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

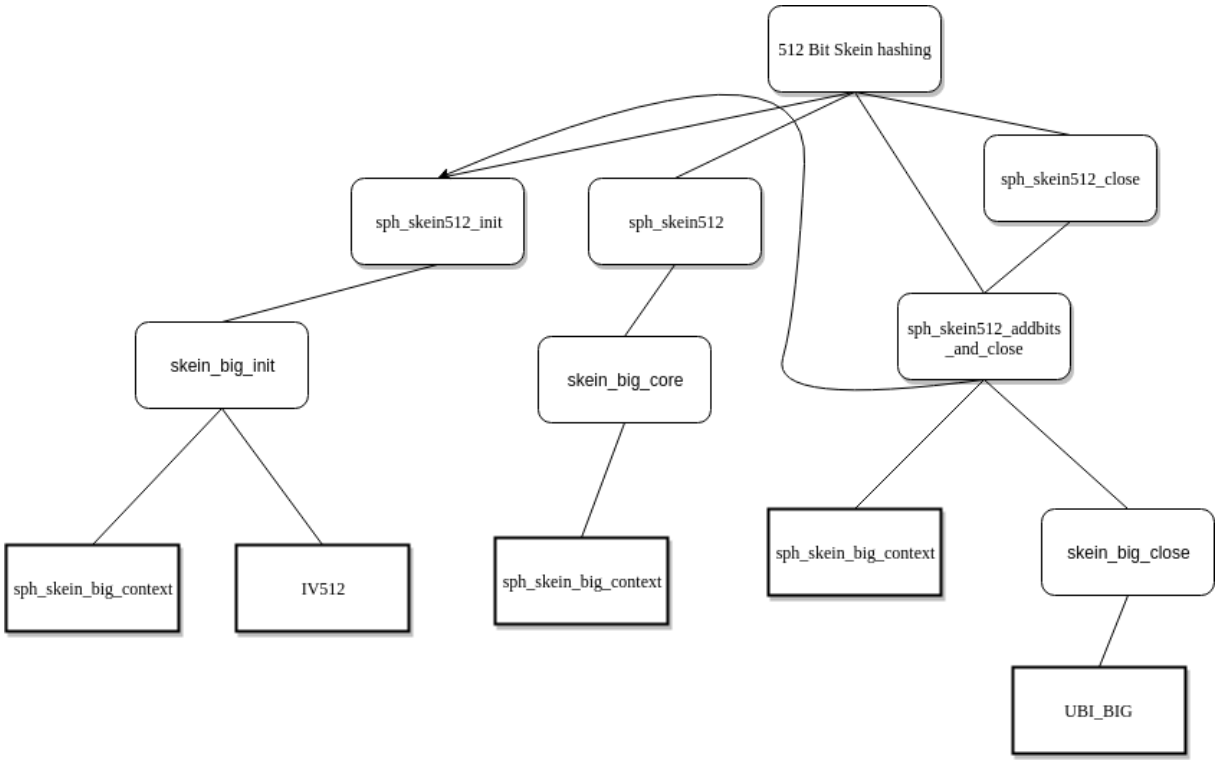
توضیییییییح اسکیییییین

فصل ۲

مدل طلایی

۱.۲ 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.



۲.۲ 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.

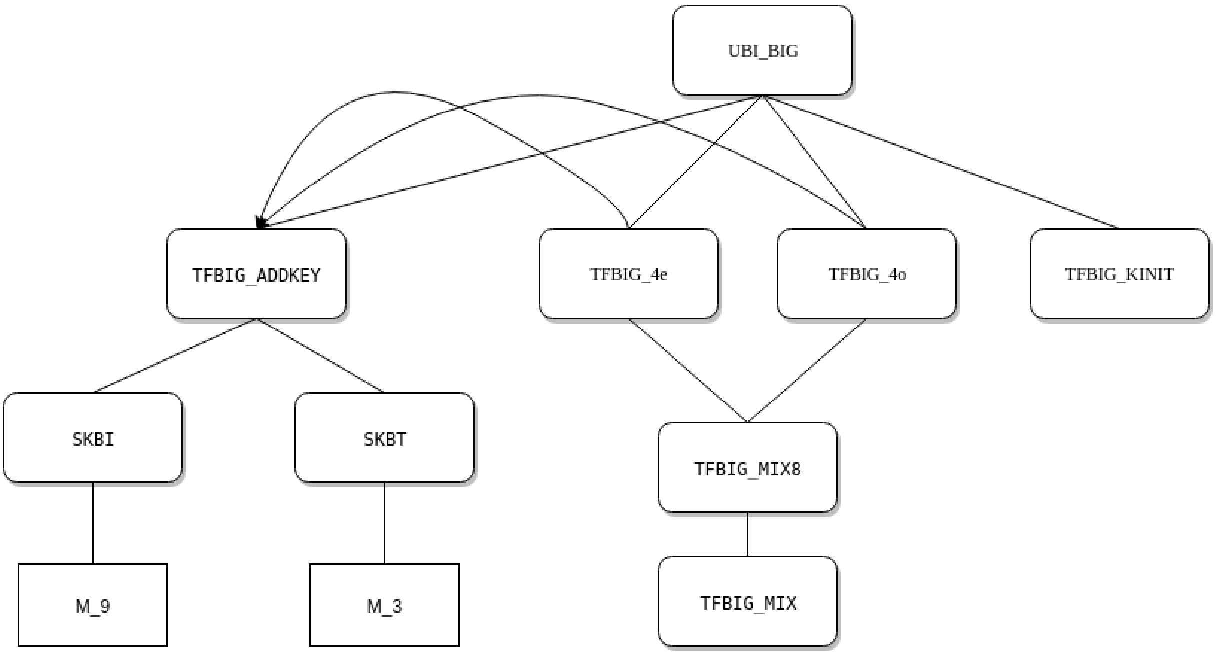
در این برنامه برای ذخیره و استفاده از هش ، از ساختاری استفاده شده است به نام sph-skein-big-context استفاده شده است و هدف برنامه اجرای الگوریتم هش و ذخیره‌ی خروجی در این ساختار است.

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 براااا یو بی آی بیگ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

فصل ۳

توابع و ساختارها

۱.۳ ساختارها

۱.۱.۳ 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.

۲.۱.۳ 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.

۳.۱.۳ 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_0, t_1 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.

۴.۱.۳ TFBIG-4e و TFBIG-4o

این تابع برای کدگذاری P_0 تا P_7 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_0 تا p_7 و h و t و s به ترتیب به عنوان w_0 تا w_7 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_0 تا p_7 اعداد

متفاوت به عنوان rc صدا شده‌است. ترتیب صدا شدن p_0 تا p_7 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 ۵.۱.۳

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.

SKBI و SKBT ۶.۱.۳

تابع SKBI برای محاسبه‌ی اندیس کلید استفاده شده‌است.
در الگوریتم برای تولید k_0 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 ۷.۱.۳

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 ۸.۱.۳

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

$$y_0 = (x_0 + x_1) \mod 2^{64}$$
$$y_1 = (x_1 <<< R_{(d \mod 8),j}) \oplus y_0$$

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 ۹.۱.۳

این تابع با ورودی‌های $t_۰$ تا $t_۲$ و $h_۰$ تا $h_۸$ مقادیر زیر را محاسبه می‌کند :

$$k_۸ = C \oplus k_۰ \oplus k_۱ \oplus ... \oplus k_۷$$

$$t_۲ = t_۱ \oplus t_۰$$

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

۲.۳ توابع

sph-skein512-init ۱.۲.۳

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 ۲.۲.۳

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 ۳.۲.۳

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