МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Кафедра математического моделирования и анализа данных

Курсовой проект

Криптография на основе функций хэширования: подписи без состояния

Болтач Антон Юрьевич
Студент 4 курса 9 группы
Научный руководитель
С. В. Агиевич

Минск, 2019 г.

Содержание

Введение

Цифровые подписи широко используются в Интернете, в частности, для аутентификации, проверки целостности и отказа от авторства. Алгоритмы цифровой подписи, наиболее часто используемые на практике - RSA, DSA и ECDSA, - основаны на допущениях твердости о задачах теории чисел, а именно факторизации составного целого числа и вычислении дискретных логарифмов. В 1994 году Питер Шор показал, что эти теоретические проблемы с числами могут стать решаемыми при наличии квантовых вычислений. Квантовые компьютеры могут решить их за полиномиальное время, ставя под угрозу безопасность схем цифровой подписи, используемых сегодня. Хотя квантовые компьютеры еще не доступны, их развитие происходит быстрыми темпами и поэтому представляет собой реальную угрозу в течение следующих десятилетий. К счастью, постквантовая криптография предоставляет множество квантовостойких альтернатив классическим схемам цифровой подписи. Подписи на основе хэша или подписи Меркля, как они также известны, являются одной из наиболее многообещающих из этих альтернатив.

1.1 Почему Hash-Based Signatures?

Есть много причин использовать схемы подписи на основе хэша и предпочитать их другим альтернативам. Хотя в самой ранней схеме подписи отсутствуют практические требования к производительности и пространству, современные схемы на основе хэшей, такие как XMSS, достаточно быстры, при небольшом размере. Также требования безопасности являются убедительными. Использование такой схемы подписи всегда требует хэш-функции. В то время как другие схемы подписи полагаются на дополнительные предположения о неразрешимости для генерации подписи, для решения на основе хэша требуется только безопасная хэш-функция. Некоторые схемы, основанные на хэше, даже уменьшают потребность в хэш-функции, устойчивой к столкновениям, до той, которая должна выдерживать атаки только на второе изображение. В качестве примера известны практические атаки средствами защиты от столкновений функции MD5, но мы до сих пор не знаем о виртуальных атаках на второе изображение.

Одноразовые подписи(OTS)

Одноразовые подписи (OTS) называются одноразовыми, поскольку сопутствующие сокращения безопасности гарантируют безопасность только при атаках с одним сообщением. Однако это не означает, что эффективные атаки возможны при атаках с двумя сообщениями. Особенно в контексте основанных на хэшировании OTS (которые являются основными строительными блоками последних предложений по стандартизации) это приводит к вопросу о том, приводит ли случайное повторное использование одноразовой пары ключей к немедленной потере безопасности. Проанализируем безопасность наиболее известных хэш-основанных OTS: $WOTS$, $WOTS^+$ при различных видах атак с двумя сообщениями. Интересно, что оказывается, что схемы все еще безопасны при двух атаках сообщений, асимптотически.

2.1 Одноразовая подпись Винтерница($WOTS$)

WOTS использует функцию сохранения длины (криптографический хэш) $F : \{0,1\}^n \to \{0,1\}^n$. Она параметризуется длиной сообщения m и параметром $Winternitz$, $w \in N$, $w > 1$, который определяет компромисс между временем и памятью. Эти два параметра используются для вычисления

$$l_1 = \left\lceil \frac{m}{log(w)} \right\rceil, l_2 = \left\lfloor \frac{log(l_1(w-1))}{log(w)} \right\rfloor + 1, l = l_1 + l_2.$$

Схема использует $w-1$ итерации F на случайном входе. Мы определяем их как

$$F^a(x) = F(F^{a-1}(x))$$

и $F^0(x) = x$.

Теперь опишем три алгоритма схемы:

Алгоритм генерации ключей ($kg(1^n)$): На входе параметр безопасности 1^n алгоритм генерации ключей выбирает l n-бит строки равномерно, случайным образом. Секретный ключ $sk = (sk_1, ..., sk_l)$ состоит из этих l случайных битовых строк. Открытый ключ проверки pk вычисляется как

$$pk = (pk_1, ..., pk_l) = (F^{w-1}(sk_1), ..., F^{w-1}(sk_l))$$

Алгоритм подписи($sign(1^n, M^*, sk)$): На входе параметр безопасности 1^n, сообщение M^* длины m и секретного ключа подписи sk, алгоритм подписи сначала вычисляет базовое w представление M^* : $M^* = (M_1^*, ..., M_{l_1}^*), M_i^* \in \{0, ..., w-1\}$. Далее он вычисляет контрольную сумму

$$C = \sum_{i=1}^{l_1} (w - 1 - M_i^*)$$

и вычисляет его базовое w представление $C = (C_1, ..., C_{l_2})$. Длина базового w представления C не более l_2, так как $C \leq l_1(w-1)$. Мы задаем $B = (B_1, ..., B_l) = M^*||C$. Подпись вычисляется как

$$\sigma = (\sigma_1, ..., \sigma_l) = (F^{B_1}(sk_1), ..., F^{B_l}(sk_l))$$

.

Алгоритм проверки $(vf(1^n, M^*, \sigma, pk))$: На входе параметр безопасности 1^n, сообщение M^* длины m, подпись σ и открытый ключ проверки pk, алгоритм проверки сначала вычисляет B_i, $1 \leq i \leq l$, как описано выше. Затем он выполняет следующее сравнение:

$$pk = (pk_1, ..., pk_l) \stackrel{?}{=} (F^{w-1-B_1}(\sigma_1), ..., (F^{w-1-B_l}(\sigma_l))$$

Если сравнение выполняется, оно возвращает true и false в противном случае.

2.2 Дополненная подпись Винтерница($WOTS^+$)

Теперь опишем $WOTS^+$. Как и все варианты $W-OTS$, $W-OTS^+$ параметризуется параметром безопасности $n \in N$, длиной сообщения m и параметром $w \in N$, $w > 1$, который определяет компромисс между временем и памятью. Последние два параметра используются для вычисления

$$l_1 = \left\lceil \frac{m}{log(w)} \right\rceil, l_2 = \left\lfloor \frac{log(l_1(w-1))}{log(w)} \right\rfloor + 1, l = l_1 + l_2.$$

Кроме того, $W-OTS^+$ использует семейство функций $F_n : \{f_k : \{0,1\}^n \rightarrow \{0,1\}^n | k \in K_n\}$ с ключевым пространством K_n. Можно предположить как о криптографическом семействе хэш-функций, которое не сжимается. Используя F_n, мы определяем следующую цепную функцию.

$c_k^i(x, r)$: На входе значения $x \in \{0,1\}^n$, счетчика итераций $i \in N$, ключа $k \in K$ и элементы рандомизации $r = (r_1, ..., r_j) \in \{0,1\}^{nxj}$ при $j \geq i$, цепная функция работает следующим образом. В случае $i = 0$, c возвращает $x(c_k^0(x, r) = x)$. Для $i > 0$ мы определяем c рекурсивно как

$$c_k^i(x, r) = f_k(c_k^{i-1}(x, r) \oplus r_i),$$

То есть в каждом раунде функция сначала принимает побитовое xor промежуточного значения и битовую маску r, затем оценивает f_k на результат. Мы пишем $r_{a,b}$ для подмножества $r_a, ..., r_b$ как r. В случае $b < a$ мы определяем $r_{a,b}$ как пустую строку. Будем считать, что параметры m, w и семейство функций F_n общеизвестны. Теперь мы опишем три алгоритма $W-OTS^+$:

Алгоритм генерации ключа $(Kg(1^n))$: При вводе параметра безопасности n в унарном, алгоритм генерации ключа выбирает $l + w - 1$ n-бит строки равномерно случайным образом. Секретный ключ $sk = (sk_1, ..., sk_l)$ состоит из

первых l случайных битовых строк. Оставшиеся $w-1$ бит строки используются в качестве элементов рандомизации $r = (r_1, ..., r_{w-1})$ для c. Далее, Kg выбирает функцию ключа $k \xleftarrow{\$} K$ равномерно случайным образом. Открытый ключ проверки pk вычисляется как

$$pk = (pk_0, pk_1, ..., pk_l) = ((r, k), c_k^{w-1}(sk_1, r), ..., c_k^{w-1}(sk_l, r)).$$

Алгоритм подписи ($Sign(M, sk, r)$): На входе m битного сообщения M, секретного ключа подписи sk и элементов рандомизации r, алгоритм подписи сначала вычисляет базовое w представление $M : M = (M_1...M_{l_1})$, $M_i \in \{0, ..., w-1\}$. Поэтому M рассматривается как двоичное представление натурального числа x, а затем вычисляется w бинарное представление x. Далее вычисляем контрольную сумму

$$C = \sum_{i=1}^{l_1}(w - 1 - M_i)$$

и его базовое w представление $C = (C_1, ..., C_{l_2})$. Длина базового w представления C не более l_2, так как $C \leq l_1(w-1)$. Мы задаем $B = (b_1, ..., b_l) = M||C$, конкатенация базовых w представлений M и C. Подпись вычисляется как

$$\sigma = (\sigma_1, ..., \sigma_l) = (c_k^{b_1}(sk_1, r), ..., c_k^{b_l}(sk_l, r)).$$

Обратите внимание, что контрольная сумма гарантирует, что с учетом $b_i, 0 < i \leq l$, соответствующего одному сообщению, b_i^* соответствующий любому другому сообщению включает по крайней мере один $b_i^* < b_i$.

Алгоритм проверки ($Vf(1^n, M, \sigma, pk)$): На входе сообщение M двоичной длины m, подпись σ и открытый ключ pk. Алгоритм проверки сначала вычисляет $b_i, 1 \leq i \leq l$, как описано выше. Затем он выполняет следующее сравнение:

$$pk = (pk_0, pk_1, ..., pk_l) \stackrel{?}{=} ((r, k), c_k^{w-1-b_1}(\sigma_1, r_{b_1+1,w-1}, ..., c_k^{w-1-b_l}(\sigma_l, r_{b_l+1,w-1}))$$

Если сравнение выполняется, оно возвращает true и false в противном случае.

Время выполнения всех трех алгоритмов ограничено lw оценками f_k. Размер подписи и секретного ключа составляет $|\sigma| = |sk| = ln$ бит. Размер открытого ключа равен $(l+w-1)n+|k|$ бит, где $|k|$ обозначает количество бит, необходимых для представления любого элемента K.

2.2.1 Обоснование стойкости($WOTS^+$)

В этом разделе мы анализируем безопасность $WOTS^+$.

Определение(ϵ-доступность обнаружения подделки). ϵ-доступность обнаружения подделки(ϵ-FDA) для одноразового $WOTS^+$ S определяется следующим экспериментом.

Эксперимент $Exp_{S,n}^{FDA}(A)$

$(sk, pk) \leftarrow S.Kg(1^n)$

$(M^*, \sigma^*) \leftarrow A^{Sign(sk,\cdot)}$

Пусть (M, σ) будьте парой запрос-ответ $Sign(sk, \cdot)$.

Вернём 1, если $S.Sign(sk, M^*) \to \sigma^*, S.Vf(pk, \sigma^*, M^*) \to 1, M^* \neq M$.

Тогда схема $WOTS^*$ S имеет $\epsilon - FDA$, если нет противника A, который преуспевает с вероятностью $\geq \epsilon$.

Построим схему $(n, \delta, L, \nu) - W - OTS^+$ следующим образом.

Введем параметр $\nu \in \{1, 2, ...\}$ определение длины блоков, в которых сообщение разбивается во время алгоритма подписи, где мы предполагаем, что L кратно ν. Введем следующие вспомогательные значения:

$$w := 2^\nu, l_1 := \lceil L/\nu \rceil, l_2 := \lfloor log_2(l_1(w-1))/\nu \rfloor + 1, l := l_1 + l_2$$

.

Затем рассмотрим семейство односторонних функций:

$$f_r^{(i)} : \{0,1\}^{n+\delta(w-i)} \to \{0,1\}^{n+\delta(w-i-1)}$$

,

где $i \in 1, ..., w-1$ и параметр r принадлежит некоторой области D. Мы предполагаем, что $f_r^{(i)}$ удовлетворяет случайному предположению оракула для равномерно случайно выбранного r из D. Использование этого параметра может соответствовать XOR некоторого семейства хэш-функций со случайной битовой маской.

Затем мы вводим цепную функцию $F_r^{(i)}$, которую мы определяем рекурсивно следующим образом:

$$F_r^{(0)}(x) = x, F_r^{(i)}(x) = f_r^{(i)}(F_r^{(i-1)}(x)), i \in \{1, ..., w-1\}.$$

Алгоритмы схемы $(n, \delta, L, \nu) - W - OTS^+$ следующие:

Алгоритм генерации пары ключей $((sk, pk) \leftarrow (n, \delta, L, \nu) - WOTS^+.Kg)$. Сначала алгоритм генерирует секретный ключ в следующем виде:

$$sk := (r, sk_1, sk_2, ..., sk_l), sk_i \xleftarrow{\$} \{0,1\}^{n+\delta(w-1)}, r \xleftarrow{\$} D$$

(Смотрите Рис. 1). Затем открытый ключ, состоящий из рандомизирующего параметра r и результаты цепной функции, используемой для sk_i следующим образом:

$$pk := (r, pk_1, pk_2, ..., pk_l), pk_i := F_r^{w-1}(sk_i)$$

Алгоритм подписи $(\sigma \leftarrow (n, \delta, L, \nu) - WOTS^+.Sign(sk, M))$. Сначала алгоритм вычисляет базовое w представление M, разбивая его на ν-битные блоки $(M = (m_1, ..., m_{l_1})$, где $m_i \in \{0, ..., w-1\})$. Мы называем это частью сообщения. Затем алгоритм вычисляет контрольную сумму

$$C := \sum_{i=1}^{l_1} (w - 1 - m_i)$$

и его базовое w представление $C = (c_1, ..., c_{l_2})$. Мы называем это контрольной суммой. Определим расширенную строку $B = (b_1, ..., b_l) := M||C$ как конкатенация частей сообщения и контрольной суммы. Наконец, подпись генерируется следующим образом:

$$\sigma = (\sigma_1, ..., \sigma_l), \sigma_i := F_r^{(b_i)}(sk_i).$$

Алгоритм проверки $(\nu \leftarrow (n, \delta, L, \nu) - WOTS^+.Vf(pk, \sigma, M))$. Идея алгоритма состоит в том, чтобы восстановить открытый ключ из заданной подписи σ и затем проверить, совпадает ли он с исходным открытым ключом pk. Во-первых, алгоритм вычисляет базовую w строку $B = (B_1, ..., B_l)$ таким же образом, как и в алгоритме подписи (см. выше). Затем для каждой части подписи σ_i алгоритм вычисляет оставшуюся часть цепочки следующим образом:

$$pk_i^{check} := f_r^{(w-1)} \circ ... \circ f_r^{(b_i+1)}(\sigma_i),$$

где $\circ$ композиция функций. Если $pk_i^{check} = pk_i$ для всех $i \in \{1, ..., l\}$, затем алгоритм выводит $\nu := 1$, иначе $\nu := 0$.

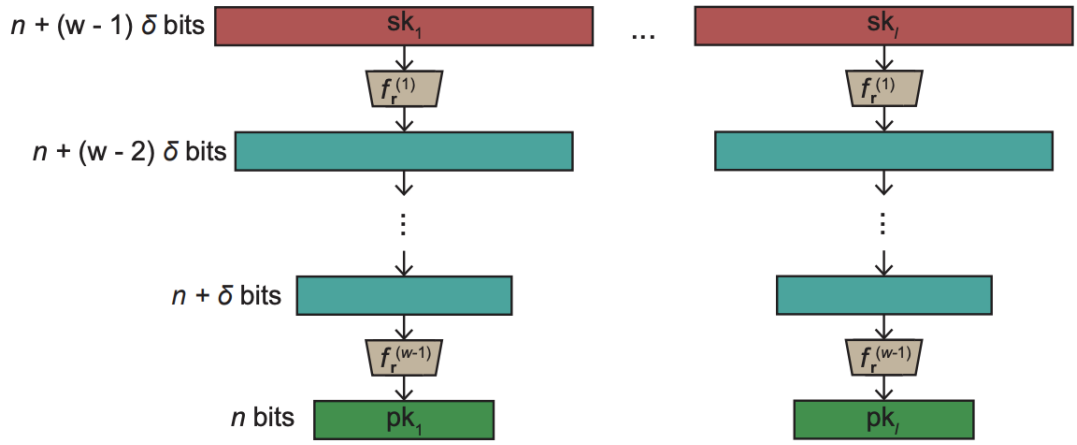


Рис. 1: Основной принцип построения открытого ключа в схеме $(n, \delta, L, \nu) - WOTS^+$

Основной результат по свойству FDA схемы $(n, \delta, L, \nu) - W - OTS^+$ можно сформулировать следующим образом:

Теорема $(n, \delta, L, \nu) - W - OTS^+$ схема имеет свойство $\epsilon - FDA$ с $\epsilon < 5.22 \times 2^{-\delta}$.

Доказательство. Рассмотрим сценарий успешного CMA на схеме $(n, \delta, L, \nu) - W - OTS^+$, в которой противник сначала заставляет законного пользователя с открытым ключом $pk = (r, pk_1, ..., pk_l)$, чтобы предоставить ему подпись $\sigma = (\sigma_1, ..., \sigma_l)$ для некоторого сообщения M, а затем генерировать действительную подпись $\sigma^* = (\sigma_1^*, ..., \sigma_l^*)$ для какого-то сообщения $M^* \neq M$. Пусть $(m_1, ..., m_l)$ и $(m_1^*, ..., m_l^*)$ будет w-образных представлений M и M^* соответственно. Рассмотрим расширенные w-базовые строки $B = (b_1^0, ..., b_l^0)$ и $B^* = (b_1^*, ..., b_l^*)$ генерируется путем добавления частей контрольной суммы.

Легко видеть, что для любого отличного M и M^* существует по крайней мере одна позиция $j \in \{1, ..., l\}$ такой, что $b_j^* < b_j$. Действительно, даже если для всех позиций $i \in \{1, ..., l_1\}$ случилось, что $m_i^* > m_i$, из определения контрольной суммы следует, что существует позиция $j \in \{l_1 + 1, ..., l_2\}$ в части суммы такие, что $b_j^* < b_j$.

Поскольку σ^* допустима ли подпись для M^* мы имеем:

$$f_r^{(w-1)} \circ ... \circ f_r^{(b_j^*+1)}(\sigma_j^*) = pk_j.$$

Можно видеть, что событие подделки будет обнаружено, если j^{th} часть подписи законного пользователя M^* отличается от фальшивого (см. также Рис. 2), так что:

$$\tilde{\sigma_j^*} := F_r^{(b_j^*)}(sk_j) \neq \sigma_j^*.$$

Рассмотрим два возможных случая. Во-первых, условие из теоремы выполня-

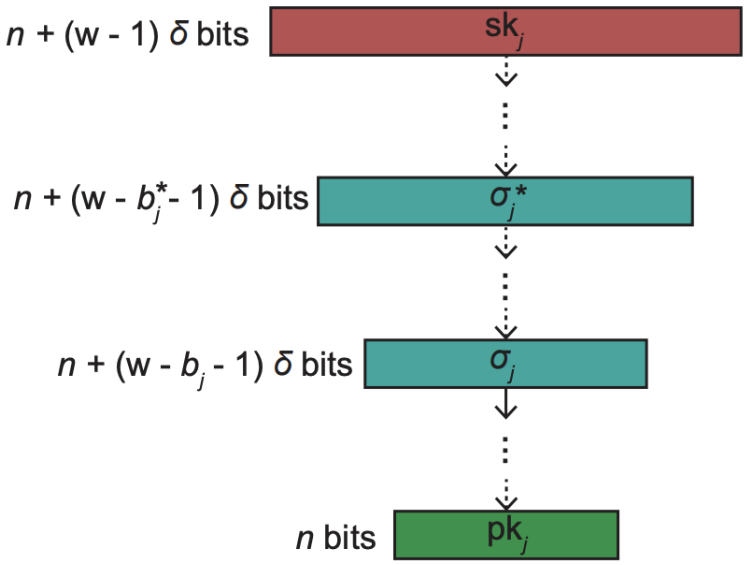


Рис. 2: Иллюстрация принципа построения доказательство подделки типа 2 для схемы $(n, \delta, L, \nu) - WOTS^+$.

ется, но справедливо следующее соотношение:

$$f_r^{(b_j)} \circ ... \circ f_r^{(b_j^*+1)}(\sigma_j^*) \neq \sigma_j.$$

В этом случае мы получаем $\tilde{\sigma_j^*} \neq \sigma_j^*$ с единичной вероятностью так как

$$\sigma_j = f_r^{(b_j)} \circ ... \circ f_r^{(b_j^*+1)}(\tilde{\sigma_j^*}) \neq f_r^{(b_j)} \circ ... \circ f_r^{(b_j^*+1)}(\sigma_j^*).$$

Во втором случае мы имеем следующее тождество:

$$f_r^{(b_j)} \circ ... \circ f_r^{(b_j^*+1)}(\sigma_j^*) = \sigma_j,$$

что автоматически подразумевает выполнение условия теоремы. Рассмотрим функцию

$$F := f_r^{(b_j)} \circ ... \circ f_r^{(b_j^*+1)} : \{0,1\}^{n^*+\delta\Delta} \to \{0,1\}^{n^*},$$

где $\Delta := b_j^0 - b_j^* \geq 1$ и $n^* := n + \delta(w - b_j^* - 1)$. Эта функция удовлетворяет случайным предположениям оракула, так как каждый из $\{f_r^{(k)}\}_{k=b_j^*}^{b_j}$. Следовательно мы имеем вероятность того, что противник получит $\sigma_j^* = \overset{\sim}{\sigma_j^*}$ с ограничением $\epsilon < 5.22 \times 2^{-\delta\Delta} \leq 5.22 \times 2^{-\delta}$. Что и требовалось доказать.

Деревья Меркля(MSS)

Первый способ создать схему многократной подписи из схемы одноразовой подписи - использовать конструкцию, предложенную Мерклом в 1989 году. Учитывая целые числа n, h и хэш-функцию $H : \{0,1\}^{2n} \to \{0,1\}^n$, так называемое Дерево Меркля представляет собой двоичное дерево высоты h, узлы которого помечены значением $x \in \{0,1\}^n$, таким образом, что значение каждого внутреннего узла вычисляется как $x = H(y||z)$, где y и z - значения левых и правых дочерних элементов.

Корневое значение r может быть сначала отправлено для последующей аутентификации любого из 2^h листового значения $v_1, ..., v_{2^h}$. Действительно, чтобы проверить, что значение v находится в листовом индексе i, нужно просто v, i и путь аутентификации i. Этот путь аутентификации содержит братьев и сестер всех узлов на пути между листом i и корнем (значения h). Это позволяет рекурсивно вычислять значения внутренних узлов вплоть до корня и сравнивать результат с r.

Эта конструкция позволяет превратить схему одноразовой подписи в схему многократной подписи следующим образом. Учитывая 2^h экземпляров OTS, подписывающий создает дерево Меркля, каждое листовое значение которого являются открытым ключом экземпляра OTS. Общий открытый ключ - это корневое значение. i-я подпись содержит подпись, сгенерированную i-м экземпляром OTS, а также путь аутентификации i.

Следовательно, открытый ключ содержит только n битов, по сравнению с подходом 2^h OTS открытых ключей. Однако время генерации ключа экспоненциально в h, потому что на этом этапе необходимо вычислить полное дерево Меркля. Например, $h = 20$ возможно, но может быть недостаточно для всех подписывающих. Кроме того, подписывающий должен отслеживать индексы i, которые уже были использованы, поэтому схема является $stateful$.

Многоразовые подписи(MTS)

В то время как одноразовые подписи обеспечивают удовлетворительную криптографическую безопасность для подписания и проверки транзакций, для них характерен существенный недостаток - их можно использовать безопасно только один раз. Поэтому существуют схемы подписи для включения более чем одной действительной одноразовой подписи, что позволяет сформировать предварительно столько подписей, сколько будет пар ключей одноразовых подписей. Логичным путем достижения этого является построение двоичного хэш-дерева, известного как дерево Меркля.

4.1 HORS

HORS - это простая схема подписи в несколько раз. Пусть f - односторонняя функция, а H - хэш-функция, которая выводит случайный размер подмножества $\{1, 2, ..., t\}$, где k и t - параметры, влияющие на безопасность с помощью $k < t$. Ключ подписи - это случайный кортеж $(s_1, ..., s_t)$, а открытым ключом является $(f(s_1), ..., f(s_t))$. Теперь, чтобы подписать m сообщение, вычислить набор $S = H(m)$ и выходной $\{s_i : i \in S\}$. Чтобы проверить, примените f к каждому элементу подписи и проверьте, соответствует ли он открытому ключу. Каждая подпись раскрывает k элементы секретного ключа, поэтому в зависимости от выбора k и t несколько сообщений могут быть подписаны до того, как безопасность будет нарушена. Это было использовано в качестве строительного блока в SPHINCS, который представляет собой схему подписи на основе хэша без состояния, которая позволяет подписывать неограниченные сообщения.

Key Generation

Input: Parameters l, k, t

Generate t random l-bit strings $s_1, s_2, \ldots, s_t$

Let $v_i = f(s_i)$ for $1 \leq i \leq t$

Output: $PK = (k, v_1, v_2, \ldots, v_t)$ and $SK = (k, s_1, s_2, \ldots, s_t)$

Signing

Input: Message m and secret key $SK = (k, s_1, s_2, \ldots, s_t)$

Let $h = Hash(m)$

Split h into k substrings $h_1, h_2, \ldots, h_k$, of length $\log_2 t$ bits each

Interpret each h_j as an integer i_j for $1 \leq j \leq k$

Output: $\sigma = (s_{i_1}, s_{i_2}, \ldots, s_{i_k})$

Verifying

Input: Message m, signature $\sigma = (s'_1, s'_2, \ldots, s'_k)$, and public key $PK = (k, v_1, v_2, \ldots, v_t)$

Let $h = Hash(m)$

Split into k substrings $h_1, h_2, \ldots, h_k$, of length $\log_2 t$ bits each

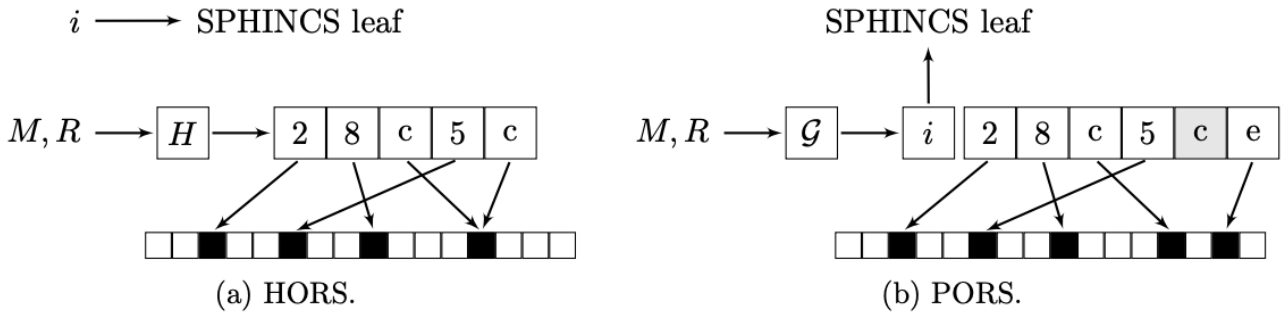
Interpret each h_j as an integer i_j for $1 \leq j \leq k$

Output: "accept" if for each j, $1 \leq j \leq k$, $f(s'_j) = v_{i_j}$; "reject" otherwise

Рис. 3: Схема подписи HORS

4.2 PORS

Начнём с того, что PORS, более безопасный вариант HORS. Как мы видели, современные схемы с несколькими временными подписями основаны на хэше для получения случайного подмножества (HORS). Тем не менее, HORS была изучена лишь частично, так как Рейзин и Рейзин(отец и сын) рассматривали только неадаптивные атаки. В частности, HORS подвержен адаптивным атакам, которые усугубляются простотой HORS: возьмите выход хэш-функции и разделите его на блоки, чтобы получить набор индексов. Действительно, ничто не мешает некоторым из этих индексов сталкиваться, уменьшая размер полученного подмножества и уменьшая безопасность. Несмотря на то, что HORS блестит своей простотой и скоростью по сравнению с более сложными методами получения случайных подмножеств гарантированного размера, его скорость не критична в сложных схемах, таких как SPHINCS, для которых голоса и деревья Меркля доминируют в вычислительных затратах. Поэтому рассмотрим новую конструкцию, использующую PRNG для получения случайного подмножества, которое мы называем PORS. Вместо того, чтобы использовать хэш-функцию, мы разделяем PRNG из сообщения и выполняем запрос к ней, пока мы не получим подмножество различных индексов к Рис. 4. Вычислительные издержки эквивалентны нескольким дополнительным вычислениям хэша для значительного повышения безопасности. В случае SPHINCS заметим, что противники имеют полный контроль над выбранным листом в гипердереве. Вместо этого мы предлагаем создать этот листовой индекс с помощью PRNG, что ещё больше повысит уровень безопасности. Этот увеличенный запас прочности позволяет уменьшить высоту гипердерева на 2 слоя, экономя 4616 байт.



(a) HORS. (b) PORS.

Рис. 4: Сравнение HORS и PORS

Подписи без состояния

5.1 SPHINCS

Представим основные идеи $SPHINCS$, описав его как комбинацию четырех типов деревьев. Ниже перечислены четыре типа деревьев (см. Рис. 5).

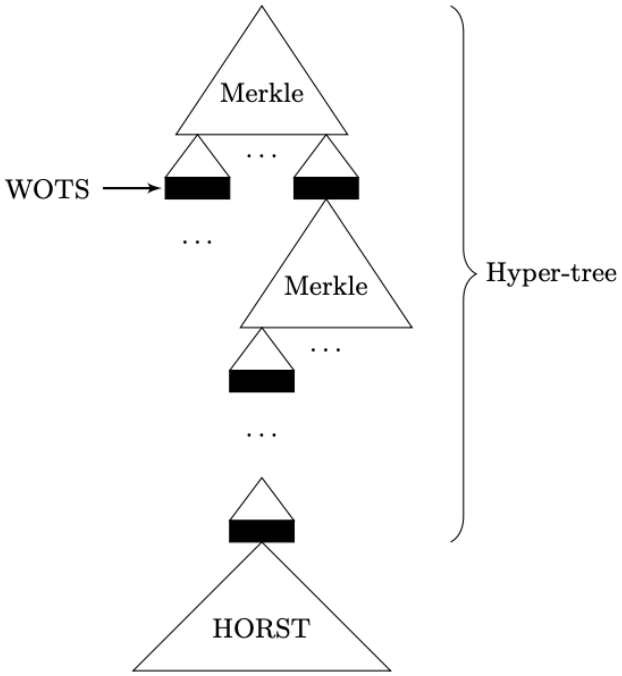


Рис. 5: Пример $SPHINCS$. Гипердерево состоит из d слоев дерева Меркля и соединены $WOTS$. Внизу дерево $HORS$(или $HORST$) соединяется с подписанным сообщением.

1. Главное Гипердерево, высотой h (60 в $SPHINCS-256$). Корень этого дерева является частью открытого ключа. Листья этого дерева экземпляры $HORST$. Это Гипердерево делится на d слоев($d=12$ в $SPHINCS-256$).

2. Поддеревья, которые являются деревьями Меркля высоты h/d ($60/12=5$ в $SPHINCS-256$). Листья этих деревьев являются корнями деревьев; указанные корни являются сжатыми открытыми ключами экземпляров $WOTS$, которые соединяются с деревом на следующем уровне.

3. Открытый ключ $WOTS$ это деревья сжатия, которые являются L-деревьями, высоты $\lceil log_2 l \rceil$, когда есть l листья. Листья этого дерева являются компонентами $WOTS$ открытого ключа (67 значений по 256 бит каждое в $SPHINCS-256$). Связанный экземпляр $WOTS$ подписывает корень дерева на следующем уровне.

4. В нижней части гипердерева, открытый ключ $HORST$ - деревья сжатия это деревья Меркля высоты $\tau = log_2 t$, где t номер элементов открытого ключа $HORST$(2^{16} в $SPHINCS-256$).

Подписание в $SPHINCS$ работает следующим образом.

1. Извлекается листовой индекс из сообщения и личного ключа. Этот индекс определяет один из экземпляров $2^h HORST$ (относительно основного гипердерева), который будет использоваться для подписи сообщения.

2. Создайте экземпляр $HORST$, который является производным от личного ключа и конечного индекса, и подпишите сообщение этим экземпляром $HORST$. Подпись $HORST$ включает k ключей и их соответствующие пути аутентификации и является частью подписи $SPHINCS$. Получите сжатый в дереве $HORST$ открытый ключ p.

3. Для каждого слоя гипердерева подпишите открытый ключ p(полученный из нижнего слоя), используя правильный экземпляр $WOTS$(полученный из листового индекса); добавьте эту подпись $WOTS$ и связанный с ней путь аутентификации к подписи $SPHINCS$. Вычислите путь аутентификации этого экземпляра $WOTS$ в поддереве. Добавьте этот путь к подписи $SPHINCS$ и p-корень поддерева.

Это краткое описание $SPHINCS$.

5.2 Gravity-SPHINCS

$Gravity - SPHINCS$ наследуют некоторые параметры от $SPHINCS$ (длина хэша, глубина $WOTS$ и др.), и имеет новые. В приведенном ниже списке h обозначает высоту поддеревьев(в отличие от высоты основного дерева в $SPHINCS$), а $B_n = \{0,1\}^n$ обозначает набор n-битовых строк. Параметры являются следующими:

- Хэш-выход длина бита n, положительное целое число.
- Глубина $WOTS$ w, степень 2-ки такой, что $w \geq 2$ и $log_2 w$ делит n.
- Размер множества $PORS$ t, положительное, степень двойки.
- Размер подмножества $PORS$ k, положительное целое такое, что $k \leq t$.
- Высота дерева Меркля h, положительное целое.
- Количество внутренних деревьев Меркля d, неотрицательное целое.
- Высота кэша c, неотрицательное целое.
- Высота $batching$ b, неотрицательное целое.
- Пространство сообщения M, обычно подмножество битовых строк $\{0,1\}^*$.

Из этих параметров получены:

- Размер $WOTS$ $l = \mu + \lfloor log_2(\mu(w-1))/log_2 w \rfloor + 1$, где $\mu = n/log_2 w$.
- Множество $PORS$, $T = \{0, ..., t-1\}$.
- Адресное пространство $A = \{0, ..., d\} \times \{0, ..., 2^{c+dh} - 1\} \times \{0, ..., max(l,t) - 1\}$.
- Пространство открытый ключей $PK = B_n$.
- Пространство личных ключей $SK = B_n^2$.
- Пространство подписи $SG = B_n \times B_n^k \times B_n^{\leq k(log_2 t - \lfloor log_2 k \rfloor)} \times (B_n^l \times B_n^h)^d \times B_n^c$.
- $SG_B = B_n^b \times \{0, ..., 2^b - 1\} \times SG$
- Размер публичного ключа n бит.
- Размер личного ключа, $2n$ бит.

- Максимальный размер подписи

$$sigsz = (1 + k + k(log_2 t - \lfloor log_2 k \rfloor) + d(l + h) + c)n$$

бит.

Подписи S одного сообщения и проверка V в $Gravity - SPHINCS$ очень похожа на $SPHINCS$.

Алгоритм генерации ключей. KG получает на вход $2n$ случайных бит и на выходе получаем личный ключ $sk \in B_n^2$, и открытый ключ $pk \in B_n$.

- Генерация личного ключа из $2n$ случайных бит $sk = (seed, salt) \xleftarrow{\$} B_n^2$.
- Для $0 \leq i < 2^{c+h}$ генерируется $Winternitz$ открытый ключ

$$x_i \leftarrow WOTS - genpk(seed, make - addr(0, i))$$

- Генерация публичного ключа $pk \leftarrow Merkle - root_{c+h}(x_0, ..., x_{2^{c+h}-1})$.

Алгоритм подписи. S на вход принимает хэш $m \in B_n$ и личный ключ $sk = (seed, salt)$, и на выходе получаем подпись.

- Вычисляем $s \leftarrow H(salt, m)$.
- Вычисляем гипердерева индекс и случайное подмножество как

$$j, (x_1, ..., x_k) \leftarrow PORS(s, m)$$

- Вычисляем $PORST$ подпись и открытый ключ

$$(\sigma_d, oct, p) \leftarrow PORST - sign(seed, make - addr(d, j), x_1, ..., x_k)$$

- Для $i \in \{d - 1, ..., 0\}$ выполняется:

1. Вычисляем $WOTS$ подпись $\sigma_i \leftarrow WOTS - sign(seed, make - addr(i, j), p)$,
2. Вычисляем $p \leftarrow WOTS - extractpk(p, \sigma_i)$,
3. $j^* \leftarrow \lfloor j/2^h \rfloor$,
4. Для $u \in \{0, ..., 2^h - 1\}$ вычислим $WOTS$ открытый ключ

$$p_u \leftarrow WOTS - genpk(seed, make - addr(i, 2^h, j^* + u))$$

5. Вычислим Меркля аутентификацию $A_i \leftarrow Merkle - auth_h(p_0, ..., p_{2^h-1}, j - 2^h j^*)$,
6. $j \leftarrow j^*$.

- Для $0 \leq u < 2^{c+h}$ вычислим WOTS открытый ключ

$$p_u \leftarrow WOTS - genpk(seed, make - addr(0, u))$$

- Вычислим Меркля аутентификацию

$$(a_1, ..., a_{h+c}) \leftarrow Merkle - auth_{h+c}(p_0, ..., p_{2^{h+c}-1}, 2^h j)$$

- $A_c \leftarrow (a_{h+1}, ..., a_{h+c})$.
- Получаем подпись $(s, \sigma_d, oct, \sigma_{d-1}, A_{d-1}, ..., \sigma_0, A_0, A_c)$.

Алгоритм проверки. V получает на вход хэш $m \in B_n$, открытый ключ $pk \in B_n$ и подпись

$$(s, \sigma_d, oct, \sigma_{d-1}, A_{d-1}, ..., \sigma_0, A_0, A_c)$$

и проверяет это следующим образом.

- Вычислим индекс гипердерева и случайное подмножество

$$j, (x_1, ..., x_k) \leftarrow PORS(s, m)$$

- Вычислим открытый ключ $PORST$,

$$p \leftarrow PORST - extractpk(x_1, ..., x_k, \sigma_d, oct).$$

- Если $p = \perp$, затем прерываем и возвращаем 0.
- Для $i \in \{d-1, ..., 0\}$ выполняем следующее:
1. Вычислим открытый ключ $WOTS$, $p \leftarrow WOTS - extractpk(p, \sigma_i)$,
2. $j^* \leftarrow \lfloor j/2^h \rfloor$,
3. Вычислим корень дерева Меркля, $p \leftarrow Merkle - extract_h(p, j - 2^h j^*, A_i)$,
4. $j \leftarrow j^*$.
- Вычислим корень дерева Меркля, $p \leftarrow Merkle - extract_c(p, j, A_c)$.
- В результате 1, если $p = pk$ и 0 иначе.

5.3 SPHINCS+

$SPHINCS^+$ использует псевдослучайную функцию PRF для псевдослучайности генерации ключей, $PRF : \{0,1\}^n \times \{0,1\}^{256} \rightarrow \{0,1\}^n$, и псевдослучайную функцию PRF_{msg} для генерации случайного сжатия сообщения: $PRF_{msg} : \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^* \rightarrow \{0,1\}^n$. Для сжатия подписываемого сообщения мы используем дополнительную хэш-функцию H_{msg}, которая может обрабатывать сообщения произвольной длины:

$$H_{msg} : \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^* \rightarrow \{0,1\}^m$$

$SPHINCS^+$ Личный и открытый ключи. Открытый ключ состоит из двух n-битных значений: корневого узла из трех верхних в гипердереве и случайного открытого начального значения PK. Кроме того, личный ключ состоит еще из двух n-битных случайных: SK, чтобы генерировать $WOTS^+$ и $FORS$ личные ключи, и $SK.prf$, используемый ниже для рандомизированного дайджеста сообщений.

$SPHINCS^+$ Подпись. Как не удивительно, что подпись состоит из $FORS$ подписи для дайджеста сообщения, $WOTS^+$ подпись соответствующих открытых ключей $FORS$, и ряд каналов аутентификации и $WOTS^+$ подписи для подтверждения того, что $WOTS^+$ открытый ключ. Чтобы проверить эту цепочку путей и подписей, проверка итеративно восстанавливает открытые ключи и корневые узлы до тех пор, пока не будет достигнут корневой узел в верхней

части гипердерева $SPHINCS^+$. Два момента еще не были рассмотрены: вычисление дайджеста сообщения и выбор листа. Здесь $SPHINCS^+$ отличается от оригинальных $SPHINCS$ тонкими, но важными деталями.

Во-первых, мы псевдо случайным образом генерируем рандомизатор R, основанный на сообщении и $SK.prf$. R может быть дополнительно сделан недетерминированным путем добавления дополнительной случайности $OptRand$. Это может противодействовать атакам бокового канала, которые полагаются на сбор нескольких следов для одного и того же вычисления. Обратите внимание, что установка этого значения в нулевую строку (или использование значения с низкой энтропией) не оказывает отрицательного влияния на псевдослучайность R. Формально, мы полагем, что $R = PRF(SK.prf, OptRand, M)$. R часть подписи. Используя R, мы затем получаем индекс конечного узла, который должен использоваться, а также дайджест сообщения $(MD||idx) = H_{msg}(R, PK, PK.root, M)$.

В отличие от $SPHINCS$, этот метод выбора индекса является публично проверяемым, не позволяя злоумышленнику свободно выбирать кажущийся случайным индекс и комбинировать его с сообщением по своему выбору. Критически важно, что это противодействует многоцелевым атакам на схему подписи нескольких раз. Поскольку индекс теперь может быть вычислен верификатором, он больше не включается в сигнатуру.

Stateful vs Stateless

Схемы с сохранением состояния имеют дерево Меркля с количеством одноразовых подписей внизу. Каждая разовая подпись может быть использована один раз, следовательно, подписывающий должен отслеживать, какие из них он использовал. То есть, когда он использует одноразовую подпись для подписи сообщения, он должен обновить свое состояние.

Схемы без состояния имеют большое дерево, но внизу у них есть несколько подписей времени. Каждая такая небольшая временная подпись может подписать несколько сообщений. Таким образом, когда подписывается сообщение, подписывающий выбирает случайную подпись с небольшим количеством времени, использует ее для подписи сообщения, а затем подтверждает ее подлинность через деревья Меркля вплоть до корня, который является открытым ключом. Поскольку мы используем несколько раз подпись, мы не против, если мы иногда выбираем одну и ту же подпись несколько раз. Схема подписи нескольких раз может справиться с этим. И, поскольку нам не нужно обновлять какое-либо состояние при генерации подписи, это считается «без сохранения состояния».

Заключение

Литература