



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.

همچنین منبع اصلی تمامی اطلاعات استفاده شده در این پروژه را [اینجا](#) میتوان دید.

فهرست مطالب

۳	۱	مقدمه
۴	۲	پیاده‌سازی سخت‌افزاری Verilog
۴	۱.۲	مقدمه
۴	۲.۲	پیاده‌سازی
۶	۳	مدل طلایی
۶	۱.۳	مقدمه
۶	۲.۳	پیاده‌سازی الگوریتم
۸	۴	توابع و ساختارها
۸	۱.۴	ساختارها
۸	۱.۱.۴	sph-skein-big-context
۸	۲.۱.۴	IV512
۸	۳.۱.۴	UBI-BIG
۸	۴.۱.۴	TFBIG-4o,TFBIG-4e
۹	۵.۱.۴	TFBIG-ADDKEY
۹	۶.۱.۴	SKBI و SKBT
۹	۷.۱.۴	TFBIG-MIX8
۹	۸.۱.۴	TFBIG-MIX
۱۰	۹.۱.۴	TFBIG-INIT
۱۰	۲.۴	توابع
۱۰	۱.۲.۴	sph-skein512-init
۱۰	۲.۲.۴	skein-big-init
۱۰	۳.۲.۴	sph-skein512

فصل ۱

مقدمه

توضیحات اسکیزم

فصل ۲

پیاده‌سازی سخت‌افزاری Verilog

۱.۲ مقدمه

دنیای نرم‌افزار و سخت‌افزار رایانه در نگاه کلی می‌توانند بسیار شبیه به هم باشند، برنامه‌های 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 می‌دهند، 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 محاسبه‌ی خروجی‌های مناسب تشکیل شده‌اند. درعمل می‌توان سخت‌افزاری خاص برای اجرای بسیاری از روند‌های نرم‌افزاری طراحی و پیاده‌سازی کرد، ساخت سخت‌افزار خاص مربوط به یک الگوریتم می‌تواند کاربردهای بسیاری داشته باشد، برای مثال قطعه‌ای که بتواند داده‌های ورودی را رمزنگاری کند می‌تواند به صورت گسترده برای ذخیره‌ی اطلاعات به صورت سریع استفاده شود، سخت‌افزارهای اختصاصی الگوریتم‌ها سریع و بهینه‌اند و میتوانند به اجرای هرچه سریع‌تر روندهای پیچیده‌ای که به الگوریتم مورد نظر وابستگی فراوان دارند کمک کلانی کنند.

همانند بسیاری از الگوریتم‌های رایانه‌ای، الگوریتم تابع Skein Hashing که در بخش قبل کلیتی از آنرا معرفی کردیم را می‌توان به صورت سخت‌افزاری پیاده‌سازی کرد، بدین صورت که قطعه‌ای طراحی و پیاده‌سازی کنیم که ورودی‌ای به اندازه‌ی دلخواه مارا دریافت و حاصل درهم‌سازی را به صورت خروجی‌ای به اندازه‌ی مورد نظر ما خروجی دهد. بر اساس نیاز و کاربرد ما از این قطعه، اندازه‌ی ورودی و خروجی را می‌توان ثابت و به مقدار دلخواه در نظر گرفت، سپس قطعه‌ای ثابت با پیاده‌سازی بهینه‌ای برای اندازه‌های مورد نظر طراحی کرد، یا این که قطعه‌ای برای ورودی و خروجی‌های با اندازه‌های متغیر طراحی و پیاده‌سازی کرد. همان‌طور که در بخش قبل توضیح داده شد، تابع درهم‌سازی Skein Hashing میتواند ورودی‌ای با اندازه‌ی دلخواه را دریافت کند و خروجی‌ای با اندازه‌ی دلخواه تحویل دهد، در این مقاله تمرکز ما روی پیاده‌سازی سخت‌افزاری حالتی از این تابع می‌باشد که اندازه‌ی بلاک‌های درونی تابع (حالت درونی تابع) ۵۱۲ بیت و حاصل درهم‌سازی نیز به صورت خروجی‌ای به اندازه‌ی ۵۱۲ بیت می‌باشد، این تابع در اصطلاح Skein 512-512 نامیده می‌شود.

مراحل طراحی و پیاده‌سازی سخت‌افزاری یک قطعه معمولاً به آن صورت است که برای اطمینان از کارکرد صحیح پیاده‌سازی، موازی با طراحی سخت‌افزاری قطعه، پیاده‌سازی دیگری از الگوریتم به نام مدل طلایی انجام می‌شود و پس از پایان طراحی‌ها، کارکرد قطعه با مدل طلایی مقایسه می‌شود 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 پیاده‌سازی سخت‌افزاری این الگوریتم به کمک زبان توصیف سخت‌افزار Verilog می‌پردازیم.

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 ورودی به اندازه دلخواه به خروجی ای به اندازه ای 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 می دهد.

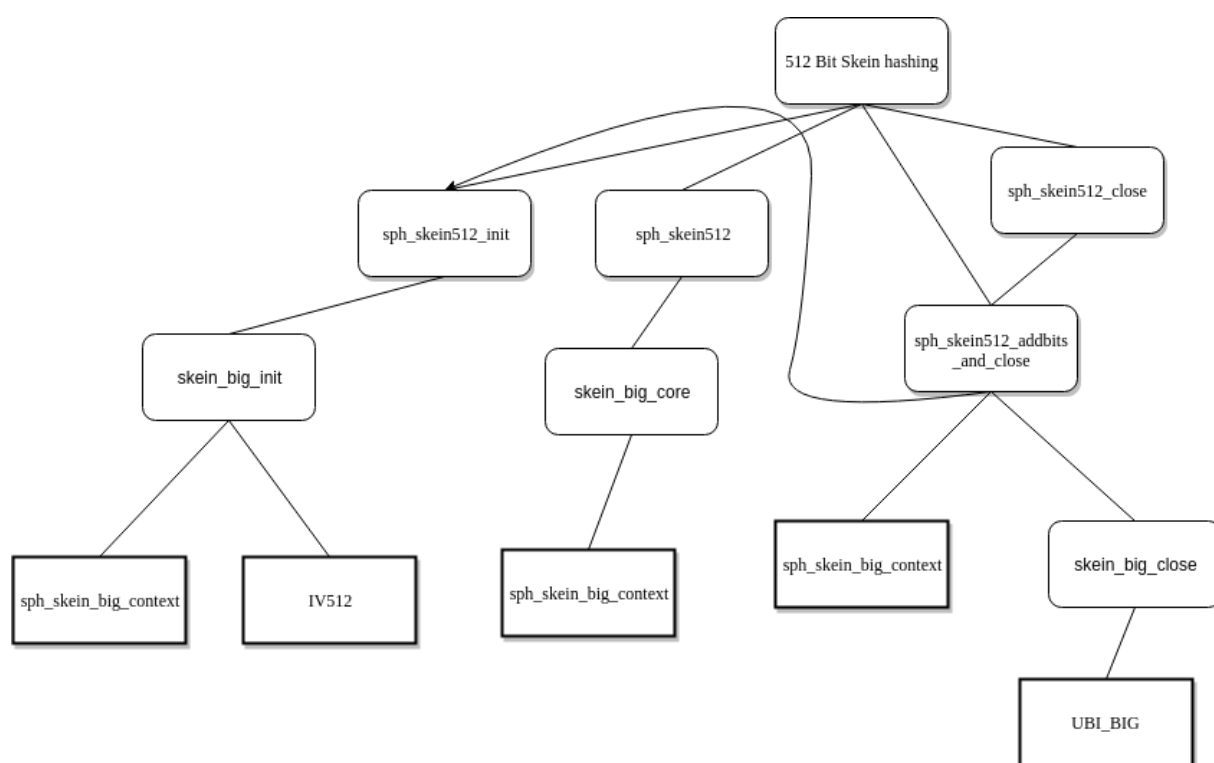
طراحی ای که ما در این پروژه به بررسی اش می پردازیم، یک طراحی بسیار ساده شده از الگوریتم Skein 512-512 می باشد. در این طراحی اندازه ی ورودی و خروجی اش ثابت و ۵۱۲ بیت می باشند، و اندازه ی حالت درونی تابع درهم سازی نیز دقیقا برابر اندازه ی ورودی و خروجی هاست، بنابراین در این طراحی اثری از پیاده سازی یک UBI پیچیده نیست. علاوه بر این، این پیاده سازی، پیاده سازی خام خود الگوریتم Skein 512-512 بوده و هیچ ویژگی اضافی ای را پشتیبانی نمی کند، بنابراین اثری از پیاده سازی Optional Argument System نیز در آن نیست. بنابراین طراحی، صرفا شامل بلاک های رمزنگاری قابل تنظیم بوده، داده ی ورودی به صورت مستقیم با این بلاک ها تزریق شده و خروجی الگوریتم نیز به صورت تقریبا مستقیم از آخرین بلاک دریافت می شود.

فصل ۳

مدل طلایی

۱.۳ مقدمه

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



۲.۳ پیاده سازی الگوریتم

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

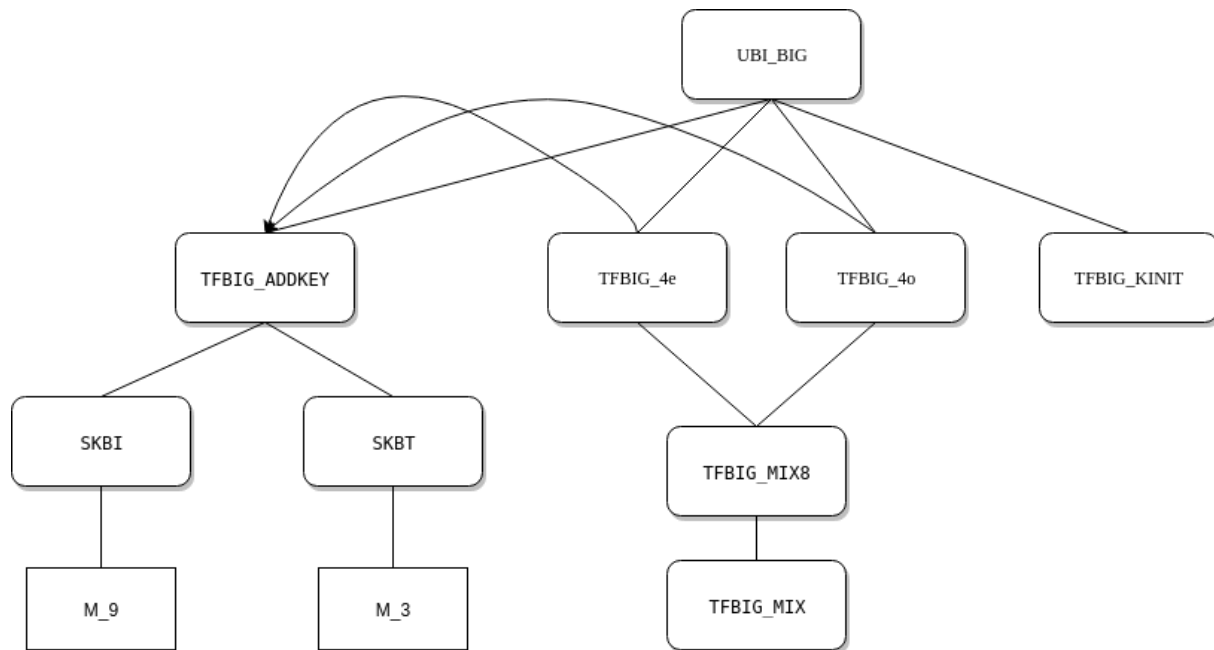
در این برنامه برای ذخیره و استفاده از هش ، از ساختاری استفاده شده است به نام sph-skein-big-context استفاده شده است و هدف برنامه اجرای الگوریتم هش و ذخیره ی خروجی در این ساختار است.

برای اجرای الگوریتم هش ۵۱۲ بیتی ، در سلسله ی 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 سائز 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 برآاا 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 مقداری 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 `bcount`، `ptr` 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 مدل‌طلائی 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_0 ، t_1 را که همان `tweak`‌ها هستند را با استفاده از ورودی‌ها به دست 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 توابع ۴ بار تابع درهم‌سازی توضیح داده شده در مقدمه صدا می‌شوند.

۴.۱.۴ TFBIG-4o و TFBIG-4e

این تابع برای کدگذاری P تا P_V طراحی شده است. 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 تا p_V و h و to S به ترتیب به عنوان w ، w_V و tk و t و s صدا شده است. h و t برای `concat` و ساختن کلید در تابع `TFBIG-ADDKEY` استفاده شده‌اند. سپس `TFBIG-MIX8` چهار بار برای ترتیب‌های متفاوتی از p تا p_V اعداد متفاوت به عنوان TC صدا شده است. ترتیب صدا شدن p تا p_V 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		
$d =$	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		
	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		
	$i =$																
$N_w =$	4	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	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.

$$\begin{aligned} k_{s,i} &= k_{(s+i) \bmod 9} & i &= 0, 1, 2, \dots, 4 \\ k_{s,5} &= k_{(s+5) \bmod 9} + t_{s \bmod 3} \\ k_{s,6} &= k_{(s+6) \bmod 9} + t_{(s+1) \bmod 3} \\ k_{s,7} &= k_{(s+7) \bmod 9} + s \end{aligned}$$

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

SKBI و SKBT ۶.۱.۴

تابع SKBI برای محاسبه‌ی اندیس کلید استفاده شده‌است. در الگوریتم برای تولید k_i تا k_8 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-MIX پشت سر هم نیاز است.

TFBIG-MIX ۸.۱.۴

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

$$\begin{aligned} y_i &= (x_i + x_1) \bmod 2^{64} \\ y_1 &= (x_1 \lll R_{(d \bmod 8),j}) \oplus y_i. \end{aligned}$$

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
$d =$	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				
	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 تا t_{γ} و h تا h_{λ} مقادیر زیر را محاسبه می‌کند:

$$k_{\lambda} = C \oplus k_{\cdot} \oplus k_{\gamma} \oplus \dots \oplus k_{\gamma}$$

$$t_{\gamma} = t_{\gamma} \oplus t.$$

که مقدار ثابت C به آن جهت در فرمول وجود دارد که از \bullet 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