دانشگاه صنعتی شریف

Skein Hashing

پروژه مستندسازی پیاده‌سازی سخت افزاری الگوریتم به زبان *Verilog*

گروه ۴

سید پارسا اسکندر

کیمیا یزدانی

الهه خدایی

وحید زهتاب

آروین آذرمینا

استاد: فرشاد بهاروند

بهار ۹۸

چکیده‌ی مقاله

در این مقاله که در مورد یکی از روش‌های hash به نام Skein توضیح داده خواهد شد. و هم‌چنین توضیح و مستند برنامه‌های داده شده به دو زبان C به عنوان مدل طلایی و verilog به عنوان طرح سخت‌افزاری ، آورده خواهد شد . در انتها نیز تفاوت‌های این دو مدل و دلایل اختلاف آن‌ها (نقاط اشتباه آن‌ها در پیاده‌سازی) تحلیل خواهد شد.

هم‌جنین منبع اصلی تمامی اطلاعات استفاده شده در این پروژه را اینجا میتوان دید.

فهرست مطالب

فصل ۱

مقدمه

توضیییییییح اسکیییییین

فصل ۲

پیاده‌سازی سخت‌افزاری با Verilog

۱.۲ مقدمه

دنیای نرم‌افزار و سخت‌افزار رایانه در نگاه کلی می‌توانند بسیار شبیه به هم باشند، برنامه های نرم‌افزاری، مقادیری را به عنوان ورودی دریافت کرده، سپس طی روند مشخصی محاسباتی روی آنها انجام داده و در نهایت مقادیری را به عنوان خروجی به کاربر خود تحویل می دهند، قطعات سخت‌افزاری نیز دارای port هایی برای ارتباط با دنیای خارجی و دریافت ورودی و تحویل خروجی‌های خود می‌باشند و از واحدهای مختلف پردازشی و عملیاتی مختلفی برای محاسبه ی خروجی‌های مناسب تشکیل شده اند. درعمل می‌توان سخت‌افزاری خاص برای اجرای بسیاری از روند های نرم‌افزاری طراحی و پیاده سازی کرد، ساخت سخت‌افزارِ خاصِ مربوط به یک الگوریتم می تواند کاربردهای بسیاری داشته باشد، برای مثال قطعه‌ای که بتواند داده های ورودی را رمزنگاری کند می‌تواند به صورت گسترده برای ذخیره ی اطلاعات به صورت سریع استفاده شود، سخت‌افزار های اختصاصیِ الگوریتم ها سریع و بهینه اند و میتوانند به اجرای هرچه سریع تر روندهای پیچیده‌ای که به الگوریتم مورد نظر وابستگی فراوان دارند کمک کلانی کنند.

همانند بسیاری از الگوریتم‌های رایانه‌ای، الگوریتم تابعِ Skein Hashing که در بخش قبل کلیتی از آنرا معرفی کردیم را می توان به صورت سخت افزاری پیاده‌سازی کرد، بدین صورت که قطعه‌ای طراحی و پیاده‌سازی کنیم که ورودی ای به اندازه ی دلخواه مارا دریافت و حاصل درهم‌سازی را به صورت خروجی ای به اندازه ی مورد نظر ما خروجی دهد. بر اساس نیاز و کاربرد ما از این قطعه، اندازه‌ی ورودی و خروجی را می توان ثابت و به مقدار دلخواه درنظر گرفت، سپس قطعه‌ای ثابت با پیاده سازیِ بهینه‌ای برای اندازه های مورد نظر طراحی کرد، یا این که قطعه‌ای برای ورودی و خروجی های با اندازه های متغیر طراحی و پیاده سازی کرد. همان طور که در بخش قبل توضیح داده شد، تابع درهم‌سازی Skein Hashing میتواند ورودی‌ای با اندازه ی دلخواه را دریافت کند و خروجی ای با اندازه‌ی دلخواه تحویل دهد، در این مقاله تمرکز ما روی پیاده سازی سخت‌افزاری حالتی از این تابع می باشد که اندازه ی بلاک های درونی تابع (حالت درونی تابع) **۵۱۲ بیت** و حاصل درهم‌سازی نیز به صورت خروجی ای به اندازه ی **۵۱۲ بیت** می باشد، این تابع در اصطلاح *Skein 512-512* نامیده می‌شود.

مراحل طراحی و پیاده‌سازی سخت‌افزاری یک قطعه معمولا به آن صورت است که برای اطمینان از کارکرد صحیح پیاده‌سازی، موازی با طراحی سخت‌افزاری قطعه، پیاده سازی دیگری از الگوریتم به نام مدل طلایی انجام می‌شود و پس از پایان طراحی‌ها، کارکرد قطعه با مدل طلایی مقایسه می شود تا قطعه‌ی نهایی مشکلی نداشته باشد. در بخش **مدل طلایی** به تفصیل درباره‌ی مدل طلایی استفاده شده در این پروژه توضیح داده شده است. در این بخش به پیاده‌سخت‌افزاری این الگوریتم به کمک زبانِ توصیف سخت‌افزارِ Verilog می پردازیم.

۲.۲ پیاده‌سازی

همان طور که توضیح داده شد، الگوریتم Skein از سه بخش اصلی تشکیل می‌شود:

- ***Threefish*** یا بلاک های رمزنگاری قابل تنظیم،
- ***Unique Block Iteration (UBI)*** که یک حالت زنجیره ای است که از Threefish به صورتی استفاده می کند که ورودیِ به اندازه‌ی دلخواه به خروجی‌ای به اندازه‌ای مشخص تبدیل شود.
- ***Optional Argument System*** که به الگوریتم توانایی پشتیبانی از بسیاری ویژگی های دلخواه را، بدون افزودن باری اضافه به پیاده سازی الگوریتم می‌دهد.

طراحی ای که ما در این پروژه به بررسی‌اش می‌پردازیم، یک طراحی بسیار ساده شده از الگوریتمِ Skein 512-512 می باشد. در این طراحی اندازه ی ورودیِ و خروجی اش ثابت و ۵۱۲ بیت می باشند، و اندازه ی حالت درونی تابع درهم‌سازی نیز دقیقا برابر اندازه‌ی ورودی و خروجی‌هاست، بنابراین در این طراحی اثری از پیاده‌سازی یک UBI پیچیده نیست. علاوه بر این، این پیاده‌سازی، پیاده‌سازی خام خود الگوریتمِ Skein 512-512 بوده و هیچ ویژگی اضافی ای را پشتیبانی نمی‌کند، بنابراین اثری از پیاده‌سازی Optional Argument System نیز در آن نیست. بنابراین طراحی، صرفا شامل بلاک های رمزنگاری قابل تنظیم بوده، داده‌ی ورودی به صورت مستقیم با این بلاک ها تزریغ شده و خروجی الگوریتم نیز به صورت تقریبا مستقیم از آخرین بلاک دریافت می‌شود.

۱.۲.۲ بررسی درستی طراحی ارائه شده

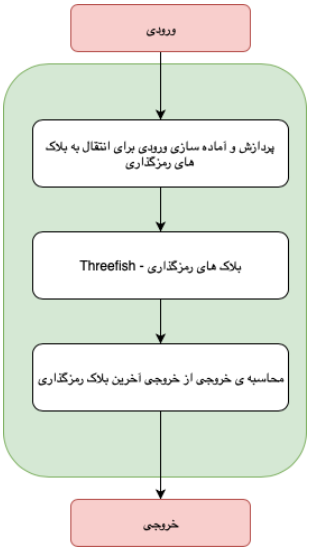
طراحی‌ای که دراختیار ما قرار گرفته بود، شامل مشکلات منطقی فراوان بود، مشکلات موجود در هر خط از کد verilog به صورت یک comment در قالب

$$todo :< ValidCode >$$

مشخص شده اند، علاوه بر این طراحی تصحیح شده نیز در کنار مقاله پیوست شده است، تمامی مستند سازی های این مقاله، براساس پیاده سازی تصحیح شده ی کدهای اولیه می‌باشد.

۲.۲.۲ ساختار طراحی

ساختار کلی طراحی شده برای این الگوریتم به سه بخش که در تصویر زیر قابل مشاهده اند قابل تفکیک است، ماژول اصلی طراحی که skein512 نام گذاری شده است، این سه وظیفه را برعهده دارد.



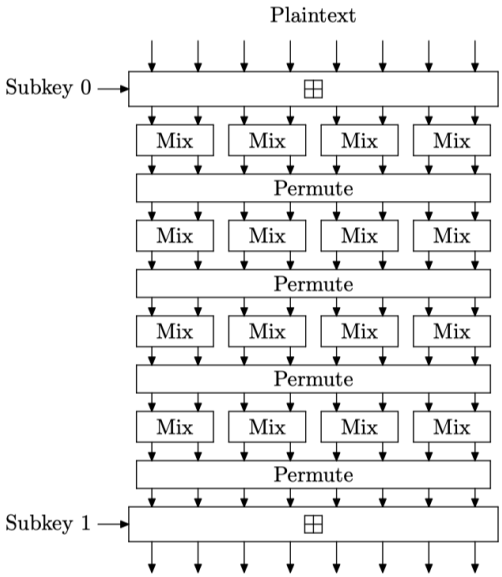
شکل ۱.۲: ساختار کلی پیاده‌سازی سخت‌افزاری

دو بخش پردازش ورودی و خروجی‌ها به صورت کامل در ماژول skein512 صورت می گیرند و به تفصیل درباره ی آنها توضیح داده خواهد شد، بخش بلاک های رمزگذاری از ماژول های skein__round و skein__round__1 تا skein__round__4 تشکیل شده و قرارگیری زنجیره‌ای آنها و محاسبه ی اولیه‌ی مقادیر کلیدهای رمزنگاری در ماژول skein512 صورت گرفته است.

۳.۲.۲ پیاده سازی بلاک‌های رمزگذاری

همان‌طور که در معرفی الگوریتم Skein Hashing در بخش اول توضیح داده شد، این الگوریتم برای تولید مقدار درهم‌سازی از بلاک های رمزگذاری که به صورت زنجیره ای یکی پس از دیگری قرار گرفته اند، استفاده می کند و این بلاک‌ها بخش عمده‌ای از طراحی سخت‌افزاری را دربر می‌گیرند.

شمای کلی حرکت داده داخل بلاک های رمزنگاری در این الگوریتم به شکل زیر می باشد:



شکل ۲.۲: شمایی از حرکت داده در بلاک های رمزنگاری

داده‌ی ورودی که به ۸ بخش ۶۴ بیتی تقسیم می‌شود، به بلاک‌های رمزنگاری تزریق شده و پس از ۴ مرحله ی متوالی از درهم‌سازی (Mix) و جابه‌جایی (Permutation) - که به هر یک از آنها یک $round$ می‌گوییم - با مقادیری به نام $Subkey$ که از کلیدهای رمزنگاری - که آنها نیز از ۸ بلاک ۶۴ بیتی تشکیل شده اند - به دست می‌آیند، جمع می‌شوند.

محاسبات دقیق Subkey ها و توابع غیرخطی درهم‌سازی (Mix) و جابه‌جایی (Permutation) هر round به تفصیل در بخش اول مقاله، توضیح داده شده است. آن چیزی که دراین میان حائز اهمیت است، استفاده‌ای هوشمندانه از نظم تکراری این توابع محاسباتی در پیاده‌سازی سخت‌افزاری مورد بررسی ما در این مقاله است.

پیاده سازی توابع جابه‌جایی (Permutation) هر round

تابع غیرخطیِ جابه‌جایی (Permutation) یک عملیات ثابت را روی مقادیر خروجی از توابع درهم‌سازی (Mix) انجام می‌دهد، این تابع در ماژول‌های skein_round_1 تا skein_round_4 به صورت توصیف رفتاری با کمک یک always block حساس به لبه‌ی بالارونده‌ی ساعت پیاده سازی شده است. مقادیر خروجی توابع درهم‌سازی (Mix) مربوط به هر ماژول - که جلوتر پیاده‌سازی آنها را بررسی خواهیم کرد -، به هنگام لبه‌ی بالارونده ی ساعت، جابه‌جا شده و به خروجی ماژول انتقال داده می‌شوند.

توصیف ارائه شده از جابه‌جایی (Permutation) در این always block ها در واقع معرف مجموعه ای از ۵۱۲ D-FlipFlop حساس به لبه‌ی بالارونده ی ساعت می باشد.

پیاده سازی توابع درهم‌سازی (Mix) هر round

برخلاف توابع جابه‌جایی (Permutation) که یک عملیات ثابت را در هر round اجرا می‌کنند، این توابع غیرخطی، بر اساس این که در کدام round قرار دارند، محاسبات خاص خود را خواهند داشت، همان طور که در بخش قبل توضیح داده شد، هر درهم‌سازی (Mix) شامل، یک جمع، یک گردش به چپ و یک xor می باشد. جمع و xor ها در همه‌ی round ها یکسان اند و به صورت یکتا پیاده می‌شوند.

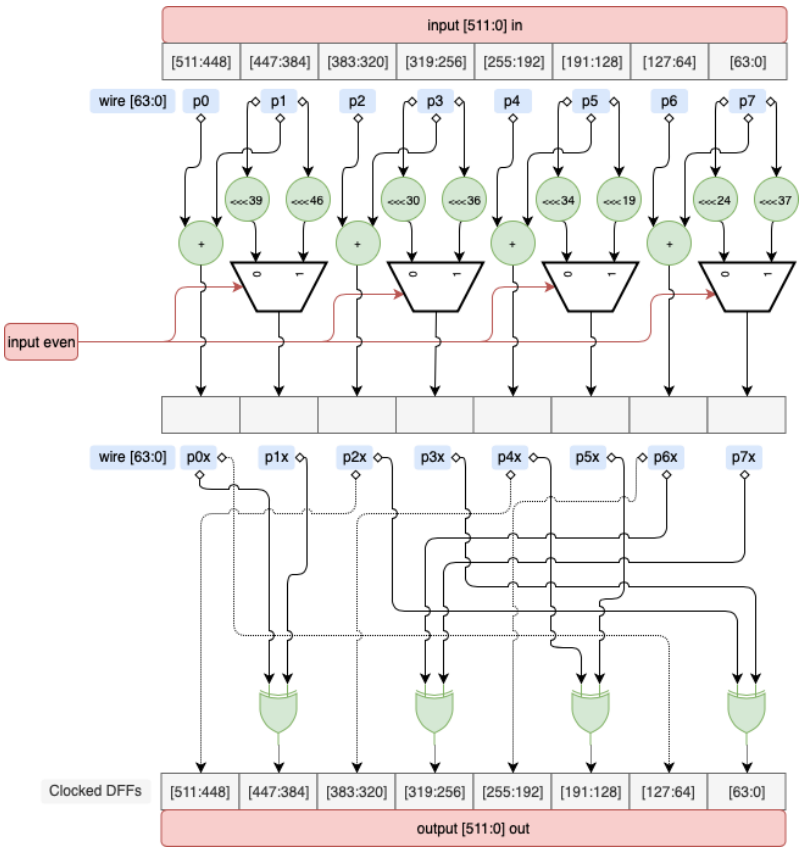
چیزی که در هر round متفاوت خواهد بود، تعداد گردش ها به چپ می‌باشد، با توجه به مقادیر ارائه شده و فورمول‌های توضیح داده شده در بخش قبل، این مقادیر هر ۸ round تکرار می شوند، مسئله ی دیگر آن است که قرار است هر ۴ round ، subkey های مربوطه به مقادیر موجود در بلاک‌های رمزگذاری افزوده شوند، برای همین در پیاده‌سازی مورد بررسی ما هر ۴ round در یک ماژول به نام skein_round در نظر گرفته شده است و پیاده سازی خود round ها در ماژول های skein_round_1 تا skein_round_4 تعریف شده اند. این بدین معنا است که برای دوره ی تناوب ۸ تایی توابع درهم‌سازی (Mix) هر round ، صرفا دانستن این که در ۴ round شماره ی فرد یا زوج قرار داریم برای ماژول های skein_round_1 تا skein_round_4 کافی می باشد. برای همین این ماژول‌ها یک ورودی تک بیتی به نام even دریافت می کنند و بر اساس آن تعداد گردش به چپ‌های مناسب را بر می گزینند.

پیاده‌سازی خود توابع درهم‌سازیِ (Mix) هر round ، در دو بخش در ماژول های skein_round_1 تا skein_round_4 تعریف شده است. در بخش اول عملیات‌های جمع و گردش به چپ‌ها به توصیفی ساختاری و به کمک یک سری continuous assignment در این ماژول‌ها مشخص شده اند، لازم به ذکر است که در verilog عملگر گردش به چپ وجود ندارد، برای همین برای پیاده سازی گردش به چپِ یک vector، از ترکیب part selection و concatenation استفاده شده است. در بخش دوم عملیات‌های xor هم‌زمان با عملیات‌های جابه‌جایی (Permutation) در always block مربوط به آنها در این ماژول‌ها معرفی شده است.

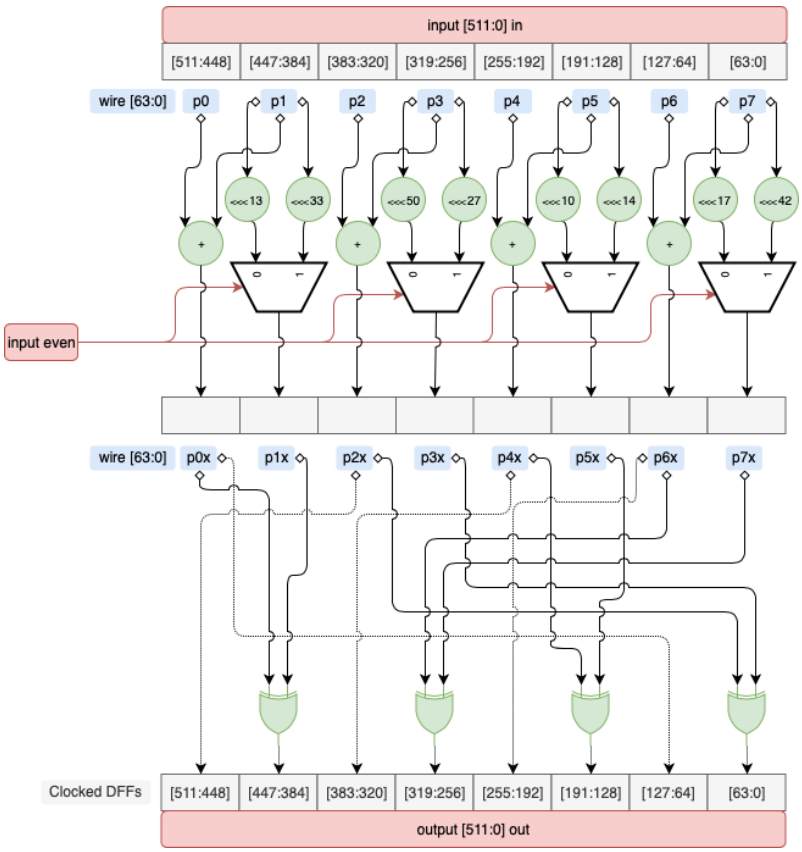
توصیف ارائه شده از توابع درهم‌سازی (Mix) در عمل معرف مجموعه ای از مدارهای ترکیبی (combinational) می باشد و منجر به تولید گیت‌های منطقی مستقل از سیگنال ساعت خواهد شد.

پیاده سازی round ها

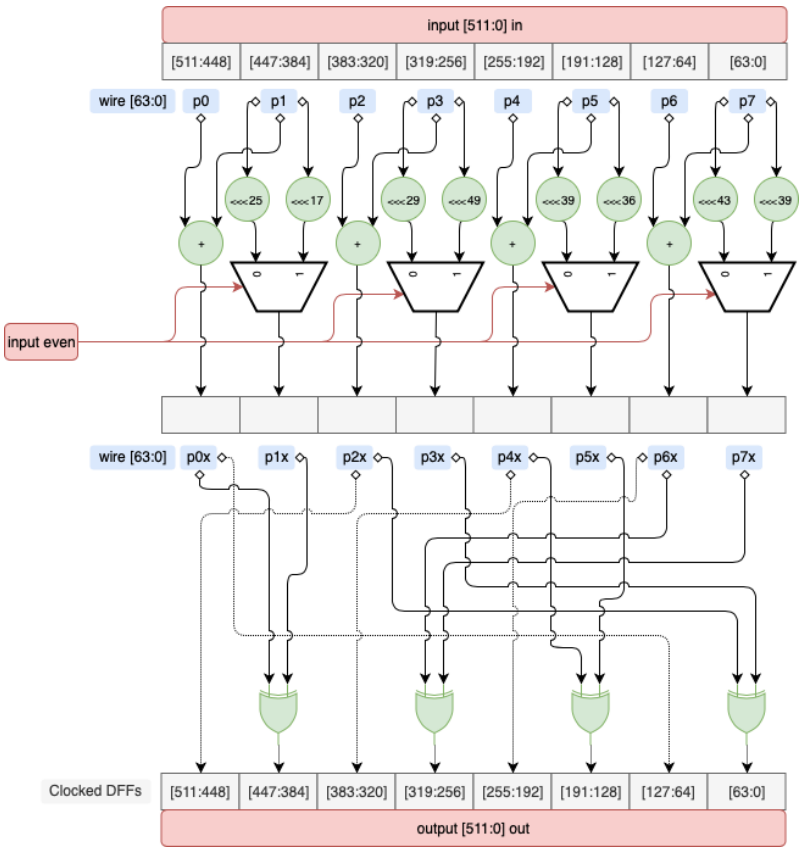
همان طور که بار ها گفته شد round شامل درهم‌سازی (Mix) و جابه‌جاییِ (Permutation) روی ورودی‌هایش می‌باشد که به تفصیل به پیاده سازی این دو تابع غیرخطی در طراحی مورد بررسی‌مان پرداختیم، تنها مسئله ی مجهول شیوه‌ی اتصال این دو بخش در ماژول های skein_round_1 تا skein_round_4 و تشکیل واحد های محاسباتی round های الگوریتم می‌باشد که بلاک دیاگرام های تهیه شده در تصاویر دو صفحه‌ی بعدی به تفصیل این مسئله و هم‌چنین شمای کلی حرکت داده داخل این ماژول‌ها را توضیح می دهند:



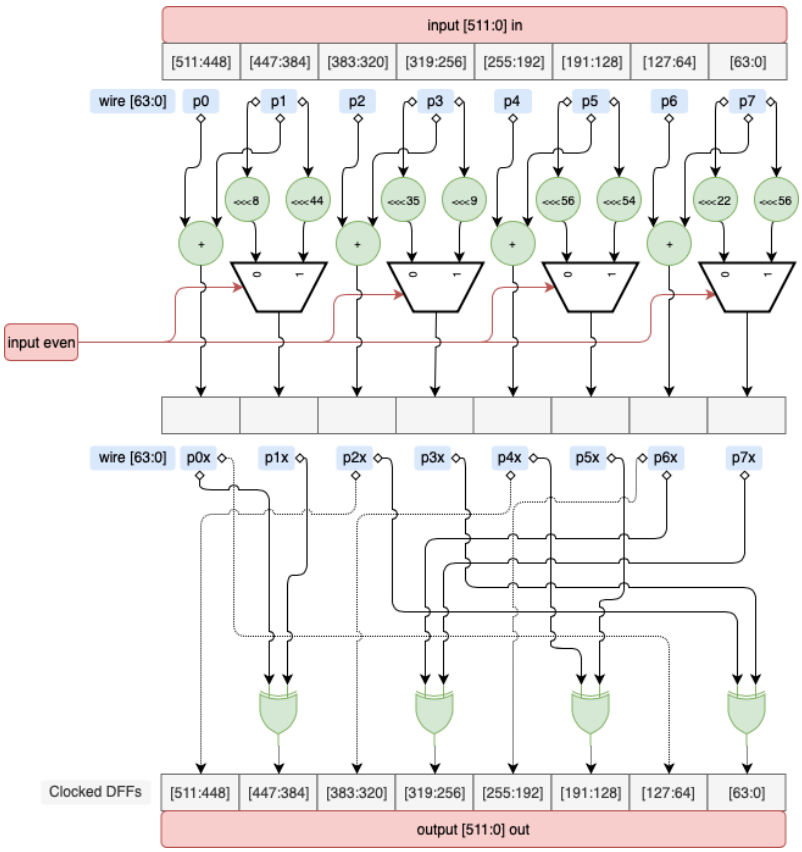
شکل ۳.۲: بلاک دیاگرام شماتیک و نحوه‌ی جریان داده در ماژول skein_round_1



شکل ۴.۲: بلاک دیاگرام شماتیک و نحوه‌ی جریان داده در ماژول skein_round_2



شکل ۵.۲: بلاک دیاگرام شماتیک و نحوه‌ی جریان داده در ماژول skein_round_3



شکل ۶.۲: بلاک دیاگرام شماتیک و نحوه‌ی جریان داده در ماژول skein_round_4

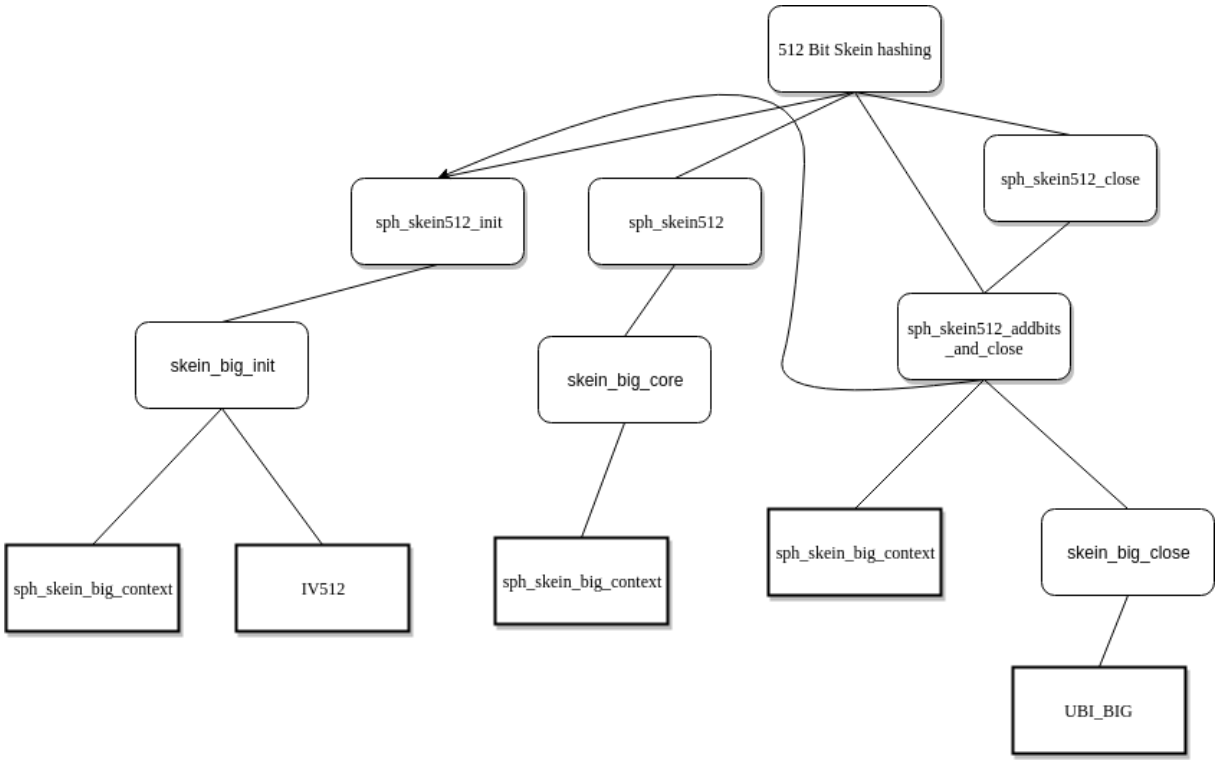
محاسبه و افزودن Subkey ها

فصل ۳

مدل طلایی

۱.۳ مقدمه

در مدل طلایی ۴ نوع متفاوت از Skein hash آورده شده‌است (۲۲۴ و ۲۵۶ و ۳۸۴ و ۵۱۲ بیت) که همانطور که در مدل طراحی شده با verilog نیز تنها نوع استاندارد (۵۱۲ بیت)آن پیاده‌سازی شده است ، در مدل طلایی نیز تنها توضیحات و مستندات این نوع ارائه خواهد شد.



۲.۳ پیاده‌سازی الگوریتم

در شکل بالا تمامی توابع و ساختارهای مورد نیاز و سلسله مراتب آن‌ها برای نوع ۵۱۲ بیتی الگوریتم آورده شده است ، برای توضیح نحوه‌ی اجرای الگوریتم با شروع از sph-skein-big-context سلسله اجرای برنامه توضیح داده خواهد شد.

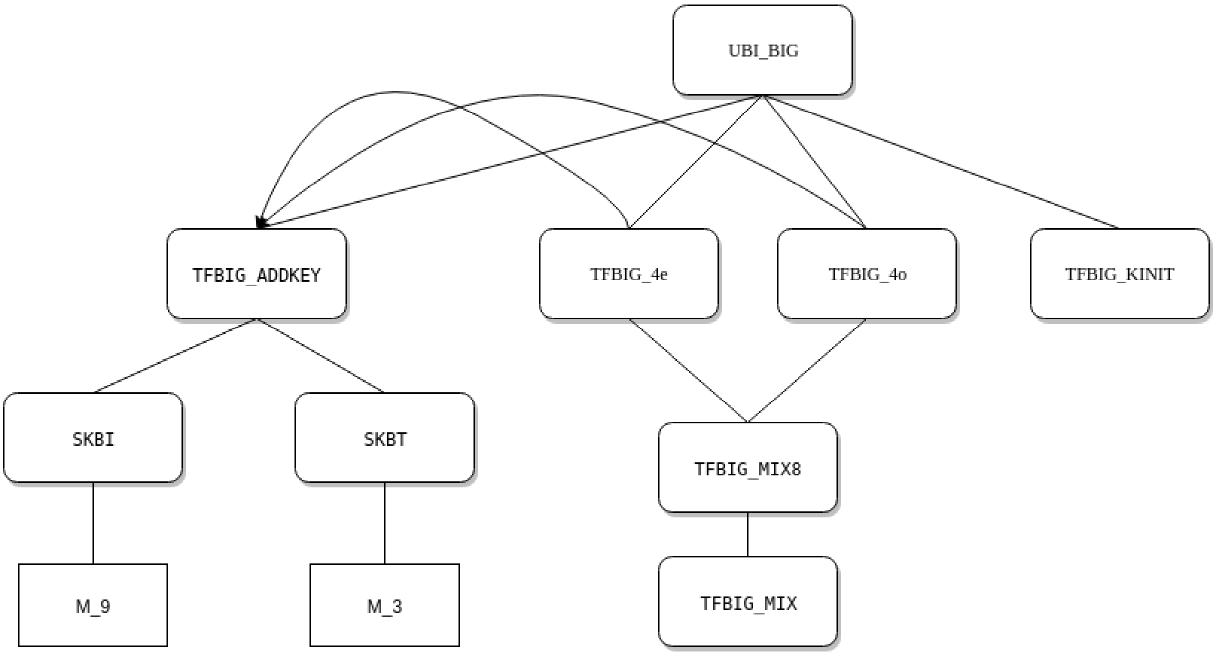
در این برنامه برای ذخیره و استفاده از هش ، از ساختاری استفاده شده است به نام sph-skein-big-context استفاده شده است و هدف برنامه اجرای الگوریتم هش و ذخیره‌ی خروجی در این ساختار است.

برای اجرای الگوریتم هش ۵۱۲ بیتی ، در سلسله‌ی اجرا از توابع زیر استفاده شده است :

در ابتدا برنامه با ذخیره‌ی مقادیر از پیش تعیین شده IV512 در ساختار معرفی شده شروع به کار می‌کند ، و این کار توسط تابع sph-skein512-init انجام میگردد.

سپس با در نظر گرفتن ورودی و سایز این ورودی، اجرای الگوریتم هش توسط تابع sph-skein512 شروع می‌شود و ورودی داده شده تبدیل به هش میشود و با ذخیره شدن در ساختار هش، این تابع پایان می‌پذیرد.

حال برای فهم درست از توابع مورد استفاده لازم است نحوه‌ی پیاده‌سازی UBI-BIG توضیح داده‌شود. تمامی سلسله مراتب طراحی آن در شکل زیر آورده شده‌است.



یه توضیح عه کوچیک برااا یو بی آی بیگ

فصل ۴

توابع و ساختارها

۱.۴ ساختارها

۱.۱.۴ sph-skein-big-context

این ساختار مورد نظر برای ذخیره و استفاده از هش است (شامل مقادیری از هش قبلی و مقادیر جدید محاسبه شده).
این ساختار شامل یک آرایه‌ی ۶۴ بیتی از کاراکترهاست که به منظور تراز کردن انواع هش استفاده می‌گردد و هشت عدد ۶۴ بیتی که برای ذخیره‌ی ۵۱۲ بیت هش استفاده می‌شوند و هم‌چنین شامل دو عدد با نام‌های ptr, bcount است که این دو عدد به طور معمول برابر ۰ هستند که همانند nonce در پیاده‌سازی وریلاگ آن است.

۲.۱.۴ IV512

این ساختار شامل مقادیر اولیه‌ی هش است. یک عدد ۵۱۲ بیتی را برای خوانا بودن در مبنای ۱۶ و در ۸ بلاک ۱۶ بیتی نگاه می‌دارد. این مقدار در برنامه به زبان وریلاگ همان midstate است.

۳.۱.۴ UBI-BIG

این تابع (در مدل‌طلایی به صورت define تعریف شده) طبق الگوریتم skein در ابتدا وظیفه‌ی ۵۱۲ بیتی کردن ورودی در بافر را بر عهده دارد، در ادامه تمامی ۷۲ مرحله‌ی هش الگوریتم skein را که در مقدمه شرح داده شده است را اجرا می‌کند.
روند اجرای این تابع بدین صورت است که در ابتدا مقادیر ۸ بیتی در بافر را با استفاده از Encoder به مقادیر ۶۴ بیتی تبدیل می‌کند، سپس دو مقدار $t_{۰}, t_{۱}$ را که همان $tweak$ ها هستند را با استفاده از ورودی‌ها به‌دست می‌آورد و سپس با استفاده از تابع TFBIG-INIT مقادیر جدیدی از داده‌های قبلی به دست می‌آورد، سپس ۱۸ بار توابع TFBIG-4e و TFBIG-4o صدا می‌شوند که در هر کدام از این تابع‌ها ۴ بار تابع درهم‌سازی توضیح داده شده در مقدمه صدا می‌شوند.

۴.۱.۴ TFBIG-4e و TFBIG-4o

این تابع برای کدگذاری $P_{۰}$ تا $P_{۷}$ طراحی شده‌است. همان‌طور که پیش‌تر توضیح داده شده است، ۷۲ بار تابع درهم‌سازی صدا می‌شود، و هر ۸ سلسله از این ۷۲ مرحله یکسان است، هم‌چنین در هر ۸ سری ۴ بار با یک کلید و ۴ بار دیگر با یک کلید دیگر اجرا می‌شود، که به همین دلیل این توابع هر کدام برای آن ۴ باری استفاده می‌شود که در مرحله‌ای زوج یا فرد قرار داریم.
این تابع یک ورودی s دارد. تابع TFBIG-ADDKEY با $p_{۰}$ تا $p_{۷}$ و h و t و s به ترتیب به عنوان $w_{۰}$ تا $w_{۷}$ و k و t و s صدا شده‌است. h و t برای concat و ساختن کلید در تابع TFBIG-ADDKEY استفاده شده‌اند. سپس TFBIG-MIX8 چهار بار برای ترتیب‌های متفاوتی از $p_{۰}$ تا $p_{۷}$ اعداد متفاوت به عنوان rc صدا شده‌است. ترتیب صدا شدن $p_{۰}$ تا $p_{۷}$ برای تعداد بلاک ۸ به صورت جدول‌های زیر است، که برای هر راند از ۰ تا ۳، بر حسب راند قبل ترتیب‌ها چهار عدد جابه‌جا شده‌اند. و تفاوت حالت‌های زوج و فرد در اعداد استفاده شده است.

N_w		4		8				16							
j		0	1	0	1	2	3	0	1	2	3	4	5	6	7
	0	14	16	46	36	19	37	24	13	8	47	8	17	22	37
	1	52	57	33	27	14	42	38	19	10	55	49	18	23	52
	2	23	40	17	49	36	39	33	4	51	13	34	41	59	17
$d=$	3	5	37	44	9	54	56	5	20	48	41	47	28	16	25
	4	25	33	39	30	34	24	41	9	37	31	12	47	44	30
	5	46	12	13	50	10	17	16	34	56	51	4	53	42	41
	6	58	22	25	29	39	43	31	44	47	46	19	42	44	25
	7	32	32	8	35	56	22	9	48	35	52	23	31	37	20

		$i=$															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	4	0	3	2	1												
$N_w=$	8	2	1	4	7	6	5	0	3								
	16	0	9	2	13	6	11	4	15	10	7	12	3	14	5	8	1

TFBIG-ADDKEY ۵.۱.۴

این تابع طبق فرمول‌های زیر مقادیر ورودی را تغییر می‌دهد و برای اینکار از تابع‌های SKBI و SKBT استفاده می‌کند که به ترتیب جمع مقادیر ورودی‌شان را به پیمانه ۳ و ۹ محاسبه می‌کنند.

$$k_{s,i} = k_{(s+i) \mod 9} \qquad i = 0, 1, 2, ..., 4$$
$$k_{s,5} = k_{(s+5) \mod 9} + t_{s \mod 3}$$
$$k_{s,6} = k_{(s+6) \mod 9} + t_{(s+1) \mod 3}$$
$$k_{s,7} = k_{(s+7) \mod 9} + s$$

دقت شود که تمامی این محاسبات برای نوع ۵۱۲ بیتی الگوریتم است.

SKBT و SKBI ۶.۱.۴

تابع SKBI برای محاسبه‌ی اندیس کلید استفاده شده‌است.

در الگوریتم برای تولید k. تا $k_۸$ از این ماکرو استفاده شده است. ،این ماکرو k و s و i را گرفته و سپس k را به M9-s-i متصل می‌کند که باقی‌مانده‌ی $s+i$بر ۹ تعریف شده‌است. تابع SKBT مشابه تابع بالاست با این تفاوت که در انتها باقی‌مانده عدد را بر ۳ محاسبه می‌کند و از این تابع برای به‌دست آوردن اندیس tweak ها استفاده می‌شود.

TFBIG-MIX8 ۷.۱.۴

همان‌طور که در مقدمه گفته شده‌است، هر سری از هشت سری، چهار round دارد، پس طراحی این تابع برای ساده‌سازی استفاده‌ی متداول از TFBIG-MIX بوده‌است. به صورت متداول در کد به چهار سری استفاده از TFBIG-MIXپشت سر هم نیاز است.

TFBIG-MIX ۸.۱.۴

وظیفه‌ی این تابع درهم سازی بلاک‌های ورودی طبق فرمول‌های زیر است.

$$y_۰ = (x_۰ + x_۱) \mod ۲^{۶۴}$$
$$y_۱ = (x_۱ <<< R_{(d \mod ۸),j}) \oplus y_۰$$

که مقادیر R در جدول زیر آمده است :

N_w		4		8				16							
j		0	1	0	1	2	3	0	1	2	3	4	5	6	7
	0	14	16	46	36	19	37	24	13	8	47	8	17	22	37
	1	52	57	33	27	14	42	38	19	10	55	49	18	23	52
	2	23	40	17	49	36	39	33	4	51	13	34	41	59	17
$d =$	3	5	37	44	9	54	56	5	20	48	41	47	28	16	25
	4	25	33	39	30	34	24	41	9	37	31	12	47	44	30
	5	46	12	13	50	10	17	16	34	56	51	4	53	42	41
	6	58	22	25	29	39	43	31	44	47	46	19	42	44	25
	7	32	32	8	35	56	22	9	48	35	52	23	31	37	20

TFBIG-INIT ۹.۱.۴

این تابع با ورودی‌های t_0 تا t_2 و h_0 تا h_8 مقادیر زیر را محاسبه می‌کند :

$$k_8 = C \oplus k_0 \oplus k_1 \oplus ... \oplus k_7$$

$$t_2 = t_1 \oplus t_0$$

که مقدار ثابت C به آن جهت در فرمول وجود دارد که از ۰ نبودن تمامی بیت‌ها اطمینان حاصل شود.

توابع ۲.۴

sph-skein512-init ۱.۲.۴

این تابع مسئولیت مقداردهی اولیه‌ی ساختار هش را بر عهده دارد، که برای آن تابع skein-big-init را با ورودی اولیه‌ی IV512 اجرا می‌کند.

skein-big-init ۲.۲.۴

این تابع دو ورودی می‌پذیرد که یکی از آن‌ها آدرس یک ساختار هش است و دیگری مقدار اولیه، که مقادیر متناظر ساختار داده شده را برابر مقادیر اولیه قرار می‌دهد. که مقدار اولیه در حالت ۵۱۲ بیتی در ساختار IV512 ذخیره شده است.

sph-skein512 ۳.۲.۴

این تابع، مقدار هش محاسبه شده تا این لحظه را از بین می‌برد و