პოსტ-კვანტური ციფრული ხელმოწერის ასინქრონული ალგორითმი

არტურო არაქელიანი

სადოქტორო ნაშრომი შესრულებულია

ინფორმატიკის აკადემიური დოქტორის ხარისხის მოსაპოვებლად

საქართველოს უნივერსიტეტის

მეცნიერებისა და ტექნოლოგების სკოლის

ინფორმატიკის დეპარტამენტი

სადისერტაციო საკონსულტაციო საბჭო

ხელმძღვანელი: ასოცირებული პროფესორი ბექარ მელაძე

თანახელმძღვანელი: მოწვეული პროფესორი მაქსიმ იავიჩი

თბილისი

2022

აბსტრაქტი

კვანტური კომპიუტერების შექმნის პერსპექტივამ წარმოაჩინა მრავალი პრობლემა, რომელიც შეიძლება აქტუალური გახდეს კვანტური კომპიუტერების წარმოებაში ჩაშვების შემდგომ. კვანტური კომპიუტერების შექმნით, მნიშვნელოვნად გაიზრდება გამოთვლით ტექნოლოგიებს შესაძლებლობები. გამოთვლითი რესურსების ზრდასთან ერთად აქტუალური ხდება ციფრული სისტემების უსაფრთოხების საკითხების კვლევა. დღესდღეისობით არსებული სისტემები, რომელებიც დაფუძნებულია მარტივი რიცხვების ფაქტორიზაციზე ხდება დაუცველი კიბერშეტევების მიმართ. ასეთი სისტემების თვალსაჩინო მაგალითია RSA ციფრული ხელმოწერა. კვანტური კომპიუტერების გამოჩენასთან ერთად უსაფრთხო სისტემების შექმნა და არსებული სისტემების დახვეწა აერთ ერთ მთავარ საკითხად წარმოჩინდება.

RSA ციფრული ხელმოწერის ალტერნატივა არის ჰეშზე დაფუძნებული ციფრული ხელმოწერები. აღნიშნული სისტემები იყენებენ კრიპტოგრაფიულ ჰეშ ფუნქციებს და ამ ხელმოწერის სქემების უსაფრთხოება დამოკიდებულია ჰეშ ფუნქციების კოლიზიის მიმართ მდგრადობაზე.

სადისერტაციო ნაშრომში წარმოდგენილი კვლევის მიზანია ისეთი კრიპტოგრაფიული ალგორითმის წარმოდგენა, რომელიც მდგრადი და ეფექტური იქნება კვანტური კომპიუტერების მხრიდან მომდინარე პოტენციური შეტევების მიმართ.

ნაშრომში განხილულია დღეს არსებული, ჰეშზე დაფუძნებული ხელმოწერის სისტემები, კერძოდ Merkle-ს ალგორითმი. აღნიშნული ალგორითმი შეიძლება მიჩნეული იყოს, როგორც სტატიკური, რადგან მის მიერ წარმოებული გამოთვლების სიჩქარე არ არის დამოკიდებული პროცესორის ნაკადებზე და მათ რაოდენობაზე. ნაშრომში შემოთავაზებულია პოსტ-კვანტური ციფრული ხელმოწერის ასინქრონული ალგორითმი, რომელიც იყენებს პროცესორის შესაძლებლობებს, კერძოდ ნაკადებს. ნაშრომში მოცემული ალგორითმის ფსევდო კოდი.

შემოთავაზებული ალგორითმის ანალიზის საფუძველზე ნაჩვენებია, რამდენად სწრაფია მიღებული ალგორითმი კლასიკურ ალგორითმთან შედარებით. კვლევის შედეგად ნაჩვენებია, რომ ახალი ასინქრონული ალგორითმის ოპერაციების რაოდენობა ბევრად ნაკლებია, რაც განპირობებულია იმ ფაქტით, რომ ალგორითმში გათვალისწინებულია პროცესორის ნაკადების რაოდენობა.

სარჩევი

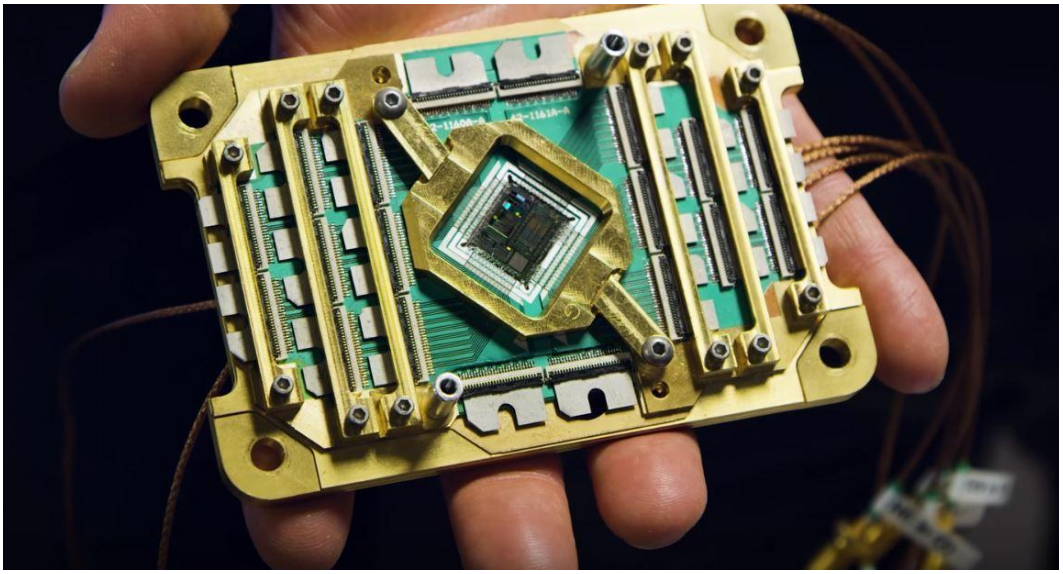
ილუსტრაციებისა და ნახაზების ჩამონათვალი

თავი 1. კრიპტოგრაფიის თანამედროვე გამოწვევები და პრობლემები

მიმოხილვა

2016 წელს გამოქვეყნდა სტატია იმის შესახებ, რომ კორპორაცია Google-მა, NASA-მ და კოსმოსური კვლევების უნივერსიტეტების ასოციაციამ (Universities Space Research Association- USRA) მოაწერეს ხელი თანამშრომლობაზე კვანტური D-Wave პროცესორების მწარმოებელთან [1].

D-Wave 2X უახლესი კვანტური პროცესორია, რომელიც შეიცავს 2048 ფიზიკურ კუბიტს (კვანტური განმუხტვები, ინფორმაციის შენახვის უმცირესი ერთეულები კვანტურ კომპიუტერში). 1152 კუბიტი კვანტური კომპიუტერის ამ მოდელში გამოიყენება გამოთვლების შესასრულებლად. თითოეული დამატებითი კუბიტი ორჯერ ზრდის ძიების სივრცეს, შესაბამისად იზრდება გამოთვლების სიჩქარეც.



ილუსტრაცია 1

D-Wave 2X - უახლესი კვანტური პროცესორი (https://www.alphr.com/technology/1003173/google-and-nasa-are-exploring-quantum-computing-with-the-d-wave-2x)

ციფრული კომუნიკაციისა და მონაცემთა ელექტრონული გაცვლის სწრაფი ზრდის გამო ინფორმაციის უსაფრთხოება მნიშვნელოვანი საკითხი გახდა სახელმწიფო ორგანიზაციებსა და ბიზნესის სფეროში. კრიპტოგრაფია შეისწავლის

კონფიდენციალურობას (უცხო ადამიანის მიერ გარკვეული ინფორმაციის ვერ წაკითხვა), მონაცემების ერთიანობას (უკვალოდ ინფორმაციის შეცვლის შეუძლებლობა), აუტენტიფიკაციას (ნამდვილობის შემოწმება) და საიმედოობას. თავდაპირველად კრიპტოგრაფია შეისწავლიდა დაშიფრვის მეთოდებს, ანუ ღია ტექსტის გარდაქმნას დაშიფრულ ტექსტში საიდუმლო ალგორითმის ან გასაღების საშუალებით. ტრადიციულ კრიპტოგრაფიაში შედის სიმეტრიული კრიპტოსისტემები, რომლებშიც ღია ტექსტის დაშიფრვა და შიფრის მოხსნა ხდება ერთი და იგივე საიდუმლო გასაღების საშუალებით. თანამედროვე კრიპტოგრაფიაში ასევე შედის: ასიმეტრიული კრიპტოსისტემები, ელექტრონული ხელმოწერის სისტემები, ჰეშ ფუნქციები, გასაღებების მართვა, დამალული ინფორმაციების მიღება და კვანტური კრიპტოგრაფია [3].

მაგალითად, ჰეშ ფუნქციები, რომელიც აქტიურად გამოიყენება სხვადასხვა ხელმოწერის სისტემებში, როგორიცაა Merkle-ს ხელმოწერის სისტემა, ასევე აქტიურად გამოიყენება სხვადასხვა მონაცემის დასაჰეშად მონაცემთა ბაზაში. ილუსტრაცია 2-ზე ნაჩვენებია პაროლის ველი მონაცემთა ბაზაში (MySQL), სადაც bcrypt-ის დახმარებით დაჰეშილი პაროლი ინახება.

password
$2y$10$qQ2ut.N9jdt3iDlQ4QrOT..epVnHYgIVwusCvLKI88rjIYNnadNMy
$2y$10$9YqwHy7CZAe.eJ8U6MR6WuVghdNaBR4pRQmyrS0GwgkXgv8zxWrb2
$2y$10$C/cq7z.fXhxH6pPYTZPlO.UbQhho7VnQUgJ5FUYXjx9RrmF1Zus5.
$2y$10$mHCKOZ/oXdSAyZrBLXB1key8LZQ1581b091EzTPJ4MeF0L2zbDYBi
$2y$10$QOtAdG5bIG3xNDFA5Mdz2.sn6cZmlUkC0.WTUCJt6UfMt3TC.cPua
$2y$10$xto.FZlzp9N.5/RQk/AVROSHDD8NSYVbMMAHgfuuBb7xX7eTwQaRC

ილუსტრაცია 2 - დაჰეშილი პაროლი.

ლიტერატურაში [33, 35] აღწერილია კრიპტოგრაფიის საკვანძო ცნებები, დეტალურად არის გარჩეული დახურული გასაღებები, შეტყობინების სინამდვილე, ჰეშ ფუნქციები, ღია გასაღების გადაცემა Puzzles-ის გამოყენებით. ინფორმაციის თვალსაჩინოებისთვის წყაროში გამოყენებულია სხვადასხვა სქემა თავისი აღწერილობით. ასევე წარმოდგენილია ალგორითმის გატეხვის მცდელობა ფსევდო კოდის გამოყენებით SHA-224 , SHA-256, SHA-384 და SHA-512 კრიპტაციის ალგორითმებზე. ალგორითმის გატეხვის მცდელობა გამორჩეულია იმით, რომ იყენებს meet-in-the-middle attack-ის პრინციპს, რომელიც თავის მხრივ იყენებს უამრავ ახალ მეთოდებს ფუნქციის ორ

დამოუკიდებელ ნაწილად დასაყოფად, რომელიც შემდეგში ერთიანდება birthday-style ბაზაში [85, 86].

შესწავლილია ახალი ალგორითმი - multivariate quadratic (MQ) assumption, რომელიც შეიძლება იყოს გამოყენებული public-key encryptions - ის შექმნისთვის. ნაშრომში ავტორებს აქვთ შესწავლილი ორი მიმართულება:

1. MQ-ს ზუსტი ასიმპტოტური ფორმულირება და წარმოადგინეს ემპირიული მტკიცებულება სირთულის დასამტკიცებლად.
2. public-key encryption სქემა და დაამტკიცეს მისი უსაფრთხოება [59]

ჰეშზე დაფუძნებული სისტემები არის ყველაზე პერსპექტიული პოსტ-კვანტურ სამყაროში, მაგრამ ამ სისტემებს აკლიათ დახვეწილობა, მაგალითად ჯგუფური ხელმოწერები (group signatures). მათ მიერ წარდგენილია G-Merkle-ს stateful ჰეშზე დაფუძნებული group signature პირველი ხელმოწერის სქემა. ხელმოწერის სიგრძე არის იგივე, რაც Merkle-ს კლასიკურ სისტემას აქვს [91].

ახლო მომავალში კვანტური პროცესორები დაინერგება პრაქტიკაში [2], შესაბამისად საჭიროა ახალი კრიპტოგრაფიული ალგორითმები, რომელიც იქნება მდგრადი კვანტური კომპიუტერების შეტევებისგან. ნაშრომი განიხილავს კლასიკურ და კვანტურ ალგორითმებს და წარმოადგენს ახალ კრიპტოგრაფიის ალგორითმს.

გარდაუვალია პოსტ-კვანტურ კრიპტოგრაფიის ალგორითმებზე გადასვლა, რომლებიც მოითხოვენ უსაფრთხო ციფრულ ხელმოწერებს. დღესდღეისობით შემოთავაზებულია რამდენიმე ციფრული ხელმოწერა, რომელიც დაფუძნებულია ჰეშ ფუნქციაზე. გაანალიზებულია ალგორითმების დაცულობის დონე და მათი გამოთვლების სიჩქარე[49].

1.1 აქტუალობა

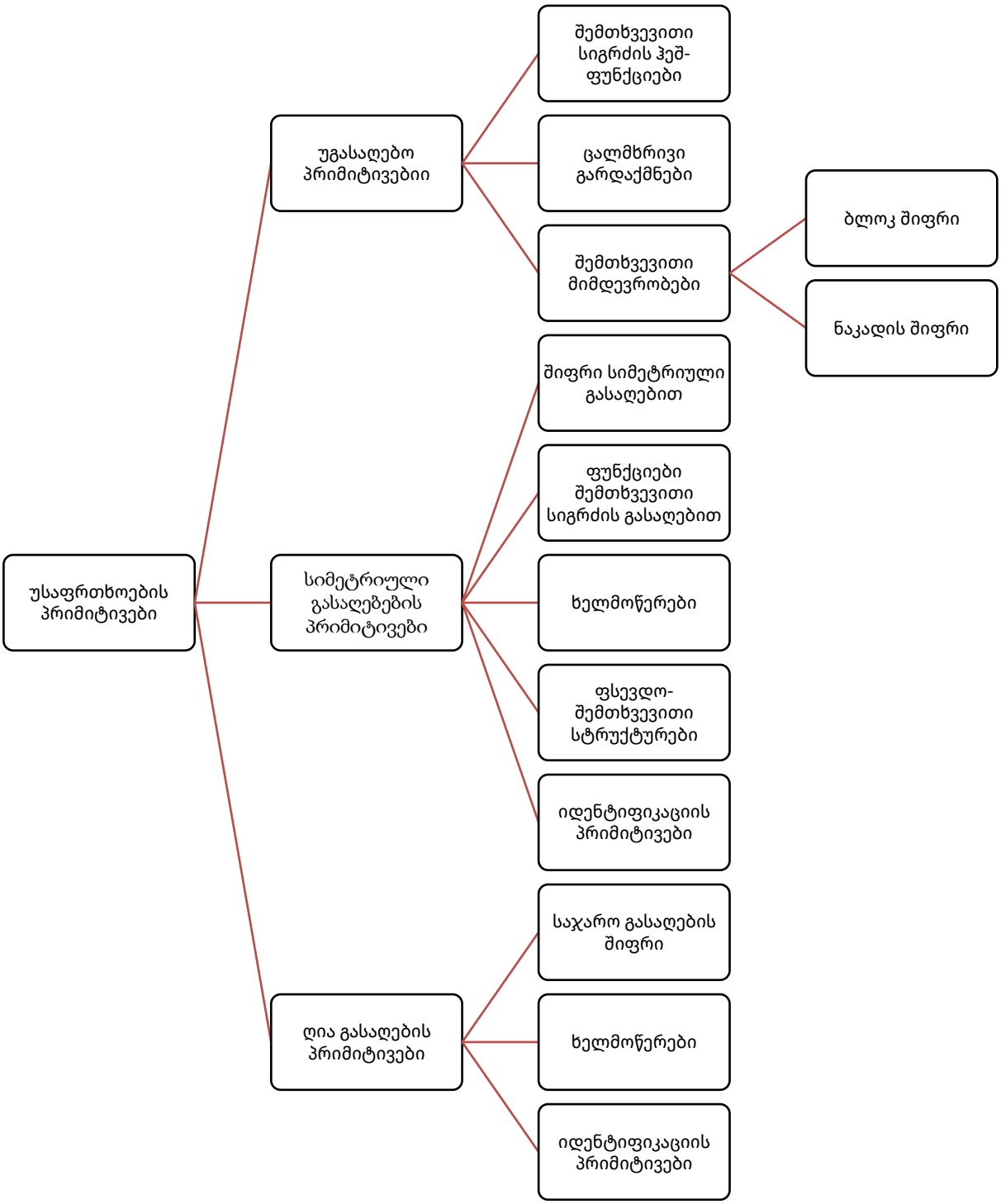
ბოლო დროს სულ უფრო აქტუალური ხდება კვანტური კომპიუტერები. შესაბამისად არსებული სისტემები, რომელიც დაფუძნებულია მარტივი რიცხვების ფაქტორიზაციზე, ხდება არაუსაფრთხო და დაუცველი. ასეთი სისტემების თვალსაჩინო მაგალითი არის RSA . შესაბამისად დგება უსაფრთხო სისტემის შექმნის ან არსებული სისტემების დახვეწის საკითხი. არსებობს მრავალი ხელმოწერის სისტემა, ერთჯერადი და არა მხოლოდ.

კრიპტოგრაფიის ფუნდამენტური მიზანი არის კონფიდენციალურობის, მონაცემების მთლიანობის, უტყუარობის, აუტენთიფიკაციის და საიმედოობის უზრუნველყოფა. სწორედ კრიპტოგრაფიის დახმარებით ხდება თაღლითობის, არასანქცირებული წვდომისა და სხვა მსგავსი დანაშაულებრივი ქმედებების აღმოჩენა და პრევენცია.

ნაშრომში განხილულია კრიტოგრაფიული მეთოდების, სტანდარტები, მათი თეორიული და პრაქტიკული გამოყენების მნიშვნელობა. ასევე წარმოდგენილია კრიპტოგრაფიული მეთოდების სხვადასხვა ალგორითმი.

შესწავლილია ლიტერატურაში მოყვანილი სტატიები და განხილულია მათი შედეგები. სტატიებიდან ნათლად ჩანს, რატომ არის მნიშვნელოვანი ახალი ალგორითმებისა და მათთვის სათანადო სიგრძის გასაღების შერჩევა. სტატიებში ასევე გაანალიზებულია ცნობილი თანამედროვე კრიპტოსისტემების შეტევები და სუსტი წერტილები.

ნახაზი 1 - პრიმიტივები



ყველა ეს პრიმიტივი შეფასებულია შემდეგი კრიტერიუმების მიხედვით:

1. დაცვის დონე - ყოველთვის რთულია ზუსტად გათვალო, თუ რა დონის დაცვა არის საჭირო. ძირითადად განხილულია, როგორც საჭირო მეთოდებისა და ოპერაციების ერთობლიობა სისტემის ყოველგვარი პრობლემის გარეშე ფუნქციონირებისათვის, ასევე ის, თუ როგორ უნდა იქნას აღკვეთილი არასანქცირებული წვდომა.
2. ფუნქციონირება - იმისათვის, რომ მოხდეს სისტემის ფუნქციონირება, აუცილებელია პრიმიტივების კომბინირება, ხოლო რომელი მათგანია ყველაზე ეფექტური, ეს დამოკიდებულია მათ თვისებებზე.
3. ოპერაციის მეთოდები - როდესაც პრიმიტივები გამოყენებული არის სხვადასხვანაირად და სხვადასხვა შემავალი მონაცემით, ისინი იძენს სხვადასხვა თვისებას. ამგვარად, ერთ პრიმიტივს, რომ ჰქონდეს სხვადასხვა ფუნქცია, ეს დამოკიდებული იმაზე, თუ რა გარემოში არის გამოყენებული.
4. წარმადობა - ამ კრიტერიუმს მიეკუთვნება პრიმიტივების ეფექტურობა კონკრეტული ამოცანის შესრულებისას. მაგალითად, დაშიფრვის ალგორითმის შეფასება ხდება წამში დაშიფრული ბიტების რაოდენობით.
5. განხორციელების სიმარტივე - ამ კრიტერიუმს მიეკუთვნება კონკრეტული ამოცანისათვის პრიმიტივების განხორციელების სირთულე. პრიმიტივების რეალიზაცია შეიძლება მოხდეს, როგორც პროგრამულად, ასევე ტექნიკურადაც.

სხვადასხვა კრიტერიუმის მნიშვნელობა დამოკიდებულია ამოცანასა და ხელთ არსებულ რესურსებზე. მაგალითად, ისეთ გარემოში, სადაც კომპიუტერული რესურსი არის შეზღუდული, უსაფრთხოების დაცვის მაღალი სტანდარტები ვერ იქნება გამოყენებული იმის გამო, რომ სისტემამ იფუნქციონიროს პრობლემების გარეშე [3-8].

შიფრაციის მეთოდი რომელიც აღწერილია წყაროში არის განსხვავებული, იქიდან გამომდინარე, რომ პირველი მეთოდია, რომელიც იყენებს გასაღების ღია მეთოდით გადაცემას. მნიშვნელოვანი არის ის, რომ გასაღების გადაცემისთვის აღარ არის საჭირო უსაფრთხოების დამატებითი ნორმების მიღება. ასევე ხდება შეტყობინების ხელმოწერა, შესაბამისი ღია გასაღების გამოყენების დროს ყველას შეუძლია ამ ხელმოწერის გადამოწმება [27, 28].

გარდა შიფრაციის მეთოდების ეს წიგნებში და სტატიები ავტორები აღწერენ არამარტო კრიპტოგრაფიის შესავალს, არამედ დაწვრილებით არის განხილული არსებული

ალგორითმების უმრავლესობა. მანდ ნაჩვენებია ალგორითმების როგორც ძლიერი, ასევე სუსტი წერტილები, აღწერილია ბევრი ალგორითმი, რაც მთავარია ისინი იძლევიან პრაქტიკულ რჩევებს, თუ როგორ შეიძლება მათი გამოყენება უსაფრთხოების პრობლემების გადასაჭრელად. ასევე მოიცავს ბევრ კონსტრუქციას კრიპტოგრაფიაში სხვადასხვა დავალების შესასრულებლად. თითოეული ამოცანისთვის წიგნებში განისაზღვრება ზუსტი მიზანი, რომლის მიღწევისთვის იქმნება კონსტრუქციები, პარალელურად არის ნაჩვენები ტიპიური შეცდომები, რომელიც შეგვიძლია თავიდან ავიცილოთ, განხილულია სისტემებზე რეალური თავდასხმები [29, 32, 34, 36, 37].

1.2 მიზნები, ამოცანები, სამეცნიერო სიახლე და შედეგები

მიზნები. ნაშრომში ჩატარებული კვლევის მიზანი არის ისეთი სისტემის წარმოდგენა, რომელიც იქნება მდგრადი კვანტური კომპიუტერების შეტევებისგან და ამავდროულად საკმაოდ ეფექტური.

ამოცანები. ნაშრომში რეალიზებულია შემდეგი ამოცანები:

1. არსებული სისტემების განხილვა.
2. კლასიკური კრიპტოგრაფიის პოსტ-კვანტურთან შედარება.
3. სწრაფი და ამავდროულად უსაფრთხო სისტემის შექმნა.
4. ახალი სისტემის შესრულების დროის დათვლა და ძველი სისტემის შესრულების დროსთან შედარება.
5. ახალი სისტემის ოპერაციების რაოდენობის შემცირება და ასევე ძველ სისტემასთან შედარება.

სამეცნიერო სიახლე. ნაშრომის სამეცნიერო სიახლეს წარმოადგენს დინამიური ალგორითმი, რომელიც იქნება სწრაფი და ამავდროულად უსაფრთხო. ამ ალგორითმში ოპერაციების რაოდენობა იქნება ბევრად ნაკლები ვიდრე არსებულ სისტემაში. ამასთან ოპერაციების რაოდენობის სიმცირე და ალგორითმის სისწრაფე იქნება დამოკიდებული პროცესორის ნაკადების (thread - ების) რაოდენობაზე.

ნაშრომის პრაქტიკული ღირებულება. შემუშავებული იქნა ალგორითმი დაპროგრამების ენა Python-ზე, რომელიც წარმოადგენს პარალელურ ალგორითმს და იყენებს პროცესორის ნაკადებს. შედეგად მიღებული ალგორითმი, რომლის მიერ გამოყენებული ოპერაციების რაოდენობა მნიშვნელოვნად არის შემცირებული და გამოთვლების სისწრაფე რამდენჯერმე აღემატება უკვე არსებულ კლასიკური მერკლეს ალგორითმის სიჩქარეს.

1.3 ისტორიული ფაქტები

კვანტური კომპიუტერის მიერ RSA, DSA და ECDSA კრიპტოგასაღების გატეხვის შემდეგ, ჩვეულებრივი ინტერნეტ მომხმარებლებისთვის შესაძლებელია შეიქმნას ისეთი წარმოდგენა, რომ თანამედროვე შეტევების მიმართ არსებული სისტემები დაუცველია. ინტერნეტ მომხმარებლებისთვის ან კომპანიებისთვის დაცვის ერთადერთი საიმედო გამოსავალი, რათა ინფორმაცია არ იყოს ხელმისაწვდომი ჰაკერებისთვის შეიძლება იყოს ფიზიკური ფარი. მაგალითად, USB მეხსიერების დამალვა კარგად დაცულ სეიფში. შეიძლება ვივარაუდოთ, რომ კვანტურ კომპიუტერებს შეუძლიათ RSA , DSA და ECDSA მარტივად გატეხვა ან ჩავთვალოთ, რომ კვანტური კომპიუტერები უსარგებლოს გახდის კრიპტოგრაფიას.

RSA - ზე განხორციელებული adaptive chosen ciphertext ახალი შეტევა არის განხილული და აღწერილი [7, 10, 11] სტატიებში. ავტორები უშვებენ იმას, რომ ჰაკერს აქვს წვდომა chosen ciphertext - ზე. შესაბამისად, RSA private-key ოპერაციის დროს სისტემა აბრუნებს ბიტებს, რომლებიც უთითებენ მონაცემების უცნობ ბლოკებზე, რომლებიც დაშიფრულია PKCS მეშვეობით. [31] წყაროში არის აღწერილი RSA -ს ალგორითმზე შეტევა. შეტევის ალგორითმი არის დაფუძნებული უწყვეტ წილადებზე, რომელიც პოლინომიალურ დროის განმავლობაში ნახულობს მრიცხველს და მნიშვნელს, როდესაც ცნობილია წილადის საკმარისად ახლო შეფასება. წყაროში დეტალურად არის მოყვანილი საჭირო ინფორმაცია, რომელიც დაგვჭირდება RSA -ს ალგორითმზე შეტევის.

გვხვდება ისეთი შემთხვევები, სადაც პროცესორის ერთ-ერთი ტექნოლოგიის გამო შეიძლება მოხდეს წვდომა დახურულ მონაცემებზე. [25] ამ წყაროში არის აღწერილი Hyper-Threading - ის უსაფრთხოების სერიოზული ხარვეზი. ეს ხარვეზი ამლევს არაპრივილეგირებულ მომხმარებელს მოიპოვოს ლოკალური ინფორმაცია და ასევე RSA დახურული გასაღები.

[26] წყაროში განიხილება ფაქტორიზაციის ალგორითმები, კონკრეტულად კი quadratic sieve და Pollard's number field sieve , რომლიც მარტივ მამრავლებად შლის RSA - ს 130 რიცხვს. განსხვავება მდგომარეობს იმაში, რომ Pollard's number field sieve - ი არის ბევრად სწრაფი ალგორითმი ვიდრე quadratic sieve , Pollard's number field sieve - ის ალგორითმს მიახლოებით სჭირდება 15% იმ დროის რაც quadratic sieve - ის ალგორითმს.

RSA, DSA და ECDSA უკან დგას კრიპტოგრაფიის ბევრი და მნიშვნელოვანი კლასი:

1. ჰეშირებაზე დაფუძნებული კრიპტოგრაფია. კლასიკური მაგალითი არის Merkle- ს ჰეშ public-key - ზე დაფუძნებული სისტემა (1979), რომელიც არის დაფუძნებული Lamport - ის და Diffie - ის იდეაზე.
2. კოდზე დაფუძნებული კრიპტოგრაფია. კლასიკური მაგალითი არის McEliece's - ის დამალული Goppa კოდი public-key - ზე დაფუძნებული სისტემა (1978).
3. Lattice - ზე დაფუძნებული კრიპტოგრაფია. მაგალითი, რომელმაც ალბათ ყველაზე დიდი ინტერესი გამოიწვია, არის Hoffstein-Pipher-Silverman NTRU public-key - ზე დაფუძნებული სისტემა (1998).
4. Secret-key კრიპტოგრაფია. ყველაზე კარგი მაგალითი არის Daemen-Rijmen Rijndael შიფრი(1998), რომელსაც შემოკლებით ჰქვია AES (Advanced Encryption Standard) .

მიჩნეული არის, რომ ეს დაცვის სისტემები უძლებს კლასიკურ და კვანტურ კომპიუტერებს. ჯერ-ჯეროობით ვერავინ ვერ გადაწყვიტა, როგორ მოარგონ

Shor - ის ალგორითმი, რომელიც ადვილად უმკლავდება RSA - ს, DSA - ს და ECDSA - ს რომელიმე ზემოთ აღწერილ სისტემას. სხვა კვანტური ალგორითმი არის Grover - ის ალგორითმი. ის არ არის ისეთი სწრაფი, როგორიც არის Shor - ის ალგორითმი[9-12].

შესაძლოა არსებობდეს ამ სისტემებზე ძლიერი შეტევები. კრიპტოგრაფიისათვის ეს არის ნაცნობი რისკი. ამიტომაც ძალიან ბევრი გაერთიანება კრიპტოანალიზის წარმოებისას ხარჯავს უამრავ დროს და ენერგიას. ხანდახან კრიპტოანალიტიკოსები ეძებენ ისეთ გამანადგურებელ შეტევებს, რომლებიც თვალსაჩინოდ აჩვენებს მათ, რომ ეს კონკრეტული სისტემა არის გამოუსადეგარი კრიპტოგრაფიისათვის. ზოგჯერ კრიპტოანალიტიკოსები ეძებენ ისეთ შეტევებს, სადაც საჭიროა გრძელი გასაღების გამოყენება, ხან კი კრიპტოანალიტიკოსებს უწევთ სისტემის წლობით შესწავლა, მაგრამ მაინც ვერ პოულობენ კარგ შეტევებს.

განვიხილოთ შემდეგი სამი შეტევა RSA - ს წინააღმდეგ:

1. 1978. Rivest Shamir და Adleman - ის ერთ-ერთ ჩანაწერში არის აღნიშნული Schroepel - ის linear sieve - ი, რომლის საშუალებითაც ხდება RSA ალგორითმის ფაქტორიზაცია და შესაბამისად, RSA ალგორითმი ტყდება $2^{(1+O(1))(lgn)1/2(lglg\,(\,lgn))1/2}$ მარტივი ოპერაციების შემდეგ. linear sieve - ის მიერ 2^b ოპერაციების არჩევა ნიშნავს ისეთი n - ის არჩევა, რომელსაც $(0.5\ +\ O(1))b^2/\,lg\ b$ რაოდენობის ბიტები ექნება.

გაფრთხილება: $(0.5 + O(1))b^2 / lg\ b$ ნიშნავს იმას, რომ რაღაც დაიყვანება 0.5 - მდე, როგორც $b \to \infty$. აქ არაფერი არ არის ნათქვამი იმაზე, რომ მაგალითად b = 128. n - ის ზუსტი ზომის გარკვევისთვის, სადაც b = 128, საჭიროა linear sieve - ის სიჩქარეზე დაკვირვება.

2. 1988. Pollard - მა წარმოადგინა ფაქტორიზაციის ახალი ალგორითმი number-field sieve . ეს ალგორითმი შემდგომში განზოგადოებული იყო Buhler - ის, Lenstra - ს და Pomerance - ის მიერ. ის ახდენს RSA - ის ფაქტორიზაციას $2^{(1.9\ldots+o(1))(lg\ n)\ 2/9\ (lg\ lg\ n)}$ მარტივი ოპერაციების გამოყენებით. number-fiel sieve - ის მიერ 2^b ოპერაციების არჩევა ნიშნავს ისეთი n - ის არჩევა, რომელსაც $(0.016 + O(1))b^3 / (lg\ lg\ b)^{\ 2}$ რაოდენობის ბიტები ექნება. დღესდღეობით, 20 წლის შემდეგ, კლასიკური კომპიუტერებისთვის განკუთვნილი ფაქტორიზაციის უსწრაფესი ალგორითმები დღემდე იყენებენ $2^{(constant+o(1)\cdot lglg\ n\ \cdot \frac{1}{3} \cdot lglg\ (\ lglg\ n\ ) \cdot \frac{2}{3}}$ ოპერაციას.[26]
3. 1994. Shor - მა წარმოადგინა ისეთი ალგორითმი, რომელიც კვანტურ კომპიუტერზე ახდენს RSA - ს ფაქტორიზაციას $(lg\ n)^{1+O(1)}$ მარტივი ოპერაციების გამოყენებით. იმისათვის, რომ ამ ალგორითმმა შეასრულოს 2^b ოპერაცია, უნდა აირჩეს ისეთი n, რომელსაც ექნება $2^{(0.5\ +\ o(1))b}$ რაოდენობის ბიტები.

დავუშვათ, შედარებისთვის გვაქვს შეტევა 30 წლის წინანდელ public-key კრიპტოსისტემაზე, რომელსაც ჰქვია McEliece's hidden-Goppa-code . McEliece - მა ასევე წარმოადგინა შეტევა, რომელიც უმკლავდება კოდებს $2^{(0.5\ +\ o(1)\ )n\ /\ lg\ n}$ ოპერაციების გამოყენებით, რომელთა სიგრძე არის n და განზომილება არის n/2. იმისათვის, რომ ამ შეტევამ შეასრულოს 2^b ოპერაცია, უნდა აირჩეს ისეთი n, რომელსაც ექნება

$2^{(0.5\ +\ o(1))b\ lg\ b}$.

შემდგომში შეტევის ოპერაციების ციფრი იყო შემცირებული დიდი ფაქტორის დახმარებით. დაახლოებით $n^{lg\ n} = 2^{(\ lg\ n\ )2}$, მაგრამ (lg n)² არის ნაკლები, ვიდრე $0.5n\ /\ lgn$ თუ n არის დიდი რიცხვი. განახლებული შეტევები დღემდე იყენებენ $2^{(0.5\ +\ o(1)\ )n\ /\ lg\ n}$ ოპერაციებს. კვანტური კომპიუტერების შემთხვევაში ხდება კონსტანტის დაწევა 0.5 - მდე [13,14].

McEliece - ის კრიპტოსისტემა არის RSA - ის ერთ-ერთი ალტერნატივა, რომელიც იყო შემუშავებული რობერტ მაკელისის მიერ 1978 წელს, მაგრამ სამწუხაროდ ასეთი

პოპულარული არ გახდა როგორც RSA . [42] ამ წყაროში ავტორების მიერ არის შეთავაზებული გაუმჯობესებული ვერსია Stern's - ის შეტევის McEliece - ის კრიპტოსისტემაზე. წყარო გვიჩვენებს, რომ McEliece - ის კრიპტოსისტემის გატეხვა შესაძლებელია 1400 დღეში თუ კომპიუტერს აქვს 2.4GHz Core 2 Quad პროცესორი ან 7 დღეში თუ გამოყენებაში იქნება 200 პროცესორისგან შემდგარი კლასტერი.

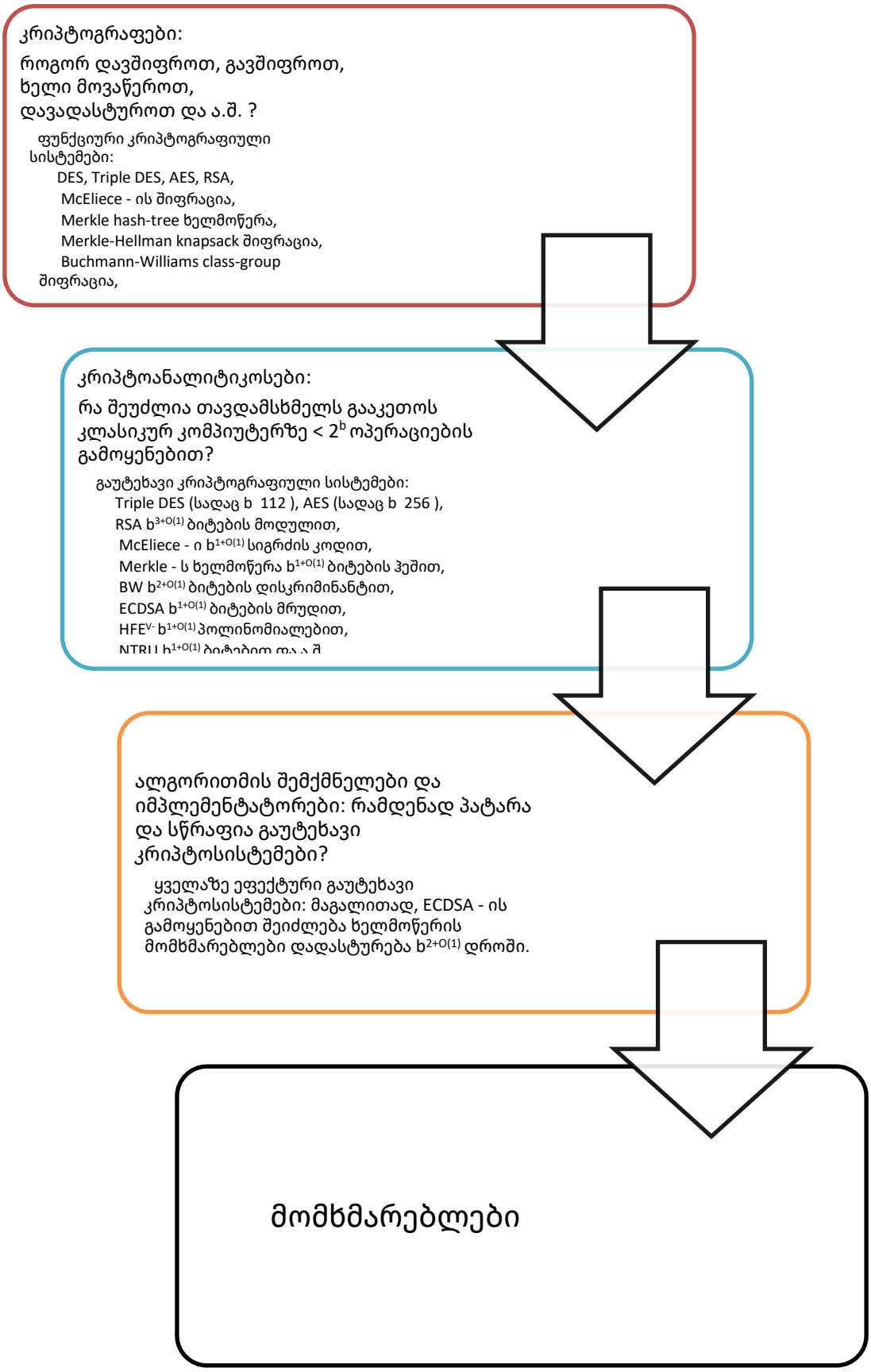
მიუხედავად იმისა, რომ ეს სისტემა არ არის პოპულარული, ავტორების მიერ არის შემოთავაზებული McEliece public-key სქემის ახალი იმპლემენტაცია, სადაც დაწვრილებით აღწერილია ალგორითმის ყველა პარამეტრი [43]. ამ წიგნში არის მოყვანილი McEliece public-key შიფრაციის სქემა და ამ სქემის რამდენიმე ვარიანტი, რომელიც პოსტ-კვანტური ღია გასაღების დაშიფვრის სტანდარტის კანდიდატები არის [50].

McEliece's - ის სანაცვლოდ RSA - ის გამოყენების მიზეზი არის გასაღების ზომა. McEliece's - ის ღია გასაღები იყენებს დაახლოებით $n^2/4 \approx b^2(lg\ lg\ b)^2$ რაოდენობის ბიტებს, რა დროსაც RSA - ის ღია გასაღები იყენებს დაახლოებით $(0.016)b^3 / (lg\ b)^2$ რაოდენობის ბიტებს. თუ b იქნებოდა $b^{2+O(1)}$ - ზე ბევრად დიდი, მაშინ McEliece's - თვის ბიტების რაოდენობა იქნებოდა უფრო პატარა, ვიდრე $b^{3+O(1)}$ ბიტების რაოდენობა RSA - თვის, მაგრამ რეალურ სამყაროში, სადაც b = 128 აძლევს RSA - ს იმის საშუალებას, რომ მისი გასაღების ზომა იყოს რამდენიმე ათასი ბიტი, რა დროსაც McEliece's - ის გასაღები ზომა თითქმის მილიონს გაუტოლდება[15, 16].

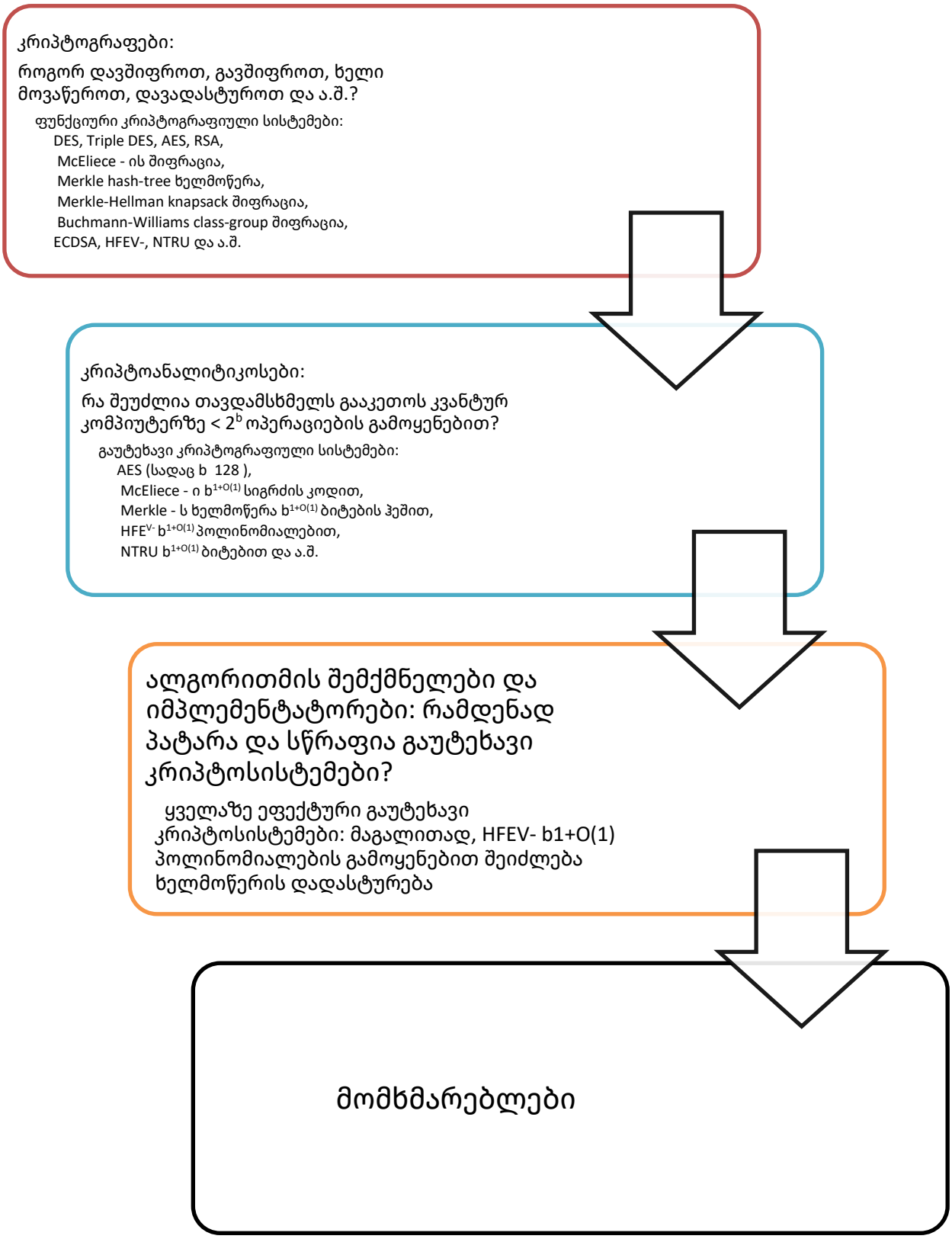
ნახაზი 2 - ზე არის ნაჩვენები კვანტური კომპიუტერების ანონსამდე გამოსვლის ანუ პრე-კვანტური კრიპტოგრაფიული სისტემების შემუშავების, ანალიზის და ოპტიმიზაციის პროცესი. ნახაზი 3 გვიჩვენებს იგივე პროცესს, ოღონდ პოსტ-კვანტური კრიპტოგრაფიისათვის. ორივე ნახაზს აქვს ერთნაირი სტრუქტურა:

1. კრიპტოგრაფები ქმნიან სისტემას დაშიფრული და დაუშიფრავი მონაცემებისათვის;
2. კრიპტოანალიტიკოსები ტეხავენ ზოგ სისტემას;
3. ალგორითმების შემქმნელები და იმპლემენტატორები ამ სისტემებს შორის ეძებენ უსწრაფესს და უსაფრთხო სისტემებს.

ნახაზი 2 - პრე-კვანტური კრიპტოგრაფია



ნახაზი 3 - პოსტ-კვანტური კრიპტოგრაფია



1.4 ჰეშირებაზე დაფუძნებული public-key ხელმოწერების სისტემა

ხელმოწერის აღნიშნულ სისტემას სჭირდება სტანდარტული კრიპტოგრაფიული ჰეშ ფუნქცია H, რომლის შედეგია 2b ბიტი. 128 ბიტისთვის H ფუნქციად შეიძლება შერჩეული იქნას SHA-256 ჰეშ ფუნქცია. უკანასკნელი რამდენიმე წლის განმავლობაში ბევრი პრობლემა წარმოიშვა ჰეშ ფუნქციების უსაფრთხოების მიმართ. შედეგად, მომავალი რამდენიმე წლის განმავლობაში NIST - ი გამოაცხადებს კონკურსს SHA-256 - ის ჩანაცვლებაზე, თუმცა ყველა ცნობილი შეტევა SHA-256 - ის წინააღმდეგ არის რესურსტევადი და მოითხოვს დიდ სიმძლავრეებს.

მაგალითად, ხელმომწერის სისტემაში public-key-ის აქვს $8b^2$ ბიტი: 16 კილობაიტი, სადაც b = 128. გასაღები შედგება 4b ჩანაწერებისგან, $y_1[0]$, $y_1[1]$, $y_2[0]$, $y_2[1]$, …. , $y_{2b}[0]$, $y_{2b}[1]$. ყოველი ჩანაწერის სიგრძე არის 2b ბიტი.

m შეტყობინებას აქვს 2b (2b + 1) ბიტი სიგრძის ხელმოწერა: მაგალითად, 8 კილობაიტი, სადაც b = 128 . ხელმოწერა შედგება 2b ბიტების r, x_1, … ,x_{2b} სტრიქონისგან. (h_1, … , h_{2b}) H(r,m) - დან აკმაყოფილებს $y_1[h_1] = H(x_1)$, $y_2[h_2] = H(x_2)$ … $y_{2b}[h_{2b}] = H(x_{2b})$.

ხელმომწერი იწყებს ხელმოწერის გასაღების - x-ის გენერაციას, ხოლო ამის შემდეგ იწყებს y = H(x) გამოთვლას. ხელმომწერის საიდუმლო გასაღებს არის $8b^2$ ბიტი სიგრძის ტოლი, რომლებიც არის 4b დამოუკიდებელი ერთგვაროვანი შემთხვევითი $x_1[0]$, $x_1[1]$, $x_2[0]$, $x_2[1]$, …. , $x_{2b}[0]$, $x_{2b}[1]$ სტრიქონები, ყოველ სტრიქონს გააჩნია 2b ბიტის სიგრძე. ხელმომწერი ითვლის $y_1[0]$, $y_1[1]$, $y_2[0]$, $y_2[1]$, …. , $y_{2b}[0]$, $y_{2b}[1]$ ღია გასაღებს, როგორც $H(x_1[0])$, $H(x_1[1])$, $H(x_2[0])$, $H(x_2[1])$, …. , $H(x_{2b}[0])$, $H(x_{2b}[1])$.

იმისათვის, რომ მოხდეს m შეტყობინების ხელმოწერა, ხელმომწერი აგენერირებს შემთხვევით r სტრიქონს, ითვლის H(r,m) - გან (h_1, … , h_{2b}) ბიტებს და ავლენს (r, $x_1[h_1]$, … , $x_{2b}[h_{2b}]$) როგორც m - ის ხელმოწერას. ამის შემდეგ, ხელმომწერი აუქმებს მიმდინარე x მნიშვნელობებს და აღარ ახდენს სხვა შეტყობინებების ხელმოწერას.

ამ სისტემას ეწოდება Lamport-Diffie ერთჯერადი ხელმოწერების სისტემა. თუ ხელმომწერს უნდა ერთზე მეტი შეტყობინების ხელმოწერა, მაშინ საჭიროა chaining - ი. ხელმომწერი რთავს ხელმოწერილ შეტყობინებაში ახლად დაგენერირებულ ღია გასაღებს, რომელიც იქნება გამოყენებული შემდეგი შეტყობინების ხელმოსაწერად. დამმოწმებელი ამოწმებს პირველ ხელმოწერილ შეტყობინებას, სადაც არის ჩართული ღია გასაღები.

შესაბამისად, მას შეუძლია შეამოწმოს შემდეგი შეტყობინების ხელმოწერა. n-ურ შეტყობინებაში შედის ყველა წინა ხელმოწერილი n-1 შეტყობინება. უფრო დახვეწილი სისტემა, როგორიცაა Merkle's hash tree - ზე დაფუძნებული ხელმოწერების სისტემა, იზრდება ლოგარითმულად ხელმოწერილი შეტყობინებების რაოდენობასთან ერთად.

Grover - ის ალგორითმი არის ზოგადი ფუნქციების ინვერტირების უსწრაფესი კვანტური ალგორითმი და მიჩნეულია, როგორც სპეციფიურ, ეფექტურად გამოთვლილი ფუნქციების (თუმცა არის ბევრი გამონაკლისი) ინვერტირების უსწრაფესს კვანტურ ალგორითმი. ჰეშზე დაფუძნებულ კრიპტოსისტემას შეუძლია მოახდინოს რთულად ინვერტირებად ფუნქციების კონვერტაცია public-key ხელმოწერების სისტემებში [17-20].

Quantum image steganography პროტოკოლი, რომელიც არის დაფუძნებული quantum image expansion - ზე და ასევე არის აღწერილი Grover - ის ძებნის ალგორითმი. Grover - ის ძებნის ალგორითმი არის გამოყენებული სწორი quantum image - ის მოსაძებნად. ეს ყველაფერი იყო შექმნილი და გატესტილი MATLAB - ის გამოყენებით [39-40].

[41] აქ ავტორების მიერ არის განხილული კვანტური კომპიუტერების ბლოკურ შიფრებზე შეტევა. აღნიშნული შეტევა იყენებს ($O\sqrt{N}$) calls - ს, რომლის დახმარებითაც ეძებს შიფრის გასაღებს, რომლის ზომა არის N. ალგორითმის იმპლემენტაცია განხორციელებულია Q# პროგრამირების ენაზე დაფუძნებით.

1.5 Code - ზე დაფუძნებული public-key დაშიფრვის სისტემა

Code - ზე დაფუძნებული public-key დაშიფრვის სისტემის განსახილველად დავუშვათ, რომ b წარმოადგენს 2 - ის ხარისხს ; $n = 4b \lg b$; $d = [\lg n]$ და $t = [0.5n / d]$. მაგალითად, თუ $b = 128$, მაშინ $n = 3584$, $d = 12$ და $t = 149$

ამ სისტემაში მიმღების public გასაღები არის $dt \times n$ K მატრიცა კოეფიციენტებით F_2 - ში. დასაშიფრი შეტყობინებების ზომა არის n ბიტი t წონით. იმისათვის, რომ მოხდეს m შეტყობინების დაშიფრვა, გამგზავნი ამრავლებს K - ს m - ზე, შედეგად იღებს dt ბიტების სიგრძის Km დაშიფრულ ტექსტს.

ჰაკერის ძირითადი პრობლემა არის syndrome-decode K , ანუ K - ზე გამრავლების ოპერაციის უკან დაბრუნება (undo) მიუხედავად იმისა, რომ შეყვანილი მონაცემის სიგრძე არის ცნობილი და არის t. ამის მარტივად გაკეთება შესაძლებელია წრფივი ალგებრის გამოყენებით. იმისათვის, რომ მივიღოთ $K_v = K_m$, საჭიროა K_m - დან ვიმოძრაოთ რომელიმე n ბიტიან ვექტორისკენ, თუმცა გვაქვს v - ს დიდი არჩევანი და შეიძლება ისე ჩანდეს, რომ t - ს მოძებნა იქნება ძალიან რთული.

იმისათვის, რომ მიმღებმა გადაწყვიტოს ზემოთ აღწერილი პრობლემა, ის აგენერირებს K public გასაღებს საიდუმლო სტრუქტურით, ე.წ. hidden Goppa code სტრუქტურით, რომელიც მიმღებს აძლევს საშუალებას მოახდინოს დეშიფრაცია შედარებით ნაკლებ დროში. შესაძლოა, ჰაკერმა public გასაღებში აღმოაჩინოს hidden Goppa code სტრუქტურა, მაგრამ ჯერჯერობით მსგავსი შეტევა არ არის ცნობილი.

კონკრეტულად, მიმღები გამოთვლებს იწყებს უნიკალური $\alpha_1, \alpha_2, \dots, \alpha_n$ ელემენტებისგან, F_{2^d} სიმრავლიდან და საიდუმლო $g \in F_{2^d}[x]$ პოლინომიალისგან. მიმღების ძირითადი სამუშაო არის $dt \; x \; n$ მატრიცის დეკოდირება

$$H = \begin{pmatrix} 1/g(\alpha_1) & \cdots & 1/g(\alpha_n) \\ \alpha_1/g(\alpha_1) & \dots & \alpha_n/g(\alpha_n) \\ \vdots & \ddots & \vdots \\ \alpha_1^{t-1}/g(\alpha_1) & \cdots & \alpha_n^{t-1}/g(\alpha_n) \end{pmatrix},$$

სადაც F_{2^d} - ს ყოველი ელემენტი განხილულია, როგორც d ელემენტების სვეტი. აღნიშნული H მატრიცა არის parity-check, მატრიცა ბინარული Goppa კოდი და შეიძლება იყოს დეკოდირებული Patterson - ის ან სხვა უფრო სწრაფი ალგორითმების საშუალებით.

მიმღების K public key არის H - ის ვერსია. უფრო ზუსტად, მიმღების საიდუმლო გასაღები ასევე მოიცავს ინვერტირებად dt x dt S მატრიცას და გადანაცვლების n x n P მატრიცას. ღია გასაღები არის SHP გამოთვლის შედეგი. დავუშვათ, გვაქვს $K_m = SHP_m$ დაშიფრული ტექსტი. მიმღები ამრავლებს მას S^{-1} - ზე იმისათვის, რომ მიიღოს HP_m , შემდეგ ახდენს H - ის დეკოდირებას იმისათვის, რომ მიიღოს P_m და ბოლოს ამრავლებს მას P^{-1} - ზე იმისათვის, რომ მიიღოს m - ი [21-26].

[51, 52] სტატიაში არის მოყვანილი Code - ზე დაფუძნებული ალგორითმი და განხილულია ის ალგორითმები, რომელიც შექმნილია MDPC - ის სქემაზე დაყრდნობით. წყაროში არის ნაჩვენები კოდზე დაფუძნებული მნიშვნელოვანი და ფუნდამენტალური შიფრაციის თვისებები. განსაკუთრებით არის აღწერილი ერთი შეცდომის შაბლონი.

1.6 მრავალრიცხოვანი კვადრატული public-key ხელმოწერის სისტემა

ამ სისტემაში public key არის $P_1, P_2, \dots, P_{2b} \in F_2[W_1, \dots, W_{4b}]$ მიმდევრობა ხოლო 2b პოლინომიალების მიმდევრობა 4b $w_1, \dots, w_{4b}$ ცვლადებში $F_2 = \{0,1\}$ კოეფიციენტებით. ყველა პოლინომიალს აუცილებლად უნდა ჰქონდეს 2 - ის ხარისხი და უნდა იქნას წარმოდგენილი $1 + 4b + 4b(4b - 1) / 2$ ბიტების მიმდევრობის სახით. აღნიშნული პოლინომი წარმოდგენილია კი $1, w_1, \dots, w_{4b}, w_1w_2, w_1w_3, \dots, w_{4b-1}w_{4b}$ კოეფიციენტებით. საერთო ჯამში public key - ს აქვს $16b^3 + 4b^2 + 2b$ რაოდენობის ბიტები. მაგალითად, b = 128 - თვის გვაქვს 4 მეგაბაიტი. m შეტყობინების ხელმოწერას აქვს სულ რაღაც 6b ბიტი. კერძოდ, 4b $w_1, \dots, w_{4b} \in F_2$ მნიშვნელობებით, 2b ბიტების სიგრძის მქონე r სტრიქონი, რომელიც აკმაყოფილებს შემდეგ პირობას:

$$H(r,m) = (P_1(w_1, \dots, w_{4b}), \dots, P_{2b}(w_1, \dots, w_{4b})) .$$

ამ შემთხვევაში H არის სტანდარტული ჰეშ ფუნქცია. ალგორითმი ხელმოწერის დადასტურებისას იყენებს H - ის ერთ შეფასებას და b^3 ბიტების ოპერაციას $P_1, \dots, P_{2b}$ შეფასებისთვის.

ხელმოწერის ალგორითმის უპირატესობა ჰეშზე დაფუძნებული ხელმოწერის სისტემაზე არის ის, რომ ყოველი ხელმოწერა არის მოკლე. სხვა მრავალრიცხოვან კვადრატულ სისტემებს აქვს უფრო მოკლე ხელმოწერები და უმეტეს შემთხვევებში აქვს უფრო მოკლე public keys - ები.

ძირითადი პრობლემა, რომელიც აქვს ჰაკერს არის 4b ბიტების $w_1, \dots, w_{4b}$ მიმდევრობა, რომლებიც აგენერირებს სპეციფიურ 2b ბიტების შედეგს:

$$(P_1(w_1, \dots, w_{4b}), \dots, P_{2b}(w_1, \dots, w_{4b})).$$

4b ბიტების მიმდევრობის გამოცნობა სწრაფია, მაგრამ საშუალოდ წარმატების შანსები არის მხოლოდ 2^{-2b} . უფრო ეფექტური ალგორითმი ამოხსნის შეტევებს, როგორიც არის XL, შეუძლია წარმატებით განახორციელოს შეტევა 2^{2b} ოპერაციაში, მაგრამ დღემდე არ არის ცნობილი შეტევები 4b ცვლადებით $P_1, \dots, P_{2b}$ კვადრატული პოლინომიალების უმრავლესობისათვის, რომელიც წარმატებით განახორციელებს შეტევას 2^b ოპერაციაში. ამ პრობლემის სირთულე არ არის გასაკვირი, რადგან ის არის ზოგადი.

ამ პრობლემის გადასაჭრელად ხელმომწერი აგენერირებს $P_1, \dots, P_{2b}$ public key - ს უცხო სტრუქტურით. კონკრეტულად, HFE^{-v} სტრუქტურით, რომელიც საშუალებას აძლევს ხელმომწერს ამოხსნას განტოლებები განსაზღვრულ დროში. ბუნებრივია, რომ

ხელმომწერს თეორიულად შეუძლია დაადგინოს HFE^{-v} სტრუქტურა public key - ში, მაგრამ მსგავსი შეტევები ჯერ არ არის ცნობილი.

განვიხილოთ შეუქცევადი $\varphi \in F_2$ [t] პოლინომი 3b ხარისხში. განვსაზღვროთ L , როგორც F_2 [t] / φ 2^{3b} ზომის მქონე. ხელმოწერაში კრიტიკული ნაბიჯი არის საიდუმლო low-degree L[x] პოლინომის ფუძის მოძებნა. არსებობს რამდენიმე სტანდარტული ალგორითმი, რომლის მეშვეობითაც შეიძლება ამის გაკეთება $b^{O(1)}$ დროში.

საიდუმლო პოლინომიალი ისე არის არჩეული, რომ $2^i + 2^j$ ან 2^i - ს ჰქონდეს არანულოვანი ექსპონენტები. თუ $x \in L$ ელემენტი ყოველი $x \in F_2$ - თან ერთად არის გამოხატული $x_0 + x_1t + \dots + x_{3b-1}t^{3b-1}$ ფორმით, მაშინ $x^2 = x_0 + x_1t^2 + \cdots + x_{3b-1}t$, $x^4 = x_0 + x_1t^4 + \cdots + x_{3b-1}t^{12b-4}$ არის კვადრატული პოლინომიალი. რამდენიმე მარტივი მანიპულაციის საშუალებით კი შესაძლებელია ზემოთ აღწერილი პოლინომიალის სტრუქტურის დამალვა, რომლის დახმარებითაც ხდება ხელმომწერის public key - ის გენერირება.

ხელმომწერის secret key - ის აქვს სამი კომპონენტი:

1. შებრუნებადი 4 x 4 S მატრიცა F_2 კოეფიციენტებით.
2. Q ∈ L[x, v_1, v_2, … , v_b] პოლინომიალი, სადაც ყოველ პირობას აქვს 6 ფორმა:
   $lx^{2^i+2^j}$, სადაც $l \in L, 2^i < 2^j, 2^i + 2^j \leq 2^b$; $lx^{2^i} v_j$,
   სადაც $l \in L, 2^i \leq 2^b$; $l\, v_i\, v_j$; lx^{2^i} ; lv_j ; l . თუ b = 128 , მაშინ არსებობს 9446 ვარიანტი, სადაც ყოველ ვარიანტს აქვს 384 ბიტიანი l კოეფიციენტი.
3. 2b x 3b T მატრიცა F_2 კოეფიციენტებით.

ალგორითმი ითვლის public key - ის შემდეგნაირად: გამოითვლება (x_0, x_1, … , x_{3b-1}, v_1, v_2, … , v_b) კომპონენტიანი ვექტორი S - ჯერ და შემდგომ ეტაპზე მიიღება (w_1, … , w_{4b}) კომპონენტიანი ვექტორი. შემდეგი გამოსახულების გამოყენებით L[w_1, … , w_{4b}] / ($w_1^2 - w_1, \dots, w_{4b}^2 - w_{4b}$) quotient ring - ში გამოითვლება $x = \sum x_i t^i$ და y = Q(x, v_1, v_2, … , v_b) . ყოველი y_i - თვის და F_2[w_1, … , w_{4b}] გამოსახულებისთვის გამოითვლება $y_0 + y_1t + \dots + y_{3b-1}t^{3b-1}$ და (P_1, P_2, … , P_{2b}) T - ჯერ და შედეგად მიიღება (y_0, y_1, … , y_{3b-1}) ვექტორი.

ქვემოთ წარმოდგენილია ხელმოწერის პროცესი:

1. გამოთვლების დაწყება ხდება P_1, P_2, … , P_{2b} მნიშვნელობით, რის შემდეგაც ხდება

$T(y_0, y_1, \dots, y_{3b-1}) = (P_1, P_2, \dots, P_{2b})$ წრფივი განტოლების ამოხსნა, რომლის შედეგად მიიღება $y_0, y_1, \dots, y_{3b-1}$ მნიშვნელობები. $(y_0, y_1, \dots, y_{3b-1})$ ვექტორისთვის გვაქვს 2b ვარიანტი და შემთხვევითობის პრინციპით უნდა აირჩეს ერთ-ერთი.

2. შემდგომ ეტაპზე ხორციელდება $v_1, v_2, \dots, v_b \in F_2$ სიდიდეებისათვის შემთხვევითი მნიშვნელობების არჩევა და ამ მნიშვნელობების შეცვლა $Q(x, v_1, v_2, \dots, v_b)$ საიდუმლო პოლინომიალით.
3. შემდეგ დაითვლება $y = y_0 + y_1t + \dots + y_{3b-1} \in L$ და $Q(x) = y$ სიდიდეები, სადაც $x \in L$. თუ არსებობს რამდენიმე x ფესვი $Q(x) = y$ - დან, მაშინ შემთხვევითობის პრინციპით უნდა აირჩეს ერთ-ერთი. თუ ფესვი არ არსებობს, მაშინ ხელმოწერის პროცესი უნდა გაეშვას თავიდან.
4. ჩავწეროთ x , როგორც $x_0 + x_1t + \dots + x_{3b-1}t^{3b-1}$, სადაც $x_0, \dots, x_{3b-1} \in F_2$. ამის შემდეგ ხორციელდება $S(w_1, \dots, w_{4b}) = (x_0, \dots, x_{3b-1}, v_1, \dots, v_b)$ წრფივი განტოლების ამოხსნა, რის შედეგადაც მიიღება $(w_1, \dots, w_{4b})$ ხელმოწერა.

ეს არის HFE^{v-} კონსტრუქციის მაგალითი, რომელიც წარმოდგენილი იყო 1996 წელს პატარინის მიერ. [29]HFE ნიშნავს Hidden Field Equation “Q(x) = y” - ს, ხოლო “-” ნიშანი კი - ზოგი ბიტების გამოტოვებას. Q(x) = y არის 3b ბიტების ექვივალენტი, მაგრამ მხოლოდ 2b განტოლებების ასახვა ხდება. v ნიშნავს vinegar - ს, $v_1, v_2, \dots, v_b$ -ცვლადებს. სუფთა HFE გამოტოვებული ბიტებისა და vinegar ცვლადების გარეშე არის ძალიან სუსტი Grobner-basis შეტევის მიმართ, რომლის გამოცნობა ხორციელდება $2^{(lglg\,b)^2}$ ოპერაციაში, მაგრამ HFE^{v-} უკვე 10 წელზე მეტია მტკიცედ იგერიებს კიბერ შეტევებს.

1.7 ალგორითმის ეფექტურობა, კონფიდენციალურობა და გამოყენებადობა

ეფექტურობა

Elliptic-curve ხელმოწერის სისტემები $O(b)$ ბიტიანი ხელმოწერებითა და $O(b)$ ბიტიანი გასაღებებით, უზრუნველყოფს b ბიტიან დაცვას კლასიკური კომპიუტერების შეტევების წინააღმდეგ [33, 34].

შეუძლია თუ არა პოსტ-კვანტური public-key ხელმოწერის სისტემებს მიაღწიოს წარმადობის იგივე დონეს, რაც აქვს კლასიკურ სისტემებს? ქვემოთ არის მოყვანილი ხელმოწერის სისტემის ორი მაგალითი, რომელიც არის ნელი: პირველი არის სისტემა, სადაც ხელმოწერის სიგრძე არის $b^{2+O(1)}$, ხოლო მეორე არის სისტემა, სადაც გასაღების სიგრძე $b^{3+O(1)}$.

არაეფექტური კრიპტოგრაფია ზოგი მომხმარებლისთვის არის უპირატესობა, ხოლო ინტერნეტ სერვერისთვის, რომელიც ყოველ წამში ამუშავებს ათობით ათას მოთხოვნას, არის დაუცველი. Google - ს აქვს ზოგი საიტი კრიპტოგრაფიულად დაცული, მაგრამ როგორც ჩანს, არ შეუძლია დაიცვას მათგან ყველაზე გამოყენებადი. თუ Google - ს უკვე აქვს პრობლემები დღევანდელი კრიპტოგრაფიული სისტემების სიჩქარესთან, მაშინ მას აუცილებლად ექნება არანაკლები პრობლემა პოსტ-კვანტური კრიპტოგრაფიული სისტემების სიჩქარესთანაც.

დროის და ადგილის შეზღუდვები კრიპტოგრაფებს ყოველთვის შეუქმნიან გამოწვევებს და ასევე გააგრძელებენ გამოწვევების შექმნას პოსტ-კვანტური კრიპტოგრაფებისთვის. გამოწვევების დადებითი მხარე არის სისტემების სისწრაფის აწევა. ანალოგიური სიტუაცია იქნება პოსტ-კვანტურ კრიპტოგრაფიაში.

Merkle - ს ჰეშ ხის public-key ხელმოწერის სისტემა და McEliece's hidden-Goppa-code დაშიფრვის სისტემები წარმოდგენილი იყო 30 წლის წინ და უამრავი კრიპტოანალიზის მიუხედავად, დღემდე პრაქტიკულად რჩება უვნებელი. სხვა დანარჩენი ჰეშზე და კოდზე დაფუძნებული კრიპტოგრაფიული სისტემები არის ბევრად ახალი, ვიდრე ზემოთ აღნიშნული ორი სისტემა; მრავალრიცხოვანი კვადრატული კრიპტოგრაფია და lattice based კრიპტოგრაფია წარმოგვიდგენს უფრო ახალ პოსტ-კვანტურ სისტემებს. თავისთავად ახალი სისტემა შესაძლოა გატეხილ იქნას კრიპტოანალიზის შემდეგ. [87] ნაშრომში არის განხილული პოსტ-კვანტური პერიოდი და შესაბამისად ძველი კრიპტოსისტემების სისუსტეები. ალტერნატივად განხილულია lattice-ზე დაფუძნებული სისტემები. ეს სისტემები ცნობილია უსაფრთხოების მაღალი დონით.

სისტემის არჩევის დროს შეიძლება აქცენტი გაკეთებული იყოს კლასიკურ სისტემებზე, რომლებმაც წლებს გაუძლო, მაგრამ ხშირად მომხმარებლებს არ აქვთ კლასიკური სისტემების გამოყენების საშუალება და იძულებულები არიან ახალი, უფრო პატარა და სწრაფი სისტემები გამოიყენონ.

იმისათვის, რომ მომხმარებელი დარწმუნებული იყოს ახალ სისტემებში, შემქმნელებმა უნდა გამოყონ დრო და მოიძიონ ინფორმაცია განხორციელებულ შეტევებზე.

RSA public-key კრიპტოსისტემა შეიქმნა trapdoor one-way function - ის საფუძველზე. სამწუხაროდ, დღესდღეობით არ შეიძლება უბრალოდ trapdoor one-way function - ის გამოყენება. თანამედროვე RSA - ით დაშიფრვის დროს არ ხდება მხოლოდ რიცხვის კუბში აყვანა, თავიდან ხდება შემთხვევითი სტრიქონის არჩევა და შემდეგ ხდება შეტყობინების ზომის გაზრდა ამ სტრიქონის საშუალებით. გარდა ამისა, იმისათვის, რომ მოხდეს დიდი შეტყობინების ეფექტურად დამუშავება, სისტემა შიფრავს შემთხვევითად არჩეულ მოკლე სტრიქონებს და მათ იყენებს სიმეტრიული შიფრის გასაღებად, თავდაპირველი შეტყობინების იდენტიფიცირებისათვის.

გარდა იმისა, რომ დაშიფრვის უსაფრთხო სისტემა არის განსაზღვრული და სტანდარტად მიღებული, საჭიროა მისი პროგრამული და აპარატურული იმპლემენტაცია იმისათვის, რომ მოხდეს მისი ფართო გავრცელება და ინტეგრაცია. იმპლემენტატორები უნდა იყვნენ ყურადღებით, რადგან მარტო სწრაფი და სწორი სისტემის მიღება არ არის საკმარისი. მათ თავიდან უნდა აირიდონ სინქრონიზაციის დროს სხვადასხვა ტიპის side-channel გაჟონვები. წლების წინ RSA - ის და AES - ის რამდენიმე იმპლემენტაცია იქნა გატეხილი cache-timing შეტევების საშუალებით. Intel - ის მხრიდან ამ პრობლემის ნაწილობრივი გადაწყვეტა იყო AES - ის ინსტრუქციების დამატება მათი მომავალი თაობის პროცესორებში [35-41].

აღწერილი შეტევები ძირითადად შეეხება AES-128 და AES-192 ალგორითმების გამოყენებას. AES ალგორითმი და მასზე timing cache შეტევა განხილულია [3-6, 8, 9] სტატიებში, სადაც ავტორებს შესწავლილი აქვთ კონკრეტული ექსპერიმენტები სერვერზე, რომლებზეც გაშვებულია მრავალ ნაკადიანი ამოცანები და სისტემა იმყოფება დატვირთვის ქვეშ, როგორ ხდება კრიპტო გასაღების მიღება. [20] აღნიშნულ წყაროში ავტორების მიერ არის აღწერილი ორი შეტევა AES - ზე. პირველი არის plain text შეტევის გაუმჯობესებული ვერსია, რომელიც იყო წარდგენილი 2006 წელს, ხოლო მეორე არის ახალი known plaintext attack შეტევა, რომელსაც შეუძლია 128 ბიტიანი გასაღების აღდგენა.

ასევე გვხვდება ისეთი შეტევები AES - ზე სადაც ავტორებს ჰქონდათ ტექნიკური სირთულეები. [22-24] ამ სტატიებში არის განხილული AES - ზე განხორციელებული სხვადასხვა შეტევა: access-driven, time-driven, cache timing. მიუხედავად იმისა, რომ ავტორებს ჰქონდათ ტექნიკური სირთულეები, მათ დეტალურად აღწერეს აღნიშნული

შეტევები ყველასთვის გასაგებ ენაზე. ყველა აღწერილი შეტევა განხორციელებულ იქნა ერთ ნაკადში (thread - ში). ასევე ავტორების მიერ არის განხილული cache based side-channel შეტევა AES ალგორითმის ერთ-ერთ ფართოდ გავრცელებულ იმპლემენტაციაზე. აქვე არის დათვლილი ამ შეტევის განხორციელებისათვის საჭირო მანქანური რესურსები [70]. ავტორების მიერ Side-channel შეტევა ასევე კარგად არის განხილული ერთ-ერთ ნაშრომში [30]. ნაშრომში მოცემულია DES ალგორითმზე განხორციელებული შეტევების შედეგები, რომელიც არის დაფუძნებული CPU - ს side-channel - ის დაგვიანების ინფორმაციაზე. დეტალურად არის აღწერილი access-driven და cache-attacks შეტევები. სტატიაში მოყვანილი მაგალითებიდან ჩანს, რომ დღეს არსებული სტანდარტული პროცესორების უმრავლესობა განიცდის cache-based - სადმი დაუცველობას [65, 66].

ხშირია შეტევები პროგრამულ იმპლემენტაციაზე. მაგალითად, [68, 69] სტატიებში ავტორების მიერ არის აღწერილი რამდენიმე novel timing " შეტევა AES - ის table-driven პროგრამული იმპლემენტაციის წინააღმდეგ. შეტევები იყო განხორციელებული OpenSSL v. 0.9.8.(a)" წინააღმდეგ, რომელიც მუშაობდა კომპიუტერებზე შემდეგი პროცესორებით: Pentium III, Pentium IV Xeon . ყველაზე მძლავრი შეტევა იყო 128 ბიტიანი AES - ის გასაღების აღდგენა 2^{13} timing samples - ებით.

დასკვნა

არსებობს მრავალგანზომილებიანი კვადრატული განტოლებების აგების ბევრი სხვადასხვა გზა , public-key სისტემები და მრავალი საინტერესო იდეა, რომელიც ზოგავს დროს, ადგილს და აჩენს ბევრ ვარიანტს პოსტ-კვანტური კრიპტოგრაფიისთვის. გასაკვირი არ არის, რომ რამდენიმე ვარიანტი უკვე გატეხილია. Dubois, Fouque, Shamir and Stern - ის ერთ-ერთი ბოლო სტატიის თანახმად, ძალიან გამარტივებული სისტემის გატეხვის შემდეგ, სადაც არ იყო vinegar ცვლადები და Q - ში იყო მხოლოდ ერთი არანულოვანი პირობა, მივიდნენ იმ დასკვნამდე, რომ ყველა მრავალგანზომილებიანი კვადრატული განტოლებები არის საშიში:

მრავალგანზომილებიანი კვადრატული განტოლებები არის ძალიან ეფექტური, მაგრამ აქვთ გამოყენებადი მათემატიკური სტრუქტურა და მათი დაცვა არ არის ბოლომდე გაგებული. შესაბამისად, მათ მიმართ ახალი შეტევები მარტივად იძებნება. ამგვარად, მიზანშეწონილი იქნება დაცვის დროს კრიტიკულ აპლიკაციებში მათი არგამოყენება [27-32].

სავარაუდოდ, იგივე ავტორების რეკომენდაცია იქნება პრე-კვანტური ეპოქაში 4096 ბიტიანი RSA - ის თავიდან არიდება, რადგან 512 ბიტიანი RSA ალგორითმი უკვე კომპრომეტირებულია [7, 10, 11]. ავტორების რეკომენდაციაა ელიფსური მრუდეების და 256 ბიტიანი AES ალგორითმების გამოყენების თავიდან არიდება.

პროცესორების განვითარება წინ მიდის, გამოდის უფრო ახალი, ეფექტური და მრავალნაკადიანი პროცესორები. არსებობს ბევრი, ჰეშზე დაფუძნებული ალგორითმი, რომელიც არის მდგრადი კვანტური კომპიუტერების შეტევებისგან, მაგრამ არის არაეფექტური.

ნაშრომში ჩატარებული კვლევის მიზანი არის ისეთი სისტემის წარმოდგენა, რომელიც იქნება მდგრადი კვანტური კომპიუტერების შეტევებისგან და ამავდროულად საკმაოდ ეფექტური.

ნაშრომში რეალიზებულია შემდეგი ამოცანები:

6. არსებული სისტემების განხილვა.
7. კლასიკური კრიპტოგრაფიის პოსტ-კვანტურთან შედარება.
8. სწრაფი და ამავდროულად უსაფრთხო სისტემის შექმნა.

9. ახალი სისტემის შესრულების დროის დათვლა და ძველი სისტემის შესრულების დროსთან შედარება.
10. ახალი სისტემის ოპერაციების რაოდენობის შემცირება და ასევე ძველ სისტემასთან შედარება.

ნაშრომის სამეცნიერო სიახლეს წარმოადგენს დინამიური ალგორითმი, რომელიც იქნება სწრაფი და ამავდროულად უსაფრთხო. ამ ალგორითმში ოპერაციების რაოდენობა იქნება ბევრად ნაკლები ვიდრე არსებულ სისტემაში. ამასთან ოპერაციების რაოდენობის სიმცირე და ალგორითმის სისწრაფე იქნება დამოკიდებული პროცესორის ნაკადების (thread - ების) რაოდენობაზე.

თავი 2 - კრიპტოგრაფიის მეთოდები და ხელმოწერის სიტემები

2.1 შედარება კვანტურ კრიპტოგრაფიასთან

კვანტურ კრიპტოგრაფიას, რომელსაც ასევე ეძახიან კვანტური გასაღებების განაწილებას (quantum key distribution) , შეუძლია საერთო გასაღების ეფექტურ უსასრულო ნაკადამდე გაფართოება. კვანტური კრიპტოგრაფიის აუცილებელი პირობა არის ის, რომ ორი სხვადასხვა Alice და Bob მომხმარებლებისთვის იყოს ცნობილი 256 ბიტიანი secret key. კვანტური კრიპტოგრაფიის შედეგი არის ის, რაც Alice - თვის და Bob - თვის არის ცნობილი 10^{12} ბიტიანი საიდუმლო გასაღები, რომელიც გამოყენებული იქნება შეტყობინებების დასაშიფრად. კრიპტაციის შედეგად, მიღებული ნაკადის ზომა გაიზრდება წრფივად Alice - ის და Bob - ის კვანტურ კრიპტოგრაფიაზე დახარჯული დროსთან მიმართებაში [42-44].

კვანტური კრიპტოგრაფიის დეტალები შეიძლება დახასიათდეს შემდეგნაირად:

1. ნაკადური შიფრი მიღებულ ნაკადს აგენერირებს მათემატიკური ფუნქციის საშუალებით. კვანტური კრიპტოგრაფია Alice - თვის იყენებს ფიზიკურ საშუალებებს, რათა უწყვეტად მოხდეს საიდუმლო გასაღების გენერირება და Bob - თვის გადაცემა.
2. ნაკადური შიფრი შეიძლება გამოყენებულ იქნას ისეთი ინფორმაციის დასაცავად, რომელიც არის მიღებული არასანდო ან სხვა ტიპის ქსელიდან. კვანტური კრიპტოგრაფია ითხოვს პირდაპირ ოპტიკურ-ბოჭკოვან კავშირს Alice - ის სანდო კვანტურ მოწყობილობასა და Bob - ის სანდო კვანტურ მოწყობილობას შორის.
3. ნაკადური შიფრის უსაფრთხოება იზომება ვარაუდით - არავის არ უნახავს შეტევები. შესაბამისად, ვვარაუდობთ, რომ ასეთი შეტევები არც არსებობს . თუ კვანტური კრიპტოგრაფია არის იდეალურად იმპლემენტირებული, მისი უსაფრთხოება იზომება კვანტური მექანიკის კანონებიდან გამომდინარე.
4. თანამედროვე ნაკადურ შიფრს შეუძლია იმუშაოს ნებისმიერ, დღესდღეობით ხელმისაწვდომ, დაბალი სიხშირის მქონე პროცესორზე და დააგენერიროს გიგაბაიტობით ნაკადი წამში. კვანტურ კრიპტოგრაფიას შეუძლია დააგენერიროს კილობაიტობით ნაკადი წამში 50000$ მოწყობილობების დახმარებით.

შეიძლება არგუმენტირებულად ვამტკიცოთ, რომ კვანტური კრიპტოგრაფია არის locked-briefcase cryptography და მას ვერ გატეხავენ კვანტური კომპიუტერები. ამასთან აღსანიშნავია, რომ კლასიკური კრიპტოგრაფია ზოგადად არის განსხვავებული კვანტური კრიპტოგრაფიისგან შემდეგი მახასიათებლებით:

1. სხვა კრიპტოგრაფიის ალგორითმების მსგავსად, კლასიკური კრიპტოგრაფიიის ალგორითმები მოიცავს ბევრ განსხვავებულ თემატიკას, რომელთა შორისაა საიდუმლო და ღია გასაღებებით შესრულებული შიფრაცია/დეშიფრაციის ოპერაციები.

2. კლასიკური კრიპტოგრაფია მოიცავს ისეთ ალგორითმებს, რომლებიც სავარაუდოდ არის უსაფრთხო კვანტური კომპიუტერების შეტევების მიმართ, მაგრამ ასევე მოიცავს ბევრ lower-cost სისტემას, რომლებიც ასევე სავარაუდოდ უსაფრთხოა. მათი გატეხვა და მათ უსაფრთხოებაზე დაზუსტებით საუბარი შეგვეძლება მხოლოდ მაშინ, როდესაც დაიწყება კვანტური კომპიუტერების გამოყენება.

2.2 კლასიკური კრიპტოგრაფია და კვანტური გამოთვლები

კვანტური გამოთვლები სულ უფრო და უფრო მეტ საფრთხეს უქმნის კლასიკურ სისტემებს. ამის ყველაზე თვალნათელი მაგალითები არის ეფექტური კვანტური ალგორითმები, რომლებიც ახდენენ დღეისათვის არსებული კრიპტოსისტემების კომპრომეტირებას

1994 წელს შორმა აღმოაჩინა კვანტური ფაქტორიზაციის ალგორითმები, რომლებიც კვანტური კომპიუტერების საშუალებით, შეიძლება გამოყენებული იქნას RSA კრიპტოსისტემის და Diffie-Hellman ალგორითმის გასატეხად.

კლასიკური კრიპტოგრაფია მოიცავს ისეთ ინსტრუმენტებს, როგორებიცაა: დაშიფრვა, გასაღების გადაცემა, ციფრული ხელმოწერები, ფსევდო-შემთხვევითი რიცხვების გენერატორი და ცალმხრივი ფუნქცია(one-way functions). არსებობს კონტრაქტების ხელმოწერის, ელექტრონული ხმის მიცემისა და უსაფრთხო დაშიფრვის აპლიკაციები. გამოდის, რომ ამ სისტემებს შეუძლიათ არსებობა, თუ არსებობს გამოთვლის სირთულე, რომელზეც იწყობა მსგავსი სისტემები. მაგალითად, RSA არის უსაფრთხო იმ შემთხვევაში, თუ ფაქტორიზაცია არის რთული კლასიკური კომპიუტერისათვის. თუმცა სირთულის თეორია(complexity theory) არ გვაძლევს ეფექტური ალგორითმის არარსებობის ინფორმაციას. დასკვნები, თუ რომელი ამოცანა არის რთულად ამოსახსნელი, დამყარებულია ჰიპოთეზებზე. ისეთი ალგორითმის პოვნა, რომელიც კვანტურ კომპიუტერს სისტემის გატეხვას გაურთულებს, არის ძალიან რთული. იმისათვის, რომ გავიგოთ რომელი ამოცანა არის კვანტური კომპიუტერისათვის რთულად ამოსახსნელი, უნდა ჩატარდეს ახალი და გაგრძელდეს არსებული კვლევები.

კრიპტოგრაფიული სისტემის დაპროექტება არის რთული ამოცანა. მიზანი არის შეიქმნას ისეთი სისტემა, რომელიც აკმაყოფილებს უსაფრთხოების ყველა სტანდარტს მიუხედავად იმისა, თუ რა მიზნებისთვის გამოიყენებს მას ჰაკერი. თანამედროვე კრიპტოგრაფია არის ფოკუსირებული მყარი ფუნდამენტის შექმნაზე, რათა მიღწეულ იქნას ზემოთ აღნიშნული მიზანი. სისტემის შექმნის დროს ერთადერთი ვარაუდი ჰაკერის შესახებ არის მისი გამოთვლითი შესაძლებლობა. სავარაუდოდ, ჰაკერს აქვს კლასიკური კომპიუტერი და არის შეზღუდული დაგენერირების დროში, მაგრამ თუ ჰაკერს აქვს კვანტური კომპიუტერი, მაშინ კლასიკური სისტემებიდან რთული მისახვედრია თუ, რომელია სუსტი და რომელი უსაფრთხო. კვანტური გამოთვლები იყენებს წესებს,

რომლებიც არის ახალი და უჩვეულო. მაგალითისათვის ჩვენ შეგვიძლია განვიხილოთ კვანტური Fourier - ის ტრანსფორმაცია, რაც კვანტურ კომპიუტერზე უფრო სწრაფად შესრულდება, ვიდრე კლასიკურ კომპიუტერზე. თუმცა ეს ყველაფერი მოითხოვს დიდ გამომთვლელ რესურსებს. მეცნიერების მიერ შემოთავაზებულია Fourier სწრაფი გარდაქმნა (FFT), რომელიც დანარჩენ ალგორითმებზე ბევრად სწრაფია. გარდაქმნის აღწერისას მოყვანილია ალგორითმის დეტალური განხილვა და შესაბამისი მაგალითები, რაც ამარტივებს ალგორითმის გამოყენებას [74-76]. Fourier - ის ტრანსფორმაციის მეთოდში ინფორმაციის შეყვანა და იქიდან შედეგის გამოტანა არის ძალიან შეზღუდული. აქედან გამომდინარე, კვანტური ალგორითმების შექმნა დაფუძნებულია დამატებით ძალებსა და ზოგ მნიშვნელოვან ასპექტს შორის კომპრომისის მოძებნაზე. დღესდღეისობით მთავარ გამოწვევას წარმოადგენს ის თუ, როგორი იქმნება ახალი კლასიკური კრიპტოსისტემები, რომლებიც იქნება უსაფრთხო და გაუძლებს კვანტური კომპიუტერების შეტევებს. ასეთი სისტემების შექმნა სამომავლოდ იძენს დიდ მნიშვნელობას. მათი იმპლემენტაცია ახლა არის შესაძლებელი, მაგრამ მათი მდგრადობა კვანტური სისტემების მიმართ წარმოადგენს შესწავლის საგანს. ქვემოთ მოყვანილ ცხრილში, ნაჩვენებია რამდენიმე კრიპტოსისტემის სტატუსი [45-47].

კრიპტოსისტემა	არის თუ არა გატეხილი კვანტური კომპიუტერის მიერ?
RSA public key encryption	გატეხილია
Diffie-Hellman key-exchange	გატეხილია
Elliptic curve cryptography	გატეხილია
Buchmann-Williams key-exchange	გატეხილია
Algebraically Homomorphic	გატეხილია
McEliece public key encryption	არ არის გატეხილი
NTRU public key encryption	არ არის გატეხილი
Lattice-based public key encryption	არ არის გატეხილი

ცხრილი 1. კვანტურ კომპიუტერებთან მიმართებაში დღევანდელი კლასიკური სისტემების სტატუსი.

კრიპტოსისტემები, რომლებიც დღესდღეობით არის გამოყენებაში, შეიძლება გატეხილი იყოს კვანტური კომპიუტერების დახმარებით. შესასწავლია თუ რა საკითხებთან არის დაკავშირებული კვანტურ სამყაროში უსაფრთხო სისტემებზე გადასვლა და რატომ არ

ხდება მათზე გადასვლა. პირველ რიგში, უნდა მოხდეს სისტემის უმტკივნეულო შეცვლა, რათა არ მოხდეს იმ სისტემების/პროგრამების გაჩერება, რომელთა გამართული მუშაობა დამოკიდებული არის კრიპტოსისტემებზე. ალტერნატიული კრიპტოსისტემები კი, როგორებიცაა lattice-based ან McEliece - ის სისტემა, პრაქტიკაში არაეფექტურია. მეორე პირობა არის ის, რომ ჩატარებული კვლევების შემდეგ, მყარი არგუმენტებით უნდა იყოს დასაბუთებული, რომ ახალი სისტემა არ იქნება გატეხილი კვანტური კომპიუტერის მიერ. ყველა ამ მოთხოვნას კი სისტემა დააკმაყოფილებს მხოლოდ მასზე ჩატარებული საფუძვლიანი კვლევების შემდეგ.

იმისათვის, რომ სისტემები იყოს კონკურენტუნარიანი, მუშავდება სპეციალური (unit) ტესტები, რომელთა მეშვეობითაც ხდება სისტემის სუსტი და ძლიერი მხარეების გამოვლენა.

2.3 კვანტური კომპიუტერების მიმართ სუსტი კრიპტოსისტემები

კლასიკური კრიპტოგრაფიის ეფექტურობა მცირდება კვანტური კომპიუტერების მხრიდან მომდინარე საფრთხის გამო და საჭირო ხდება პოსტ-კვანტური კრიპტოგრაფიული ალგორითმების განვითარება. უსადენო კავშირის (wireless) უსაფრთხოების მიმოხილვისას ავტორების მიერ წარმოდგენილია elliptic curve-ის უპირატესობა RSA ალგორითმთან მიმართებაში. ასევე განხილულია სისტემა, რომელიც ეფუძნება Clipped Hopfield - ის ნეირონულ ქსელს [60-62]. [44-46] სტატიებში წარმოდგენილია public-key სისტემის მიმოხილვა, რომელიც არის დაფუძნებული კოდების შესწორებაზე - error-correcting და განხილულია შეტევა კრიპტოგრაფიის ალგორითმზეა. აღნიშნულ სტატიებში მოყვანილი რამდენიმე მოდელი, რომელიც შესაძლოა გამოყენებული იქნას public-key encryption სქემის შექმნაში. [56-58] წყაროში წარმოდგენილია Multivariate Public Key - ის ძირითადი კონცეფცია და ასევე მოყვანილია რამდენიმე პრაქტიკული მაგალითი Multivariate Public Key-ის გამოყენების.

კლასიკური კრიპტოგრაფია ასევე გამოიყენება ინტერნეტ ტრაფიკის დასაცავად. მისი უსაფრთხოება ეყრდნობა სხვადასხვა ალგორითმებს. როგორც ჩანს, აღნიშნული ალგორითმების ამოხსნის სისწრაფე ბევრად გაიზარდა კვანტურ კომპიუტერებში. ასეთი ალგორითმების მაგალითები არის ფაქტორიზაცია და Pell - ის განტოლება. ამ ალგორითმების არსებობა ნიშნავს იმას, რომ კვანტურ კომპიუტერებს შეუძლია გატეხოს RSA , Diffie-Hellman და eliptic curve cryptography ალგორითმები, რომლებიც დღეს აქტიურად გამოიყენება. ასევე გვხდება Buchmann-Williams გასაღების გაცვლის პროტოკოლი, რომელიც მიიჩნევა ბევრად უსაფრთხოდ. დღესდღეობით, კრიპტოგრაფიაში ერთ-ერთი უმთავრესი კითხვა არის, თუ რომელი კრიპტოსისტემა იქნება უსაფრთხო კვანტური კომპიუტერების მიმართ.

მაგალითად, ფაქტორიზაცია არის პრობლემა, რომლის შესწავლა უკვე დიდი ხანია მიმდინარეობს. ასევე ცნობილია ფაქტორიზაციის რამდენიმე ალგორითმი: Lehman's მეთოდი, Pollard's ρ მეთოდი და Shanks's class group მეთოდი. ამ ამოცანამ წინ წამოიწია მაშინ, როდესაც გვიან 1970-იან წლებში მოხდა RSA public-key კრიპტოსისტემის შექმნა, ხოლო Shor - ის ფაქტორიზაციის ეფექტური კვანტური ალგორითმი, რომელიც კვანტურ კომპიუტერზე ახდენს RSA - ს ფაქტორიზაციას $(lg\ n)^{1+O(1)}$ მარტივი ოპერაციების გამოყენებით შეიქმნა 1994 წელს. [67] წყაროში აღწერილია პირველი შეტევა კრიპტო

სისტემის public გასაღების წინააღმდეგ phoenic side channel - ის გამოყენებით. ნაშრომში გაანალიზებულია გასაღების სიგრძე და მისი მდგრადობა ალგორითმზე განხორციელებული შეტევის დროს, როგორც გაირკვა 2048 ბიტიან გასაღების დეშიფრაციის დროს 9%-ით მეტი ოპერაციის შესრულება არის საჭირო, ვიდრე 1024 ბიტიან გასაღების დროს.

კვანტური ალგორითმების ექსპონენციურად აჩქარების ამოცანა აღმოჩნდა ძალიან რთული. Shor - ის ალგორითმის გამოსვლიდან 8 წლის შემდეგ წარმოდგენილი იყო Pell - ის განტოლებისთვის კვანტური ალგორითმი. მოცემული d დადებითი მთელი რიცხვის გათვალისწინებით Pell - ის განტოლებას აქვს შემდეგი სახე: $x^2-dy^2 = 1$, ხოლო ალგორითმის მიზანია (x, y) წყვილის გამოთვლა. Pell - ის განტოლების ამოხსნა ისეთივე სირთულეს წარმოადგენს, როგორც ფაქტორიზაცია, ხოლო ამ განტოლების ამოხსნის ყველაზე ეფექტური ალგორითმი არის უფრო ნელი, ვიდრე ფაქტორიზაციის ყველაზე ეფექტური ალგორითმი. იმისათვის, რომ ეს რთული გამოთვლა ყოფილიყო გამოსადეგი, Buchmann-მა და Williams - მა დააფუძნეს მათ მიერ შემოთავაზებული გასაღების მიმოცვლის პროტოკოლი Pell - ის განტოლებაზე. მათი მიზანი იყო ისეთი უსაფრთხო სისტემის შექმნა, რომელიც დარჩებოდა უსაფრთხო ფაქტორიზაციის შემთხვევაში. თუმცა კვანტური ალგორითმი კვანტური კომპიუტერის გამოყენებით ახდენს Buchmann - Williams - ის სისტემის კომპრომეტირებას. გასაღების მიმოცვლის პროტოკოლები, რომლებიც დაფუძნებული არის Pell - ის განტოლებაზე, აღარ წარმოადგენენ უსაფრთხო სისტემებს კვანტური კომპიუტერებიდან მომდინარე საფრთხეების წინაშე.

კვლევები კვანტური ალგორითმების სფეროში ძირითადად ეხება hidden subgroup problem (HSP) - ს. (HSP) - ი წარმოადგენს ამოცანას, რომელიც ეხება კონკრეტულ ჯგუფს და სხვა დანარჩენი ამოცანის დაყვანა ხდება ამ ძირითად ამოცანამდე. Pell - ის განტოლება დაიყვანება (HSP) - მდე მაშინ, როდესაც ის ხდება დაუთვლელი. ასეთი შემთხვევებისთვის გვაქვს სხვადასხვა ეფექტური კვანტური ალგორითმი, რომლებიც ხსნის (HSP) - ის. გრაფის იზომორფიზმი სიმეტრიული ჯგუფისთვის დაიყვანება (HSP) - მდე და ასევე უნიკალური, უმოკლესი lattice ვექტორი არის დაკავშირებული (HSP) - თან მაშინ, როდესაც ჯგუფი არის ორწახნაგოვანი. ზემოთ აღწერილი ორი ჯგუფი არის nonabelian. ბოლო 10 წლის განმავლობაში ფოკუსირება ხდება abelian HSP - ის წარმატების გამეორებაზე. [77] ნაშრომში განხილულია თუ როგორ მუშაობს (HSP) ალგორითმი კვანტურ კომპიუტერებზე და ასევე, ნაჩვენებია, თუ როგორ შეიძლება აღიწეროს და გაანალიზდეს abelian - ის დამალული

ქვეჯგუფის ზოგადი პრობლემები. არსებობს იმის ვარაუდი, რომ Fourier - ის ალგორითმი მდგრადი იქნება კვანტური კომპიუტერების მიმართ [48-52].

[54] წყაროში არის მოყვანილი ასიმეტრიული კოდი, რომელიც არის მიჩნეული RSA - ს და ECC - ს ალტერნატივად. ასიმეტრიული კოდის მთავარი პრობლემა მდგომარეობს გასაღების სიგრძეში, რაც თავისთავად ხდის ამ სისტემას არაეფექტურს და ნელს. მისი მინიმალური ზომა იწყება 50 კილობაიტიდან, რომელიც შეიძლება გაიზარდოს რამდენიმე მეგაბაიტამდე.

2.4 სხვადასხვა კრიპტოგრაფიული პრიმიტივი

ფსევდო-შემთხვევითი რიცხვების გენერატორი არის ერთ-ერთი მნიშვნელოვანი საშუალება კრიპტოგრაფიაში. მისი საშუალებით ხდება მოკლე სტრიქონის ისეთნაირად გარდაქმნა გრძელ სტრიქონში, რომ ის არ იყოს მარტივად ამოსაცნობი. თუ კომპიუტერისთვის ის არ არის ადვილად ამოსაცნობი და პროგნოზირებადი, მაშინ ასეთ თანმიმდევრობას უწოდებენ ერთგვაროვანს. რადგან ზემოთ აღნიშნული განსაზღვრა გამომდინარეობს კლასიკური კომპიუტერების გამოთვლებისგან, უნდა მოხდეს პრიმიტივების გადახედვა და თავიდან შეფასება კვანტური კომპიუტერებისთვის.

ავტორების მიერ განხილულია პოსტ-კვანტური სისტემების სხვადასხვა ასპექტი, მათზე შეტევები. ასევე მათ მიერ არის შეთავაზებული Merkle - ს გაუმჯობესებული ვარიანტი. ამ ვარიანტში არის წარმოდგენილი ახალი ფსევდო შემთხვევითი რიცხვების გენერატორი (PRNG), რომელიც არის დაფუძნებული კვანტურ შემთხვევით wandering - ზე [17-19].

ავტორების მიერ ასევე შეთავაზებულია Merkle - ს გაუმჯობესებული ვარიანტი. ამ ვარიანტში არის წარმოდგენილი ახალი ნამდვილი შემთხვევითი რიცხვების გენერატორი (TRNG), რომელიც არის PRNG - ზე უსაფრთხო [93].

მეორე, არანაკლებ მნიშვნელოვანი კონცეფცია კრიპტოგრაფიაში არის zero-knowledge პროტოკოლი. ამ პროტოკოლის საშუალებით prover - ს შეუძლია დაუმტკიცოს verifier - ს, რომ მისთვის ცნობილია საიდუმლო ინფორმაცია ისე, რომ verifier - ს საშუალება არ ექნება გაეცნოს ამ საიდუმლო ინფორმაციას. პრაქტიკაში იყენებენ ამ პროტოკოლს იმისათვის, რომ ერთმა მხარემ დაუმტკიცოს თავისი ვინაობა მეორე მხარეს საიდუმლო ინფორმაციის საშუალებით. იმისათვის, რომ პროტოკოლი იყოს zero-knowledge , არანაირი ინფორმაცია არ უნდა გახდეს ცნობილი, მიუხედავად იმისა, თუ რა ტექნოლოგიას და სტრატეგიას იყენებს ე.წ. cheating veifier - ი. აქედან გამომდინარე, იბადება კითხვა: რა მოუვა კლასიკურ პროტოკოლებს, როდესაც cheating veifier - ი არის კვანტური კომპიუტერი?

ჯონ უოტრაუსმა წარმოადგინა ორი კარგად ცნობილი კლასიკური პროტოკოლი, რომლებიც არის zero-knowledge proof ალგორითმი, რომელიც უძლებს კვანტური კომპიუტერების შეტევას. უოტრაუსმა გვიჩვენა, რომ Goldreich-Micali-Wigderson graph isomorphism პროტოკოლი არის უსაფრთხო ისევე, როგორც the graph 3-coloring

პროტოკოლი, თუ გამოვიყენებთ კლასიკურ სქემებს ამ პროტოკოლის კვანტური კომპიუტერებისგან დასაცავად.

2.5 ჰეშზე დაფუძნებული ხელმოწერის სქემები

ციფრული ხელმოწერები ბოლო დროს იქცა დაცვის საკვანძო ტექნოლოგიად ინტერნეტისთვის და სხვა IT ინფრასტრუქტურებისთვის. ციფრული ხელმოწერები ფართოდ არის გავრცელებული იდენტიფიკაციის და აუტენტიფიკაციის პროტოკოლებში. ამიტომ, ხელმოწერების უსაფრთხო ალგორითმების არსებობა არის უმნიშვნელოვანესი IT უსაფრთხოებისათვის.

დღესდღეობით პრაქტიკაში გამოყენებადი ციფრული ხელმოწერის ალგორითმები არის: RSA , DSA და ECDSA . ისინი არ არის დაცული კვანტური კომპიუტერისგან იმიტომ, რომ მათი უსაფრთხოება ეყრდნობა მთელი რიცხვების ფაქტორიზაციას.

ჰეშზე დაფუძნებული ხელმოწერების სქემები გვთავაზობს საინტერესო ალტერნატივას. ისევე, როგორც სხვა ციფრული ხელმოწერის სქემა, ჰეშზე დაფუძნებული ხელმოწერის სქემაც იყენებს კრიპტოგრაფიულ ჰეშ ფუნქციებს. მათი უსაფრთხოება ემყარება კონკრეტული ჰეშ ფუნქციის უსაფრთხოებას. უსაფრთხო ჰეშ ფუნქციის არსებობა შეიძლება იყოს განხილული, როგორც მინიმალური მოთხოვნა უსაფრთხო ციფრული ხელმოწერის სქემისთვის, რომელსაც შეეძლება უამრავი დოკუმენტების ხელმოწერა ერთი საიდუმლო გასაღების გამოყენებით. ამ ხელმოწერის (სხვადასხვა ზომის მქონე სტრიქონი) დახმარებით ხდება დოკუმენტების გარდაქმნა ციფრულ ხელმოწერად (ფიქსირებული ზომის მქონე სტრიქონი). აქედან გამომდინარე, ციფრული ხელმოწერის ალგორითმები ფაქტიურად არის ჰეშ ფუნქციები. ეს ჰეშ ფუნქციები უნდა იყოს collision resistant, ანუ თუ შესაძლებელია ორი დოკუმენტის შექმნა იმავე ციფრული ხელმოწერით, მაშინ ხელმოწერის ეს სისტემა აღარ არის უსაფრთხო. [13] ავტორების მიერ არის წარმოდგენილი Merkle - ს ხის ალგორითმი, სადაც არ არის საჭირო collision resistant ჰეშ ფუნქციის გამოყენება. მათი გადაწყვეტილება ასევე არის მდგრადი როგორც birthday შეტევის მიმართ, ასევე სხვადასხვა collision ალგორითმების მიმართ.

ჰეშზე დაფუძნებული ხელმოწერის სქემები არის მთავარი და ყველაზე მნიშვნელოვანი პოსტ-კვანტური ხელმოწერის კანდიდატები. თუმცა დღემდე არ არსებობს მტკიცებულება მათი უსაფრთხოებისა კვანტური კომპიუტერების წინააღმდეგ. მათი უსაფრთხოების მოთხოვნები არის ძალიან მინიმალური. ასევე ყოველი ახალი ჰეშ ფუნქცია გვაძლევს ახალ ჰეშზე დაფუძნებულ ხელმოწერის სქემას. აქედან გამომდინარე, უსაფრთხო ხელმოწერის სქემის შექმნა არ არის დამოკიდებული რთულ თეორიებზე. ფუძემდებელი

ჰეშ ფუნქციის არჩევა ხდება ტექნიკური და პროგრამული შესაძლებლობებიდან გამომდინარე.

ჰეშზე დაფუძნებული ხელმოწერის სქემები Ralph Merkle - ს მიერ იყო წარმოდგენილი. Merkle - მ თავიდან დაიწყო ერთჯერადი ხელმოწერის სქემებით. კერძოდ, Lamport and Diffie [53, 54]. ზოგადად, ერთჯერადი ხელმოწერის სტრუქტურა არის ძალიან ფუნდამენტური. მისი სტრუქტურა მოითხოვს ერთმიმართულებიან ფუნქციას. Rompel - ის მიხედვით ერთმიმართულებიანი ფუნქციები საჭიროა უსაფრთხო ციფრული ხელმოწერებისთვის. აქედან გამომდინარე, ერთჯერადი ხელმოწერის სქემები არის ციფრული ხელმოწერის სქემების ფუნდამენტი. თუმცა, აქვს რამდენიმე ნაკლი. ერთი key-pair , რომელიც შედგება secret signature key და public verification key - სგან, შეიძლება გამოყენებული იყოს მხოლოდ ერთი დოკუმენტის ხელმოსაწერად. ზოგი აპლიკაციისთვის ასეთი რამ არის მიუღებელი. Merkle-ს იდეა იყო ჰეშ ხე, რომელიც აგვარებდა ბევრი ერთჯერადი დადასტურებისთვის განკუთვნილი გასაღებების (ჰეშ ხის leaves - ები) პრობლემას. მათ ნაცვლად გამოიყენება ერთი public key, რომელიც არის ამ ხის ბოლო წერტილი(root). თავიდან Merkle - ს იდეა იყო არაეფექტური RSA ხელმოწერის სქემასთან შედარებით. თუმცა, შემდგომში მოხდა მისი გაუმჯობესება. დღესდღეობით ჰეშზე დაფუძნებული ხელმოწერები არის RSA ალტერნატივა [55-58].

ავტორების მიერ არის წარმოდგენილი CMSS . ავტორები აღნიშნავენ, რომ Merkle-ს ხის ალგორითმი არის RSA, DSA და ECDSA - ს ალტერნატივა და ასევე იმასაც, რომ ის არის მდგრადი კვანტური კომპიუტერების შეტევების მიმართ. ამ შემთხვევაში მათ შემოგვთავაზეს Merkle - ს ხის რეალიზაცია Java - ზე, სადაც აქვთ გამოყენებული Java Cryptographic Service Provider FlexiProvider [14, 16]. ასევე განხილულია CMSS-ი MSS -ის (Merkle signature scheme) ვარიანტი, შემცირებული private გასაღების ზომით, გასაღებების წყვილების გენერაციის დროით და ხელმოწერების შექმნის დროით. CMSS-ი პრაქტიკაში კონკურენტუნარიანია, Java– ს კრიპტოგრაფიული სერვისის პროვაიდერის FlexiProvider– ის მაღალეფექტური განხორციელების წარდგენით [88-89].

2.6 Lamport-Diffie ერთჯერადი ხელმოწერის სქემა

განვიხილოთ Lamport-Diffie ერთჯერადი ხელმოწერის სქემა (LD-OTS) . დავუშვათ, n არის დადებითი რიცხვი, რომელიც არის LD-OTS security პარამეტრი. LD-OTS იყენებს ერთმიმართულებიან (one-way) ფუნქციას

$f : \{0,1\}^n \rightarrow \{0,1\}^n$,

და კრიპტოგრაფიულ ჰეშ ფუნქციას

$$g : \{0,1\}^n \rightarrow \{0,1\}^n.$$

LD-OTS - ის გასაღებების წყვილის გენერაცია. LD-OTS - ის ხელმოწერის X გასაღები შედგება 2n ბიტიანი სტრიქონებისგან, რომლებიც არის უნიკალური და შემთხვევითად შერჩეული.

$X = (x_{n-1}[0], x_{n-1}[1], \dots, x_1[0], x_1[1], x_0[0], x_0[1]) \in_R \{0,1\}^{(n,2n)}$. (1)

LD-OTS დადასტურების Y გასაღები არის

$Y = (y_{n-1}[0], y_{n-1}[1], \dots, y_1[0], y_1[1], y_0[0], y_0[1]) \in \{0,1\}^{(n,2n)}$, (2)

სადაც

$y_i[j] = f(x_i[j]), 0 \leq i \leq n-1, j = 0, 1$. (3)

აქედან გამომდინარე, გასაღების გენერაციისთვის საჭიროა f - ის 2n ხარისხში აყვანა. ხელმოწერის და დადასტურების გასაღებების სიგრძე არის 2n ბიტი.

LD-OTS - ის ხელმოწერის გენერაცია. " $M \in \{0,1\}^*$ " დოკუმენტის ხელმოწერა ხდება LD-OTS - ით X ხელმოწერის გასაღების გამოყენებით, ისე როგორც არის (1) ნაჩვენები განტოლებაში. დავუშვათ, $g(M) = d = (d_{n-1}, \dots, d_0)$ არის M შეტყობინების კრებული, მაშინ LD-OTS - ის ხელმოწერა არის

$\sigma = (x_{n-1}[d_{n-1}], \dots, x_1[d_1], x_0[d_0]) \in \{0,1\}^{(n,n)}$ (4)

ხელმოწერა არის n ბიტიანი სტრიქონების თანმიმდევრობა, სადაც ყოველი სტრიქონის სიგრძე არის n - ის ტოლი. ისინი არჩეულია d შეტყობინების კრებულიდან. ხელმოწერაში სტრიქონის i - ური ბიტი არის $x_i[0]$ თუ i - ური ბიტი d - ში არის 0, ხოლო სხვა შემთხვევაში არის $x_i[1]$. ხელმოწერა არ მოითხოვს f-ის რაიმე ხარისხში აყვანას. ხელმოწერის სიგრძე არის n^2 .

LD-OTS - ის ვერიფიკაცია. იმისათვის, რომ მოხდეს M - ის $\sigma = (\sigma_{n-1}, \dots, \sigma_0)$ ხელმოწერის ვერიფიკაცია, როგორც არის ნაჩვენები (4) - ში, საჭიროა შეტყობინების $d = (d_{n-1}, \dots, d_0)$ კრებულის დათვლა. შემდეგ ხდება გასაღების შემოწმება.

$$\big(f(\sigma_{n-1}), \dots, f(\sigma_0)\big) = (y_{n-1}[d_{n-1}], \dots, y_0[d_0]) \qquad (5)$$

ხელმოწერის დადასტურება მოითხოვს f - ის n ხარისხში აყვანას.

მაგალითი 1. დავუშვათ, n = 3, $f : \{0,1\}^3 \to \{0,1\}^3$, $x \to x + 1 \; mod \; 8$ და d = (1,0,1) არის M შეტყობინების ჰეშ მნიშვნელობა. ვირჩევთ ხელმოწერის გასაღებს

$$X = (x_2[0], x_2[1], x_1[0], x_1[1], x_0[0], x_0[1]) = (1\,0 \;\; 1\,0 \;\; 1\,0 \;\; 0\,1 \;\; 1\,1 \;\; 1\,0 \;\; 1\,0 \;\; 0\,1 \;\; 1\,0) \in \{0,1\}^{(3,6)},$$

შემდეგ ხდება verification key - ს გამოთვლა

$$Y = (y_2[0], y_2[1], y_1[0], y_1[1], y_0[0], y_0[1]) = (0\,0 \;\; 0\,0 \;\; 0\,1 \;\; 1\,1 \;\; 0\,1 \;\; 0\,1 \;\; 1\,0 \;\; 1\,1 \;\; 0\,1) \in \{0,1\}^{(3,6)}.$$

d = (1, 0, 1) - ის ხელმოწერა არის

$$\sigma = (\sigma_2, \sigma_1, \sigma_0) = (x_2[1], x_1[0], x_0[1]) = (0\,0\,0\,0\,1\,1\,0\,1\,0) \in \{0,1\}^{(3,6)}.$$

მაგალითი 2. ეს მაგალითი ასახავს იმას, თუ რატომ უნდა იყოს გამოყენებული LD-OTS - ის ხელმოწერის გასაღებები მხოლოდ ერთხელ. დავუშვათ, n = 4 და ხელმომწერი ხელს აწერს ორ შეტყობინებას d_1 = (1, 0, 1, 1) და d_2 = (1, 1, 1, 0) იგივე გასაღებით. ამ შეტყობინებების ხელმოწერებია: "$\sigma_1 = (x_3[1], x_2[0], x_1[1], x_0[1])$" და "$\sigma_2 = (x_3[1], x_2[1], x_1[1], x_0[0])$" შესაბამისად, მოწინააღმდეგემ იცის $x_3[1], x_2[0], x_2[1], x_1[1], x_0[0], x_0[1]$ მნიშვნელობები ხელმოწერის გასაღებიდან გამომდინარე. მას შეუძლია დააგენერიროს ვალიდური ხელმოწერა შემდეგი შეტყობინებებისთვის: d_3 = (1, 0, 1, 0) და d_4 = (1, 1, 1, 1) . ეს მაგალითი შეიძლება იყოს განზოგადებული ნებისმიერი n - თვის. ასევე ჰაკერს შეუძლია დააგენერიროს ვალიდური ხელმოწერა მხოლოდ გარკვეული შეტყობინებებისთვის. რაც უფრო გრძელ ჰეშ ფუნქციას გამოვიყენებთ, მით უფრო უსაფრთხო იქნება ხელმოწერა, რადგან ჰაკერს გაუჭირდება შესაბამისი შეტყობინების მოძებნა.

2.7 Winternitz ერთჯერადი ხელმოწერის სქემა

მიუხედავად იმისა, რომ LD-OTS - ის ხელმოწერა და გასაღებები არის ძალიან ეფექტური, ხელმოწერის ზომა არის დიდი. Winternitz - ის ერთჯერადი ხელმოწერა (W-OTS) არის საგრძნობლად მოკლე. ამ სქემის იდეა მდგომარეობს ერთჯერადი ხელმოწერის ერთი გასაღების გამოყენებაში და ერთი შეტყობინების რამდენიმე ბიტის ერთდროულად ხელმოწერაში. როგორც LD-OTS - ი W-OTS - იც იყენებს one-way ფუნქციას

$$f : \{0,1\}^n \rightarrow \{0,1\}^n$$

და კრიპტოგრაფიულ ჰეშ ფუნქციას

$$g : \{0,1\}^n \rightarrow \{0,1\}^n.$$

W-OTS გასაღებთა წყვილის გენერაცია. w ≥ 2 არის Winternitz - ის პარამეტრი, რომელიც არის ერთდროულად ხელმოსაწერი ბიტების რაოდენობა. მაშინ

$$t_1 = \left\lceil \frac{n}{w} \right\rceil, t_2 = \left\lceil \frac{\lfloor log_2 t_1 \rfloor + 1 + w}{w} \right\rceil, t = t_1 + t_2. \quad (6)$$

ხელმოწერის X გასაღები არის

$$X = (x_{t-1}, \dots, x_1, x_0) \in_R \{0,1\}^{(n,t)}. \quad (7)$$

სადაც x_i უნიკალური სტრიქონები არჩეულია შემთხვევითად.

დადასტურების Y გასაღები გამოითვლება შემდეგნაირად: f ფუნქციაში გადის ხელმოწერის ყოველი ბიტი 2^w-1 - ჯერ. აქედან გამომდინარე, გვაქვს

$$Y = (y_{t-1}, \dots, y_1, y_0) \in \{0,1\}^{(n,t)}, \quad (8)$$

სადაც

$$y_i = f^{2^w-1}(x_i), 0 \leq i \leq t-1. \quad (9)$$

გასაღების გენერაციისთვის საჭიროა f-ის $t(2^w-1)$ ჯერ ხარისხში აყვანა, ხოლო t და n ბიტები არის ხელმოწერის სიგრძე და დადასტურების გასაღები.

W-OTS ხელმოწერის დაგენერირება. M შეტყობინება არის ხელმოწერილი g(M) = d = $(d_{n-1}, \dots, d_0)$. პირველ რიგში ხდება d - ს გაზრდა ნულებით, რათა d - ს სიგრძე იყოფოდეს w - ზე. დაგრძელებული d სტრიქონი იყოფა t_1 ბიტ სტრიქონებად, რომელთა სიგრძე არის: $b_{t-1}, \dots, b_{t-t_1}$. შესაბამისად, გვაქვს:

$$d = b_{t-1} \,||\, \dots \,||\, b_{t-t_1}, \tag{10}$$

სადაც || აღნიშნავს კონკატენაციას. შემდეგ, b_i სტრიქონების იდენტიფიკაცია ხდება {0, 1, … , 2_w - 1} მთელი რიცხვებით, ხოლო ჯამი გამოითვლება შემდეგი ფორმულით:

$$c = \sum_{i=t-t_1}^{t-1} (2^w - b_i) \tag{11}$$

როდესაც $c \leq t_1 2^w$ c - ს ბინარული გამოსახულების სიგრძე არის ნაკლები, ვიდრე

$$[log_2 t_1 2^w] + 1 = [log_2 t_1] + w + 1 . \tag{12}$$

იმისათვის, რომ (12) გამოსახულებიდან მიღებული გასაღების სიგრძე იყოფოდეს w - ზე, ხდება მისი გაზრდა ნულების ხარჯზე. შემდეგ ხდება მიღებული სტრიქონის t_2 ბლოკებად ($b_{t_2-1}, \dots , b_0$) დაყოფა. შესაბამისად $c = b_{t_2-1} \,||\, \dots \,||\, b_0$.

ბოლოს ხდება M - ის ხელმოწერის გამოთვლა

$$\sigma = \left(f^{b_{t-1}}(x_{t-1}), \dots , f^{b_1}(x_1), f^{b_0}(x_0)\right) \tag{13}$$

ყველაზე ცუდ შემთხვევაში, ხელმოწერის გამოსათვლელად საჭიროა $t(2^w - 1)$ დრო. W-OTS ხელმოწერის ზომა არის $t \cdot n$.

W - OTS ვერიფიკაცია. $\sigma = (\sigma_{t-1}, \dots , \sigma_0)$ ხელმოწერის ვერიფიკაციისათვის საჭიროა $b_{t-1}, \dots , b_0$ სტრიქონების გამოთვლა, რის შემდეგაც ვამოწმებთ შემდეგ სიდიდეებს:

$$\left(f^{2^{w-1-b_{t-1}}}(\sigma_{n-1}) , \dots , f^{2^{w-1-b_0}}(\sigma_0) \right) = (y_{n-1} , \dots , y_0). \tag{14}$$

თუ ხელმოწერა არის ნამდვილი, მაშინ " $\sigma_i = f^{b_i}(x_i)$ ". შესაბამისად,

$$f^{2^{w-1-b_i}}(\sigma_i) = f^{2^{w-1}}(x_i)) = y_i \tag{15}$$

[83] ავტორებს მიაჩნიათ, რომ Winternitz - ის ერთჯერადი ხელმოწერის სქემა არის მდგრადი ადაპტირებული შეტყობინებების შეტევის დროს, როდესაც ის არის რეალიზებული ფსევდო შემთხვევითი ფუნქციების გამოყენებით. Winternitz - ის ერთჯერადი ხელმოწერის სქემის გამოყენების დროს მცირდება ხელმოწერის სიგრძე, ხოლო უსაფრთხოების დონე რჩება უცვლელი.

2.8 Merkle ხის აუტენტიფიკაციის სქემა

ერთჯერადი ხელმოწერის სქემა არ არის საკმარისი ძირითადი პრაქტიკული სიტუაციებისთვის, რადგან ერთი გასაღები შეიძლება გამოყენებული იქნას მხოლოდ ერთი ხელმოწერის დასაგენერირებლად. 1979 წელს რალფ Merkle-მ წარმოადგინა აღნიშნული პრობლემის მოგვარების გზები. მისი იდეა ეფუძნება ერთიანი ბინარული ხის გამოყენებას, სადაც public key არის აღნიშნული ხის ფუძე (ბოლო ელემენტი).

Merkle- ს ხელმოწერის სქემა ეფუძნება ჰეშის ფუნქციას და ითვლება, რომ იგი მდგრადია კვანტური კომპიუტერების შეტევების მიმართ. სტატიაში განხილულია hardware -ის ოპტიმალური არქიტექტურა, რომელიც მნიშვნელოვნად ზრდის პროგრამული უზრუნველყოფის ეფექტურობას [90].

Merkle- ს ხელმოწერის სქემა (MSS) მუშაობს ნებისმიერ კრიპტოგრაფიულ ჰეშ ფუნქციასთან და ერთჯერადი ხელმოწერის სქემასთან. შემდეგ ორ თავში განხილულია აღნიშნული სქემის გასაღებების წყვილის გენერაციის მაგალითი, ფუძის (root) გამოთვლა, ხელმოწერის გენერაცია და მიღებული ციფრული ხელმოწერის ვერიფიკაცია.

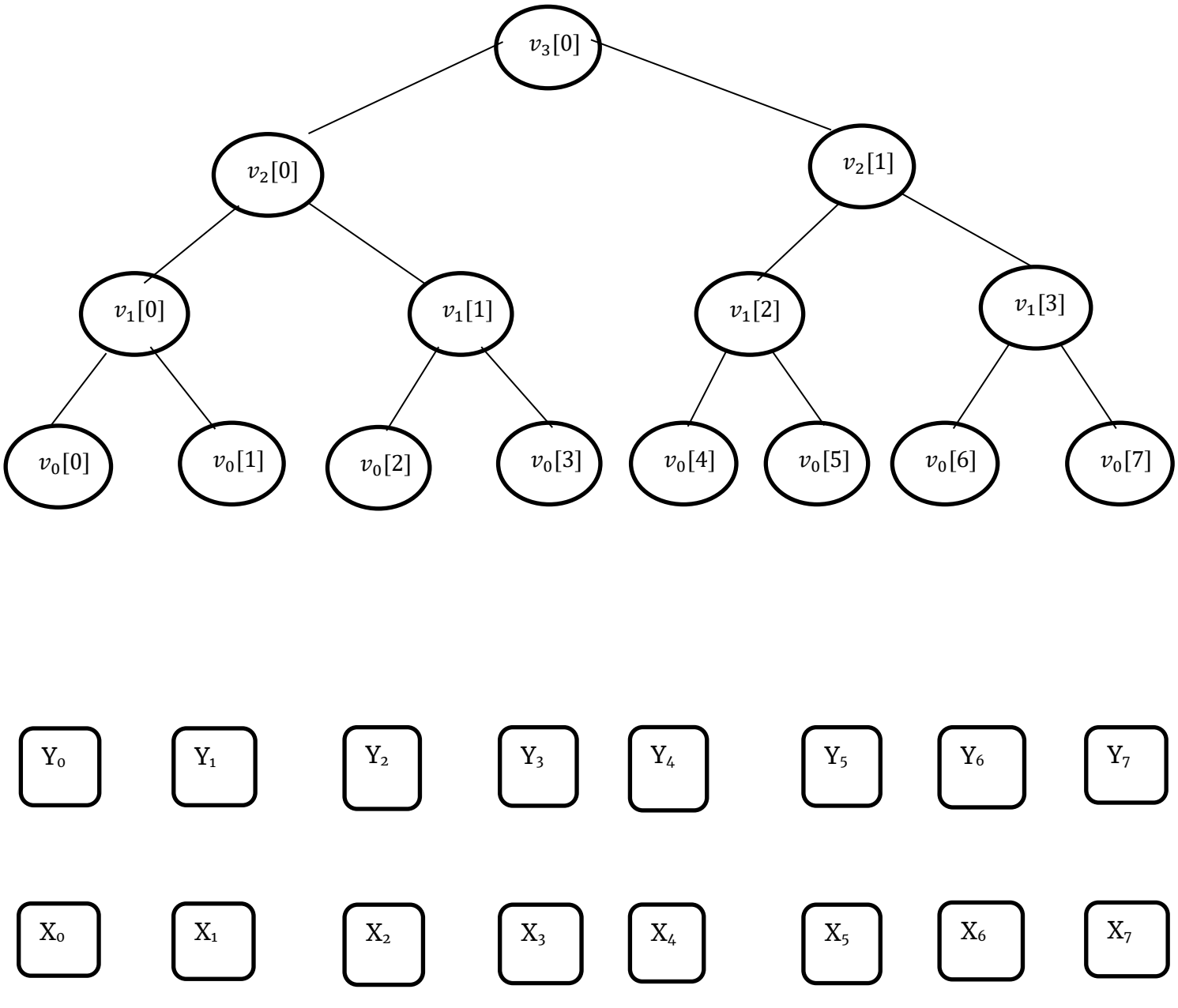
2.9 Mss - ის გასაღებთა წყვილის გენერაცია

გასაღებთა წყვილის გენერაციისას ხელმომწერი ირჩევს $H \in N, H \geq 2$. შემდგომ ეტაპზე, გასაღებთა წყვილს, რომელიც იქნება დაგენერირებული, შესაძლებლობა ექნება მოაწეროს/დაადასტუროს 2^H დოკუმენტები. საყურადღებოა ის გარემოება, რომ ეს სქემა განსხვავდება RSA - ის და ECDSA - ის სქემებისგან, სადაც ასევე ერთი გასაღებების წყვილით შეიძლება ბევრი დოკუმენტის ხელმოწერა/დადასტურება. თუმცა პრაქტიკაში დოკუმენტების რაოდენობა, რომელთა ხელმოწერა/დადასტურება შესაძლებელი იქნება გასაღებთა ერთი წყვილით, არის შეზღუდული მოწყობილობით, სადაც ხდება ხელმოწერის გენერაცია. ხელმომწერი აგენერირებს 2^H ერთჯერად გასაღებთა წყვილს (X_j, Y_j), $0 \leq j \leq 2^H$. X_j არის ხელმოწერის გასაღები და Y_j არის დადასტურების გასაღები. ორივე არის ბიტების სტრიქონი. Merkle- ს ხის ფოთლები არის $g(Y_j)$, $0 \leq j \leq 2^H$. Merkle- ს ხის შიდა კვანძები ითვლება შემდეგი წესის მიხედვით: მშობელი კვანძის მნიშვნელობა არის მარცხენა და მარჯვენა შვილი კვანძების კონკატენაციის ჰეში. MSS - ის public key არის ხის ბოლო ელემენტი. MSS - ის private key არის 2^H ერთჯერადი ხელმოწერის გასაღებების მიმდევრობა. Merkle- ს ხეში კვანძების აღნიშვნა ხდება შემდეგნაირად:

$v_h[j], 0 \leq j \leq 2^{H-h}$, სადაც $h \in \{0, \dots, H\}$ არის კვანძის სიმაღლე. შესაბამისად,

$$v_h[j] = g(v_{h-1}[2j] \,||\, v_{h-1}[2j+1]), 1 \leq h \leq H, 0 \leq j < 2^{H-h}. \quad (16)$$

ნახაზი 4 - ზე არის ნაჩვენები H = 3 მაგალითი:



ნახაზი 4-Merkle - ს ხე H = 3 სიმაღლით

MSS - ის გასაღებთა წყვილის გენერაციისათვის საჭიროა 2^H ერთჯერადი გასაღებთა წყვილის გამოთვლა და $2^{H+1} - 1$ ჰეშ ფუნქციის შეფასება.

2.10 ეფექტური ფუძის (root) გამოთვლა

იმისათვის, რომ მოხდეს Merkle- ს ხის ეფექტური ფუძის გამოთვლა, ამისთვის არ არის საჭირო მთელი ხის შენახვა. იმისათვის, რომ მოხდეს კვანძების შენახვა გამოყენებული არის სტეკი, ჩვეულებრივი push და pop ოპერაციებით. ამ ალგორითმის შემავალი პარამეტრი არის H , რომელიც არის Merkle- ს ხის სიმაღლე. შედეგი არის Merkle- ს ხის ფუძე, ანუ MSS public key . ქვემოთ არის მოყვანილი ალგორითმის ფსევდო კოდი:

Input: Height H ≥ 2

Output: Root of the Merkle tree

1. for i = 0, . . . , 2H − 1 do

1.1 Compute the ith leaf: N1 ← Calc(i)

1.2 While N1 has the same height as the top node on Stack do

1.2.1 Pop the top node: N2 ← Stack.pop()

1.2.2 Compute their parent node: N1 ← g(N2_N1)

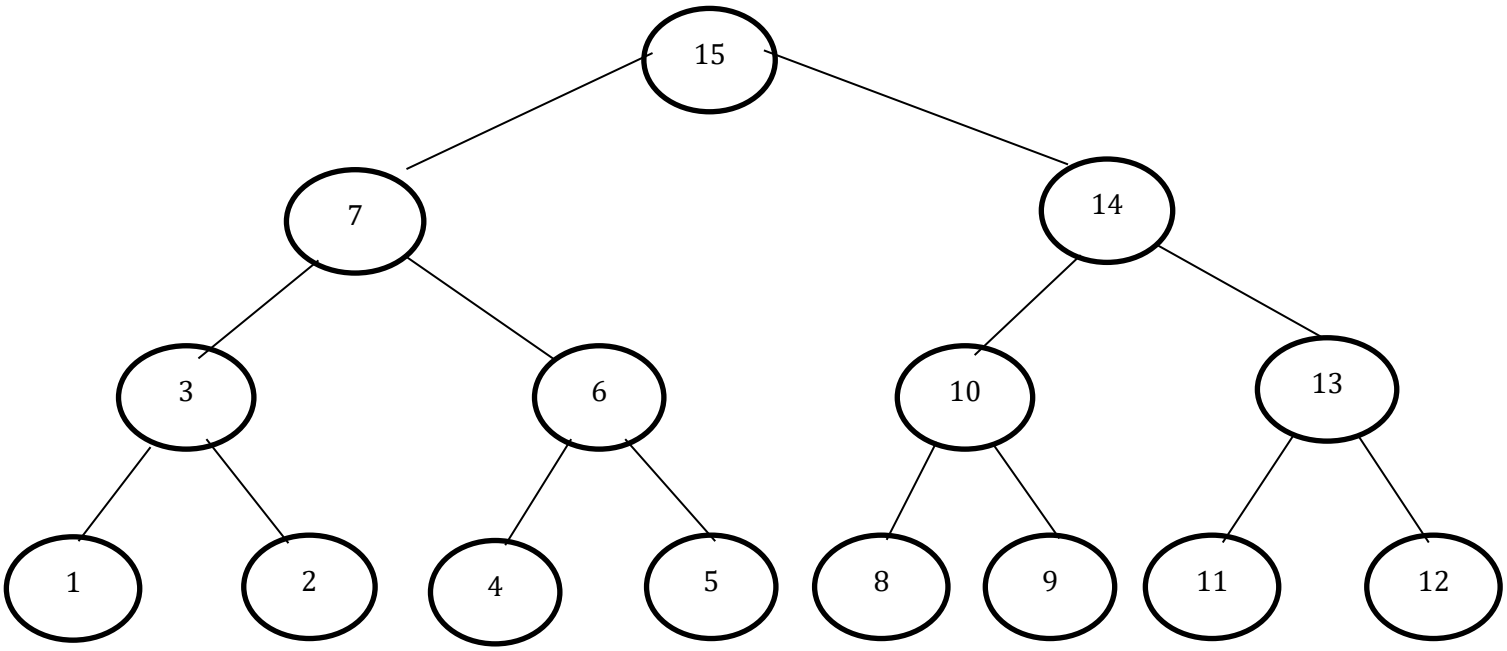
1.3 Push the parent node on the stack: Stack.push(N1)

2 SN is the single node stored on the stack: SN ← Stack.pop()

3 return SN

აღნიშნული ალგორითმი იყენებს Calc(i) , სადაც ხდება i - ური კვანძის გამოთვლა.

ნახაზი 5 - ზე ნაჩვენებია Merkle-ს ხის კვანძების გამოთვლის თანმიმდევრობა treehash ალგორითმის გამოყენებით. ამ მაგალითში ჩანს, რომ კვანძების მაქსიმალური რაოდენობა, რომლებიც არის შენახული სტეკში, არის 3. ჩვეულებრივი ალგორითმის შემთხვევაში, სტეკში შესანახი კვანძების მაქსიმალური რაოდენობა არის H . იმისათვის, რომ მოხდეს მერკლეს ხის ფუძის (root) გამოთვლა, ზემოთ აღწერილი ალგორითმის მიხედვით ხდება Calc(i) ფუნქციის გამოძახება 2H -ჯერ.



ნახაზი 5 - კვანძების გამოთვლის თანმიმდევრობა

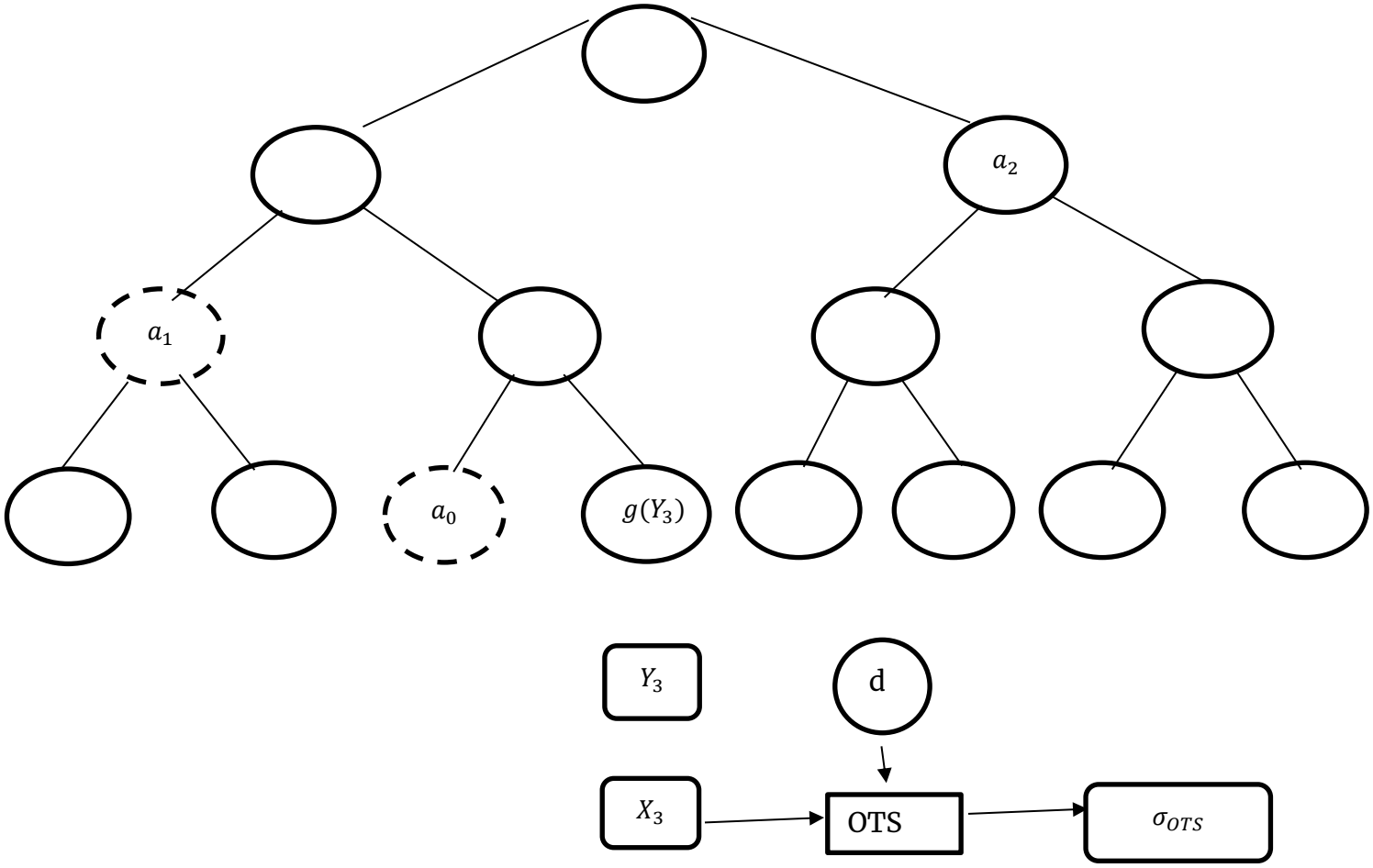
2.11 MSS ხელმოწერის გენერაცია

MSS -ი იყენებს one-time signature - ის გასაღებებს ხელმოწერის გენერაციისთვის. იმისათვის, რომ მოხდეს M შეტყობინების ხელმოწერა, თავიდან ხელმომწერი ითვლის n ბიტიან d = g(M) მიმდევრობას. ამის შემდეგ ის აგენერირებს σ_{OTS} one-time ხელმოწერას "sth one-time signature"-ის გასაღების გამოყენებით$X_s, s \in \{0, \dots, 2^H - 1\}$. Merkle-ს ხელმოწერა შეიცავს one-time signature - ს და one-time verification Y_ε გასაღებს. იმისათვის, რომ მოხდებს verifier - თვის Y_ε - ის ავთენტურობის დადასტურება, ხელმომწერი ამატებს, როგორც s ინდექსს, ასევე authentication path - ს Y_ε - ის verification გასაღებისთვის, რომელიც არის Merkle-ს ხის კვანძების $A_\varepsilon = (a_0, \dots, a_{H-1})$ მიმდევრობა. აღნიშნული ინდექსი და აუტენტიფიკაციის path - ი საშუალებას იძლევა მოხდეს გზის გაკვლევას $g(Y_\varepsilon)$ კვანძიდან Merkle-ს ხის ფუძემდე. კვანძი h - ი აუტენტიფიკაციის path - ში არის h კვანძის სიმაღლის ე.წ. sibling - ი $g(Y_\varepsilon)$ კვანძიდან Merkle-ს ხის ფუძემდე:

$$a_h = \{V_h\left[\frac{s}{2^h} - 1\right], \text{თუ } \left[\frac{s}{2^h}\right] \equiv 1 \, mod \, 2 \; V_h\left[\frac{s}{2^h} + 1\right], \text{თუ } \left[\frac{s}{2^h}\right] \equiv 0 \, mod \, 2 \quad (17)$$

სადაც h = 0, … H - 1 . ნახაზი 6 - ზე ნაჩვენებია მაგალითი, სადაც s = 3 . ამგვარად, s - ური Merkle-ს ხელმოწერა არის:

$$\sigma_s = (s, \sigma_{OTS}, Y_s, (a_0, \dots, a_{H-1})) \quad (18)$$



ნახაზი 6 - Merkle-ს ხელმოწერის დაგენერირება, სადაც s = 3 . ხაზიანი კვანძები აღნიშნავს $g(Y_\varepsilon)$ კვანძის path - ს ხოლო ისრები აღნიშნავს $g(Y_\varepsilon)$ კვანძიდან ხის ფუძემდე.

2.12 MSS ხელმოწერის ვერიფიკაცია

ზემოთ აღწერილი ხელმოწერის ვერიფიკაცია შედგება ორი ნაბიჯისგან. პირველ ნაბიჯში verifier - ი იყენებს ერთჯერად ვერიფიკაციის Y_ε გასაღებს, რათა მოახდინოს σ_{OTS} ხელმოწერის ვერიფიკაცია, რომელიც შედგება d მიმდევრობისგან. მეორე ნაბიჯად verifier - ი ახდენს Y_ε ერთჯერადი გასაღების ვალიდურობის შემოწმებას. ეს შემოწმება ხდება $(p_0, \dots, p_h)$ path - ის აწყობით დაწყებული s - ური $g(Y_s)$ კვანძიდან Merkle-ს ხის ფუძემდე. ალგორითმი იყენებს s ინდექსს, აუტენტიფიქაციის $(a_0, \dots, a_{H-1})$ path - ს ხოლო p_h სიდიდეების გამოსათვლელად იყენებს შემდეგ გამოსახულებას:

$$p_h = \{g(a_{h-1}||p_{h-1}), თუ \left\lfloor \frac{s}{2^{h-1}} \right\rfloor \equiv 1 \ mod\ 2 \ g(p_{h-1}||a_{h-1}), თუ \left\lfloor \frac{s}{2^{h-1}} \right\rfloor \equiv 0 \ mod\ 2 \quad (19)$$

სადაც h = 1, … H და $p_0 = g(Y_s)$. s ინდექსი გამოიყენება იმისათვის, რომ გავიგოთ path - ში თუ რა მიმდევრობით იქნება Merkle-ს ხეში $g(Y_s)$ კვანძები კონკატენირებული. Y_ε ერთჯერადი ხელმოწერის აუტენთიფიკაცია არის წარმატებული მხოლოდ იმ შემთხვევაში, თუ p_H უდრის public გასაღებს [59-62].

2.13 ერთჯერადი key-pair გენერაცია PRNG - ის გამოყენებით

MSS - ის private გასაღების შედგება 2^H ერთჯერადი გასაღებებისგან. ასეთი დიდი მონაცემების შენახვა ზოგი აპლიკაციისათვის არის მიუღწეველი. იმისათვის, რომ მოხდეს მეხსიერების დაზოგვა, ხდება ციფრების ფსევდო შემთხვევითი გენერატორის გამოყენება(PRNG). ამ შემთხვევაში, საჭიროა მხოლოდ PRNG - თვის საჭირო seed - ის შენახვა. ყოველი ერთჯერადი ხელმოწერის დაგენერირება ხდება ორჯერ: ერთხელ, public გასაღების გენერირების დროს და მეორედ, ხელმოწერის დროს.

ქვემოთ არის მოყვანილი PRNG , რომელიც არის უსაფრთხო. ის იღებს n სიგრძის seed($Seed_{in}$) ბიტებს, ხოლო შედეგად გვაძლევს შემთხვევით(Rand) რიცხვებს და განახლებულ seed($Seed_{out}$) - ს. ორივე არის n სიგრძის [63, 64].

$$PRNG : \{ 0,1 \}^n \rightarrow \{ 0,1 \}^n \text{ x } \{ 0,1 \}^n \tag{20}$$

$$SEED_{IN} \rightarrow (RAND, SEED_{OUT})$$

2.14 MSS key pair გენერაცია PRNG - ის გამოყენებით

განვიხილოთ, თუ როგორ ხდება MSS key pair გენერაცია შემთხვევითი რიცხვების გენერატორის PRNG - ის გამოყენებით. პირველი ნაბიჯი არის n სიგრძის seed($Seed_0$) ბიტების არჩევა. ერთჯერადი ხელმოწერის დაგენერირებისთვის გამოიყენება $SeedOTS_j$, $0 \leq j < 2^H$

seeds - ების მიმდევრობა. გამოთვლა ხდება შემდეგნაირად:

$$(SeedOTS_j , Seed_{j+1}) = PRNG(Seed_j), 0 \leq j < 2^H. (21)$$

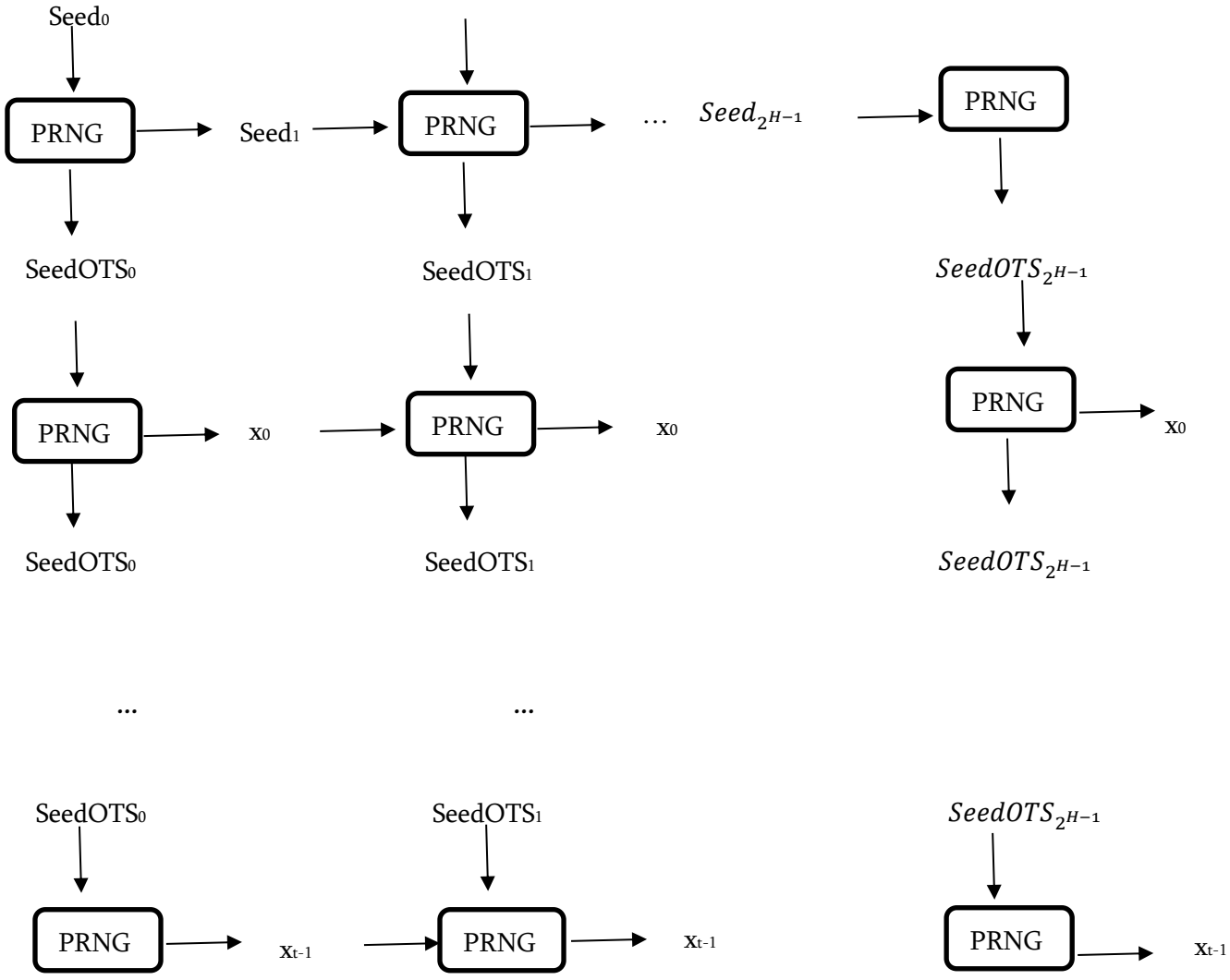
$SeedOTS_j$ გამოიყენება j - ური ერთჯერადი ხელმოწერის გასაღების გამოსათვლელად. მაგალითად, W-OTS - ის შემთხვევაში j - ური გასაღები არის

$X_j = (x_{t-1} , \dots , x_0)$. t ბიტებისგან შემდგარი n სიგრძის სტრიქონები არის დაგენერირებული $SeedOTS_j$ გამოყენებით.

$$(x, SeedOTS_j) = PRNG(SeedOTS_j), i = t - 1 , \dots , 0 (22)$$

$SeedOTS_j$ seed - ის განახლება ხდება PRNG - ზე ყოველი მიმართვისას. იმისათვის, რომ მოხდეს X_j ხელმოწერის გასაღების გამოთვლა, საჭიროა მხოლოდ $Seed_j$ - ის ცოდნა. $SeedOTS_j$ სიდიდის გამოთვლისას ასევე გამოითვლება ახალი $Seed_{j+1}$, რომელიც საჭიროა X_j ხელმოწერის გასაღების გამოთვლისთვის. ნახაზი 7 - ზე არის ნაჩვენები ერთჯერადი ხელმოწერის გასაღების გენერაცია PRNG - ის გამოყენებით.

თუ გამოყენებული ზემოთ აღწერილი მეთოდი, მაშინ MSS private key წარმოადგენს $Seed_0$ - ს, რომლის სიგრძე არის n .



ნახაზი 7. ერთჯერადი ხელმოწერის გასაღების დაგენერირება PRNG - ის გამოყენებით.

2.15 აუტენტიფიკაციის path - ის გამოთვლა

ქვემოთ მოყვანილია H სიმაღლის Merkle-ს ხის სხვადასხვა traversal ვარიანტები. ეს ვარიანტები არის გამჭირვალე verfier -სთვის, რომლისთვისაც არ არის საჭირო იმის ცოდნა, თუ როგორ არის დაგენერირებული შედეგი. მისთვის საჭიროა მხოლოდ შედეგის ცოდნა. პირველი traversal ალგორითმი სტრუქტურულად არის ძალიან მარტივი. მისთვის საჭირო პარამეტრების მოცულობა არის შეზღუდული $1.5H^2$ / logH ჰეშ მნიშვნელობით. ცუდ შემთხვევაში, გამოთვლითი დატვირთვა არის 2H / logH ხის კვანძების გამოთვლები ერთი შედეგისთვის.

Merkle-ს ხის traversal ალგორითმს აქვს უკეთესი მოცულობა და დროის სირთულე, ვიდრე წინა ალგორითმებს. აღნიშნული ალგორითმი ითხოვს 2H ხის კვანძების გამოთვლას ერთ ციკლზე და ითხოვს პარამეტრების 3H - ზე ნაკლებ მოცულობას. Merkle-ს ხის traversal ალგორითმი არის ერთ-ერთი ოპტიმალური, რადგან ჯერჯერობით არ არსებობს Merkle-ს ხის ალგორითმი, რომელიც ითხოვს O(H) - ზე ნაკლებ დროს და O(H) - ზე ნაკლებ მოცულობას.

Merkle-ს ხის მესამე traversal ალგორითმს აქვს იგივე მოცულობისა და დროის სირთულე, რაც მეორეს. თუმცა, მას მაინც აქვს თავისი უპირატესობები. ის ასხვავებს ფოთლების გამოთვლის ოპერაციას შიდა კვანძების გამოთვლის ოპერაციისგან. იმისათვის, რომ მოხდეს H სიმაღლის ხის გავლა , ის მოითხოვს H/2 ფოთლების და 3H/2 შიდა კვანძების გამოთვლას.

2.16 კლასიკური traversal ალგორითმი

Merkle-ს ხის traversal - ის მთავარი გამოწვევა არის ყველა კვანძის მნიშვნელობის შენახვა და ციფრული ხელმოწერის გამოთვლის პროცესში მათი მონაწილეობა. ამავდროულად ალგორითმის გამოთვლის დრო და მისთვის გამოყოფილი მეხსიერება არ უნდა აღემატებოდეს Merkle-ს კლასიკურ ალგორითმის მუშაობის დროს და დაკავებულ მეხსიერებას.

საშუალო ხარჯები(Average costs). ხეში ყოველი კვანძი საბოლოო ჯამში არის authentication path - ის ნაწილი. ასე რომ, ერთ-ერთი ეფექტური საშუალება არის ყოველი კვანძის გამოთვლის საერთო ხარჯის დათვლა. დავუშვათ, გვაქვს 2^{H-h} მარჯვენა(შესაბამისად მარცხენა) კვანძები h სიმაღლის ხეში. თუ დავითვლით ყოველ კვანძს დამოუკიდებლად, ყოველი მათგანის საფასური იქნება 2^{h+1} - 1 ოპერაცია და თუ შევაჯამებთ გქვექნება 2^{H+1} = 2N ოპერაცია. ყოველი h ($0 \leq h \leq H$) სიმაღლისთვის ყველა საფასურის დაჯამებისას საშუალოდ გვექნება 2H = 2log(N) სავალდებულო ოპერაცია.

სამი კომპონენტი. ხელმოწერის ციფრულ სქემაში, Merkle-ს ხის traversal ალგორითმიც შედგება სამი კომპონენტისგან: გასაღების გენერაცია, შედეგი და ვერიფიკაცია. გასაღების გენერაციის დროს ხორციელდება პირველი authentication path - ი და სხვა შემდეგი კვანძების მნიშვნელობების გამოთვლა.

შედეგის გამოთვლის ფაზა შედგება N ციკლისგან, თითო ფოთლისთვის გვაქვს $s \in \{0, \dots, N-1\}$. ყოველი s ციკლის დროს, authentication path - ი s - ური ფოთლისთვის შედეგი არის $AUTH_i, i = 0, \dots, H-1$.

ვერიფიკაციის ფაზა არის კლასიკური Merkle-ს ხის ვერიფიკაციის ფაზის იდენტური.

დამატებით მიმდინარე authentication $AUTH_h$ კვანძების აღნიშვნის გარდა, ასევე გვჭირდება რაიმე აღნიშვნა, რათა აღვწეროთ stack - ი, რომელიც გამოყენებული იქნება შემდეგი კვანძების გამოსათვლელად [65-67].

2.17 Traversal ალგორითმის პრეზენტაცია

გასაღების გენერაცია და setup-ი. გასაღების გენერაციის მთავარი ამოცანა არის ფუძის მნიშვნელობის გამოთვლა და გამოტანა. გამოთვლის პროცესში ყოველი კვანძის მნიშვნელობის გამოთვლის გარდა, ასევე მნიშვნელოვანია Auth_i საწყისი მნიშვნელობის დაფიქსირება.

თუ აღვნიშნავთ j - ურ კვანძს h სიმაღლეზე, როგორც $V_h[j]$, მაშინ გვექნება $\text{Auth}_j = V_h[1]$ (მარჯვენა კვანძები). შემდეგი კვანძი h სიმაღლეზე არის $V_h[0]$ (მარცხენა კვანძები).

ალგორითმი 1 გასაღების გენერაცია და setup - ი:

1. For each $h \in \{ 0, 1, \dots H-1 \}$: Calculate $\text{Auth}_h = V_h[1]$.

2. For each $h \in \{ 0, 1, \dots H-1 \}$: Setup STACK_h with the single node value $\text{Auth}_h = V_h[0]$.

3. Calculate and publish tree root, VH[0].

შედეგი. Merkle-ს ხის traversal ალგორითმი ყოველი h სიმაღლისთვის ითვლის შემდეგი authentication კვანძის მნიშვნელობას. ყოველი 2^h ციკლისთვის authentication path - ი გადაინაცვლებს მარჯვნივ.

ყოველი ციკლის დროს ალგორითმის მდგომარეობა (state) იცვლება. ყოველი 2^h ციკლის შემდეგ, კვანძის გამოთვლის პროცედურა დასრულდების შემდგომ გაეშვება ხის ახალი instance - ი შემდეგი authentication კვანძის დონესთვის (level - ი).

იმისათვის, რომ გავიგოთ, როგორ უნდა მოხდეს Auth კვანძების განახლება (refresh), ჩვენ ვახდენთ იმის დადგენას, თუ რომელ სიმაღლეზე უნდა მოხდეს განახლება: h სიმაღლის განახლება ხდება მხოლოდ იმ შემთხვევაში, თუ 2^h იყოფა უნაშთოდ, სადაც

$s \in \{ 0, \dots, N-1 \}$ აღნიშნავს არსებულ ციკლს. გარდა ამისა, აღსანიშნავია, რომ $s + 1 + 2^h$ ციკლზე authentication path - ი გაივლის $(s + 1 + 2^h) / 2^h$ კვანძზე h სიმაღლეზე. ამრიგად, მისი sibling - ის მნიშვნელობა (ახალი სავალდებულო შემდეგი Auth_h) გამოითვლება 2^h ფოთლის მნიშვნელობისგან, დაწყებული $(s + 1 + 2^h) \oplus 2^h$ ფოთლიდან.

ალგორითმი 2 Merkle-ს კლასიკური traversal ხის ალგორითმი:

1. Set s = 0.

2. Output:

- For each $h \in [\,0, H-1\,]$ output Auth_h.

3. Refresh Auth Nodes:

For all h such that 2^h divides s + 1:

- Set Auth_h be the sole node value in STACK_h.
- Set startnode = (s + 1 + 2^h) $\oplus$ 2^h.
- STACK_h.*initialize*(startnode, h)/

4. Build Stacks:

For all $h \in [\,0, H-1\,]$:

- STACK_h.*update*(2). (Each stack receives two updates)

5. Loop:

- Set s = s + 1.
- If s < 2^H go to Step 2.

[12] წყაროში არის შეთავაზებული Merkle - ს ხის traversal ალგორითმის გაუმჯობესებული ვერსია. სტატიაში შეფასებულია შემოთავაზებული ალგორითმის უპირატესობები სხვა, უკვე არსებულ ალგორითმებთან შედარებით.

GMSS -ი არის ხელმოწერის პირველი სქემა , რომელიც საშუალებას იძლევა გასაღების ერთი წყვილის გამოყენებით განხორციელდეს 2^{80} რაოდენობის დოკუმენტების ხელმოწერა [84]. კიდევ ერთი გაუმჯობესებული Merkle-ს ხის მოდიფიკაცია არის CMSS . შემოთავაზებულმა ალგორითმმა შეამცირა private გასაღების სიგრძე, გასაღებების წყვილისა და ხელმოწერის გენერაციის დროები [92].

ნაჩვენებია Merkle Tree Traversal სისტემის გაუმჯობესებული კიდევ ერთი ვარიანტი, კერძოდ არის წარდგენილი ახალი ალგორითმი, რომელიც ითვლის authentication path - ს. ავტორების მტკიცებით, ყველაზე ცუდ შემთხვევაში მათი ალგორითმის შესრულების დრო არის ნაკლები კლასიკურ Merkle Tree Traversal ალგორითმთან შედარებით [94-96].

2.18 Merkle-ს ფრაქტალური traversal ხე

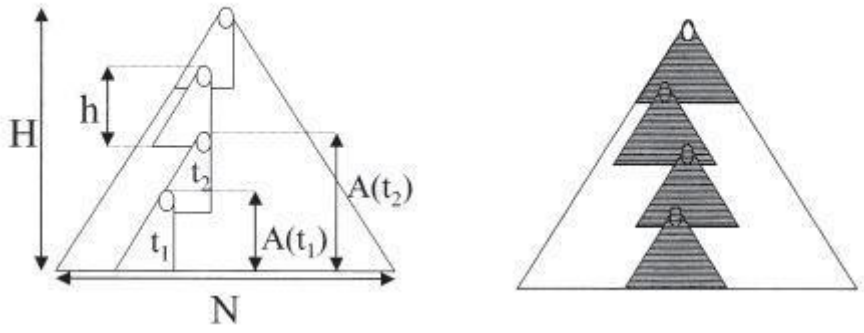
ტერმინი ფრაქტალური არჩეული იყო იმის გამო, რომ დიდი Merkle-ს ხის ჩაშლა ხდება პატარა ხეების ნაწილებად.

ამ ალგორითმის არსი არის selection-ები, სადაც ხდება კვანძების მნიშვნელობების გამოთვლა და ყოველ ბიჯზე მათი შენახვა. ამ selection-ების აღწერა ხდება ფიქსირებული h სიმაღლის ქვეხეების (subtrees) გამოყენებით.

ამ ალგორითმში H სიმაღლის Merkle-ს ხიდან დაწყებული, აღნიშვნები გამოიყენება იმისათვის, რომ მოხდეს ქვეხეების (subtrees) გამოთვლა. თავიდან ვირჩევთ ქვეხეს , რომლის სიმაღლე არის h < H . დავუშვათ ის, რომ v კვანძის სიმაღლე ხეში არის ტოლი path - ის სიგრძის v - დან ხის ფოთლამდე (აქედან გამომდინარე, ხის ფოთლის სიმაღლე არის 0). დავუშვათ, გვაქვს მინიმუმ h სიმაღლის v კვანძი. განვსაზღვროთ h subtree , რომლის სიმაღლე არის h და v ფუძე. სიმარტივისთვის დავუშვათ, რომ h არის H - ის გამყოფი და ასევე დავუშვათ, რომ სიხშირე L = H / h არის რიცხვი, რომელიც აღნიშნავს subtree-ების დონეს. h subtree არის i - ურ დონეზე, როდესაც მისი სიმაღლე არის ih , სადაც

$i \in \{ 1,2, \dots, H \}$. ყოველი i - თვის, გვაქვს 2^{H-ih} h subtree- ების i - ურ დონეზე.

ქვემოთ ნახაზზე ნაჩვენები არის h - subtree-ების სერია $Tree_i$ (i = 1 … L) . თუ i < L , მაშინ $Tree_i$ - ის ფუძე არის $Tree_{i+1}$ - ის ფოთოლი.



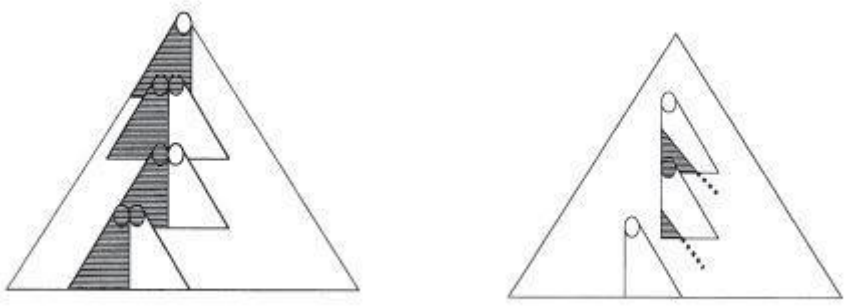
ნახაზი 8.

2.19 არსებული და სასურველი Subtrees - ები

Static view. როგორც წინა თავში იყო აღნიშნული, ალგორითმი ინახავს ზოგიერთი კვანძის მნიშვნელობას და დროთა განმავლობაში ხდება მათი განახლება. ნებისმიერი ციკლის დროს, სადაც გვაქვს შედეგი, გვექნება არსებული ხეების stack - ი. შეგვიძლია ვთქვათ, რომ ჩვენ ვადებთ pebble - ს TREE ხის v კვანძზე, როდესაც ვახდენთ ამ კვანძის შენახვას. ყოველთვის გვექნება L რაოდენობის ისეთი $Exist_i$ ქვეხეები (subtrees - ები), სადაც ყოველი $i \in \{1, \dots L\}$ და მათ ყოველ კვანძს აქვთ თავიანთ pebbles - ები (გარდა მათი ფუძეებისა). სტრუქტურის მიხედვით, ნებისმიერი ფოთლისთვის $Exist_1$ - ში შესაბამისი authentication path-ი უკვე არის შენახული არსებული ქვეხეების (subtrees - ების) stack - ში.

Dynamic view. გარდა ზემოთ აღწერილი არსებული ქვეხეების (subtrees - ების) მიმდევრობისა, რომლებიც შეიცავს შემდეგ საჭირო authentication path-ს, ასევე გვექნება სასურველი ქვეხეების (subtrees - ების) მიმდევრობა. თუ $Exist_i$ - ი ხის ფუძეს აქვს ინდექსი a , მისი ih კვანძების განლაგებიდან გამომდინარე, $Desire_i$ არის h ქვეხე (subtree - ი) a + 1 ინდექსით (გვაქვს $a < 2^{H-i \cdot h} - 1$). თუ $a = 2^{H-i \cdot h} - 1$, მაშინ $Exist_i$ - ი არის ბოლო ქვეხე (subtree - ი) ამ დონეზე (level - ზე) და შესაბამისად, აღარ გვექნება სასურველი ქვეხე (subtree - ი). ნახაზი 9 - ის მარცხენა მხარეს არის ნაჩვენები არსებული და სასურველი ქვეხეები (subtrees - ები). ასევე ჩვენ უნდა გამოვთვალოთ pebbles - ები სასურველ ქვეხეებში (subtrees - ები). ამის განხორციელება შესაძლებელია აპლიკაციაში treehash ალგორითმის ადაპტაციით $Desire_i$ ხის ფუძეზე.

ასეთი შემთხვევებისთვის treehash ალგორითმი არის შეცვლილი ისეთი სახით, რომ შესაძლებელი იყოს pebbles - ების შენახვა $Desire_i$ - თვის და არა მათი იგნორირება. ასევე ამ ალგორითმში ხდება ისეთი ცვლილების შეტანა, რომლის მიხედვითაც საჭიროა იმის გათვალისწინება, რომ არ მოხდეს ხის ფუძის გამოთვლა ერთი ციკლით ადრე. ამ ვარიანტის treehash ალგორითმის გამოყენების დროს ვხედავთ, რომ ყოველ გამოთვლილ სასურველ ქვეხეს (subtree - ი) მოჰყვება შენახული შუალედური pebbles მნიშვნელობები. ეს ყველაფერი არის ნაჩვენები ნახაზი 9 - ის მარჯვენა მხარეს, სადაც ნაწილობრივად გამოთვლილი გვაქვს ქვეხეები (subtrees - ები) და მათთან ასოცირებული pebbles - ები.



ნახაზი 9. (მარცხენა) ნაცრისფერი ქვეხეები (subtrees - ები) აღნიშნავს არსებულ ქვეხეებს (subtrees - ებს), ხოლო თეთრი ქვეხეები (subtrees - ები) აღნიშნავს სასურველ ქვეხეებს (subtrees - ებს). არსებული ქვეხეების (subtrees - ების) გამოყენებისას ხდება სასურველი ქვეხეების (subtrees - ების) აგება. (მარჯვენა) ნახაზზე არის ნაჩვენები სასურველი ქვეხეების (subtrees - ების) მიმდევრობა მარცხენა ნახაზიდან, მაგრამ ნაცრისფერი გამოთვლილი კვანძებით და წყვეტილი ხაზებით, რომლებიც აღნიშნავს pebbles - ებს [68,69].

დასკვნა

ზემოთ აღწერილი სისტემებიდან გამომდინარე შეგვიძლია დავასკვნათ, რომ ზოგი მათგანი არის მდგრადი კლასიკური კომპიუტერების შეტევებისგან, მაგრამ არ წარმოადგენენ მდგრად ალგორითმებს კვანტური კომპიუტერების შეტევებთან მიმართებაში.

ასევე ზემოთ აღწერილი Merkle-ს კლასიკური ალგორითმის და მისი სხვადასხვა ვარიაციებიდან გამომდინარე შეგვიძლია დავასკვნათ, რომ ისინი მდგრადია კვანტური კომპიუტერების შეტევებისგან, მაგრამ სისწრაფის და ოპერაციების რაოდენობის მხრივ ისინი არაეფექტურებია.

შემდეგ თავში განხილულია ჩვენს მიერ შემოთავაზებული ალგორითმი, რომელიც არის სისწრაფის და ოპერაციების რაოდენობის კუთხით უფრო ეფექტური ვიდრე Merkle-ს კლასიკური ალგორითმი.

თავი 3-Merkle-ს კლასიკური ალგორითმის გაუმჯობესება

3.1 გაუმჯობესებული სისტემა

მეცნიერები აქტიურად მუშაობენ კვანტური კომპიუტერების შექმნაზე. კვანტურ კომპიუტერებს შეეძლება დიდი ციფრების ფაქტორიზაციის განხორციელება. შესაბამისად, კვანტურ კომპიუტერებს შეეძლება RSA ალგორითმის გატეხვა, რომელსაც დღესდღეისობით ბევრი პროგრამა იყენებს. ჰეშზე დაფუძნებული ციფრული ხელმოწერები არის RSA - ს ალტერნატივა. ეს სისტემები იყენებს ჰეშ ფუნცქიებს. ამ სისტემების დაცულობა დამოკიდებული არის ჰეშ ფუნქციების კოლიზიაზე. ამ თავში განხილული იქნება ციფრული ხელმოწერების სქემები და Merkle-ს ფრაქტალური სქემის გაუმჯობესებული ვარიანტები. Merkle-ს კლასიკური ალგორითმი შეიძლება მიჩნეულ იქნას როგორც სტატიკური, რადგან ის არ არის დამოკიდებული პროცესორის ნაკადების რაოდენობაზე. ჩვენ გთავაზობთ ალგორითმს, რომელიც იყენებს პროცესორის ნაკადებს. აქ არის წარმოდგენილი ამ ალგორითმის მათემატიკური მოდელი და ალგორითმის ფსევდო კოდი. მისი ეფექტურობა დასტურდება მიღებული შედეგებით .

რადგან კვანტური კომპიუტერების შემუშავება აქტიურ ფაზაშია და Shor - ის ალგორითმის დახმარებით შეუძლიათ მარტივად გატეხონ სისტემები, რომლებიც იყენებს რიცხვების ფაქტორიზაციას, ბევრი არსებული სისტემა ხდება დაუცველი. ქვემოთ განვიხილავთ ხელმოწერების სხვადასხვა სქემებს. [78-82] ამ წყაროებში არის მოყვანილი კვანტური გამოთვლების წარმატებები, მათ შორის Shor - ის ფაქტორიზაციის ალგორითმი და დისკრეტული ლოგარითმების შეფასება (evaluation of discrete logarithms). განხილულია ამ ალგორითმების ძირითადი პრინციპები და ე.წ. დამალული ქვეჯგუფების პრობლემები.

3.2 Lamport-Diffe ერთჯერადი ხელმოწერების სისტემა

Lamport-Diffe ერთჯერადი ხელმოწერების სისტემაში X გასაღებისთვის გენერირდება n ზომის 2n შემთხვევითი ხაზი:

$$X= (x_{n-1}[0], x_{n-1}[1], \ldots, x_0[0], x_0[1]) \in \{0,1\}^{n,2n}. \quad (23)$$

აღნიშნულ სისტემაში Verification გასაღები არის

$$Y= (y_{n-1}[0], y_{n-1}[1], \ldots, y_0[0], y_0[1]) \in \{0,1\}^{n,2n}. \quad (24)$$

მისი გამოთვლა ხორციელდება შემდეგნაირად:

f არის ცალმხრივი ფუნქცია:

$$f: \{0,1\}^n \to \{0,1\}^n (4). \quad (25)$$

იმისათვის, რომ მოხდეს m შეტყობინების ხელმოწერა, ხდება

$$h(m)=hash = (hash_{n-1}, \ldots , hash_0) \quad (26)$$

ჰეშ მნიშვნელობების გამოთვლა, სადაც h არის კრიპტოგრაფიული ჰეშ ფუნქცია:

$$h: \{0,1\}^* \to \{0,1\}^n. \quad (27)$$

ხელმოწერის გამოთვლა ხდება შემდეგნაირად:

$$sig = (x_{n-1}[hash_{n-1}], \ldots, x_0[hash_0]) \in \{0,1\}^{n,n}, \quad (28)$$

ხელმოწერის ზომა არის n^2. იმისათვის, რომ მოხდეს sig ხელმოწერის დადასტურება, ხდება შეტყობინების ჰეშირება

$$hash = (hash_{n-1}, \ldots , hash_0). \quad (29)$$

რის შემდგომაც ხორციელდება განტოლების ამოხსნა:

$$(f(sig_{n-1}), \ldots, f(sig_0)) = (y_{n-1}[hash_{n-1}], \ldots, y_0[hash_0]) \quad (30)$$

თუ აღნიშნული სიდიდეები აღმოჩნდება ტოლი, მაშინ ციფრული ხელმოწერა არის ვალიდური.

3.3 Winternitz - ის ერთჯერადი ხელმოწერა

Lamport - ის სქემაში ხელმოწერის ზომა არის n^2 . იმისათვის, რომ მოხდეს მისი შემცირება, ჩვენ ვიყენებთ Winternitz - ის ერთჯერადი ხელმოწერის სქემას. აღნიშნულ სქემაში ხდება ჰეშირებული შეტყობინების რამდენიმე ბიტის ერთდროულად ხელმოწერა. Winternitz - ის პარამეტრი არის ციფრი, რომელიც წარმოადგენს შეტყობინებაში იმ ბიტების რაოდენობას, რომლებიც იქნება ხელმოწერილი. მაგალითად, აიღება w >= 2 რის შემდეგაც ხდება შემდეგი სიდის გამოთვლა:

$$p_1=n / w \text{ and } p_2 = (\log_2 p_1 +1 + w) / w, p = p_1+ p_2 \quad (31)$$

ციფრული ხელმოწერის შემდგომ ეტაპზე ვაგენერირებთ ხელმოწერის შემთხვევით გასაღების:

$$X = (x_{p-1}[0], \ldots, x_0) \in \{0,1\}^{n,p} \quad (32)$$

ამის შემდეგ ვახდენთ verification გასაღების გამოთვლას:

$$Y = (y_{p-1}[0], \ldots, y_0) \in \{0,1\}^{n,p} \quad (33),$$

სადაც

$$y_i = f^{2\wedge w-1}(x_i), 0 <= i <= p-1. \quad (34)$$

ხელმოწერის და verification გასაღების სიგრძე უდრის np ბიტს, ხოლო f ფუნქცია ხელმოწერის ფორმირების პროცესში არის გამოყენებული $p(2^w-1)$ ჯერ. იმისათვის, რომ მოხდეს შეტყობინების ხელმოწერა, ხდება მისი ჰეშირება h ფუნქციის გამოყენებით - hash=h(m). იმისათვის, რომ ჰეშირების შედეგად მიღებული მონაცემის სიგრძე იყოფოდეს w პარამეტრზე ხორციელდება მისთვის ნულების დამატება. აღნიშნული ოპერაციის შედეგად მიღებული მიღებული ახალი სიდიდის სიგრძე იყოფა w ზომის p_1 ნაწილებად:

$$hash = k_{p-1},\ldots, k_{p-p1} \quad (35)$$

გამოთვლების შედეგად მიიღება შემდეგი სიდიდეები:

$$c = \sum_{i=p-p1}^{p-1}(2^w-k_i) \quad (36)$$

სადაც $c <= p_1 2^w$ ბინარული გამოსახულების სიგრძე არის $\log_2 p_1 2^w+1$ - ზე ნაკლები. ამ ბინარულ გამოსახულებაში ნულების დამატება პრაქტიკულად არ ხდება. ამგვარად, ჩვენ ვყოფთ მას w ზომის p_2 ნაწილებად:

$$c = k_{p2-1},\ldots, k_0 \quad (37)$$

შეტყობინების ხელმოწერის გამოთვლა ხდება შემდეგნაირად:

$$sig = (f\hat{}k_{p-1}(x_{p-1}), \ldots, f\hat{}k_0(x_0)) \quad (38)$$

ხელმოწერის ზომა უდრის pn - ს.

იმისათვის, რომ მოხდეს $sig = (sig_{n-1}, \ldots, sig_0)$ ხელმოწერის დადასტურება, ხორციელდება $k_{p-1},\ldots, k_0$ სიდიდეების გამოთვლა. შედეგად კი ხდება შემდეგი განტოლების ამოხსნა:

$$(f\hat{}(2^w-1-k_{p-1}))(sig_{n-1}) , \ldots, (f\hat{}(2^w-1-k_0))(sig_0) = y_{n-1}, \ldots y_0 \quad (39)$$

Public key - ს ბევრჯერ გადაცემის გამო, ასეთი სისტემები არის არაეფექტური. Merkle-ს კრიპტოსისტემა აგვარებს ამ პრობლემას. ამ სისტემაში ერთი და იგივე Public key - ი შეიძლება რამდენჯერმე იყოს გამოყენებული სხვადასხვა შეტყობინების დასაშიფრად.

3.4 Merkle-ს კრიპტოსისტემა

Merkle-ს ხელმოწერის სისტემა საშუალებას გვაძლევს ხელი მოვაწეროთ სხვადასხვა შეტყობინებას ერთი და იგივე ხელმოწერის გასაღებით. აღნიშნული სისტემა იყენებს ერთჯერად ხელმოწერებს და ბინარულ ხეს, რომლის ფუძე (root) არის public-key .

გასაღების გენერაცია: მერკლეს ხის სიმაღლე უნდა იყოს H >= 2 . ალგორითმი ითვალისწინებს ხელმოწერისა და დადასტურების გასაღებების გენერაციას: Xi, Yi, 0<=i<=2H . X_i არის ხელმოწერის გასაღები, ხოლო Y_i არის დადასტურების გასაღები. იმისათვის, რომ მივიღოთ მშობელი კვანძი, გვჭირდება 2 წინა კვანძი(შვილი) გავაერთიანოთ და მოვახდინოთ ჰეშირება. მაგალითად, a[1,0]=h(a[0,0] || a[0,1]) .

შეტყობინების ხელმოწერა: იმისათვის, რომ მოვახდინოთ შეტყობინების ხელმოწერა, ვახდენთ მის ტრანსფორმაციას n ზომამდე ჰეშირების საშუალებით. h(m) = hash , იმისათვის, რომ მოხდეს შეტყობინების ხელმოწერა, უნდა გამოვიყენოთ ნებისმიერი ერთჯერადი X_{any} გასაღების, ერთჯერადი ხელმოწერისა და verification გასაღების Signature = (sig || any || Y_{any} || $auth_0$,…, $auth_{H-1}$)

ხელმოწერის დადასტურება: იმისათვის, რომ მოხდეს ხელმოწერის დადასტურება საჭიროა ერთჯერადი ხელმოწერის შემოწმება verification გასაღების დახმარებით. თუ გაივლის დადასტურება, მაშინ ხდება ყველა a[i, j] -ის გამოთვლა auth , index , any , Y_{any} -ის გამოყენებით. თუ ხის ფუძე (root) ემთხვევა public-key - ს, მაშინ ხელმოწერა სწორია. ქვემოთ არის მოყვანილი ალგორითმის რეალიზაციის ფსევდო კოდი:

1. Importing necessary libs
2. Define class
3. Defining alt_hashes(hashes) method
4. Set list arr
5. If hashes == , raise Exception
6. Foreach loop
    6.1. Sorting hashes and appending into arr
7. Length_of_block == length of arr
8. While loop, if length is odd, copy last element in list

8.1. A=Append it into arr list

9. Set list another_arr
10. Foreach loop
    10.1. For loop with range from 0 to length of arr and iteration by 2
        10.1.1. Define variable with sha512() value
        10.1.2. Hash elements that are in arr list
        10.1.3. Append them into new another_arr list
        10.1.4. Return this list in hex
11. Set list hash_arr
12. Foreach loop
    12.1. Generate Hex and put it into hash_arr list
13. Create message put it in st variable
14. Convert st value in binary
15. First_secret_key = hash_arr[0]
16. Second_secret_key = hash_arr[1]
17. Generate one-time signature
    17.1. If st == 0
        17.1.1. Choose First_secret_key bit
    17.2. Else
        17.2.1. Choose Second _secret_key bit
18. First_pub_key = hash(hash_arr[0])
19. Second_pub_key = hash (hash_arr[1])
20. Encryption
    20.1. Concatenate one-time signature with message's hash
21. Verification of one-time signature
    21.1. If bit of one-time signature == 0
        21.1.1. Compare with First_secret_key bit
    21.2. Else
        21.2.1. Compare with Second _secret_key bit
22. Verification of signature
    22.1. Concatenate siblings with each other
    22.2. If this equals to public key

22.2.1. Sign is correct

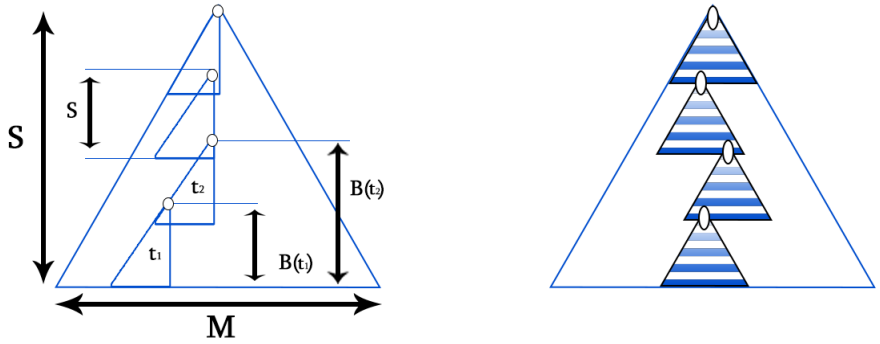
22.3. Else

Sign is not correct

იმისათვის, რომ მომხდარიყო Merkle-ს ალგორითმის ეფექტურობის გაუმჯობესება, წარმოდგენილი იყო Fractal Merkle ალგორითმი.

3.5 Fractal Merkle ალგორითმი

დავუშვათ მოცემული გვაქვს Merkle-ს ხე, რომლის სიმაღლე არის S. თავიდან ვირჩევთ a ქვეხეს (subtree), რომლის სიმაღლე არის s < S . დავუშვათ, ხეში n კვანძის სიმაღლე არის ტოლი n - დან ხის ფოთლამდე path - ის სიგრძის (შესაბამისად ხის ფოთლის სიმაღლე არის 0). დავუშვათ, გვაქვს n კვანძი, რომლის სიმაღლე არის s . აქვე ვახდენთ h ქვეხის წარდგენას, რომელიც ამ ხეში არის უნიკალური და რომლის სიმაღლე არის s - ი. სიმარტივისთვის დავუშვათ, რომ s არის S - ის გამყოფი და ასევე დავუშვათ, რომ სიხშირე ტოლია ქვეხეების დონეების: R = S / s . ანუ s ქვეხე არის j - ურ დონეზე, როდესაც მისი სიმაღლე ტოლია js , სადაც $j \in \{1, 2, \dots, S\}$. აქედან გამომდინარე, j - ური ხე შედგება s - ური ქვეხეების stack-ისგან და თუ ყველა j < R , მაშინ j - ური ხის ფუძე არის j + 1 ხის ფოთოლი. ეს ყველაფერი არის ნაჩვენები ქვემოთ მოყვანილ ნახაზზე:



a) b)

ნახაზი N 10. ქვეხეები (a) და s - ური ქვეხეების სტეკი(b)

ნახაზი 10. a) Merkle-ს ხის სიმაღლე არის S , ხოლო ფოთლების რაოდენობა არის M = 2S . ყოველი ქვეხის სიმაღლე არის s . B(t1) და B(t2) არის t1 და t2 ქვეხეების სიმაღლის შესაბამისად.

ნახაზი 10. b) ხეების კვანძების შენახვის ნაცვლად, ხდება პატარა სიმრავლეების შენახვა.

3.6 Threads - ებზე დაფუძნებული ალგორითმი

ზემოთ აღნიშნული ალგორითმი არის მიჩნეული როგორც სტატიკური, რადგან ის არ არის დამოკიდებული პროცესორის ნაკადების რაოდენობაზე. ჩვენ მიერ წარმოდგენილი ალგორითმი იყენებს პროცესორის ნაკადებს.

გასაღების გენერაცია: ზომა უნდა იყოს H >= 2 იმისათვის, რომ მოხდეს ერთი

public key - ით 2H დოკუმენტის ხელმოწერა. აქვე ხდება ხელმოწერისა და დადასტურების გასაღებების გენერაცია: Xi, Yi, 0<=i<=2H . X_i არის ხელმოწერის გასაღები, ხოლო Y_i არის დადასტურების გასაღები.

იმისათვის, რომ მივიღოთ მშობელი კვანძი, საჭიროა გავაერთიანოთ 2 წინა კვანძი(შვილი) და მოვახდინოთ ჰეშირება; a[i, j] არის ხის კვანძები;

$$a[1,0]=h(a[0,0] \,||\, a[0,1]) . \qquad (40)$$

ხის გაყოფა ხდება პროცესორის ნაკადების რაოდენობაზე. ციკლში, რომლის სიგრძე ტოლია ნაკადების რაოდენობის, ვახდენთ მშობელი კვანძების გამოთვლას. დავუშვათ, გვაქვს t რაოდენობის ნაკადი და d კვანძი. d კვანძების რაოდენობას ვყოფთ t ნაკადების რაოდენობაზე: d / t . d / t კვანძები ეშვება ცალკეულ ნაკადებში. მშობელი კვანძები მიიღება (40) გამოსახულების მიხედვით. ხდება მიღებული სიმრავლეების კონკატენაცია და შემდგომ ხდება მათი t სიმრავლეებად დაყოფა. ეს პროცესი გრძელდება მანამ, სანამ არ მივიღებთ ხის ფუძეს. ხის ფუძე არის public key .

შეტყობინების ხელმოწერა: იმისათვის, რომ მოვახდინოთ შეტყობინების ხელმოწერა, ჰეშირების საშუალებით ვახდენთ მის ტრანსფორმაციას n ზომამდე. h(m) = hash. იმისათვის, რომ მოხდეს შეტყობინების ხელმოწერა, საჭიროა ნებისმიერი ერთჯერადი X_{any} გასაღების, ერთჯერადი ხელმოწერის, ერთჯერადი verification გასაღებისა და ყველა მეზობელი კვანძების გამოყენება.

$$\text{Signature} = (\text{sig} \,||\, \text{any} \,||\, Y_{any} \,||\, \text{auth}_0, \ldots, \text{auth}_{H-1}). \qquad (41)$$

ხელმოწერის დადასტურება: იმისათვის, რომ მოხდეს ხელმოწერის დადასტურება საჭიროა ერთჯერადი ხელმოწერის შემოწმება verification გასაღების დახმარებით. იმ შემთხვევაში თუ განხორციელდება ხელმოწერის დადასტურება, მაშინ ხდება ყველა a[i, j] ელემენტის გამოთვლა auth , index , any , Y_{any} გამოყენებით. თუ ხის ფუძე (root) ემთხვევა

public-key - ს, მაშინ ხელმოწერა სწორია. ქვემოთ მოყვანილია ალგორითმის რეალიზაციის ფსევდო კოდი:

1. Importing necessary libs
2. Set queue q
3. Define class
4. Defining alt_hashes(hashes) method
5. Set list arr
6. If hashes == , raise Exception
7. Foreach loop
    7.1. sorting hashes and appending into arr
8. Length_of_block == length of arr
9. While loop, if length is odd, copy last element in list
    9.1. append it into arr list
10. Set list another_arr
11. Foreach loop
    11.1. For loop with range from 0 to length of arr and iteration by 2
        11.1.1. Define variable with sha512() value
        11.1.2. Hash elements that are in arr list
        11.1.3. Apennd them into new another_arr list
        11.1.4. put hex into q
12. Set list hash_arr
13. Foreach loop
    13.1. Generate Hex and put it into hash_arr list
14. Create message put it in st variable
15. Convert st value in binary
16. hash_arr3 equals to hash_arr
17. length_of_arr equals to the length of hash_arr list
18. add first element of hash_arr to auth_list
19. While loop, Length_of_block > 1

19.1. If length_of_arr!= 2

19.1.1. halfLength equals to length_of_arr / 2

19.1.2. Define j equals to 0

19.1.3. Define new list hash_arr3

19.1.4. For loop which range is 2

19.1.4.1. Define new list new_hash_arr

19.1.4.2. Call method in thread

19.1.4.3. j equals to halfLength

19.1.4.4. halfLength equals to length_of_arr

19.1.4.5. join all threads

19.1.4.6. append result from queue to hash_arr3

19.1.5. reset thread_hash_arr list

19.1.6. For loop which range is length of hash_arr3

19.1.6.1. concatenate hash_arr3's elements

19.1.7. Append them to auth_list

19.1.8. Reset hash_arr3

19.1.9. Reset queue

19.1.10. length_of_arr equals to length_of_arr / 2

19.2. else

19.2.1. call thread_method

19.2.2. reset thread_hash_arr

19.2.3. reset queue

19.2.4. set value of thread_hash_arr[0] to thread_pub variable

20. First_secret_key = hash_arr[0]
21. Second_secret_key = hash_arr[1]
22. Generate one-time signature

22.1. If st == 0

22.1.1. Choose First_secret_key bit

22.2. Else

22.2.1. Choose Second _secret_key bit

23. First_pub_key = hash(hash_arr[0])
24. Second_pub_key = hash (hash_arr[1])

25. Encryption
    25.1. Concatenate one-time signature with message's hash
26. Verification of one-time signature
    26.1. If bit of one-time signature == 0
        26.1.1. Compare with First_secret_key bit
    26.2. Else
        26.2.1. Compare with Second _secret_key bit
27. Verification of signature
    27.1. Concatenate siblings with each other
    27.2. If this equals to public key
        27.2.1. Sign is correct
    27.3. Else

Sign is not correct

3.7 ანალიზი

ჩვენს მიერ მიღებული ალგორითმი შევადარეთ კლასიკურ ალგორითმს. ტესტი ჩატარებული იქნა ისეთ კომპიუტერზე, რომლის პროცესორსაც გააჩნდა 2 ნაკადი. შეტყობინების სიგრძე კი იყო 128 ბიტი.

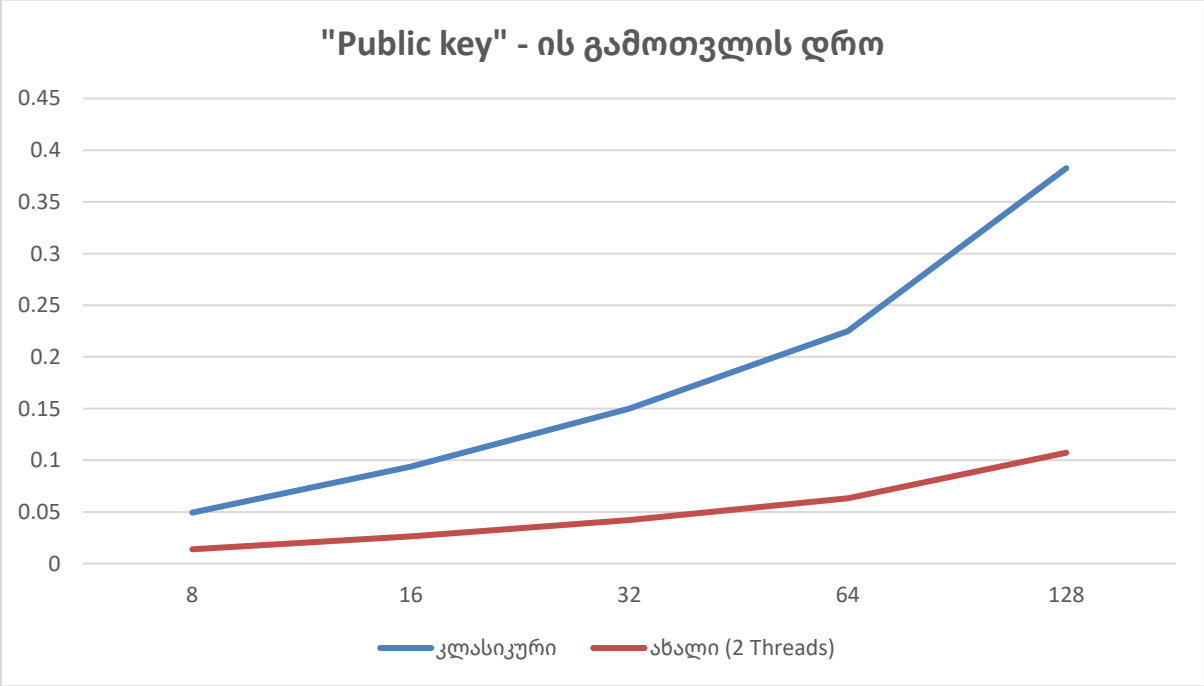
კლასიკური ალგორითმის შედეგები:

1. გასაღების გენერაციის დრო: 0.049351 წამი
2. ხელმოწერის დრო: 0.0002425 წამი
3. დადასტურების დრო: 0.0038651 წამი

Threads - ებზე დაფუძნებული ალგორითმის შედეგები:

1. გასაღების გენერაციის დრო: 0.013841 წამი
2. ხელმოწერის დრო: 0.0002425 წამი
3. დადასტურების დრო: 0.0038651 წამი

ამ ექსპერიმენტიდან გამომდინარე, შეგვიძლია ვნახოთ, რომ წარმოდგენილი ალგორითმი კლასიკურ ალგორითმზე არის 3,57- ჯერ სწრაფი.



გრაფიკი 1. Public key-ის გამოთვლის დრო

3.8 Merkle-ს კლასიკური ალგორითმის, ფრაქტალური ალგორითმის და პროცესორის ნაკადებზე დაფუძნებული ალგორითმის ოპერაციების დათვლა.

კლასიკური ალგორითმის საშუალო ხარჯები(Average costs). ხეში ყოველი კვანძი საბოლოო ჯამში არის authentication path - ის ნაწილი. დავუშვათ, გვაქვს 2^{H-h} მარჯვენა(შესაბამისად, მარცხენა) კვანძები h სიმაღლის ხეში. თუ დავითვლით ყოველ კვანძს დამოუკიდებლად, ყველა მათგანის საფასური იქნება 2^{h+1} - 1 ოპერაცია. რომ შევაჯამოთ გვექნება 2^{H+1} = 2N ოპერაცია. ყოველი h ($0 \leq h \leq H$) სიმაღლისთვის, ყველა საფასურის დაჯამებისას, საშუალოდ გვექნება 2H = 2log(N) სავალდებულო ოპერაცია.

ალგორითმის შედეგის გამოთვლის ფაზა შედგება N ციკლისგან, თითო ფოთლისთვის გვაქვს $s \in \{0, \dots, N-1\}$. ყოველი s ციკლის დროს, authentication path-ი s - ური ფოთლისთვის შედეგი არის $AUTH_i, i = 0, \dots, H-1$.

ფრაქტალური ალგორითმის დრო. იქედან გამომდინარე, რომ გვაქვს R - 1 რაოდენობის ხე, საერთო გამოთვლითი საფასური ერთი ციკლისთვის იქნება:

$$T_{max} = 2(R-1) < 2S/s. \qquad (42)$$

1. Set l = 0.
2. Output Authentication Path for leaf number l.
3. Next Subtree For each $j \in \{1, 2, \dots, S\}$ for which $EXISTS_j$ is no longer needed, i.e, l = 0 (mod 2^{sj}):
    a. Remove Pebbles in $EXIST_j$.
    b. Rename tree $DESIRE_j$ as tree $EXIST_j$.
    c. Create new, empty tree $DESIRE_j$ (if $l + 2^{sj} < 2^S$).
4. Grow Subtrees For each $j \in \{1, 2, \dots, s\}$: Grow tree $DESIRE_j$ by applying 2 units to the modified treehash algorithm (unless $DESIRE_j$ is completed).
5. Increment l and loop back to step 2 (while $l < 2^H$).

ფრაქტალური ალგორითმის სივრცე (Space). აღნიშნული ალგორითმისთვის საჭირო დისკური მეხსიერების ზომა შეგვიძლია დავადგინოთ არსებული ქვეხეების, desired ქვეხეების და ე.წ. tails - ების დახმარებით.

ვთქვათ მოცემული გვაქვს R არსებული ქვეხეები და R - 1 desired ქვეხეები და ყოველი მათგანი შედგება 2^{s+1}-2 pebbles - გან. დამატებით tail - თან ასოცირებული desired ქვეხე j > 1 დონეზე(level) შეიცავს $s \cdot j + 1$ pebbles - ებს. აქედან გამომდინარე, გვაქვს:

$$SPACE_{max} \leq (2R-1)\,(2^{s+1}-2) + R-2 + s\,(R-2)\,(R-1)\,/\,2. \qquad (43)$$

ყველაზე ცუდი შედეგისათვის გვექნება:

$$SPACE_{max} < 2\,R\,2^{s+1} + S\,R\,/\,2. \qquad (44)$$

ნაკადებზე დაფუძნებული ალგორითმის ოპერაციების რაოდენობა.. ახლა დავითვალოთ მიმდევრობითი ოპერაციების რაოდენობა ახალ ალგორითმში და ასევე შევადაროთ ის კლასიკურ ალგორითმს. ქვემოთ არის მოყვანილი ცხრილები, სადაც ნაჩვენებია ოპერაციების რაოდენობა კლასიკური და ახალი ალგორითმებისთვის, სხვადასხვა რაოდენობის ნაკადების მიხედვით.

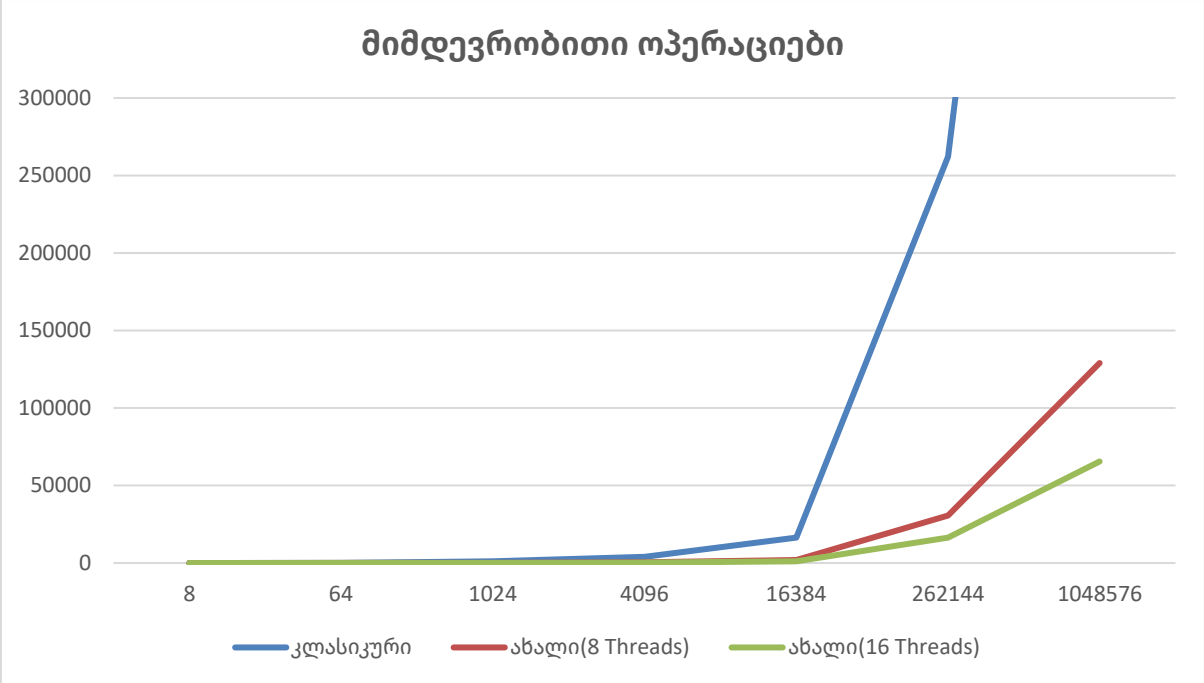
ახალი ალგორითმი 8 thread - იან პროცესორში	
კვანძების რაოდენობა	მიმდევრობითი ოპერაციების რაოდენობა
8	3
64	10
1024	130
4096	514
16384	2050
262144	30722
1048576	129026

ახალი ალგორითმი 16 thread - იან პროცესორში	
კვანძების რაოდენობა	მიმდევრობითი ოპერაციების რაოდენობა
8	3
64	7
1024	67
4096	259
16384	1027
262144	16387
1048576	65539

ძველი ალგორითმი	
კვანძების რაოდენობა	მიმდევრობითი ოპერაციების რაოდენობა
8	7
64	63
1024	1023
4096	4095
16384	16383
262144	262143
1048576	1048575

აღნიშნულ მონაცემებზე დაყრდნობით შეგვიძლია მივიღოთ ახალი ალგორითმისთვის მიმდევრობითი ოპერაციების რაოდენობის გამოსათვლელი ფორმულა. დავუშვათ t არის ნაკადების რაოდენობა და n არის კვანძების რაოდენობა, გამოვთვალოთ O :

$$O = n / t + \log_2 t/2, \text{ თუ } n = t, \text{ მაშინ } t = t / 2 \qquad (45)$$



გრაფიკი 2. მიმდევრობითი ოპერაციები

მიღებული შედეგები

დისერტაციაში მიღებული შედეგების ანალიზის საფუძველზე დგინდება რომ, ჩვენს მიერ შემუშავებული ალგორითმის გამოთვლის სიჩქარე, რომელიც ეფუძნება Merkle-ს კლასიკურ ალგორითმს, არის რამდენჯერმე უფრო სწრაფი სხვა არსებულ ალგორითმებთან შედარებით.

წარმოდგენილი ალგორითმი საშუალებას იძლევა სრულად იქნას გამოყენებული თანამედროვე პროცესორის რესურსები, პროცესორის ნაკადებზე დაყრდნობით უზრუნველყოფილი იქნას პარალელური გამოთვლები, რის შედეგადაც ხორციელდება ოპერაციების რაოდენობისა და გამოთვლების საწარმოებლად საჭირო დროის მნიშვნელოვნად შემცირება.

დისერტაციის ფარგლებში შესრულდა შემდეგი სახის ამოცანები:

1. კრიპტოგრაფიის არსებული სიმეტრიული და ასიმეტრიული ალგორითმების ანალიზი.
2. დღესდღეისობით არსებული კლასიკური (ფართოდ გამოყენებადი) კრიპტოგრაფიის ალგორითმების პოსტ-კვანტურ ალგორითმებთან შედარება.
3. ნაშრომში განხილულია ახალი ალგორითმის აგების პროცესი, რომელიც ეფუძნება Merkle-ს ხეს და ამავდროულად წარმოადგენს პარალელურ ალგორითმს.
4. განხორციელდა მიღებული ალგორითმის ეფექტურობის ანალიზი და მისი სისწრაფის (ოპერაციების რაოდენობის) შედარება არსებულ ალგორითმებთან.

შემუშავებული ალგორითმის ეფექტურობა დასტურდება ექსპერიმენტების შედეგებით. კერძოდ, 8 ნაკადიანი პროცესორის შემთხვევაში ოპერაციების რაოდენობა შემცირდა 2.3 ჯერ, ხოლო 16 ნაკადიანი პროცესორის შემთხვევაში ოპერაციების რაოდენობა შემცირდა 9 ჯერ. აღნიშნული ალგორითმის დახმარებით მნიშვნელოვნად შემცირდება შიფრაცია/ვერიფიკაციის დროც. ექსპერიმენტების პროცესში, გამოთვლების წარმოებისას 2 ნაკადიან პროცესორზე ახალი ალგორითმი შესრულდა 3,57 ჯერ უფრო სწრაფად ვიდრე კლასიკური. პრაქტიკაში გვხვდება უფრო მეტი ნაკადის მქონე

პროცესორები, სადაც ჩვენს მიერ მიღებული ალგორითმის შესრულების სიჩქარე იქნება კიდევ უფრო სწრაფი.

გამოყენებული ლიტერატურა

1. Emily Chung (2016) Google, NASA put big money on D-Wave's quantum computer https://www.cbc.ca/news/science/d-wave-quantum-1.3525566
2. Gary Fowler 2020, When Will Quantum Computers Impact Our Day-To-Day?, https://www.forbes.com/sites/forbesbusinessdevelopmentcouncil/2021/04/28/when-will-quantum-computers-impact-our-day-to-day/?sh=4e0e40ca43d9
3. Aciic, O., Schindler, W., Koc, C., (2007). *Cache based remotetiming attack on the AES*. San Francisco: CT-RSA.
4. Bellare, M., Rogaway, P. (1994). *Optimal asymmetric encryption. In EUROCRYPT '94*. Berlin: Springer-Verlag.
5. Bernstein, D. J. (2004). *Cache-timing attacks on AES*. http://cr.yp.to/papers.html. მოძიებულია 2017 წლის 23 აპრილს.
6. Biham, E., Shamir, A. (1993). Differential Cryptanalysis of the Data Encryption Standard. Berlin: Springer-Verlag.
7. Bleichenbacher, D. (1998). Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. Berlin: Springer-Verlag.
8. Bonneau, J., Mironov, I. (2006). *Cache-collision timing attacks against AES*. https://link.springer.com/chapter/10.1007/11894063_16. მოძიებულია 2017 წლის 23 აპრილს.
9. Brumley, B., Hakala, R. (2009). *Cache-timing template attacks*. Berlin: Springer.
10. Boneh, D., DeMillo, R., Lipton, R. (1997). *On the importance of checking cryptographic protocols for faults*. Berlin: Springer-Verlag.
11. Boneh, D., Durfee, G. (1998). *New results on cryptanalysis of low private exponent RSA*. Berlin: Springer-Verlag.
12. Buchmann, J., Dahmen, E., Schneider, M. (2008). *Merkle tree traversal revisited*. Berlin: Springer
13. Dahmen, E., Okeya, K., Takagi, T., Vuillaume.C (2008). *Digital Signatures out of Second-Preimage Resistant Hash Functions*.
14. https://www.cdc.informatik.tu-darmstadt.de/~dahmen/papers/DOTV08.pdf. მოძიებულია 2017 წლის 11 მაისს.

15. Dahmen, E., Dring, M., Klintsevich, E., Buchmann, J. (2006). *CMSS - an improved merkle signature scheme. Progress in Cryptology.* Berlin: Springer.
16. Deavours, A., Louis, K. (1985). *Machine Cryptography and Modern Cryptanalysis*. Norwood, MA: Artech House.
17. Gagnidze, A., Iavich, M., Iashvili, G. (2016). *Some Aspects of Post-Quantum Cryptosystems*. http://eurasianpublications.com/Eurasian-Journal-of-Business-and-Management/Vol.-5-No.1-2017/EJBM-2.pdf. მოძიებულია 2017 წლის 30 მარტს.
18. Gagnidze, A., Iavich, M., Iashvili G. (2016). *Attacks on post-quantum cryptosystems.* Kiev: Abstract Book.
19. Gagnidze, A., Iavich, M., Iashvili, G. (2017). *Improved version of Merkle crypto system*. http://web.snauka.ru/issues/2017/05/81949. მოძიებულია 2017 წლის 30 მარტს.
20. Gallais, J.F., Kizhvatov, I., Tunstall, M. (2010). *Improved tracedriven cache-collision attacks against embedded AES implementations.* http://eprint.iacr.org/2010/408.pdf. მოძიებულია 2017 წლის 23 აპრილს
21. Iavich, M.P., Isaev, P.D. (2014). *Problems associated with the creation of the own cryptosystems Modern scientific researches and innovations*. http://web.snauka.ru/en/issues/2014/04/33150 მოძიებულია 2017 წლის 30 მარტს.
22. Neve, M., Seifert, J.P., Wang, Z. (2006). *A refined look at Bernstein's AES side-channel analysis.* Taipei, Taiwan: ASIACCS.
23. Neve, M., Seifert, J.P. (2006). *Advances on access-driven cache attacks on AES*. Berlin: Springer-Verlag.
24. Osvik, D. A., Shamir, A., Tromer, E. (2006). *Cache attacks and countermeasures: The case of AES*. San Francisco: CT-RSA.
25. Percival, C. (2006). *Cache missing for fun and profit*. http://www.daemonology.net/hyperthreading-considered-harmful/. მოძიებულია 2017 წლის 23 აპრილს.
26. Pomerance, C. (2001). *A tale of two sieves*. http://www.ams.org/notices/199612/pomerance.pdf. მოძიებულია 2017 წლის 23 აპრილს
27. Rivest, R.L., Shamir A., Adleman, L.M (2004). *A method for obtaining digital signatures and public key cryptosystems.* New York: Commun. of the ACM.

28. Rivest, R.L., Shamir, A., Adleman, L.M., (1978), *A method for obtaining digital signatures and public-key cryptosystems*. New York: Communications of the ACM.
29. Schneier, B. (1996). *Applied Cryptography*. Hoboken, New Jersey: John Wiley & Sons.
30. Tsunoo, Y., Saito, T., Suzaki, T., Shigeri, M., Miyauchi, H., (2003). *Cryptanalysis of DES implemented on computers with cache.* Berlin: Springer.
31. Wiener, M. (1990). Cryptanalysis of short RSA secret exponents. IEEE Transactions on Information Theory. Piscataway, NJ: IEEE Press.
32. Mollin, R.A. An Introduction to Cryptography (2nd ed.). Chapman and Hall/CRC. https://doi.org/10.1201/9781420011241, 2006
33. Hans DelfsHelmut Knebl. Introduction to Cryptography, Part of the Information Security and Cryptography book series (ISC), Springer, 2015
34. Dan Boneh and Victor Shoup, A Graduate Course in Applied Cryptography, http://toc.cryptobook.us/, 2020
35. Jonathan Katz, Yehuda Lindell, Introduction to Modern Cryptography, Chapman and Hall/CRC, 2020
36. B. Schneier, Applied Cryptography, 2nd ed., John Wiley & Sons, 1996
37. Deavours: Cipher A. Deavours and Louis Kruh, ``Machine Cryptography and Modern Cryptanalysis'', Artech House, 1985
38. P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," Proceedings 35th Annual Symposium on Foundations of Computer Science, 1994, pp. 124-134, doi: 10.1109/SFCS.1994.365700.
39. Z. Qu, Z. Li, G. Xu, S. Wu and X. Wang, "Quantum Image Steganography Protocol Based on Quantum Image Expansion and Grover Search Algorithm," in IEEE Access, vol. 7, pp. 50849-50857, 2019, doi: 10.1109/ACCESS.2019.2909906.
40. Toyama, F.M., van Dijk, W. & Nogami, Y. Quantum search with certainty based on modified Grover algorithms: optimum choice of parameters. Quantum Inf Process 12, 1897–1914 (2013). https://doi.org/10.1007/s11128-012-0498-0
41. Jaques S., Naehrig M., Roetteler M., Virdia F. (2020) Implementing Grover Oracles for Quantum Key Search on AES and LowMC. In: Canteaut A., Ishai Y. (eds) Advances in Cryptology-EUROCRYPT 2020. EUROCRYPT 2020. Lecture Notes in Computer Science, vol 12106. Springer, Cham. https://doi.org/10.1007/978-3-030-45724-2_10

42. Bernstein D.J., Lange T., Peters C. (2008) Attacking and Defending the McEliece Cryptosystem. In: Buchmann J., Ding J. (eds) Post-Quantum Cryptography. PQCrypto 2008. Lecture Notes in Computer Science, vol 5299. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-88403-3_3
43. Biswas B., Sendrier N. (2008) McEliece Cryptosystem Implementation: Theory and Practice. In: Buchmann J., Ding J. (eds) Post-Quantum Cryptography. PQCrypto 2008. Lecture Notes in Computer Science, vol 5299. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-88403-3_4
44. Canteaut A., Sendrier N. (2000) Cryptanalysis of the Original McEliece Cryptosystem. In: Ohta K., Pei D. (eds) Advances in Cryptology — ASIACRYPT'98. ASIACRYPT 1998. Lecture Notes in Computer Science, vol 1514. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-49649-1_16
45. Loidreau P. (2000) Strengthening McEliece Cryptosystem. In: Okamoto T. (eds) Advances in Cryptology — ASIACRYPT 2000. ASIACRYPT 2000. Lecture Notes in Computer Science, vol 1976. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-44448-3_45
46. Bernstein D.J. (2009) Introduction to post-quantum cryptography. In: Bernstein D.J., Buchmann J., Dahmen E. (eds) Post-Quantum Cryptography. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-88702-7_1
47. Bernstein D.J. et al. (2015) SPHINCS: Practical Stateless Hash-Based Signatures. In: Oswald E., Fischlin M. (eds) Advances in Cryptology -- EUROCRYPT 2015. EUROCRYPT 2015. Lecture Notes in Computer Science, vol 9056. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-662-46800-5_15
48. Rohde S., Eisenbarth T., Dahmen E., Buchmann J., Paar C. (2008) Fast Hash-Based Signatures on Constrained Devices. In: Grimaud G., Standaert FX. (eds) Smart Card Research and Advanced Applications. CARDIS 2008. Lecture Notes in Computer Science, vol 5189. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-85893-5_8
49. McGrew D., Kampanakis P., Fluhrer S., Gazdag SL., Butin D., Buchmann J. (2016) State Management for Hash-Based Signatures. In: Chen L., McGrew D., Mitchell C. (eds) Security Standardisation Research. SSR 2016. Lecture Notes in Computer Science, vol 10074. Springer, Cham. https://doi.org/10.1007/978-3-319-49100-4_11
50. N. Sendrier, "Code-Based Cryptography: State of the Art and Perspectives," in IEEE Security & Privacy, vol. 15, no. 4, pp. 44-50, 2017, doi: 10.1109/MSP.2017.3151345.

51. Bernstein D.J., Chou T., Schwabe P. (2013) McBits: Fast Constant-Time Code-Based Cryptography. In: Bertoni G., Coron JS. (eds) Cryptographic Hardware and Embedded Systems - CHES 2013. CHES 2013. Lecture Notes in Computer Science, vol 8086. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-40349-1_15
52. Nilsson, A., Johansson, T., & Stankovski Wagner, P. (2018). Error Amplification in Code-based Cryptography. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2019(1), 238-258. https://doi.org/10.13154/tches.v2019.i1.238-258
53. Chou T. (2016) QcBits: Constant-Time Small-Key Code-Based Cryptography. In: Gierlichs B., Poschmann A. (eds) Cryptographic Hardware and Embedded Systems-CHES 2016. CHES 2016. Lecture Notes in Computer Science, vol 9813. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-662-53140-2_14
54. Ingo von Maurich and T. Güneysu, "Lightweight code-based cryptography: QC-MDPC McEliece encryption on reconfigurable devices," 2014 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2014, pp. 1-6, doi: 10.7873/DATE.2014.051.
55. Finiasz M., Sendrier N. (2009) Security Bounds for the Design of Code-Based Cryptosystems. In: Matsui M. (eds) Advances in Cryptology-ASIACRYPT 2009. ASIACRYPT 2009. Lecture Notes in Computer Science, vol 5912. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-10366-7_6
56. J. Ding and A. Petzoldt, "Current State of Multivariate Cryptography," in IEEE Security & Privacy, vol. 15, no. 4, pp. 28-36, 2017, doi: 10.1109/MSP.2017.3151328.
57. Ding J., Yang BY. (2009) Multivariate Public Key Cryptography. In: Bernstein D.J., Buchmann J., Dahmen E. (eds) Post-Quantum Cryptography. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-88702-7_6
58. Gouget A., Patarin J. (2006) Probabilistic Multivariate Cryptography. In: Nguyen P.Q. (eds) Progress in Cryptology - VIETCRYPT 2006. VIETCRYPT 2006. Lecture Notes in Computer Science, vol 4341. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11958239_1
59. Huang YJ., Liu FH., Yang BY. (2012) Public-Key Cryptography from New Multivariate Quadratic Assumptions. In: Fischlin M., Buchmann J., Manulis M. (eds) Public Key Cryptography-PKC 2012. PKC 2012. Lecture Notes in Computer Science, vol 7293. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-30057-8_12

60. J. Wang, L. Cheng and T. Su, "Multivariate Cryptography Based on Clipped Hopfield Neural Network," in IEEE Transactions on Neural Networks and Learning Systems, vol. 29, no. 2, pp. 353-363, Feb. 2018, doi: 10.1109/TNNLS.2016.2626466.
61. J. Buchmann, K. Lauter and M. Mosca, "Postquantum Cryptography—State of the Art," in IEEE Security & Privacy, vol. 15, no. 4, pp. 12-13, 2017, doi: 10.1109/MSP.2017.3151326.
62. K. Lauter, "The advantages of elliptic curve cryptography for wireless security," in IEEE Wireless Communications, vol. 11, no. 1, pp. 62-67, Feb. 2004, doi: 10.1109/MWC.2004.1269719.
63. Hankerson D., López Hernandez J., Menezes A. (2000) Software Implementation of Elliptic Curve Cryptography over Binary Fields. In: Koç Ç.K., Paar C. (eds) Cryptographic Hardware and Embedded Systems — CHES 2000. CHES 2000. Lecture Notes in Computer Science, vol 1965. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-44499-8_1
64. Neve M., Seifert JP. (2007) Advances on Access-Driven Cache Attacks on AES. In: Biham E., Youssef A.M. (eds) Selected Areas in Cryptography. SAC 2006. Lecture Notes in Computer Science, vol 4356. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-74462-7_11
65. D. Gullasch, E. Bangerter and S. Krenn, "Cache Games -- Bringing Access-Based Cache Attacks on AES to Practice," 2011 IEEE Symposium on Security and Privacy, 2011, pp. 490-505, doi: 10.1109/SP.2011.22.
66. Irazoqui G., Inci M.S., Eisenbarth T., Sunar B. (2014) Wait a Minute! A fast, Cross-VM Attack on AES. In: Stavrou A., Bos H., Portokalidis G. (eds) Research in Attacks, Intrusions and Defenses. RAID 2014. Lecture Notes in Computer Science, vol 8688. Springer, Cham. https://doi.org/10.1007/978-3-319-11379-1_15
67. E. Carmon, J. Seifert and A. Wool, "Photonic side channel attacks against RSA," 2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), 2017, pp. 74-78, doi: 10.1109/HST.2017.7951801.
68. Gülmezoğlu B., İnci M.S., Irazoqui G., Eisenbarth T., Sunar B. (2015) A Faster and More Realistic Flush+Reload Attack on AES. In: Mangard S., Poschmann A. (eds) Constructive Side-Channel Analysis and Secure Design. COSADE 2015. Lecture Notes in Computer Science, vol 9064. Springer, Cham. https://doi.org/10.1007/978-3-319-21476-4_8
69. Bonneau J., Mironov I. (2006) Cache-Collision Timing Attacks Against AES. In: Goubin L., Matsui M. (eds) Cryptographic Hardware and Embedded Systems - CHES 2006. CHES 2006.

Lecture Notes in Computer Science, vol 4249. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11894063_16

70. Acıiçmez O., Koç Ç.K. (2006) Trace-Driven Cache Attacks on AES (Short Paper). In: Ning P., Qing S., Li N. (eds) Information and Communications Security. ICICS 2006. Lecture Notes in Computer Science, vol 4307. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11935308_9
71. Bennett, C.H., Bessette, F., Brassard, G. et al. Experimental quantum cryptography. J. Cryptology 5, 3–28 (1992). https://doi.org/10.1007/BF00191318
72. C. Elliott, "Quantum cryptography," in IEEE Security & Privacy, vol. 2, no. 4, pp. 57-61, July-Aug. 2004, doi: 10.1109/MSP.2004.54.
73. Bennett C.H., Brassard G. (1985) An Update on Quantum Cryptography. In: Blakley G.R., Chaum D. (eds) Advances in Cryptology. CRYPTO 1984. Lecture Notes in Computer Science, vol 196. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-39568-7_39
74. E. O. Brigham and R. E. Morrow, "The fast Fourier transform," in IEEE Spectrum, vol. 4, no. 12, pp. 63-70, Dec. 1967, doi: 10.1109/MSPEC.1967.5217220.
75. W. T. Cochran et al., "What is the fast Fourier transform?," in Proceedings of the IEEE, vol. 55, no. 10, pp. 1664-1674, Oct. 1967, doi: 10.1109/PROC.1967.5957.
76. Nussbaumer H.J. (1981) The Fast Fourier Transform. In: Fast Fourier Transform and Convolution Algorithms. Springer Series in Information Sciences, vol 2. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-662-00551-4_4
77. Mosca M., Ekert A. (1999) The Hidden Subgroup Problem and Eigenvalue Estimation on a Quantum Computer. In: Williams C.P. (eds) Quantum Computing and Quantum Communications. QCQC 1998. Lecture Notes in Computer Science, vol 1509. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-49208-9_15
78. P. Sen, "Random measurement bases, quantum state distinction and applications to the hidden subgroup problem," 21st Annual IEEE Conference on Computational Complexity (CCC'06), 2006, pp. 14 pp.-287, doi: 10.1109/CCC.2006.37.
79. Childs A.M., Harrow A.W., Wocjan P. (2007) Weak Fourier-Schur Sampling, the Hidden Subgroup Problem, and the Quantum Collision Problem. In: Thomas W., Weil P. (eds) STACS 2007. STACS 2007. Lecture Notes in Computer Science, vol 4393. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-70918-3_51

80. Radhakrishnan, J., Rötteler, M. & Sen, P. Random Measurement Bases, Quantum State Distinction and Applications to the Hidden Subgroup Problem. Algorithmica 55, 490–516 (2009). https://doi.org/10.1007/s00453-008-9231-x
81. R. Jozsa, "Quantum factoring, discrete logarithms, and the hidden subgroup problem," in Computing in Science & Engineering, vol. 3, no. 2, pp. 34-43, March-April 2001, doi: 10.1109/5992.909000.
82. P. W. Shor. (1994) Algorithms for quantum computation: discrete logarithms and factoring, Proceedings 35th Annual Symposium on Foundations of Computer Science pp. 124-134.
83. Buchmann J., Dahmen E., Ereth S., Hülsing A., Rückert M. (2011) On the Security of the Winternitz One-Time Signature Scheme In: Nitaj A., Pointcheval D. (eds) Progress in Cryptology-AFRICACRYPT 2011. Lecture Notes in Computer Science, vol 6737. Springer, Berlin, Heidelberg
84. Buchmann J., Dahmen E., Klintsevich E., Okeya K., Vuillaume C. (2007) Merkle Signatures with Virtually Unlimited Signature Capacity. In: Katz J., Yung M. (eds) Applied Cryptography and Network Security. ACNS 2007. Lecture Notes in Computer Science, vol 4521. Springer, Berlin, Heidelberg
85. R. Merkle. (1979) Secrecy, authentication and public key systems / A certified digital signature Ph.D. dissertation, Dept. of Electrical Engineering, Stanford University.
86. Aoki K., Guo J., Matusiewicz K., Sasaki Y., Wang L. (2009) Preimages for Step-Reduced SHA-2. In: Matsui M. (eds) Advances in Cryptology-ASIACRYPT 2009. ASIACRYPT 2009. Lecture Notes in Computer Science, vol 5912. Springer, Berlin, Heidelberg
87. Gagnidze A., Iavich M., Iashvili G. (2017) Analysis of Post Quantum Cryptography use in Practice, Bulletin of the Georgian National Academy of Sciences, vol. 11, no. 2, pp. 29-36.
88. Buchmann J., García L.C.C., Dahmen E., Döring M., Klintsevich E. (2006) CMSS-An Improved Merkle Signature Scheme. In: Barua R., Lange T. (eds) Progress in Cryptology - INDOCRYPT 2006. INDOCRYPT 2006. Lecture Notes in Computer Science, vol 4329. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11941378_25
89. Buchmann J., Dahmen E., Klintsevich E., Okeya K., Vuillaume C. (2007) Merkle Signatures with Virtually Unlimited Signature Capacity. In: Katz J., Yung M. (eds) Applied Cryptography and Network Security. ACNS 2007. Lecture Notes in Computer Science, vol 4521. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-72738-5_3

90. Shoufan and N. Huber, "A fast hash tree generator for Merkle signature scheme," Proceedings of 2010 IEEE International Symposium on Circuits and Systems, 2010, pp. 3945-3948, doi: 10.1109/ISCAS.2010.5537673.
91. El Bansarkhani R., Misoczki R. (2018) G-Merkle: A Hash-Based Group Signature Scheme from Standard Assumptions. In: Lange T., Steinwandt R. (eds) Post-Quantum Cryptography. PQCrypto 2018. Lecture Notes in Computer Science, vol 10786. Springer, Cham. https://doi.org/10.1007/978-3-319-79063-3_21
92. Buchmann J., García L.C.C., Dahmen E., Döring M., Klintsevich E. (2006) CMSS-An Improved Merkle Signature Scheme. In: Barua R., Lange T. (eds) Progress in Cryptology - INDOCRYPT 2006. INDOCRYPT 2006. Lecture Notes in Computer Science, vol 4329. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11941378_25
93. Iavich, M., Gagnidze, A., Iashvili, G., Hash based digital signature scheme with integrated TRNG, CEUR Workshop Proceedings, 2018
94. Szydlo M. (2004) Merkle Tree Traversal in Log Space and Time. In: Cachin C., Camenisch J.L. (eds) Advances in Cryptology - EUROCRYPT 2004. EUROCRYPT 2004. Lecture Notes in Computer Science, vol 3027. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-24676-3_32
95. Buchmann J., Dahmen E., Schneider M. (2008) Merkle Tree Traversal Revisited. In: Buchmann J., Ding J. (eds) Post-Quantum Cryptography. PQCrypto 2008. Lecture Notes in Computer Science, vol 5299. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-88403-3_5
96. Jakobsson M., Leighton T., Micali S., Szydlo M. (2003) Fractal Merkle Tree Representation and Traversal. In: Joye M. (eds) Topics in Cryptology — CT-RSA 2003. CT-RSA 2003. Lecture Notes in Computer Science, vol 2612. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-36563-X_21
97. D. Naor, A. Shenhav and A. Wool, "One-Time Signatures Revisited: Practical Fast Signatures Using Fractal Merkle Tree Traversal," 2006 IEEE 24th Convention of Electrical & Electronics Engineers in Israel, 2006, pp. 255-259, doi: 10.1109/EEEI.2006.321066.
98. Merkle R.C. (1990) A Certified Digital Signature. In: Brassard G. (eds) Advances in Cryptology — CRYPTO' 89 Proceedings. CRYPTO 1989. Lecture Notes in Computer Science, vol 435. Springer, New York, NY. https://doi.org/10.1007/0-387-34805-0_21