S_1)/S_2=2^{\left(\left(\log_2 S_1+\log_2 S_1\right)\bmod 255\right)}\Big/S_2=$$

$$=2^{\left(\left(2\cdot\log_2 S_1\right)\bmod 255+(255-\log_2 S_2)\right)\bmod 255}=2^{\left(2\cdot\log_2 S_1-\log_2 S_2+255\right)\bmod 255}.$$

Таким образом, вычисление величины ошибки можно свести ряду простых операций: из таблицы логарифмов считываются логарифмы $\log_2 S_1$ и $\log_2 S_2$, далее логарифм $\log_2 S_1$ удваивается, после этого из него вычитается логарифм $\log_2 S_2$, далее к разности добавляется 255 и от полученного результата берется остаток по модулю 255. После этого примитивный элемент 2 возводится в полученную степень, это делается путем считывания соответствующей степени примитивного элемента поля Галуа $GF(2^8)$ из таблицы степеней.

Контрольные вопросы

1. Что называют синдромом ошибок и что он характеризует?

2. Для чего вводится полином локаторов ошибок и как его коэффициенты связаны с компонентами синдрома ошибок?

3. В чем суть алгоритма Берлекэмпа-Месси? Какое основное его преимущество по сравнению с другими алгоритмами нахождения полинома локаторов ошибок?

4. Для чего требуется приравнивать полином локаторов ошибок нулю и искать корни этого уравнения? Чему должно быть равно количество корней в случае исправимого искажения? Как эти корни связаны с локаторами ошибок?

5. Каким образом находятся величины ошибок? В чем суть метода Форни?

6. По каким косвенным признакам при декодировании можно судить о том, что имело место быть неисправимое искажение, и декодирование ошибок невозможно?

7. Выполните декодирование принятого из сети искаженного полинома-кадра $C(x) = 83 \cdot x^8 \oplus 111 \cdot x^7 \oplus 117 \cdot x^6 \oplus 116 \cdot x^5 \oplus 104 \cdot x^4 \oplus 222 \cdot x^3 \oplus 143 \cdot x^2 \oplus 56 \cdot x \oplus 221$ заданного над полем Галуа $GF(2^8)$ с неприводимым многочленом $p(x) = x^8 \oplus x^4 \oplus x^3 \oplus x^2 \oplus 1$ и примитивным элементом $\alpha = 2$, при условии, что изначально кадр был сформирован при помощи порождающего полинома $g(x) = x^4 \oplus 30 \cdot x^3 \oplus 216 \cdot x^2 \oplus 231 \cdot x \oplus 116$, при параметре $b = 1$:
   - Вычислите синдромы ошибок.
   - Найдите полином локаторов ошибок $\Lambda(x)$ и предполагаемое число ошибок.
   - Найдите корни уравнения $\Lambda(x) = 0$ и вычислите локаторы ошибок.
   - Вычислите величины ошибок и сформируйте полином искажений.
   - Выполните восстановление искаженного полинома-кадра. Выделите из восстановленного полинома-кадра информационное сообщение, и сравните его с тем, которое было в искаженном полиноме-кадре.

6. Вероятностный анализ искажений.

Базовый анализ вероятностей искажений

Информация может искажаться либо при передаче по каналам передачи данных, либо при хранении на каком-либо носителе информации (жесткий диск, оптический диск и т.п.).

Для каналов передачи данных, как правило, известна вероятность искажения бита. Данные передаются в виде последовательности битов, и каждый из них может подвергнуться искажению. Здесь мы сделаем несколько важных допущений (ради упрощения анализа):

- Вероятность искажения того или иного бита в том или ином байте в канале передачи данных одна и та же, и не зависит от позиции байта в кадре и бита внутри байта.
- Канал передачи данных не обладает «памятью», и вероятность искажения очередного передаваемого бита не зависит от того, были ли искажены предыдущие биты.
- Вероятность искажения бита не меняется со временем или меняется достаточно медленно, и в пределах отрезка времени, требуемого для передачи кадра, вероятность искажения бита можно считать постоянной.

При соблюдении вышеперечисленных условий, передачу кадра длины n можно считать последовательностью из n независимых испытаний по передаче отдельных байтов, в свою очередь, в рамках передачи отдельного байта мы имеем дело с последовательностью из 8 независимых элементарных испытаний по передаче отдельных битов. В итоге мы имеем дело с последовательностью из $8 \cdot n$ элементарных независимых испытаний, которую мы можем также интерпретировать, как n независимых серий по 8 независимых элементарных испытаний в каждой серии. Особо отметим также, что в случае искажения нескольких битов, они 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 [16].

Пусть, p – заданная вероятность искажения бита.

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

$$P_{byte} = 1 - (1 - p)^8 \tag{6.1}$$

Тогда, согласно биномиальному закону распределения числа искаженных байтов, получаем вероятность того, что исказится ровно h байтов в кадре, состоящего из n байтов, при заданной вероятности p искажения бита:

$$P(T = h) = C_n^h \cdot \left(1 - (1 - p)^8\right)^h \cdot \left((1 - p)^8\right)^{(n-h)} \tag{6.2}$$

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

Следует особо отметить, что может возникнуть ложная иллюзия, что вероятность ошибок большей кратности (искажения большего числа байтов), вроде как меньше вероятности ошибок меньшей кратности (искажения меньшего числа байтов). В действительности же, все зависит от базовой вероятности искажения одного бита и количества байтов в кадре.

При большой 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. 6.1) зависимостей вероятности искажения ровно *h* байтов в кадре длиной *n* для некоторых значений вероятности *p* искажения бита.

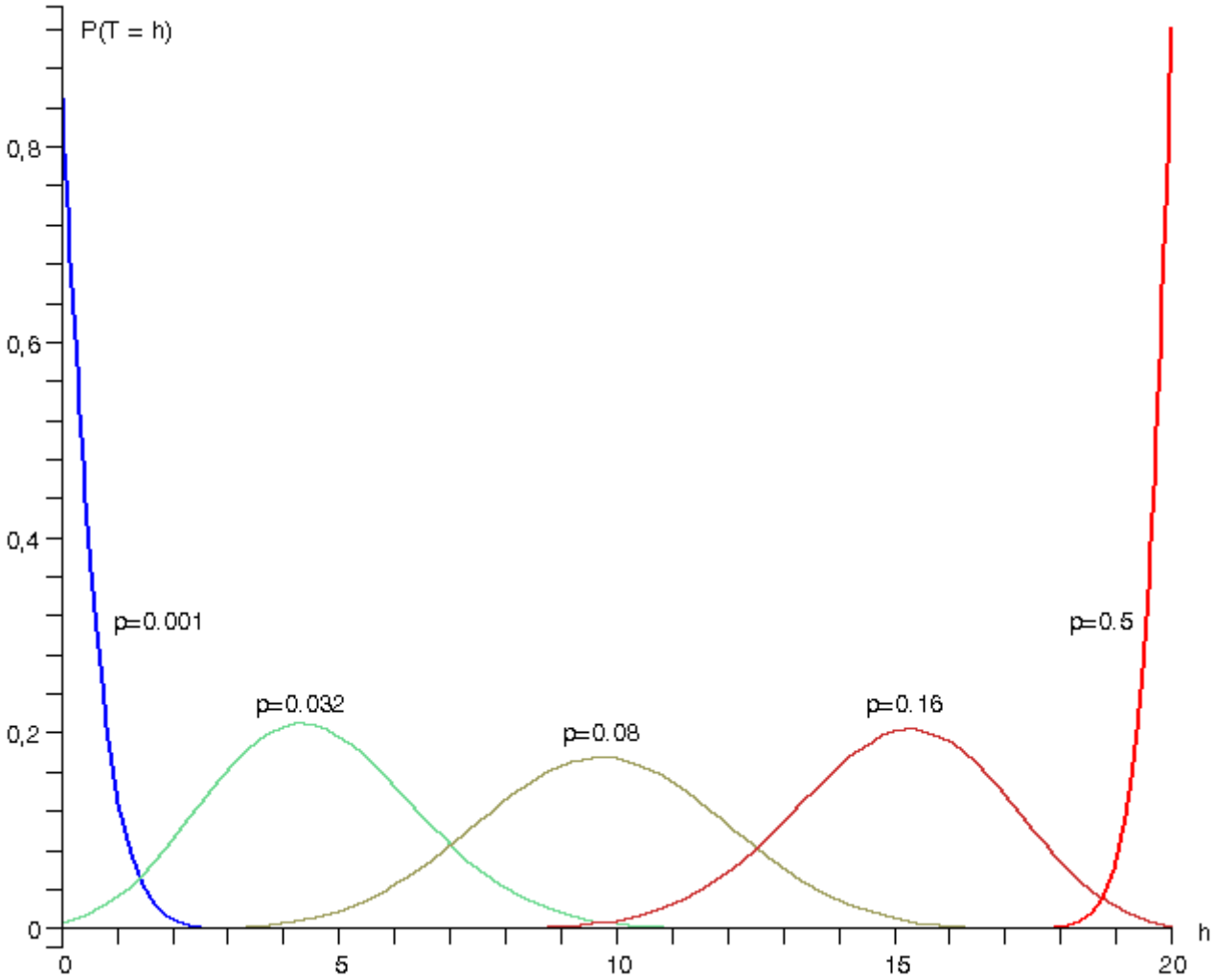

Рис. 6.1. Графики зависимостей вероятности искажения ровно *h* байтов в кадре длиной *n* = 20 байтов для некоторых значений вероятности *p* искажения бита.

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 конкретной кратности, вплоть до крайнего случая, когда наиболее вероятно искажение всех байтов. Математическое ожидание количества искаженных байтов составляет: $n \cdot (1-(1-p)^8)$.

Теперь, имея формулу для вероятности искажения ровно *h* байтов в кадре длиной *n* при заданной вероятности искажения бита *p*, нетрудно вывести формулы для вероятности отсутствия искажения ($T = 0$), вероятности восстанавливаемого искажения ($1 \le T \le t$) и вероятности невосстанавливаемого искажения ($T > t$).

Тогда, формула для вероятности отсутствия искажения:

$$P(T=0)=C_n^0\cdot\left(1-(1-p)^8\right)^0\cdot\left((1-p)^8\right)^{(n-0)}=(1-p)^{8n} \qquad (6.3.1)$$

Формула для вероятности восстанавливаемого искажения:

$$P(1\le T\le t)=\sum_{h=1}^{t}C_n^h\cdot\left(1-(1-p)^8\right)^h\cdot\left((1-p)^8\right)^{(n-h)} \qquad (6.3.2)$$

Наконец, формула для вероятности невосстанавливаемого искажения:

$$P(T>t)=\sum_{h=t+1}^{n}C_n^h\cdot\left(1-(1-p)^8\right)^h\cdot\left((1-p)^8\right)^{(n-h)} \qquad (6.3.3)$$

Ниже приведены графики (рис. 6.2) зависимостей вероятности искажения $T>t$ байтов в кадре длиной n байтов для некоторых значений вероятности p искажения бита.

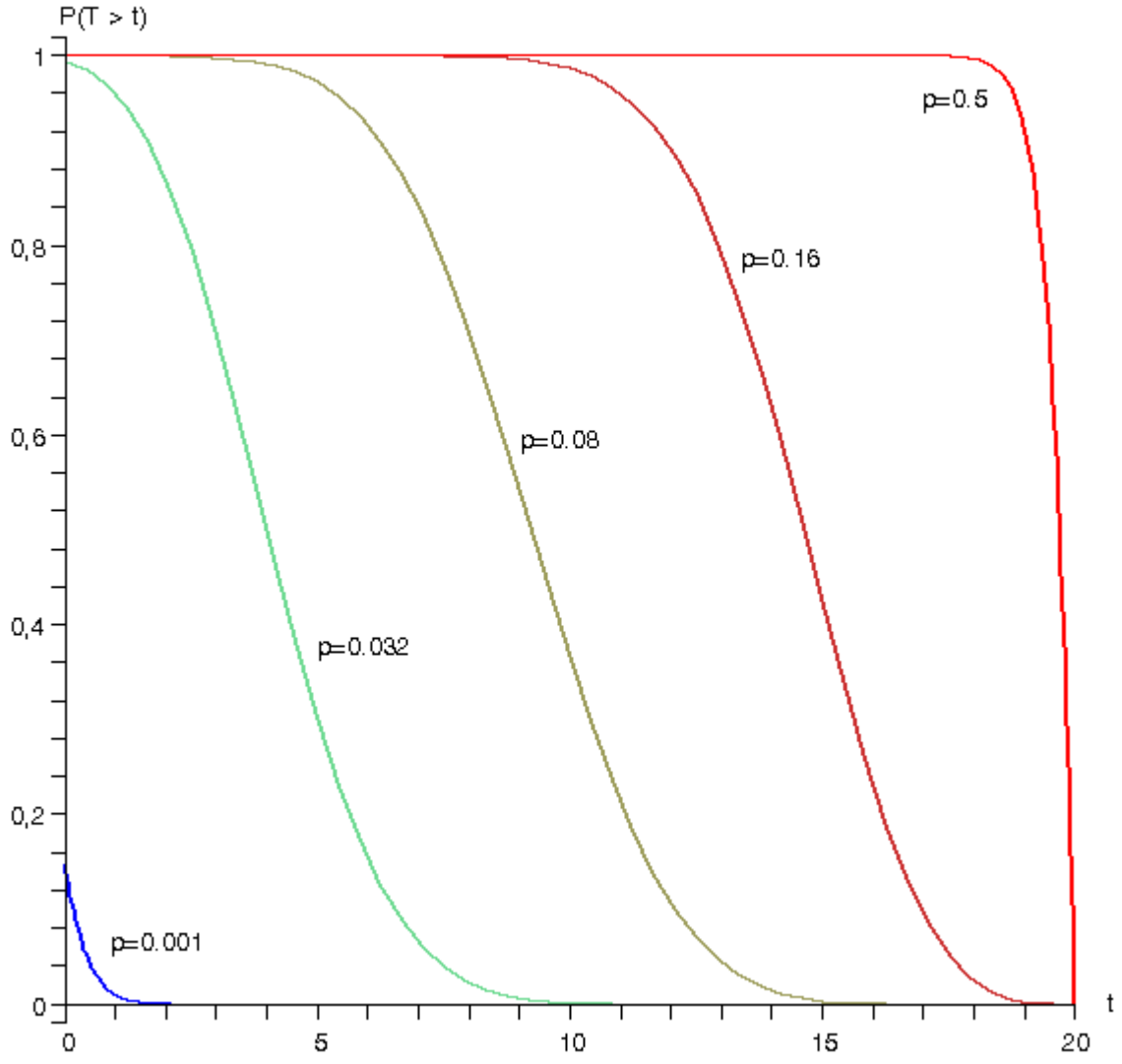

Рис. 6.2. Графики зависимостей вероятности искажения более *t* байтов в кадре длиной *n* = 20 байтов для некоторых значений вероятности *p* искажения бита.

Как видно из графиков, вероятность невосстанавливаемого искажения существенным образом определяется вероятностью искажения бита. При вероятности искажения бита близкой к 1/2, вероятность невосстанавливаемого искажения близка к единице даже при больших кратностях исправляемых ошибок.

Частный случай. Оценим теперь вероятность отсутствия искажения, вероятность восстанавливаемого искажения и вероятность невосстанавливаемого искажения при вероятности искажения бита $p = 1/2$. То есть любой бит может с шансами «50 на 50» либо исказиться, либо нет. Такая картина может наблюдаться в сильно зашумленных каналах передачи данных, а также на носителях информации, подвергшихся сильному разрушающему воздействию, например, взрыву большой мощности. При $p = 1/2$, искажения отдельно байта составляет: $P_{byte} = 1 - (1 - 1/2)^8 = 255/256$.

Тогда вероятность отсутствия искажений:

$$\lim_{p \to 1/2} P(T = 0) = 1/256^n \qquad (6.4.1)$$

Соответственно, вероятность восстанавливаемого искажения:

$$\lim_{p \to 1/2} P(1 \le T \le t) = \sum_{h=1}^{t} C_n^h \cdot (255/256)^h \cdot (1/256)^{(n-h)} = \frac{1}{256^n} \sum_{h=1}^{t} C_n^h \cdot 255^h \qquad (6.4.2)$$

Соответственно, вероятность невосстанавливаемого искажения:

$$\lim_{p \to 1/2} P(T > t) = \sum_{h=t+1}^{n} C_n^h \cdot (255/256)^h \cdot (1/256)^{(n-h)} = \frac{1}{256^n} \sum_{h=t+1}^{n} C_n^h \cdot 255^h \qquad (6.4.3)$$

Нетрудно заметить, что вероятность «невосстанавливаемого» искажения $P(T > t)$ при вероятности искажения бита $p = 1/2$, весьма существенная, больше $(255/256)^n$, даже при наибольшей кратности исправляемых ошибок $t = \lfloor (n-1)/2 \rfloor$ для заданной длины кадра *n*.

Анализ вероятностей особых случаев невосстанавливаемых искажений

Как мы определили, невосстанавливаемое искажение – это такое искажение, при котором искажается более *t* байтов ($T > t$), и восстановление исходного кадра – невозможно. Как мы установили ранее, вероятность невосстанавливаемого искажения кадра равна $P(T > t) = \sum_{h=t+1}^{n} C_n^h \cdot \left(1 - (1-p)^8\right)^h \cdot \left((1-p)^8\right)^{(n-h)}$. Однако, при декодировании кадров, у которых искажено более *t* байтов, возможны три особых случая:

- Кадр находится на расстоянии более *t* от всех возможных «допустимых» кадров и однозначное «притягивание» к какому-либо «допустимому» кадру невозможно. В этом случае декодер принимает правильное решение о «неисправимости» кадра.
- Кадр оказывается на расстоянии не более *t* от некоторого «допустимого» кадра *U*, который, разумеется, не является исходным, и декодер имеет возможность (и реализует это возможность) «притянуть» искаженный кадр к «допустимому» кадру *U*. Такой случай мы называем «смежным исправлением», а также «частичным обманом декодера» – факт искажения декодер обнаруживает, но исправление приводит к «смежному» кадру *U*, поскольку он на расстоянии не более *t* от искаженного кадра.
- Кадр полностью совпадает с некоторым «допустимым» кадром *U*, который, разумеется, не является исходным. Такое возможно только при искажении более $2 \cdot t$ байтов. В этом случае сам факт искажения остается необнаруженным, и мы это называем случаем «маскирования искажения», а также «полным обманом декодера».

Теперь обозначим вероятности трех случаев «невосстанавливаемых искажений»:

- $P_{н.и.}$ - вероятность «неисправимого искажения».
- $P_{с.и.}$ - вероятность «смежно-исправимого искажения».
- $P_{м.и.}$ - вероятность «маскируемого искажения».

Поскольку рассмотренные три случая, не имеют пересечений (несовместны), и при этом охватывают все случаи невосстанавливаемых искажений, то сумма вероятностей этих случаев, очевидно, равна вероятности невосстанавливаемого искажения:

$$P_{н.и.} + P_{с.и.} + P_{м.и.} = P(T > t) \qquad (6.5.1)$$

Из этого равенства также следует «верхняя оценка» для вероятности смежно-исправимого или маскируемого искажения:

$$P_{с.и.} + P_{м.и.} < P(T > t) \qquad (6.5.2)$$

Иными словами, вероятность смежно-исправимого или маскируемого искажения всегда меньше вероятности невосстанавливаемого искажения.

Кроме того, поскольку маскирование возможно только при искажении более $2{\cdot}t$ байтов, то мы также имеем «верхнюю оценку» для вероятности маскируемого искажения:

$$P_{м.и.} < P(T > 2t) \qquad (6.5.3)$$

Однако, «верхние оценки» – это, конечно, уже кое-что, но хотелось бы также иметь возможность точно вычислять вероятности для всех трех случаев.

Поскольку вероятность невосстанавливаемого искажения $P(T > t)$ нетрудно вычислить, то из трех вероятностей, достаточно знать любые две, чтобы получить оставшуюся неизвестную вероятность из равенства $P_{н.и.} + P_{с.и.} + P_{м.и.} = P(T > t)$. В качестве «третьей неизвестной» мы выберем $P_{н.и.}$ - вероятность неисправимого искажения. Тогда остается выяснить только $P_{с.и.}$ и $P_{м.и.}$.

Анализ $P_{с.и.}$ и $P_{м.и.}$ требует нетривиального комбинаторного анализа в пространстве кадров, поэтому прибегнем к наглядной геометрической картине (рис. 6.3).

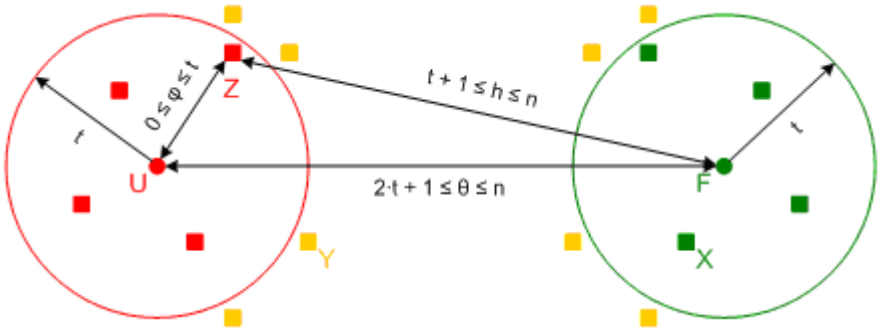

Рис. 6.3. Геометрическая картина смежно-исправимого или маскируемого искажения.

Итак, пусть имеется некоторый исходный кадр F. Рассмотрим множество кадров, которые находятся на расстоянии h от исходного кадра, причем $t+1 \le h \le n$, и при этом находятся на расстоянии φ от некоторого «допустимого» кадра U, причем $0 \le \varphi \le t$. При этом рассматриваемый «допустимый» кадр U находится на расстоянии θ от исходного кадра, причем $2t+1 \le \theta \le n$, поскольку ближе, чем на минимальном кодовом расстоянии $d = 2t+1$, «допустимых» кадров не может быть согласно теории кодирования. Смежное исправление возможно только тогда, когда $t+1 \le h \le n$ и $1 \le \varphi \le t$, поскольку только в этом случае искаженный кадр Z с одной стороны не совпадает с «допустимым» кадром U, и в то же время находится от него на расстоянии не более t, и декодер сможет «исправить» искаженный кадр, превратив его в «допустимый» кадр U. Что касается маскируемого искажения, то оно происходит только тогда, когда $2t+1 \le h \le n$ и $\varphi = 0$, поскольку в этом случае происходит полное совпадение искаженного кадра с «допустимым» кадром U.

Примечание. Особо отметим, что в *n*-мерном пространстве никакой кадр не может находиться на расстоянии более *n* от другого кадра. Поэтому в случае рассмотрения «допустимых» кадров *U*, находящихся на «приграничных» в n-мерном пространстве расстояниях $\theta > n - t$ от исходного кадра *F*, кадры *Z*, отстоящие на расстоянии $1 \le \varphi \le t$ от «допустимого» кадра *U*, находятся на расстояниях не более *n* от исходного кадра *F*, даже если $\varphi + \theta > n$. Однако, это вовсе не означает, что «шар» радиуса *t* с центром в кадре *U*, находящемся на расстоянии $\theta > n - t$ от исходного кадра *F*, содержит меньшее количество кадров, чем другой «шар» радиуса *t* с центром в кадре *U*, находящемся на расстоянии $\theta \le n - t$ от исходного кадра *F*. Просто «шары» радиуса *t* с центрами, находящимися на «приграничных» расстояниях $\theta > n - t$ от исходного кадра *F*, по форме это уже не совсем «шары», а более сложные *n*-мерные геометрические объекты. Тем не менее, такие объекты содержат точно такое же количество $\sum_{\varphi=0}^{t} C_n^{\varphi} \cdot 255^{\varphi}$ кадров, как и любой другой «шар» радиуса *t* с центром в кадре *U*, находящемся на расстоянии $\theta \le n - t$ от исходного кадра *F*.

Теперь попробуем вычислить количество искаженных кадров, находящихся на расстоянии h от исходного кадра, причем $t+1 \le h \le n$, и при этом находящихся на расстоянии φ от некоторого «допустимого» кадра *U*, причем $0 \le \varphi \le t$. Рассмотрим частный случай, когда исходным кадром *F* является нулевой кадр, состоящий из *n* байтов, содержащих нуль. Мы обращаемся к частному случаю только для того, чтобы сделать дальнейшие рассуждения более простыми и наглядными, однако, полученный результат также будет справедливым для любого другого исходного кадра.

Итак, пусть исходный кадр состоит *F* из *n* байтов, содержащих нуль. Тогда некоторый «допустимый» кадр *U*, находящийся на расстоянии θ от исходного кадра *F*, очевидно, содержит θ ненулевых байтов и $n - \theta$ нулевых байтов.

Рассмотрим теперь, какими способами можно «отклоняться» от «допустимого» кадра *U*. Во-первых, среди $n - \theta$ нулевых байтов можно выбрать q байтов ($C_{n-\theta}^{q}$ способами), причем $0 \le q \le n - \theta$, и заменить каждое из них на одно из 255 ненулевых значений. Во-вторых, среди θ ненулевых байтов можно выбрать i байтов (C_{θ}^{i} способами), причем $0 \le i \le \theta$, и заменить каждое из них на одно из 254 ненулевых значений (мы исключаем одно нулевое значение исходного кадра и одно ненулевое значение «допустимого» кадра в соответствующих байтах). В-третьих, в оставшихся $\theta - i$ ненулевых байтах можно выбрать j байтов ($C_{\theta-i}^{j}$ способами), причем $0 \le j \le \theta - i$, и заменить их нулевым значением. Теперь, учтем то, что подходящие для нас варианты «отклоненных» кадров, должны находиться на расстоянии ровно φ от «допустимого» кадра *U*. Поскольку мы «отклоняем» кадры от «допустимого» кадра *U*, путем изменения $q + i + j$ байтов значениями, отличными от значений в соответствующих байтах кадра *U*, то, очевидно, что только при условии $i + j + q = \varphi$ «отклоненный» кадр находится на расстоянии φ от кадра *U*. Также учтем, что подходящие для нас варианты «отклоненных» кадров, должны находиться на расстоянии ровно h от исходного кадра *F*, то есть содержать h ненулевых байтов и $n - h$ нулевых байтов. В качестве вариантов «отклонений», мы q нулевых байтов среди $n - \theta$ байтов заменяем ненулевыми значениями, затем i ненулевых байтов среди θ байтов заменяем другим ненулевым значением, наконец, среди оставшихся $\theta - i$ ненулевых j байтов заменяем нулями (при этом $\theta - i - j$ байтов остаются ненулевыми) – в итоге общее число ненулевых байтов равно $q + i + (\theta - i - j) = q + \theta - j$, и, очевидно, только при условии $q + \theta - j = h$, «отклоненный» кадр будет находиться на расстоянии h от исходного кадра *F*.

Ниже приведен простой наглядный пример «отклонения» некоторого «допустимого» кадра U в некоторый кадр Z при заданном исходном кадре F:

$$F=\left(\overbrace{0\ldots0}^{n}\right);\quad U=\left(\underbrace{\overbrace{0\ldots0}^{n-\theta}\overbrace{A\ldots A}^{\theta}}\right)\Rightarrow Z=\left(\underbrace{\overbrace{0\ldots0}^{n-\theta}\underbrace{A\ldots A}_{q}\overbrace{B\ldots B}^{i}\overbrace{A\ldots A}^{\theta-i}\underbrace{0\ldots0}_{j}}\right);\quad \begin{matrix} D(F,U)=\theta \\ D(U,Z)=q+i+j \\ D(F,Z)=q+\theta-j \end{matrix}$$

Тогда в итоге чтобы подсчитать общее количество (обозначим его $N_{h\theta\varphi}$) искаженных кадров, находящихся на расстоянии h от исходного кадра, и при этом находящихся на расстоянии φ от некоторого «допустимого» кадра U, находящегося на расстоянии θ от исходного кадра F, необходимо просуммировать все рассмотренные варианты «отклонений» от «допустимого» кадра при соблюдении двух рассмотренных условий. В итоге получаем следующую тройную сумму с двумя условиями для индексов:

$$N_{h\theta\varphi}=\sum_{q=0}^{n-\theta}\left(C_{n-\theta}^{q}\cdot 255^{q}\cdot\left(\sum_{i=0}^{\theta}C_{\theta}^{i}\cdot 254^{i}\cdot\left(\sum_{j=0}^{\theta-i}C_{\theta-i}^{j}\right)\right)\right) \qquad (6.6.1)$$

$$i+j+q=\varphi\quad q+\theta-j=h$$

В формуле ключевую роль играют именно два условия, которые связывают «внутренние» индексы q,i,j с «внешними» параметрами φ,θ,h. Нетрудно заметить, что если убрать эти два условия, то внутренняя сумма: $\sum_{j=0}^{\theta-i}C_{\theta-i}^{j}=2^{\theta-i}$, вторая сумма при этом станет: $\sum_{i=0}^{\theta}C_{\theta}^{i}\cdot 254^{i}\cdot 2^{\theta-i}=256^{\theta}$, а третья сумма: $\sum_{q=0}^{n-\theta}\left(C_{n-\theta}^{q}\cdot 255^{q}\right)=256^{n-\theta}$.

В итоге получим, что $N_{h\theta\varphi}=256^{n}$. То есть, само по себе суммирование производится по всем без исключения кадрам n-мерного пространства, и только два условия определяют то, какие кадры являются «подходящими» для нашего рассмотрения, а какие нет.

Заметим также, что если $\varphi=0$, то в силу условия $i+j+q=\varphi$ все индексы суммирования q,i,j смогут принимать только значение 0. Тогда, очевидно, $N_{h\theta(\varphi=0)}=1$. Это вполне логично – существует только один кадр, находящийся на расстоянии $\varphi=0$ от некоторого «допустимого» кадра U, находящегося на расстоянии θ от исходного кадра F, это сам кадр U. Кроме того, так как из $\varphi=0$ следует $i=j=q=0$, то второе условие, очевидно, становится $\theta=h$, что тоже логично, ведь рассматриваемый кадр совпадает с кадром кадр U, и он находится на том же самом расстоянии θ от исходного кадра F.

Полученная формула достаточно наглядная с точки зрения отражения сути рассматриваемой комбинаторной задачи, однако, достаточно громоздкая и трудоемкая с вычислительной точки зрения. Поэтому воспользуемся условиями для индексов и избавимся от суммирования по индексу q. Выразим q из второго условия и получим: $q=h+j-\theta$. Кроме того, подставляя $q=h+j-\theta$ в первое условие, получим преобразованное условие: $i+2j+h=\varphi+\theta$. Тогда получим следующую формулу для расчета $N_{h\theta\varphi}$:

$$N_{h\theta\varphi}=\sum_{i=0}^{\theta}C_{\theta}^{i}\cdot 254^{i}\cdot\left(\sum_{j=0}^{\theta-i}C_{\theta-i}^{j}\cdot C_{n-\theta}^{h+j-\theta}\cdot 255^{h+j-\theta}\right) \qquad (6.6.2)$$

$$i+2j+h=\varphi+\theta$$

Таким образом, мы выяснили количество искаженных кадров, находящихся на расстоянии h от исходного кадра *F*, и при этом также находящихся на расстоянии φ от некоторого «допустимого» кадра *U*, находящегося на расстоянии θ от исходного кадра *F*.

Теперь же оценим вероятность появления конкретной конфигурации искажения. Как было установлено ранее, вероятность того, что исказится ровно h байтов при заданной вероятности искажения бита, равной *p*, и размерности пространства кадров *n*, равна: $C_n^h \cdot \left(1-(1-p)^8\right)^h \cdot \left((1-p)^8\right)^{(n-h)}$. Тогда, вероятность того, что исказится конкретный набор из *h* байтов, причем в каждом из искаженных байтов будет реализован один из 255 вариантов искажения, будет равна $\left(\frac{1-(1-p)^8}{255}\right)^h \cdot \left((1-p)^8\right)^{(n-h)}$. Для того, чтобы формула была менее громоздкой, используем ранее полученную формулу вероятности искажения отдельного байта при заданной вероятности искажения отдельного бита $P_{byte} = 1-(1-p)^8$.

Тогда окончательно получаем: $\left(\frac{P_{byte}}{255}\right)^h \cdot \left(1-P_{byte}\right)^{(n-h)}$.

Тогда, мы, используя $N_{h\theta\varphi}$, можем оценить вероятность того, что искаженный кадр будет находиться на расстоянии φ (причем $0 \le \varphi \le t$) от некоторого «допустимого» кадра *U*, находящегося на расстоянии θ (причем $2t+1 \le \theta \le n$) от исходного кадра *F*. Для этого нам необходимо просто умножить полученную выше вероятность конкретной конфигурации искажения на количество интересующих нас искаженных кадров $N_{h\theta\varphi}$, и просуммировать по всем *h*, причем $t+1 \le h \le n$. Обозначим интересующую нас вероятность $V_{\theta\varphi}$ и получим:

$$V_{\theta\varphi} = \sum_{h=t+1}^{n} \left(\left(\frac{P_{byte}}{255}\right)^h \cdot \left(1-P_{byte}\right)^{(n-h)} \cdot N_{h\theta\varphi} \right) \qquad (6.7.1)$$

Заметим, что в формуле для расчета вероятности $V_{\theta\varphi}$ используется одинарное суммирование, в рамках которого производится двойное суммирование при вычислении $N_{h\theta\varphi}$ для каждого h, в итоге имеем тройное суммирование, что представляет большую трудоемкость с вычислительной точки зрения. Поэтому, попробуем подставить формулу для расчета $N_{h\theta\varphi}$ в формулу для расчета $V_{\theta\varphi}$, и выполнить ряд преобразований. Тогда имеем:

$$\sum_{h=t+1}^{n} \left(\left(\frac{P_{byte}}{255}\right)^h \cdot \left(1-P_{byte}\right)^{(n-h)} \cdot \sum_{i=0}^{\theta} C_\theta^i \cdot 254^i \cdot \left(\sum_{\substack{j=0 \\ i+2j+h=\varphi+\theta}}^{\theta-i} C_{\theta-i}^j \cdot C_{n-\theta}^{h+j-\theta} \cdot 255^{h+j-\theta} \right) \right) =$$

$$= \sum_{h=t+1}^{n} \left(\sum_{i=0}^{\theta} C_\theta^i \cdot 254^i \cdot \left(\sum_{\substack{j=0 \\ i+2j+h=\varphi+\theta}}^{\theta-i} C_{\theta-i}^j \cdot C_{n-\theta}^{h+j-\theta} \cdot \frac{P_{byte}^h}{255^{\theta-j}} \cdot \left(1-P_{byte}\right)^{(n-h)} \right) \right).$$

Воспользуемся условием $i+2j+h=\varphi+\theta$, из которого можно выразить $h=\varphi+\theta-i-2j$, и тогда избавимся от внешнего суммирования по индексу h:

$$\sum_{i=0}^{\theta} C_{\theta}^{i}\cdot 254^{i}\cdot\left(\sum_{j=0}^{\theta-i} C_{\theta-i}^{j}\cdot C_{n-\theta}^{\varphi-i-j}\cdot\frac{P_{byte}^{\varphi+\theta-i-2j}}{255^{\theta-j}}\cdot\left(1-P_{byte}\right)^{(n-(\varphi+\theta-i-2j))}\right)$$

Заметим, что биномиальный коэффициент $C_{n-\theta}^{\varphi-i-j}$ равен нулю, если $\varphi-i-j<0$. Отсюда можно получить дополнительное ограничение на сумму индексов $i+j\le\varphi$. Тогда, поскольку сами по себе индексы i и j могут принимать только неотрицательные значения $i\ge 0$ и $j\ge 0$, то, очевидно, что каждый из них в отдельности также не может быть больше φ, иными словами, $0\le i\le\varphi$ и $0\le j\le\varphi$. А учитывая, что $i+j\le\varphi$, можем переписать пределы для индексов i и j: $0\le i\le\varphi$ и $0\le j\le\varphi-i$. Отметим также, что даже в случае сознательного нарушения условия $2t+1\le\theta\le n$, при котором θ может оказаться меньше φ, так же как и $\theta-i$ может оказаться меньше $\varphi-i$, для всех $i>\theta$ биномиальный коэффициент C_{θ}^{i} будет равен нулю, также как и для всех $j>\theta-i$ биномиальный коэффициент $C_{\theta-i}^{j}$ будет равен нулю. Соответственно, лишние «нулевые слагаемые» никак не повлияют на результаты суммирования. Поэтому мы можем произвести замену верхних пределов соответствующих суммирований без лишних опасений.

Тогда с учетом вышеприведенных рассуждений получаем:

$$\sum_{i=0}^{\varphi} C_{\theta}^{i}\cdot 254^{i}\cdot\left(\sum_{j=0}^{\varphi-i} C_{\theta-i}^{j}\cdot C_{n-\theta}^{\varphi-i-j}\cdot\frac{P_{byte}^{\varphi+\theta-i-2j}}{255^{\theta-j}}\cdot\left(1-P_{byte}\right)^{(n-(\varphi+\theta-i-2j))}\right)$$

Теперь сделаем, еще одну замену индексов. Обозначим, $\mu=i+j$, а индекс j выразим как $j=\mu-i$. Пределы для нового индекса μ, учитывая, что $i+j\le\varphi$, $i\ge 0$, $j\ge 0$, очевидно, будут следующими: $0\le\mu\le\varphi$. Пределы для индекса i останутся прежними: $0\le i\le\varphi$. Тогда, заметив также, что $\varphi+\theta-i-2j=\varphi+\theta+i-2\mu$, получаем следующее выражение:

$$\sum_{i=0}^{\varphi} C_{\theta}^{i}\cdot 254^{i}\cdot\left(\sum_{\mu=0}^{\varphi} C_{\theta-i}^{\mu-i}\cdot C_{n-\theta}^{\varphi-\mu}\cdot\frac{P_{byte}^{\varphi+\theta+i-2\mu}}{255^{\theta-\mu+i}}\cdot\left(1-P_{byte}\right)^{(n-(\varphi+\theta+i-2\mu))}\right)$$

Теперь мы можем переставить порядок суммирования, поскольку пределы внутреннего суммирования не содержат индекса i внешнего суммирования:

$$\sum_{\mu=0}^{\varphi}\left(\sum_{i=0}^{\varphi} C_{\theta}^{i}\cdot 254^{i}\cdot C_{\theta-i}^{\mu-i}\cdot C_{n-\theta}^{\varphi-\mu}\cdot\frac{P_{byte}^{\varphi+\theta+i-2\mu}}{255^{\theta-\mu+i}}\cdot\left(1-P_{byte}\right)^{(n-(\varphi+\theta+i-2\mu))}\right)$$

Теперь заметим, что биномиальный коэффициент $C_{\theta-i}^{\mu-i}$ равен нулю, если $i>\mu$, тогда, очевидно, верхний предел для индекса i можно уменьшить до μ. Также заметим, что произведение $C_{\theta}^{i}\cdot C_{\theta-i}^{\mu-i}=\frac{\theta!}{i!(\theta-i)!}\cdot\frac{(\theta-i)!}{(\mu-i)!(\theta-\mu)!}=\frac{\theta!}{i!}\cdot\frac{1}{(\mu-i)!(\theta-\mu)!}\cdot\frac{\mu!}{\mu!}=C_{\mu}^{i}\cdot C_{\theta}^{\mu}$.

Тогда получаем следующее выражение:

$$\sum_{\mu=0}^{\varphi}\left(C_{\theta}^{\mu}\cdot C_{n-\theta}^{\varphi-\mu}\cdot\sum_{i=0}^{\mu}C_{\mu}^{i}\cdot 254^{i}\cdot\frac{P_{byte}^{\varphi+\theta+i-2\mu}\cdot\left(1-P_{byte}\right)^{(n-(\varphi+\theta+i-2\mu))}}{255^{\theta-\mu+i}}\right)$$

Теперь вынесем все независящие от индекса i составляющие из внутренней суммы:

$$\sum_{\mu=0}^{\varphi}\left(C_{\theta}^{\mu}\cdot C_{n-\theta}^{\varphi-\mu}\cdot\frac{P_{byte}^{\varphi+\theta-2\mu}\cdot\left(1-P_{byte}\right)^{(n-(\varphi+\theta-2\mu))}}{255^{\theta-\mu}}\cdot\sum_{i=0}^{\mu}C_{\mu}^{i}\cdot\frac{254^{i}}{255^{i}}\cdot\frac{P_{byte}^{i}}{\left(1-P_{byte}\right)^{i}}\right)$$

Нетрудно заметить, что теперь внутренняя сумма представляет собой разложение в степенной ряд функции: $\left(1+\frac{254}{255}\cdot\frac{P_{byte}}{\left(1-P_{byte}\right)}\right)^{\mu}=\left(\frac{255-P_{byte}}{255\cdot\left(1-P_{byte}\right)}\right)^{\mu}=\left(\frac{1-P_{byte}/255}{1-P_{byte}}\right)^{\mu}$.

Тогда окончательно получаем формулу для расчета вероятности того, что искаженный кадр будет находиться на расстоянии φ, $0\le\varphi\le t$, от некоторого «допустимого» кадра *U*, находящегося на расстоянии θ, $2t+1\le\theta\le n$, от исходного кадра *F*:

$$V_{\theta\varphi}=\sum_{\mu=0}^{\varphi}\left(C_{\theta}^{\mu}\cdot C_{n-\theta}^{\varphi-\mu}\cdot\frac{P_{byte}^{\varphi+\theta-2\mu}\cdot\left(1-P_{byte}\right)^{(n-(\varphi+\theta-\mu))}}{255^{\theta-\mu}}\cdot\left(1-\frac{P_{byte}}{255}\right)^{\mu}\right)\qquad(6.7.2)$$

Полученная в итоге формула не очень наглядна, и достаточно трудна для понимания, зато она содержит всего одно суммирование (вместо тройного, которое мы имели в начале наших преобразований), и потому выгодна с точки зрения вычислительной сложности.

Примечание. Отметим, что при значениях $\theta>n-t$, появляется возможность того, что $\varphi-\mu$ может оказаться больше $n-\theta$, и биномиальный коэффициент $C_{n-\theta}^{\varphi-\mu}$ в этих случаях равен нулю, и поэтому можно «оптимизировать» нижний предел для индекса μ: начинать суммирование не с 0, а с величины $\max\{0,\varphi-(n-\theta)\}$. К тому же такая «оптимизация» обеспечит то, что степень $n-(\varphi+\theta-\mu)=(n-\theta)-(\varphi-\mu)$ будет всегда неотрицательной, и избавит нас от проблем в случае, если вероятность искажения бита будет задана равной $p=1$, при которой $1-P_{byte}=0$, и возведение нуля в отрицательную степень равносильно делению на нуль. Аналогично, если θ сознательно будет задана с нарушением условия $2t+1\le\theta\le n$, тогда также возможна ситуация, когда μ может оказаться больше θ, и биномиальный коэффициент C_{θ}^{μ} в этих случаях также равен нулю. Поэтому, можно также «оптимизировать» для таких случаев верхний предел для индекса μ: заканчивать суммирование не на φ, а на $\min\{\varphi,\theta\}$. К тому же такая «оптимизация» обеспечит то, что степень $\varphi+\theta-2\mu=(\varphi-\mu)+(\theta-\mu)$ будет также всегда неотрицательной, и избавит нас от проблем, в случае если вероятность искажения бита будет задана равной $p=0$, при которой $P_{byte}=0$, и возведение нуля в отрицательную степень равносильно делению на нуль. Мы не будем делать «оптимизацию пределов» в рамках наших теоретических рассуждений, дабы не усложнять их, однако, при разработке программных реализаций ее следует учесть.

Теперь рассмотрим несколько важных частных случаев этой формулы, которые нам пригодятся при дальнейших рассуждениях.

Во-первых, рассмотрим случай $\varphi = 0$. В этом случае, суммирование в формуле превращается единственное слагаемое при $\mu = 0$, и, соответственно, получаем:

$$\lim_{\varphi \to 0} V_{\theta\varphi} = \frac{1}{255^{\theta}} \cdot P_{byte}^{\theta} \cdot \left(1 - P_{byte}\right)^{(n-\theta)} \tag{6.7.3}$$

Во-вторых, рассмотрим случай $\theta = 0$. В этом случае, поскольку биномиальный коэффициент C_{θ}^{μ} не равен нулю, только при $\mu = 0$, то получаем:

$$\lim_{\theta \to 0} V_{\theta\varphi} = C_{n}^{\varphi} \cdot P_{byte}^{\varphi} \cdot \left(1 - P_{byte}\right)^{(n-\varphi)} \tag{6.7.4}$$

Таким образом, мы, наконец, получили формулу для расчета вероятности $V_{\theta\varphi}$ того, что искаженный кадр окажется на заданном расстоянии φ, $0 \le \varphi \le t$, от некоторого «допустимого» кадра *U*, находящегося на расстоянии θ, $2t+1 \le \theta \le n$, от исходного кадра *F*.

Здесь особо отметим, что вероятность $V_{\theta\varphi}$ зависит от расстояний θ и φ, но не зависит от «направлений». Иными словами, вероятность не зависит от того, в каком конкретном направлении находится «допустимый» кадр *U* относительно исходного кадра *F*. Кроме того, в формуле $V_{\theta\varphi}$ уже учтены и просуммированы всевозможные «направления» искаженного кадра *Z* относительно «допустимого» кадра *U*. То есть $V_{\theta\varphi}$ – это вероятность появления любого из множества кадров, находящихся на «поверхности сферы» радиуса φ с центром в некотором «допустимом» кадре *U*. Наконец, особо подчеркнем, что формула $V_{\theta\varphi}$ учитывает кадры *Z* только вокруг одного и только одного некоторого «допустимого» кадра *U*, никакого суммирования по «допустимым» кадрам в формуле нет.

Поэтому, чтобы получить вероятность смежно-исправимого искажения, нам потребуется суммирование по всем расстояниям $\varphi = 1 \ldots t$ (только в этом случае искаженный кадр находится на расстоянии не более *t* от некоторого «допустимого» кадра и при этом не совпадает с ним), суммирование по всем «допустимым» кадрам, находящимся на расстоянии θ от исходного кадра, и, наконец, суммирование по всем расстояниям $\theta = 2t+1 \ldots n$.

Аналогично, чтобы получить вероятность маскируемого искажения, нам нужно положить $\varphi = 0$ (только в этом случае искаженный кадр совпадает с «допустимым» кадром), затем выполнить суммирование по всем «допустимым» кадрам, находящимся на расстоянии θ от исходного кадра, и, наконец, суммирование по всем расстояниям $\theta = 2t+1 \ldots n$.

Ключевой вопрос заключается в том, как выполнять суммирование по всем «допустимым» кадрам, находящимся на расстоянии θ. Учитывая, что $V_{\theta\varphi}$ зависит θ, но не зависит от конкретного «направления», на котором находится тот или иной «допустимый» кадр, то, очевидно, что суммирование сведется к простому умножению на количество A_{θ} «допустимых» кадров, находящихся на заданном расстоянии θ от исходного кадра *F*.

Как было установлено ранее (в разделе, посвященного кодированию) количество «допустимых» кадров, находящихся на заданном расстоянии θ от исходного кадра, определяется формулой $\begin{cases} A_0 = 1; \quad A_1 = \ldots = A_{2t} = 0 \\ A_{\theta} = C_n^{\theta} \cdot \left(\sum\limits_{i=0}^{\theta-2t-1} (-1)^i \cdot C_{\theta}^{i} \cdot (256^{\theta-2t-i} - 1) \right); \quad 2t+1 \le \theta \le n \end{cases}$.

Теперь, наконец, у нас имеются все «составляющие части» для вывода формул вероятности смежно-исправимого и вероятности маскируемого искажения.

Тогда с учетом всех вышеприведенных рассуждений, запишем окончательную формулу вероятности смежно-исправимого искажения:

$$P_{с.и.} = \sum_{\theta = 2t+1}^{n} A_\theta \cdot \left(\sum_{\varphi = 1}^{t} V_{\theta\varphi} \right) \qquad (6.8.1)$$

Соответственно, окончательная формула вероятности маскируемого искажения:

$$P_{м.и.} = \sum_{\theta = 2t+1}^{n} A_\theta \cdot \lim_{\varphi \to 0} V_{\theta\varphi} \qquad (6.8.2)$$

Учитывая, что $\lim_{\varphi \to 0} V_{\theta\varphi} = \frac{1}{255^\theta} \cdot P_{byte}^{\theta} \cdot \left(1 - P_{byte}\right)^{(n-\theta)}$, формулу для вычисления вероятности маскируемого искажения можно также записать в следующем виде:

$$P_{м.и.} = \sum_{\theta = 2t+1}^{n} A_\theta \cdot \left(P_{byte} / 255 \right)^{\theta} \cdot \left(1 - P_{byte}\right)^{(n-\theta)} \qquad (6.8.2^*)$$

Нетрудно заметить, что $P_{с.и.} + P_{м.и.} = \sum_{\theta = 2t+1}^{n} A_\theta \cdot \left(\sum_{\varphi = 0}^{t} V_{\theta\varphi} \right)$.

Кроме того ранее мы установили, $P_{н.и.} + P_{с.и.} + P_{м.и.} = P(T > t)$, причем $P(T > t) = \sum_{h = t+1}^{n} C_n^h \cdot P_{byte}^h \cdot \left(1 - P_{byte}\right)^{(n-h)}$. Тогда, учитывая, что $P_{н.и.} = P(T > t) - (P_{с.и.} + P_{м.и.})$ мы можем записать окончательную формулу для вероятности неисправимого искажения:

$$P_{н.и.} = \sum_{\varphi = t+1}^{n} \left(C_n^\varphi \cdot P_{byte}^\varphi \cdot \left(1 - P_{byte}\right)^{(n-\varphi)} \right) - \sum_{\theta = 2t+1}^{n} A_\theta \cdot \left(\sum_{\varphi = 0}^{t} V_{\theta\varphi} \right) \qquad (6.8.3)$$

По сути, эта формула означает следующее: вероятность неисправимого искажения – это вероятность того, что кадр окажется за пределами расстояний $0 \le \varphi \le t$ как от исходного кадра *F*, так и от любого другого «допустимого» кадра *U*, находящегося на любом из расстояний $2t+1 \le \theta \le n$ от исходного кадра *F*.

Учитывая, что $\lim_{\theta \to 0} V_{\theta\varphi} = C_n^\varphi \cdot P_{byte}^\varphi \cdot \left(1 - P_{byte}\right)^{(n-\varphi)}$, мы можем полученную формулу вероятности неисправимого искажения записать в более компактном виде:

$$P_{н.и.} = \sum_{\varphi = t+1}^{n} \left(\lim_{\theta \to 0} V_{\theta\varphi} \right) - \sum_{\theta = 2t+1}^{n} A_\theta \cdot \left(\sum_{\varphi = 0}^{t} V_{\theta\varphi} \right) \qquad (6.8.3^*)$$

Частный случай. Рассмотрим важные частные случаи вероятностей смежного исправления, маскирования искажения и неисправимого искажения, когда вероятность искажения отдельного бита равна $p = 1/2$. То есть любой бит может с шансами «50 на 50» либо исказиться, либо нет. Такая картина может наблюдаться в сильно зашумленных каналах передачи данных, а также на носителях информации, подвергшихся сильному разрушающему воздействию, например, взрыву большой мощности.

Очевидно, что при $p = 1/2$, $P_{byte} = 1 - (1-p)^8 = 255/256$. Подставляя это значение $P_{byte} = 255/256$ в формулу для расчета $V_{\theta\varphi}$, получаем следующее выражение:

$$\sum_{\mu=0}^{\varphi}\left(C_{\theta}^{\mu}\cdot C_{n-\theta}^{\varphi-\mu}\cdot\frac{(255/256)^{\varphi+\theta-2\mu}\cdot(1-255/256)^{(n-(\varphi+\theta-\mu))}}{255^{\theta-\mu}}\cdot\left(1-\frac{255/256}{255}\right)^{\mu}\right)=$$

$$\sum_{\mu=0}^{\varphi}\left(C_{\theta}^{\mu}\cdot C_{n-\theta}^{\varphi-\mu}\cdot\frac{255^{\varphi+\theta-2\mu}\cdot(1/256)^{(n+\mu))}}{255^{\theta-\mu}}\cdot\left(\frac{255}{256}\right)^{\mu}\right)=\sum_{\mu=0}^{\varphi}\left(C_{\theta}^{\mu}\cdot C_{n-\theta}^{\varphi-\mu}\cdot\frac{255^{\varphi}}{256^{n}}\right).$$

Воспользуемся свойством биномиальных коэффициентов $\sum_{\mu=0}^{\varphi}\left(C_{\theta}^{\mu}\cdot C_{n-\theta}^{\varphi-\mu}\right)=C_{n}^{\varphi}$.

Свойство нетрудно доказать с помощью «производящих функций». С одной стороны $(1+x)^n = \sum_{\varphi=0}^{n} C_n^{\varphi}\cdot x^{\varphi}$. С другой стороны, раскладывая $(1+x)^n$ на два сомножителя, имеем

$$(1+x)^{\theta}(1+x)^{n-\theta}=\left(\sum_{\mu=0}^{\theta}C_{\theta}^{\mu}\cdot x^{\mu}\right)\cdot\left(\sum_{\lambda=0}^{n-\theta}C_{n-\theta}^{\lambda}\cdot x^{\lambda}\right)=\sum_{\mu=0}^{\theta}\left(\sum_{\lambda=0}^{n-\theta}C_{\theta}^{\mu}\cdot C_{n-\theta}^{\lambda}\cdot x^{\mu+\lambda}\right).$$

Обозначая $\varphi = \mu + \lambda$ и выражая $\lambda = \varphi - \mu$, получим $\sum_{\mu=0}^{\theta}\left(\sum_{\varphi=\mu}^{n-\theta+\mu}C_{\theta}^{\mu}\cdot C_{n-\theta}^{\varphi-\mu}\cdot x^{\varphi}\right)$.

Заметим, что поскольку при $0 \le \varphi < \mu$ и $n-\theta+\mu < \varphi \le n$ коэффициент $C_{n-\theta}^{\varphi-\mu}$ обращается в нуль, то мы можем, без опасений расширить пределы суммирования по φ до $0 \le \varphi \le n$, и получить: $\sum_{\mu=0}^{\theta}\left(\sum_{\varphi=0}^{n}C_{\theta}^{\mu}\cdot C_{n-\theta}^{\varphi-\mu}\cdot x^{\varphi}\right)$. Теперь переставим порядок суммирования $\sum_{\varphi=0}^{n}\left(\sum_{\mu=0}^{\theta}C_{\theta}^{\mu}\cdot C_{n-\theta}^{\varphi-\mu}\cdot x^{\varphi}\right)$ и заметим, что $C_{n-\theta}^{\varphi-\mu}$ равен нулю при $\mu > \varphi$, так что верхний предел суммирования по μ можно снизить до φ: $\sum_{\varphi=0}^{n}\left(\sum_{\mu=0}^{\varphi}C_{\theta}^{\mu}\cdot C_{n-\theta}^{\varphi-\mu}\right)\cdot x^{\varphi}$.

Сравнивая выражение при x^{φ} в полученной сумме с аналогичным коэффициентом в сумме $\sum_{\varphi=0}^{n}C_{n}^{\varphi}\cdot x^{\varphi}$, нетрудно убедиться, что $\sum_{\mu=0}^{\varphi}\left(C_{\theta}^{\mu}\cdot C_{n-\theta}^{\varphi-\mu}\right)=C_{n}^{\varphi}$.

Тогда в итоге получаем формулу для вероятности $V_{\theta\varphi}$ для частного случая, когда вероятность искажения отдельного бита равна $p = 1/2$: $\lim_{p\to 1/2} V_{\theta\varphi} = \frac{255^{\varphi}}{256^{n}}\cdot C_{n}^{\varphi}$. Заметим, что «асимптотическая» вероятность $\lim_{p\to 1/2} V_{\theta\varphi}$ вообще не зависит от расстояния θ. Также легко заметить, что при $\varphi = 0$: $\lim_{\varphi\to 0}\left(\lim_{p\to 1/2} V_{\theta\varphi}\right)=\lim_{\varphi\to 0}\left(\frac{255^{\varphi}}{256^{n}}\cdot C_{n}^{\varphi}\right)=\frac{1}{256^{n}}$.

Теперь нетрудно вывести формулу вероятности смежно-исправимого искажения: $\lim\limits_{p \to 1/2} P_{с.и.} = \lim\limits_{p \to 1/2} \left(\sum\limits_{\theta = 2t+1}^{n} A_\theta \cdot \left(\sum\limits_{\varphi = 1}^{t} V_{\theta\varphi} \right) \right) = \left(\sum\limits_{\varphi = 1}^{t} C_n^\varphi \cdot \frac{255^\varphi}{256^n} \right) \cdot \sum\limits_{\theta = 2t+1}^{n} A_\theta$. Теперь заметим, что ранее нами было установлено, что: $\sum\limits_{\theta = 2t+1}^{n} A_\theta = 256^{n-2t} - 1$.

Тогда, итоговая формула вероятности смежно-исправимого искажения при $p = 1/2$:

$$\lim_{p \to 1/2} P_{с.и.} = \left(\frac{1}{256^{2t}} - \frac{1}{256^n} \right) \cdot \left(\sum_{\varphi = 1}^{t} C_n^\varphi \cdot 255^\varphi \right) \qquad (6.9.1)$$

Также нетрудно вывести формулу вероятности маскируемого искажения при $p = 1/2$:

$\lim\limits_{p \to 1/2} P_{м.и.} = \lim\limits_{p \to 1/2} \left(\sum\limits_{\theta = 2t+1}^{n} A_\theta \cdot \left(\lim\limits_{\varphi \to 0} V_{\theta\varphi} \right) \right) = \sum\limits_{\theta = 2t+1}^{n} A_\theta \cdot \left(\lim\limits_{\varphi \to 0} \left(\lim\limits_{p \to 1/2} \left(V_{\theta\varphi} \right) \right) \right)$

$= \sum\limits_{\theta = 2t+1}^{n} A_\theta \cdot \frac{1}{256^n}$. Учитывая, что $\sum\limits_{\theta = 2t+1}^{n} A_\theta = 256^{n-2t} - 1$, окончательно имеем:

$$\lim_{p \to 1/2} P_{м.и.} = \frac{1}{256^{2t}} - \frac{1}{256^n} \qquad (6.9.2)$$

Наконец, используя аналогичные рассуждения, можно вывести формулу вероятности неисправимого искажения при $p = 1/2$. Учтем, что $P_{н.и.} = P(T > t) - (P_{с.и.} + P_{м.и.})$, и то, что ранее мы установили $\lim\limits_{p \to 1/2} P(T > t) = \frac{1}{256^n} \sum\limits_{h = t+1}^{n} C_n^h \cdot 255^h = 1 - \frac{1}{256^n} \sum\limits_{\varphi = 0}^{t} C_n^\varphi \cdot 255^\varphi$.

Также нетрудно заметить, что $\lim\limits_{p \to 1/2} (P_{с.и.} + P_{м.и.}) = \left(\frac{1}{256^{2t}} - \frac{1}{256^n} \right) \cdot \left(\sum\limits_{\varphi = 0}^{t} C_n^\varphi \cdot 255^\varphi \right)$.

Тогда, окончательно получаем вероятность неисправимого искажения при $p = 1/2$:

$$\lim_{p \to 1/2} P_{н.и.} = 1 - \frac{1}{256^{2t}} \cdot \left(\sum_{\varphi = 0}^{t} C_n^\varphi \cdot 255^\varphi \right) \qquad (6.9.3)$$

Полученные формулы фактически являются «асимптотическими оценками» вероятностей смежно-исправимого, маскируемого и неисправимого искажений для «худшего случая», когда любой бит может с равной вероятностью либо исказиться, либо остаться неискаженным. Ниже на рисунках 10 и 11 приведены графики зависимостей вероятности смежно-исправимого искажения и вероятности маскируемого искажения от кратности исправляемых ошибок *t* для некоторых значений длины кадра *n* для частного случая $p = 1/2$.

Из рисунка 6.4 нетрудно заметить, чем больше длина кадра и меньше кратность исправляемых ошибок, тем выше вероятность смежно-исправимого искажения. Наихудшее сочетание, это когда длина кадра *n* = 255 и кратность исправляемых ошибок *t* = 1. В этом случае вероятность смежно-исправимого искажения при вероятности искажения бита $p = 1/2$, близка к 1 (≈ 0,9922), то есть почти каждый кадр оказывается искаженным в более чем *t* байтах, и при этом происходит смежное исправление. Однако, с ростом кратности исправляемых ошибок, вероятность смежно-исправимого искажения быстро убывает.

Из рисунка 6.5 видно, вероятность маскирования искажения при вероятности искажения бита $p = 1/2$, даже при наихудшем сочетании длины кадра *n* = 255 и кратности исправляемых ошибок *t* = 1, довольно небольшая (≈ 0,00001526), и с ростом кратности исправляемых ошибок очень быстро убывает.

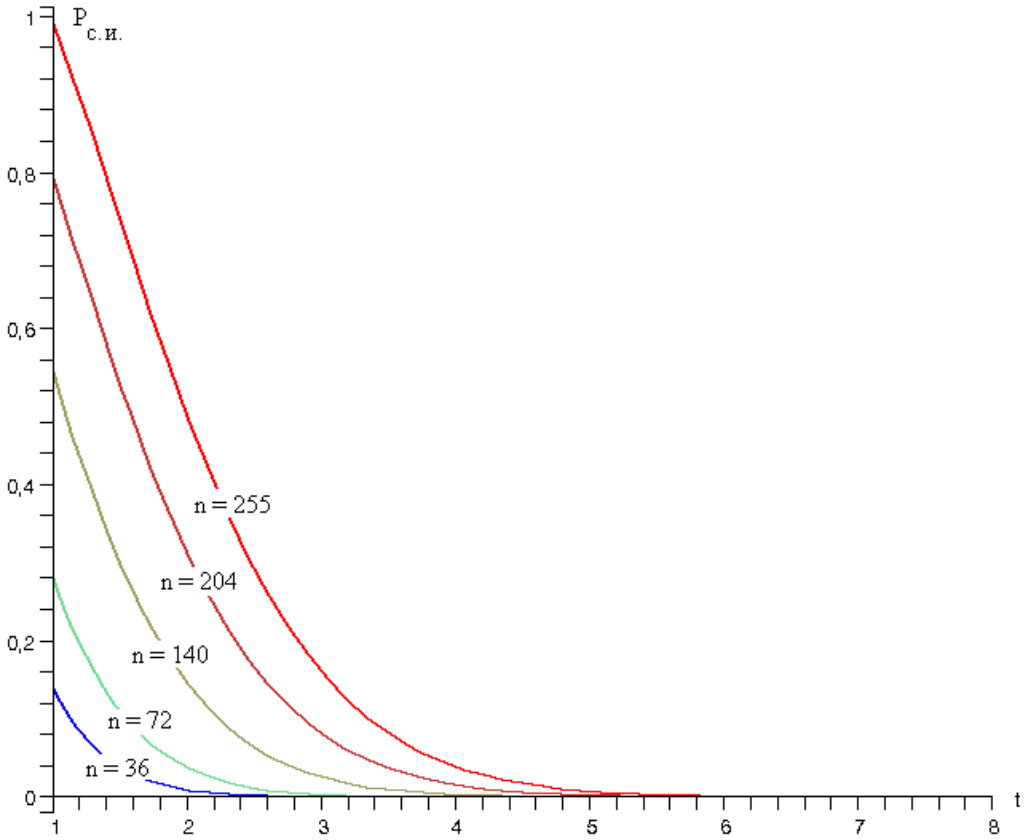

Рис. 6.4. Графики зависимостей вероятности смежно-исправимого искажения от кратности исправляемых ошибок при вероятности искажения бита p = 1 / 2.

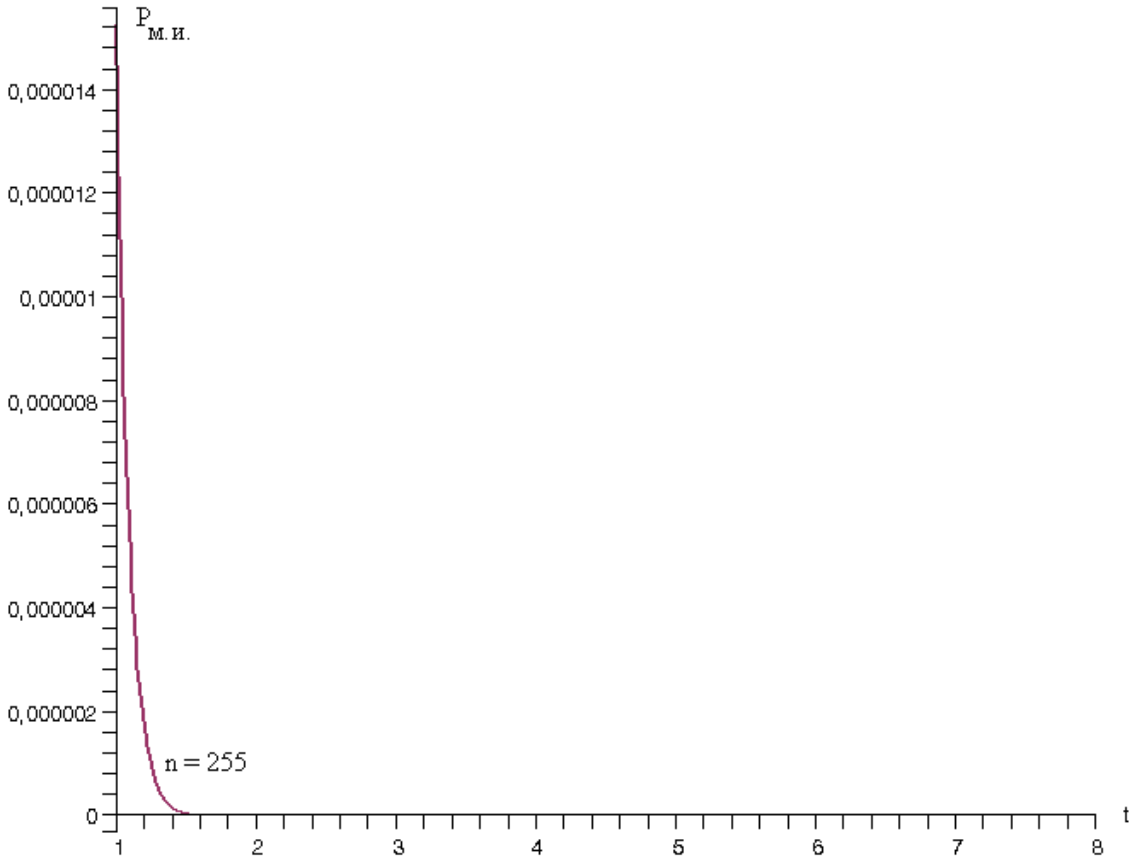

Рис. 6.5. График зависимости вероятности маскируемого искажения от кратности исправляемых ошибок при вероятности искажения бита p = 1 / 2.

Подведем итоги вероятностного анализа, и выпишем формулы для расчета вероятностей для пяти рассматриваемых нами случаев искажения для заданной длины кадра n, кратности исправляемых ошибок t, причем $n \geq 2 \cdot t + 1$, и вероятности искажения бита p:

1) **Вероятность отсутствия искажения**:

$$P(T=0) = \left(1 - P_{byte}\right)^{n}, \text{ где } P_{byte} = 1 - (1-p)^{8} \tag{6.10.1}$$

2) **Вероятность восстанавливаемого искажения**:

$$P(1 \leq T \leq t) = \sum_{\varphi=1}^{t} C_n^{\varphi} \cdot P_{byte}^{\varphi} \cdot \left(1 - P_{byte}\right)^{(n-\varphi)} \tag{6.10.2}$$

3) **Вероятность невосстанавливаемого неисправимого искажения**:

$$P_{н.и.} = \sum_{\varphi=t+1}^{n} \left(C_n^{\varphi} \cdot P_{byte}^{\varphi} \cdot \left(1 - P_{byte}\right)^{(n-\varphi)} \right) - \sum_{\theta=2t+1}^{n} A_{\theta} \cdot \left(\sum_{\varphi=0}^{t} V_{\theta\varphi} \right) \tag{6.10.3}$$

4) **Вероятность невосстанавливаемого смежно-исправимого искажения**:

$$P_{с.и.} = \sum_{\theta=2t+1}^{n} A_{\theta} \cdot \left(\sum_{\varphi=1}^{t} V_{\theta\varphi} \right) \tag{6.10.4}$$

5) **Вероятность невосстанавливаемого маскируемого искажения**:

$$P_{м.и.} = \sum_{\theta=2t+1}^{n} A_{\theta} \cdot \left(P_{byte} / 255 \right)^{\theta} \cdot \left(1 - P_{byte}\right)^{(n-\theta)} \tag{6.10.5}$$

Где,

$$A_{\theta} = C_n^{\theta} \cdot \left(\sum_{i=0}^{\theta-2t-1} (-1)^{i} \cdot C_{\theta}^{i} \cdot (256^{\theta-2t-i} - 1) \right); \quad 2t+1 \leq \theta \leq n$$

$$V_{\theta\varphi} = \sum_{\mu=0}^{\varphi} \left(C_{\theta}^{\mu} \cdot C_{n-\theta}^{\varphi-\mu} \cdot \frac{P_{byte}^{\varphi+\theta-2\mu} \cdot \left(1 - P_{byte}\right)^{(n-(\varphi+\theta-\mu))}}{255^{\theta-\mu}} \cdot \left(1 - \frac{P_{byte}}{255}\right)^{\mu} \right)$$

Причем,

$$\sum_{\theta=2t+1}^{n} A_{\theta} = 256^{n-2t} - 1$$

$$\lim_{\varphi \to 0} V_{\theta\varphi} = \left(P_{byte} / 255 \right)^{\theta} \cdot \left(1 - P_{byte}\right)^{(n-\theta)}$$

$$\lim_{\theta \to 0} V_{\theta\varphi} = C_n^{\varphi} \cdot P_{byte}^{\varphi} \cdot \left(1 - P_{byte}\right)^{(n-\varphi)}$$

$$\lim_{p \to 1/2} V_{\theta\varphi} = \frac{255^{\varphi}}{256^{n}} \cdot C_n^{\varphi}$$

Общая (суммарная) вероятность невосстанавливаемого искажения:

$$P(T > t) = P_{н.и.} + P_{с.и.} + P_{м.и.} = \sum_{\varphi=t+1}^{n} C_n^{\varphi} \cdot P_{byte}^{\varphi} \cdot \left(1 - P_{byte}\right)^{(n-\varphi)}$$

Наконец, сумма вероятностей всевозможных случаев, разумеется, равна 1:

$$P(T=0) + P(1 \leq T \leq t) + P(T > t) = 1$$

Ниже в таблице 3 приведены рассчитанные по полученным формулам вероятности рассмотренных нами пяти случаев искажений при некоторых значениях длины кадра n, кратности исправляемых ошибок t и вероятности искажения бита p.

Таблица 3.

n	t	p	$P(T=0)$	$P(1 \le T \le t)$	$P_{н.и.}$	$P_{с.и.}$	$P_{м.и.}$
255	1	0.5	$7{,}92\cdot10^{-615}$	$5{,}15\cdot10^{-610}$	0,007782	0,992203	$1{,}53\cdot10^{-5}$
255	1	0.0005	0,360503	0,368542	0,002120	0,268834	$1{,}28\cdot10^{-6}$
255	1	0.0000005	0,998981	0,001019	$4{,}06\cdot10^{-9}$	$5{,}14\cdot10^{-7}$	$2{,}69\cdot10^{-15}$
255	8	0.5	$7{,}92\cdot10^{-615}$	$5{,}62\cdot10^{-581}$	0,999979	$2{,}09\cdot10^{-5}$	$2{,}94\cdot10^{-39}$
255	8	0.0005	0,360503	0,639496	$1{,}16\cdot10^{-6}$	$1{,}97\cdot10^{-11}$	$2{,}81\cdot10^{-54}$
255	8	0.0000005	0,998981	0,001019	$2{,}85\cdot10^{-33}$	$4{,}73\cdot10^{-38}$	$7{,}13\cdot10^{-105}$
204	1	0.5	$5{,}24\cdot10^{-492}$	$2{,}72\cdot10^{-487}$	0,206223	0,793762	$1{,}53\cdot10^{-5}$
204	1	0.0005	0,442107	0,361572	0,040724	0,155597	$7{,}55\cdot10^{-7}$
204	1	0.0000005	0,999184	0,000815	$6{,}88\cdot10^{-8}$	$2{,}62\cdot10^{-7}$	$1{,}37\cdot10^{-15}$
204	8	0.5	$5{,}24\cdot10^{-492}$	$6{,}06\cdot10^{-459}$	0,999997	$3{,}40\cdot10^{-6}$	$2{,}94\cdot10^{-39}$
204	8	0.0005	0,442107	0,557893	$1{,}81\cdot10^{-7}$	$4{,}64\cdot10^{-13}$	$6{,}68\cdot10^{-56}$
204	8	0.0000005	0,999184	0,000816	$3{,}69\cdot10^{-34}$	$9{,}26\cdot10^{-40}$	$1{,}40\cdot10^{-106}$
140	1	0.5	$7{,}02\cdot10^{-338}$	$2{,}51\cdot10^{-333}$	0,455246	0,544739	$1{,}53\cdot10^{-5}$
140	1	0.0005	0,571129	0,320553	0,049630	0,058687	$2{,}92\cdot10^{-7}$
140	1	0.0000005	0,999440	0,000560	$7{,}14\cdot10^{-8}$	$8{,}42\cdot10^{-8}$	$4{,}40\cdot10^{-16}$
140	6	0.5	$7{,}02\cdot10^{-338}$	$1{,}81\cdot10^{-313}$	0,999967	$3{,}26\cdot10^{-5}$	$1{,}26\cdot10^{-29}$
140	6	0.0005	0,571129	0,428869	$1{,}83\cdot10^{-6}$	$4{,}64\cdot10^{-11}$	$3{,}88\cdot10^{-43}$
140	6	0.0000005	0,999440	0,000560	$2{,}94\cdot10^{-27}$	$7{,}33\cdot10^{-32}$	$6{,}37\cdot10^{-82}$
72	1	0.5	$4{,}04\cdot10^{-174}$	$7{,}42\cdot10^{-170}$	0,719833	0,280151	$1{,}53\cdot10^{-5}$
72	1	0.0005	0,749708	0,216402	0,024569	0,009321	$4{,}75\cdot10^{-8}$
72	1	0.0000005	0,999712	0,000288	$2{,}97\cdot10^{-8}$	$1{,}12\cdot10^{-8}$	$5{,}87\cdot10^{-17}$
72	4	0.5	$4{,}04\cdot10^{-174}$	$1{,}76\cdot10^{-158}$	0,999764	0,000236	$5{,}42\cdot10^{-20}$
72	4	0.0005	0,749708	0,250281	$1{,}14\cdot10^{-5}$	$2{,}09\cdot10^{-9}$	$9{,}79\cdot10^{-31}$
72	4	0.0000005	0,999712	0,000288	$1{,}43\cdot10^{-20}$	$2{,}60\cdot10^{-24}$	$1{,}25\cdot10^{-57}$
36	1	0.5	$2{,}01\cdot10^{-87}$	$1{,}85\cdot10^{-83}$	0,859909	0,140076	$1{,}53\cdot10^{-5}$
36	1	0.0005	0,865857	0,124964	0,007952	0,001227	$6{,}33\cdot10^{-9}$
36	1	0.0000005	0,999856	0,000144	$8{,}74\cdot10^{-9}$	$1{,}34\cdot10^{-9}$	$7{,}03\cdot10^{-18}$
36	2	0.5	$2{,}01\cdot10^{-87}$	$8{,}24\cdot10^{-80}$	0,990460	0,009540	$2{,}33\cdot10^{-10}$
36	2	0.0005	0,865857	0,133732	0,000409	$3{,}36\cdot10^{-6}$	$8{,}16\cdot10^{-17}$
36	2	0.0000005	0,999856	0,000144	$4{,}53\cdot10^{-13}$	$3{,}71\cdot10^{-15}$	$9{,}13\cdot10^{-32}$

Выбор кратности исправляемых ошибок

При использовании кодов Рида-Соломона за возможность исправления t искаженных байтов приходится «платить» $r = 2 \cdot t$ контрольными байтами, и в условиях, когда у нас задана фиксированная длина кадра n, при увеличении кратности исправляемых ошибок t на долю полезной информации остается все меньшее $k = n - r$ число байтов. В пределе, коды Рида-Соломона могут обеспечить исправление вплоть до $t = \lfloor (n-1)/2 \rfloor$ байтов, при этом, если n нечетно, $n-1$ байтов будут контрольными, так как $r = 2t = 2 \cdot \lfloor (n-1)/2 \rfloor = n-1$, и всего один байт останется на долю полезной информации, так как $k = n - (n-1) = 1$.

Очевидно, что для выбора «разумной» кратности исправляемых ошибок необходимо отталкиваться от каких-то критериев и ограничений. Пусть заданы следующие параметры:

n – фиксированная длина кадра в байтах.

p – базовая вероятность искажения бита.

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

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

Далее, исходя из условия что, должно соблюдаться требование по допустимой вероятности невосстанавливаемого искажения кадра и требование на минимальное число байтов полезной информации, можно поставить следующую простую задачу выбора кратности исправляемой ошибки:

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

Если, невозможно найти параметр t, удовлетворяющий обоим условиям, то, очевидно, применение кодов Рида-Соломона нецелесообразно. Кроме того, очевидно, если вероятность искажения вообще хотя бы одного байта меньше заданной допустимой границы, то есть $P(T > 0) < P_{\max}$, то применение кодов Рида-Соломона также нецелесообразно. Ниже приведена схема алгоритма для оценки целесообразности применения кодов Рида-Соломона и выбора кратности исправляемых ошибок (рис. 6.6).

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

Рассчитаем сначала вероятность возникновения искажения хотя бы в одном байте кадра, и тем самым, оценим целесообразность использования кодов Рида-Соломона:

$$P(T > 0) = 0{,}13414$$

Очевидно, что эта вероятность значительно выше допустимой границы $P_{\max} = 0{,}001$, так что применение кодов Рида-Соломона вполне оправдано. Теперь попробуем выбрать кратность t исправляемой ошибки, исходя из двух условий:

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

Очевидно, для заданного требуемого числа байтов полезной информации, можно рассматривать только два варианта $t = 1$ и $t = 2$. В этих вариантах вероятности невосстанавливаемого искажения кадра равны: $P(T > 1) = 0{,}00918$ и $P(T > 2) = 0{,}000412$. Очевидно, что вариант $t = 2$, удовлетворяет обоим условием. Таким образом, применение кодов Рида-Соломона целесообразно, и выбранная кратность исправляемых ошибок $t = 2$.

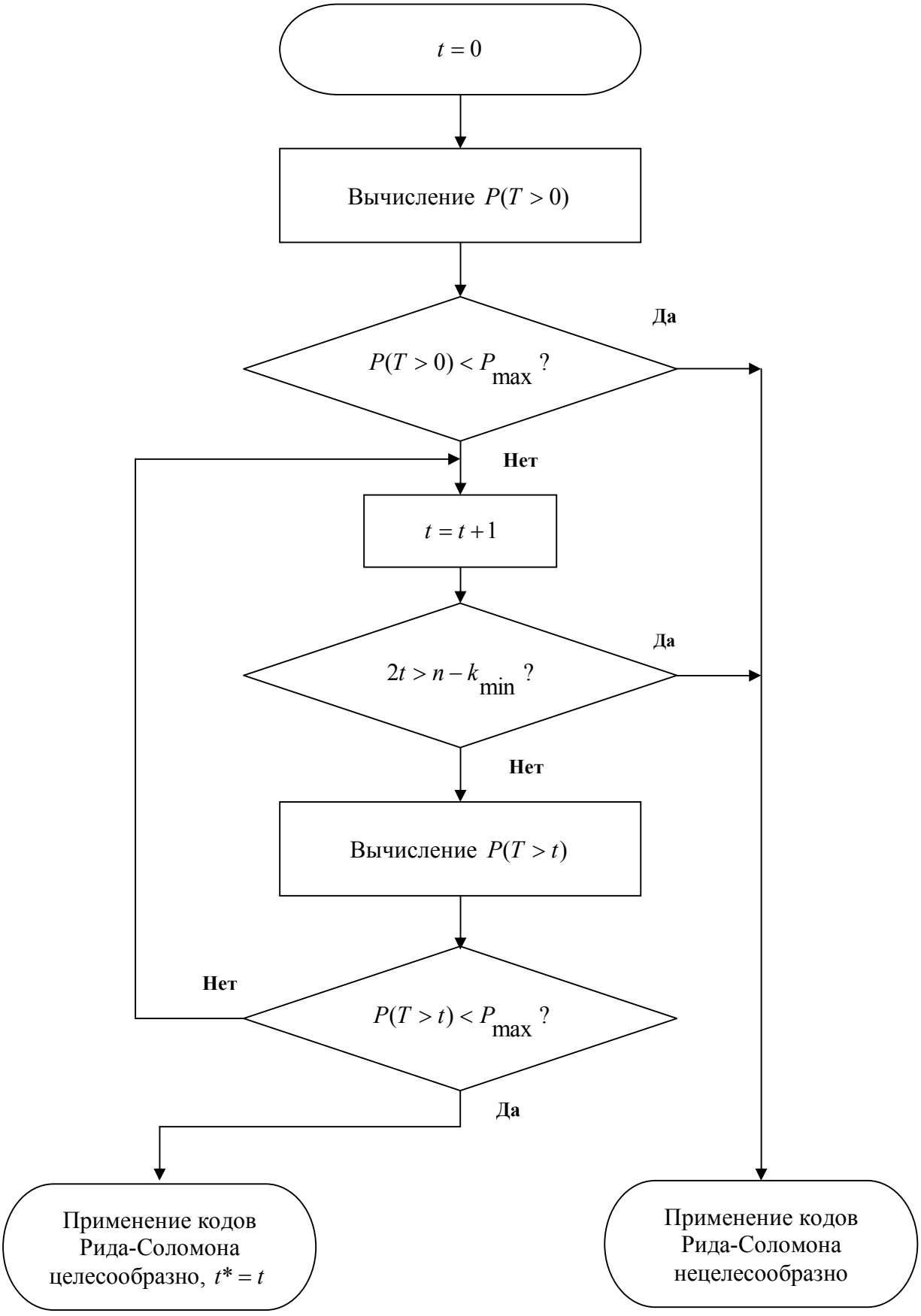

Рис. 6.6. Схема алгоритма для оценки целесообразности применения кодов Рида-Соломона и выбора кратности исправляемых ошибок.

Контрольные вопросы

1. Какие основные допущения делаются в вероятностном анализе искажений?

2. Почему с уменьшением кратности исправляемых ошибок или ростом длины кадра вероятность смежно-исправимых искажений увеличивается?

3. Почему с ростом кратности исправляемых ошибок t увеличивается относительная доля неисправимых искажений среди невосстанавливаемых искажений?

4. Вычислите вероятности отсутствия искажения, а также восстанавливаемого и невосстанавливаемого искажения при длине кадра n = 150 байтов, кратности исправляемых ошибок $t = 2$ и вероятности искажения бита $p = 0{,}001$.

5. Определите математически ожидаемое количество искажаемых байтов при длине кадра $n = 65$ и вероятности искажения бита $p = 0{,}01$.

6. Вычислите вероятности неисправимого, смежно-исправимого и маскируемого искажений при длине кадра $n = 220$ байтов, кратности исправляемых ошибок $t = 2$ и вероятности искажения бита $p = 0{,}001$.

7. Выберите минимальную длину кадра из условий, что количество информационных байтов составляет $k = 53$, вероятность искажения бита $p = 0{,}001$, а требуемая вероятность невосстанавливаемого искажения кадра не должна превышать 0,000001.

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 неумышленного искажения при передаче данных по каналам связи, а также искажения данных на носителях информации при их повреждении.
- Обнаружение и коррекция умышленной модификации информационных сообщений с целью дезинформации (особенно в театре военных действий).
- 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 помощи использования специальных «настоящих» и «ложных» ошибок в секторах.
- Восстановление данных одного или нескольких дисков в отказоустойчивых системах (массивах) хранения данных (RAID, FTDS). Восстановление одного или нескольких томов многотомного архива данных.
- Обнаружение и исправление ошибок в последовательности команд в конвейере (очереди) процессора, вычислительного узла или иной специализированной системе.
- 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. **Б.Л. ван дер Варден**. Алгебра. – М.: Наука, 1979.
2. **Лидл Р., Нидеррайтер Г.** Конечные поля. В 2-х томах. – М.: Мир, 1988.
3. **Чеботарев Н.Г.** Основы теории Галуа. – М.: Едиториал УРСС, 2004.
4. **Кнут Д.** Искусство программирования. В 3-х томах. – М.: Вильямс, 2005.
5. **Новиков Ф.А**. Дискретная математика для программистов. 3-е изд. – СПб.: Питер, 2009.
6. **Блейхут Р.** Теория и практика кодов, контролирующих ошибки. – М.: Мир, 1986.
7. **Берлекэмп Э.** Алгебраическая теория кодирования. – М. Мир, 1971.
8. **Форни Д.** Каскадные коды. – М.: Мир, 1970.
9. **Хэмминг Р.В.** Теория кодирования и теория информации. – М.: Радио и связь, 1983.
10. **У. Питерсон, Э. Уэлдон.** Коды, исправляющие ошибки. – М.: Мир, 1976.
11. **Мак-Вильямс Ф., Слоэн Н.** Теория кодов, исправляющих ошибки. – М.: Связь, 1979.
12. **Кларк Дж., Кейн Дж.** Кодирование с исправлением ошибок в системах цифровой связи. – М.: Радио и связь, 1987.
13. **C.K.P Clarke**. Reed-Solomon error correction. White Paper WHP 031. – British Broadcasting Corporation Research and Development, 2002.
14. **Todd K. Moon**. Error correcting coding: mathematical methods and algorithms. – Hoboken, New Jersey: John Wiley & Sons Inc., 2005.
15. **Advanced Encryption Standard (AES)**. FIPS PUB 197. – National Institute of Standards and Technology, U.S. Department of Commerce, November 2001.
16. **Гнеденко Б.В.** Курс теории вероятностей. – М.: Едиториал УРСС, 2005.

Приложение 1. Примитивные элементы простых полей *GF(p)*, $p < 100$.

- $p = 2$, $\alpha = 1$.
- $p = 3$, $\alpha = 2$.
- $p = 5$, $\alpha = 2, 3$.
- $p = 7$, $\alpha = 3, 5$.
- $p = 11$, $\alpha = 2, 6, 7, 8$.
- $p = 13$, $\alpha = 2, 6, 7, 11$.
- $p = 17$, $\alpha = 3, 5, 6, 7, 10, 11, 12, 14$.
- $p = 19$, $\alpha = 2, 3, 10, 13, 14, 15$.
- $p = 23$, $\alpha = 5, 7, 10, 11, 14, 15, 17, 19, 20, 21$.
- $p = 29$, $\alpha = 2, 3, 8, 10, 11, 14, 15, 18, 19, 21, 26, 27$.
- $p = 31$, $\alpha = 3, 11, 12, 13, 17, 21, 22, 24$.
- $p = 37$, $\alpha = 2, 5, 13, 15, 17, 18, 19, 20, 22, 24, 32, 35$.
- $p = 41$, $\alpha = 6, 7, 11, 12, 13, 15, 17, 19, 22, 24, 26, 28, 29, 30, 34, 35$.
- $p = 43$, $\alpha = 3, 5, 12, 18, 19, 20, 26, 28, 29, 30, 33, 34$.
- $p = 47$, $\alpha = 5, 10, 11, 13, 15, 19, 20, 22, 23, 26, 29, 30, 31, 33, 35, 38, 39, 40, 41, 43, 44, 45$.
- $p = 53$, $\alpha = 2, 3, 5, 8, 12, 14, 18, 19, 20, 21, 22, 26, 27, 31, 32, 33, 34, 35, 39, 41, 45, 48, 50, 51$.
- $p = 59$, $\alpha = 2, 6, 8, 10, 11, 13, 14, 18, 23, 24, 30, 31, 32, 33, 34, 37, 38, 39, 40, 42, 43, 44, 47, 50, 52, 54, 55, 56$.
- $p = 61$, $\alpha = 2, 6, 7, 10, 17, 18, 26, 30, 31, 35, 43, 44, 51, 54, 55, 59$.
- $p = 67$, $\alpha = 2, 7, 11, 12, 13, 18, 20, 28, 31, 32, 34, 41, 44, 46, 48, 50, 51, 57, 61, 63$.
- $p = 71$, $\alpha = 7, 11, 13, 21, 22, 28, 31, 33, 35, 42, 44, 47, 52, 53, 55, 56, 59, 61, 62, 63, 65, 67, 68, 69$.
- $p = 73$, $\alpha = 5, 11, 13, 14, 15, 20, 26, 28, 29, 31, 33, 34, 39, 40, 42, 44, 45, 47, 53, 58, 59, 60, 62, 68$.
- $p = 79$, $\alpha = 3, 6, 7, 28, 29, 30, 34, 35, 37, 39, 43, 47, 48, 53, 54, 59, 60, 63, 66, 68, 70, 74, 75, 77$.
- $p = 83$, $\alpha = 2, 5, 6, 8, 13, 14, 15, 18, 19, 20, 22, 24, 32, 34, 35, 39, 42, 43, 45, 46, 47, 50, 52, 53, 54, 55, 56, 57, 58, 60, 62, 66, 67, 71, 72, 73, 74, 76, 79, 80$.
- $p = 89$, $\alpha = 3, 6, 7, 13, 14, 15, 19, 23, 24, 26, 27, 28, 29, 30, 31, 33, 35, 38, 41, 43, 46, 48, 51, 54, 56, 58, 59, 60, 61, 62, 63, 65, 66, 70, 74, 75, 76, 82, 83, 86$.
- $p = 97$, $\alpha = 5, 7, 10, 13, 14, 15, 17, 21, 23, 26, 29, 37, 38, 39, 40, 41, 56, 57, 58, 59, 60, 68, 71, 74, 76, 80, 82, 83, 84, 87, 90, 92$.

.

<u>Приложение 2</u>. Примеры примитивных неприводимых многочленов над простым полем *GF(p)*, *p* < 100, для некоторых степеней *m*.

GF(2^2)	x^2+x+1	GF(2^3)	x^3+x+1
GF(2^4)	x^4+x+1	GF(2^5)	x^5+x^2+1
GF(2^6)	x^6+x+1	GF(2^7)	x^7+x^3+1
GF(2^8)	x^8+x^4+x^3+x^2+1	GF(2^9)	x^9+x^4+1
GF(2^10)	x^10+x^3+1	GF(2^11)	x^11+x^2+1
GF(2^12)	x^12+x^6+x^4+x+1	GF(2^13)	x^13+x^4+x^3+x+1
GF(2^14)	x^14+x^10+x^6+x+1	GF(2^15)	x^15+x+1
GF(2^16)	x^16+x^12+x^3+x+1	GF(2^17)	x^17+x^3+1
GF(2^18)	x^18+x^7+1	GF(2^19)	x^19+x^5+x^2+x+1
GF(2^20)	x^20+x^3+1	GF(2^21)	x^21+x^2+1
GF(2^22)	x^22+x+1	GF(2^23)	x^23+x^5+1
GF(2^24)	x^24+x^7+x^2+x+1	GF(2^25)	x^25+x^3+1
GF(2^26)	x^26+x^6+x^2+x+1	GF(2^27)	x^27+x^5+x^2+x+1
GF(2^28)	x^28+x^3+1	GF(2^29)	x^29+x^2+1
GF(2^30)	x^30+x^23+x^2+x+1	GF(2^31)	x^31+x^3+1
GF(2^32)	x^32+x^22+x^2+x+1	GF(2^33)	x^33+x^13+1
GF(2^34)	x^34+x^8+x^4+x^3+1	GF(2^35)	x^35+x^2+1
GF(2^36)	x^36+x^11+1	GF(2^37)	x^37+x^6+x^4+x+1
GF(2^38)	x^38+x^6+x^5+x+1	GF(2^39)	x^39+x^4+1
GF(2^40)	x^40+x^5+x^4+x^3+1	GF(2^41)	x^41+x^3+1
GF(2^42)	x^42+x^7+x^4+x^3+1	GF(2^43)	x^43+x^6+x^4+x^3+1
GF(2^44)	x^44+x^6+x^5+x^2+1	GF(2^45)	x^45+x^4+x^3+x+1
GF(2^46)	x^46+x^8+x^7+x^6+1	GF(2^47)	x^47+x^5+1
GF(2^48)	x^48+x^9+x^7+x^4+1	GF(2^49)	x^49+x^9+1
GF(2^50)	x^50+x^4+x^3+x^2+1	GF(2^51)	x^51+x^6+x^3+x+1
GF(2^52)	x^52+x^3+1	GF(2^53)	x^53+x^6+x^2+x+1
GF(2^54)	x^54+x^8+x^6+x^3+1	GF(2^55)	x^55+x^24+1
GF(2^56)	x^56+x^7+x^4+x^2+1	GF(2^57)	x^57+x^7+1
GF(2^58)	x^58+x^19+1	GF(2^59)	x^59+x^7+x^4+x^2+1
GF(2^60)	x^60+x+1	GF(2^61)	x^61+x^5+x^2+x+1
GF(2^62)	x^62+x^6+x^5+x^3+1	GF(2^63)	x^63+x+1
GF(2^64)	x^64+x^4+x^3+x+1	GF(2^65)	x^65+x^18+1
GF(2^66)	x^66+x^9+x^8+x^6+1	GF(2^67)	x^67+x^5+x^2+x+1
GF(2^68)	x^68+x^9+1	GF(2^69)	x^69+x^6+x^5+x^2+1
GF(2^70)	x^70+x^5+x^3+x+1	GF(2^71)	x^71+x^6+1
GF(2^72)	x^72+x^10+x^9+x^3+1	GF(2^73)	x^73+x^25+1
GF(2^74)	x^74+x^7+x^4+x^3+1	GF(2^75)	x^75+x^6+x^3+x+1
GF(2^76)	x^76+x^5+x^4+x^2+1	GF(2^77)	x^77+x^6+x^5+x^2+1
GF(2^78)	x^78+x^7+x^2+x+1	GF(2^79)	x^79+x^9+1
GF(2^80)	x^80+x^9+x^4+x^2+1	GF(2^81)	x^81+x^4+1
GF(2^82)	x^82+x^9+x^6+x^4+1	GF(2^83)	x^83+x^7+x^4+x^2+1
GF(2^84)	x^84+x^13+1	GF(2^85)	x^85+x^8+x^2+x+1
GF(2^86)	x^86+x^6+x^5+x^2+1	GF(2^87)	x^87+x^13+1
GF(2^88)	x^88+x^11+x^9+x^8+1	GF(2^89)	x^89+x^38+1
GF(2^90)	x^90+x^5+x^3+x^2+1	GF(2^91)	x^91+x^8+x^5+x+1
GF(2^92)	x^92+x^6+x^5+x^2+1	GF(2^93)	x^93+x^2+1
GF(2^94)	x^94+x^21+1	GF(2^95)	x^95+x^11+1
GF(2^96)	x^96+x^10+x^9+x^6+1	GF(2^97)	x^97+x^6+1
GF(2^98)	x^98+x^11+1	GF(2^99)	x^99+x^7+x^5+x^4+1
GF(2^100)	x^100+x^37+1	GF(2^101)	x^101+x^7+x^6+x+1
GF(2^102)	x^102+x^6+x^5+x^3+1	GF(2^103)	x^103+x^9+1
GF(2^104)	x^104+x^11+x^10+x+1	GF(2^105)	x^105+x^16+1
GF(2^106)	x^106+x^15+1	GF(2^107)	x^107+x^9+x^7+x^4+1
GF(2^108)	x^108+x^31+1	GF(2^109)	x^109+x^5+x^4+x^2+1
GF(2^110)	x^110+x^6+x^4+x+1	GF(2^111)	x^111+x^10+1

GF(2^112)	x^112+x^11+x^6+x^4+1	GF(2^113)	x^113+x^9+1
GF(2^114)	x^114+x^11+x^2+x+1	GF(2^115)	x^115+x^8+x^7+x^5+1
GF(2^116)	x^116+x^6+x^5+x^2+1	GF(2^117)	x^117+x^5+x^2+x+1
GF(2^118)	x^118+x^33+1	GF(2^119)	x^119+x^8+1
GF(2^120)	x^120+x^9+x^6+x^2+1	GF(2^121)	x^121+x^18+1
GF(2^122)	x^122+x^6+x^2+x+1	GF(2^123)	x^123+x^2+1
GF(2^124)	x^124+x^37+1	GF(2^125)	x^125+x^7+x^6+x^5+1
GF(2^126)	x^126+x^7+x^4+x^2+1	GF(2^127)	x^127+x+1
GF(2^128)	x^128+x^7+x^2+x+1	GF(2^144)	x^144+x^7+x^4+x^2+1
GF(2^160)	x^160+x^5+x^3+x^2+1	GF(2^176)	x^176+x^12+x^11+x^9+1
GF(2^192)	x^192+x^15+x^11+x^5+1	GF(2^200)	x^200+x^5+x^3+x^2+1
GF(2^208)	x^208+x^9+x^3+x+1	GF(2^224)	x^224+x^12+x^7+x^2+1
GF(2^240)	x^240+x^8+x^5+x^3+1	GF(2^256)	x^256+x^10+x^5+x^2+1
GF(2^300)	x^300+x^7+1	GF(2^320)	x^320+x^4+x^3+x+1
GF(2^360)	x^360+x^26+x^25+x+1	GF(2^384)	x^384+x^16+x^15+x^6+1
GF(2^400)	x^400+x^5+x^3+x^2+1	GF(2^480)	x^480+x^16+x^13+x^7+1
GF(2^512)	x^512+x^8+x^5+x^2+1	GF(2^600)	x^600+x^11+x^10+x+1
GF(2^720)	x^720+x^11+x^8+x^2+1	GF(2^840)	x^840+x^11+x^5+x+1
GF(2^960)	x^960+x^13+x^9+x^6+1	GF(2^1200)	x^1200+x^23+x^8+x^5+1

GF(3^2)	x^2+x+2	GF(3^3)	x^3+2*x+1
GF(3^4)	x^4+x+2	GF(3^5)	x^5+2*x+1
GF(3^6)	x^6+x+2	GF(3^7)	x^7+2*x^2+1
GF(3^8)	x^8+x^3+2	GF(3^9)	x^9+2*x^4+1
GF(3^10)	x^10+x^3+x+2	GF(3^11)	x^11+2*x^2+1
GF(3^12)	x^12+x^5+x+2	GF(3^13)	x^13+2*x+1
GF(3^14)	x^14+x+2	GF(3^15)	x^15+2*x^2+1
GF(3^16)	x^16+x^7+2	GF(3^17)	x^17+2*x+1
GF(3^18)	x^18+x^9+x^5+2	GF(3^19)	x^19+2*x^2+1
GF(3^20)	x^20+x^5+x+2	GF(3^21)	x^21+2*x^5+1
GF(3^22)	x^22+x^5+2	GF(3^23)	x^23+2*x^3+1
GF(3^24)	x^24+x^13+x^5+2	GF(3^25)	x^25+2*x^3+1
GF(3^26)	x^26+x^7+2	GF(3^27)	x^27+2*x^7+1
GF(3^28)	x^28+x^13+2	GF(3^29)	x^29+2*x^4+1
GF(3^30)	x^30+x+2	GF(3^31)	x^31+2*x^5+1
GF(3^32)	x^32+x^5+2	GF(3^33)	x^33+2*x^5+1
GF(3^34)	x^34+x^3+x+2	GF(3^35)	x^35+2*x^2+1
GF(3^36)	x^36+x^17+x+2	GF(3^37)	x^37+2*x^6+1
GF(3^38)	x^38+x^13+x^5+2	GF(3^39)	x^39+2*x^5+x^4+1
GF(3^40)	x^40+x+2	GF(3^41)	x^41+2*x+1
GF(3^42)	x^42+x^9+x^7+2	GF(3^43)	x^43+2*x^17+1
GF(3^44)	x^44+x^3+2	GF(3^45)	x^45+2*x^17+1
GF(3^46)	x^46+x^5+2	GF(3^47)	x^47+2*x^15+1
GF(3^48)	x^48+x^11+x^6+2*x^4+2	GF(3^49)	x^49+x^9+2*x^6+1
GF(3^50)	x^50+x^11+x^9+2	GF(3^51)	x^51+2*x+1
GF(3^52)	x^52+x^7+2	GF(3^53)	x^53+2*x^13+1
GF(3^54)	x^54+x+2	GF(3^55)	x^55+2*x^23+1
GF(3^56)	x^56+x^3+2	GF(3^57)	x^57+x^7+2*x^2+1
GF(3^58)	x^58+x^13+x^11+2	GF(3^59)	x^59+2*x^17+1
GF(3^60)	x^60+x^5+x+2	GF(3^61)	x^61+2*x^7+1
GF(3^62)	x^62+x^9+x^7+2	GF(3^63)	x^63+2*x^26+1
GF(3^64)	x^64+x^3+2	GF(3^72)	x^72+x^18+x^3+2*x^2+2
GF(3^80)	x^80+x^21+2	GF(3^96)	x^96+x^7+x^6+2*x^4+2
GF(3^100)	x^100+x^17+x+2	GF(3^112)	x^112+x^43+2
GF(3^128)	x^128+x^5+2*x^3+x^2+2	GF(3^144)	x^144+x^5+2*x^4+x^2+2
GF(3^160)	x^160+x^27+2	GF(3^176)	x^176+x^15+2
GF(3^192)	x^192+x^5+2*x^4+x^2+2	GF(3^200)	x^200+x^3+2
GF(3^208)	x^208+x^51+2	GF(3^224)	x^224+x^23+2

GF(3^240)	x^240+x^35+x^19+2	GF(3^256)	x^256+x^7+2*x^3+x^2+2
GF(3^300)	x^300+x^9+2*x^6+x^4+2		
GF(5^2)	x^2+x+2	GF(5^3)	x^3+3*x+2
GF(5^4)	x^4+x^2+2*x+2	GF(5^5)	x^5+x^2+2
GF(5^6)	x^6+x+2	GF(5^7)	x^7+3*x+2
GF(5^8)	x^8+x^2+2*x+3	GF(5^9)	x^9+2*x^4+3
GF(5^10)	x^10+x^2+x+3	GF(5^11)	x^11+x^2+2
GF(5^12)	x^12+x^3+2*x+3	GF(5^13)	x^13+2*x^6+3
GF(5^14)	x^14+x^9+x+3	GF(5^15)	x^15+x^2+2
GF(5^16)	x^16+x^3+3*x+2	GF(5^17)	x^17+x^14+2
GF(5^18)	x^18+x^4+2*x+2	GF(5^19)	x^19+4*x^9+2
GF(5^20)	x^20+x^2+2*x+3	GF(5^21)	x^21+4*x+2
GF(5^22)	x^22+x^5+2	GF(5^23)	x^23+2*x^2+3
GF(5^24)	x^24+x^3+2*x^2+2	GF(5^25)	x^25+3*x^7+2
GF(5^26)	x^26+x^5+4*x^3+3	GF(5^27)	x^27+4*x+2
GF(5^28)	x^28+x^3+2*x+3	GF(5^29)	x^29+2*x^6+3
GF(5^30)	x^30+x^5+3*x^2+3	GF(5^31)	x^31+3*x+2
GF(5^32)	x^32+x^5+2*x^3+3	GF(5^36)	x^36+x^7+3*x+2
GF(5^40)	x^40+x^11+2*x^8+2	GF(5^48)	x^48+x^7+4*x^6+3
GF(5^56)	x^56+x^7+4*x^6+3	GF(5^64)	x^64+x^7+2*x^4+2
GF(5^72)	x^72+x^6+2*x^5+2	GF(5^80)	x^80+x^5+2*x^2+2
GF(5^96)	x^96+x^3+4*x^2+2	GF(5^100)	x^100+x^9+3*x^6+3
GF(5^112)	x^112+x^7+4*x^2+2	GF(5^128)	x^128+x^3+x+3
GF(5^144)	x^144+x^15+3*x^10+3	GF(5^160)	x^160+x^6+2*x^5+3
GF(5^200)	x^200+x^8+x^5+3		
GF(7^2)	x^2+x+3	GF(7^3)	x^3+3*x+2
GF(7^4)	x^4+x^2+3*x+5	GF(7^5)	x^5+x+4
GF(7^6)	x^6+x^3+x+5	GF(7^7)	x^7+x^4+2
GF(7^8)	x^8+x+3	GF(7^9)	x^9+3*x^2+4
GF(7^10)	x^10+x^5+x+3	GF(7^11)	x^11+x+4
GF(7^12)	x^12+x^5+3*x+5	GF(7^13)	x^13+5*x^2+2
GF(7^14)	x^14+x^9+3	GF(7^15)	x^15+x^8+5*x^2+4
GF(7^16)	x^16+x^15+3	GF(7^17)	x^17+x+4
GF(7^18)	x^18+x^7+6*x+3	GF(7^19)	x^19+x^8+4
GF(7^20)	x^20+x^3+3	GF(7^21)	x^21+3*x^8+4
GF(7^22)	x^22+x^3+3	GF(7^23)	x^23+x^10+4
GF(7^24)	x^24+x^5+6*x+3	GF(7^25)	x^25+x^4+2
GF(7^26)	x^26+x^9+3	GF(7^27)	x^27+3*x^8+4
GF(7^28)	x^28+x^4+3*x+3	GF(7^29)	x^29+x^13+4
GF(7^30)	x^30+x^2+x+5	GF(7^31)	x^31+x^4+2
GF(7^32)	x^32+x^7+3	GF(7^36)	x^36+x^5+3*x^3+5
GF(7^40)	x^40+x^9+3	GF(7^48)	x^48+x^5+3*x+5
GF(7^56)	x^56+x^2+x+5	GF(7^64)	x^64+x^21+3
GF(7^72)	x^72+x^9+x+5	GF(7^80)	x^80+x^69+3
GF(7^96)	x^96+x^7+x^3+5	GF(7^100)	x^100+x^5+5*x+3
GF(7^112)	x^112+x^7+5*x^4+3	GF(7^128)	x^128+x^6+3*x^3+3
GF(7^144)	x^144+x^5+4*x+5	GF(7^160)	x^160+x^5+x^4+5
GF(7^200)	x^200+x^2+x+3		
GF(11^2)	x^2+x+7	GF(11^3)	x^3+x+4
GF(11^4)	x^4+x+2	GF(11^5)	x^5+2*x^2+9
GF(11^6)	x^6+x^2+2*x+8	GF(11^7)	x^7+x+4
GF(11^8)	x^8+x^2+2*x+6	GF(11^12)	x^12+x+7
GF(11^16)	x^16+x^7+7	GF(11^24)	x^24+x+2
GF(11^32)	x^32+x^11+7	GF(11^48)	x^48+x^11+8
GF(11^64)	x^64+x^3+8	GF(11^80)	x^80+x^4+2*x+8

GF(11^96)	x^96+x^8+4*x+8	GF(11^100)	x^100+x^3+4*x^2+6
GF(13^2)	x^2+x+2	GF(13^3)	x^3+x+6
GF(13^4)	x^4+x^2+x+2	GF(13^5)	x^5+3*x^2+2
GF(13^6)	x^6+x^2+2*x+2	GF(13^7)	x^7+x^4+2
GF(13^8)	x^8+x^3+x+2	GF(13^12)	x^12+x^2+x+2
GF(13^16)	x^16+x^3+6	GF(13^24)	x^24+x^2+6*x+2
GF(13^32)	x^32+x^4+2*x+11	GF(13^48)	x^48+x^2+2*x+6
GF(13^64)	x^64+x^3+x+11	GF(13^80)	x^80+x^3+4*x^2+2
GF(13^96)	x^96+x^5+8*x^4+6	GF(13^100)	x^100+x^4+2*x+2
GF(17^2)	x^2+x+3	GF(17^3)	x^3+x+3
GF(17^4)	x^4+x+11	GF(17^5)	x^5+x+3
GF(17^6)	x^6+x+12	GF(17^7)	x^7+x+5
GF(17^8)	x^8+x^3+5	GF(17^12)	x^12+x^5+6
GF(17^16)	x^16+x^3+2*x+7	GF(17^24)	x^24+x^2+2*x+7
GF(17^32)	x^32+x^2+x+6	GF(17^48)	x^48+x^2+2*x+6
GF(17^64)	x^64+x^7+4*x+7	GF(17^80)	x^80+x^5+2*x^2+12
GF(17^96)	x^96+x^2+8*x+3	GF(17^100)	x^100+x^3+x+7
GF(19^2)	x^2+x+2	GF(19^3)	x^3+x+4
GF(19^4)	x^4+2*x+10	GF(19^5)	x^5+x+9
GF(19^6)	x^6+x+3	GF(19^7)	x^7+x^2+6
GF(19^8)	x^8+x+2	GF(19^12)	x^12+x+15
GF(19^16)	x^16+x^2+5*x+13	GF(19^24)	x^24+x^5+14
GF(19^32)	x^32+x^13+3	GF(19^48)	x^48+x^5+2*x+13
GF(19^64)	x^64+x^13+2	GF(19^80)	x^80+x+14
GF(19^96)	x^96+x^4+5*x+13	GF(19^100)	x^100+x^2+8*x+3
GF(23^2)	x^2+x+7	GF(23^3)	x^3+x+3
GF(23^4)	x^4+x+11	GF(23^5)	x^5+x+3
GF(23^6)	x^6+x^5+7	GF(23^7)	x^7+x^2+3
GF(23^8)	x^8+x^3+7	GF(23^12)	x^12+x^5+11
GF(23^16)	x^16+x^15+11	GF(23^24)	x^24+x+7
GF(23^32)	x^32+x^9+19	GF(23^48)	x^48+2*x^13+19
GF(23^64)	x^64+x^13+11	GF(23^80)	x^80+x^33+17
GF(29^2)	x^2+x+3	GF(29^3)	x^3+x+11
GF(29^4)	x^4+x+19	GF(29^5)	x^5+x+8
GF(29^6)	x^6+x+3	GF(29^7)	x^7+x+11
GF(29^8)	x^8+x^2+3*x+3	GF(29^12)	x^12+x+15
GF(29^16)	x^16+x^3+21	GF(29^24)	x^24+x+21
GF(29^32)	x^32+x+19	GF(29^48)	x^48+x^19+19
GF(29^64)	x^64+2*x^17+8	GF(29^80)	x^80+x^2+x+2
GF(31^2)	x^2+x+2	GF(31^3)	x^3+x+14
GF(31^4)	x^4+x^3+13	GF(31^5)	x^5+x+18
GF(31^6)	x^6+x^5+12	GF(31^7)	x^7+x^2+7
GF(31^8)	x^8+x+22	GF(31^12)	x^12+x^3+2*x+13
GF(31^16)	x^16+x^3+12	GF(31^24)	x^24+x^11+22
GF(31^32)	x^32+x+12	GF(31^48)	x^48+x^35+11
GF(31^64)	x^64+x^5+3	GF(31^80)	x^80+x^3+12
GF(37^2)	x^2+x+5	GF(37^3)	x^3+x+13
GF(37^4)	x^4+x+2	GF(37^5)	x^5+x+5
GF(37^6)	x^6+x+20	GF(37^7)	x^7+x^2+19
GF(37^8)	x^8+x+18	GF(37^12)	x^12+x+15
GF(37^16)	x^16+x^3+15	GF(37^24)	x^24+x^11+18

GF(37^32)	x^32+x^17+18	GF(37^48)	x^48+x^11+20
GF(37^64)	x^64+x^29+22		
GF(41^2)	x^2+x+12	GF(41^3)	x^3+x+6
GF(41^4)	x^4+x+17	GF(41^5)	x^5+x^3+7
GF(41^6)	x^6+x+7	GF(41^7)	x^7+x^2+7
GF(41^8)	x^8+x^3+15	GF(41^12)	x^12+x^3+2*x+12
GF(41^16)	x^16+x^7+7	GF(41^24)	x^24+x+29
GF(41^32)	x^32+x^5+2*x+17	GF(41^48)	x^48+x^11+22
GF(43^2)	x^2+x+3	GF(43^3)	x^3+x+14
GF(43^4)	x^4+x+20	GF(43^5)	x^5+x+13
GF(43^6)	x^6+x^5+3	GF(43^7)	x^7+x+9
GF(43^8)	x^8+x^3+28	GF(43^12)	x^12+x+33
GF(43^16)	x^16+x^5+34	GF(43^24)	x^24+x^3+x+19
GF(43^32)	x^32+x^17+5	GF(43^48)	x^48+x+29
GF(47^2)	x^2+x+13	GF(47^3)	x^3+x+4
GF(47^4)	x^4+x+39	GF(47^5)	x^5+x+3
GF(47^6)	x^6+x+5	GF(47^7)	x^7+x+3
GF(47^8)	x^8+x+20	GF(47^12)	x^12+x^7+22
GF(47^16)	x^16+x^5+11	GF(47^24)	x^24+x^19+20
GF(47^32)	x^32+x^15+30	GF(47^48)	x^48+x+20
GF(47^64)	x^64+x^13+29		
GF(53^2)	x^2+x+5	GF(53^3)	x^3+x+5
GF(53^4)	x^4+x+18	GF(53^5)	x^5+x+26
GF(53^6)	x^6+x+8	GF(53^7)	x^7+x+5
GF(53^8)	x^8+x^5+21	GF(53^12)	x^12+x+12
GF(53^16)	x^16+x+8	GF(53^24)	x^24+x^11+2
GF(53^32)	x^32+x+27	GF(53^48)	x^48+x^2+x+3
GF(59^2)	x^2+x+2	GF(59^3)	x^3+x+3
GF(59^4)	x^4+x+14	GF(59^5)	x^5+x+12
GF(59^6)	x^6+x+23	GF(59^7)	x^7+x+9
GF(59^8)	x^8+x+14	GF(59^12)	x^12+x+10
GF(59^16)	x^16+x+31	GF(59^24)	x^24+x+8
GF(59^32)	x^32+x^5+55	GF(59^48)	x^48+x^11+43
GF(61^2)	x^2+x+2	GF(61^3)	x^3+x+17
GF(61^4)	x^4+x+2	GF(61^5)	x^5+x+7
GF(61^6)	x^6+x^5+6	GF(61^7)	x^7+x^2+10
GF(61^8)	x^8+x^3+2	GF(61^12)	x^12+x^11+6
GF(61^16)	x^16+x+2	GF(61^24)	x^24+x^19+2
GF(61^32)	x^32+x+17	GF(61^48)	x^48+x^17+35
GF(61^64)	x^64+4*x+7		
GF(67^2)	x^2+x+12	GF(67^3)	x^3+x+6
GF(67^4)	x^4+x+2	GF(67^5)	x^5+x+16
GF(67^6)	x^6+x^3+x+32	GF(67^7)	x^7+x+17
GF(67^8)	x^8+x^3+18	GF(67^12)	x^12+x^11+18
GF(67^16)	x^16+x+28	GF(67^24)	x^24+x^7+12
GF(67^32)	x^32+x^15+11	GF(67^48)	x^48+x^17+32
GF(67^64)	x^64+x^3+46		
GF(71^2)	x^2+x+11	GF(71^3)	x^3+x+8
GF(71^4)	x^4+x+11	GF(71^5)	x^5+x^2+4
GF(71^6)	x^6+2*x+13	GF(71^7)	x^7+x+4

GF(71^8)	x^8+x^3+22	GF(71^12)	x^12+2*x+13
GF(71^16)	x^16+x^3+47	GF(71^24)	x^24+x+56
GF(71^32)	x^32+x+11	GF(71^48)	x^48+x^5+56
GF(73^2)	x^2+x+11	GF(73^3)	x^3+x+13
GF(73^4)	x^4+x+13	GF(73^5)	x^5+x^2+13
GF(73^6)	x^6+x+5	GF(73^7)	x^7+x^2+13
GF(73^8)	x^8+x^3+53	GF(73^12)	x^12+x+29
GF(73^16)	x^16+x^13+14	GF(73^24)	x^24+x^5+15
GF(73^32)	x^32+x+47	GF(73^48)	x^48+x^3+x+34
GF(79^2)	x^2+x+3	GF(79^3)	x^3+x+9
GF(79^4)	x^4+x+3	GF(79^5)	x^5+x^2+4
GF(79^6)	x^6+x+6	GF(79^7)	x^7+x+4
GF(79^8)	x^8+x^3+6	GF(79^12)	x^12+x+48
GF(79^16)	x^16+x^5+3	GF(79^24)	x^24+x^13+30
GF(79^32)	x^32+x^9+37	GF(79^48)	x^48+x^7+68
GF(83^2)	x^2+x+2	GF(83^3)	x^3+x+7
GF(83^4)	x^4+x+22	GF(83^5)	x^5+x+12
GF(83^6)	x^6+x+34	GF(83^7)	x^7+x^3+10
GF(83^8)	x^8+x+8	GF(83^12)	x^12+x^5+13
GF(83^16)	x^16+x^3+24	GF(83^24)	x^24+x^2+x+22
GF(83^32)	x^32+x^13+19	GF(83^48)	x^48+x+53
GF(89^2)	x^2+x+6	GF(89^3)	x^3+x+19
GF(89^4)	x^4+x+27	GF(89^5)	x^5+x+3
GF(89^6)	x^6+x+6	GF(89^7)	x^7+x^2+15
GF(89^8)	x^8+x^5+24	GF(89^12)	x^12+x^7+6
GF(89^16)	x^16+x^3+15	GF(89^24)	x^24+x^11+15
GF(89^32)	x^32+x^3+3	GF(89^48)	x^48+x^7+28
GF(97^2)	x^2+x+5	GF(97^3)	x^3+x+7
GF(97^4)	x^4+x+23	GF(97^5)	x^5+x+7
GF(97^6)	x^6+x+10	GF(97^7)	x^7+x+13
GF(97^8)	x^8+x^5+5	GF(97^12)	x^12+x+68
GF(97^16)	x^16+x^5+10	GF(97^24)	x^24+x+83
GF(97^32)	x^32+x^3+5	GF(97^48)	x^48+x^11+13

Содержание