گزارشی از مقاله

CARE[1]: Lightweight Attack Resilient Secure Boot Architecture with Onboard Recovery for RISC-V based SOC

CARE:یک معماری بوت امن سبک مقاوم در برابر حمله با بازیابی یکپارچه برایSOCمبتنی بر RISC-V

دیو آوانی

ارائه از : **مسعود باقری**

شماره دانشجویی: **۳۹۹۱۲۳۴۱۰۵۷۰۴۱**

ترم تحصیلی: **۲-۹۹**

درس: **معماری پیشرفته**

استاد: **دکتر جاسبی**

[1] Code Authentication and Resilience Engine

❖ **تعریف مسأله و هدف اصلی مقاله:**

سوال اصلی مطرح شده در مقاله:

با توجه به رشد سرسام آور اینترنت اشیا (IoT[۲]) و استفاده گسترده از سیستم‌های کامپیوتر توکار و سیستم‌های روی تراشه، بحث قابلیت اتکا برای این اشیاء هوشمند تبدیل به یک چالش اصلی و همیشگی شده. اتکا پذیری ویژگی یک سیستم کامپیوتری است که جنبه های مختلفی از جمله: ایمنی، امنیت، محرمانگی، دسترس‌پذیری، قابلیت اطمینان و نگهداری را شامل می شود. تحمل‌پذیری خطا مکانیزمی است که یک سیستم را قابل اتکا می‌نماید. اگر قبل از بالا آمدن یا روشن شدن این سیستم‌ها، خطایی رخ داده باشد و یا تهاجم و تخریبی صورت گرفته باشد آیا می توان این اشیا را آنقدر هوشمند ساخت که قابلیت خودبازیابی بدون دخالت انسان را داشته باشند؟

چه مشکلی باید برطرف شود؟

پیشرفت های اخیر فن آوری، باعث افزایش فاجعه بار استفاده از دستگاه های کوچک تعبیه شده و IoT ، در کاربردهای مختلفی از سیستم های کنترل صنعتی، سنجش و تحریک توزیع شده، سیستم های اتوماسیون خانگی و وسایل نقلیه، شده است. این افزایش استفاده و ارتباط متقابل [این دستگاهها] برای جمع آوری، انتقال و پردازش اطلاعات مهم امنیتی، دستگاه های کوچک تعبیه شده و اینترنت اشیا را به اهدافی جذاب برای حملات هکرها تبدیل کرده است. از نمونه های برجسته این حملات می توان به روتکیت[۳] [۱]، حملات بوت امن و بایوس[۴] [۲]، استاکس نت[۵] [۳] و هک جیپ[۶] [۴] اشاره کرد. برای توضیح بیشتر چنین حملاتی، لازم است بدانیم که مهاجم، وضعیت نرم افزار دستگاه مورد نظر را بمنظور نشت اطلاعات مهم امنیتی، سرقت، دستکاری یا سواستفاده از آن‌ها، برای فعالیت‌های مخرب تغییر می‌دهد. چنین حملاتی می‌تواند دستگاه را به حالت غیرقابل استفاده درآورد. این نوع حملات معمولاً به عنوان حملات تغییر کد مخرب یا آلوده شدن به بدافزار شناخته می‌شوند. دستگاه قربانی معمولا به سبب از کار افتادن ابزارهای تعمیر و بازیابی از راه دور، می‌بایست از شبکه جدا شده و به صورت دستی فلش شود، که وقت انرژی و خسارت زیادی ممکن است به دنبال داشته باشد. **مقاله حاضر روشی برای تشخیص، پاکسازی و بازنویسی کدهای آلوده قبل از بالا آمدن سیستم ارائه داده که تضمین می کند سیستم با کد غیر مخرب بالا خواهد آمد.**

[2] Internet of Things
[3] Rootkit
[4] the bios and secure boot attacks
[5] the Stuxnet
[6] Jeep hack

چه ضرورتی برای مطرح شدن مسئله است؟

فرآیند بوت یا راه اندازی امن، یکپارچگی و بی عیبی و اصالت حالت نرم افزاری دستگاه ها را در زمان راه اندازی تأیید می کند و از بوت شدن دستگاه با کد کاملا شناخته شده اطمینان می‌یابد. چندین روش راه اندازی امن بر اساس سخت افزار[۵ ، ۶]، نرم افزار [۷ ، ۸] و طراحی مشترک سخت افزار/ نرم افزار[۹–۱۲] ارائه شده است. در حالی که تکنیک های قبلی بوت امن بر روی شناسایی حملات مخرب تغییر کد تمرکز دارند، اما مسئله پاکسازی دستگاه های آسیب دیده کاملاً نادیده گرفته شده است. یک دستگاه آسیب دیده، برای بازیابی حالت عملیاتی خود به فلش مجدد کد به صورت دستی یا وایرلس (از راه دور) نیاز دارد. یک مهاجم هوشمند می‌تواند با خراب کردن پشته شبکه، از فلش مجدد از راه دور کد جلوگیری کند. این کار نیاز به مداخله دستی دارد. بعضی اوقات به دلیل قرارگیری دستگاهها (در سنسورها و دوربین های امنیتی منزل، سیستم های کنترل صنعتی و خودرویی، کشتی ها) ، فلش مجدد دستی نسبتاً دشوار می شود.

این مقاله، برای برطرف کردن این شکاف، CARE را که اولین چارچوب بوت ایمن سبک است ارائه می دهد. CARE امکان شناسایی کد غیر مجاز، مقاومت در برابر تخریب و مکانیسم بازیابی سیستم درحال کار (روی بورد[۷]) را برای دستگاه های کوچک تعبیه[۸] شده و اینترنت اشیا فراهم می کند.

چه روشهایی قبلا برای این کار انجام شده؟

آرباغ و همکاران، اولین مکانیزم راه اندازی ایمن [۱۳] را پیشنهاد کرده است که یکپارچگی سیستم را با تأیید یکپارچگی کد نرم افزار بوت یا راه انداز، به صورت چند مرحله‌ای اندازه گیری می‌کند. این مکانیزم یک بوت اندازه گیری شده را انجام می‌دهد که در آن، هر مرحله قبل از اجرا شدن ، یکپارچگی مرحله قبل خود را بررسی و تأیید می‌کند. بوت تأیید شده یا مجوز داده شده، تأیید می کند که نرم افزاری که روی سیستم اجرا می شود از طرف یک فروشنده مجاز ارائه می شود.

رابط فریمور قابل توسعه یکپارچه (UEFI[۹]) از نسخه ۲,۲ بوت امن را به عنوان فرایندی برای بررسی یکپارچگی و اصالت هر مرحله از فرآیند بوت با محاسبه دایجست (digest) و مقایسه نتیجه با یک امضای رمزنگاری، تعریف می کند. [۶] برای تأیید امضا نیاز به دسترسی به یک پایگاه داده معتبر با کلید عمومی

[7] onboard
[8] Embedded system
[9] Unified Extensible Firmware Interface

است. اکثر اجراهای قبلی سیستم‌های بوت امن یکی از دو روش بوت اندازه گیری شده، یا احراز هویت شده را انجام می‌دهند و تعداد کمی از آنها هر دو را انجام می‌دهند.

یکی از روش‌های محبوب برای بوت امن، استفاده از یک پردازنده مشترک مجزا به نام (TPM[10]) است [۹]. TPM، دارای ثبات‌های[11] ویژه و خاص منظوره‌ای به نام (PCR[12]) است که نمی تواند بازنویسی[13] یا جایگزین شود. PCR، فقط با هش کردن اندازه گیری‌های نرم افزار همراه با مقادیر قبلی PCR قابل توسعه یا تغییر است. TPM می‌تواند PCR ها را با یک کلید تأیید خصوصی امضا کند تا یک قطعه گواهی تأیید، تولید کند. با این حال، TPM به دلیل محدودیت‌های فضا، اندازه و هزینه، برای دستگاه‌های کوچک جاسازی شده یا اینترنت اشیا مناسب نیست.

پردازنده اینتل، از دو حالت بوت امن اندازه گیری شده و تأیید شده پشتیبانی می‌کند و از میکرو کد به عنوان ریشه اعتماد (RoT[14]) استفاده می کند [۱۴]. برای بوت اندازه گیری شده ، از TPM استفاده می کند و برای بوت تأیید شده، هر جزء، توسط کلید سازنده امضا می‌شود و قبل از بارگذاری آن جزء، امضاها تأیید می‌شوند.

Microsoft's fTPM [۱۰] یک استفاده موردی و موقعیت خاص از منطقه امن Arm، مبتنی بر بوت و تصدیق امن را فراهم می کند.

RISC-V مبتنی بر Sanctum [۷] از بوت امن و تصدیق از راه دور مبتنی بر نرم افزار استفاده می‌کند.

SMART [۸] ، معماری RoT دینامیکی را برای دستگاههای سطح پایین فراهم می‌کند.

کی استون (Keystone) [۱۲] یک استفاده موردی از محیط اجرایی قابل اعتماد (TEE[15]) را با انکلاوها [حافظه های محصور یا پنهان شده] به نمایش می گذارد.

حاج و همکاران [۵] معماری بوت امن مبتنی بر سخت افزار را برای SoC مبتنی بر RISC-V ارائه می دهد.

پروژه متن باز ریشه اعتماد (RoT) اخیر گوگل [۱۱]، نمونه ای از پیاده سازی RoT امن را ارائه می دهد.

با این حال، هیچ یک از راه حل‌های موجود مکانیسم بازیابی امن ندارند. پیاده سازی‌های اخیر هیلاد [۱۵] و [۱۶]، مکانیسم‌های بازیابی را نشان می‌دهد. با این حال، هر دو از اجرای مناسب بوت امن برخوردار نیستند.

[10] Trusted Platform Module
[11] registers
[12] Platform Configuration Records
[13] overwritten
[14] Root of Trust
[15] Trusted Execution Environment

نویسنده ادعا دارد که، کار پیشنهادی، اولین اجرای یک معماری بوت امن سبک با موتور انعطاف پذیر یکپارچه و بازیابی توکار[16] برای دستگاه‌های کوچک تعبیه شده و اینترنت اشیا است.

روش پیشنهادی ارائه شده چیست؟

ماژول ترکیبی CARE، دارای دو جز اصلی، یعنی واحد احراز هویت یا اعتبارسنجی کد (CA) و موتور انعطاف پذیری (RE)، است. این چارچوب با استفاده از زیر ماژول CA، یکپارچگی و اصالت flash image را اندازه گیری می‌کند. با شناسایی وجود کد مخرب، زیر ماژول RE را به کار می‌اندازد، در غیر این صورت روند راه‌اندازی بعدی را ادامه می‌دهد. کار RE فقط شناسایی مناطق خراب شده حافظه فلش و فلش مجدد انحصاری آن مناطق با کد خوب شناخته شده از ROM امن (پشتیبان) می‌باشد. این چارچوب با استفاده از سیاست‌های کنترل دسترسی با استفاده از مکانیزم محافظت از حافظه فیزیکی (PMP) پردازنده RISC-V، درصورت تشخیص خطا، اجرای کد و نوشتن غیر مجاز را از RAM غیرفعال می‌کند. این چارچوب، دوباره بررسی یکپارچگی و اصالت سخت‌افزار و صحت نرم‌افزار را انجام می‌دهد و روند راه‌اندازی امن را تا جایی که کلیه بلوک‌های کد، صحت سنجی و بارگذاری شوند، ادامه می‌دهد. این روش اطمینان حاصل می‌کند که صرف نظر از هرگونه حمله مخرب و تغییر کد، دستگاه خود بازیابی شده و همیشه با یک کد خوب شناخته شده راه‌اندازی می‌شود.

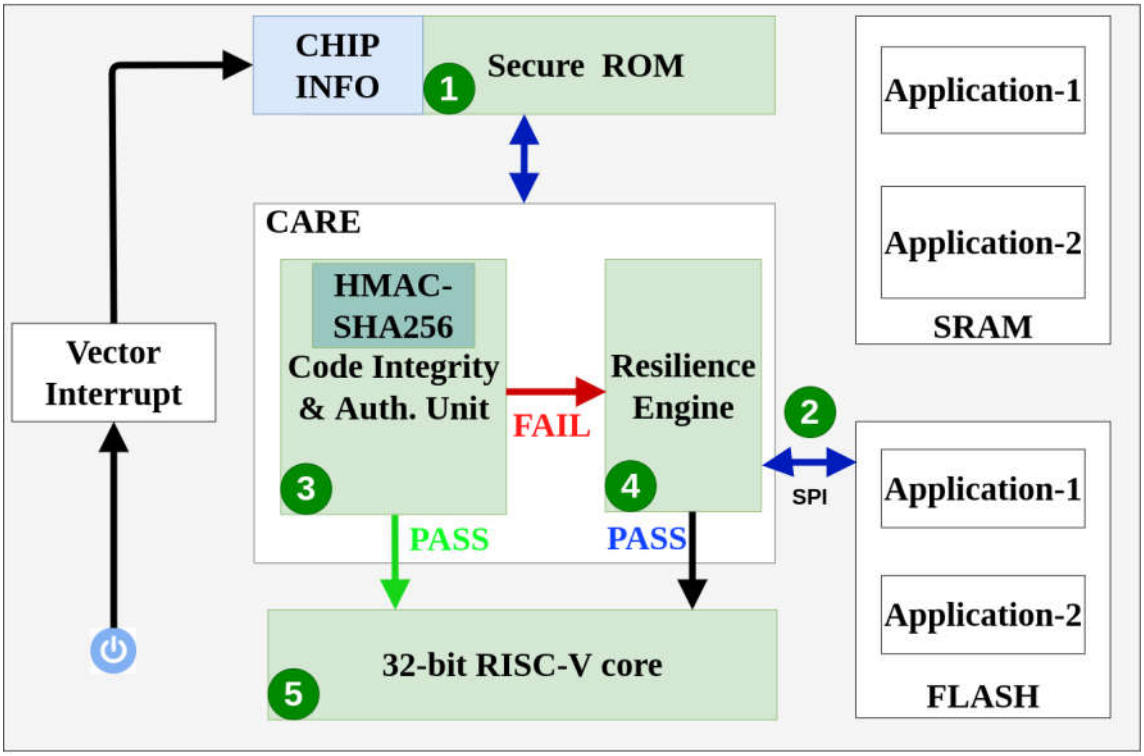
❖ **.توضیح راه حل پیشنهادی مقاله برای حل مسئله:**

روش پیاده سازی شده برای حل مسئله مقاله به چه صورت است؟

طراحی معماری چارچوب پیشنهادی در شکل فوق نشان داده شده است و عملکرد سیستم به سه مرحله اصلی تقسیم می شود: (۱) مقدار دهی اولیه سیستم؛ (۲) بررسی یکپارچگی و اصالت کد؛ و (۳) موتور انعطاف پذیری (RE).

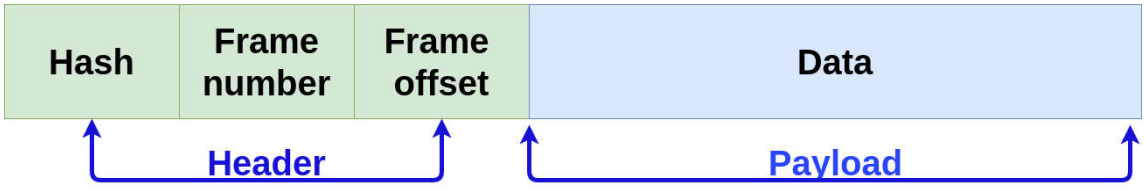
[16] onboard

۱. مقدار دهی اولیه سیستم: با روشن شدن پاور، سیستم، کد FSBL[۱۷] را از ROM امن برای یافتن SPI[۱۸] و کنترل کننده‌های فلش پیدا و اجرا می‌کند. سپس قوانین محافظت از حافظه فیزیکی را با استفاده از PMP[۱۹] اعمال می‌کند و خواندن، نوشتن و اجرای کد غیر مجاز را که از حافظه فلش محافظت نشده آغاز می‌شود، مسدود می‌نماید. (PMP یکی از ویژگی‌های استاندارد معماری RISC-V است که به فریمور اجازه می‌دهد قسمتی از حافظه فیزیکی را مشخص نموده و برای آن حقوق دسترسی خاصی تعیین کند.) اطلاعاتی از تراشه مانند – UUID[۲۰] دستگاه، نسخه برد[۲۱] و کلید اشتراک متقارن را می‌خواند ، کلیدهای مشتق شده تولید می‌کند و کنترل را به مرحله دوم بوت یا راه اندازی موسوم به بوت استرپ می‌دهد.



[17] First Stage Boot Loader
[18] Serial Peripheral Interface
[19] Physical memory protection
[20] Universally unique identifier
[21] board version

۲. سیستم Bootstrapping: بوت امن با فرآیند بوت استرپ شروع می‌شود و هربار که سیستم روشن شود یا بازنشانی شود، چه سخت افزاری و با فشردن کلید پاور باشد، چه نرم افزاری و با دستور یک میزبان خارجی از طریق پین هفتِ GPIO[۲۲] اتفاق می‌افتد. هنگامی که بوت استرپ فعال می‌شود، تصویر فلش اجرایی به قطعات قاب داده ۱ کیلوبایتی تقسیم می‌شود و از طریق گذرگاه SPI به ترتیب به میزبان ارسال می‌شود. همانطور که در شکل زیر نشان داده شده است، هر قاب داده از یک هدر و پی لود یا بسته اطلاعاتی مربوطه تشکیل شده است. هدر حاوی خلاصه امضا شده قاب داده‌ها است. قسمت مکان افست، محل حافظه فلش را نشان می‌دهد، که برای فلش مجدد کد استفاده می‌شود. پی لود یا بسته اطلاعاتی اصلی شامل ۹۶۸ بایت داده برای هر قاب است.



چارچوب پیشنهادی از ویژگی HMAC-SHA256 مربوط به کد اصالت‌سنجی پیام بر پایه درهم‌سازی HMAC[۲۳] برای تأیید داده‌ها و انجام تأیید امضا و از الگوریتم هش امن SHA256 برای محاسبه خلاصه هر قاب استفاده کرده است. چند API برای استفاده مجدد از همان سخت افزار رمزنگاری اساسی HMAC-SHA256 ساخته شده است. ماژول HW-SHA256 خلاصه هر قاب داده KB۱ را محاسبه می‌کند و آن را با هش از قبل محاسبه شده برای بررسی یکپارچگی یا بی عیب بودن مقایسه می‌کند. HMACSHA256، از یک کلید مشتق شده برای امضای خلاصه محاسبه شده استفاده می‌کند، و آن را با قسمت "Hash" در هدر قاب داده برای بررسی اصالت، مطابقت می‌دهد. این فریم ورک یا چارچوب نرم افزاری، برای هر قاب داده بعدی از همان فرایند بوت استرپینگ پیروی می‌کند. اگر همه چیز درست بود، دستگاه با کد تایید شده بوت می‌شود، در غیر این صورت موتور انعطاف پذیری را تحریک یا راه اندازی می‌کند.

۳. موتور انعطاف پذیری (RE): موتور انعطاف پذیری به این شرح عمل می کند:

۳,۱. شماره قاب و محل جابجایی قاب خراب را مشخص می‌کند و داده‌های قاب طلایی مربوطه را از EEPROM امن واکشی می‌کند.

[22] General Purpose Input Output

[23] Hashed based Message Authentication Code

۳,۲. منطقه منحصر به فرد حافظه فلش خراب را با کد خوب شناخته شده، مجدداً فلش می‌کند.

۳,۳. دسترسی خواندن و نوشتن غیر مجاز به حافظه را با استفاده از سازوکار PMP قفل می‌کند. با این مراحل اطمینان حاصل می‌شود که دستگاه پیشنهادی بدون توجه به حملات تغییر حافظه، همیشه با کد خوب شناخته شده راه‌اندازی می‌شود.

روش ارائه شده به چه صورت پیاده سازی شده؟(نرم افزاری یا بصورت اثبات ریاضی دقیقا توضیح داده شود)

پیاده‌سازی بر روی fpga بوده و با اندازه‌گیری مولفه‌های مختلف المان‌های به کار رفته ازجمله مصرف انرژی و زمان تأخیر انتشار مربوطه، عملکرد هسته پردازشی تعیین شده است که همزمان با مقایسه با سایر ایده‌های مطرح بررسی می‌شود.

نحوه مقایسه ایده مطرح شده با دیگر ایده‌های مطرح شده در مقاله:

۱. مقایسه کارایی در هسته مبتنی بر نرم‌افزار و سخت‌افزار: هسته رمزنگاری HMAC-SHA256، جز اصلی واحد یکپارچگی و احراز هویت کد CA[۲۴] است. همانطور که در جدول زیر نشان داده شده است، تنظیمات آزمایش، ابتدا از اجرای هم سخت افزاری و هم نرم افزاری هسته رمزنگاری که بر روی FPGA برای ارزیابی عملکرد اجرا می‌شود، استفاده می‌کند. این سیستم، خلاصه ۲۵۶ بایت داده را برای ارزیابی عملکرد و کارایی انرژی محاسبه می‌کند. داده‌های جدول، افزایش کارایی ۱۶ برابر با ۹۲ درصد مصرف انرژی کمتر با استفاده از هسته رمزنگاری مبتنی بر سخت افزار را نشان می دهد.

Parameters	Software	Hardware
Cycles (c)	**47033**	**2926**
Frequency (F) (MHz)	100	100
Block (b)	256	256
Throughput (T) (Mbps)	.54	8.74
Time (μsec)	470.33	29.26
Energy Consumption (E)	197.06	12.25
Energy Efficiency	**92.68**	**0.358**

[24] Code Authentication

۲. مقایسه هسته رمزنگاری پیشنهادی با هسته‌های مشابه موجود: هسته رمزنگاری پیشنهادی، سبک است و انرژی کمتری نسبت به اجراهای HMAC-SHA256، در کارهای اخیر مصرف می‌کند که در جدول زیر نشان داده شده است. همانطور که مشاهده می‌شود، [۲۱] به منطقه نسبتاً کمی نیاز دارد اما انرژی زیادی مصرف می‌کند. [۲۲] هسته و رمزنگاری انعطاف پذیر پایه و DPA را ارائه می‌دهد اما از منطقه و توان بیشتری استفاده می‌کند.

[۲۰]، هسته را برای توان بالا بهینه می‌کند ولی منطقه و مصرف انرژی را بالا می‌برد، که هردوی این موارد باعث می‌شود برای دستگاه‌های کوچک تعبیه شده و IoT مناسب نباشند. هسته رمزنگاری مورد استفاده در کار پیشنهادی، نسخه‌ای بهینه شده برای انرژی و مساحت از opentitan است [۱۱].

Work	*Area*	Freq.	Energy Con.	Device
This work	2591	100	.012	Artix-7
Opentitan [11]	2693	100	.022	Artix-7
He et al.[20]	7219	116.24	1.20	Arria II GX
He et al.[20]	10918	87	1.80	Arria II GX
Juliato et al. [21]	2347	138.10	.48	Stratix III
Juliato et al. [22]	4281	67	.285	CycloneII
Juliato et al. [22]	6874	41.25	.431	CycloneII

۳. مقایسه زمان و انرژی مصرفی با و بدون زیر ماژول موتور انعطاف پذیری: RE[۲۵] در نرم افزار برای اثبات مفهوم POC[۲۶] پیاده سازی شده است. RE به ۶۱ خط کد اضافی (زبان C) برای راه اندازی امن نیاز دارد و فضای ROM امن را برای ذخیره اطلاعات بازیابی، ۵ کیلوبایت افزایش می‌دهد. (یعنی کد، زمان و منابع بیشتری استفاده خواهد شد.)

این سیستم، برنامه آزمایشی ۵,۶ کیلوبایتی را به شش قاب داده ۱ کیلوبایتی تقسیم کرده و بررسی‌های صحت یا یکپارچگی و اعتبار یا اصالت را برای ارزیابی عملکرد انجام می‌دهد. کل زمان بوت و مصرف انرژی با و بدون بوت امن مبتنی بر CARE برای برنامه آزمایشی اجرا شده روی FPGA محاسبه شده است. جزئیات تجزیه و تحلیل زمان بندی آن در جدول زیر نشان داده شده است.

قاب داده اول به چرخه‌ها و زمان بیشتری نیاز دارد. بقیه قاب داده‌ها، تعداد مساوی چرخه را مصرف می‌کنند. بوت امن با CARE، 8 درصد انرژی بیشتری مصرف می‌کند و به D_{Δ} = ۵۲۹ میکرو ثانیه زمان بیشتر برای بوت نیاز دارد. ماژول فرعی RE پیشنهادی به ۳۳۴,۴۷۵ میکروثانیه اضافی برای فلش مجدد ۹۶۸ بایت داده برای هر قاب آسیب دیده نیاز دارد. این سربار عملکرد در مقایسه با امنیت و

[25] Resilience Engine

[26] Proof-Of-Concept

انعطاف پذیری که ایجاد می‌کند، ناچیز است. (تنها ۸ درصد افزونگی برای برنامه مورد آزمایش البته درصورت عدم نیاز به فلش مجدد)

Parameters	Without *CARE*	With *CARE*
Cycles req. for the first frame (c)	553611	576083
Cycles (rest of frames) (c)	103330	133790
Total Cycles (C)	**656941**	**709873**
Frequency (F) (MHz)	100	100
Time (t) (μsec)	**6569.41**	**7098.73**
Energy Consumption (E)	**2752.58**	**2974.36**
Time difference D_{Δ} = **529.32** μsec		

۴. مقایسه کمی با پیشرفته‌ترین یا آخرین راه‌حل‌ها: همانطور که در جدول زیر قابل رویت است، غیر از [۵] که صرفا مبتنی بر سخت افزار است، همه روش‌های دیگر از سیستم بوت امن ترکیبی استفاده می‌کنند. غیر از [۲۳] که از پردازنده مشترک مجزا TPM استفاده کرده باقی روش‌ها از بوت امن مبتنی برناحیه ROM امن استفاده کرده‌اند. اکثرا یکپارچگی و اصالت کد را بررسی می‌کنند اما هیچکدام مثل CARE قابلیت بازیابی ندارند و به ندرت قابل استفاده در سیستم‌های کوچک و توکار هستند.

Parameters	*CARE*	Optitan[11]	Haj[5]	Ref.[23]
Design Type	Hybrid	Hybrid	HW	Hybrid
Secure boot function	hmacSha2	hmacSha2	Sha3	Sha2
Rom for Secure boot	yes	yes	yes	no/TPM
Integrity Check	yes	yes	yes	yes
Authenticity Check	yes	no	yes	yes
Recovery	yes	no	no	no
Lightweight	yes	yes	no	no

نقاط قوت و ضعف مقاله:

توضیح مسأله و راه حل‌های موجود بسیار قوی بود و بطور گسترده‌ای تمام انواع تهدیدها و همچنین سبک‌های مختلف مقابله را دربرمی‌گرفت. جزئیات پیاده‌سازی سخت‌افزاری و نرم‌افزاری و در برخی موارد حتی کلیات آن نیز به هیچ وجه ارائه نشده. مثل کدهای زبان C ، FPGA و ...

جمع بندی و پیشنهادات برای کارهای آتی:

در این مقاله یک چارچوب بوت سبک و امن با مکانیسم بازیابی و محافظت برای دستگاه های کوچک تعبیه شده و اینترنت اشیا ارائه شد تا از آن در برابر حملات مخرب دستکاری کد محافظت کند. این چارچوب، ابزار شناسایی، بازیابی و پیشگیری از حمله تغییر کد را فراهم می کند که به کاربر اطمینان می دهد دستگاه همیشه با یک کد خوب شناخته شده راه اندازی می شود.

ولی با توجه به استفاده گسترده از بردهایی همچون رسپبری‌پای، آردینو، اورنج‌پای، بیگل‌بن، تینکربورد و ... و با تجه به ساختار و عملکرد این بوردها شاید قدم بعدی در توسعه موضوع مقاله روشی باشد که به دستگاه اجازه دهد از روی کارت SD یا درایو فلش USB بوت شود.

درضمن با توجه به محدودیت این روش در دریافت به‌روزرسانی، یکی دیگر از ایده‌ها می‌تواند اتخاذ ترتیبی برای حفظ امنیت دستگاه، درحین دریافت به‌روزرسانی باشد. به این ترتیب دامنه استفاده از این روش در دستگاه‌های بیشتری توسعه خواهد یافت.

شبیه سازی:

ارائه و شبیه سازی آنچه در مقاله شبیه سازی یا پیاده سازی شده است