دانشگاه تهران

پردیس دانشکده های فنی

دانشکده مهندسی برق و کامپیوتر

گزارش کارآموزی

عنوان کارآموزی:

تسریع الگوریتم درهم سازی nexus با پیاده سازی سخت افزار بهینه

نام محل کارآموزی:

ازمایشگاه سیستم های نهفته چند هسته ای

نام و نام خانوادگی دانشجو:

عرفان ایروانی

شماره دانشجویی: ۸۱۰۱۹۷۴۶۲

نام استاد کارآموزی : دکتر مصطفی ارسالی صالحی

تاریخ انجام کارآموزی : تابستان ۱۴۰۰

فهرست مطالب

چکیده ۳

معرفی محل کارآموزی ۴

موارد انجام شده ۴

روند کلی کار ٤

پیاده سازی skein ٥

پیاده سازی threefish ٥

تابع Add ٦

تابع mixloop ٦

تابع permute ٦

پیاده سازی makesubkey ۷

پیاده سازیKeccak ۷

تابع Theta ۸

تابع Rho ۹

تابع pi ۹

تابع chi ۹

تابع Iota ۱۰

ورودی دادن ۱۱

پیشنهادات و ارزیابی ۱۱

مراجع ۱۲

چکیده

با توجه به گسترش استفاده عموما از ارز های دیجیتال و قابلیت ها و فرصت های بیشماری که این ارز ها ایجاد میکنند،نیاز به دانش کافی در این زمینه بیش از پیش احساس میشود.استخراج این رمز ارز ها نیاز به پردازش های محاسباتی دارد

در پردازش های محاسباتی، شتاب دهنده سخت افزاری برای بهره بردن از ویژگیهای سخت افزار خاص منظوره جهت اجرای کارامدتر توابع نرم افزاری استفاده می شود. هر گونه تغییر داده یا روال که می تواند محاسبه شود، می تواند تنها در نرم افزار در حال اجرا بر روی یک پردازنده عمومی، تنها در سخت افزار سفارشی یا در برخی ترکیب هر دو، انجام شود. با این همطراحی می توان در سخت افزار خاص منظوره و یا شتابدهنده عملیات را با سرعت بیشتری نسبت به نرم افزار و بر روی یک پردازنده رایانه های عمومی انجام داد.

از زمینه های مناسب برای استفاده از پیاده سازی سخت افزاری می توان به الگوریتم های درهم سازی موجود در شبکه بلاکچین (جهت افزایش کارایی) اشاره کرد. در شبکه بلاک چین اطلاعات به صورت بلاک ذخیره می شوند و برای تایید و رسمیت هر بلاک تولید شده، نیاز به ذخیره سازی یک هش از بلاک های قبل می باشد. این عمل موجب میشود که امکان دستکاری اطلاعات و تخریب آ نها به حداقل برسدوامنیت داد ه های ذخیره شده بالا برود

میباشد و روند کلی کار به این صورت است که کد های متن باز و قابل nexus الگوریتم انتخاب شده برای کارآموزی بنده دسترس را بررسی میکنیم و پس از حاصل کردن اطمینان از درستی عملکرد آن ها ، قسمت هایی که از نظر پردازشی نیازمند سخت افزاری جداگانه هستند را با زبان برنامه نویسی verilog پیاده سازی میکنیم

معرفی محل کارآموزی

دوره کارآموزی بنده به صورت مجازی برگذار شد. در هفته های ابتدایی هدف کلی یادگیری کلیت کار از قبیل درک الگوریتم بود. منتور کارآموزی بنده آقای رضا احمدی بودند و همواره نظارت vivado ها و فرایند ماینینگ و کار کردن با نرم افزار دقیق و منظمی روی کار داشتند و همواره در اسرع وقت به سوالات و اشکالات من پاسخ میدادند و به صورت کلی نظارت و حضور ایشان به شدت روی فرایند پروژه تاثیر مثبت داشت و باعث میشد از مستر اصلی منحرف نشویم. همچنین جلسات به صورت هفتگی با حضور دکتر صالحی و آقای احمدی برگذار میشد که پیشرفت ما در مسیر بررسی شده و اگر ایرادی وجود داشت برطرف میشد و همچنین کار هایی که تا هفته آینده باید انجام میشد تعیین میگشت. این جلسات نیز تاثیر بسیار مثبتی در روند اجرا کار ها داشت. به صورت کلی روند کارآموزی بنده بسیار خوب و منظم بود

موارد انجام شده

روند کلی کار

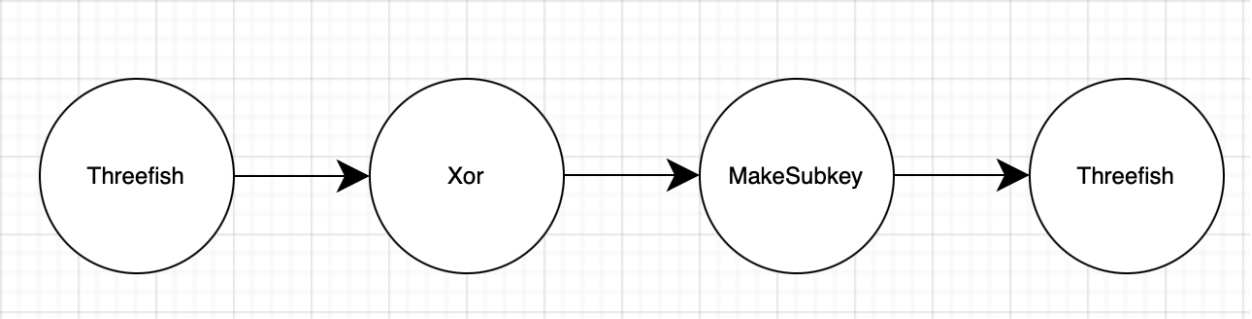
در ابتدا از بین کد های متن باز موجود یکی را انتخاب کرده و از صحت عملکرد آن اطمینان حاصل میکنیم(آدرس git کدی که انتخاب شده در پیوست ذکر گردیده) . پس از دانلود فایل وارد پوشه build میشویم و با دستورات .. cmake و . came —build برنامه را build میکنیم. در انتها در فال configue باید آدرس IP خود را وارد کنیم و پس از آن با دستور ./NexusMiner ../miner.conf -c برنامه اجرا میشود و مشاهده میکنیم که برنامه کار میکند و قابل اطمینان است

حال به بررسی توالی کد nexus minerمیپردازیم تا با روند کلی اجرای برنامه آشنا شویم.

در ابتدا شروع برنامه تابع set_block صدا زده میشود. در این تابع روی داده هایی که از ورودی گرفته میشود پردازش های اولیه انجام میشود و داده ها آماده برای پردازش اصلی میشوند. به دلیل اینکه این تابع در هر بار اجرا ۱ بار استفاده میشود، نیازی به پیاده سازی سخت افزاری آن نداریم و داده هایی که این تابع تولید میکند را به عنوان ورودی سخت افزار در نظر میگیریم. در ادامه داده های تابع set_blockوارد تابع run میشود که در یک حلقه while قرار دارد و نیاز به توان پردازشی بالایی برای اجرا آن داریم و هدفمان تولید کد سخت افزاری این قسمت میباشد. در این تابع در ابتدا با استفاده از ورودی ها الگوریتم هش skein اجرا میشود و ۱۰۲۴ بیت خروجی تولید میکند. سوس این خروجی وارد الگوریتم keccak شده و ۶۴ بیت خروجی میدهد. این خروجی با مقدار ثابتی که از قبل تعیین شده and میشود و اگر حاصل این and برابر ۰ شد، ورودی که ابتدا وارد حلقه شده بود را به عنوان مقدار قابل قبول در نظر خواهیم گرفت

پیاده سازی skein

قسمتی از باید در این بخش پیاده سازی شود، در کد با نام skein.calculateHash صدا زده میشود. روند

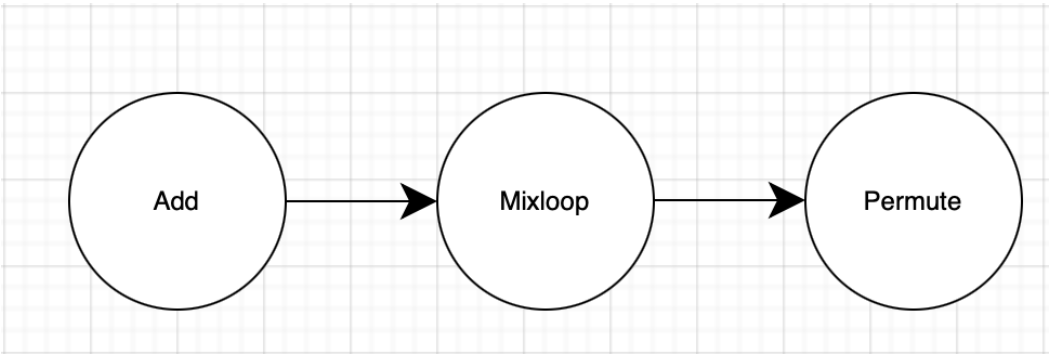


کلی الگوریتم در شکل زیر نشان داده شده است.

در ابتدا ۲ ورودی به عرض بیت های ۱۰۲۴ و ۲۱۵۰۴ وارد الگوریتم threefish میشود و خروجی ۱۰۲۴ بیتی تولید میکند که این خروجی با ۱۰۲۴ بیت اولیه xor شده و ورودی تابع makesubkey را تشکیل میدهند. این تابع خروجی ۲۱۵۰۴ بیتی تولید میکند که همراه با ۱۰۲۴ بیت ۰ ورودی threefish نهایی را تشکیل میدهند.

پیاده سازی threefish

این تابع از ۸۰ مرحله تشکیل شده که اسم هر مرحله را fishloop گذاشتیم و آن را ۸۰ بار تکرار میکنیم. هر تابع fishloop عملیات زیر را انجام میدهد



تابع Add

در این ماجول با توجه به شماره مرحله ای که در آن قرار داریم، تصمیم میگیریم عملیات جمع را انجام بدهم یا ندهیم. اگر شماره حلقه ای که در آن قرار داشتیم مضرب ۴ بود، جمع را انجام میدهیم و در غیر این صورت همان مقدار ورودی را به عنوان خروجی پاس میدهیم. ۲ ورودی این تابع ۱۰۲۴ بیت میباشد اما جمع آنها به صورت ۶۴ بیت ۶۴ بیت انجام میشود و بین این ۱۶ مرحله جمع مقدار carry برایمان بی اهمیت است. بنابراین ۱۶ جمع کننده ۶۴ بیتی به صورت موازی اجرا میشوند

تابع mixloop

در این ماجول ۲ ماجول mix1 و mix2 وجود دارد که از هر کدام ۸ عدد داریم. ماجول mix1 تنها یک جمع کننده است و ماجول mix2 از یک rotator و یک xor تشکیل شده. ۱۰۲۴ بیت ورودی mixloop به صورت ۱۶ واحد ۶۴ بیتی در نظر گرفته میشود که در ابتدا به صورت موازی، ۲ به ۲ توسط ۸ ماجول mix1 با هم جمع میشوند و سپس خروجی هر mix1 ، وارد ماجول های mix2میشوند و با توجه به موقعیت بیت ها به مقدار های ثابتی rotate شده xor میشوند.مقداری که در mix2 ، rotate داده میشود با استفاده از یک case statement پیاده سازی شده که بهینه تر باشد

تابع permute

این ماجول از نظر پردازشی عملیاتی روی ۱۰۲۴ داده ورودی خود انجام نمیدهد و تنها ترتیب بیت ها را عوض میکند

ماجول fishloop به صورت combinational پیاده سازی شده اما میتوانیم به سادگی با قرار دادن یک رجیستر بین تابع Add و mixloop اگر مشکلی از نظر فرکانس کاری وجود داشت، برطرف کنیم

ماجول threefish از ۸۰ ماجول fishloopتشکیل شده و خروجی هر کدام پس از رجیتسر شدن به عنوان ورودی وارد fishloop بعدی میشود بنابراین برای پایپ کردن این ماجول مشکلی نخواهیم داشت و به دلیل combinational بودن fishloop این کار به راحتی صورت میگیرد. اما مشکلی که در این ماجول هنگام پایپ کردن داریم، حجم بسیار بالای رجیستر های پایپ میباشد که به دلیل ورودی ۲۱۵۰۴ بیتی است که به ۸۰ رجیستر نیاز دارد تا هیچ داده ای گم نشود. همچنین اگر مشکل فرکانسی داشته باشیم و مجبور باشیم تابع fishloop را به صورت sequential پیاده سازی کنیم، این مشکل بزرگ تر نیز خواهد شد زیرا دیتا را باید در تعداد رجیستر بیشتری ذخیره کنیم(۱۶۰ رجیستر) .البته باید توجه کنیم که threefish اول نیازی به پایپ

۲۱۵۰۴ بیت ندار زیرا این دیتا همیشه برایش ثابت است و فقط دیتا ۱۰۲۴ بیتی آن عوض میشود اما threefish دوم نیاز دارد که همه ورودی هایش پایپ شوند

پیاده سازی makesubkey

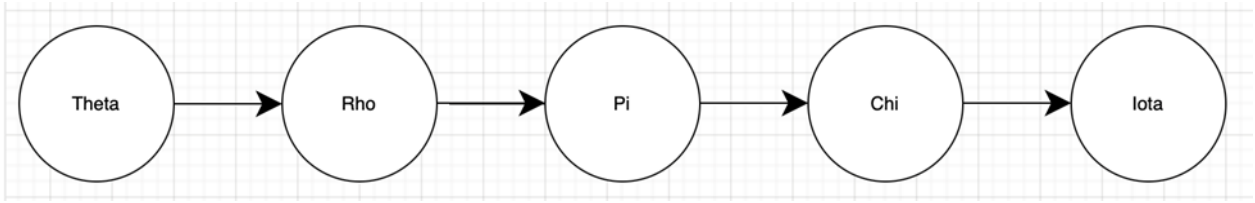
این ماجول ۱۰۲۴ بیت ورودی از مرحله قبل گرفته و ۲۱۵۰۴ بیت داده خروجی تولید میکند که در واقع به صورت ۲۱ داده ۱۰۲۴ بیتی در ۸۰ مرحله ماجول threefish استفاده میشود. در این ماجول یک ماجول به نام keystate دارد که ۱۰۲۴ بیت ورودی میگیرد و ۱۰۸۸ بیت خروجی تولید میکند. در keystate داده ۱۰۲۴ بیتی به ۱۶ قسمت ۶۴ بیتی تقسیم شده و با یکدیگر xor میشوند به این صورت که هر بار هر قسمت با xor شده قسمت قبل باید پردازش شود و در نهایت این مقادیر xor شده در انتهای ورودی concatenate میشود. به بیان دیگر خروجی keystate همان ورودی است که ۶۴ بیت داده جدید که حاصل xor شدن همان داده های ورودی است، به انتهای آن اضافه شده و از این مکانیزم به عنوان parity استفاده میکنیم. بنابراین نیاز داریم که خروجی هر قسمت را رجیستر کنیم. در نهایت خروجی keystate پس از ۱۵ کلاک آماده میشود و سپس ۲۱۵۰۴ بیت خروجی اصلی ماجول makesubkey با شیفت دادن و جمع کردن با مقادیر ثابتی از این ۱۰۸۸ بیت حاصل میشود . برای پایپ این ماجول مشکلی نخواهیم داشت و رجیستر های سنگین و زیادی نیاز نداریم زیرا ۲۱۵۰۴ بیت نهایی در یک کلاک و در انتها تولید میشود و نیازی به رجیستر شدن به صورت داخلی ندارد

در نهایت با قرار دان ماجول ها به ترتیب مطابق شکل ماجول skein آماده میشود و برای پایپ آن کافی است خروجی هر قسمت را در ابتدا رجیستر کنیم و سپس به عنوان ورودی وارد قسمت بعدی کنیم. همچنین باید دقت کنیم که ورودی ۱۰۲۴ بیتی این ماجول هم در threefish اول و هم در xor استفاده میشود. بنابراین هنگام پایپ کردن باید داده ۱۰۲۴ بیتی ورودی را در رجیستر ها ذخیره داشته باشیم تا خروجی threefish آماده شود و بعد این دو را با یکدیگر xorکنیم

پیاده سازیKeccak

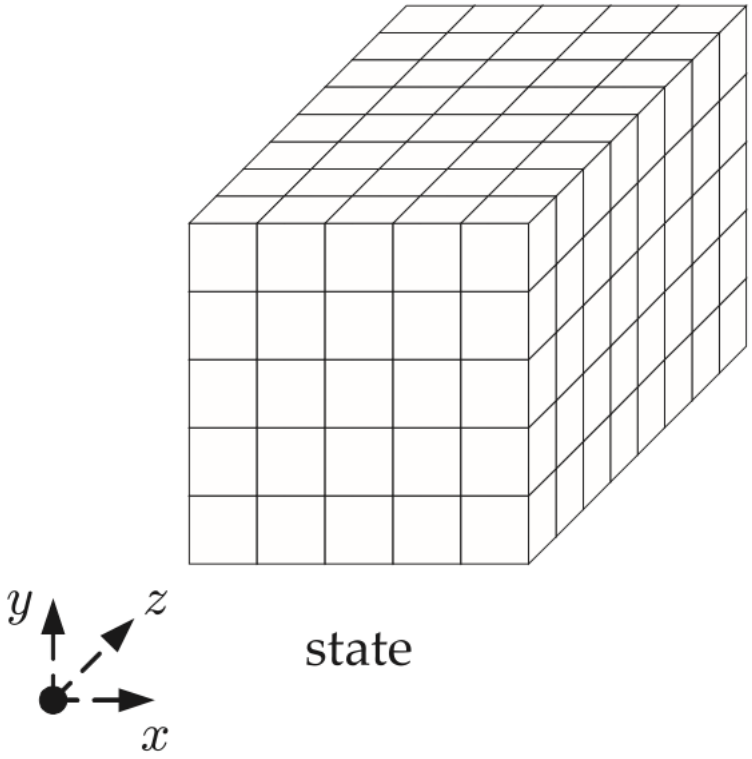
ماجول keccak ، ۱۰۲۴ بیت داده ورودی خود را از خروجی skein دریافت میکند و ۶۴ بیت خروجی به ما میدهد. الگوریتم keccak که در اینجا استفاده میکنیم در واقع از ۳ تابع keccak تشکیل شده که پشت سر هم قرار گرفته اند. حال به بررسی خود keccak میپردازیم.

هر keccak از ۲۴ مرحله تشکیل شده که همانند threefish خروجی هر قسمت رجیستر شده و به عنوان ورودی وارد مرحله بعد میشود .عملیاتی که در هر مرحله اجرا میشود به صورت زیر میباشد .باید توجه داشته باشیم که ورودی و خروجی هر کدام از توابع زیر ۱۶۰۰ بیت است



تابع Theta

برای فهم بهتر این تابع ورودی را به صورت یک مکعب ۵*۵*۶۴ در نظر میگیریم



عملیات هایی که در این تابع انجام میگیرد به صورت زیر است که A ورودی تابع میباشد و تابع $rot(C[],1)$ بیانگر rotation به اندازه ۱ بیت است. همچنین ایندکس ها همه باقی مانده به ۵ هستند یعنی $C[-1]$ به همان $C[4]$ اشاره میکند

$$B[x] = A[x,0] \oplus A[x,1] \oplus A[x,2] \oplus A[x,3] \oplus A[x,4], x = 0,1,2,3,4$$
$$C[x,y] = A[x,y] \oplus B[x-1] \oplus rot(B[x+1],1), x,y = 0,1,2,3,4$$

تابع Rho

در این تابع ورودی با مقادیر ثابتی rotate میشوند

$$D[k] = rot(c[x,y], r[x,y]), x, y = 0,1,2,3,4, k = 0,1,\dots,24$$

مقادیر C از خروجی قسمت قبل(Theta) به دست میاید و مقدار که باید rotate داده شود در یک جدول مشخص شده

	x=3	x=4	x=0	x=1	x=2
y=2	25	39	3	10	43
y=1	55	20	36	44	6
y=0	28	27	0	1	62
y=4	56	14	18	2	61
y=3	21	8	41	45	15

تابع pi

در این تابع داده های ورودی که از مرحله Rho میاید را در موقعیت های جدید قرار میدهیم. درواقع تنها ترتیب قرار گیری بیت ها عوض میشود

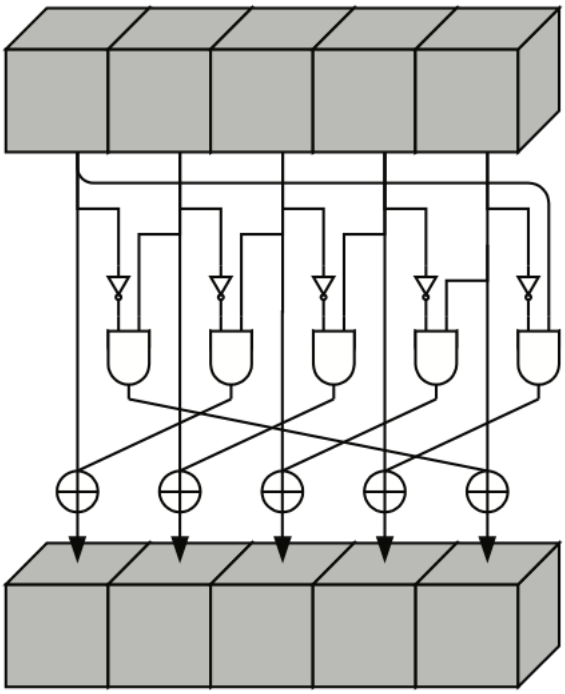
$$E[y, 2x + y] = D[k], x, y = 0,1,2,3,4, k = 0,1,\dots,24$$

تابع chi

عملیاتی که در این قسمت انجام میشود به صورت زیر است

$$f[x,y] = E[x,y] \oplus (\overline{E}[x+1,y] \wedge E[x+2,y]), x, y = 0,1,2,3,4$$

عبارت $\overline{E}$ بیانگر not شده ی تابع Eبه صورت بیت به بیت است و عبارت $\wedge$ بیانگر and به صورت بیت به بیت است. شکل زیر یک نمای کلی از تابع این قسمت را نشان میدهد



تابع Iota

در این مرحله تنها کاری که میکنیم xor کردن مقادیر ثابتی با خروجی قسمت قبل است .

در قسمت های قبل عملیاتی که در هر round تابع keccak اجرا میشود را توضیح دادیم. ماجولی که ما با آن سر و کار داریم در واقع از ۳ keccak تشکیل شده که به صورت کلی شامل ۷۲ round میشود. ورودی ۱۰۲۴ بیتی که از skein وارد این ماجول میشود به ۲ قسمت تبدیل میشود. قسمت اول شامل ۵۸۰ بیت پر ارزش و قسمت دوم ۴۴۸ بیت باقی مانده میباشد. ورودی اولین keccak از concatenate شدن ۵۸۰ بیت ورودی با ۱۰۲۰ بیت ۰ به دست میاید. ورودی دومین keccak همان خروجی keccak اول است که ۴۴۸ بیت پر ارزش آن با قسمت دوم ورودی xor میشود. و در نهایت ورودی سومین keccak همان خروجی keccak دوم است.

برای پایپ کردن این ماجول همانند threefish تنها کافی است خروجی هر round را رجیستر کرده و سپس وارد round بعدی کنیم. فقط باید توجه داشته باشیم که ورودی در keccak دوم نیز استفاده میشود . درواقع ورودی را باید تا جایی پایپ کنیم تا خروجی keccak اول آماده شود

ورودی دادن

ورودی های اصلی که وارد skein میشوند، یک subkey به اندازه ۲۱۵۰۴ بیت که ثابت است و یک message به اندازه ۱۰۲۴ بیت است که در هر کلاک با توجه به افزایش nonce ، تغییر میکند. nonce در واقع مقداری است که هر سری به message اضافه میشود و هنگامی که هش نهایی تولید شده شرایط لازم را داشت، مقدار nonce است که یرای ما مهم است و به عنوان خروجی برمیگردد.

مقدار message به طور مستقیم از تابع set_block میاید که به صورت نرم افزاری اجرا میشود اما داده ۲۱۵۰۴ بیتی را مستقیما از بیرون دریافت نمیکنیم و آن را با استفاده از تابع makesubkey خودمان تولید میکنیم. برای این کار نیاز به ۱۰۲۴ بیت داده برای ورودی makesubkey داریم. ۱۰۲۴ بیت داده در تابع set_block قبل از اینکه وارد makesubkey شود، وارد keystate شده و ۱۰۸۸ بیت داده تولید میکند که در واقع همان داده اصلی است که ۶۴ بیت parity به آن اضافه شده. داده ای که ما به عنوان ورودی از set_block دریافت میکنیم همین ۱۰۸۸ بیت است. پس از دریافت این داده نیاز داریم تا کاری برعکس کاری که keystate انجام میدهد انجام بدهم. درواقع با این کار از صحت ورودی اطمینان حاصل میکنیم به این صورت که ۱۰۲۴ بیت پر ارزش را ۶۴ بیت ۶۴ بیت با هم xor میکنیم(مانند کاری که در keystate انجام میدهیم) و با ۶۴ بیت کم ارزش مقایسه میکنیم. اگر این دو با هم برابر بودند یعنی ورودی ما معتبر بوده و از ۱۰۸۸ بیت ورودی ۱۰۲۴ بیت پر ارزش آن را وارد makesubkey میکنیم و اگر برابر نبود متوجه میشویم که ورودی اشتباه است و عملیاتی انجام نمیدهیم تا ورودی جدیدی بیاید و از صحت آن مطمئن شویم.

پیشنهادات و ارزیابی

با توجه به همه گیری بیماری کرونا و نبود امکان برگذاری کارآموزی به صورت حضوری، برگذاری جلسات هفتگی و نظارت منظم و دقیق باعث شد مشکلی در این مورد به وجود نیاید و بنده مورد برای انتقاد پیدا نکردم

مراجع

1] سایت رسمی Nexus و توضیحات تکمیلی الگوریتم

https://nexus.io

[2] پیاده سازی الگوریتم Nexus برای استفاده در CPU/GPU

https://github.com/Nexusoft/NexusMiner

[3] توضیحات الگوریتم keccak

http://professor.unisinos.br/linds/teoinfo/Keccak.pdf