КуАпп

ОБЩЕСТВО С ОГРАНИЧЕННОЙ ОТВЕТСТВЕННОСТЬЮ
«КуАпп»

СХЕМА ЦИФРОВОЙ ПОДПИСИ НА ОСНОВЕ ФУНКЦИИ ХЭШИРОВАНИЯ БЕЗ СОХРАНЕНИЯ СОСТОЯНИЯ «Гиперикум»

Москва

2025

КуАпп

Содержание

Введение

Настоящий проект содержит описание процессов формирования и проверки постквантовой электронной цифровой подписи (ЭЦП) без сохранения состояния, реализуемой на основе функции хэширования.

Необходимость разработки настоящего проекта вызвана потребностью в реализации постквантовых схем разной степени стойкости в связи с повышением уровня развития вычислительной техники. Стойкость электронной цифровой подписи основывается на стойкости используемой хэш–функции по ГОСТ Р 34.11–2018.

Настоящий проект разработан с учетом терминологии и концепций межгосударственного стандарта ГОСТ 34.10–2018.

Примечание — Основная часть настоящего проекта дополнена приложением.

1 Область применения

Настоящий проект определяет схему постквантовой электронной цифровой подписи (ЭЦП) (далее — постквантовая цифровая подпись), процессы формирования и проверки цифровой подписи заданного сообщения (документа), передаваемых по незащищенным телекоммуникационным каналам общего пользования в системах обработки информации различного назначения.

Внедрение подписи на основе настоящего проекта повышает, по сравнению с действующей схемой цифровой подписи, уровень защищенности передаваемых сообщений от атак при помощи квантового компьютера.

Настоящий проект рекомендуется применять при создании, эксплуатации и модернизации систем обработки информации различного назначения.

2 Ссылочные документы

2.1 Нормативные ссылки

Указанные в этом разделе проекта ссылочные документы являются обязательными для их применения. Для датированных ссылок используют только указанное здесь издание. Для недатированных ссылок – последнее и актуальное издание со всеми изменениями и дополнениями.

ГОСТ 34.10-2018 — Межгосударственный стандарт. Информационная технология. Криптографическая защита информации. Процессы формирования и проверки электронной цифровой подписи.

ГОСТ Р 34.11-2018 — Межгосударственный стандарт. Информационная технология. Криптографическая защита информации. Функция хэширования.

Р 50.1.113-2016 — Рекомендации по стандартизации. Информационная технология. Криптографическая защита информации. Криптографические алгоритмы, сопутствующие применению алгоритмов электронной цифровой подписи и функции хэширования.

3 Основные понятия, термины, определения

В настоящем проекте применены следующие термины с соответствующими определениями.

1. Дополнение: Строка бит, формируемая из цифровой подписи и произвольного текстового поля. [ГОСТ 34.10–2018, пункт 3.1.1]
2. Ключ подписи: Элемент секретных данных, специфичный для субъекта и используемый только данным субъектом в процессе формирования цифровой подписи. [ГОСТ 34.10–2018, пункт 3.1.2]
3. Ключ проверки подписи: Элемент данных, математически связанный с ключом подписи и используемый проверяющей стороной в процессе проверки цифровой подписи. [ГОСТ 34.10–2018, пункт 3.1.3]
4. Параметр схемы ЭЦП: Элемент данных, общий для всех субъектов схемы цифровой подписи, известный или доступный всем этим субъектам. [ГОСТ 34.10–2018, пункт 3.1.4]
5. Подписанное сообщение: Набор элементов данных, состоящий из сообщения и дополнения, являющегося частью сообщения. [ГОСТ 34.10–2018, пункт 3.1.5]
6. Подстановка: Взаимно однозначное отображение конечного множества на себя.
7. Процесс проверки подписи: Процесс, в качестве исходных данных которого используются: подписанное сообщение, ключ проверки подписи и параметры схемы ЭЦП, результатом которого является заключение о правильности или ошибочности цифровой подписи. [ГОСТ 34.10–2018, пункт 3.1.8]
8. Процесс формирования подписи: Процесс, в качестве исходных данных которого используются: сообщение, ключ подписи и параметры схемы

ЭЦП, а в результате формируется цифровая подпись. [ГОСТ 34.10–2018, пункт 3.1.9]

9. Свидетельство: Элемент данных, представляющий соответствующее доказательство достоверности (недостоверности) подписи проверяющей стороне. [ГОСТ Р 34.10–2018, пункт 3.1.10]

10. Случайный вектор: Список, состоящий из чисел, выбранных из определенного набора таким образом, что каждое число из данного набора может быть выбрано с одинаковой вероятностью.

11. Сообщение: Строка бит произвольной конечной длины. [ГОСТ 34.10–2018, пункт 3.1.12]

12. Хэш–код: Строка бит, являющаяся выходным результатом хэш-функции. [ГОСТ 34.10–2018, пункт 3.1.13]

13. Хэш–функция: Функция, отображающая строки бит в строки бит фиксированной длины и удовлетворяющая следующим свойствам:

    а) по данному значению функции сложно вычислить исходные данные, отображаемые в это значение;

    б) для заданных исходных данных сложно вычислить другие исходные данные, отображаемые в то же значение функции;

    в) сложно вычислить какую-либо пару исходных данных, отображаемых в одно и то же значение.

    [ГОСТ 34.10–2018, пункт 3.1.14]

    Примечания

    а) Применительно к области электронной цифровой подписи свойство по перечислению 1) подразумевает, что по известной электронной цифровой подписи невозможно восстановить исходное сообщение;

свойство по перечислению 2) подразумевает, что для заданного подписанного сообщения трудно подобрать другое (фальсифицированное) сообщение, имеющее ту же электронную цифровую подпись; свойство по перечислению 3) подразумевает, что трудно подобрать какую-либо пару сообщений, имеющих одну и ту же подпись.

б) В настоящем проекте в целях сохранения терминологической преемственности с действующими отечественными нормативными документами и опубликованными научно-техническими изданиями установлено, что термины «хэш-функция», «криптографическая хэш-функция», «функция хэширования» и «криптографическая функция хэширования» являются синонимами.

14. (Электронная цифровая) подпись; ЭЦП: Строка бит, полученная в результате процесса формирования подписи. [ГОСТ 34.10–2018, пункт 3.1.15]

Примечания

а) Строка бит, являющаяся подписью, может иметь внутреннюю структуру, зависящую от конкретного механизма формирования подписи.

б) В настоящем проекте в целях сохранения терминологической преемственности с действующими отечественными нормативными документами и опубликованными научно-техническими изданиями установлено, что термины «электронная подпись», «цифровая подпись» и «электронная цифровая подпись» являются синонимами.

4 Обозначения

- $\{0,1\}$: множество $\{0,1\}$
- $\{0,1\}^s$: множество битовых векторов длины s; нумерация подвекторов и компонентов вектора осуществляется справа налево, начиная с нуля; при сложении, умножении или делении с числовыми значениями векторы рассматриваются как элементы кольца целых чисел, то есть вектору вида $z = (z_{n-1}, ..., z_0)$ сопоставляется число $z_0 + 2z_1 + ... + 2^{s-1}z_{s-1}$
- $\{0,1\}^*$: множество битовых векторов произвольной длины
- $\oplus$: операция покомпонентного сложения по модулю 2 двух двоичных векторов одинаковой размерности
- &: операция логического «И»
- $z \gg l$: операция битового сдвига вектора z на l элементов вправо
- $z \ll l$: операция битового сдвига вектора z на l элементов влево
- $z \mod x$: функция, возвращающая остаток от деления z на x; если z является вектором из $\{0,1\}^s$, то при делении z рассматривается как элемент кольца целых чисел
- $A||B$: конкатенация векторов A из $\{0,1\}^a$ и B из $\{0,1\}^b$, являющаяся вектором из $\{0,1\}^{a+b}$ и имеющая вид $A||B = (A_{a-1}, A_{a-2}, ..., A_0, B_{b-1}, B_{b-2}, ..., B_0)$.
- **Str256**: хэш-функция с длиной выхода 256 бит [ГОСТ 34.11–2018]
- **Str512**: хэш-функция с длиной выхода 512 бит [ГОСТ 34.11–2018]
- Классы настраиваемых хэш-функций, в зависимости от области определе-

ния, обозначаются как

$$\mathbf{F} : \{0,1\}^{256} \times \{0,1\}^{224} \times \{0,1\}^{256} \to \{0,1\}^{256}$$
$$\mathbf{H} : \{0,1\}^{256} \times \{0,1\}^{224} \times \{0,1\}^{512} \to \{0,1\}^{256}$$
$$\mathbf{H_s} : \{0,1\}^{256} \times \{0,1\}^{224} \times \{0,1\}^{288} \to \{0,1\}^{256}$$
$$\mathbf{Th}_l : \{0,1\}^{256} \times \{0,1\}^{224} \times \{0,1\}^{64 \cdot 256} \to \{0,1\}^{256}$$
$$\mathbf{Th}_k : \{0,1\}^{256} \times \{0,1\}^{224} \times \{0,1\}^{k \cdot 256} \to \{0,1\}^{256}$$

- **ADRS**.УстановитьУровень(x): функция, устанавливающая значение x в поле адреса «Уровень».
- **ADRS**.УстановитьДерево(x): функция, устанавливающая значение x в поле адреса «Дерево».
- **ADRS**.УстановитьНомерКлюча(x): функция, устанавливающая значение x в поле адреса «Ключ».
- **ADRS**.УстановитьТип(T): функция, устанавливающая значение типа T в поле адреса «Тип».
- **ADRS**.УстановитьСуффикс(x): функция, устанавливающая значение x в суффикс адреса.
- **ADRS**.УстановитьЦепочку(x): функция, устанавливающая значение x в поле адреса «Цепочка».
- **ADRS**.УстановитьЭлемент(x): функция, устанавливающая значение x в поле адреса «Элемент».
- **ADRS**.УстановитьВысоту(x): функция, устанавливающая значение x в поле адреса «Высота».
- **ADRS**.УстановитьИндекс(x): функция, устанавливающая значение x в поле адреса «Индекс».

- **ADRS**.ВернутьИндекс(): функция, возвращающая значение в поле адреса «Индекс».
- **ADRS**.ВернутьВысоту(): функция, возвращающая значение в поле адреса «Высота».
- Stack: структура хранения данных типа стек («последний-пришел-первый-ушел»), реализуемая разработчиком. Каждый элемент стека представляет собой пару значений $(node, height)$, где $node$ из $\{0,1\}^{256}$, а $height$ из $\{0,1\}^{8}$
- Stack.ДобавитьПару($(node, height)$): функция, добавляющая элемент $(node, height)$ на вершину стека.
- Stack.ИзъятьУзел(): функция, возвращающая значение $node$ из элемента с вершины стека и удаляющая элемент из стека.
- Stack.ВернутьВысотуУзла(): функция, возвращающая значение $height$ из элемента с вершины стека. Не удаляет элемент из стека. При вызове для пустого стека возвращает 0.
- sec_rand: криптографически стойкая функция выработки псевдослучайной последовательности.

5 Вспомогательные схемы

В настоящем разделе описываются отдельные компоненты алгоритма «Гиперикум».

5.1 Настраиваемые хэш-функции

Настраиваемая хэш-функция получает на вход открытое начальное значение **PK**.seed, контекстную информацию в виде адреса **ADRS**, входные данные M. Данный подход позволяет отделить вызовы в структуре «Гиперикум» друг от друга.

Общий вид настраиваемой функции:

$$\mathbf{Th} : \{0,1\}^{256} \times \{0,1\}^{224} \times \{0,1\}^{*} \to \{0,1\}^{256}$$

$$\mathsf{MD} \leftarrow \mathbf{Th}(\mathbf{PK}.\mathsf{seed}, \mathbf{ADRS}, M)$$

5.2 Функции вычисления хэш-кода сообщения и формирования псевдослучайных значений

«Гиперикум» использует функции **PRF**, $\mathbf{PRF}_{\mathbf{msg}}$ и $\mathbf{H}_{\mathbf{msg}}$.

PRF используется для псевдослучайного формирования элементов секретного ключа из секретного начального значения. $\mathbf{PRF}_{\mathbf{msg}}$ используется для формирования псевдослучайного значения, используемого во время сжатия исходного сообщения. $\mathbf{H}_{\mathbf{msg}}$ используется для вычисления хэш-кода сообщения.

$$\mathbf{PRF}_{\mathbf{msg}} : \{0,1\}^{256} \times \{0,1\}^{256} \times \{0,1\}^{256} \times \{0,1\}^{*} \to \{0,1\}^{256}$$

$$\mathbf{PRF} : \{0,1\}^{256} \times \{0,1\}^{256} \times \{0,1\}^{224} \to \{0,1\}^{256}$$

$$\mathbf{H}_{\mathbf{msg}} : \{0,1\}^{256} \times \{0,1\}^{256} \times \{0,1\}^{256} \times \{0,1\}^{*} \to \{0,1\}^{512}$$

5.3 Адресация

5.3.1 Общая структура адреса

Адрес представляет собой 224-битное значение. Каждому вызову хэш-функции в структуре «Гиперикум» присвоен свой уникальный адрес.

Используется 8 типов хэширований, начиная с нулевого.

0 – Хэширование в структуре WOTS+C (WOTS_HASH);

1 – Хэширование ключа проверки WOTS+C (WOTS_PK);

2 – Хэширование в деревьях Меркла в гипердереве (TREE);

3 – Хэширование в деревьях Меркла в структуре FORS+C (FORS_TREE);

4 – Хэширование корней деревьев FORS+C (FORS_ROOTS);

5 – Хэширование сообщения для подписи (SIGN_MSG_WOTS);

6 – Формирование ключей WOTS+C (KEYGEN_WOTS);

7 – Формирование ключей FORS+C (KEYGEN_FORS).

Структура адреса содержит следующий префикс (**ADRS**.**prefix**):

- Адрес уровня, кодируется $\{0,1\}^{32}$;
- Адрес дерева на этом уровне, кодируется $\{0,1\}^{64}$;
- Тип хэширования (один из восьми вышеперечисленных), кодируется $\{0,1\}^{32}$;
- Адрес экземпляра используемой ключевой пары в дереве (0^{32} для типа TREE), кодируется $\{0,1\}^{32}$.

Адрес уровня i задает значение от 1 до d. Адрес дерева определяет дерево на уровне i из множества $(h/d)^{i-1}$ деревьев. Адрес экземпляра используемой ключевой пары обозначает номер ключевой пары схем WOTS+C или FORS, либо заполняется нулями.

Далее в зависимости от типа хэширования используются различные суффиксы (**ADRS**.**suffix**).

5.3.2 Хэширование в структуре WOTS+C

При использовании типа со значением 0 (WOTS_HASH) суффикс адреса имеет вид:

- Адрес цепочки, кодируется $\{0,1\}^{32}$;
- Адрес элемента цепочки, кодируется $\{0,1\}^{32}$.

Адрес цепочки i представляет собой значение от 0 до 63. Адрес элемента цепочки j представляет собой значение от 0 до 15.

На рисунке 1 представлена структура адреса с типом 0 («Хэширование в структуре WOTS+C»).

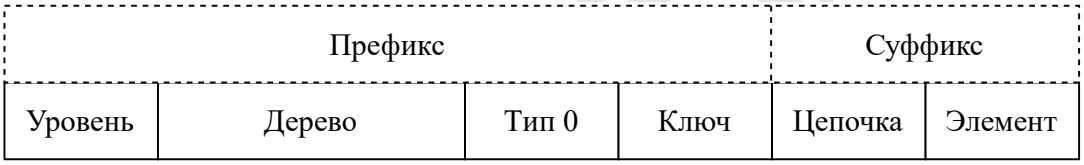

Рисунок 1 – *Адрес с типом «Хэширование в структуре WOTS+C»*

5.3.3 Хэширование ключа проверки WOTS+C

При использовании типа со значением 1 (WOTS_PK) адрес необходимого вызова функции хэширования задается типом хэширования и номером ключевой пары WOTS+C. Суффикс не используется и заполняется нулями.

- Пустой суффикс, кодируется 0^{64}.

На рисунке 2 представлена структура адреса с типом 1 («Хэширование ключа проверки WOTS+C»).

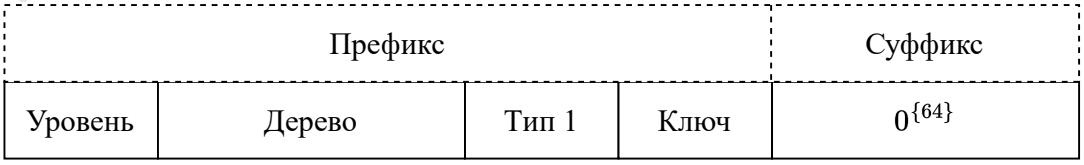

Рисунок 2 – *Адрес с типом «Хэширование ключа проверки WOTS+C»*

5.3.4 Хэширование в деревьях Меркла в гипердереве

При использовании типа со значением 2 (TREE) суффикс используется для определения хэширования в заданном дереве Меркла. В адресе определяется

высота, на которой осуществляется хэширование - значение от 0 до h/d. На уровне i индекс узла варьируется в диапазоне 0 до $2^i - 1$. Суффикс имеет вид:

- Высота в рамках дерева, кодируется $\{0,1\}^{32}$;
- Индекс узла на заданной высоте, кодируется $\{0,1\}^{32}$.

Адрес ключевой пары в данном типе адреса не используется и заполняется нулями.

На рисунке 3 представлена структура адреса с типом 2 («Хэширование в деревьях Меркла в гипердереве»).

Префикс				Суффикс	
Уровень	Дерево	Тип 2	$0^{\{32\}}$	Высота	Индекс

Рисунок 3 – *Адрес с типом «Хэширование в деревьях Меркла в гипердереве»*

5.3.5 Хэширование в деревьях Меркла в структуре FORS

При использовании типа со значением 3 (FORS_TREE) определяется высота, на которой осуществляется хэширование - значение от 0 до b. На уровне i индекс узла варьируется в диапазоне 0 до $2^i - 1$. Суффикс имеет вид:

- Высота в рамках дерева, кодируется $\{0,1\}^{32}$;
- Индекс узла на заданной высоте, кодируется $\{0,1\}^{32}$.

На рисунке 4 представлена структура адреса с типом 3 («Хэширование в деревьях Меркла в структуре FORS»).

Префикс				Суффикс	
Уровень	Дерево	Тип 3	Ключ	Высота	Индекс

Рисунок 4 – *Адрес с типом «Хэширование в деревьях Меркла в структуре FORS»*

5.3.6 Хэширование корней деревьев FORS

При использовании типа со значением 4 (FORS_ROOTS) адрес необходимого вызова функции хэширования задается типом хэширования и номером ключевой пары FORS. Суффикс не используется и заполняется нулями.

• Пустой суффикс, кодируется 0^{64}.

На рисунке 5 представлена структура адреса с типом 4 («Хэширование корней деревьев FORS»).

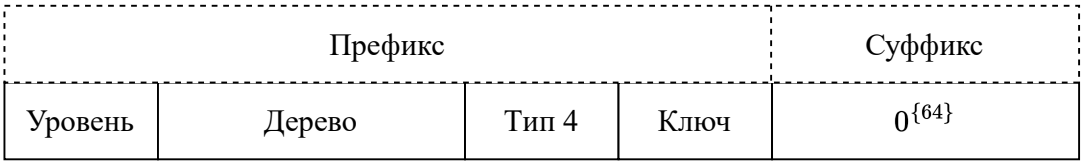

Рисунок 5 – *Адрес с типом «Хэширование корней деревьев FORS»*

5.3.7 Хэширование сообщения для подписи

При использовании типа со значением 5 (SIGN_MSG_WOTS) адрес необходимого вызова функции хэширования задается типом хэширования и номером ключевой пары WOTS+C. Суффикс не используется и заполняется нулями.

• Пустой суффикс, кодируется 0^{64}.

На рисунке 6 представлена структура адреса с типом 5 («Хэширование сообщения для подписи»).

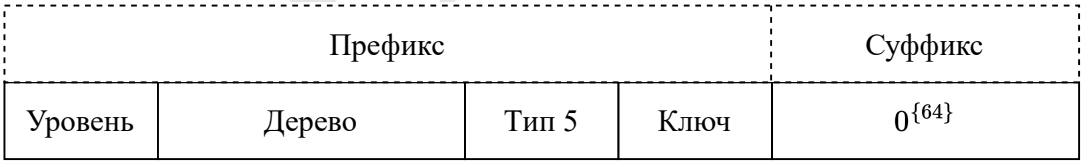

Рисунок 6 – *Адрес с типом «Хэширование сообщения для подписи»*

5.3.8 Формирование ключей WOTS+C

При использовании типа со значением 6 (KEYGEN_WOTS) определяется индекс узла в диапазоне 0 до $2^{h/d} - 1$, для которого формируется ключ. Суффикс адреса имеет следующую структуру:

• Адрес цепочки, кодируется $\{0, 1\}^{32}$;

• Адрес элемента, нулевой блок 0^{32}.

На рисунке 7 представлена структура адреса с типом 6 («Формирование ключей WOTS+C»).

Рисунок 7 – *Адрес с типом «Формирование ключей WOTS+C»*

5.3.9 Формирование ключей FORS

При использовании типа со значением 7 (KEYGEN_FORS) определяется индекс узла в диапазоне 0 до $2^b - 1$, для которого формируется ключ. Суффикс адреса имеет следующую структуру:

- Адрес высоты, нулевой блок 0^{32};
- Индекс узла на нулевой высоте, кодируется $\{0,1\}^{32}$.

На рисунке 8 представлена структура адреса с типом 7 («Формирование ключей FORS»).

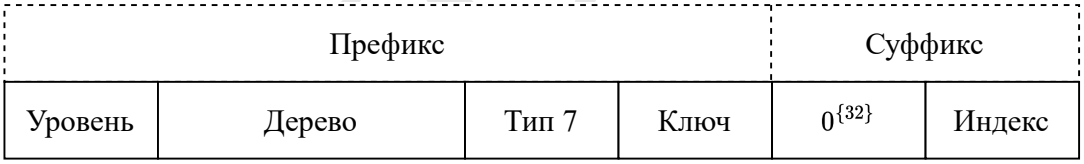

Рисунок 8 – *Адрес с типом «Формирование ключей FORS»*

5.4 Особенности программной реализации

5.4.1 Байтовое представление строки

Пусть X — байтовая строка длины n. Тогда $X[i]$ для $i \in \{0,\ldots,n-1\}$ обозначает i-й элемент строки X. Если X является массивом из m n-байтовых строк, то $X[i]$ для $i \in \{0,\ldots,m-1\}$ обозначает i-ю n-байтовую строку в X, а X интерпретируется как $m \cdot n$-байтовая строка

$$X[0]\|X[1]\|\ldots\|X[m-1].$$

Байтовая строка представляется как целое число в представлении «big-endian» (старший байт числа хранится в памяти по младшему адресу, а младший байт — по старшему). В этом случае байтовая строка X длины n преобразуется в

целое число x по формуле:

$$x = X[0] \cdot 256^{n-1} + X[1] \cdot 256^{n-2} + \cdots + X[n-2] \cdot 256 + X[n-1].$$

Аналогично целое число x можно преобразовать в байтовую строку X длины n, путем поиска необходимых коэффициентов $x_0, x_1, \ldots, x_{n-2}, x_{n-1} \in \{0, 255\}$ таких, что:

$$x = x_0 \cdot 256^{n-1} + x_1 \cdot 256^{n-2} + \cdots + x_{n-2} \cdot 256 + x_{n-1}$$

после чего байтовая строка X определяется как конкатенация:

$$X = x_0 x_1 \ldots x_{n-2} x_{n-1}$$

Замечание 1. Запись битовой строки $z = (z_{s-1}, z_{s-2}, \ldots, z_1, z_0) = z_{s-1} z_{s-2} \ldots z_1 z_0$ эквивалентна $z = z[0]z[1] \ldots z[s-2]z[s-1]$ в представлении «big-endian», где $z \in \{0,1\}^s$. Аналогично для байтовой строки.

5.4.2 Хэш-функция «Стрибог»

Пусть **Str** : $\{0,1\}^* \to \{0,1\}^n$ — это хэш-функция ГОСТ Р 34.11-2012 «Стрибог», где n — это размер выходного хэш-кода. Пусть **H** настраиваемая хэш-функция, принимающая на вход набор строк. Тогда верно следующее:

$$\begin{aligned}&\mathbf{H}(X, Y, M1 \| M2) = \mathbf{Str}(M2 \| M1 \| Y \| X) = \\ &= \mathbf{Str}(M2[0] \ldots M2[n_2 - 1] \,\|\, M1[0] \ldots M1[n_1 - 1] \,\| \\ &\| \, Y[0] \ldots Y[s_2 - 1] \,\|\, X[0] \ldots X[s_1 - 1]).\end{aligned}$$

Функция **Str** начинает обрабатывать полученную строчку справа налево, начиная с $X[s_1 - 1]$. Строки $X, Y, M1$ и $M2$ записаны в представлении «big-endian».

5.4.3 Вспомогательные алгоритмы

Алгоритм 1 – функция, преобразующая байтовую строку X длиной n в целое число. Алгоритм 2 – функция, преобразующая целое число x в байтовую строку длиной n.

Алгоритм 1: Функция преобразования байтовой строки в целое число.; $toInt(X, n)$

Входные значения : X — входная строка байт, n — длина строки X

Выходные значения : x — целое неотрицательное число

$x := 0$

цикл $i := 0; i < n; i + 1$ **выполнять**

  $x := x \cdot 256 + X[i]$

вернуть x.

Алгоритм 2: Функция преобразования целого числа в байтовую строку; $toByte(x, n)$

Входные значения : x — целое число, n — длина выходной строки

Выходные значения : S — байтовая строка длина n

$total := x$

цикл $i := 0; i < n; i + 1$ **выполнять**

  $S[n - 1 - j] := total \bmod 256$

  $total := total \gg 8$

вернуть S.

Алгоритм 3 используется в алгоритмах 20 и 21.

Алгоритм 3: Функция извлечения индексов из строки; $indexes(M, b, i)$

Входные значения : b, i — целые неотрицательные числа, M — входная строка байт

Выходные значения : idx — целое неотрицательное число

$idx := 0$

цикл $j := i \cdot b; i < (i + 1) \cdot b; j + 1$ **выполнять**

  $byte := \lfloor j/8 \rfloor$

  $bit := 7 - (j \bmod 8)$

  $idx := 2 \cdot idx + \big((M[byte] \gg bit) \bmod 2\big)$

вернуть idx.

5.5 Одноразовая подпись WOTS+C

В настоящем разделе описана схема одноразовой подписи WOTS+C — составная часть алгоритма «Гиперикум». Одноразовая схема подписи WOTS+C используется для подписи корней деревьев.

5.5.1 WOTS+C: Функция построения цепочки

Функция построения цепочки вычисляет несколько итераций функции хэширования F на n-байтных значениях, используя адреса **ADRS** и открытое начальное значение **PK**.seed.

Алгоритм 4: Функция построения цепочки; chain(**PK**.seed, **ADRS**, X, j, k)

Входные значения : Открытое начальное значение **PK**.seed, адрес **ADRS**, входная строка X, начальный индекс цепочки j, число шагов в цепочке k.

Выходные значения : Элемент в цепочке с индексом $j + k$.

если $k \leq 0$ **тогда**
  вернуть X
если $(j + k) > (15)$ **тогда**
  вернуть Ошибка
Element := chain(**PK**.seed, **ADRS**, $X, j, k - 1$)
ADRS.УстановитьЭлемент$(j + k - 1)$
Element := **F**(**PK**.seed, **ADRS**, Element)
вернуть Element.

5.5.2 Выработка ключей WOTS+C

Алгоритмы выработки секретного ключа и ключа проверки принимают в качестве входных данных: открытое начальное значение **PK**.seed, секретное начальное значение **SK**.seed и адрес **ADRS** данного экземпляра WOTS+C. Секретный ключ $SK = (sk_0, ..., sk_{63})$ представляет собой 64 значения по 256 бит, вычисляемых с помощью функции **PRF**. Для вычисления блоков открытого ключа необходимо выполнить 15 итераций хэширования над блоками секретного ключа с помощью функции вычисления цепочки chain, а затем вычислить хэш-код от результирующих 64 блоков. Описание работы chain приведено в алгоритме 4.

Описание алгоритма выработки секретного ключа представлено в алгоритме 5, блок-схема алгоритма представлена на рисунке 9. Описание алгоритма выработки ключа проверки представлено в алгоритме алгоритме 6, блок-схема алгоритма представлена на рисунке 10.

Алгоритм 5: Функция выработки секретного ключа WOTS+C; WOTS.SkGen (**SK**.seed, **PK**.seed, **ADRS**)

Входные значения : Секретное начальное значение **SK**.seed, открытое начальное значение **PK**.seed, адрес **ADRS**.

Выходные значения : Секретный ключ SK.

ADRS.УстановитьТип(KEYGEN_WOTS)
ADRS.УстановитьЭлемент(0)
цикл $i := 0; i < 64; i + 1$ **выполнять**
    ADRS.УстановитьЦепочку(i)
    $\mathsf{sk}_i :=$ **PRF**(**SK**.seed, **PK**.seed, **ADRS**)
$\mathsf{SK} := (\mathsf{sk}_0, \ldots, \mathsf{sk}_{63})$
Вернуть(SK).

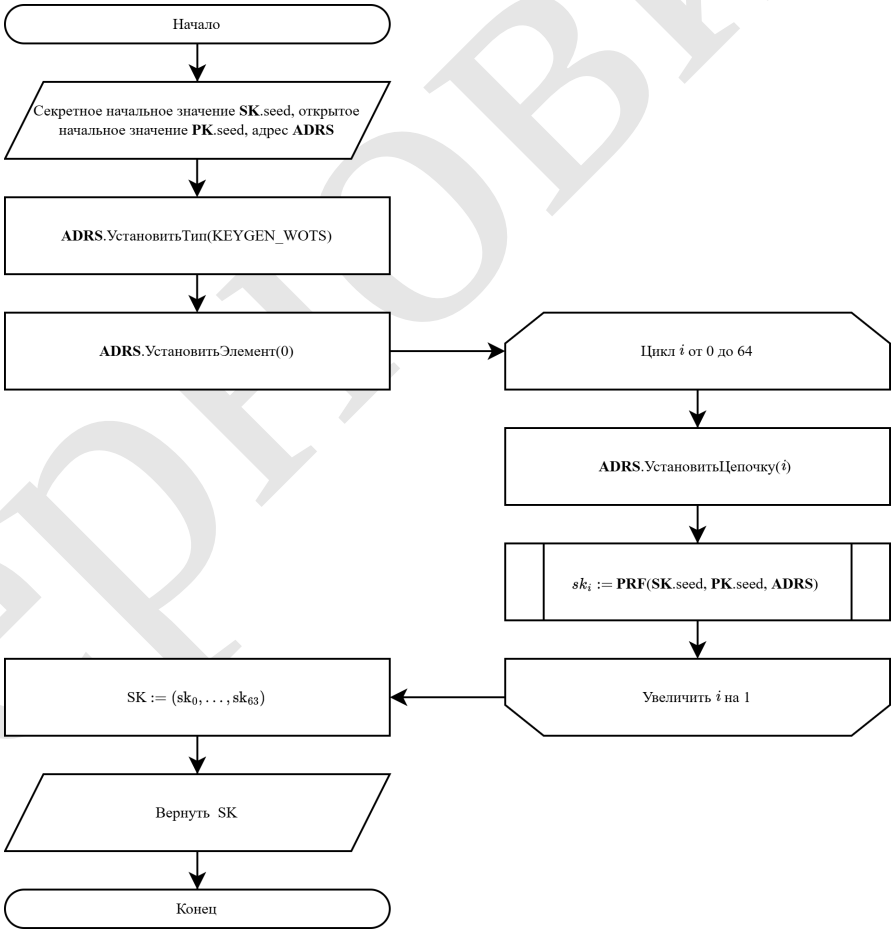

Рисунок 9 – *Функция выработки секретного ключа WOTS+C*

Алгоритм 6: Функция выработки открытого ключа WOTS+C; WOTS.PkGen (**SK**.seed, **PK**.seed, **ADRS**)

Входные значения : Секретное начальное значение **SK**.seed, открытое начальное значение **PK**.seed, адрес **ADRS**.

Выходные значения : Открытый ключ PK.

1 SK := WOTS.SkGen(**SK**.seed, **PK**.seed, **ADRS**)
2 **ADRS**.УстановитьТип(WOTS_HASH)
3 **цикл** $i := 0; i < 64; i + 1$ **выполнять**
4 **ADRS**.УстановитьЦепочку(i)
5 $\mathsf{pk}_i := \mathsf{chain}(\mathbf{PK}.\mathsf{seed}, \mathbf{ADRS}, \mathsf{sk}_i, 0, 15)$
6 $\mathsf{PK}_{tmp} := (\mathsf{pk}_0, \ldots, \mathsf{pk}_{63})$
7 **ADRS**.УстановитьТип(WOTS_PK)
8 **ADRS**.УстановитьСуффикс(0)
9 $\mathsf{PK} := \mathbf{Th}_l(\mathbf{PK}.\mathsf{seed}, \mathbf{ADRS}, \mathsf{PK}_{tmp})$
10 Вернуть(PK).

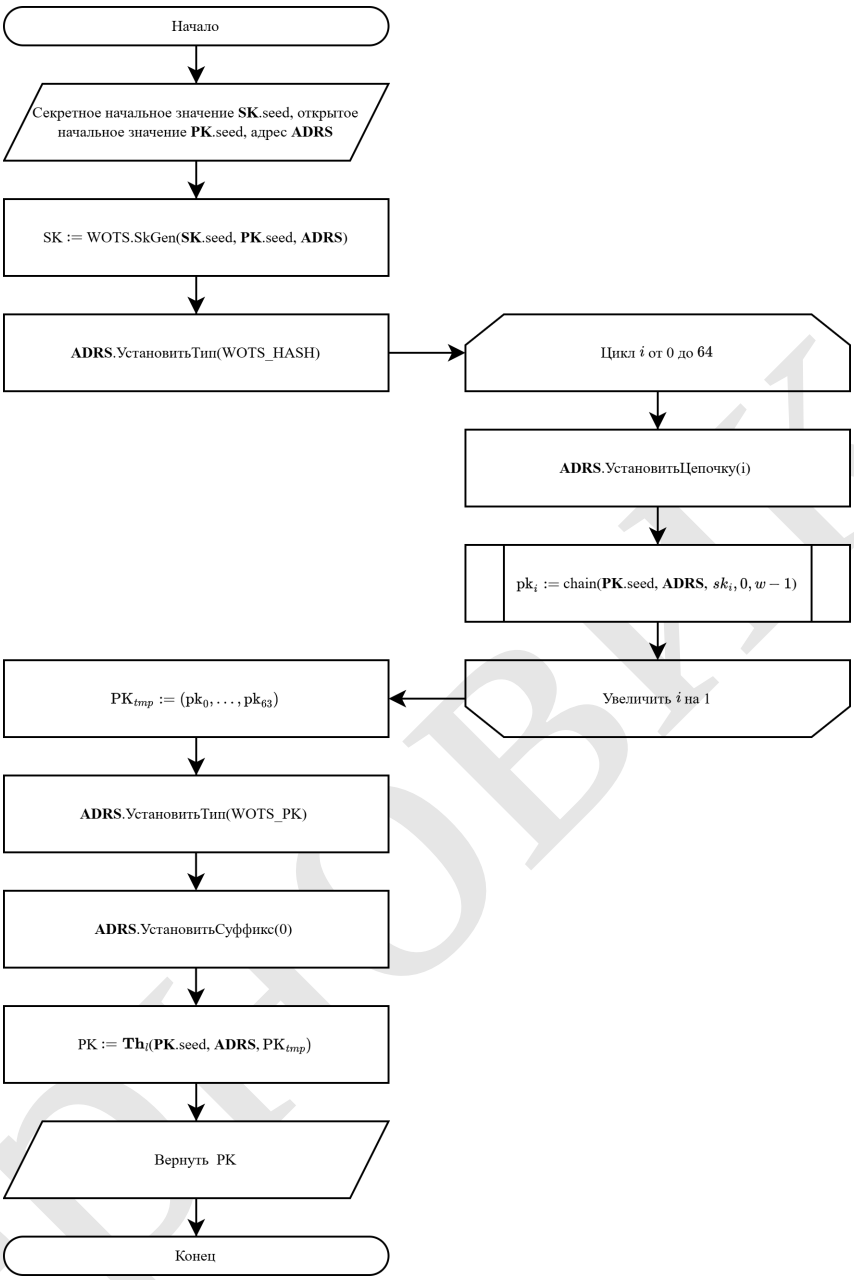

Рисунок 10 – *Функция выработки открытого ключа WOTS+C*

5.5.3 Алгоритм подписи WOTS+C

Для вычисления подписи WOTS+C для n-битного сообщения M вычисляется 16-ичное представление $M : M = (M_1, ..., M_{64})$, $M_i \in \{0, \ldots, 15\}$. Затем вычисляется хэш $d = \mathbf{H}(\mathbf{PK}.\mathsf{seed}, T, s||M)$, где s — 32-битный счетчик. Значение s инкрементируется начиная с 0 до тех пор, пока для отображения результирующего хэш-кода d в элементы цепочки $a_1, \ldots, a_{64} \in [16]$ не будет выполняться равенство $\sum_{i=1}^{64} a_i = S_{w,n}$. Функция вычисления 16-ичного представления сообщения в виде массива байт описана в алгоритме 7. Выполнение отображения d в элементы цепочки описано в алгоритме 8.

Алгоритм выработки подписи WOTS+C принимает на вход сообщение M, открытое начальное значение **PK**.seed, секретное начальное значение **SK**.seed и адрес **ADRS**. Полученная подпись σ представляет собой кортеж $(\sigma_0, \ldots, \sigma_{63}, s)$ из 65 блоков, где первые 64 — это значения по 256 бит, полученные при помощи функции chain, а последний блок размером 32 бита представляет собой значение счетчика из функции hash-convert. Описание формирования подписи представлено в алгоритме 9,блок-схема алгоритма представлена на рисунке 11.

Алгоритм 7: Функция вычисления 16-ичного представления; convert-16(M)

Входные значения : Сообщение M

Выходные значения : 16-ичное представление сообщения M

```
1  in := 0
2  out := 0
3  total := 0
4  bits := 0
5  цикл out := 0; out < 64; out + 1 выполнять
6      если bits == 0 тогда
7          total := M[in]
8          in := in + 1
9          bits := bits + 8
10     bits := bits − 4
11     basew[out] := (total ≫ bits)&(15)
12 Вернуть(basew).
```

Алгоритм 8: Функция вычисления w-ичного представления хэш-кода; hash-convert(**PK**.seed, **ADRS**, M, $S_{w,n}$)

Входные значения : Открытое начальное значение **PK**.seed, адрес **ADRS**, сообщение M, целевая сложность $S_{w,n}$

Выходные значения : 16-ичное представление хэш-кода сообщения M, начальное значение s

ADRS.УстановитьТип(SIGN_MSG_WOTS)
ADRS.УстановитьСуффикс(0)
do $:= 1$
$s := 0$
Пока *do == 1* **выполнять:**
    $d :=$ **H**$_{\mathbf{s}}$(**PK**.seed, **ADRS**, $s||M$)
    basew $:=$ convert-16(d)
    если $\sum_{i=0}^{63}$ basew[i] $== S_{w,n}$ **тогда**
        do $:= 0$
    иначе
        $s := s + 1$
Вернуть(basew, s).

Алгоритм 9: Алгоритм подписи WOTS+C; WOTS.Sign(M, **SK**.seed, **PK**.seed, **ADRS**)

Входные значения : Сообщение M, секретное начальное значение **SK**.seed, открытое начальное значение **PK**.seed, адрес **ADRS**

Выходные значения : Подпись σ

Начать Подготовка вычисления подписи
    $\{B, s\} := \{(b_0, \ldots, b_{63}), s\} =$ hash-convert(**PK**.seed, **ADRS**, M, $S_{w,n}$), где $S_{w,n}$ — параметр подписи
SK $:=$ WOTS.SkGen (**SK**.seed, **PK**.seed, **ADRS**)
ADRS.УстановитьТип(WOTS_HASH)
цикл $i := 0; i < 64; i + 1$ **выполнять**
    ADRS.УстановитьЦепочку(i)
    $\sigma_i :=$ chain(**PK**.seed, **ADRS**, sk$_i$, $0, b_i$)
$\sigma := (\sigma_0, \ldots, \sigma_{63}, s)$
Вернуть(σ).

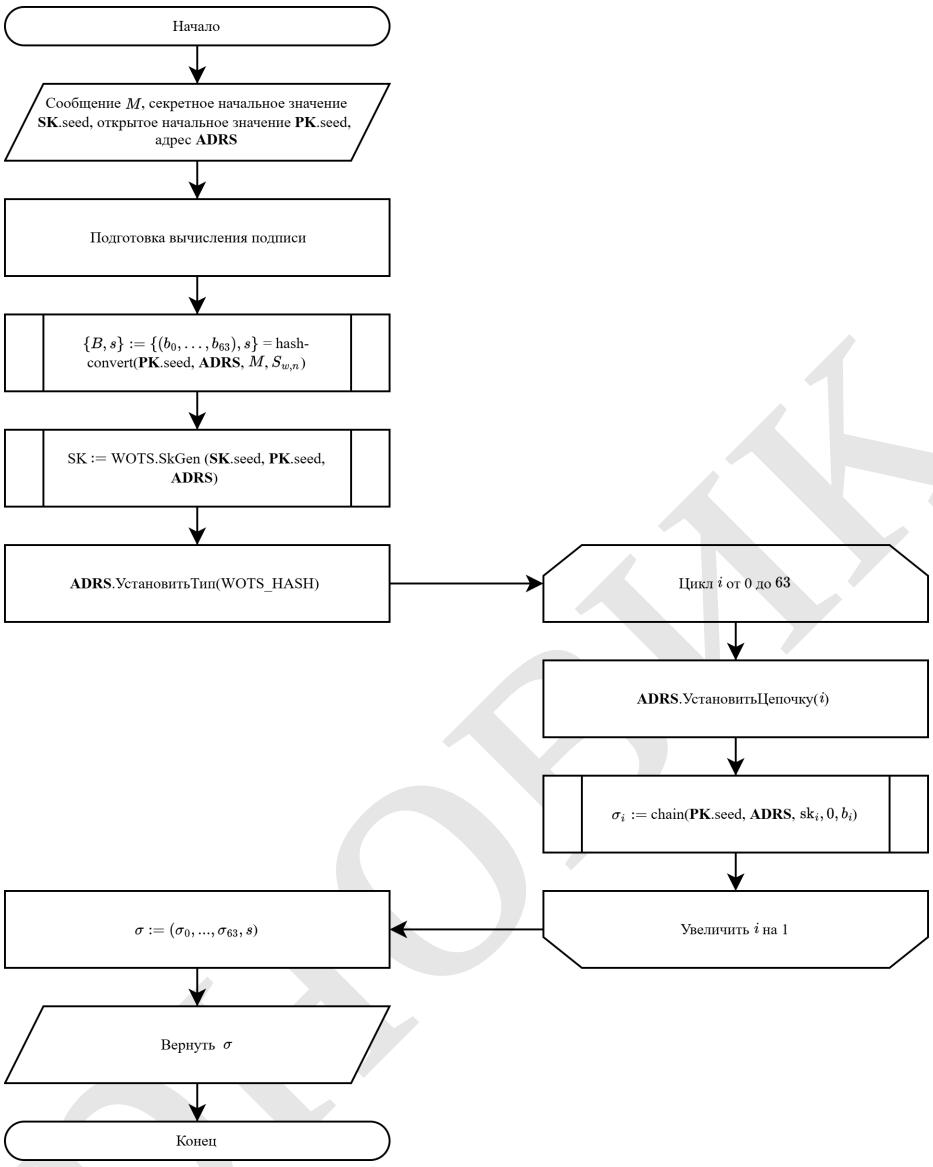

Рисунок 11 – *Алгоритм подписи WOTS+C*

5.5.4 Алгоритм вычисления ключа проверки из подписи WOTS+C

Для проверки подписи необходимо вычислить все цепочки из подписи до конца и сравнить вычисленные элементы с соответствующими элементами ключа проверки. В алгоритме «Гиперикум» не выполняется проверка подписи схемы WOTS+C. Вместо этого используется вычисленный элемент ключа проверки для дальнейшей проверки подписи «Гиперикум». Для этого на вход алгоритма 10 принимаются следующие значения: сообщение M, подпись σ, открытое начальное значение **PK**.seed, адрес **ADRS**. Полученный элемент ключа $\mathsf{PK}' = \mathbf{Th}_l(\mathbf{PK}.\mathsf{seed}, \mathbf{ADRS}, \mathsf{PK}'_{tmp})$ представляет собой набор из 64 значений по 256 бит, полученные при помощи функции chain. Описание вычисления эле-

мента ключа проверки из подписи приведено в алгоритме 10, блок-схема алгоритма представлена на рисунке 12.

Алгоритм 10: Алгоритм вычисления ключа проверки из подписи WOTS+C; WOTS.pkFromSig($M, \sigma,$ **PK**.seed, **ADRS**)

Входные значения : Сообщение M, подпись σ, открытое начальное значение **PK**.seed, адрес **ADRS**.

Выходные значения : Вычисленный из подписи элемент ключа проверки pk'.

1 **Начать** Обработка сообщения
2 **ADRS**.УстановитьТип(SIGN_MSG_WOTS)
3 **ADRS**.УстановитьСуффикс(0)
4 $d := \mathbf{H_s}(\mathbf{PK}.\mathsf{seed}, \mathbf{ADRS}, (\sigma[64] = s)||M)$
5 $B := (b_1, \ldots, b_{64}) =$ convert-16(d)
6 **если** $\sum_{i=1}^{64} b_i \neq S_{w,n}$ **тогда**
7 Вернуть(Ошибка)
8 **Начать** Вычисление pk'
9 **ADRS**.УстановитьТип(WOTS_HASH)
10 **цикл** *i:=0; i<64; i+1* **выполнять**
11 **ADRS**.УстановитьЦепочку(i)
12 $\mathsf{pk}'_i :=$ chain$(\mathbf{PK}.\mathsf{seed}, \mathbf{ADRS}, \sigma_i, b_i, 15 - b_i)$
13 $\mathsf{PK}'_{tmp} := (\mathsf{pk}'_0, \ldots, \mathsf{pk}'_{63})$
14 **ADRS**.УстановитьТип(WOTS_PK)
15 $\mathsf{PK}' := \mathbf{Th}_l(\mathbf{PK}.\mathsf{seed}, \mathbf{ADRS}, \mathsf{PK}'_{tmp})$
16 Вернуть(PK').

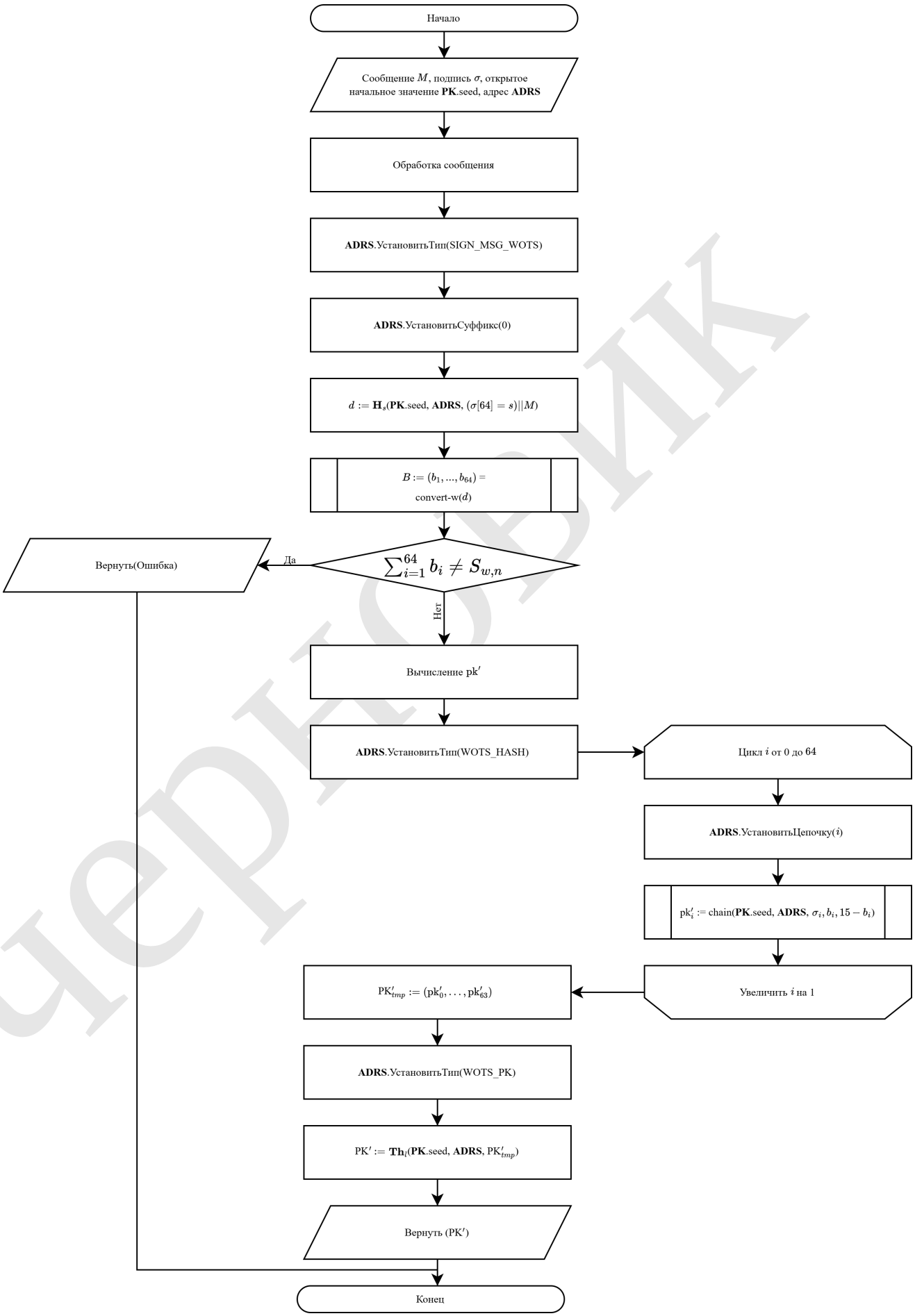

Рисунок 12 – *Алгоритм вычисления ключа проверки из подписи WOTS+C*

5.6 Расширенная схема подписи Меркла

В данном разделе описывается расширенная схема подписи Меркла (eXtended Merkle Signature Scheme, XMSS) [2].

5.6.1 Параметры XMSS

В схеме XMSS используется следующий набор параметров:

- h' - высота дерева;
- n - размер подписываемого сообщения, равный 256 бит.

Дерево Меркла строится из $2^{h'}$ листьев.

5.6.2 Функции построения дерева treehash

Для алгоритмов выработки ключа проверки требуется подфункция построения дерева.

Для вычисления внутренних n-битовых узлов дерева Меркла алгоритм treehash (алгоритм 11) принимает секретное начальное значение **SK**.seed, открытое начальное значение **PK**.seed, число s (начальный индекс), число z (высота целевого узла) и адрес **ADRS**. На выходе алгоритм возвращает корневой узел дерева высотой z с крайним левым листом, представляющим собой ключ проверки WOTS+C с индексом s.

Для нахождения корня дерева вычисления должны начинаться с листьев, находящихся на нулевой высоте.

Для реализации алгоритма 11 используется структура данных типа «стек» (последний-пришел-первый-ушел), содержащий до $(z - 1)$ пар элементов, где первое значение элемента представляет собой блок размером 256 бит, а второе значение элемента - высоту, на которой находится блок. Размер поля в элементе для хранения высоты равен 8 битам. Описание вычисления внутренних узлов дерева Меркела приведено в алгоритме 11, блок-схема алгоритма представлена на рисунке 13.

Алгоритм 11: Алгоритм вычисления внутренних узлов дерева Меркла; XMSS.treehash (**SK**.seed, **PK**.seed, s, z, **ADRS**)

Входные значения : Секретное начальное значение **SK**.seed, открытое начальное значение **PK**.seed, начальный индекс s, высота целевого узла z, адрес **ADRS**.

Выходные значения : Целевой узел

1 **если** $s \neq 0 \mod 2^z$ **тогда**
2 Вернуть(Ошибка)
3 **цикл** $i := 0; i < 2^z; i + 1$ **выполнять**
4 **ADRS**.УстановитьНомерКлюча($s + i$);
5 $node :=$ WOTS.PkGen (**SK**.seed, **PK**.seed, **ADRS**);
6 **ADRS**.УстановитьТип(TREE);
7 **ADRS**.УстановитьНомерКлюча(0);
8 **ADRS**.УстановитьВысоту(1);
9 **ADRS**.УстановитьИндекс($s + i$);
10 **Пока** Stack.ВернутьВысотуУзла() == **ADRS**.ВернутьВысоту() **выполнять:**
11 **ADRS**.УстановитьИндекс(**ADRS**.ВернутьИндекс()$\gg 1$);
12 $node :=$ **H**(**PK**.seed, **ADRS**, (Stack.ИзъятьУзел()$||node$));
13 **ADRS**.УстановитьВысоту(**ADRS**.ВернутьВысоту()+1);
14 Stack.ДобавитьПару($node$, **ADRS**.ВернутьВысоту());
15 Вернуть(Stack.ИзъятьУзел()).

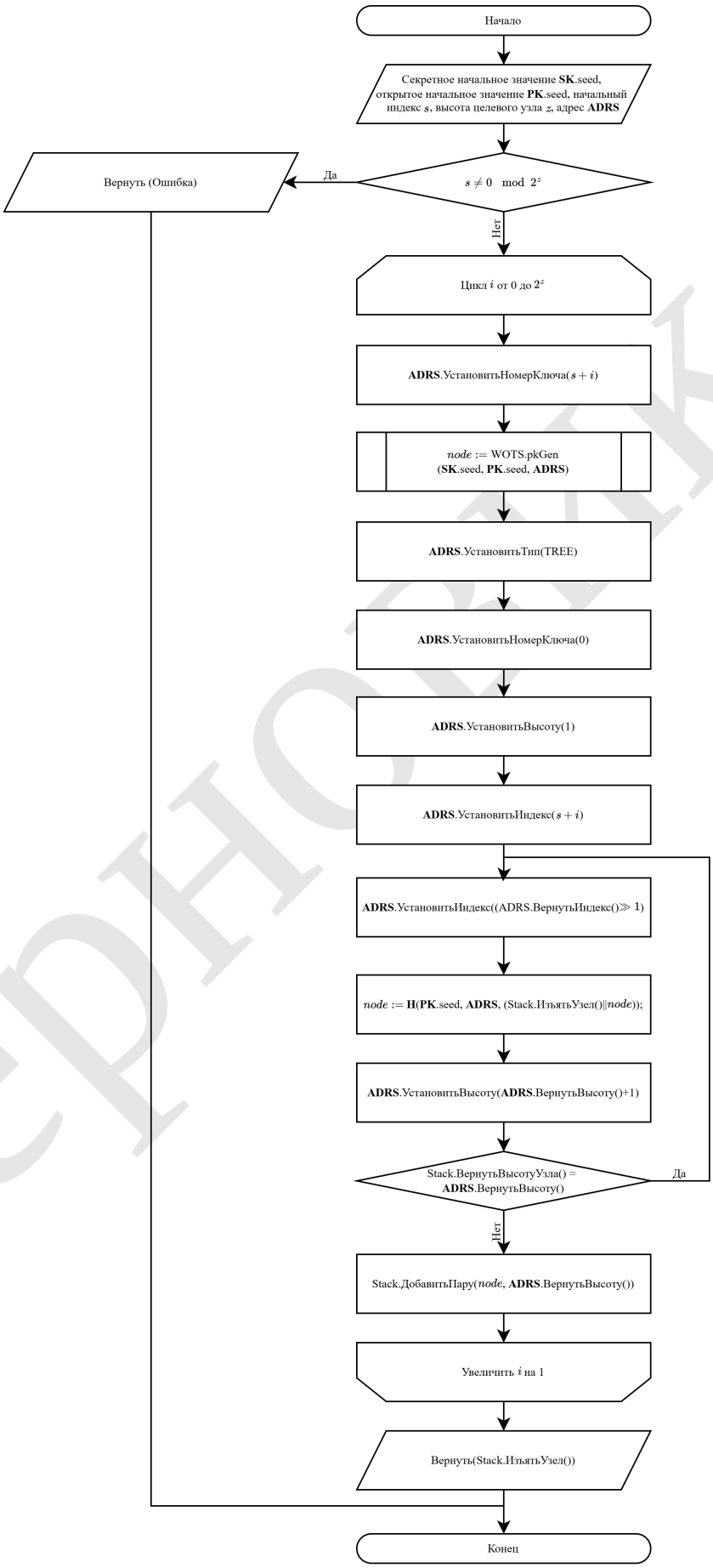

Рисунок 13 – *Алгоритм вычисления внутренних узлов дерева Меркла*

5.6.3 Выработка ключа проверки XMSS

В контексте «Гиперикум» ключ проверки XMSS PK является корнем дерева Меркла. Алгоритм вычисления ключа проверки XMSS принимает на вход секретное начальное значение **SK**.seed, открытое начальное значение **PK**.seed, адрес **ADRS**. Полученный открытый ключ вычисляется с помощью функции XMSS.treehash. Описание вычисления ключа проверки XMSS приведено в алгоритме 12, блок-схема алгоритма представлена на рисунке 14.

Алгоритм 12: Алгоритм вычисления ключа проверки XMSS; XMSS.PKgen (**SK**.seed, **PK**.seed, **ADRS**)

Входные значения : Секретное начальное значение **SK**.seed, открытое начальное значение **PK**.seed, адрес **ADRS**

Выходные значения : Открытый ключ

1 pk := XMSS.treehash (**SK**.seed, **PK**.seed, 0, h', **ADRS**);

2 Вернуть(pk)

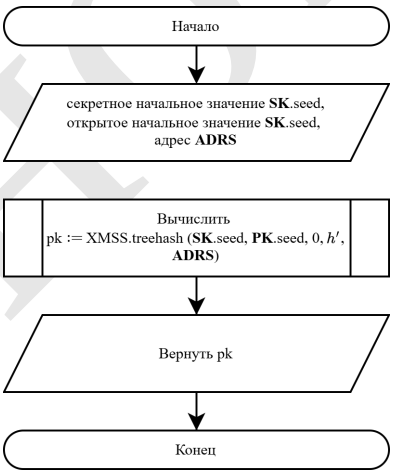

Рисунок 14 – *Алгоритм вычисления ключа проверки XMSS*

5.6.4 Формирование подписи XMSS

Для вычисления подписи XMSS сообщения M в контексте «Гиперикум» используется секретное начальное значение **SK**.seed, открытое начальное значение **PK**.seed, индекс idx пары ключей WOTS+C и адрес **ADRS**, определяющий конкретное XMSS дерево. Сначала формируется подпись WOTS+C сообщения M, используя лист XMSS дерева с индексом idx. Далее вычисляется путь аутентификации. Путь аутентификации представляет собой массив из h' блоков по 256 бит. В нем содержатся необходимые соседние узлы на пути вычисления

вершины.

Описание вычисления подписи XMSS приведено в алгоритме 13, блок-схема алгоритма представлена на рисунке 15.

Алгоритм 13: Алгоритм вычисления подписи XMSS;
XMSS.Sign $(M, \textbf{SK}.\text{seed}, \textbf{PK}.\text{seed}, \text{idx}, \textbf{ADRS})$

Входные значения : Сообщение M, секретное начальное значение **SK**.seed, открытое начальное значение **PK**.seed, индекс idx, адрес **ADRS**.

Выходные значения : Подпись SIG_XMSS.

1 **Начать** Вычисление пути аутентификации
2 **цикл** $j := 0; j < h'; j + 1$ **выполнять**
3 $k := (idx \gg j) \oplus 1$;
4 AUTH[j] := XMSS.treehash $(\textbf{SK}.\text{seed}, \textbf{PK}.\text{seed}, k * 2^j, j, \textbf{ADRS})$;
5 **ADRS**.УстановитьТип(WOTS_HASH);
6 **ADRS**.УстановитьНомерКлюча(idx);
7 sig := WOTS.Sign $(M, \textbf{SK}.\text{seed}, \textbf{PK}.\text{seed}, \textbf{ADRS})$;
8 SIG_XMSS := sig || AUTH;
9 Вернуть(SIG_XMSS).

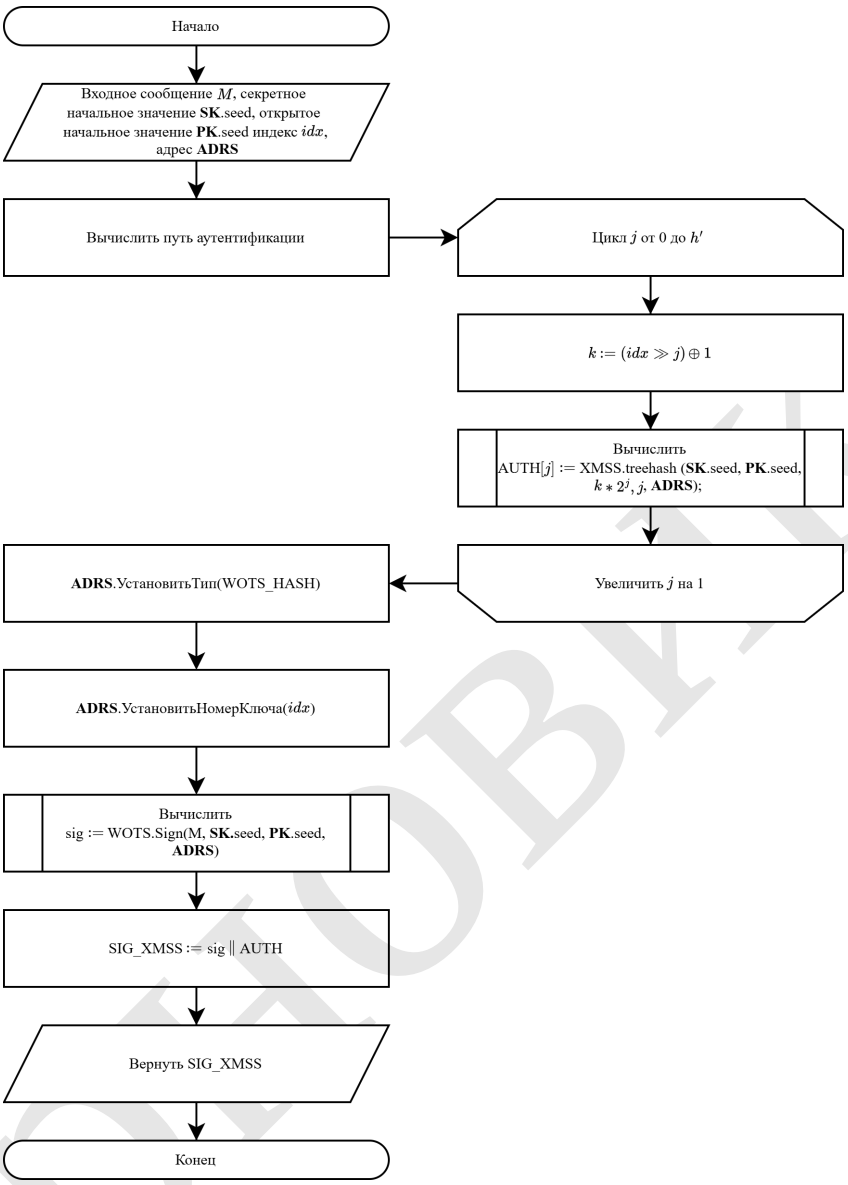

Рисунок 15 – *Алгоритм вычисления подписи XMSS*

5.6.5 Вычисление ключа проверки XMSS из подписи

Схема «Гиперикум» использует неявную проверку подписей XMSS. Подпись XMSS используется для вычисления ключа проверки XMSS. Данное значение используется в дальнейших вычислениях. Чтобы вычислить корень дерева, алгоритм принимает на вход следующие значения: сообщение M, XMSS подпись SIG_XMSS, открытое значение **PK**.seed, индекс idx, адрес **ADRS**. Полученный корень дерева XMSS — это последовательно вычисленный хэш-код из предшествующих узлов. Описание вычисления корня дерева XMSS, предложенного в [1], приведено в алгоритме 14, блок-схема алгоритма представлена на рисунке 16.

Алгоритм 14: Алгоритм вычисления корня дерева XMSS; XMSS.pkFromSig $(M, \text{SIG\_XMSS}, \textbf{PK}.\text{seed}, idx, \textbf{ADRS})$

Входные значения : Сообщение M, XMSS подпись SIG_XMSS, открытое значение **PK**.seed, индекс idx, адрес **ADRS**.

Выходные значения : Корень XMSS дерева.

Начать Вычисление ключа проверки WOTS+C из подписи
- **ADRS**.УстановитьНомерКлюча(idx);
- Положить в переменные sig и AUTH подпись и путь аутентификации соответственно из распакованной SIG_XMSS. $node[0]$ = WOTS.pkFromSign $(sig, M, \textbf{PK}.\text{seed}, \textbf{ADRS})$;

Начать Вычисление корня XMSS дерева
- **ADRS**.УстановитьТип(TREE);
- **ADRS**.УстановитьНомерКлюча(0);
- **ADRS**.УстановитьИндекс(idx);
- **цикл** $(k := 0; k < h'; k + 1)$ **выполнять**
  - **ADRS**.УстановитьВысоту(k+1);
  - **ADRS**.УстановитьИндекс(ADRS.ВернутьИндекс() $\gg 1$);
  - **если** $idx \gg k == 0 \mod 2$ **тогда**
    - $node[1] :=$ **H**(**PK**.seed, **ADRS**, ($node[0]$ || AUTH[k]));
  - **иначе**
    - $node[1] :=$ **H**(**PK**.seed, **ADRS**, (AUTH[k] || $node[0]$));
  - $node[0] := node[1]$;
- Вернуть($node[0]$).

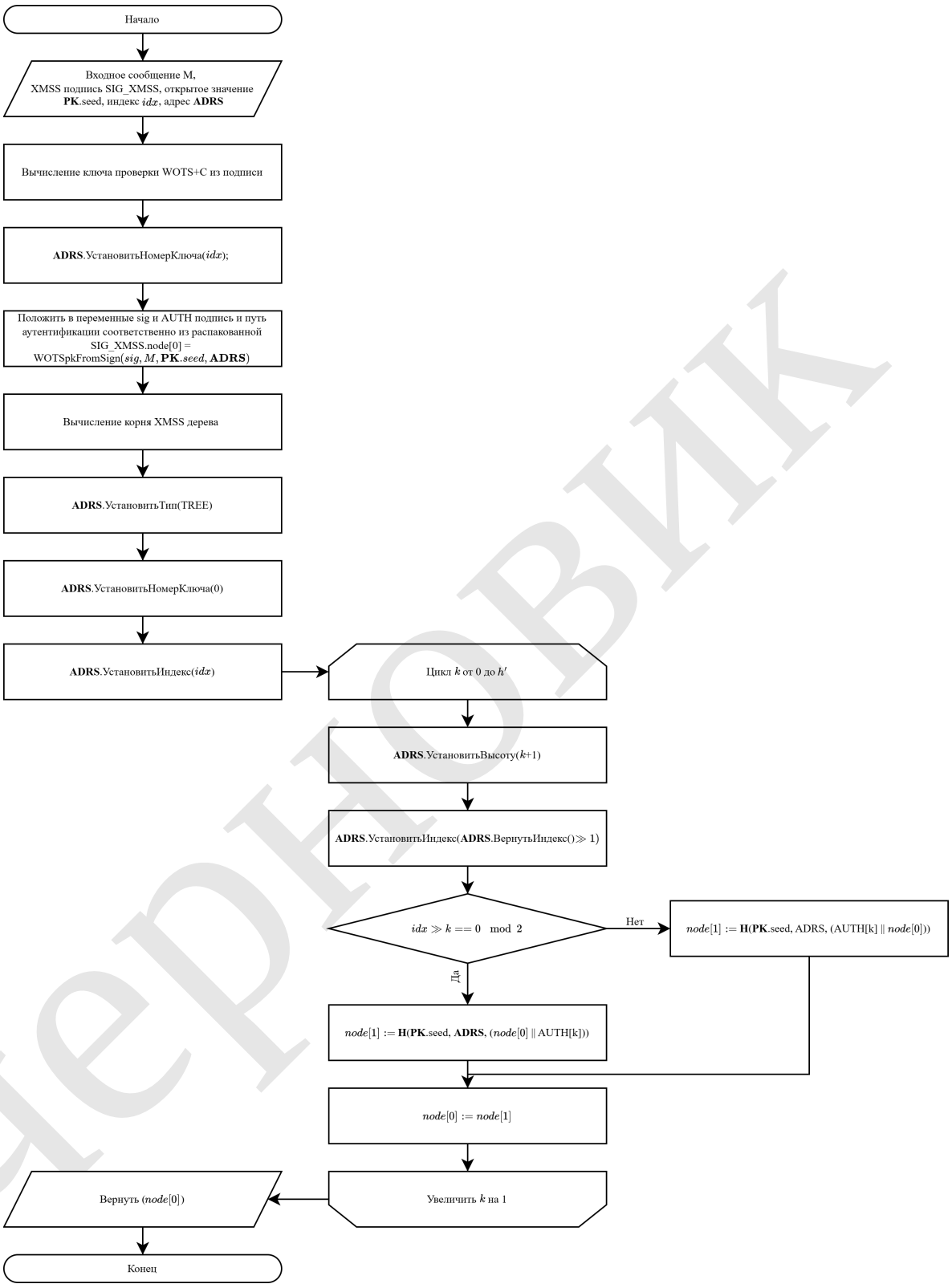

Рисунок 16 – *Алгоритм вычисления корня дерева XMSS*

5.7 Гипердерево

Схема «Гиперикум» использует концепцию гипердерева (HyperTree (HT)). Гипердерево представляет собой дерево из нескольких слоев деревьев XMSS. Деревья на верхнем и промежуточном уровнях используются для подписи от-

крытых ключей XMSS (корней деревьев XMSS на следующем ниже уровне). Деревья на нижнем уровне используются для подписи открытых ключей FORS. Все деревья XMSS в гипердереве имеют одинаковую высоту. Общая высота гипердерева равна h, а количество слоев равно d. XMSS-дерево имеет высоту $h' = h/d$. Уровень $d-1$ содержит одно дерево XMSS, уровень $d-2$ содержит $2^{h'}$ деревьев XMSS и так далее. Нижний слой на уровне 0 содержит $2^{h-h'}$ XMSS-деревьев.

5.7.1 Параметры гипердерева

В схеме гипердерева используется следующий набор параметров:

- h - общая высота гипердерева;
- d - число уровней в гипердереве;
- $h' = h/d$ - высота одного XMSS дерева;
- n - размер подписываемого сообщения, равный 256 бит.

5.7.2 Выработка ключей гипердерева

Секретный ключ гипердерева — это секретное начальное число SK.seed, которое используется для формирования всех секретных ключей WOTS-TW. Открытый ключ гипердерева — это корневой узел XMSS дерева на верхнем уровне. Описание генерации ключей гипердерева приведено в алгоритме 15 [1], блок-схема алгоритма представлена на рисунке 17.

Алгоритм 15: Алгоритм генерации ключей гипердерева;
HT.PKgen (**SK**.seed, **PK**.seed)

Входные значения : Секретное начальное значение **SK**.seed, открытое начальное значение **PK**.seed.

Выходные значения : Открытый ключ гипердерева **PK**.root.

1 **ADRS**.УстановитьУровень($d-1$);
2 **ADRS**.УстановитьДерево(0);
3 **PK**.root:= XMSS.PKgen (**SK**.seed, **PK**.seed, **ADRS**);
4 Вернуть(**PK**.root).

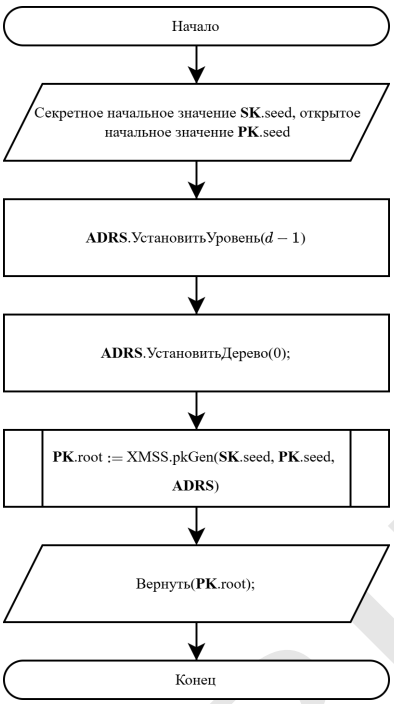

Рисунок 17 – *Алгоритм выработки ключей гипердерева*

5.7.3 Алгоритм формирования подписи гипердерева

Чтобы вычислить подпись гипердерева SIG_HT сообщения M, используется алгоритм HT.sign (алгоритм 16, рисунок 18). Алгоритм HT.sign принимает в качестве входных данных: сообщение M, секретное начальное значение **SK**.seed, открытое начальное значение **PK**. seed и индекс idx. Индекс определяет лист гипердерева, который будет использоваться для подписи сообщения. Затем подпись гипердерева формируется как набор подписей XMSS на пути от листа с индексом idx до верхнего дерева (общее число подписей XMSS равно d). Индекс передается как два отдельных аргумента, разделенных для обращения к конкретным дереву и листу в выбранном дереве.

Алгоритм 16: Алгоритм подписи гипердерева;
HT.sign (M, **SK**.seed, **PK**.seed, idx_tree, idx_leaf)

Входные значения : Сообщение M, секретное начальное значение **SK**.seed, открытое начальное значение **PK**.seed, индекс дерева idx_tree, индекс листа idx_leaf.

Выходные значения : Подпись SIG_HT.

ADRS.УстановитьУровень(0);
ADRS.УстановитьДерево(idx_tree);
SIG_HT := XMSS.Sign (M, **SK**.seed, **PK**.seed, idx_leaf, **ADRS**);
root := XMSS.pkFromSig (M, SIG_HT, **PK**.seed, idx_leaf, **ADRS**);
$h' := h/d$;
цикл $j := 1;\ j < d;\ j + 1$ **выполнять**
    idx_leaf := idx_tree mod $(1 \ll h')$;
    idx_tree := idx_tree $\gg h'$;
    ADRS.УстановитьУровень(j);
    ADRS.УстановитьДерево(idx_tree);
    SIG_tmp := XMSS.Sign (root, **SK**.seed, **PK**.seed, idx_leaf, **ADRS**);
    SIG_HT := SIG_HT || SIG_tmp;
    если $j < d - 1$ **тогда**
        root := XMSS.pkFromSig (root, SIG_tmp, **PK**.seed, idx_leaf, **ADRS**);
Вернуть(SIG_HT).

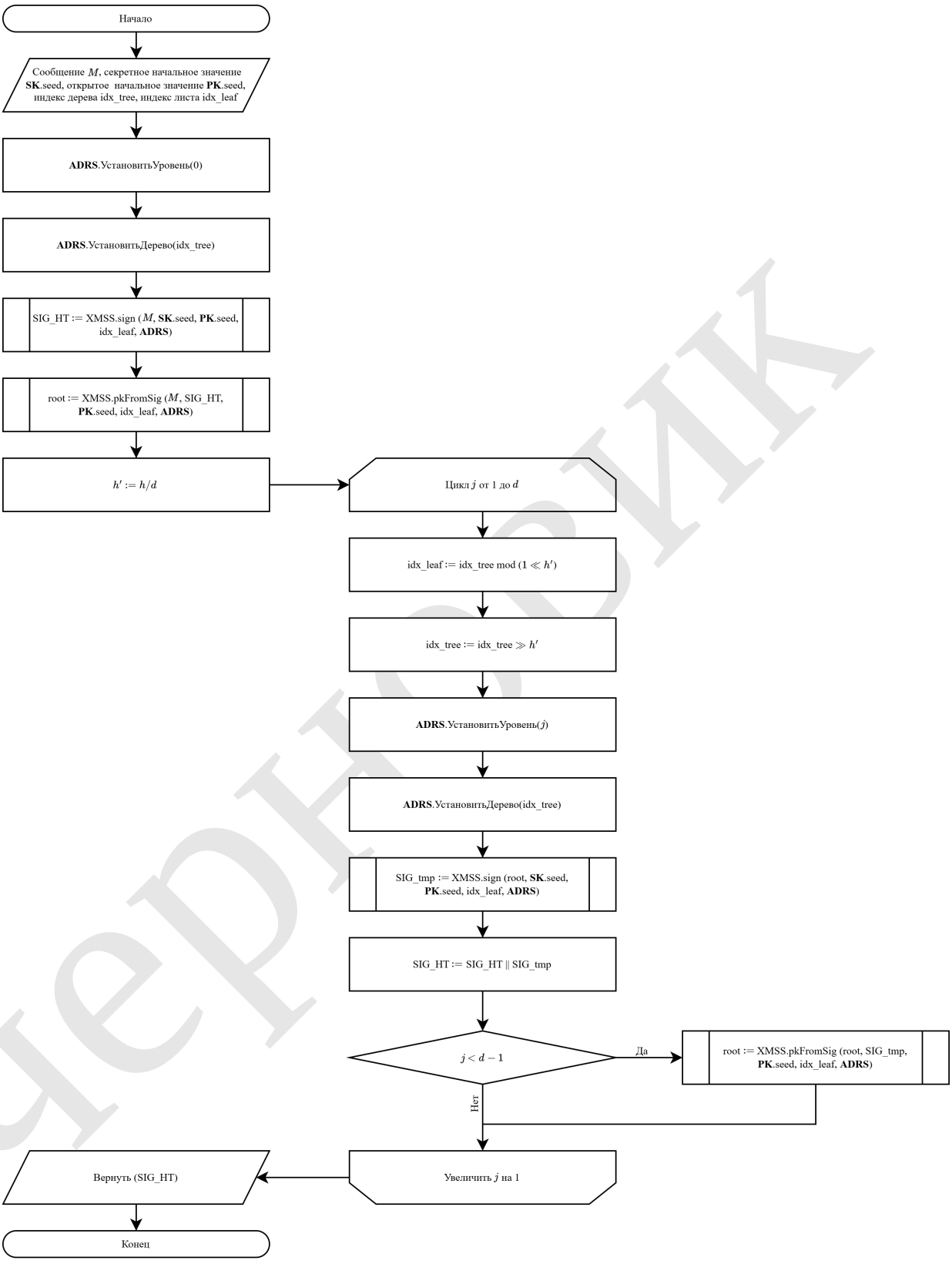

Рисунок 18 – *Алгоритм подписи гипердерева*

5.7.4 Алгоритм проверки подписи гипердерева

Проверка подписи гипердерева состоит из d вызовов XMSS.pkFromSig и сравнения с ключом проверки. Для проверки подписи гипердерева требуется сообщение M, подпись SIG_HT, открытое начальное значение **PK**.seed, индекс дерева idx_tree, индекс листа idx_leaf и ключ проверки PK_HT. Описание проверки подписи гипердерева приведено в алгоритме 17, блок-схема алгоритма представлена на рисунке 19.

Алгоритм 17: Алгоритм проверки подписи гипердерева;
HT.verify (M, SIG_HT, **PK**.seed, idx_tree, idx_leaf, **PK**.root)

Входные значения : Сообщение M, подпись SIG_HT, открытое начальное значение **PK**.seed, индекс дерева idx_tree, индекс листа idx_leaf, ключ проверки гипердерева **PK**.root.

Выходные значения : Результат проверки подписи.

```
1  Положить первый элемент подписи XMSS в SIG_tmp из SIG_HT;
2  ADRS.УстановитьУровень(0);
3  ADRS.УстановитьДерево(idx_tree);
4  node := XMSS.pkFromSig (M, SIG_tmp, PK.seed, idx_leaf, ADRS);
5  h' := h/d;
6  цикл j := 1; j < d; j + 1 выполнять
7      idx_leaf := idx_tree  mod (1 ≪ h');
8      idx_tree := idx_tree ≫ h';
9      Положить j-ый элемент подписи XMSS в SIG_tmp из SIG_HT;
10     ADRS.УстановитьУровень(j);
11     ADRS.УстановитьДерево(idx_tree);
12     node := XMSS.pkFromSig (node, SIG_tmp, PK.seed, idx_leaf, ADRS);
13 если  node == PK.root тогда
14     Вернуть(Проверка успешна);
15 иначе
16     Вернуть(Проверка не успешна).
```

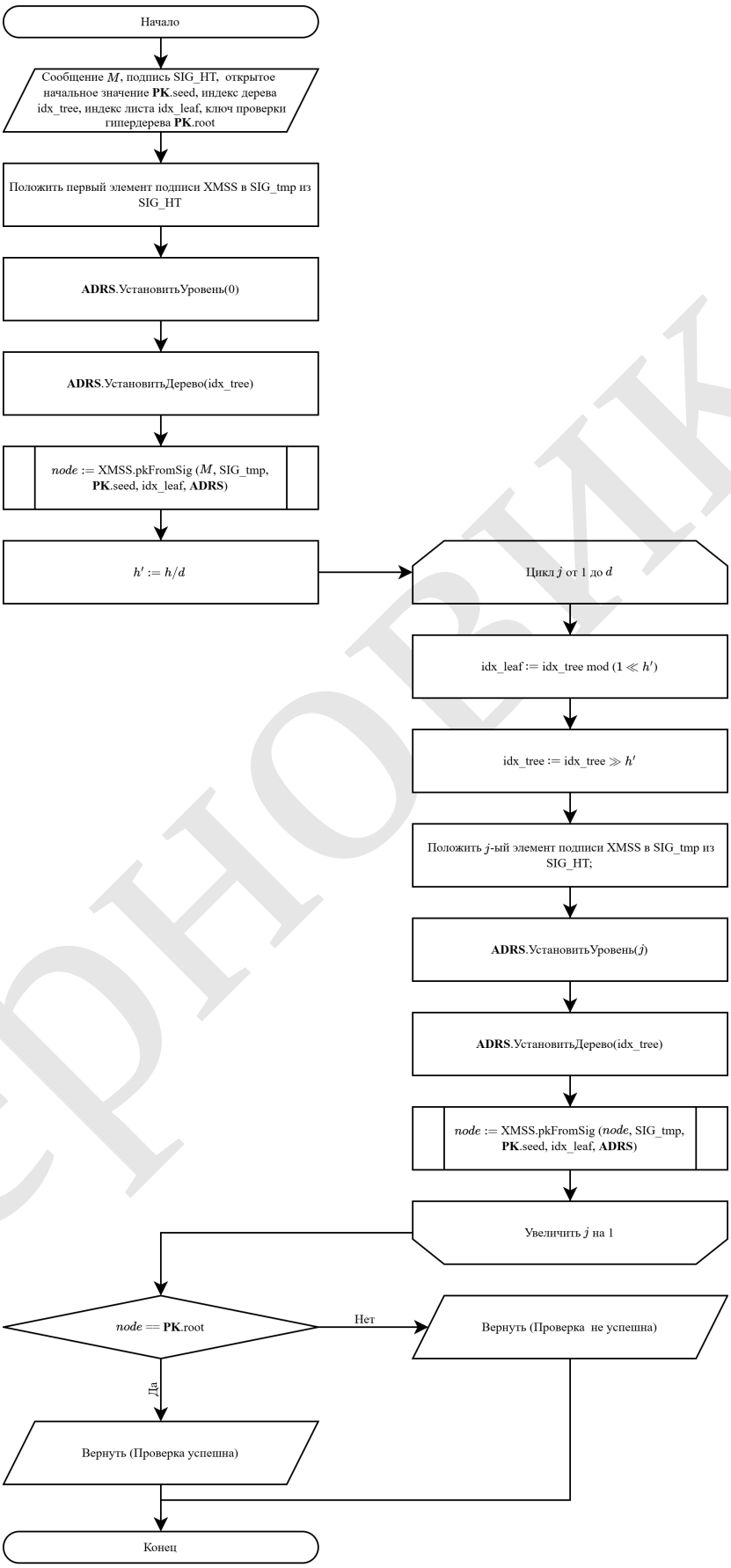

Рисунок 19 – *Алгоритм проверки подписи гипердерева*

5.8 Лес случайных подмножеств (FORS+C)

Гипердерево HT используется в «Гиперикум» для подписи открытых ключей экземпляров FORS [1], которые, в свою очередь, используются для подписи хэшей сообщений. FORS, сокращение от «лес случайных подмножеств» (Forest of random subsets), представляет собой схему подписи нескольких раз (Few time signature scheme (FTS)).

5.8.1 Параметры FORS+C

В схеме FORS+C используется следующий набор параметров:

- $k' = k - 1$ — количество наборов секретных ключей, деревьев и число блоков сообщения;
- b — высота дерева FORS+C и длина одного блока сообщения в битах;
- $t = 2^b$ — количество элементов в наборе секретных ключей и количество листьев в дереве FORS+C.

5.8.2 Секретный ключ FORS+C

В контексте «Гиперикум» секретный ключ FORS+C — это секретное начальное значение **SK**.seed, содержащееся в закрытом ключе «Гиперикум». Он используется для выработки $k \cdot t$ 256-битных значений секретного ключа с использованием псевдослучайной функции. Описание выработки одного секретного значения приведено в алгоритме 18, блок-схема алгоритма представлена на рисунке 20.

Алгоритм 18: Алгоритм выработки одного секретного значения FORS+C;
FORS.SKgen (**SK**.seed, **PK**.seed, **ADRS**, idx)

Входные значения : Секретное начальное значение **SK**.seed, открытое начальное значение **PK**.seed, адрес **ADRS**, индекс idx.

Выходные значения : Секретное значение sk.

1 **ADRS**.УстановитьТип(KEYGEN_FORS);
2 **ADRS**.УстановитьВысоту(0);
3 **ADRS**.УстановитьИндекс(idx);
4 sk := **PRF** (**SK**.seed, **PK**.seed, ADRS);
5 Вернуть(*sk*)

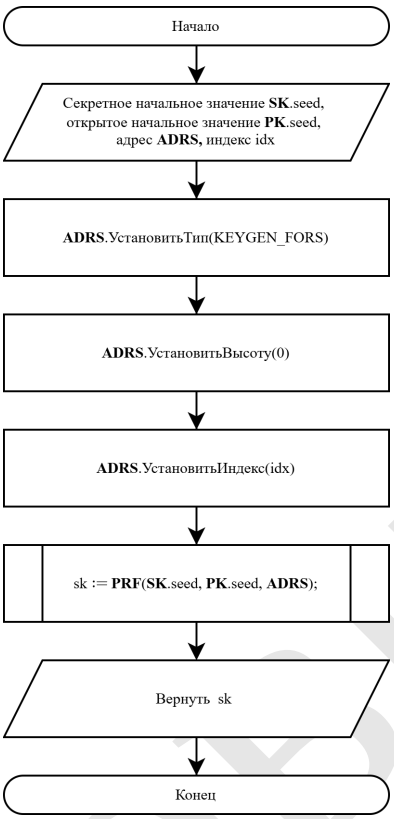

Рисунок 20 – *Алгоритм выработки одного секретного значения FORS+C*

5.8.3 Хэширование деревьев FORS+C

Для вычисления n-битовых узлов в хэш-деревьях FORS+C используется алгоритм FORS.treehash. Данный алгоритм аналогичен XMSS.treehash (Алгоритм 11). Различия заключаются в том, как вычисляются листовые узлы и как обрабатываются адреса. Описание хэширования деревьев FORS приведено в алгоритме 19, блок-схема алгоритма представлена на рисунке 21.

Алгоритм 19: Алгоритм хэширования деревьев FORS;
FORS.treehash (**SK**.seed, **PK**.seed, s, z, **ADRS**)

Входные значения : Секретное начальное значение **SK**.seed, открытое начальное значение **PK**.seed, начальный индекс s, высота целевого узла z, адрес **ADRS**.

Выходные значения : Целевой узел.

```
1  если s ≠ 0 mod 2^z тогда
2      Вернуть(Ошибка)
3  цикл i := 0; i < 2^z; i + 1 выполнять
4      sk := FORS.SKgen(SK.seed, PK.seed, ADRS, s + i);
5      ADRS.УстановитьТип(FORS_TREE);
6      ADRS.УстановитьВысоту(0);
7      ADRS.УстановитьИндекс(s + i);
8      node := F(PK.seed, ADRS, sk);
9      ADRS.УстановитьВысоту(1);
10     Пока Stack.ВернутьВысотуУзла() == ADRS.ВернутьВысоту() выполнять:
11         ADRS.УстановитьИндекс(ADRS.ВернутьИндекс() ≫ 1);
12         node := H (PK.seed, ADRS, (Stack.ИзъятьУзел() || node));
13         ADRS.УстановитьВысоту(ADRS.ВернутьВысоту() + 1);
14     Stack.ДобавитьПару(node, ADRS.ВернутьВысоту());
15 Вернуть(Stack.ИзъятьУзел()).
```

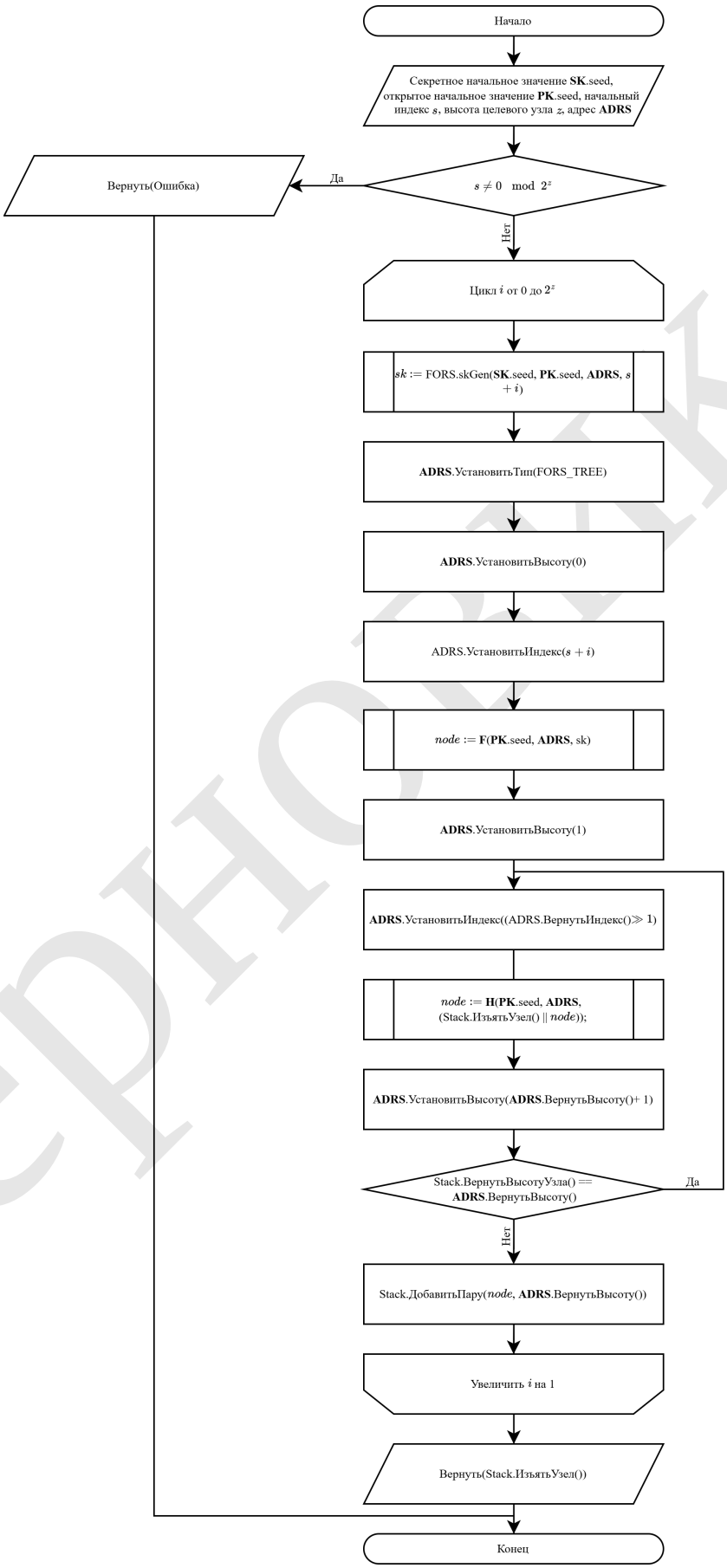

Рисунок 21 – *Алгоритм хэширования деревьев FORS*

5.8.4 Алгоритм формирования подписи FORS

Для формирования подписи алгоритм принимает следующие входные параметры: сообщение M, секретное начальное значение **SK**.seed, открытое начальное значение **PK**.seed, адрес **ADRS**. Полученная подпись FORS+C представляет собой массив длиной $k'(\log_2 t + 1)$ 256-битовых строк. Она содержит k' значений закрытого ключа по 256 бит и связанные с ними пути аутентификации - $\log_2 t$ значений по 256 бит каждый. Описание процедуры формирования подписи FORS приведено в алгоритме 20, блок-схема алгоритма представлена на рисунке 22.

Алгоритм 20: Алгоритм формирования подписи FORS;
FORS.sign (M, **SK**.seed, **PK**.seed, **ADRS**)

Входные значения : Сообщение M, секретное начальное значение **SK**.seed, открытое начальное значение **PK**.seed, адрес **ADRS**.

Выходные значения : Подпись FORS.

1 **цикл** $i := 0; i < k'; i + 1$ **выполнять**
2 idx := $indexes(M, b, i)$;
3 SIG_FORS := SIG_FORS || FORS.SKgen (**SK**.seed, **PK**.seed, **ADRS**, $i \cdot t$ + idx);
4 **Начать** Вычисление путей аутентификации
5 **цикл** $j := 0; j < b; j + 1$ **выполнять**
6 $s := (idx \gg j) \oplus 1$;
7 AUTH[j] := FORS.treehash (**SK**.seed, **PK**.seed, $i \cdot t + s \cdot 2^j$, j, **ADRS**);
8 SIG_FORS := SIG_FORS || AUTH;
9 Вернуть(SIG_FORS).

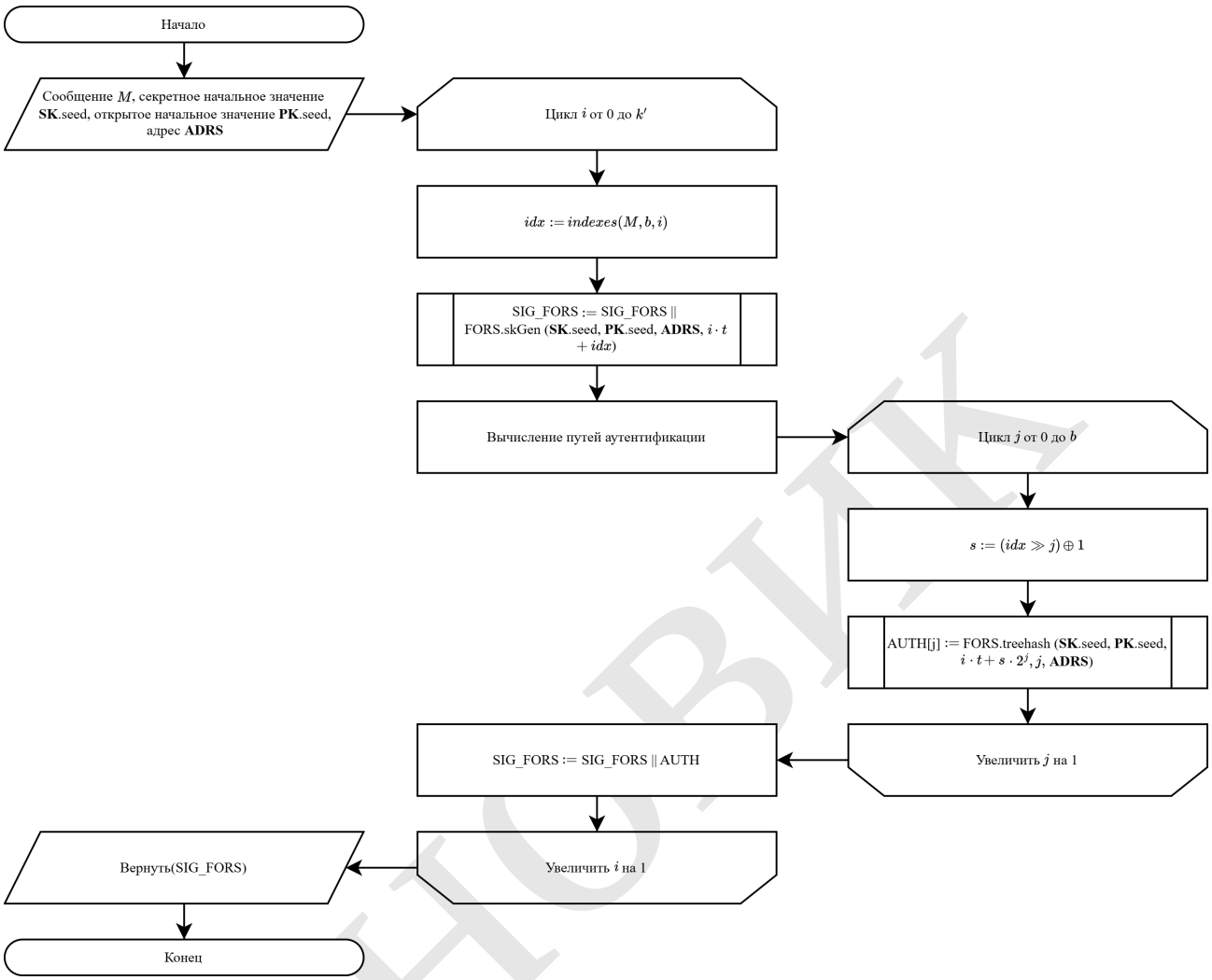

Рисунок 22 – *Алгоритм формирования подписи FORS*

5.8.5 Алгоритм вычисления ключа проверки FORS+C из подписи

«Гиперикум» использует неявную проверку подписей FORS. Подпись FORS+C используется для формирования ключа проверки FORS. Алгоритм вычисления ключа проверки FORS+C из подписи принимает на вход сообщение M, подпись SIG_FORS, открытое начальное значение **PK**.seed, адрес **ADRS**. Полученный ключ представляет собой хэш-код от адреса, открытого параметра и корней дерева. Далее ключ проверки используется в последующих вычислениях. Описание вычисления ключа проверки FORS+C из подписи приведено в алгоритме 21, блок-схема алгоритма представлена на рисунках 23 и 24.

Алгоритм 21: Алгоритм вычисления ключа проверки FORS+C из подписи; FORS.pkFromSig (M, SIG_FORS, **PK**.seed, **ADRS**)

Входные значения : Сообщение M, подпись SIG_FORS, открытое начальное значение **PK**.seed, адрес **ADRS**.

Выходные значения : Ключ проверки FORS.

Начать Вычисление корней деревьев
  ADRS.УстановитьТип(FORS_TREE);
  цикл $i := 0; i < k'; i + 1$ **выполнять**
    idx := $indexes(M, b, i)$;
    Положить i-ый секретный ключ в sk из SIG_FORS;
    ADRS.УстановитьВысоту(0);
    ADRS.УстановитьИндекс($i \cdot t + idx$);
    $node[0]$:= **F** (**PK**.seed, **ADRS**, sk);
    Положить i-ый путь аутентификации в $auth$ из SIG_FORS;
    цикл $j := 0; j < b; j + 1$ **выполнять**
      ADRS.УстановитьВысоту($j + 1$);
      ADRS.УстановитьИндекс(**ADRS**.ВернутьИндекс() $\gg 1$);
      если $idx \gg j == 0 \mod 2$ **тогда**
        $node[1]$:= **H** (**PK**.seed, **ADRS**, ($node[0]||auth[j]$));
      иначе
        $node[1]$:= **H** (**PK**.seed, **ADRS**, ($auth[j]||node[0]$));
      $node[0] := node[1]$;
    $roots[i] := node[0]$;
Начать Вычисление ключа проверки
  ADRS.УстановитьТип(FORS_ROOTS);
  ADRS.УстановитьСуффикс(0);
  pk := $\mathbf{Th}_{k'}$(**PK**.seed, **ADRS**, roots);
  Вернуть(*pk*).

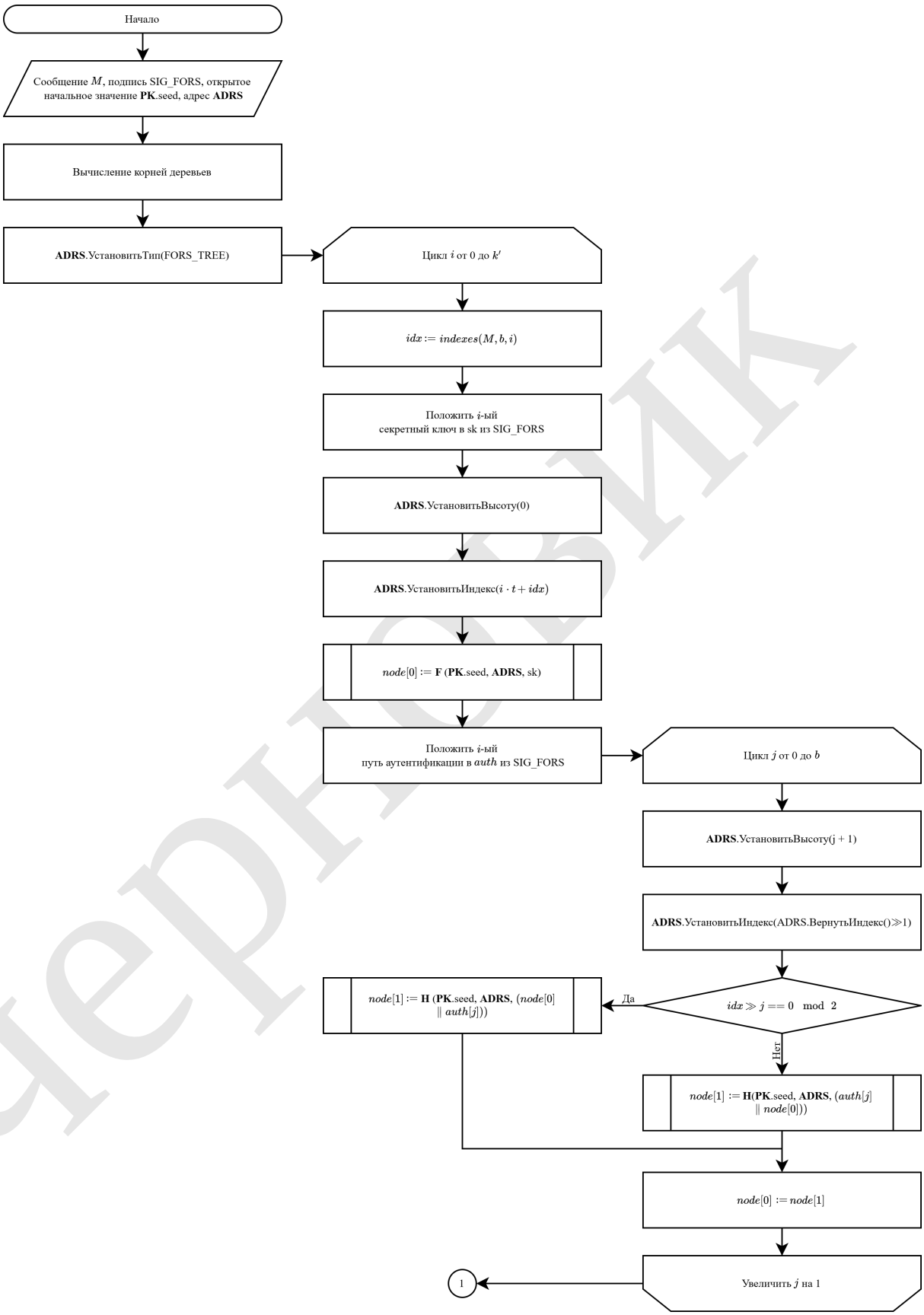

Рисунок 23 – *Алгоритм вычисления ключа проверки FORS+C из подписи часть 1*

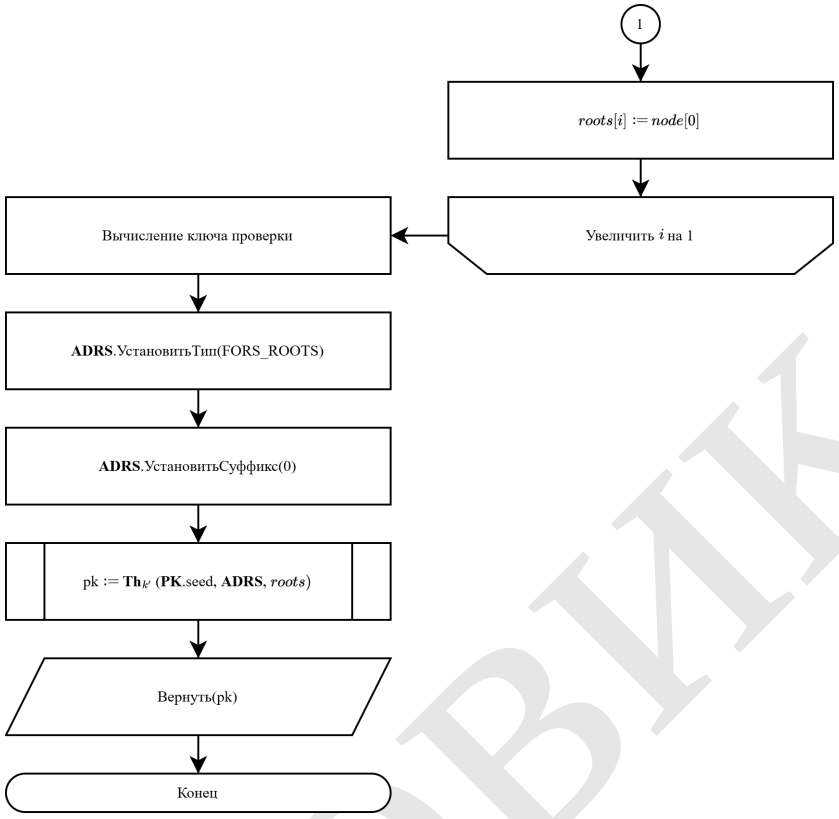

Рисунок 24 – *Алгоритм вычисления ключа проверки FORS+C из подписи часть 2*

5.9 Реализация функций на основе хэш-функции ГОСТ Р 34.11-2012

Для реализации функций используется хэш-функция согласно ГОСТ Р 34.11-2012 Хэш-функция с длиной выхода 256 бит обозначается как **Str256**. Хэш-функция с длиной выхода 512 бит обозначается как **Str512**.

$$\mathbf{PRF_{msg}}(\mathbf{SK}.\text{prf}, \mathbf{PK}.\text{seed}, opt, M) = \mathbf{Str256}(M||opt||\mathbf{PK}.\text{seed}||0^{256}||\mathbf{SK}.\text{prf}),$$

$$\mathbf{PRF}(\mathbf{SK}.\text{seed}, \mathbf{PK}.\text{seed}, \mathbf{ADRS}) = \mathbf{Str256}(\mathbf{ADRS}||\mathbf{PK}.\text{seed}||0^{256}||\mathbf{SK}.\text{seed}),$$

$$\mathbf{H_{msg}}(R, \mathbf{PK}.\text{seed}, \mathbf{PK}.\text{root}, s||M) = \mathbf{Str512}(M||s||\mathbf{PK}.\text{root}||\mathbf{PK}.\text{seed}||0^{256}||R),$$

$$\mathbf{F}(\mathbf{PK}.\text{seed}, \mathbf{ADRS}, M1) = \mathbf{Str256}(M1||\mathbf{ADRS}||0^{256}||\mathbf{PK}.\text{seed}),$$

$$\mathbf{H}(\mathbf{PK}.\text{seed}, \mathbf{ADRS}, M1||M2) = \mathbf{Str256}(M2||M1||\mathbf{ADRS}||0^{256}||\mathbf{PK}.\text{seed}),$$

$$\mathbf{H_s}(\mathbf{PK}.\text{seed}, \mathbf{ADRS}, s||M) = \mathbf{Str256}(M||s||\mathbf{ADRS}||0^{256}||\mathbf{PK}.\text{seed}),$$

$$\mathbf{Th}_i(\mathbf{PK}.\text{seed}, \mathbf{ADRS}, M) = \mathbf{Str256}(M||\mathbf{ADRS}||0^{256}||\mathbf{PK}.\text{seed}).$$

6 «Гиперикум»

Алгоритм «Гиперикум» представляет собой комбинацию всех схем, описанных ранее.

6.1 Параметры «Гиперикум»

«Гиперикум» использует следующий набор параметров:

- h — высота гипердерева;
- d — число уровней в гипердереве;
- $k' = k - 1$ — число деревьев в схеме FORS+C;
- b — высота одного дерева FORS+C;
- $hash\_size$ — размер выходного значения используемой хэш-функции;
- m — длина подписи сообщения в байтах:

$$m = 36 + k'(b+1) \cdot hash\_size + d \cdot (hash\_size \cdot 64 + 4 + h/d \cdot hash\_size)$$

6.2 Наборы параметров «Гиперикум»

В таблице 1 приведены рекомендованные наборы параметров. В первом столбце указывается тип набора. Следующие четыре столбца описывают параметры схемы «Гиперикум». В столбце «Подпись» указан размер подписи в байтах. Столбцы «**Th** для под.» и «**Th** для пров.» обозначают приблизительное число вызовов функции **Th** для формирования и проверки подписи соответственно.

Таблица 1 – *Наборы параметров схемы «Гиперикум»*

Название	h	d	b	k	Подпись, Б	**Th** для под.	**Th** для пров.
Гиперикум_Б_256_64	66	22	9	38	59 132	220 309	11 663
Гиперикум_М_256_64	68	4	18	15	18 932	544 997 215	2 368
Гиперикум_Б_256_20	21	7	10	36	27 392	130 233	3 955
Гиперикум_М_256_20	26	2	18	15	13 484	24 395 435	1 302
Гиперикум_Б_128_20	20	5	9	18	16 376	99 864	2 733
Гиперикум_М_128_20	20	2	11	14	9 772	2 154 158	1 187

Названия наборов содержат три блока символов, характеризующих различные свойства подписи. Первый блок с буквами «Б» (быстрая подпись) и «М» (маленькая подпись) отражает свойства размера подписи и времени ее вычисления. Второй блок с числами «128» и «256» отвечает за оценку уровня стойкости. Третий блок с числами «20» и «64» указывает на максимально допустимое число подписей при использовании одного секретного ключа. В частности, наборы с значением «20» на конце допускают формирование не более чем 2^{20} подписей при использовании одного секретного ключа. Аналогично, наборы с значением «64» на конце допускают формирование не более чем 2^{64} подписей при использовании одного секретного ключа.

6.3 Алгоритм выработки ключей «Гиперикум»

Ключ подписи «Гиперикум» состоит из двух элементов. Первый элемент — 256-битовое секретное начальное значение **SK**.seed, которое используется для выработки всех элементов секретного ключа WOTS+C и FORS+C. Второй элемент — 256-битовый ключ **SK**.prf, который используется для детерминированной выработки значения рандомизации для вычисления хэш-кода сообщения.

Ключ проверки также содержит два элемента. Первый элемент — ключ проверки гипердерева, то есть корень дерева на верхнем уровне. Второй элемент — 256-битовое общедоступное начальное значение **PK**.seed, которое формируется случайным образом. Ключ подписи содержит копию ключа проверки. Описание выработки ключей «Гиперикум» приведено в алгоритме 22, блок-схема алгоритма представлена на рисунке 25.

Алгоритм 22: Алгоритм выработки ключей «Гиперикум»;
Hypericum.PKgen ( )

Входные значения : Отсутствуют

Выходные значения : Открытый ключ PK и секретный ключ SK

1 **SK**.seed := sec_rand(n);
2 **SK**.prf := sec_rand(n);
3 **PK**.seed := sec_rand(n);
4 **PK**.root := HT.PKgen (**SK**.seed, **PK**.seed);
5 PK := (**PK**.seed, **PK**.root);
6 SK := (**SK**.seed, **SK**.prf, PK);
7 Вернуть(PK, SK).

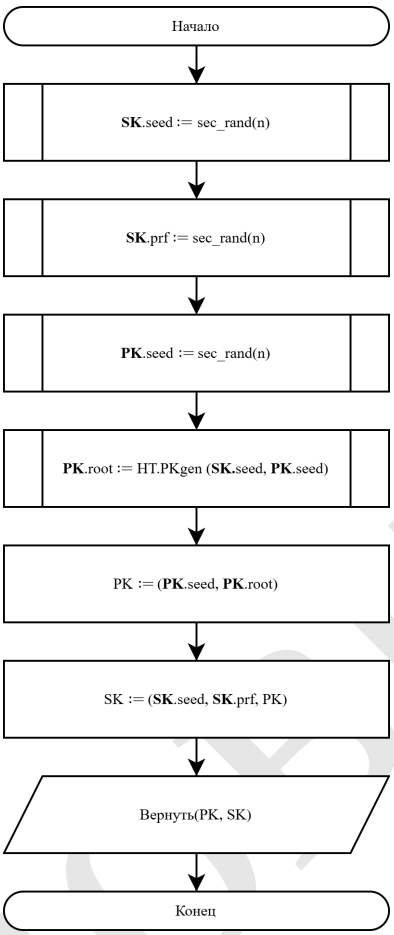

Рисунок 25 – *Алгоритм выработки ключей*

6.4 Алгоритм формирования подписи «Гиперикум»

Подпись «Гиперикум» представляет собой комбинацию подписей предыдущих схем. Формирование подписи можно разделить на пять этапов.

На первом этапе вычисляется случайное зерно R от **SK**.prf, **PK**.seed, переменной opt и сообщения M. В зависимости от режима работы выбирается значение opt. Первый режим работы является режимом по умолчанию, в котором переменная opt является фиксированным шестнадцатеричным числом и имеет следующее значение:

opt_const =
5475726368656e6b6f7c4b5544494e4f567c477265626e65767c514150500d0a

Во втором режиме — режиме со включенной случайностью, переменная opt формируется случайно.

На втором этапе происходит вычисление индексации, определяющее набор индексов для формирования подписей FORS+C и гипердерева, а также реализующее модификацию FORS+C.

На третьем этапе вычисляется подпись FORS+C.

На четвертом этапе выполняется вычисление ключа проверки из подписи FORS+C.

На пятом этапе выполняется вычисление подписи гипердерева от ключа проверки подписи FORS+C.

Подпись «Гиперикум» состоит из случайной строки R, счетчика s, подписи FORS+C и подписи гипердерева (см. рисунок 26, все размеры указаны в байтах). Описание формирования подписи «Гиперикум» приведено в алгоритме 23, блок-схема алгоритма представлена на рисунке 27.

32	4	$k'(\log_2 t + 1) \cdot 32$	$d \cdot (2052 + \frac{h}{d} \cdot 32)$
R	s	SIG_FORS	SIG_HT

Рисунок 26 – *Структура подписи алгоритма «Гиперикум» SIG*

Алгоритм 23: Алгоритм формирования подписи «Гиперикум»;
Hypericum.sign (M, SK)

Входные значения : Сообщение M, ключ подписи SK = (**SK**.seed, **SK**.prf, **PK**.seed, **PK**.root)

Выходные значения : Подпись SIG

Начать Формирование случайности
  opt := opt_const;
  если *случайность включена* **тогда**
    opt $\leftarrow_{\$} \{0,1\}^{256}$;
  R := **PRF**$_{\mathbf{msg}}$ (**SK**.prf, **PK**.seed, opt, M);
  SIG := R;
Начать Подготовка сообщения
  $s := 0$;
  digest := **H**$_{\mathbf{msg}}$ (R, **PK**.seed, PK.root, $s||M$);
  tmp_md := старшие $\lfloor (kb + 7)/8 \rfloor$ байт от digest;
  md := старшие kb бит от tmp_md;
  если *младшие b бит от md* $\neq 0$ **тогда**
    $s := s + 1$;
    Перейти на шаг 9.;
  tmp_idx_tree := следующие $\lfloor (h - h/d + 7)/8 \rfloor$ байт от digest;
  tmp_idx_leaf := следующие $\lfloor (h/d + 7)/8 \rfloor$ байт от digest;
  idx_tree := старшие $h - h/d$ бит от tmp_idx_tree;
  idx_leaf := старшие h/d бит от tmp_idx_leaf;
  SIG := SIG || s;
Начать Вычисление подписи FORS
  ADRS.УстановитьУровень(0);
  ADRS.УстановитьДерево(idx_tree);
  ADRS.УстановитьТип(FORS_TREE);
  ADRS.УстановитьНомерКлюча(idx_leaf);
  SIG_FORS := FORS.sign (tmp_md, **SK**.seed, **PK**.seed, **ADRS**);
  SIG := SIG || SIG_FORS;
Начать Вычисление ключа проверки FORS+C
  PK_FORS := FORS.pkFromSig (tmp_md, SIG_FORS, **PK**.seed, **ADRS**);
Начать Вычисление подписи гипердерева
  SIG_HT := HT.sign (PK_FORS, **SK**.seed, **PK**.seed, idx_tree, idx_leaf);
  SIG := SIG || SIG_HT;
  Вернуть(SIG).

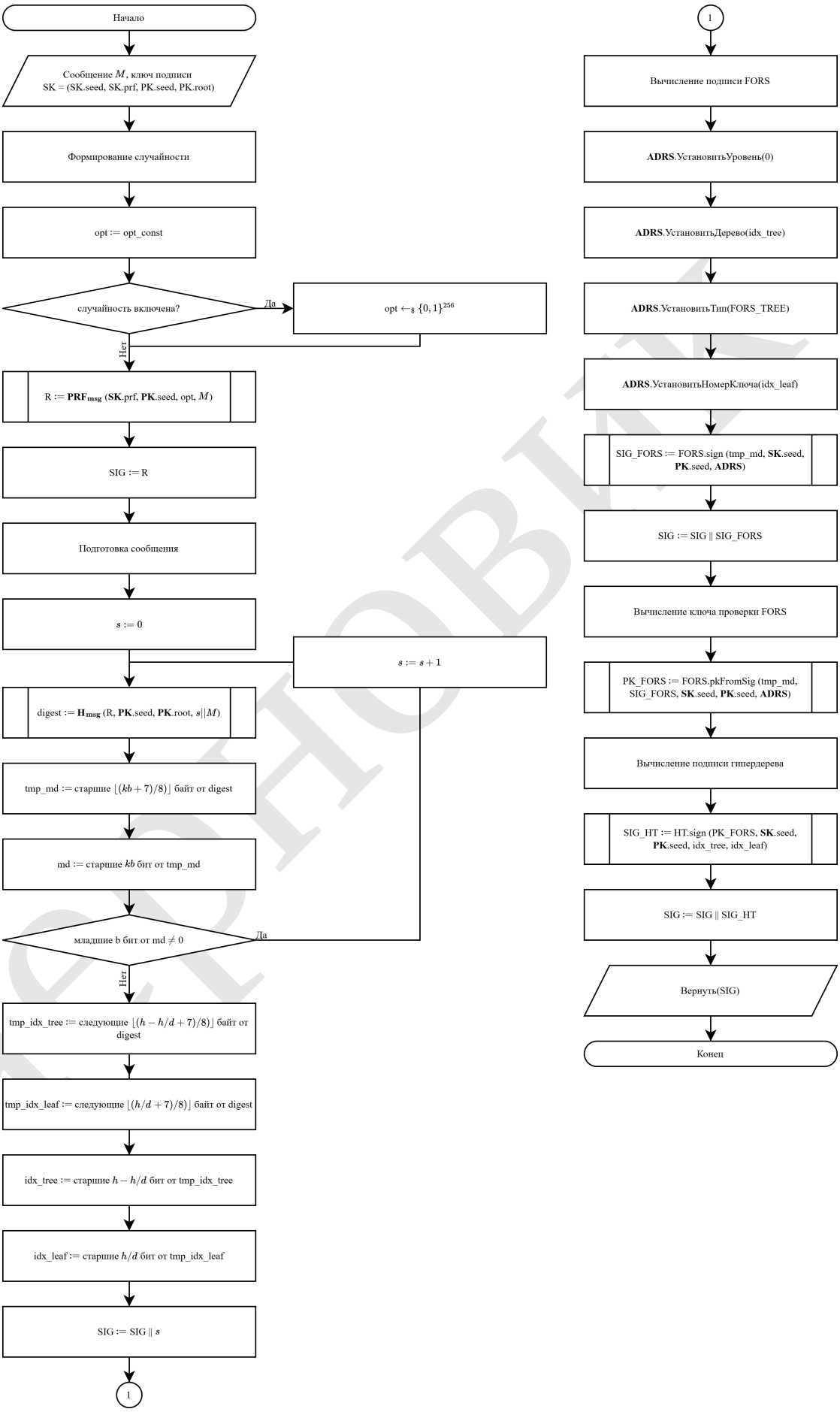

Рисунок 27 – *Алгоритм формирования подписи*

6.5 Алгоритм проверки подписи «Гиперикум»

Проверка подписи «Гиперикум» может быть реализована как повторное вычисление хэш-кода и индекса сообщения, вычисление ключа проверки FORS+C и проверка подписи гипердерева на этом открытом ключе. Описание проверки подписи «Гиперикум» приведено в алгоритме 24, блок-схема алгоритма представлена на рисунке28.

Алгоритм 24: Алгоритм проверки подписи «Гиперикум»;
Hypericum.Verify (M, SIG, PK)

Входные значения : Сообщение M, подпись SIG, ключ проверки PK.

Выходные значения : Результат проверки подписи.

1 **Начать** Инициализация
2 Положить в переменные R, s, SIG_FORS и SIG_HT соответствующие значения из подписи SIG;
3 **Начать** Вычисление хэш-кода сообщения
4 digest := $\mathbf{H_{msg}}$ (R, **PK**.seed, **PK**.root, $s||M$);
5 tmp_md := старшие $\lfloor (kb + 7)/8 \rfloor$ байт от digest;
6 md := старшие kb бит от tmp_md;
7 **если** *младшие b бит от md* $\neq 0$ **тогда**
8 Вернуть(Проверка не успешна)
9 tmp_idx_tree := следующие $\lfloor (h - h/d + 7)/8 \rfloor$ байт от digest;
10 tmp_idx_leaf := следующие $\lfloor (h/d + 7)/8 \rfloor$ байт от digest;
11 idx_tree := старшие $h - h/d$ бит от tmp_idx_tree;
12 idx_leaf := старшие h/d бит от tmp_idx_leaf;
13 **Начать** Вычисление ключа проверки FORS
14 **ADRS**.УстановитьУровень(0);
15 **ADRS**.УстановитьДерево(idx_tree);
16 **ADRS**.УстановитьТип(FORS_TREE);
17 **ADRS**.УстановитьНомерКлюча(idx_leaf);
18 PK_FORS := FORS.pkFromSig (tmp_md, SIG_FORS, **PK**.seed, **ADRS**);
19 **Начать** Проверка подписи гипердерева
20 Вернуть(HT.verify (PK_FORS, SIG_HT, **PK**.seed, idx_tree, idx_leaf, **PK**.root)).

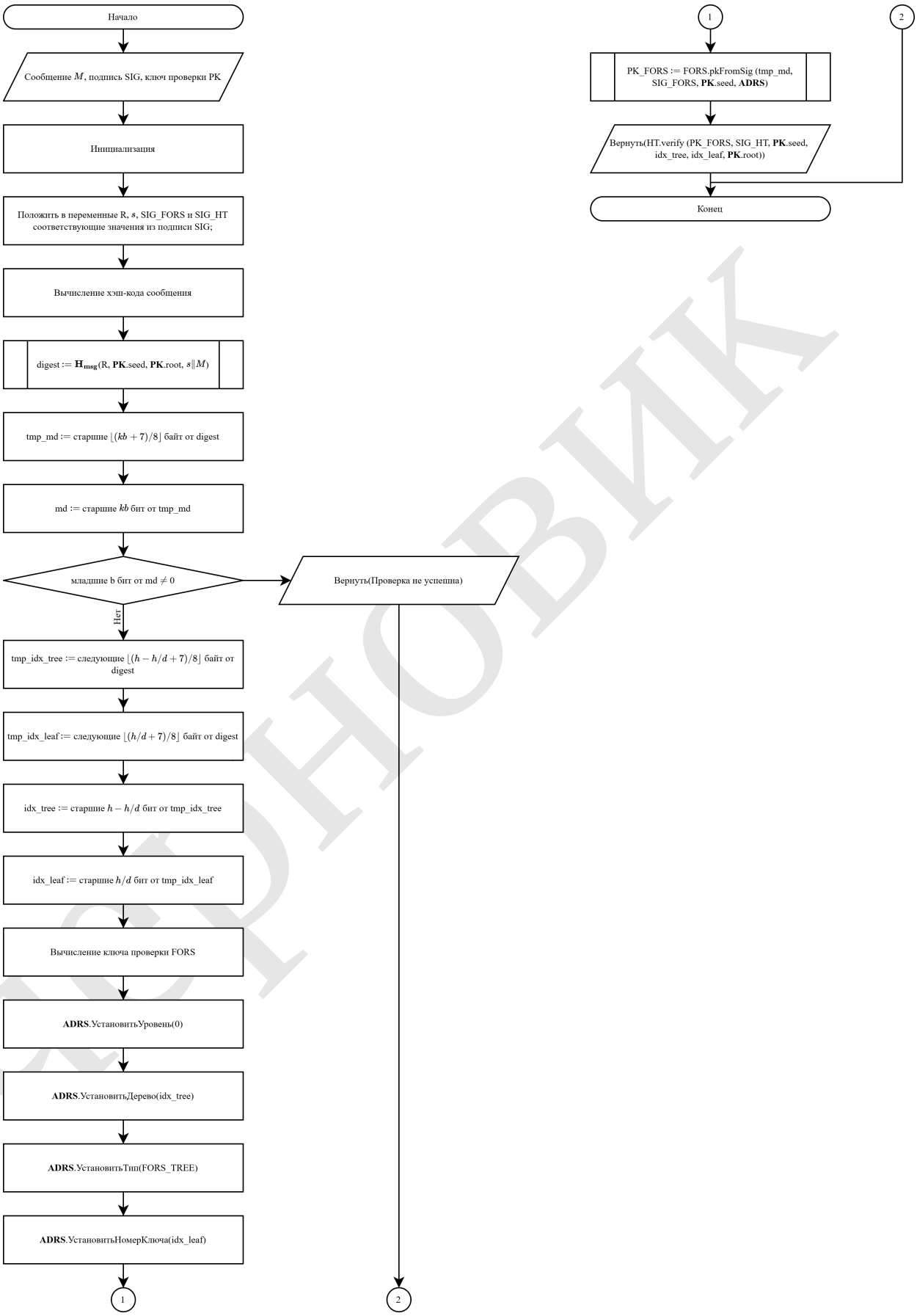

Рисунок 28 – *Алгоритм проверки подписи*

Список литературы

[1] Jean-Philippe Aumasson и др. *SPHINCS+ – Submission to the 3rd round of the NIST post-quantum project. v3.1*. 2022. URL: https://sphincs.org/data/sphincs+-r3.1-specification.pdf.

[2] A. Huelsing и др. *XMSS: eXtended Merkle Signature Scheme*. RFC 8391, DOI 10.17487/RFC8391. 2018. URL: https://www.rfc-editor.org/info/rfc8391.

Ключевые слова: обработка данных, передача данных, обмен информацией, сообщения, цифровые подписи, защита информации, формирование цифровой подписи, проверка цифровой подписи, постквантовая криптография.

Руководители рабочей группы ТК 26 по постквантовым криптографическим механизмам:

И.В. Чижов, АО «НПК «Криптонит», i.chizhov@kryptonite.ru

В.А. Шишкин, АО «НПК «Криптонит», v.shishkin@kryptonite.ru

Авторы документа:

О.Ю. Турченко, ООО «КуАпп», oturchenko@qapp.tech

С.В. Гребнев, ООО «КуАпп», sg@qapp.tech

Ж.-М. Дакуо, ООО «КуАпп», djeanmichelle@qapp.tech