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

۱۰ 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

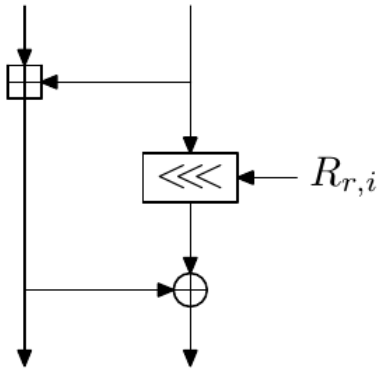
هدف الگوریتم درهم‌سازی skein مانند دیگر الگوریتم‌های درهم‌سازی امنیتی ایجاد یک تابع برای درهم‌سازی داده‌های مختلف است به شکلی که ویژگی‌ها زیر برای آنان برقرار باشند.

- **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

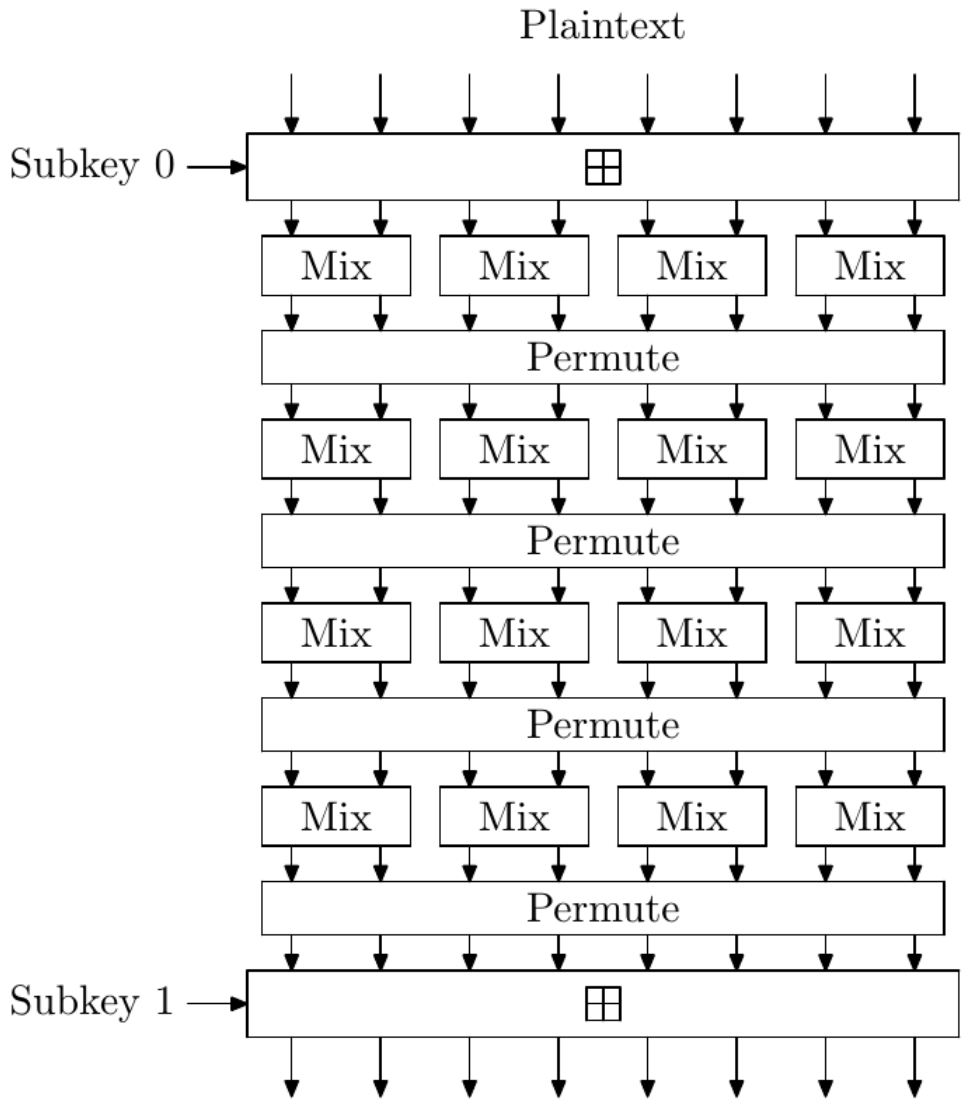
Threefish یک بلوک رمزگذاری قابل تنظیم است که برای سه سایز بلوک مختلف تعریف شده است، ۲۵۶، ۵۱۲ و ۱۰۲۴ بیت. اصل اساسی در طراحی Threefish توجه به این مورد است که تعداد زیادی از مراحل ساده امن‌تر از تعداد کمی مراحل پیچیده است. Threefish فقط از سه عمل‌گر اصلی XOR ، جمع کردن و دوران به اندازه یک عدد ثابت [۲] 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]

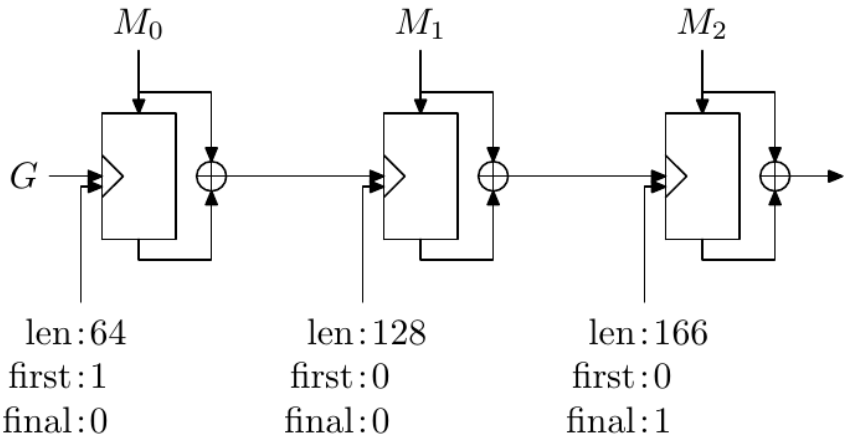
۲.۴.۱ Unique Block Iteration

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

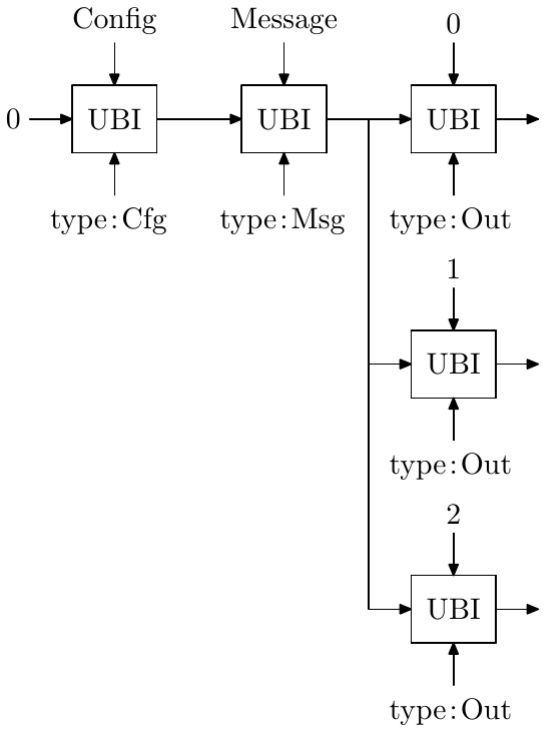
تابع اصلی درهم‌سازی در حالت نرمال که مد نظر این نوشتار است برای ایجاد هش از چندین درخواست از UBI و باالتبع از Threefish block cipher هش یک داده ورودی را حساب می‌کند، برای محاسبه هش سه بار UBI با ورودی‌های مختلف زیر صدا زده می‌شود، شکل ۴.۱ توضیحات زیر را به صورت شماتیک نشان می‌دهد.

- **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

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

۱.۲ 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 های‌Testbench مختلف
- گرفتن Waveform و مقادیر خروجی (اصلی و میانی)

۲.۲ مشاهدهٔ ورودی‌ها و خروجی‌های اصلی و میانی

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

۱.۲.۲ توضیح نحوهٔ عملکرد Testbench

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

Testbench 1

```
//First Testbench
module top;
    reg clk = 1'b0;
    reg [511:0] midstate = 72;
    reg [95:0] data = "hello" ;
    reg [31:0] nonce = 13;
    wire [511:0] hash;
    always #1 clk = !clk;

    skein512 skein(clk, midstate , data ,nonce , hash);

    initial
  begin
        #300 data = "how are you?";
        #300 data = "bye";
        #5000 $stop;
  end
endmodule
```

در این testbench ابتدا مقادیر ورودی ها ست میشود به ترتیب به ازای clock و midstate و data و nonce مقادیر ۰ و ۷۲ و hello و ۱۳ ست میشوند پس از ۳۰۰ واحد زمانی مقدارdata تغییرمی کند و به you are how 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

```
//Second Testbench
module top;
    reg clk = 1'b0;
    reg [511:0] midstate = 72;
    reg [95:0] data = "hello" ;
    reg [31:0] nonce = 23;
    wire [511:0] hash;
    always #1 clk = !clk;

    skein512 skein(clk, midstate , data ,nonce , hash);

    initial
  begin
        #300 data = "how are you?";
        #300 data = "bye";
        #5000 $stop;
  end
endmodule
```

در این testbench نیز ابتدا مقادیر ورودی ها ست میشود به ترتیب به ازای clock و midstate و data و nonce مقادیر ۰ و ۷۲ و hello و ۲۳ست میشوند پس از ۳۰۰ واحد زمانی مقدارdata تغییرمی کند و به you are how 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

```
//Third Testbench
module top;
    reg clk = 1'b0;
    reg [511:0] midstate = 72;
    reg [95:0] data = "still awake" ;
    reg [31:0] nonce = 23;
    wire [511:0] hash;
    always #1 clk = !clk;

    skein512 skein(clk, midstate , data ,nonce , hash);

    initial
  begin
        #300 data = "working";
        #6000 $stop;
  end

endmodule
```

در این testbench ابتدا مقادیر ورودی ها ست میشود به ترتیب به ازای clock و midstate و data و nonce مقادیر ۰ و ۷۲ و awake still و ۲۳ ست میشوند پس از ۳۰۰ واحد زمانی مقدارdata تغییرمی کند وبه working تبدیل میشود. ترتیب و مقدار hash در بخش مربوط به آن آمده است. در این بخش با تغییر مقادیر اولیه و ثانویه data خروجی ها را با قسمت قبل مقایسه کردیم.