МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«МИРЭА - Российский технологический университет»
РТУ МИРЭА

Институт кибернетики
Кафедра информационной безопасности

КУРСОВАЯ РАБОТА
по дисциплине
«Методы программирования»

Тема курсовой работы
«ЭЦП по схеме Эль-Гамаля с хэш-функцией SHA-3»

Студент группы ККСО-01-19: Алабин А.Ю.____________

Руководитель курсовой работы: Кирюхин В.А. ____________

Работа представлена к защите «__» __________ 2021 г.

Допущен к защите «__» __________ 2021 г.

Москва 2021

Содержание

Введение

В данной работе работе рассматривается алгоритм ЭЦП по схеме Эль-Гамаля и ряд вспомогательных математических задач, связанных с генерацией ключей, созданием электронной подписи документа и обработкой докумета перед подписью.

1. Теоретическая часть

1.1. Описание схемы Эль-Гамаля [1]

.

Схема цифровой подписи Эль-Гамаля основана на сложности задачи вычисления значения логарифмов(т.е. обращение функции g^x) в конечном поле. В схеме Эль-Гамаля, для начала, нужно **вычислить ключи:**

Пусть p - простое число и g - примитивный элемент поля Z_p, то есть $g : g^{\varphi(p)} \equiv 1 \pmod p \wedge g^l \not\equiv 1 \pmod p \wedge g \in \overline{2, \varphi(\mathrm{p}) - 1}$, где $l \neq \varphi(p)$. Выберем случайное число k $\in \overline{2, \mathrm{p} - 2}$ и вычислим значение y = $g^k \pmod p$, тогда:

Определение 1 *priv_k = k - секретный ключ*

Определение 2 *pub_k = (p,g,y) - публичный ключ*

Как следует из названий: k - держится в секрете, а (p,g,y) - доступен всем желающим. Можно вычислить(с большой вероятностью верное) g, используя делители числа $p - 1 = \varphi(p)$: зная, что $p_i \mid \varphi(p)$, можно сразу исключить числа $g : g^{\varphi(p)/p_i} \equiv 1 \pmod p$ - т.к. это противоречит определению g. Число g прошедшее проверку со всеми p_i - возьмём, как возможно-первообразный корень.

Чтобы сделать **подпись для сообщения M**(которое, как правило, хэшируется перед подписью), вычисляем:

1) случайное целое число(сессионный ключ) $k' \in \overline{1, \mathrm{p} - 2}$.
2) r = $g^{k'} \pmod p$.
3) s = $(M - kr)(k')^{-1} \pmod{p - 1}$ $(*)$

Определение 3 *Подписью сообщения M - считается пара (r, s).*

Чтобы получить $(k')^{-1}$ - используем делители числа $p - 1 = \varphi(p)$, т.к. p - простое. Пусть $m = p-1$, то $\varphi(m) = m(1-1/t_1)...(1-1/t_k)$ где $\overline{\mathrm{t}_1, \mathrm{t}_\mathrm{k}}$ - простые делители $m = p - 1 \implies (k')^{-1} = (k')^{Phi(m)-1} \pmod m$

● Формально подпись $sign()$ сообщения M можно описать так:

$$sign(M, pub\_k) = (r, s)$$

Проверка подписи заключается в проверке равенства: $y^r r^s \equiv g^M \pmod p$
Убедимся в корректности проверки:
для этого заметим, что из $(*) \implies sk' = M - xr \pmod p - 1 \implies M = xr + sk' \pmod p - 1$ Тогда $g^M \iff g^{xr} g^{k's} \iff (g^x)^r (g^{k'})^s \iff y^r r^s \square$

● Формально проверку $chk()$ подписи (r, s) сообщения M можно описать так:

$$chk(M, sign(M, pub\_k), pub\_k) = is\_correct$$

, где $is\_correct \in \{0, 1\}$, 1 - означает корректность подписи, 0 - противное.

Ниже приведено графическое представление подписи по схеме Эль-Гамаля:

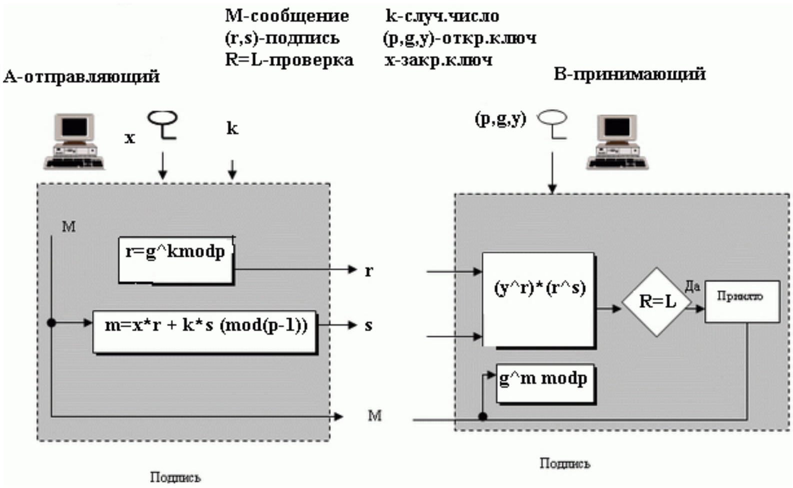

Рисунок 1 – Схема ЭЦП Эль-Гамаля

1.2. Генерация больших простых чисел

Пусть перед нами стоит задача определить, является ли число p - простым, обычный перебор делителей до $\sqrt{p}$ при больших p зациклится ”на вечность”и даже различные решета не помогают справиться с этой проблемой, поэтому появляется необходимость в более быстром алгоритме определения простоты - для этого подходит вероятностный алгоритм Миллера-Рабина - он быстрый, но не гарантирует простоту числа(лишь высокую вероятность простоты).

1.2.1. Алгоритм Миллера-Рабина [2]

Алгоритм Миллера-Рабина - вероятностный тест, который не может выдать простое число за составное, однако обратное возможно(с тем меньшей вероятностью, чем больше проведено тестов).

Пусть нужно определить(с большой вероятностью), что число p - является простым, представим $p - 1 = 2^s t$, где s - степень двойки в канон. разложении числа $p - 1 \implies t$ - нечётная часть числа $p - 1$, тогда алгоритм выглядит следующим образом:

1) выбираем случайное $a \in \overline{2, \mathrm{p} - 1}$: $(a, p) = 1$
2) проверяем, что выполнено хотя бы одно условие:
   $a^t \equiv 1 \pmod p \vee \exists k \in \overline{0, \mathrm{s} - 1} : a^{2^k t} \equiv -1 \pmod p$
3) если прошлый шаг пройден, то a - "свидетель простоты" $\implies$ переходим к п.1 алгоритма - повторяем пока не найдём $log_2(p)$ свидетелей "простоты
   иначе - число p точно составное

На выходе алгоритма имеем информацию о том, каким является p: вероятно простым либо точно составным.

1.2.2. Генерация простого числа p(с оглядкой на на разложение p-1)

Чтобы получить большое простое(вероятно простое) число p - достаточно воспользоваться алгоритмом Миллера-Рабина с $log_2(p)$ тестами, однако в схеме Эль-Гамаля для быстрой генерации g и $(k')^{-1}$,как мы уже увидели, пригодится знать разложение $p - 1$, учитывая это, и полагая, что нам нужно простое число порядка 2^{1024}, поступим следующим образом - построим составное число $p - 1$ с заданным нами разложением, чтобы p - оказалось простым:

1) сгенерируем 2-а вероятно простых числа p_1, p_2 в диапазоне $\overline{2^{512}, 2^{513}}$ при помощи алгоритма Миллера-Рабина, тогда разложение $p - 1 = 2 \cdot p_1 p_2 p_3$, 2 присутствует в разложении, т.к. $2|\varphi(p)$ при простом p, а p_3 - подбирём в п.2
2) далее будем искать как можно меньшее простое $p_3 : p = 2 \cdot p_1 p_2 p_3 + 1$ - тоже простое

На выходе имеем простое $p = 2 \cdot p_1 p_2 p_3 + 1$ и разложение $p - 1 = 2 \cdot p_1 p_2 p_3$.

1.3. Криптографическая хэш-функция SHA3 [3]

Как правило, подписываемый документ весит > 1024 бит, поэтому возникает необходимость использовать хэш функцию: она сопоставляет сообщению фиксированную по числу бит(в SHA3-256 - 256 бит) строку, которую мы уже и подписываем. Полученная строка является, как бы ”слепком”сообщения, т.к. тяжело(такое требование предъявляют к хэш-функциям) найти другое сообщение, дающее такой же хэш.

Определение 4 *Криптографическая хэш-функция - хеш-функция, способная противостоять всем известным типам криптоаналитических атак.*

Типичный пример атаки на хэш-функцию - поиск коллизии, т.е. поиск m_1, m_2 : $h(m_1) = h(m_2)$ После нахождения коллизии злоумышленник может подменить сообщение m_1 на m_2.

SHA3(Keccak) - является криптографической хэш-функцией, выдающей хэш произвольной разрядности.

SHA-3 основан на функции криптографической губки - рассмотрим её ниже.

Определение 5 *Губкой(состоянием губки) S - называют строку поделённую на 2-е части: S_1 размера r(rate) - скорость и S_2 размера c(capacity) - ёмкость.*

Условимся, что **длина хэша должна быть d(digest) бит**.
- Входное сообщение M дополняется последовательностью бит(по некоторому правилу) до размера кратного $|S_1| = r$ и делится на $n = |M|/r$ блоков длины n (блоки можно сравнить с капельками воды, а M - со всем обёмом, которую должна впитать часть губки(S_1))

1) на стадии инициализации губка S - ”сухая состоит из одних нулей.

2) сначала производится многократное впитывание:
Часть губки S_1 вбирает в себя очередной поксоренный об себя блок P_i, затем вся губка S пропускается через функцию f: S $\rightarrow S'$ говорят, что губка S перешла в новое состояние S'
-продолжаем до тех пор, пока не впитаем все n блоков дополненного M.

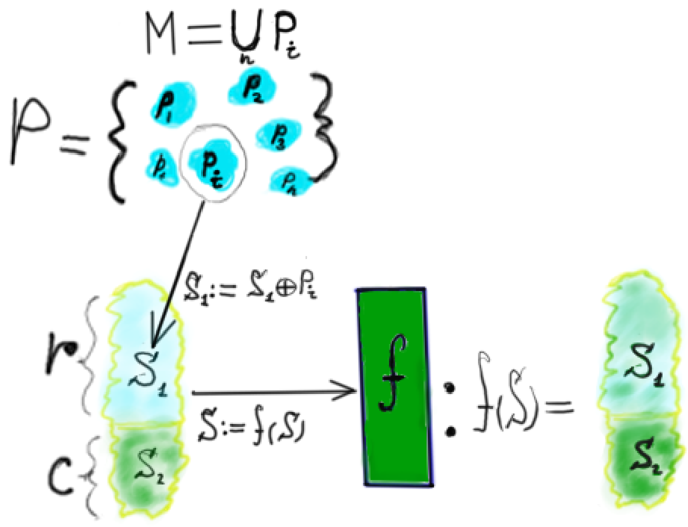

Рисунок 2 – i-ый шаг впитывания

где $f : \{0, 1\}^n \to \{0, 1\}^*$ - многораундовая бесключевая псевдослучайная перестановка

3) производим отжим:
Если после n впитываний губка $S : |S_1| = r \geq d \implies$ возвращаем первые d бит из S_1
иначе(бит не хватает) мы запоминаем $Z = S_1$ и пропускаем S через f, получая $S' = S(f)$ и конкатенируем Z и S'_1, если $|Z| \leq d$ - продолжаем конкатенировать Z с S''_1, S'''_1, ... до тех пор, пока не получим $d \leq |Z|$ - возвращаем первые d бит из Z.

На выходе имеем d-битную хэш-последовательность.

”Keccak”имеет множество настраиваемых параметров(число раундов, размер S и т.д.), в стандарте SHA-3 ”Keccak”принимает следующую конфигурацию:

- губка S - это массив из 5x5 слов длины 64 бита, т.е. общая длина S равна 1600 бит, в которых r = 1088, c = 512.
- функция губки имеет особое правило дополнения pad10*1:
  - в наиболее общем случае к сообщению M, не кратному по длине к r, конкатенируется 1, затем последовательность 0...01 так, чтоб в результате дополненное M - было кратно r
  - когда длина M уже кратна r - происходит дополнение блоком 10...01 - длины r. это делается потому, что без дополнения данное M совпало бы с дополненным сообщенем M' и их хэш был бы одинаков

- когда длина $M = r - 1$ к M добавляется 1, завершающая первый блок, а следующий блок состоит из последовательности 00...01 - длины r(размер блока), выходит мы опять дополнили сообщение 10...01 последовательностью

● функция f имеет пять шагов по 24 раунда. На каждом из 5-ти шагов($\theta, \rho, \pi, \xi, \iota$) губка S воспринимается, как 3-ёхмерный массив размера 5x5x64 и над ним производятся определённые преобразования(многие из которых удобно воспринимать, как некоторые действия в 3-ёх мерном пространстве).

2. Практическая часть

В этой части приведена общая логика программы и инструкции по работе с программой, реализующей учебную модель/симуляцию PKI с ЭЦП по схеме Эль-Гамаля.

2.1. Общая логика программы

Рассмотрим логику программы, реализующую ЭЦП:

- существует доверенное лицо Trent, которое выдаёт всем пользователям (при инициализации (”gen”) или запросе (”crt”)) квази-сертификат - подпись пуб.ключа автора; приставка ”квази”добавленна, т.к. понятие сертификата подразумевает хранение внутри себя кроме подписи M - само M (в нашем случае M - пуб.ключ).
- Можно (пере)создать пользователя с любым разрешённым именем(все, кроме ”$Trent$” и ”$\_\_pychache\_\_$”).
- Инициализированный пользователь имеет собственную папку $./User/$ в которой хранятся его приватный ключ, пуб.ключ и квазисертификат
- каждый пользователь может подписывать файлы, находящиеся в той же директории, что и программа El_gamal.py, после чего у него (в его папке $./User/$) появляется подпись file_name.sig
- проверка подписанного документа включает в себя два этапа:
  - удостоверение личности автора(сверка квазисертификата и пуб. ключа автора)
  - если личность автора подтверждена, то происходит проверка на отсутствие модификации документа после подписи(проверяется подпись самого документа)

- существует возможность отозвать все существующие (старые) сертификаты, чтобы существующие/новые авторы заново подтвердили свою личность, иначе подписанные ими файлы не пройдут проверку(т.к. нельзя установить личность автора)

Стоит отметить, что описанная выше логика вписывается в известностную концепцию PKI - инфраструктура открытых ключей. Её основные принципы:

- закрытый ключ (private key) известен только его владельцу
- удостоверяющий центр (CA - Certificate Authority) создает электронный документ - сертификат открытого ключа, таким образом удостоверяя факт того, что закрытый (секретный) ключ известен эксклюзивно владельцу этого сертификата
- открытый ключ (public key) свободно передается
- никто не доверяет друг другу, но все доверяют удостоверяющему центру
- удостоверяющий центр подтверждает или опровергает принадлежность открытого ключа заданному лицу, которое владеет соответствующим закрытым ключом

2.2. Режимы работы программы

Пусть далее Bob - выбранный при запуске программы пользователь.

- **gen** - генерация ключей и квази-сертификата для Bob;
  после генерации выводится ссылка с проверкой полученного p на простоту в виде: $http://factordb.com/index.php?query=p\_number$,
  где p_number - p из пуб.ключа
- **sgn** - подпись документа от лица Bob
- **chk** - проверка на то, что Bob является автором документа и на то, что документ не был изменён после подписи
- **new** - просьба к Trent: перегенерировать его ключи $\implies$ все ранее выданные сертификаты станут недействительны и тем, авторам, которые захотят продолжить взаимодействие придётся снова получать квази- сертификат у Trent
- **cert** - выдача нового квази-сертификата для Bob(пригодится, если был использован режим ”new”)

Заключение

В результате проделанной работы получилось познакомиться с процессом ЭЦП, концепцией PKI, а также реализовать их на ЯП python3, кроме того были рассмотренны криптографическая хэш-функция, важная задача определения простоты больших чисел и их построение под конкретную задачу.

Список литературы

[1] Алфёров А.П., Зубов А.Ю., Кузьмин А.С., Черемушкин А.В. ”Основы криптографии”начиная со стр.372

[2] Ю. В. Нестеренко ”Введение в криптографию”Гл.4

[3] https://habr.com/ru/post/534596/ - SHA3

Приложения

Реализация ЭЦП находится в файле El_Gamal.py, lib_prrime.py - вспомогательная библиотека для генерации больших простых чисел(написана самостоятельно).