Протокол именной ЭЦП на основе схемы Шнорра с использованием эллиптических кривых

Т. А. Бжезинский
Научный руководитель - С. В. Агиевич

Содержание

1 Общие сведения об ЭЦП, схема Шнорра

В настоящее время в связи с все большими объемами информации и построением информационного общества очень остро встает проблема защиты и хранения информации. В связи с этим ведутся разработки в области шифрования, кодирования и хэширования данных. Однако немаловажным аспектом в работе с информацией является верификация каких-то данных, некий аналог подписи документа. Таким аналогом является электронная цифровая подпись (далее – ЭЦП). Приведем выдержку из постановления Совета Министров Республики Беларусь:

> ЭЦП – последовательность символов, являющаяся реквизитом электронного документа и предназначенная для подтверждения целостности и подлинности электронного документа. Средство электронной цифровой подписи – программное, программно-аппаратное или техническое средство, реализующее одну или несколько следующих функций: выработку электронной цифровой подписи, проверку электронной цифровой подписи, создание личного ключа подписи или открытого ключа.

Рассмотрим принципы работы ЭЦП на основе одной из самых изученных и стойких схем – схемы Шнорра [3].

Схема Шнорра включает в себя 3 алгоритма:

- Алгоритм $\mathcal{GEN}$ - генерация параметров ТДС (третья доверенная сторона, Key Authentication center).
- Алгоритм $\mathcal{SIGN}$ - выработка ЭЦП.
- Алгоритм $\mathcal{VERIFY}$ - проверка ЭЦП.

Следует отметить, что данные 3 этапа являются общими для всех схем ЭЦП. Рассмотрим каждый из этапов подробнее.

На этапе $\mathcal{GEN}$ выполняются следующие действия:

1. Выбираются простые числа $p,\ q$ такие, что $q|p-1, q \geq 2^{140}, p \geq 2^{512}$.
2. Выбирается $\alpha \in \mathbb{Z}_p$ такое, что $\alpha^q \equiv 1\ (mod\ p),\ \alpha \neq 1$.
3. Выбирается необратимая хэш-функция $H : \mathbb{Z}_q \times \mathbb{Z} \longrightarrow \{0, 1, \dots, 2^t - 1\}$, где t - параметр безопасности схемы (в классической работе Шнорра [3] принимается $t = 72$).

После инициализации публикуются параметры p, q, α, h. Каждый пользователь генерирует свой личный и открытый ключи по следующему алгоритму:

1. $s \xleftarrow{R} \{1, 2, \dots, q-1\}$.
2. $v \leftarrow \alpha^{-s}\ (mod\ p)$.
3. Вернуть пару $(s, v) =$ (*личный ключ, открытый ключ*).

Алгоритм $\mathcal{SIGN}$ принимает на вход 2 параметра: сообщение m и личный ключ s и формирует подпись по следующему алгоритму:

1. $r \xleftarrow{R} \{1, 2, \dots, q-1\}$.
2. $x \leftarrow \alpha^r\ (mod\ p)$.

3. $e \leftarrow H(x, m)$.

4. $y \leftarrow r + s \cdot e \ (mod\ q)$.

5. Уничтожается число r и пара (e, y) является подписью для сообщения m.

Алгоритм $\mathcal{VERIFY}$ проверяет подпись для сообщения m, подписи (e, y) и открытого ключа v следующим образом:

1. $\bar{x} \leftarrow \alpha^y \cdot v^e \ (mod\ p)$.

2. Если $e = H(\bar{x}, m)$, то вернуть ***ДА***. Иначе вернуть ***НЕТ***.

При рассмотрении данной схемы становится достаточно очевидно, что алгоритмы ЭЦП Шнорра можно перенести в произвольную циклическую группу. Будем рассматривать группу, порожденную элементом G, и использовать аддитивные обозначения.

Пусть $|\langle G\rangle| = q$, $d \in \{1, 2, .., q-1\}$ - личный ключ, $Q = dG$ - открытый ключ. Тогда скажем, что подписью Шнорра к сообщению X является решение $\sigma = (s_0, s_1) \in \mathbb{Z}_q \times \mathbb{Z}_q$ следующего уравнения:

$$\varphi(X, s_1G + s_0Q) = s_0 \tag{1}$$

Владелец личного ключа решает его следующим образом:

1. $k \xleftarrow{R} \{1, 2, \ldots, q-1\}$.

2. $R \leftarrow kG$.

3. $s_0 \leftarrow \varphi(X, R)$.

4. $s_1 \leftarrow k - s_0d \ (mod\ q)$.

Так как $s_1G + s_0Q = (k - s_0d)G + s_0dG = (k - s_0d + s_0d)G = kG = R$, то, следовательно,

$$\varphi(X, s_1G + s_0Q) = \varphi(X, R) = s_0$$

Уравнение 1 составлено так, что противнику, который не знает d, для определения (s_0, s_1) требуется решать трудные задачи, связанные с обращением φ или дискретным логарифмированием в $\langle G\rangle$.

Схема ЭЦП считается криптографически надежной, если для нее вычислительно трудно решить задачу «единичная подделка», т.е. при заданных параметрах ТДС и Q сложно решается 1 относительно σ. Также отметим, что при решении задачи "единичная подделка"разрешается получать и использовать другие решения, с другими подписываемыми сообщениями X.

При доказательстве стойкости схем ЭЦП часто используется модель *Random Oracle Model* (*ROM*): значения φ определяются гипотетическим оракулом (вероятностным алгоритмом) H случайно, независимо, равновероятно. Одним из доказательств стойкости схемы ЭЦП Шнорра служит следующая теорема (Пойнтчеваль, Стерн, 2000):

Теорема 1. *Пусть имеется алгоритм, который решает задачу «единичная подделка» для ЭЦП Шнорра в модели ROM за время T с вероятностью успеха $\varepsilon \geq \frac{7D}{q}$, где D — число вопросов оракулу H. Тогда данный алгоритм можно преобразовать в алгоритм решения задачи дискретного логарифмирования за время, меньшее $\frac{84480DT}{\varepsilon}$.*

2 Выбор $\langle G \rangle$

Введем следующее определение:

Определение 1. *Группой Шнорра - циклическая группа* $\langle G \rangle \subset \mathbb{F}_p^{\star}, |G| = q$, p, q *- простые.*

Заметим, что группа, использованная в классической работе [3], является как раз группой Шнорра(требование $q|p-1$ вытекает немедленно из теоремы Лагранжа). До недавнего времени были известны лишь экспоненциальные алгоритмы решения задачи DL для групп Шнорра, к примеру ρ-метод Полларда, имеющий сложность $O(\sqrt{q})$. Однако, в последнее время были открыты и субкспоненциальные алгоритмы решения задачи DL в группах Шнорра, например GNFS (General Number Field Sieve, Общее решето цифрового поля), который имеет сложность $L_n[\alpha, c]$ при $\alpha = \frac{1}{3}$ и $c = \left(\frac{64}{9}\right)^{\frac{1}{3}}$, где

$$L_n[\alpha, c] = \exp\left(c + o(1)\right) (\log n)^{\alpha} (\log \log n)^{1-\alpha} \tag{2}$$

Данный алгоритм делает схему ЭЦП Шнорра менее стойкой, и как следствие, необходимо строить схему ЭЦП на такой группе $\langle G \rangle$, для которой не существует субэкспоненциальных алгоритмов на данный момент.

Одной из таких возможных групп является группа точек эллиптической кривой.

Определение 2. *Эллиптическая кривая - это кривая над* $\mathbb{F}_p$ *в двумерном евклидовом пространстве (p - простое) вида*

$$\begin{cases} y^2 = x^3 + ax + b, \\ a, b, x, y \in \mathbb{F}_p \end{cases} \tag{3}$$

Обозначим $E_{a,b}^{\star}(\mathbb{F}_p)$ - множество таких точек (x, y), которые удовлетворяют уравнению 3 . Данное множество называют аффиными точками кривой. Добавим к аффинным точкам специальную бесконечно удаленную точку O и образуем множество $E_{a,b}(\mathbb{F}_p)$. Пусть $4a^3 + 27b^2 \neq 0 \ (mod\ p)$, тогда множество $E_{a,b}(\mathbb{F}_p)$ будет являться аддитивной группой (все вычисления ведутся в $\mathbb{F}_p$) при следующих правилах сложения:

1. $O + P = P + O = P \quad \forall P \in E_{a,b}(\mathbb{F}_p)$.
2. Если $P = (x, y) \in E_{a,b}(\mathbb{F}_p)$, то $-P = (x, p - y)$ и $P + (-P) = O$
3. Если $P_1 = (x_1, y_1) \in E_{a,b}(\mathbb{F}_p)$ и $P_2 = (x_2, y_2) \in E_{a,b}(\mathbb{F}_p)$ и $P2 \neq (-P1)$, то$P_1 + P_2 = P_3 = (x_3, y_3)$, где

$$x_3 = \lambda^2 - x_1 - x_2,\ y_3 = \lambda(x_1 - x_3) - y_1,\ \lambda = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1}, & P_1 \neq P_2 \\ \frac{3x_1^2 + a}{2y_1}, & P_1 = P_2 \end{cases}$$

.

Сумма k экземпляров точки P называется k-кратной ей точкой и обозначается через kP. Также считается, что $0P = O$. Использование такой группы точек используется в некоторых алгоритмах и стандартах, таких как ECDSA, ГОСТ Р 34.10-2001, [8] и других.

3 Концепция IBS - Id Based Signature

3.1 История развития именных криптосистем и общие сведения об IBS

В рассмотренных выше схемах ЭЦП на практике появляется некоторая прослойка - это сертификат открытого ключа - цифровой или бумажный документ, подтверждающий соответствие между открытым ключом и информацией, идентифицирующей владельца ключа. Содержит информацию о владельце ключа, сведения об открытом ключе, его назначении и области применения, название центра сертификации и т. д. При достаточно большой вычислительной сложности ЭЦП наличие сертификата замедляет работу схемы. Также немаловажную роль играет стоимость сертификата, что не является фактором в его пользу.

Шамиром в 1984 году в [4] была предлжена концепция идентификационных (именных, ID) криптосистем. Согласно этой концепции, открытый ключ не распространяется некоторым достоверным образом, а вычисляется по идентификатору пользователя и открытому ключу ТДС.

Долгое время концепция ID-криптосистем оставалась только концепцией – не было найдено способов ее реализации с помощью доступных криптографических и математических примитивов. И только в начале нашего века ID-криптография начала развиваться в практической плоскости. В частности, были найдены способы реализации концепции ID-систем ЭЦП (IBS, ID-based signature).

В IBS применяется следующая схема:

- Каждого пользователя характеризирует определенный ID, при этом ID должны быть различны. ID является общедоступной информацией. ID выступает аналогом открытого ключа.
- По входному параметру безопасности l вырабатываются личный и открытые ключи ТДС.
- На основе своего ID каждый пользователь по алгоритму $\mathcal{EXTRACT}$ генерирует личный ключ d_{ID} который является секретной информацией. Алгоритм $\mathcal{EXTRACT}$ использует ID пользователя, личный d_T и открытый Q_T ключи ТДС.
- Алгоритм $\mathcal{SIGN}$ принимает на вход личный ключ пользователя d_{ID}, открытый ключ ТДС Q_T, сообщение X и возвращает подпись σ.
- Алгоритм $\mathcal{VERIFY}$ принимает на вход сообщение X, подпись σ, идентификатор пользователя ID, открытый ключ ТДС Q_T и выдает вердикт о подлинности подписи.

Одним из новшеств подобных схем является то, что часть личного ключа d_{ID} является доступной информацией.

Рассмотрим 2 схемы IBS [1, 2] и на их примере проведем анализ и структуру алгоритмов, а также оценим их сложность. Схема [1] была предложена европейскими учеными - криптографами на основе вышеупомянутой схемы Шнорра в 2009 году. Схема [2] была предложена в 2010 году китайскими авторами. Целесообразно сравнить по общим критериям, таким как:

1. Использование эллиптических кривых;
2. Основа стойкости;
3. Личный и открытый ключи ТДС;
4. Инициализация ТДС;

5. Вычислительная сложность алгоритмов $\mathcal{SIGN}$ и $\mathcal{VERIFY}$;

6. Принцип генерации личного ключа пользователя;

7. Стойкость схемы.

Данные удобно поместить в таблицу. Не нарушая общности, будем считать что все хэш-функции, рассматриваемые нами ниже, действуют из множества бинарных слов в классы вычетов по какому-то модулю, т.е. $\{0,1\}^\star \longrightarrow \mathbb{F}_c$, где c - простое.

Схема/Критерий	[1]	[2]
Использование э.к.	$-$	$+$
Основа стойкости	Задача DL над конечным полем	Задача DL над эл. кривыми
Параметры инициализации ТДС	Уровень безопасности $l \in \mathbb{N}$	Уровень безопасности $l \in \mathbb{N}$
Открытый ключ ТДС	Группа $\mathbb{F}_q$, q - простое хэш-функции φ_0, φ_1 генератор группы G $d_T G$	$E_{a,b}(\mathbb{F}_p)$,$\vert E_{a,b}(\mathbb{F}_p)\vert = q$, q, p - простые хэш-функции φ_0, φ_1 генератор группы G $d_T G$
Личный ключ ТДС	$d_T \overset{R}{\leftarrow} \mathbb{F}_q$	$d_T \overset{R}{\leftarrow} \mathbb{F}_q$
Вычислительная сложность инициализации ТДС	Генерация q - простого и нахождение первообразного корня	Генерация $E_{a,b}(\mathbb{F}_p)$ заданного порядка и генератора группы

Таблица 1: Сравнительная характеристика 2 схем IBS

Из таблицы видно, что, в отличие от классических схем типа Шнорра, в данных схемах используется уже две хэш-функции. Из таблицы можно заметить, что схемы практически не отличаются в принципе генерации ключей ТДС и сложности его инициализации. При подробном анализе алгоритмов $\mathcal{EXTRACT}$, $\mathcal{VERIFY}$, $\mathcal{SIGN}$ в обеих схемах можно прийти к аналогичному выводу, поэтому приведем описание лишь одной из схем, например [1] (в аддитивных обозначениях).

3.2 Пример реализации IBS [1]

3.2.1 Алгоритм $\mathcal{EXTRACT}$

Алгоритм $\mathcal{EXTRACT}$ принимает на вход идентификатор пользователя ID и генерирует личный ключ d_{ID} следующим образом:

1. $k_{ID} \overset{R}{\leftarrow} \mathbb{F}_q$.

2. $R_{ID} \leftarrow k_{ID} G$.

3. $s_{ID} \leftarrow k_{ID} + d_T \cdot \varphi_0(ID, R_{ID}) \ (mod\ q)$.

4. $d_{ID} \leftarrow (s_{ID}, R_{ID})$.

3.2.2 Алгоритм $\mathcal{SIGN}$

Алгоритм $\mathcal{SIGN}$ принимает на вход сообщение X, открытый ключ ТДС Q_T, личный ключ пользователя $d_{ID} = (s_{ID}, R_{ID})$ и вырабатывает подпись σ таким методом:

1. $k \overset{R}{\leftarrow} \mathbb{F}_q$.
2. $R \leftarrow kG$.
3. $s \leftarrow k + s_{ID} \cdot \varphi_1(ID, X, R)$.
4. $\sigma \leftarrow (s, R, R_{ID})$
5. Вернуть σ в качестве результата.

Заметим, что kG является известной информацией (часть подписи), являясь одновременно и частью личного ключа.

3.2.3 Алгоритм $\mathcal{VERIFY}$

Алгоритм $\mathcal{VERIFY}$ проверяет подпись σ по сообщению X, открытому ключу ТДС Q_T и идентификатору пользователя ID:

1. $c \leftarrow \varphi_0(ID, R_{ID})$
2. $e \leftarrow \varphi_1(ID, X, R)$
3. Если $sG = R + e\,(cQ_T + R_{ID})$, то вернуть ***ДА***. Иначе вернуть ***НЕТ***.

Так как (вычисления всех скаляров ведутся в $\mathbb{F}_q$)

$$\begin{aligned} sG &= (k + s_{ID}\varphi_1(ID, X, R))G = kG + \varphi_1(ID, X, R)(k_{ID} + d_T\varphi_0(ID, R_{ID}))G = \\ &= R + \varphi_1(ID, X, R)k_{ID}G + \varphi_1(ID, X, R)\varphi_0(ID, R_{ID})d_TG = \\ &= R + \varphi_1(ID, X, R)R_{ID} + \varphi_1(ID, X, R)\varphi_0(ID, R_{ID})Q_T = \\ &= [e = \varphi_1(ID, X, R), c = \varphi_0(ID, R_{ID})] = R + e(cQ_T + R_{ID}), \end{aligned} \tag{4}$$

то алгоритм $\mathcal{VERIFY}$ является корректным.

3.3 Стойкость схем

Обратимся к доказательству стойкости схем. В [1, 2] приводятся следующие теоремы:

Теорема 2. *Если у нас есть противник A , который умеет решать задачу DL над конечным полем с вероятностью p_A , то тогда при наличии противника B, решающего задачу взлома [1] за Q обращений к хэш-функции φ_0 с вероятностью p_B , мы можем оценить p_A снизу:*

$$p_A \geq \frac{p_(\nu)}{Q} \cdot \left(\frac{p_B(\nu)}{Q^2} - \frac{1}{2^\nu}\right)$$

Теорема 3. *Если [2] взламывается с помощью q_H запросов к хэш-функции H и q_S обращений к алгоритму подписи с вероятностью $\varepsilon \geq \frac{10(q_H+1)(q_H+q_S)}{2^\nu}$ и за время t, то в данном случае задача DL над эллиптическими кривыми решаема за время $t\prime \leq \frac{23q_Ht}{\varepsilon}$ с вероятностью $\varepsilon\prime \geq \frac{1}{9}$.*

Обе теоремы доказываются через обобщенную лемму о разветвлении (general forking lemma) из [5].

Как мы видим, обе схемы достаточно надежны. Мы выберем схему [1] в виду более простых вычислений в стандартных алгоритмах, и на ее основе построим свою схему именной ЭЦП.

3.4 Некоторые нововведения и оптимизации нашей схемы

При сравнении схем ЭЦП наиболее ключевым критерием после стойкости и теоретической основы является:

1. Длины подписи и ключа.
2. Быстрота работы алгоритмов ЭЦП.

Применим некоторые оптимизации к схеме [1]. Так, рассмотрим шаг 3 в алгоритме $\mathcal{SIGN}$. На этом шаге в качестве одного из аргументов хэш-функции φ_1 используется идентификаторп пользователя ID. Однако, можно заметить, что идентификатор ID используется и в генерации личного ключа, а значит, используется еще раз в алгоритме $\mathcal{SIGN}$. Попробуем исключить использование ID при вычислении значения функции φ_1 на шаге 3 алгоритма $\mathcal{SIGN}$. Будем вести запись в терминах алгоритмов ЭЦП. Заменим шаг 3 на вычисление следующего выражения:

$$s \leftarrow k + s_{ID} \cdot \varphi_1(X, R) \tag{5}$$

Предположим, что алгоритм $\mathcal{SIGN}$ модифицирован в соотвествие с 5, посмотрим на изменения в алгоритме $\mathcal{VERIFY}$ и ответим на вопрос, допустимы ли такие изменения.

Как известно, алгоритм $\mathcal{VERIFY}$ принимает на вход параметры, и каждый из этих параметров является важным и несет смысл. Рассмотрим шаг 2 алгоритма $\mathcal{VERIFY}$. Заменим его вычислением следующего выражения:

$$e \leftarrow \varphi_1(X, R) \tag{6}$$

Проверим, являются ли новые версии алгоритмов корректными. В алгоритме $\mathcal{VERIFY}$ продолжают использоваться все 4 входных параметра, следовательно нам необходимо ответить на вопрос о корректности в вычислительном смысле. Очевидно, что изменится лишь значение элемента, равного значению функции φ_1, и при подстановке в 4 равенство будет соблюдаться. Используем эту оптимизацию в нашей схеме.

Рассмотрим далее следующую модификацию алгоритма $\mathcal{SIGN}$: Будем возвращать не $\sigma \leftarrow (s, R, R_{ID})$, а

$$\sigma \leftarrow (sG - R, R, R_{ID})$$

. При этом увеличивается вычислительная сложность алгоритма выработки ЭЦП из-за необходимости вычисления значения выражения вида $\alpha A + \beta B$, где α, β - коэффициенты из $\mathbb{F}_p$, а A, B - точки эллиптической кривой.

Однако, в таком случае эта операция не будет выполняться в алгоритме проверки ЭЦП, что ведет к ускорению работы алгоритма проверки. При предположении, что подпись может быть проверена многократно, ускорение является существенным. Однако, при подобном ускорении увеличивается на l бит длина подписи, следовательно, важным также будет являться механизм передачи σ и скорость этой передачи. В нашей схеме данная модификация использоваться не будет.

Обсудим необходимость введения 2 хэш-функций. В доказательстве стойкости схемы [1] для этих функций используются два ROMа, при этом одна из оценок стойкости (приведенная в настоящей работе) дается лишь с использованием числа обращений к φ_0. Рассмотрим случай, когда $\varphi_0 = \varphi_1$.

При предположении того, что обе функции являются надежными, обратим внимание на следующее: функции φ_0, φ_1 принимают на вход различные аргументы, а поскольку каждая из них является надежной, то при их равенстве соотвествующие значения будут "разведены"и не будут кореллировать. В связи с этим будет целесообразно свести все к одной хэш-функции

$$\varphi = \varphi_0 = \varphi_1$$

, что и будет сделано в нашей схеме.

Таким образом, нашими нововведениями будут:

- Оптимизация алгоритмов выработки и проверки ЭЦП (нету необходимости включать ID в аргументы функции φ_1.
- Будем использовать одну хэш-функцию вместо 2.

4 Описание нашей схемы ЭЦП

В Республике Беларусь вводится стандарт ЭЦП на ЭК [8]. Стандарт определяет алгоритмы генерации и проверки параметров ЭК, генерации и проверки ключей ЭЦП Шнорра, собственно алгоритмы выработки и проверки ЭЦП, вспомогательные алгоритмы и примитивы.

Для решения задачи построения IBS схемы для РБ естественно использовать примитивы [8] в качестве основы.

В настоящем разделе определяется спецификация IBS, основанная на стандарте ЭЦП.

4.1 Назначение

Рассмотрим примитивы и вспомогательные алгоритмы, а также определения, необходимые для работы IBS схемы. Будем использовать обозначения [8] и возьмем оттуда некоторые примитивы и обозначения.

4.1.1 Определения и обозначения

1. $E_{a,b}(\mathbb{F}_p)$ - группа точек эллиптической кривой, определенная в разделе 2.
2. Параметр безопасности l схемы - такое минимальное $l \in \mathbb{N}$, что $2^{l-1} < p < 2^l$, где p определено при построении группы $E_{a,b}(\mathbb{F}_p)$.
3. d_{ID} - личный ключ пользователя с идентификатором ID.
4. (a, b, p, q, G) - общеизвестные параметры ТДС, где G - генератор группы, q - порядок группы $E_{a,b}(\mathbb{F}_p)$.
5. $d_T \in \{1, 2, \dots, q-1\}$ - личный ключ ТДС
6. $Q_T = d_T G$ - открытый ключ ТДС.

4.1.2 Алгоритмы [8]

Возьмем следующие алгоритмы, определенные в [8]:

- Генерация $E_{a,b}(\mathbb{F}_p)$ и генератора G по заданному l с порядком группы q - $\mathcal{GENERATEEC}$
- Функция хэширования $hBelt_l$.
- Генерация псевдослучайных чисел.

4.1.3 Примитивы нашей схемы

Введем дополнительные обозначения:

- Функция хэширования φ, которую определим следующим образом:
$$\varphi(x) = hBelt_l(x) \ (mod \ q)$$
.
- Алгоритм $\mathcal{EXTRACT}$, генерирующий ключи ТДС.
- Функция $\pi : E_{a,b}(\mathbb{F}_p) \to \{0,1\}^{2l}$, которая каждой точке ставит в соотвествие двоичное число и работает следующим образом:
$$C = (x, y) \to (\langle x \rangle_l \ || \ \langle y \rangle_l)$$
, где $\langle val \rangle_{len}$ - двоичная запись числа val, дополненная при необходимости лидирующими нулями до длины len.

4.1.4 Работа схемы

Работу схемы представим следующим образом (считаем, что l задано):

1. $\mathcal{GENERATEEC}(l)$.
2. Вызвать алгоритм $\mathcal{EXTRACT}$ и сгенерировать d_T, QT.
3. Если в какой-то момент времени появляется пользователь с идентификатором ID, то сгенерировать личный ключ d_{ID}.
4. При необходимости подписи пользователем с идентификатором ID сообщения X, вызвать алгоритм $\mathcal{SIGN}(d_{ID}, X)$ и возвратить σ в качестве подписи.
5. При необходимости проверки подписи σ на сообщение X от пользователя, имеющего идентификатор ID, вызвать алгоритм $\mathcal{VERIFY}(\sigma, X, ID, Q_T)$ и вынести вердикт об истинности подписи.

5 Алгоритмы IBS схемы

5.1 Генерация и проверка параметров эллиптической кривой

5.1.1 Входные и выходные данные

Входными данными алгоритма генерации параметров эллиптической кривой являются уровень безопасности l, простой модуль p и целый коэффициент a. Должны выполняться следующие условия: $2^{l-1} < p < 2^l$, $p \equiv 3 \ (mod\ 4)$, $0 < a < p$. Выходными данными алгоритма генерации параметров являются параметр $seed \in \{0,1\}^\star$, коэффициент $b, 0 < b < p$, порядок $q, 2^{l-1} < q < 2^l$, q - простое, и генератор группы $G \in E_{a,b}(\mathbb{F}_p)$.

Входными данными алгоритма проверки параметров эллиптической кривой являются модуль p, коэффициенты a и b, параметр $seed$, порядок q и базовая точка G. Параметры p, a, b, q являются целыми числами, $seed \in \{0,1\}^\star$, точка G задается двумя целыми координатами. Выходными данными алгоритма проверки параметров является ответ ***ДА*** или ***НЕТ***. Ответ ***ДА*** означает, что переданные параметры описывают допустимую группу точек эллиптической кривой и были сгенерированы надлежащим образом. Ответ ***НЕТ*** означает противное. Будем считать, что после генерации параметры ЭК являются общедоступной информацией.

5.1.2 Вспомогательные алгоритмы

Вычисление порядка группы точек ЭК

На шаге 5 алгоритма генерации параметров определяется порядок группы точек эллиптической кривой. Для вычисления порядка может быть использован алгоритм Шуфа или его модернизации, например, алгоритм Шуфа - Элкиса - Аткина.

Проверка простоты

В перечислении 2) на шаге 6 алгоритма генерации параметров и в перечисление 3) на шаге 2 алгоритма проверки параметров контролируется простота чисел. Для проверки простоты рекомендуется использовать алгоритм Миллера - Рабина.

5.2 Генерация ключей ТДС (алгоритм $\mathcal{EXTRACT}$)

Входные и выходные данные

Алгоритм $\mathcal{EXTRACT}$ принимает на вход параметры эллиптической кривой p, a, b, q, G. Выходными данными являются личнй ключ ТДС $d_T \in \mathbb{F}_q$ и открытый ключ ТДС $Q_T = dTG$.

Вспомогательные алгоритмы

Используется арифметика на ЭК и генерация псевдослучайных чисел.

5.3 Алгоритм генерации личного ключа пользователя

Входные и выходные данные

Входными данными алгоритма генерации личного ключа являются ключи ТДС Q_T и d_T, ID - уникальный идентификатор пользователя. Выходными данными алгоритма генерации является личный ключ d_{ID}.

Вспомогательные алгоритмы

Используется арифметика на ЭК, генерация псевдослучайных чисел и функция хэширования φ, а также функции μ, π.

5.4 Алгоритм подписи сообщения $\mathcal{SIGN}$

Входные и выходные данные

Входными данными алгоритма подписи сообщения являются d_{ID} - личный ключ пользователя ID, сообщение X. Выходными данными алгоритма подписи является подпись σ.

Вспомогательные алгоритмы

Используются арифметика на ЭК, генерация псевдослучайных чисел, функции mu, π, функция хэширования φ.

5.5 Алгоритм проверки подписи

Входные и выходные данные

Входными данными алгоритма проверки подписи сообщения являются открытый ключ ТДС Q_T, сообщение X, подпись σ, идентификатор ID. Выходными данными алгоритма проверки подписи является ответ $\boldsymbol{ДА}$, который означает, что подпись верна, либо ответ $\boldsymbol{НЕТ}$ в противном случае.

Вспомогательные алгоритмы

Используется функция хэширования φ, функции μ, π, арифметика на ЭК.

6 Реализация IBS схемы

6.1 Генерация и проверка параметров эллиптической кривой

6.1.1 Алгоритм генерации параметров эллиптической кривой

Генерация параметров эллиптической кривой состоит в выполнении следующих шагов:

1. Выбрать произвольным образом $seed$.
2. $B \leftarrow hBelt(\langle p\rangle_l||\langle a\rangle_l||seed)||hBelt(\langle p\rangle_l||\langle a\rangle_l||seed \oplus \langle 1\rangle_{64})$.
3. $b \leftarrow B\ (mod\ p)$.
4. Если нарушается одно из условий:
   - $b \neq 0$.
   - $b^{p-1} \equiv 1\ (mod\ p)$.
   - $4a^3 + 27b^2 \neq 0\ (mod\ p)$.

   то вернуться к шагу 1.
5. $q \leftarrow |E_{a,b}(\mathbb{F}_p)|$.
6. Если нарушается одно из условий:
   - $2^{l-1} < q < 2^l$.
   - q - простое.
   - $p \neq q$.
   - $p^m \neq 1\ (mod\ q), m \in \overline{1,50}$.

   то вернуться к шагу 1.
7. $G \leftarrow (0, b^{\frac{p+1}{4}}\ mod\ p)$.
8. Возвратить $(seed, b, q, G)$.

6.1.2 Алгоритм проверки параметров эллиптической кривой

Проверка параметров эллиптической кривой состоит в выполнении следующих шагов:

1. Определить уровень стойкости l как минимальное натуральное, такое что $p < 2^l$.
2. Если нарушается одно из условий:
   - $2^{l-1} < p, q < 2^l$.
   - p, q - простые.
   - $p \equiv 3\ (mod\ 4)$.
   - $q \neq p$.
   - $p^m \neq 1\ (mod\ q), m \in \overline{1,50}$.

   то вернуть `НЕТ`.
3. $B \leftarrow hBelt(\langle p\rangle_l||\langle a\rangle_l||seed)||hBelt(\langle p\rangle_l||\langle a\rangle_l||seed \oplus \langle 1\rangle_{64})$.

4. Если нарушается одно из условий:
   - $0 < a, b < p$.
   - $b \equiv B\ (mod\ p)$.
   - $4a^3 + 27b^2 \neq 0\ (mod\ p)$.
   - $G = (0, b^{\frac{p+1}{4}}\ mod\ p)$.
   - $qG = O$.

   то вернуть `НЕТ`.
5. Вернуть `ДА`.

6.2 Алгоритм генерации и проверки личного ключа пользователю ID

6.2.1 Генерация личного ключа

Личный ключ генерируется по следующему принципу:

1. $k_{ID} \xleftarrow{R} \mathbb{F}_q$.
2. $R_{ID} \leftarrow k_{ID}G$.
3. $D \leftarrow ID \mid\mid \pi(R_{ID})$.
4. $s_{ID} \leftarrow k_{ID} + d_T \cdot \varphi(D)\ (mod\ q)$.
5. Вернуть $d_{ID} = (s_{ID}, R_{ID})$.

6.2.2 Проверка личного ключа

1. Если нарушается одно из условий:
   - $0 \leq x_{R_{ID}}, y_{R_{ID}} < p$.
   - $y^2_{R_{ID}} \equiv x^3_{R_{ID}} + ax_{R_{ID}} + b\ (mod\ p)$.
   - $0 \leq s_{ID} < q$.

   то вернуть `НЕТ`.
2. Вернуть `ДА`.

6.3 Алгоритм подписи сообщения

Подпись сообщения X пользователем ID происходит по следующему алгоритму:

1. $k \xleftarrow{R} \{0, 1, \cdots, q-1\}$.
2. $R \leftarrow kG$.
3. $D \leftarrow X \mid\mid \pi(R)$.
4. $s \leftarrow k + s_{ID}\varphi(D)\ (mod\ q)$.
5. $\sigma \leftarrow (s, \pi(R), \pi(R_{ID}))$.
6. вернуть σ.

6.4 Алгоритм проверки подписи

Проверка ЭЦП σ пользователя ID на сообщение X состоит в следующем:

1. Если $|\sigma| \neq 5l$, то вернуть `НЕТ`.
2. Представить σ в виде $\sigma_0||\sigma_1||\sigma_2$, где $\sigma_0 \in \{0,1\}^l, \sigma_1 \in \{0,1\}^{2l}, \sigma_2 \in \{0,1\}^{2l}$.
3. $s \leftarrow \sigma_0$.
4. $D \leftarrow ID \mid\mid \sigma_2$.
5. $c \leftarrow \varphi(D)$.
6. $D \leftarrow X \mid\mid \sigma_1$.
7. $e \leftarrow \varphi(D)$.
8. $R_{ID} \leftarrow \pi^{-1}(\sigma_2)$.
9. $R \leftarrow \pi^{-1}(\sigma_1)$.
10. Если выполняется $sG = R + eR_{ID} + ecQ_T$, то вернуть `ДА`. В противном случае вернуть `НЕТ`.

7 Оценка быстродействия и работы нашей схемы в сравнение с реализацией схемы Шнорра из [8]

7.1 Общие положения, трюк Шамира

Сравним 2 схемы по каждому алгоритму в отдельности, на основе чего можно будет сделать некоторые общие выводы.

Рассмотрим лишь алгоритмы непосредственно схемы, а именно $\mathcal{EXTRACT}$, $\mathcal{VERIFY}$ и $\mathcal{SIGN}$, так как этапы генерации параметров ЭК, хэш-функции и другие примитивы взяты непосредственно из [8]. Введем обозначения времени, необходимого для выполнения некоторых примитивов:

1. τ_K - время, необходимое для вычисления кратной точки ЭК.
2. τ_H - время, необходимое для хэширования сообщения.

Рассмотрим следующее выражение: $\alpha A + \beta B$, где $\alpha,\ \beta \in \{1, \ldots, q-1\}$, а $A, B \in E_{a,b}(\mathbb{F}_p)$, $|E_{a,b}(\mathbb{F}_p)| = q$. При тривиальной реализации вычисление выражения займет $2\tau_K$, однако при использовании алгоритма из [6], называемого также трюком Шамира, вычисление занимает уже $1.5\tau_K$ и дает еще больший выигрыш при большем количестве точек.

Дадим краткое описание алгоритму в случае суммы 2 кратных точек в предположении, что α, β имеют одинаковое число двоичных разрядов (если это не так, то просто дополним одно из чисел лидирующими нулями). Обозначим α_m – m-тый по старшинству бит числа α, β_m – m-тый по старшинству бит числа β. Алгоритм состоит из следующих шагов:

1. $R \leftarrow O$.
2. $m \leftarrow Highest$, где $Highest$ - максимальный номер единичного бита, выбираемый из 2 чисел(нумерация битов ведется с нуля).
3. $C \leftarrow A + B$.
4. `ПОКА` $m \geq 0$, `ВЫПОЛНЯТЬ` следующие шаги:
   (a) $R \leftarrow R + R$.
   (b) `ЕСЛИ` $\alpha_m = 1$ `И` $\beta_m = 1$, `ТО` $R \leftarrow R + C$.
   (c) `ЕСЛИ` $\alpha_m = 1$ `И` $\beta_m = 0$, `ТО` $R \leftarrow R + A$.
   (d) `ЕСЛИ` $\alpha_m = 0$ `И` $\beta_m = 1$, `ТО` $R \leftarrow R + B$.
   (e) $m \leftarrow m - 1$.
5. Возвратить R в качестве результата.

Рассмотрим другие примитивы вычислительного процесса. Отметим, что примитивы μ, π являются по сути лишь переводом чисел в двоичную систему исчисления и их последующую контакенацию, в связи с этим не будем учитывать время работы этих алгоритмов, так как оно достаточно сильно варьируется от конкретной реализации (собственно, как и время работы с длинными целыми числами, а следовательно и сумма 2 точек ЭК). Также не будем учитывать время обращения к датчику псевдослучайных чисел.

7.2 Некоторые сравнительные характеристики

	Наша схема	[8]
Генерация ключей	$\tau_K + \tau_H$	τ_K
Алгоритм $\mathcal{SIGN}$	$\tau_K + \tau_H$	$\tau_K + 2\tau_H$
Алгоритм $\mathcal{VERIFY}$	$2\tau_H + 1.5\tau_K$	$2\tau_H + 1.5\tau_K$

Таблица 2: Сравнительная характеристика времени выполнения алгоритмов

	Наша схема	[8]
Длина открытого ключа	–	$4l$
Длина личного ключа	$3l$	$2l$
Длина подписи	$5l$	$3l$

Таблица 3: Сравнительная характеристика длин ключей и подписи

Таким образом можно заметить, что при одинаковом времени проверки подписи одного пользователя (а при многократных подписях пользователя время работы нашей схемы будем меньше) нашу схему отличает больший размер подписи. Однако, увеличение подписи в $\delta_0 = 1.(6)$ раз (или на $\Delta_0 = 2l$ бит) не сможет полностью перекрыть такие очевидные преимущества схемы как:

- Быстрое время работы с пользователем при его многократных обращениях.
- Отсутствие сертификатов.
- Меньшая суммарная длина ключей – уменьшение достигло $\Delta_1 = 2l$ бит (в $\delta_1 = 2$ раза).

Стоит также отметить, что работу схемы можно ускорить программными методами, такими как:

1. Хранение длинных чисел по базе 2^k, где k - размер машинного слова.
2. Умножение длинных чисел (как по модулю, так и в общем виде) с помощью дискретного преобразования Фурье либо Хартли.
3. Оптимизированная реализация трюка Шамира.

8 Заключение

В настоящей работе была рассмотрена общая концепция ЭЦП, рассмотрена классическая схема Шнорра, которые явились основой для нового направления - IBS. Рассмотрена также проблема выбора группы, на которой будем строить арифметику ЭЦП, обоснован выбор группы точек ЭК. На основе 2 схем IBS была предложена общая концепция новой схемы IBS. Был проведен анализ схемы и введены некоторые модификации. Результатом явилось создание нашей схемы ЭЦП, а также ее инстанциирование под существующие белорусские стандарты и примитивы, используемые в [8]. Были рассмотрены вопросы быстроты базовых алгоритмов и размеров входных/выходных данных в сравнение с [8]. В приложении к настоящей работе приведен исходный код реализации предложенной схемы на языке C++ с использованием динамической библиотеки `BIGN.dll`.

Список литературы

[1] David Galindo and Flavio D. Garcia. A Schnorr-Like Lightweight Identity-Based Signature Scheme

[2] He Debiao, Chen Jianhua, Hu Jin. Identity-based Digital Signature Scheme Without Bilinear Pairings. School of Mathematics and Statistics, Wuhan University

[3] C.P. Schnorr, Frankfurt University, Efficient identification and signatures for smart cards.

[4] Adi Shamir, Identity-Based Cryptosystems and Signature Schemes. Advances in Cryptology: Proceedings of CRYPTO 84, Lecture Notes in Computer Science, 7:47-53, 1984

[5] Bellare M., Neven G. Multi-signatures in the plain public-key model and a general forking lemma. In: ACM CCS 06, ACM Press, 2006, 390–399.

[6] Strauss: Addition chains of vectors. American Mathematical Monthly 71(7), 806–808, 1964.

[7] Neal Koblitz. Introduction to Elliptic Curves and Modular Forms.

[8] Проект СТБ "Информационные технологии и безопасность. Алгоритмы выработки и проверки электронной цифровой подписи на основе элиптических кривых".

A Реализация нашей схемы на языке C++. Исходный код

```
//jte

#include <cstdio>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include "long.h"
#include "bign.h"
#include "belt.h"
#include "point.h"
#include "typedef.h"
#include <time.h>


LongInt zero;
LongInt one;
const uint32 l = 1 << 7;

void Make_zero()
{
    memset(zero.arr, 0, sizeof zero.arr);
    zero.len = 8;
}

    LongInt d_T;
    PointAfEC Q_T;

typedef struct
{
    LongInt s_ID;
    PointAfEC R_ID;
}   Ibs_private_key, *pIbs_private_key;

typedef struct
{
    LongInt s;
    PointAfEC R, R_ID;
} ibs_sign_data;

ParamEC params;

BOOL Init(pParamEC params, pPointAfEC Q_T)
{
    bign_ec *pre_params = new bign_ec;
    pre_params->l = l;
    bign_prng_state random;
    srand(time(NULL));
    for (int i = 0; i < 32; ++i)
    {
        random.theta[i] = (uint8)255 & rand();
        random.s[i] = (uint8)255 & rand();
    }

    BOOL result = bign_prng_init(&random);
    uint8 buffer[200];
    result &= bign_prng_proc(pre_params->p, 32, &random);

    C_ReadLongInt(32, pre_params->p, &params->p);
    while (C_TestMR(&params->p) == FALSE)
```

```
    {
        for (int i = 0; i < 32; ++i)
    {
        random.theta[i] = (uint8)255 & rand();
        random.s[i] = (uint8)255 & rand();
    }
        result &= bign_prng_proc(pre_params->p, 32, &random);
        C_ReadLongInt(32, pre_params->p, &params->p);
    }
    result &= bign_prng_proc(pre_params->a, 32, &random);
    C_ReadLongInt(32, pre_params->a, &params->a);
    C_DivMod(NULL, &params->a, &params->a, &params->p);
    memcpy(pre_params->a, &params->a, 32);
    result = bign_stdec(pre_params);
    C_ReadLongInt(32, pre_params->b, &params->b);
    C_ReadLongInt(32, pre_params->q, &params->q);
    C_ReadLongInt(32, pre_params->yG, &params->G.y);
    params->G.type = 1;
    Make_zero();
    memcpy(&params->G.x, &zero, sizeof (LongInt));
    result &= bign_valec(pre_params);
    result &= bign_prng_proc(buffer, 32, &random);
    C_ReadLongInt(32, buffer, &d_T);
    result &= C_DivMod(NULL, &d_T, &d_T, &params->p) == AE_SUCCESS;
    memcpy(Q_T, &params->G, sizeof PointAfEC);
    EC_MultPointAf(Q_T, &d_T, params);
    return result;
}



BOOL generate_private_key(pParamEC params, pIbs_private_key key,
                          LongInt * ID)
{
    bign_prng_state random;
    srand(time(NULL));
    for (int i = 0; i < 32; ++i)
    {
        random.theta[i] = (uint8)255 & rand();
        random.s[i] = (uint8)255 & rand();
    }
    BOOL result = bign_prng_init(&random);
    uint8 buffer[200];
    result &= bign_prng_proc(buffer, 32, &random);
    LongInt k_ID;
    C_ReadLongInt(32, buffer, &k_ID);
    result &= C_DivMod(NULL, &k_ID, &k_ID, &params->q) == AE_SUCCESS ;
    memcpy(&key->R_ID, &params->G, sizeof PointAfEC);
    EC_MultPointAf(&key->R_ID, &k_ID, params);
    int hash[256];
    int length = ID->len + key->R_ID.x.len + key->R_ID.y.len;
    for (int i = 0;i < ID->len; ++i)
        hash[i] = ID->arr[i];
    for (int i = 0; i  < key->R_ID.x.len; ++i)
        hash[ID->len + i] = key->R_ID.x.arr[i];
    for (int i = 0; i < key->R_ID.y.len; ++i)
        hash[ID->len + key->R_ID.x.len + i] = key->R_ID.y.arr[i];
    int hashed[256];
    Belt_HashCalculate(hash, length, hashed);
    LongInt phi;
    C_ReadLongInt(32, hashed, &phi);
    C_MultMod(&phi, &d_T, &phi, &params->q);
```

```
    C_AddMod(&key->s_ID, &k_ID, &phi, &params->q);
    return result;
}

BOOL check_private_key(pParamEC params, pIbs_private_key key)
{
    BOOL result = true;
    result &= C_Cmp(&key->s_ID, &zero)>=0 && C_Cmp(&key->s_ID, &params->q) < 0;
    LongInt checker;
    C_MultMod(&checker, &key->R_ID.x,  &key->R_ID.x, &params->p);
    C_MultMod(&checker, &checker,  &key->R_ID.x, &params->p);
    LongInt checker2;
    result &= C_MultMod(&checker2, &key->R_ID.x, &params->a, &params->p) == AE_SUCCESS;
    C_AddMod(&checker, &checker, &checker, &params->p);
    C_AddMod(&checker, &checker, &params->b, &params->p);
    C_MultMod(&checker2, &key->R_ID.y,  &key->R_ID.y, &params->p);
    result &= C_Cmp(&checker2, &checker) == 0;
    return result;

}

void ibs_sign(pParamEC params, pIbs_private_key d_ID,
              uint8* X, int length, ibs_sign_data* sigma)
{
    bign_prng_state random;
    srand(time(NULL));
        uint8 buffer[200];
    for (int i = 0; i < 32; ++i)
    {
        random.theta[i] = (uint8)255 & rand();
        random.s[i] = (uint8)255 & rand();
    }

    BOOL result = bign_prng_init(&random);
    result &= bign_prng_proc(buffer, 32, &random);
    LongInt k;
    C_ReadLongInt(32, buffer, &k);
    result &= C_DivMod(NULL, &k, &k, &params->q) == AE_SUCCESS;
    memcpy(&sigma->R_ID, &d_ID->R_ID, sizeof PointAfEC);
    memcpy(&sigma->R,  &params->G, sizeof PointAfEC);
    EC_MultPointAf(&sigma->R, &k, params);
    int hash[256];
    for (int i = 0; i < length / 4; i += 4)
        hash[i] = (X[i] << 24) | (X[i + 1] << 16) | (X[i + 2] << 8)
        | X[i + 3];
    int cur_l = length / 4;
    if (length % 4)
    {
        int cur = length % 4;
        int i = length - length % 4;
        hash[length / 4] = (cur == 3)? (X[i] << 24) | (X[i + 1] << 16) |
            (X[i + 2] << 8) :
            cur == 2? (X[i] << 24) | (X[i + 1] << 16) : (X[i] << 24);
        cur_l += 1;
    }
    for (int i = 0; i < sigma->R.x.len; ++i)
        hash[cur_l + i] = sigma->R.x.arr[i];
    cur_l += sigma->R.x.len;
    for (int i = 0; i < sigma->R.y.len; ++i)
        hash[cur_l + i] = sigma->R.y.arr[i];
    cur_l += sigma->R.y.len;
    int hashed[256];
```

```
    Belt_HashCalculate(hash, length, hashed);
    LongInt phi;
    C_ReadLongInt(32, hashed, &phi);
    C_MultMod(&phi, &d_ID->s_ID, &phi, &params->q);
    C_AddMod(&sigma->s, &k, &phi, &params->q);
    memcpy(&sigma->R_ID, &d_ID->R_ID, sizeof PointAfEC);
}

BOOL ibs_verify(pParamEC params, uint8* X, int length,
                ibs_sign_data* sigma, LongInt* ID, pPointAfEC)
{
    int hash[256];
    BOOL result = true;
    int len = ID->len + sigma->R_ID.x.len + sigma->R_ID.y.len;
    for (int i = 0;i < ID->len; ++i)
        hash[i] = ID->arr[i];
    for (int i = 0; i  < sigma->R_ID.x.len; ++i)
        hash[ID->len + i] = sigma->R_ID.x.arr[i];
    for (int i = 0; i < sigma->R_ID.y.len; ++i)
        hash[ID->len + sigma->R_ID.x.len + i] = sigma->R_ID.y.arr[i];
    int hashed[256];
    Belt_HashCalculate(hash, len, hashed);
    LongInt c;
    C_ReadLongInt(32, hashed, &c);
    for (int i = 0; i < length / 4; i += 4)
        hash[i] = (X[i] << 24) | (X[i + 1] << 16) | (X[i + 2] << 8)
        |   X[i + 3];
    int cur_l = length / 4;
    if (length % 4)
    {
        int cur = length % 4;
        int i = length - length % 4;
        hash[length / 4] = (cur == 3)? (X[i] << 24) | (X[i + 1] << 16) |
            (X[i + 2] << 8) :
            cur == 2? (X[i] << 24) | (X[i + 1] << 16) : (X[i] << 24);
        cur_l += 1;
    }
    for (int i = 0; i < sigma->R.x.len; ++i)
        hash[cur_l + i] = sigma->R.x.arr[i];
    Belt_HashCalculate(hash, length, hashed);
    LongInt e;
    C_ReadLongInt(32, hashed, &e);
    C_MultMod(&c, &c, &e, &params->q);
    PointAfEC right;
    EC_AddMultPointAf(&right, &sigma->R_ID, &e, &Q_T, &e, params);
    EC_AddPointAf(&right, &right, &sigma->R, params);
    PointAfEC left;
    memcpy(&left, &params->G, sizeof PointAfEC );
    EC_MultPointAf(&left, &sigma->s, params);
    result &= (C_Cmp(&left.x, &right.x) == 0) && (C_Cmp(&left.y, &right.y) == 0);
    return result;
}
```

IBS.h