

دانشکده مهندسی کامپیوتر
طراحی سیستم‌های دیجیتال
مستند پروژه

بررسی الگوریتم درهم‌سازی skein

نگارندگان:
حسن سندانی
محمد صالح سعیدی
مریم حاک
محمد مهدی عرفانیان
علی جندقی

۱۲ تیر ۱۳۹۸



فهرست مطالب

۲	۱	مقدمه
۳	۱.۱	توضیح الگوریتم
۳	۱.۱.۱	مثال‌هایی از درهم‌سازی
۳	۲.۱	مختصری درباره الگوریتم‌های درهم‌سازی امنیتی
۴	۳.۱	هدف الگوریتم درهم‌سازی skein
۴	۴.۱	نحوه کلی عملکرد الگوریتم
۵	۱.۴.۱	The Threefish block cipher
۵	۲.۴.۱	Unique Block Iteration
۷	۳.۴.۱	Skein تابع درهم‌سازی
۷	۴.۴.۱	Optional Arguments
۸	۵.۱	کاربردهای الگوریتم درهم‌سازی Skein
۱۱	۲	توصیف معماری سیستم
۱۲	۱.۲	اینترفیس‌های سیستم
۱۲	۱.۱.۲	ورودی‌ها
۱۲	۲.۱.۲	خروجی
۱۲	۲.۲	کلاک‌ها و نحوه راه‌اندازی سیستم
۱۲	۳.۲	دیاگرام بلوکی سخت‌افزار
۱۳	۴.۲	توصیف ماژول‌های سخت‌افزار
۱۳	۱.۴.۲	Skein-512
۱۵	۳	شبیه‌سازی
۱۶	۱.۳	توضیح روند شبیه‌سازی سخت‌افزار و گام‌های اجرایی
۱۶	۲.۳	مشاهده ورودی‌ها و خروجی‌های اصلی و میانی
۱۶	۱.۲.۳	توضیح نحوه عملکرد Testbench
۱۸	۲.۲.۳	مقادیر خروجی‌های حاصل از Testbench ها
۱۹	۳.۲.۳	جدول خروجی‌های هش برای Testbench ها

فصل ۱

مقدمه

توضیحی اولیه مشتمل بر تعریف الگوریتم، نحوه کلی عملکرد الگوریتم، پایه‌های ریاضی، کاربردها و استانداردها

۱.۱ توضیح الگوریتم

الگوریتمی که در ادامه این مستند شرح و توضیح آن آمده است الگوریتم درهم سازی skein یا cryptographic hash function است. این الگوریتم از سری الگوریتم‌های درهم‌سازی امنیتی یا cryptographic hash function و یکی از نامزدهای نهایی مسابقه انتخاب بهترین تابع درهم‌سازی NIST می‌باشد. این مسابقه برای انتخاب بهترین الگوریتم درهم‌سازی برای استاندارد جدید SHA-3 برگزار شد. طبق ادعای طراحان الگوریتم این الگوریتم می‌تواند در 6.1 کلاک در بایت داده‌ها را هش کند، که به این معنیست که در پردازنده دوهسته‌ای 64 بیتی با فرکانس پردازشی 3.1 GHz می‌تواند با سرعت 500 مگابایت بر ثانیه داده‌ها را هش کند. این مقدار سرعت تقریباً دو برابر سرعت هش کردن داده الگوریتم SHA-512 است. همچنین با گزینه درخت درهم‌سازی که می‌تواند به صورت اختیاری در الگوریتم پیاده‌سازی شود می‌توان در پیاده‌سازی موازی الگوریتم سرعت را به بیش از این هم رساند. نکته دیگری که در مورد الگوریتم skein لازم به ذکر است این است که این الگوریتم پیاده‌سازی آسان و ساده‌ای دارد و فقط از سه عملگر اصلی برای محاسبه هش استفاده می‌کند و نحوه عملکرد الگوریتم به راحتی قابل به خاطر سپاری و یادگیری است.

الگوریتم درهم‌سازی skein برای حالت‌های ورودی ۲۵۶، ۵۱۲ و ۱۰۲۴ بیتی و هر مقداری خروجی پیاده‌سازی شده است که این خاصیت در انعطاف الگوریتم در حالت‌های مختلف بسیار حیاتی است. در پیاده‌سازی سخت‌افزاری نیز این الگوریتم قوی عمل می‌کند، برای پیاده‌سازی skein-512 بر سخت‌افزار به حدود ۲۰۰ بایت فضای مموری نیاز داریم، برای skein-256 این مقدار به حدود ۱۰۰ بایت کاهش پیاده می‌کند که این الگوریتم را به یک الگوریتم مناسب برای پیاده‌سازی‌های روی قطعات کوچک سخت‌افزاری تبدیل می‌کند، مثلاً می‌توان از skein-256 در پیاده‌سازی smart card استفاده کرد. [۱]

۱.۱.۱ مثال‌هایی از درهم‌سازی

• Skein-256-256(“”)

c8877087da56e072870daa843f176e9453115929094c3a40c463a196c29bf7ba

• Skein-512-256(“”)

39ccc4554a8b31853b9de7a1fe638a24cce6b35a55f2431009e18780335d2621

• Skein-512-512(“”)

*bc5b4c50925519c290cc634277ae3d6257212395cba733bbad37a4af0fa06af4
1fca7903d06564fea7a2d3730dbdb80c1f85562dfcc070334ea4d1d9e72cba7a*

۲.۱ مختصری درباره الگوریتم‌های درهم‌سازی امنیتی

در دنیای امروز الگوریتم‌های درهم‌سازی امنیتی تقریباً در تمامی نقاط مختلفی که با اینترنت سر و کار دارند پیدا می‌شوند، بزرگ‌ترین کاربرد این الگوریتم‌ها ایجاد امضای دیجیتالی یا digital signature است که در ذخیره رمزهای عبور، اتصالات امنیتی به سرورها، مدیریت رمزنگاری‌ها و اسکن ویروس‌ها و بدافزارها به کار می‌رود، تقریباً تمامی پروتکل‌های امنیتی در دنیای اینترنت امروز بدون الگوریتم‌های درهم‌سازی امنیتی به سختی قابل پیاده‌سازی خواهند بود.

بزرگترین الگوریتم‌های درهم‌سازی امنیتی فعلی الگوریتم‌های خانواده SHA می‌باشند، الگوریتم‌های خانواده SHA به اختصار و فقط ذکر نام موارد زیر اند.

• SHA-0

• SHA-1

• SHA-256

• SHA-512

تمامی موارد بالا از روی الگوریتم‌های MD4 و MD5 اقتباس شده اند. در سال‌های اخیر کاستی‌ها و مشکلات امنیتی زیادی در الگوریتم‌های MD4, MD5, SHA-0, SHA-1 یافت شده‌اند اما هنوز باگ امنیتی بزرگی برای الگوریتم‌های SHA-256, SHA-512 یافت نشده است اما به دلیل وابستگی زیاد صنعت و امنیت فعلی اطلاعات به الگوریتم‌های درهم‌سازی در سال ۲۰۱۲ تصمیم بر این شد تا جایگزین مناسب و جدیدی برای الگوریتم‌های SHA-256, SHA-512 نیز انتخاب شود تا در صورتی که این الگوریتم‌ها شکسته شدند به سرعت الگوریتم‌های جدید در قالب نام SHA-3 جایگزین شوند.

۳.۱ هدف الگوریتم درهم‌سازی skein

هدف الگوریتم درهم‌سازی skein مانند دیگر الگوریتم‌های درهم‌سازی امنیتی ایجاد یک تابع برای درهم‌سازی داده‌های مختلف است به شکلی که ویژگی‌ها زیر برای آنان برقرار باشند.

- **قطعی بودن:** به شکلی که به ازای ورودی یکسان مقدار درهم‌سازی با تکرار الگوریتم برابر باشد، مثلاً با دادن ورودی "salam" به صورت متوالی به تابع مقدار هش تغییر نکند.
- **یک طرفه بودن:** نتوان از مقدار خروجی مقدار ورودی را یافت.
- **یک به یک بودن:** نتوان دو ورودی پیدا کرد به شکلی که به ازای این دو ورودی مقدار خروجی مساوی شود.
- **حساس بودن:** با تغییر اندک در ورودی خروجی به شکل قابل ملاحظه‌ای تغییر کند تا مقدار هش قابل حدس زدن نباشد.
- **سریع بودن:** الگوریتم باید بتواند هش را در مدت زمانی کوتاهی حساب کند تا به کاربردی بودن برسد.

۴.۱ نحوه کلی عملکرد الگوریتم

ایده اصلی الگوریتم بر ایجاد بلوک‌های رمزگذاری قابل تنظیم یا به زبان نویسندگان الگوریتم tweakable block cipher بنا نهاده شده است؛ به صورت دقیق‌تر می‌توان گفت که Skein از سه قسمت اصلی زیر تشکیل شده است و برای درهم‌سازی از ایشان استفاده می‌کند.

• Threefish

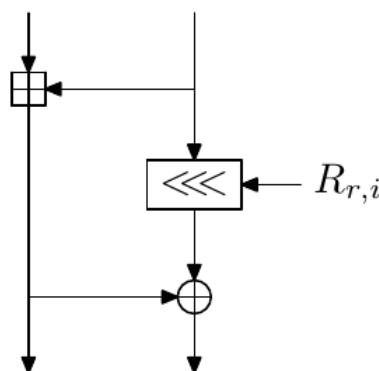
این قسمت یک بلوک رمزگذاری قابل تنظیم است که در هسته اصلی الگوریتم پیاده‌سازی شده است، این بلوک‌ها در سایزهای ۲۵۶، ۵۱۲، ۱۰۲۴ بیتی تعریف شده اند.

• Unique Block Iteration (UBI)

UBI یک حالت زنجیریست که با استفاده از بلوک قبلی به عنوان ورودی خود سعی در ایجاد یک الگوریتم فشرده‌سازی مخصوص ورودی می‌کند که بلوک ورودی با سایز دلخواه را به یک خروجی با سایز مشخص تبدیل کند.

• Optional Argument System

این ویژگی به الگوریتم اجازه می‌دهد تا از تعدادی ویژگی اختیاری بدون تحمیل هزینه بیش از حد اجرایی استفاده کند. [۲]



شکل ۱.۱: تابع MIX

همراهی سه بخش یادشده باهم ویژگی‌های جالب و کاربردی بسیاری را به الگوریتم درهم‌سازی Skein افزوده است، در ادامه به صورت خلاصه به نحوه عملکرد هر بخش می‌پردازیم.^۱

۱.۴.۱ The Threefish block cipher

Threefish یک بلوک رمزگذاری قابل تنظیم است که برای سه سایز بلوک مختلف تعریف شده است، ۲۵۶، ۵۱۲ و ۱۰۲۴ بیت. اصل اساسی در طراحی Threefish توجه به این مورد است که تعداد زیادی از مراحل ساده امن‌تر از تعداد کمی مراحل پیچیده است. Threefish فقط از سه عملگر اصلی XOR، جمع کردن و دوران به اندازه یک عدد ثابت^۲ استفاده می‌کند. شکل ۱.۱ نحوه عملکرد تابع غیرخطی استفاده شده در Threefish را نشان می‌دهد، این تابع در زبان طراحان الگوریتم MIX نامیده می‌شود و بر روی دو کلمه ۶۴ بیتی اجرا می‌شود. هر تابع MIX شامل یک جمع، یک دوران و یک XOR است.

۱.۱ نحوه عملکرد Threefish-512 را نشان می‌دهد، هر یک از مراحل هفتاد و دو گانه الگوریتم Skein-512 از چهار تابع MIX به همراه ضرب در یک کلمه ۶۴ بیتی انجام می‌شوند. ثابت‌های چرخش به شکلی انتخاب می‌شوند تا پخش‌شدگی را در هشت به حداکثر خود برسانند. برای به دست آوردن مقدار Threefish-512 ۷۲ بار الگوریتم شکل ۲.۱ تکرار می‌شود.^۳

۲.۴.۱ Unique Block Iteration

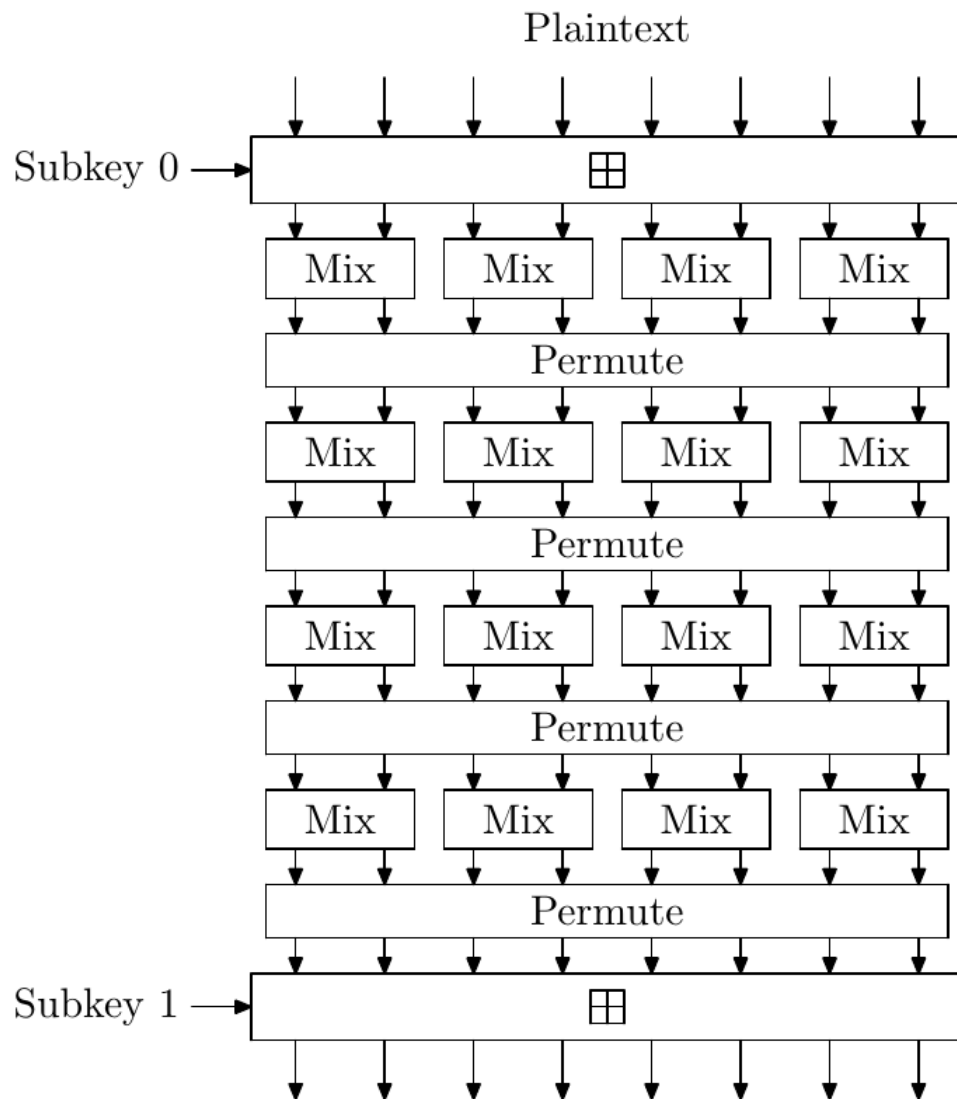
Unique Block Iteration یا به اختصار UBI زنجیره‌ای از ورودی‌ها را با یک رشته با طول دلخواه تلفیق می‌کند تا یک خروجی با اندازه مورد نظر و ثابت به دست آورد، در حقیقت UBI مقدار The Threefish block cipher را که مقداری با اندازه نامشخص و تعیین نشده‌ست را به خروجی با مقداری با اندازه ثابت تبدیل می‌کند، شکل ۳.۱ نحوه محاسبه UBI برای الگوریتم Skein-512 را نشان می‌دهد، اندازه ورودی ۱۶۶ بایت است که در سه بلوک ریخته شده است، بلوک‌های M_0 و M_1 هر کدام ۶۴ بایت دارند و M_2 که برچسب آخرین بلوک^۴ را دارد باقی‌مانده اندازه یعنی ۳۸ بایت دارد. با استفاده از tweak بلوک که قلب اصلی UBI را تشکیل می‌دهد UBI متوجه می‌شود که آیا تمامی بلوک‌ها برای ایجاد خروجی پردازش شده اند یا خیر و این که آیا به بلوک پایانی (پایان زنجیره) رسیده است یا خیر. UBI یکی از انواع Matyas-Meyer-Oseas

^۱ برای مطالعه بیشتر می‌توانید به بخش سوم [۲] مراجعه کنید.

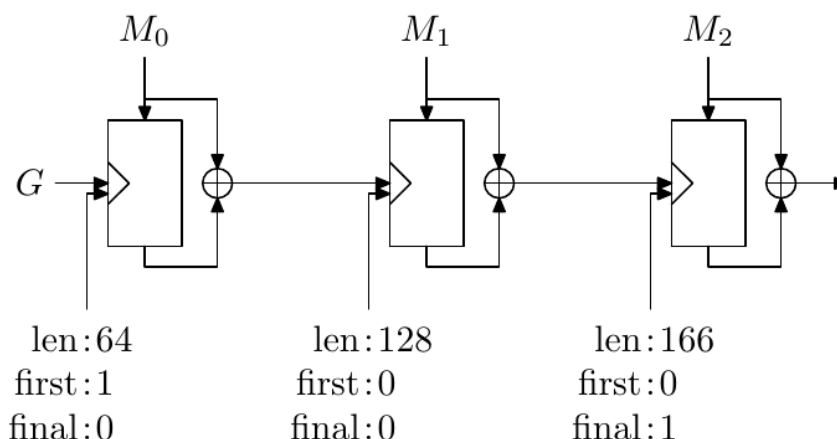
^۲ Rotation Constant

^۳ برای مطالعه جزئی‌تر می‌توانید به [۲] مراجعه کنید.

^۴ final block



شکل ۲.۱: چهار مرحله از ۷۲ مرحله Threefish-512 block cipher



شکل ۳.۱: درهم‌سازی پیام سه بلوکه با UBI

ها است. [۳]

۳.۴.۱ تابع درهم‌سازی Skein

تابع اصلی درهم‌سازی در حالت نرمال که مد نظر این نوشتار است برای ایجاد هش از چندین درخواست از UBI و بالتبع از Threefish block cipher هش یک داده ورودی را حساب می‌کند، برای محاسبه هش سه بار UBI با ورودی‌های مختلف زیر صدا زده می‌شود، شکل ۴.۱ توضیحات زیر را به صورت شماتیک نشان می‌دهد.

- **Config** این ورودی مقدار اندازه خروجی و تعدادی از پارامترها برای Tree-hashing را فراهم می‌کند، در صورتی که از حالت استاندارد و نرمال Skein برایش درهم‌سازی استفاده شود این مقدار قابل پیش‌پردازش است.

- **Message** مقدار داده ورودی است.

- **Counter** شمارنده‌ای برای نشان دادن تعداد بار تکرار الگوریتم ایجاد خروجی برای رسیدن به خروجی با اندازه مورد نظر است، در صورتی که خروجی بیش از اندازه‌ای مورد انتظار باشد، دوباره تابع ایجاد خروجی فراخوانی می‌شود.

۴.۴.۱ Optional Arguments

در راستای افزایش انعطاف‌پذیری الگوریتم درهم‌سازی skein برای کاربردهای مختلف تعدادی ورودی به صورت اختیاری به الگوریتم افزوده شده‌اند، در ادامه مختصراً به توضیح ایشان می‌پردازیم.

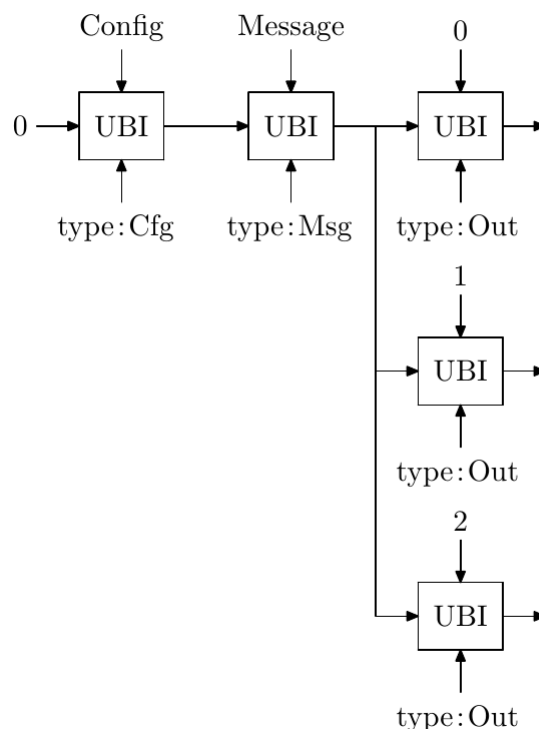
- **Key** (اختیاری) کلیدی برای تبدیل skein به تابع MAC یا KDF.

- **Configuration** (اجباری) همان مقدار Config که پیش‌تر توضیح داده شد.

- **Personalization** (اختیاری) رشته‌ای که برنامه می‌تواند با استفاده از آن تابع‌های مختلفی برای کاربردهای مختلفی بسازد.

- **Nonce** (اختیاری)

مقدار Nonce برای استفاده در حالت stream cipher و حالت درهم‌سازی تصادفی.



شکل ۴.۱: تابع ایجاد هش با خروجی بزرگ‌تر از اندازه مورد انتظار

- **Message** (اختیاری)
ورودی نرمال تابع درهم‌سازی.
- **Output** (اجباری) مقدار خروجی الگوریتم.
در محاسبه هش تابع درهم‌سازی Skein به ترتیب ذکر شده در بالا UBI ورودی‌ها محاسبه می‌شود.

۵.۱ کاربردهای الگوریتم درهم‌سازی Skein

- **Skein به عنوان تابع درهم‌سازی** ساده‌ترین راه استفاده از الگوریتم Skein استفاده به عنوان تابعی برای به دست آوردن هش ورودی‌ست، در این حالت Skein مانند تمام الگوریتم‌های دیگر درهم‌سازی عمل می‌کند و رشته‌ای را به عنوان هش با اندازه از پیش تعیین شده خروجی می‌دهد.
- **Skein به عنوان MAC** از تابع درهم‌سازی Skein می‌توان برای تولید MAC^۵ استفاده کرد، از MAC برای واریسی این که یک پیام از یک فرستنده معتبر بدون تغییر ارسال شده یا که در طی مسیر دست‌کاری شده است استفاده می‌شود.

• HMAC

• Randomized Hashing

• Digital Signatures

• Key Derivation Function (KDF)

^۵Message authentication code

• Password-Based Key Derivation Function (PBKDF)

• PRNG

• Stream Cipher

کتاب نامه

<http://www.skein-hash.info/about> [۱]

The Skein Hash Function Family [۲]
Version 1.3 — 1 Oct 2010
<http://www.skein-hash.info/sites/default/files/skein1.3.pdf>

S.M. Matyas, C.H. Meyer, and J. Oseas, “Generating strong one-way functions with [۳]
crypto- graphic algorithms
” IBM Technical Disclosure Bulletin, Vol. 27, No. 10A, 1985, pp. 5658–5659.

فصل ۲

توصیف معماری سیستم

تشریح اینترفیس‌های سیستم، کلاک‌ها و نحوه راه‌اندازی سیستم، دیاگرام بلوکی سخت‌افزار، ساختار درختی سیستم و توصیف ماژول‌های سخت‌افزار

۱.۲ اینترفیس‌های سیستم

در ابتدا به صورت خلاصه اینترفیس‌های سیستم سخت‌افزاری الگوریتم Skein بیان می‌شود، اینترفیس یک سیستم شامل ورودی‌ها و خروجی‌ها و مشخصات ایشان است.

۱.۱.۲ ورودی‌ها

ورودی‌ها کد verilog الگوریتم Skein به شرح زیر اند.

• clk

ورودی کلاک سیستم است که با آن سیستم کار خود را به صورت ترتیبی^۱ انجام می‌دهد، فرکانس کلاک با توجه به نحوه پیاده‌سازی سخت‌افزاری و نتایج حاصل از سنتر تعیین می‌شود.

• midstate

ورودی‌ای ۵۱۲ بیتی برای الگوریتم Skein-512 است که حالت میانی در هش را معلوم می‌کند.

• nonce

nonce مقداری دلخواه است که برای به حداکثر رساندن تصادفی و غیرقابل شکستن بودن هش در محاسبه هش استفاده می‌شود، این مقدار می‌تواند عددی دلخواه باشد. در الگوریتم Skein-512 اندازه این ورودی ۳۲ بیت به اندازه طول عدد در Integer گرفته شده است.

• data

ورودی اصلی‌ست که باید هش آن محاسبه شود، در کد verilog داده شده اندازه این ورودی ۹۶ بیت در نظر گرفته شده است.

۲.۱.۲ خروجی

تنها خروجی سیستم مقدار هش در output است که ۵۱۲ بیت طول دارد. (الگوریتم مورد بحث Skein-512 است)

۲.۲ کلاک‌ها و نحوه راه‌اندازی سیستم

این سیستم فقط از یک کلاک استفاده می‌کند و برای راه‌اندازی سیستم انجام کارهای زیر ضروری‌ست.

۱. وصل کردن کلاک با فرکانس مناسب به سیستم

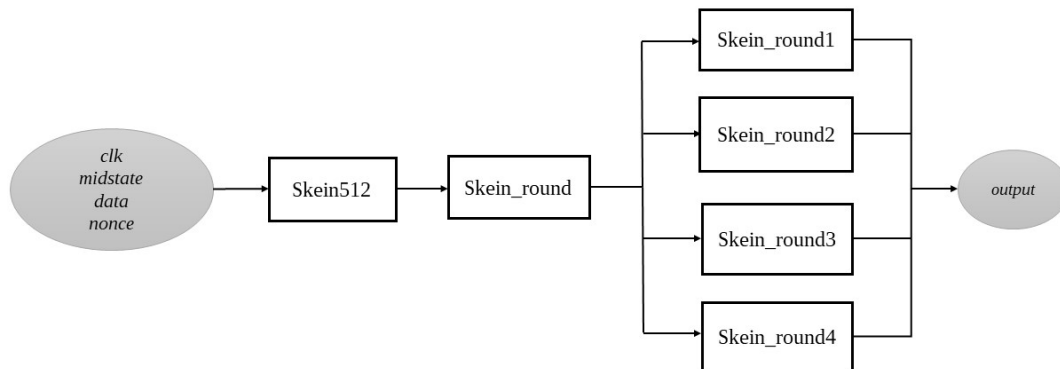
۲. تعیین ورودی‌های اولیه

۳. راه‌اندازی سیستم

۳.۲ دیاگرام بلوکی سخت‌افزار

دیاگرام بلوکی کلی سخت‌افزار در شکل ۱.۲ آمده است.

^۱ Sequential



شکل ۱.۲: دیاگرام بلوکی سخت‌افزار

۴.۲ توصیف ماژول‌های سخت‌افزار

۱.۴.۲ Skein-512

ابتدا تعدادی reg و wire گرفته شده است. دو reg به نام های phase_d و phase_q تعریف شده اند که یک‌بیتی اند و مقدار صفر به آنها داده شده است. عبارت assign داریم:

۱. در reg ۳۲ بیتی با نام nonce_le که در خطوط بالاتر تعریف شده است مقادیر nonce (که ورودی ۳۲ بیتی ماژول هستند) به صورت ۸ بیت - ۸ بیت و به صورت برعکس ذخیره می‌شوند. یعنی به طور مثال ۸ بیت کم ارزش در nonce ۸ بیت پر ارزش nonce_le ذخیره شده اند. (خط ۵۵)

۲. در reg ۳۲ بیتی با نام nonce2_le که در خطوط بالاتر تعریف شده است مقادیر nonce2 (که برعکس nonce، ورودی ماژول نیست و خود در خطوط بالاتر به صورت یک reg ۳۲ بیتی تعریف شده است و در واقع در حال حاضر مقداری را به خود اختصاص نداده است) به صورت ۸ بیت - ۸ بیت و به صورت برعکس ذخیره می‌شوند. یعنی به طور مثال ۸ بیت کم ارزش در nonce2 ۸ بیت پر ارزش nonce2_le ذخیره شده اند. (خط ۵۶)

یک عبارت assign بزرگ مربوط به hash دیده می‌شود:

- در این عبارت بیت‌های reg ی به نام h_q که ۵۱۲ بیت دارد و در خطوط بالاتر تعریف شده است، به بیت های خروجی hash اساین می‌شود.
- ۶۴ مجموعه ۸ بیتی از h_q به بیت های hash اساین می‌شود که نظم این مقداردهی در زیر توضیح داده می‌شود. در این توضیحات hash را به ترتیب از پر ارزش ترین بیت شروع به پر کردن می‌کنیم.
- پر ارزش ترین بیت‌های hash با بیت های ۴۶۳ تا ۴۵۶ پر شده است. (یعنی پر ارزش ترین بیت hash با بیت ۴۶۳ h_q پر شده است و به همین ترتیب)
- مجموعه بعدی ۸ تایی از ۴۶۴ تا ۴۷۱ هستند که در دومین ۸ تایی با ارزش hash قرار می‌گیرند.

- این روند تا هشتمین ۸ بیت ارزشمند hash ادامه پیدا میکند جایی که در این جایگاه مجموعه [۵۱۱:۵۰۴] از h_q جای میگیرد. (تا اینجا نظم داشتیم)
 - نهمین ۸ بیت ارزشمند hash توسط بیت های [۳۹۱:۳۸۴] از h_q پر میشوند.
 - این روند ادامه پیدا میکند (یعنی دهمین ۸ بیت ارزشمند با [۳۹۹:۳۹۲] پر میشوند). تا ۱۶امین ۸ بیت ارزشمند hash که با مجموعه [۴۴۷:۴۴۰] پر شده اند.
 - ۱۷امین ۸ بیت ارزشمند با مجموعه [۳۲۷:۳۲۰] پر میشود.
 - این روند مانند قبل به صورت صعودی ادامه پیدا خواهد کرد تا به ۲۵امین مجموعه ۸ بیتی برسیم.
- درواقع هر ۸ بار که مجموعه بیت های ۸ بیتی را assign میکنیم، یک بی‌نظمی داریم.**

- ۲۵امین ۸ بیتی hash با بیت های [۲۶۳:۲۵۶] پر میشود.
- دوباره روند سابق و صعودی را داریم تا به ۳۳امین assignment برسیم.
- ۳۳امین ۸ بیتی hash با بیت های [۱۹۹:۱۹۲]

هر بار بی‌نظمی داریم بازه جدید بعد از بی‌نظمی ۱۲۰ واحد کمتر از بازه قبلی خواهد بود مثلاً ۳۲امین ۸ بیت پر ارزش hash با بیت های [۳۱۹:۳۱۲] پر شده اند که ۱۲۰ واحد از بازه ای که برای ۳۳امین ۸ بیت ارزشمند hash اختصاص داده میشود بیشتر است. (در بالا ۳۳امین نوشته شده است)

- باز ۸ مجموعه که به صورت صعودی پیش برویم به ۴۰امین ۸ بیت میرسیم که طبق نظم با بیت های [۲۵۵:۲۴۸] پر شده است و ۴۱امین ۸ بیتی با بازه [۱۳۵:۱۲۸] پر شده است.
- باز ۸ مجموعه که به صورت صعودی پیش برویم به ۴۸امین ۸ بیت میرسیم که طبق نظم با بیت های [۱۹۱:۱۸۴] پر شده است و ۴۹امین ۸ بیتی با بازه [۷۱:۶۴] پر شده است.
- باز ۸ مجموعه که به صورت صعودی پیش برویم به ۵۶امین ۸ بیت میرسیم که طبق نظم با بیت های [۱۲۷:۱۲۰] پر شده است و ۵۷امین ۸ بیتی با بازه [۷:۰] پر شده است.
- از ۵۷امین مجموعه ۸ تایی با ارزش hash تا آخرین مجموعه با ارزش hash (۶۴امین) نیز به صورت صعودی و طبق نظم پیش میرود. (خط ۱۲۱)

بعد از خطوط، assignment ۱۸ instance از ماژول skein_round گرفته شده است. این - in stance ها را از ۰ تا ۰H نام گذاری کردیم (نامگذاری در مبنای بالاتر از ۱۰ شده است)

فصل ۳

شبیه‌سازی

توصیف روند شبیه‌سازی سخت‌افزار و گام‌های اجرایی، مشاهده ورودی‌ها و خروجی‌های اصلی و میانی، مقایسه با مقادیر حاصل از اجرای کد نرم‌افزاری (مدل طلایی)، توصیف مراحل اجرای الگوریتم به همراه شکل موج‌ها، نحوه عملکرد Testbench

۱.۳ توضیح روند شبیه‌سازی سخت‌افزار و گام‌های اجرایی

برای شبیه‌سازی سخت‌افزاری کد verilog الگوریتم Skein را در محیط شبیه‌سازی Modelsim اجرا کردیم. گام‌های اجرایی به صورت کلی برای شبیه‌سازی کد سخت‌افزاری موارد زیر بود.

- مطالعه کد الگوریتم و تعیین ورودی‌ها
- نوشتن Testbench
- اجرای کد در محیط Modelsim با Testbench های مختلف
- گرفتن Waveform و مقادیر خروجی (اصلی و میانی)

۲.۳ مشاهده ورودی‌ها و خروجی‌های اصلی و میانی

در ادامه ابتدا کد های Testbench اجرا شده بر الگوریتم و سپس Waveform های حاصله و در انتها خروجی‌ها به صورت متنی آورده می‌شود.

۱.۲.۳ توضیح نحوه عملکرد Testbench

در ادامه ابتدا کد verilog نوشته شده برای هر Testbench آورده و سپس توضیحاتی درباره آن ایراد شده است.

Testbench 1

```

1 //First Testbench
2 module top;
3     reg clk = 1'b0;
4     reg [511:0] midstate = 72;
5     reg [95:0] data = "hello" ;
6     reg [31:0] nonce = 13;
7     wire [511:0] hash;
8     always #1 clk = !clk;
9
10    skein512 skein(clk, midstate , data ,nonce , hash);
11
12    initial
13    begin
14        #300 data = "how are you?";
15        #300 data = "bye";
16        #5000 $stop;
17    end
18 endmodule

```

در این testbench ابتدا مقادیر ورودی‌ها تنظیم می‌شوند به ترتیب به ازای clock و midstate و data و nonce مقادیر ۰ و ۷۲ و "hello" و ۱۳ تنظیم می‌شوند پس از ۳۰۰ واحد زمانی مقدار data تغییر می‌کند و به "how are you?" تبدیل می‌شود و پس از ۳۰۰ واحد زمانی دیگر به "bye" تغییر می‌کند. ترتیب و مقدار hash در بخش مربوط به آن آمده است.

Testbench 2

```

1 //Second Testbench
2 module top;
3     reg clk = 1'b0;
4     reg [511:0] midstate = 72;
5     reg [95:0] data = "hello" ;
6     reg [31:0] nonce = 23;
7     wire [511:0] hash;
8     always #1 clk = !clk;
9
10    skein512 skein(clk, midstate , data ,nonce , hash);
11
12    initial
13    begin
14        #300 data = "how are you?";
15        #300 data = "bye";
16        #5000 $stop;
17    end
18 endmodule

```

در این testbench نیز ابتدا مقادیر ورودی‌ها تنظیم می‌شوند، به ترتیب به ازای clock و midstate و data و nonce مقادیر ۰ و ۷۲ و "hello" و ۲۳ تنظیم می‌شوند پس از ۳۰۰ واحد زمانی مقدار data تغییر می‌کند و به "how are you?" تبدیل می‌شود و پس از ۳۰۰ واحد زمانی دیگر به "bye" تغییر می‌کند. ترتیب و مقدار hash در بخش مربوط به آن آمده است تنها تفاوت این بخش و بخش قبلی در مقادیر ورودی nonce است که از ۱۳ به ۲۳ تغییر داده شده است.

Testbench 3

```

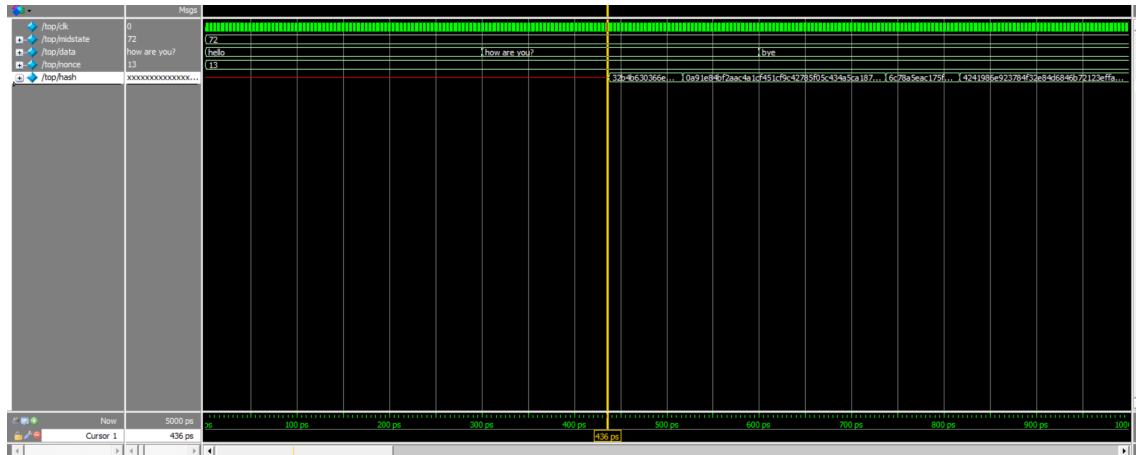
1 //Third Testbench
2 module top;
3     reg clk = 1'b0;
4     reg [511:0] midstate = 72;
5     reg [95:0] data = "still awake" ;
6     reg [31:0] nonce = 23;
7     wire [511:0] hash;
8     always #1 clk = !clk;
9
10    skein512 skein(clk, midstate , data ,nonce , hash);
11
12    initial
13    begin
14        #300 data = "working";
15        #6000 $stop;
16    end
17 endmodule

```

در این testbench ابتدا مقادیر ورودی‌ها تنظیم می‌شوند، به ترتیب به ازای clock و midstate و data و nonce مقادیر ۰ و ۷۲ و "still" و "awake" و ۲۳ تنظیم می‌شوند پس از ۳۰۰ واحد زمانی مقدار data تغییر می‌کند و به "working" تبدیل می‌شود. ترتیب و مقدار hash در بخش مربوط به آن آمده است. در این بخش با تغییر مقادیر اولیه و ثانویه data خروجی‌ها را با قسمت قبل مقایسه کردیم.

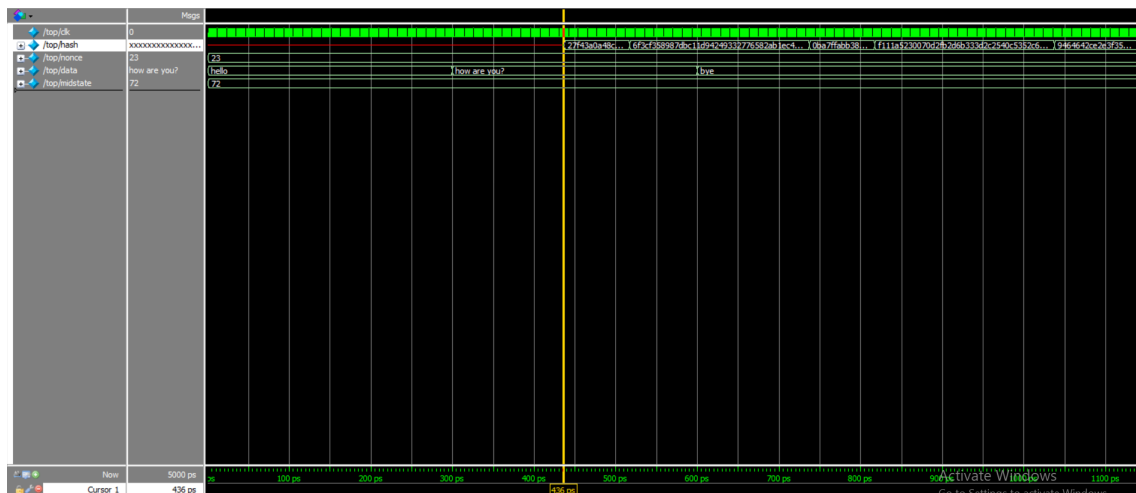
۲.۲.۳ مقادیر خروجی‌های حاصل از Testbench ها

Waveform 1



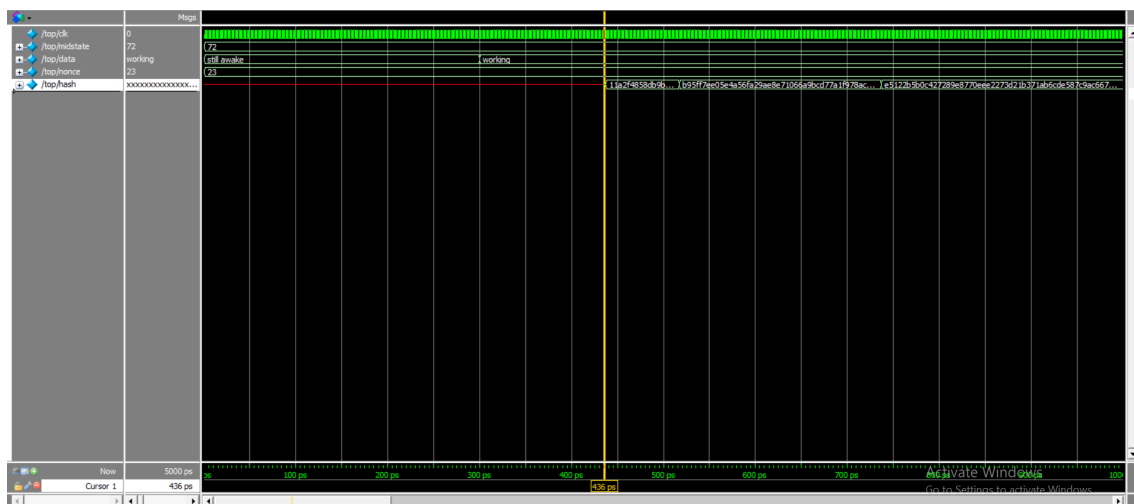
شکل ۱.۳: شبیه‌سازی با Testbench 1

Waveform 2



شکل ۲.۳: شبیه‌سازی با Testbench 2

Waveform 3



شکل ۳.۳: شبیه‌سازی با Testbench 3

۳.۲.۳ جدول خروجی‌های هش برای Testbench ها

Output 1

Data	Time(clk)	Hash	Final
hello	436	32b4b630366ea0d126796328ad99cf0dbb95b4b3b08bdac2c9965ab01993886499e4a150a5b7f93ac75eaad9c70cc3b2e7147887c5953849c9ddc96b2cbfa4170a91e84bf2aac4a1cf451cf9c42785f05c434a5ca1876c64382210c0c2102aa52fa8064a67e7510a670d0558a54efbc789d6c9b0198258d85e797e82bfc0f26e6c78a5eac175f4456c22f40253f9c2b877a1f8dc177ba179f9778f468c7ce8b379a8d2f259227c9e2320eff4040ff3d81461a0ae93cb5d6fd4fa0b6c37334df94241986e923784f32e84d6846b72123effa16f143d874cbeefc815dd972e15d9b3591d2d409de47c364dd6d7676ce4050220b28a762fa1dc39099280459c8e74aed5a975b556508072d512af282bbe56877874bd463a786544e3406875931cf8e5f8a7e823193eed2904bc00ac5d3e7f81d8162667b7ac171f1fbd92dcda7e5	False
hello			True
you? are how			False
you? are how			True
bye			True

جدول ۱.۳: مقادیر درهم‌سازی و زمان با توجه به Testbench 1

Output 2

Data	Time(clk)	Hash	Final
hello	436	27f43a0a48c65a7b90d2915a252b4ad97063a43818089082a95c2dbe5148ed70c07604d0cdf5310df8b18c3f368b83516f7f4195b435d53900d5fe31c8f6ee206f3cf358987dbc11d94249332776582ab1ec4e81ce512fa28e5f8d77dd9c6e04224c88f2b28bda6c0becf728f5e45c60bf20f22ca4f08a030f38db131694ce300ba7ffabb38454cdf8f29d2b33ef4a9a3ab103e20d9eeb8b55ef36707ced6795280c9695df1d5d1072939177e48994faaf226161fb3998b90254f5bc0e51ba2bf111a5230070d2fb2d6b333d2c2540c5352c61585f4e5ba3b485c7a33400a0417ff4898cba78441791013f571a1aafdf287d597b5162937d194ff67f6620f2709464642ce2e3f35da6af64bb4b5ffda92151870f81c98934946ff4921364cf80cd134e927accac3ebe7d35d0e1ac9bf4933063a0a182a53d84f0c65e588e1b24	False
hello			True
you? are how			False
you? are how			True
bye			True

جدول ۲.۳: مقادیر درهم‌سازی و زمان با توجه به Testbench 2