دانشکده مهندسی کامپیوتر

طراحی سیستم‌های دیجیتال

مستند پروژه

# بررسی الگوریتم درهم‌سازی skein

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

۱۴ تیر ۱۳۹۸

# فهرست مطالب

# 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 ادامهٔ این مستند شرح و توضیح آن آمده است الگوریتم درهم سازی skein یا skein 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 پردازندهٔ دوهسته‌ای 64 بیتی با فرکانس پردازشی 3.1 GHz می‌تواند با سرعت 500 مگابایت بر ثانیه داده‌ها را هش کند. این مقدار سرعت تقریبا دوبرابر سرعت هش کردن دادهٔ 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 نحوهٔ عملکرد الگوریتم به راحتی قابل به خاطرسپاری و یادگیری‌ست.

الگوریتم درهم‌سازی skein برای حالت‌های ورودی ۲۵۶، ۵۱۲ و ۱۰۲۴ بایتی و هرمقداری خروجی پیاده‌سازی شده است که این خاصیت در انعطاف الگوریتم در حالت‌های مختلف بسیار حیاتی‌ست.
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، مثلا 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 دربارهٔ الگوریتم‌های درهم‌سازی امنیتی

در دنیای امروز الگوریتم‌های درهم‌سازی امنیتی تقریبا 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 ذخیرهٔ رمزهای عبور، اتصالات امنیتی به سرورها، مدیریت رمزنگاری‌ها و اسکن ویروس‌ها و بدافزارها به کار می‌رود، تقریبا 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

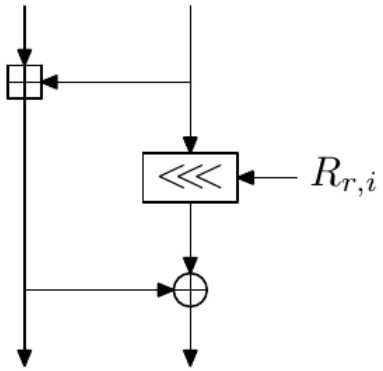
هدف الگوریتم درهم‌سازی skein مانند دیگر الگوریتم‌های درهم‌سازی امنیتی ایجاد یک تابع برای درهم‌سازی داده‌های مختلف است به شکلی که ویژگی‌ها زیر برای آنان برقرار باشند.

- **قطعی بودن:** به شکلی که به ازای ورودی یکسان مقدار درهم‌سازی با تکرار الگوریتم برابر باشد، مثلا 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 نحوهٔ کلی عملکرد الگوریتم

ایدهٔ اصلی الگوریتم بر ایجاد بلوک‌های زمزگذاری 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 نحوهٔ عملکرد هر بخش می‌پردازیم. [۱]

### ۱.۴.۱ The Threefish block cipher

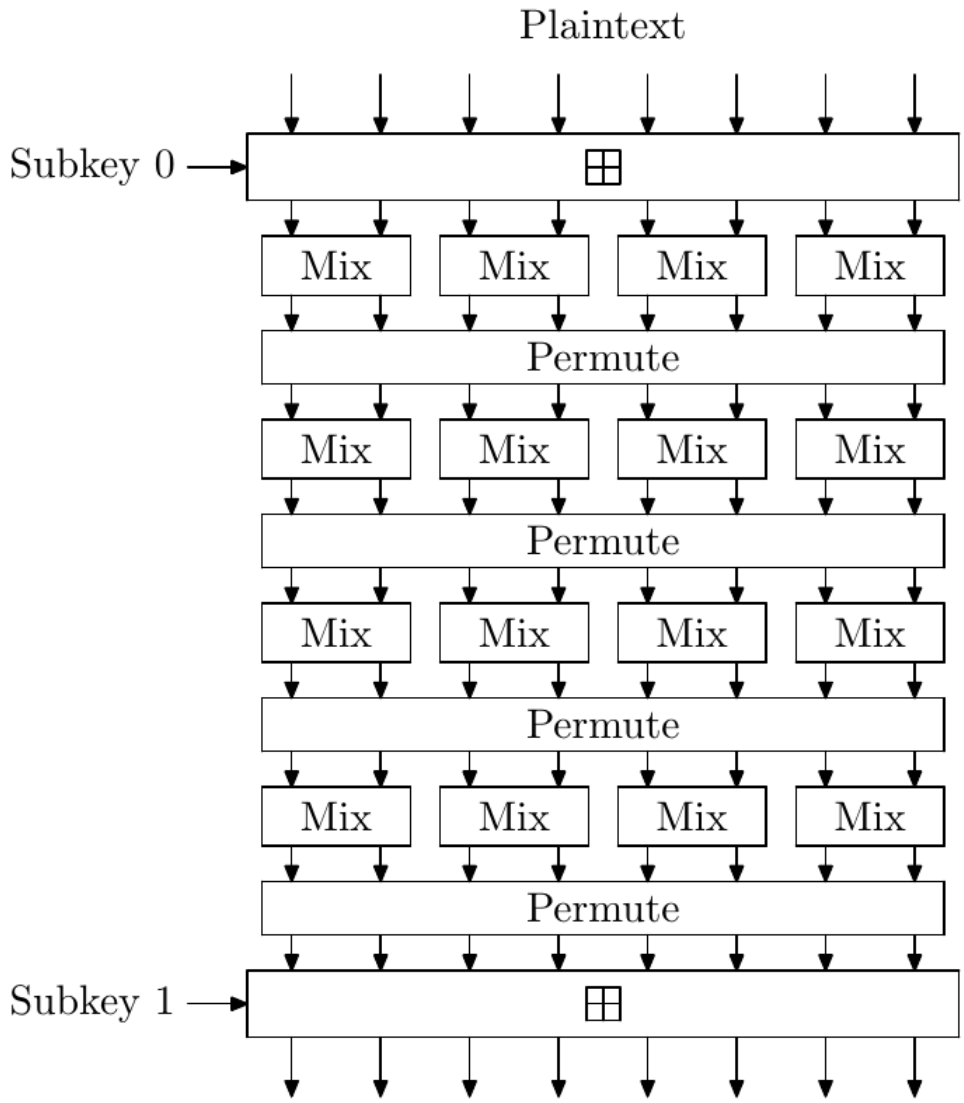
Threefish یک بلوک رمزگذاری قابل تنظیم است که برای سه سایز بلوک مختلف تعریف شده است، ۲۵۶، ۵۱۲ و ۱۰۲۴ بیت. اصل اساسی در طراحی Threefish توجه به این مورد است که تعداد زیادی از مراحل ساده امن‌تر از تعداد کمی مراحل پیچیده است. Threefish فقط از سه عمل‌گر اصلی XOR ، جمع کردن و دوران به اندازه یک عدد ثابت [۲] 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 نحوهٔ عملکرد Threefish-512 را نشان می‌دهد، هر یک از مراحل هفتاد و دوگانهٔ 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 هش به حداکثر خود برسانند. برای به دست آوردن مقدار Threefish-512 ۷۲ بار الگوریتم شکل ۲.۱ تکرار می‌شود. [۳]

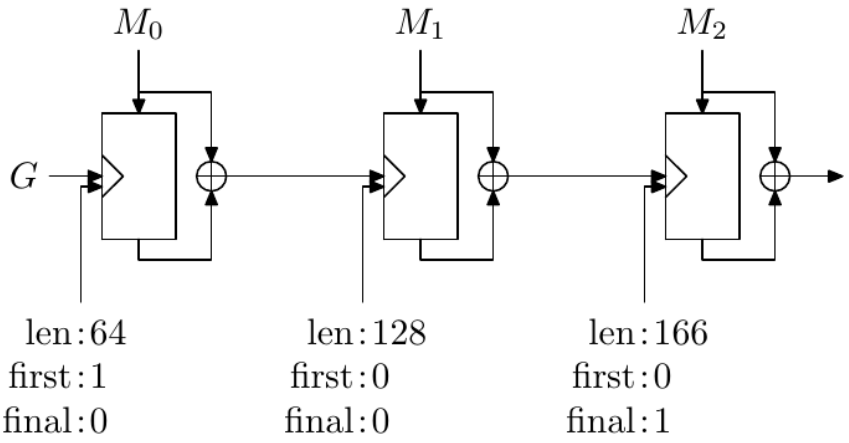
### ۲.۴.۱ Unique Block Iteration

Unique Block Iteration یا به اختصار UBI زنجیره‌ای از ورودی‌ها را با یک رشته با طول دلخواه تلفیق می‌کند تا یک خروجی با اندازهٔ مورد نظر و ثابت به دست آورد، در حقیقت UBI مقدار The Threefish block cipher را که مقداری با اندازهٔ نامشخص و تعیین‌نشده‌ست را به خروجی با مقداری با اندازهٔ ثابت تبدیل می‌کند، شکل ۳.۱ نحوهٔ محاسبه UBI برای الگوریتم Skein-512 را نشان می‌دهد، اندازهٔ 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 جزیی‌تر می‌توانید به [۲] مراجعه کنید.
[۴] final block




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




شکل ۳.۱: درهم‌سازی پیام سه بلوکه با UBI

ها است. [۳]

### ۳.۴.۱ تابع درهم‌سازی Skein

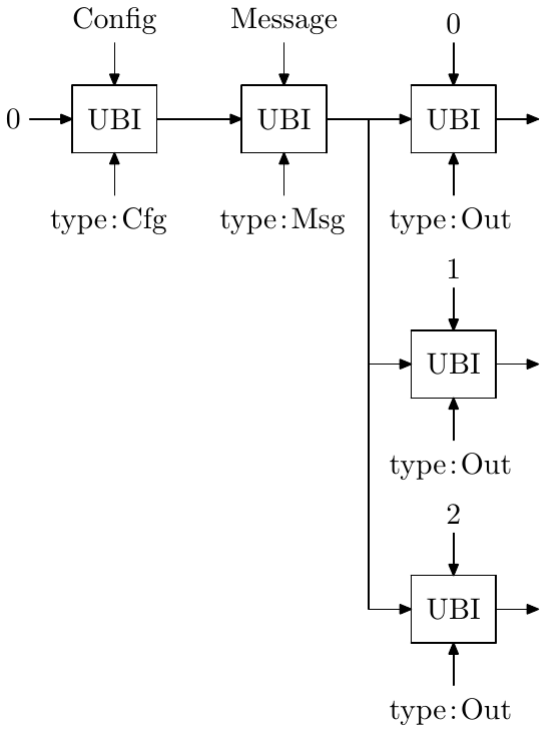
تابع اصلی درهم‌سازی در حالت نرمال که مد نظر این نوشتار است برای ایجاد هش از چندین درخواست از UBI و باالتبع از Threefish block cipher هش یک داده ورودی را حساب می‌کند، برای محاسبه هش سه بار UBI با ورودی‌های مختلف زیر صدا زده می‌شود، شکل ۴.۱ توضیحات زیر را به صورت شماتیک نشان می‌دهد.

- **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 براش درهم‌سازی استفاده شود این مقدار قابل پیش‌پردازش است.
- **Message** مقدار داده ورودی‌ست.
- **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 شده اند، در ادامه مختصرا 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 ازپیش‌تعیین‌شده خروجی می‌دهد.
- **Skein به عنوان MAC** از تابع درهم‌سازی Skein می‌توان برای تولید MAC [۵] استفاده کرد، از MAC برای وارسی 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)**

[۵] Message authentication code

- **Password-Based Key Derivation Function (PBKDF)**
- **PRNG**
- **Stream Cipher**

# کتاب‌نامه


[۱] http://www.skein-hash.info/about

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

[۳] S.M. Matyas, C.H. Meyer, and J. Oseas, “Generating strong one-way functions with 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 نحوهٔ 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 نحوهٔ پیاده‌سازی سخت‌افزاری و نتایج حاصل از سنتز 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 اندازهٔ 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 نحوهٔ 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. اعمال ریست کلی بر سیستم [۲]

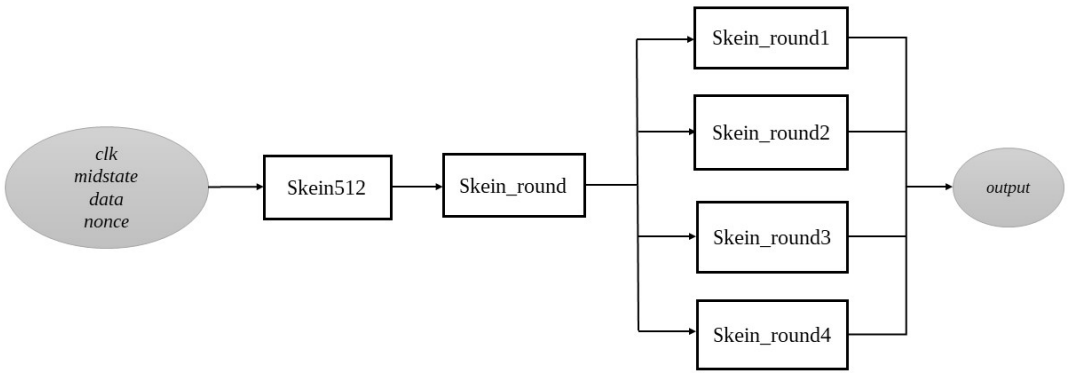
۳. تعیین ورودی‌های اولیه

۴. راه‌اندازی سیستم

## ۳.۲ دیاگرام بلوکی سخت‌افزار

دیاگرام بلوکی کلی سخت‌افزار در شکل ۱.۲ آمده است.

[۱] Sequential
[۲] Global Reset




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.
دو عبارت assign در کد وجود دارد.

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 درnonce ۸ بیت پرارزش nonce_le ذخیره شده اند. (خط lk(۵۵

۲. در reg ۳۲ بیتی با نام nonce۲_le که در خطوط بالاتر تعریف شده است مقادیر nonce۲ (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 درnonce۲ ۸ بیت پرارزش nonce۲_le ذخیره شده اند. (خط ۵۶)

یک عبارت assign طویل 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 پرارزش‌ترین بیت شروع به پر کردن میکنیم.
- پرارزش‌ترین بیت‌های hash با بیت های ۴۶۳ تا ۴۵۶ پر شده است. (یعنی پرارزش ترین بیت hash با بیت ۴۶۳ ام h_q پر شده است و به همین ترتیب)
- مجموعه بعدی ۸ تایی از۴۶۴ 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 ۲۵ امین مجموعه ۸ بیتی برسیم.

***درواقع هر ۸ بار که مجموعه بیت های ۸ بیتی را assign میکنیم، یک بی‌نظمی داریم.***

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

***هر بار بی‌نظمی داریم بازه جدید بعد از بی نظمی ۱۲۰ واحد کمتر از بازه قبلی خواهد بود مثلا ۳۲ امین ۸بیت پرارزش 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 ۴۰ امین ۸بیت 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 ۴۸ امین ۸بیت 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 ۵۶ امین ۸بیت میرسیم که طبق نظم با بیت های [۱۲۷:۱۲۰] پر شده است و ۵۷امین ۸ بیتی با بازه [۷:۰] پر شده است.
- از ۵۷ امین مجموعه ۸ تایی با ارزش hash تا آخرین مجموعه باارزش hash (۶۴ امین) 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 00 تا $0H$ نام گذاری کردیم (نامگذاری در مبنای بالاتر از ۱۰ شده است)

### ۲.۴.۲ ورودی skein_round ها

- **کلاک** که همه به کلاک سیستم متصل اند.
- **Round** رجیستر ۳۲ بیتی که به ترتیب ورودی ۰ تا ۱۷ به هر اینستنس داده شده است.
- **p** رجیستر ۵۱۲ بیتی – که اینستنس شماره ۰۱ تا ۰H به ترتیب p۰۱ تا p۰H وصل شده است. به اینستنس شماره ۰۰ هم p۰۰_q وصل شده است.
- **H** رجیستر ۵۷۶ بیتی – که اینستنس شماره ۰۱ تا ۰H به ترتیب h۰۱ تا h۰H وصل شده است. به اینستنس شماره ۰۰ هم h۰۰_q وصل شده است.
- **T۰** رجیستر ۶۴ بیتی – که از اینستنس شماره ۰۰ تا ۰H به ترتیب این دنباله ۳ تایی وصل شده است: t۰_q، t۱_q و t۲_q این دنباله ۳ جمله ای به ترتیب تکرار میشود.

- **T۱** رجیستر ۶۴ بیتی – دقیقا مثل T۰ با این تفاوت که دنباله ۳ تایی t۰_q ، t۱_q و t۲_q به ای شکل است.
- **P۰** رجیستر ۵۱۲ بیتی– که اینستنس شماره ۰۰ تا ۰H به ترتیب o۰۰ تا o۰H وصل شده است.
- **H۰** رجیستر ۵۷۶ بیتی– که اینستنس شماره ۰۰ تا ۰H به ترتیب ho۰۰ تا ho۰H وصل شده است. خط(۱۴۱)

***در ادامه یک always بلاک داریم که حساس به تغییرات همه چیز است. (خط ۱۴۳) در این بلاک متغیر هایی که در انتها d_ دارند مقداردهی میشوند.*** ابتدا phase_d مقدار not متغیر phase_q را به خود اختصاص میدهد.

- **اگر phase_q یک باشد**
  - مقداردهی به p۰۰_d (۵۱۲ بیتی): ۶۴ بیت کم ارزش ( [۶۳:۰]) از data در ۶۴ بیت پرارزش p۰۰_d قرار میگیرد. سپس در ۳۲ بیت بعدی p۰۰_d (از چپ) عینا nonce_le قرار داده میشود. سپس ۳۲ بیت باقیمانده از data ([۹۵:۶۴]) طبق روند قرار داده میشود. باقی بیت های این رجیستر هم با صفر پر میشوند (۳۸۴'d۰) – (خط ۱۴۸)
  - مقداردهی به h۰۰_d (۵۷۶ بیتی): در ۶۴ بیت کم ارزش این ،reg مقدار صفر قرار داده میشود و باقی بیت ها دقیقا به midstate (ورودی ۵۱۲ بیتی) متصل میشوند.
  - مقداردهی به t۰_d (۶۴ بیتی) : h۰۰۰۰۰۰۰۰۰۰۰۰۰۰۵۰
  - مقداردهی به t۱_d (۶۴ بیتی) : hb۰۰۰۰۰۰۰۰۰۰۰۰۰۰۰۰
  - مقداردهی به t۲_d (۶۴ بیتی) : hb۰۰۰۰۰۰۰۰۰۰۰۰۰۰۵۰
  - h_d هم مقدار h_q را به خود میگیرد.
- اگر phase_q صفر باشد
  - مقداردهی به p۰۰_d (۵۱۲ بیتی): این reg با صفر پر میشود.
  - مقداردهی به h۰۰_d (۵۷۶ بیتی):
    - بیت های [۵۷۵:۵۱۲]: hH[۵۷۵:۵۱۲]) + oH[۵۱۱:۴۴۸] )^ data[۶۳:۰]
    - بیت های [۵۱۱:۴۴۸]: hH[۵۱۱:۴۴۸]) + oH[۴۴۷:۳۸۴] )^ data[۹۵:۶۴] nonce۲_le،
    - بیت های [۴۴۷:۳۸۴]: hH[۴۴۷:۳۸۴] + oH[۳۸۳:۳۲۰]
    - بیت های [۳۸۳:۳۲۰]: hH[۳۸۳:۳۲۰] + oH[۳۱۹:۲۵۶]
    - بیت های [۳۱۹:۲۵۶]: hH[۳۱۹:۲۵۶] + oH[۲۵۵:۱۹۲]
    - بیت های [۲۵۵:۱۹۲]: ۶۴'h۰۰۰۰۰۰۰۰۰۰۰۰۰۰۵۰ + hH[۲۵۵:۱۹۲] + oH[۱۹۱:۱۲۸]
    - بیت های [۱۹۱:۱۲۸]: ۶۴'hb۰۰۰۰۰۰۰۰۰۰۰۰۰۰۰۰ + hH[۱۹۱:۱۲۸] + [۶۴ oH[۱۲۷:
    - بیت های [۱۲۷:۶۴]: ]oH[۶۳: ۰] + [۶۴ hH[۱۲۷: + ۱۸
  - مقداردهی به t۰_d (۶۴ بیتی) : h۰۰۰۰۰۰۰۰۰۰۰۰۰۰۰۸
  - مقداردهی به t۱_d (۶۴ بیتی) : hFF۰۰۰۰۰۰۰۰۰۰۰۰۰۰
  - مقداردهی به t۲_d (۶۴ بیتی) : hFF۰۰۰۰۰۰۰۰۰۰۰۰۰۸
  - مقداردهی به h_d (۵۱۲ بیتی):
    - بیت های [۵۱۱:۴۴۸]: ho۰H[۵۷۵:۵۱۲] + o۰H[۵۱۱:۴۴۸]
    - بیت های [۴۴۷:۳۸۴]: ho۰H[۵۱۱:۴۴۸] + o۰H[۴۴۷:۳۸۴]
    - بیت های [۳۸۳:۳۲۰]: ho۰H[۴۴۷:۳۸۴] + o۰H[۳۸۳:۳۲۰]
    - بیت های [۳۱۹:۲۵۶]: ho۰H[۳۸۳:۳۲۰] + o۰H[۳۱۹:۲۵۶]

* بیت های [۲۵۵:۱۹۲]: ho۰H[۳۱۹:۲۵۶] + o۰H[۲۵۵:۱۹۲]
* بیت های [۱۹۱:۱۲۸]: ۶۴'h۰۰۰۰۰۰۰۰۰۰۰۰۰۰۰۸ + ho۰H[۲۵۵:۱۹۲] + o۰H[۱۹۱:۱۲۸]
* بیت های [۱۲۷:۶۴]: ۶۴'hFF۰۰۰۰۰۰۰۰۰۰۰۰۰۰۰ + ho۰H[۱۹۱:۱۲۸] + [۶۴ o۰H[۱۲۷:
* بیت های [۶۳:۰]: o۰H[ :۶۳ ۰ :] + [۶۴ ho۰H[۱۲۷: + ۱۸

***نظم مناسبی دیده میشود به این شکل که به ترتیب ۶۴ بیت پرارزش h_d با مجموع ۶۴ بیت پرارزش o۰H و ۶۴ بیت پرارزش ho۰H پر میشود. به جز ۳ مورد آخر که با اعدادی ثابت جمع میشوند. Always بلاک دوم فقط به لبه مثبت کلاک حساس است. (خط ۲۱۱) (عموما متغیر های q_ مقادیر متناظر d_ را به خود میگیرند)***

- hH مقدار ho۰H را به خود میگیرد.
- oH مقدار o۰H را به خود میگیرد.
- Phase_q مقدار phase_d را به خود میگیرد.
- h_q مقدار h_d را به خود میگیرد.
- t۰_q مقدار t۰_d را به خود میگیرد.
- t۱_q مقدار t۱_d را به خود میگیرد.
- t۲_q مقدار t۲_d را به خود میگیرد.
- در ادامه مجموعه ای از مقدار دهی ها را مربوط به reg های (x p۰x) منظور از ۱ تا (H و x h۰x) منظور از ۱ تا (H داریم. (خط ۲۲۶ تا ۲۶۱)
- h۰x ها: مقدار ho۰y را میگیرند با این تفاوت که y از x یک واحد کمتر است. (به طور مثال h۰۱ مقدار ho۰۰ را به خود میگیرد)
- p۰x ها: مقدار o۰y را میگیرند با این تفاوت که y از x یک واحد کمتر است. (به طور مثال h۰۱ مقدار o۰۰ را به خود میگیرد)
- p۰۰_q مقدار p۰۰_d را میگیرد.
- h۰۰_q مقدار h۰۰_d را میگیرد.
- nonce۲ هم که در ابتدای فایل مقداری مجهول داشت اینجا مقدار nonce (ورودی) منهای ۳۲'d۵۴ را میگیرد.

# فصل ۳

# شبیه‌سازی

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

در ادامه ابتدا کد verilog نوشته‌شده برای Testbench آورده و سپس توضیحاتی دربارهٔ آن ایراد شده است.

### Testbench 1

```
//Master Testbench example

module skein_tb;

// Inputs
reg clk;
reg [511:0] midstate;
reg [95:0] data;
reg [31:0] nonce;

// Outputs
wire [511:0] hash;

// Instantiate the Unit Under Test (UUT)
skein512 uut (
.clk(clk),
.midstate(midstate),
.data(data),
.nonce(nonce),
.hash(hash)
);

initial begin
// Initialize Inputs
clk = 0;
midstate = 0;
data = 0;
nonce = 0;

// Wait 100 ns for global reset to finish
#1000
data = 512'd12345609823;
midstate = 96'd456;
nonce = 32'd453;
#1000;
data = 512'd765943209455554312229760000000654;
midstate = 96'd456;
nonce = 32'd453;

end
always
#1 clk = clk;endmodule
```

### ۲.۲.۳ شکل موج حاصل از Testbench

**Waveform 1**


شکل ۱.۳: شبیه‌سازی با Testbench

### ۳.۲.۳ جدول ورودی‌ها و خروجی‌های Testbench

| (clk) Time | Data | Nonce | Midstate |
|---|---|---|---|
| 0 - 1000 | 0 | 0 | 0 |
| 1000 - 2000 | 512’d12345609823 | 32’d453 | 96’d456 |
| 2000 - End | 512’d765943209455554312229760000000000654 | 32’d453 | 96’d456 |

جدول ۱.۳: مقادیر ورودی‌ها و زمان

| | Hash |
|---|---|
| 433 - 1217 | ab5283d68df053ac62d053789d4b45b81a02c959d7cab97fc43451166351f117<br>f949fe918475f762ba8056704633821146164831 6d4432e6c505edc3b5ee6ff5 |
| 1217 - 1436 | dd477bfb0f07e299560b050c7aedb947bad77571f9a7d886a06f197a55f7946b<br>8a9cecbb948a5478380168f8bfaf8e6d7d828459564973272b18cdf99d0234f2 |
| 1436 - 2217 | 0c0dea4dfd9994c6eb97f500589565239347be8a5b2e4ce4832c6cc9095baa51<br>bf2bdde45ef619f4086e71e7d86f637314357e6d20632c31612f5424644cc223 |
| 2217 - 2437 | 6d383e0cceb223c20c45b816a165072ad200b8091682e8e5c31295ee62ca3719<br>afbd493a4b85859d1cbe08d98bf01e66be18f3d3536987eeef06cc7965851bf8 |
| 2437 - End | 6b722c1b1fb150c850e02ee44e03a447401ca4ac3cde4de6eb95b2e853d0d34b<br>53583685f4b21f9b98229734756d7b835e46c2f589e461ab7c3177fb7e572b64 |

جدول ۲.۳: مقادیر درهم‌سازی و زمان