Implementando o SMART, uma abordagem prática para verificação remota em dispositivos embarcados

Gabriel Capella[1]

[1]Instituto de Matemática e Estatística – Universidade de São Paulo (USP)
São Paulo – SP – Brasil

gabriel@capella.pro

Abstract. *Verificação remota é um método que serve para atestar à distância a integridade de um dispositivo. Este artigo apresenta os principais aspectos dessa técnica e mostra uma implementação conhecida como SMART, acrônimo para Secure and Minimal Architecture for (Establishing a Dynamic) Root of Trust. Essa implementação é voltada principalmente para dispositivos embarcados de baixo custo, utilizados frequentemente no cenário da Internet das Coisas. Muitos desses dispositivos estão sendo desenvolvidos e usados em cenários críticos, mas constantemente são descobertas novas vulnerabilidade neles. A verificação remota visa identificar se a memória desses dispositivos foi indevidamente modificada de maneira remota. Além de refazer os passos de implementação da técnica, este artigo propõe uma nova implementação para assegurar as mesmas garantias do SMART. Foi desenvolvido um protótipo em uma FPGA para testar se a implementação teórica sugerida funciona. Alguns ataques foram simulados no dispositivo e no meio de comunicação. Todos os ataques foram identificados com êxito. Também é feita uma análise do consumo de hardware e memória necessários para introduzir essa técnica de verificação remota.*

Resumo. *Verificação remota é um método que serve para atestar à distância a integridade de um dispositivo. Este artigo apresenta os principais aspectos dessa técnica e mostra uma implementação conhecida como SMART, acrônimo para Secure and Minimal Architecture for (Establishing a Dynamic) Root of Trust. Essa implementação é voltada principalmente para dispositivos embarcados de baixo custo, utilizados frequentemente no cenário da Internet das Coisas. Muitos desses dispositivos estão sendo desenvolvidos e usados em cenários críticos, mas constantemente são descobertas novas vulnerabilidade neles. A verificação remota visa identificar se a memória desses dispositivos foi indevidamente modificada de maneira remota. Além de refazer os passos de implementação da técnica, este artigo propõe uma nova implementação para assegurar as mesmas garantias do SMART. Foi desenvolvido um protótipo em uma FPGA para testar se a implementação teórica sugerida funciona. Alguns ataques foram simulados no dispositivo e no meio de comunicação. Todos os ataques foram identificados com êxito. Também é feita uma análise do consumo de hardware e memória necessários para introduzir essa técnica de verificação remota.*

1. **Introdução**
2. **Verificação Remota**
3. **SMART**
4. **Implementação Proposta**
5. **Resultados**

Introducão

Devido à nova indústria de Internet das Coisas (*Internet of Things, IoT*), há uma demanda crescente por novos dispositivos conectados à Internet. Eles são responsáveis principalmente por sensoriar informações e atuar no ambiente.

Apesar de serem dispositivos simples, eles têm sido alvos de diversos ataques [**?**]. Quando sofrem um ataque, é difícil perceber que o mesmo ocorreu, pois muitas vezes o dispositivo não altera o seu funcionamento. Existem ataques que visam infectar o dispositivo para posteriormente utilizá-lo para cometer atos ilícitos e outros que visam alterar o seu funcionamento ou comportamento.

Identificar esses ataques é uma tarefa importante. Existe uma técnica conhecida como verificação remota (*Remote Attestation*) que provê uma solução para esse problema. Essa técnica verifica o estado interno do dispositivo remotamente. Entende-se por estado a configuração da memória do dispositivo em um dado momento. Ela pode ser implementada de diversas maneiras.

Para computadores existem soluções bem estabelecidas. No entanto, para dispositivos embarcados há pouca pesquisa na área.

Objetivos

O objetivo principal deste trabalho é implementar uma solução de verificação remota em um dispositivo. Além disso, realizar ataques nesse protótipo e ver se eles são corretamente identificados.

A implementação de verificação remota escolhida foi a sugerida no artigo *SMART: Secure and Minimal Architecture for (Establishing a Dynamic) Root of Trust* [**?**]. Ela foi escolhida, pois é referência na área e é utilizada como base para diversos artigos posteriores. Além disso, ela tem como foco manter o baixo custo desses dispositivos.

No processo de verificação remota existem dois elementos principais, o contestado e o verificador [**?**]. O contestado é o dispositivo inseguro sobre o qual queremos realizar a verificação remota. Já o verificador seria, por exemplo, o servidor que enviará desafios para o contestado com o objetivo de validar seu estado.

Na implementação sugerida pelo SMART, o processo de verificação inicia-se com o verificador criando uma sequência de bytes aleatórias, chamado de desafio. Esse desafio junto com uma descrição de uma espaço de memória do dispositivo é enviado para o contestado. Ele deve pegar esses valores e calcular um *hash* deles concatenado com o conteúdo da região de memória descrita na solicitação. Esse valor calculado é enviado novamente para o verificador. O verificador sabe o estado (conteúdo da memória) que

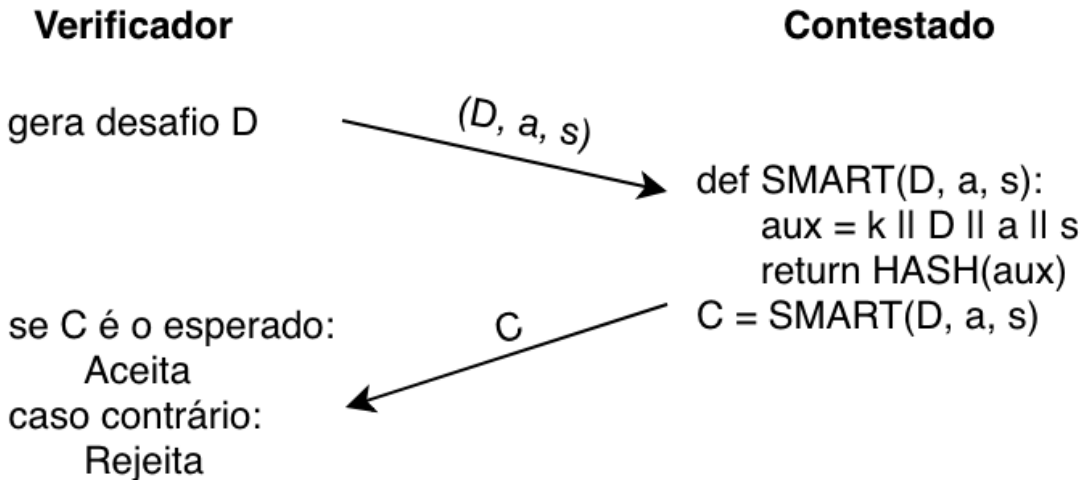

Figura 1. Esquema do das informações trocadas durante o processo de verificação remota. Para simplificar o diagrama foi utilizado D para simbolizar o desafio, a para a posição inicial da memória, s para o números de bytes da memória para serem considerados a partir de a, k para a chave e C para o resultado da computação.

o dispositivo deve possuir, sendo assim é capaz de verificar se o valor calculado pelo contestado é correto.

Durante o artigo algumas suposições são consideradas. A primeira é que o dispositivo (contestado) não sofrerá nenhum ataque físico. A segunda é que devemos considerar que o seu software é totalmente violável. Ou seja, pode conter erros que possibilitem que um atacante mude ele por completo.

Para conseguir prover a verificação remota levando em consideração essas suposições é necessário que o dispositivo contenha uma função de *hash* inviolável - que não possa ser reprogramada e que quando chamada não possa ser desviada e nem interrompida. Além disso, essa função deve conter uma chave, a qual somente ela tem acesso. O principal objetivo dessa função e dessa chave é impedir que uma solicitação realizada pelo verificador seja forjada. Ou seja, o dispositivo não deve ser capaz de computar uma resposta que seja idêntica à esperada se não possuir as informações corretas.

Com esses elementos, o cálculo da função inviolável, chamado de código do SMART, será o *hash* do valor concatenada da chave, desafio, posição inicial da memória e número de bytes da memória a serem considerados. Como a função é inviolável, o atacante somente consegue mudar a entrada dela, o que resultaria em um valor diferente do esperado pelo verificador. Note que nessa disposição o verificador deve possuir a chave da função de *hash* para computar o resultado esperado. A figura 1 exemplifica esse processo.

Implementação

Para implementar um dispositivo com todas essas características, foi escolhido utilizar como base o microcontrolador openMSP430 [**?**], o qual foi contruído em um FPGA Xilinx Spartan 6 XC6SLX9. Além disso, o dispositivo foi conectado a um módulo ESP8266, tornando-o capaz de trocar informações com a Internet.

Para criar um código inviolável e uma chave só disponível para ele, foram criados

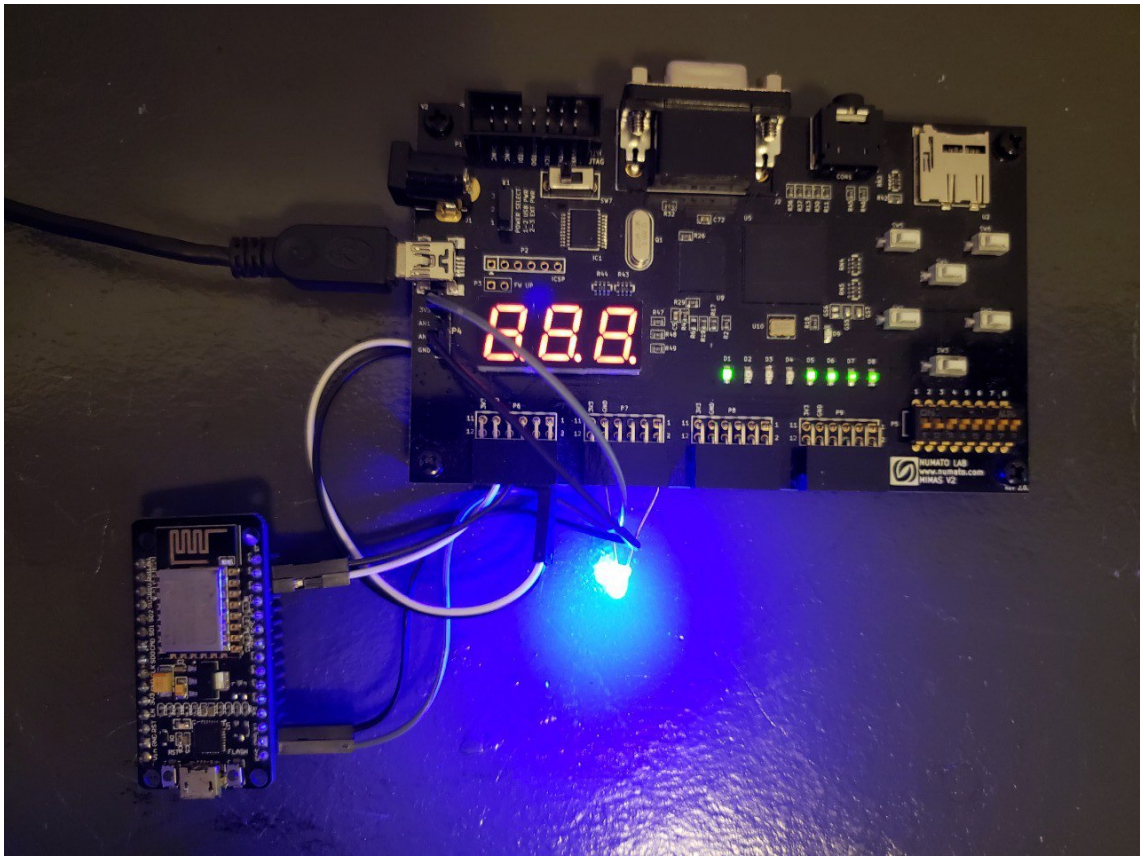

Figura 2. Disposição final do dispositivo do protótipo. Na parte superior a placa de prototipagem Numato Mimas V2, contendo o FPGA Spartan 6. Na parte inferior o módulo de acesso a internet ESP8266 na placa Amica Node MCU. A comunicação entre eles é serial, utilizando o protocolo de comunicação RS-232. Nessa foto o cabo USB está ligado a um carregador de parede e o dispositivo está com o status do seu LED sendo controlado remotamente.

dois módulos de acesso de controle de memória. Esses módulos foram escritos na linguagem de descrição de *hardware* Verilog e foram colocados no barramento de memória entre microcontrolador e sua memória. Ambos são iguais entre si, no entanto são configurados com parâmetros distintos. Na Figura 3 é possível a disposição final deles.

A ideia básica desse módulo é observar constantemente o registrador contendo o ponteiro de instruções do microcontrolador e, dependendo do seu valor, liberar o acesso à uma região específica da memória. Caso haja uma tentativa de acesso à essa memória sem o ponteiro de instrução passar por uma posição específica, o dispositivo é reiniciado.

O primeiro módulo é responsável por proteger o próprio código do SMART. Ele impede que o código seja chamado parcialmente. O segundo, visa proteger a chave. Ela só se torna acessível se o código do SMART for chamado.

Outra propriedade importante do módulo é que ele possui um estado interno que caracteriza se o acesso à memória está ativo. Esse estado se torna ativo somente se o ponteiro de instrução aponta para a primeira instrução de uma função específica. Ele se torna inativo se o ponteiro sai dessa função.

Para programar o microcontrolador, foi necessário reescrever os arquivos de mon-

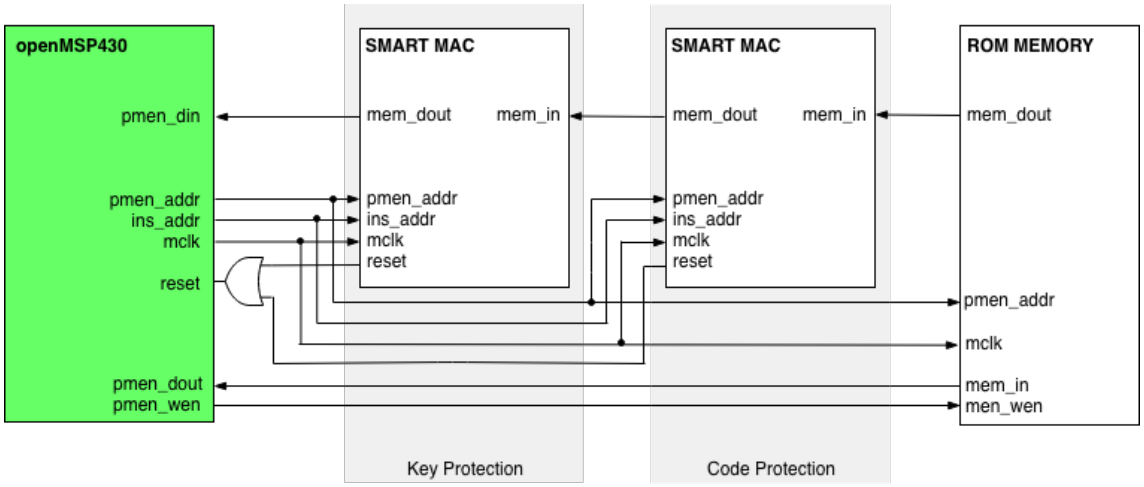

Figura 3. Interface entre a memória ROM e o microcontrolador openMSP430. No meio, estão os módulos de controle de acesso à memória.

tagem (*Linker Scripts*) utilizada pelo GCC. Esses arquivos são responsáveis por descrever onde cada função, variável ou informação do programa deve ficar na memória.

Outra diferença em relação ao artigo original foi a utilização da função *hash* SHA256 implementada *hardware*.

Resultados

	Número of Slice Registers	**Número of Slice LUTs**
SMART sem o *hardware* do SHA256	1025	2492
SMART com o *hardware* do SHA256	2702	4764
Microcontrolador sem os módulos com o *hardware* do SHA256	2694	4710

Tabela 1. Número de LUTs e registradores do FPGA dependendo da ativação dos módulos de controle de acessa a memória e do hadware do SHA256.

Foi possível implantar o verificador e o contestado e testá-los com sucesso. Além disso, foram simulados alguns tipos ataques ao dispositivo, os quais foram identificados com êxito. Na Figura 2 é possível ver como ficou o protótipo final.

Na Tabela 1 existe uma análise dos recursos consumidos pela implementação do SMART. Apesar da função de cálculo de *hash* parecer causar um acréscimo significativo no tamanho do dispositivo, ele não o causa, pois quando utilizamos uma implementação de uma função *hash* em *hardware* podemos criar um dispositivo com uma memória menor - não há necessidade de salvar o código que computa a função *hash*.

Conclusão

Implementar e testar programas escritos em linguagem de descrição de *hardware* é uma tarefa relativamente simples. No entanto executar esses programas em um FPGA de verdade pode resultar em diversos erros.

Appesar deles, através dos testes e experimentos realizados pode-se concluir que introduzir a funcionalidade de verificação remota em dispositivos embarcados é uma tarefa simples que não envolve profundas modificações no hardware dos mesmos. Além disso, as modificações causadas pelo o acréscimo da função de hash são significativas para o custo final deles, idependentemente se é implementada em memória ou em *hardware*.

6. Conclusão

All full papers and posters (short papers) submitted to some SBC conference, including any supporting documents, should be written in English or in Portuguese. The format paper should be A4 with single column, 3.5 cm for upper margin, 2.5 cm for bottom margin and 3.0 cm for lateral margins, without headers or footers. The main font must be Times, 12 point nominal size, with 6 points of space before each paragraph. Page numbers must be suppressed.

Full papers must respect the page limits defined by the conference. Conferences that publish just abstracts ask for **one**-page texts.

7. First Page

The first page must display the paper title, the name and address of the authors, the abstract in English and "resumo" in Portuguese ("resumos" are required only for papers written in Portuguese). The title must be centered over the whole page, in 16 point boldface font and with 12 points of space before itself. Author names must be centered in 12 point font, bold, all of them disposed in the same line, separated by commas and with 12 points of space after the title. Addresses must be centered in 12 point font, also with 12 points of space after the authors' names. E-mail addresses should be written using font Courier New, 10 point nominal size, with 6 points of space before and 6 points of space after.

The abstract and "resumo" (if is the case) must be in 12 point Times font, indented 0.8cm on both sides. The word **Abstract** and **Resumo**, should be written in boldface and must precede the text.

8. CD-ROMs and Printed Proceedings

In some conferences, the papers are published on CD-ROM while only the abstract is published in the printed Proceedings. In this case, authors are invited to prepare two final versions of the paper. One, complete, to be published on the CD and the other, containing only the first page, with abstract and "resumo" (for papers in Portuguese).

9. Sections and Paragraphs

The primary objective of this research is to implement and test the hardware and ideas proposed in SMART: Secure and Minimal Architecture for (Establishing a Dynamic) Root of Trust [6]. This article talks about the minimum hardware requirements for embedded

devices to make remote attestation viable. The basic idea in the article is to change the memory access to get the exclusive access to the key in a ROM memory (read-only memory). Changing it, they made the key only accessible when the device program counter points to the first address of the Attest. Other changes have also been made to prevent the Attest to be executed from its middle. The memory cleaning, to prevent the leak of the key, is implemented as part of the Attest function. The uninterruptedly is guaranteed by a software instruction to disable all interruptions at the beginning of the Attest function. The Attest function in the device is made using a software SHA-1 hash implementation. It concatenates the device memory, and the nonce sent by the challenger and obtain a hash that represents the device state. Also, the function can look only to one part of the memory, and this makes possible the attestation of single parts of the device in a relatively short time. Another functionality that has been added to the Attest function is the capability to the challenger to send a function address to be called at the end of the SMART code execution. This feature makes possible to attest one memory region and, afterward, call its code in a safety way. The article shows the benefits of this in two examples: the first is to attest the reading of measurements and the second is to prove if the device has been successfully reset. To test the functionality of the idea, they have implemented the SMART modifications in two different micro-controllers units: the Atmel AVR and the Texas Instruments MSP430. According to the article, adding SMART to both chips caused only a 10respective surface areas. That is, adding the SMART hardware do not increase the cost of the device significantly. All the results from the article are obtained from simulations. The authors did not mention that they have tested the SMART code in FPGAs or ASIC devices. Also, they affirm that more experiments using current implementations need to be performed for better overhead evaluation. The SMART implementation takes several assumptions. Among them is that the attacker has full control of the device program. It can change it or call other functions. Note that the program can be safely saved in read-only memory, but this does not prevent this type of attacks. Some attacks, such as a technique called Return-Oriented Programming (ROP) [18], only change the return pointer in the data memory and reuse the program code in a different memory alignment to perform the attack. Therefore, in the article, the authors assumed that the attacker has full access and can modify the code outside the protected memory region of SMART. Another critical assumption is that the device will not suffer any hardware attacks. This assumption is important because if it is possible, the attacker can, for example, reset the

9.1. Subsections

The subsection titles must be in boldface, 12pt, flush left.

10. Figures and Captions

Figure and table captions should be centered if less than one line (Figure 4), otherwise justified and indented by 0.8cm on both margins, as shown in Figure 5. The caption font must be Helvetica, 10 point, boldface, with 6 points of space before and after each caption.

In tables, try to avoid the use of colored or shaded backgrounds, and avoid thick, doubled, or unnecessary framing lines. When reporting empirical data, do not use more decimal digits than warranted by their precision and reproducibility. Table caption must be placed before the table (see Table 1) and the font used must also be Helvetica, 10 point, boldface, with 6 points of space before and after each caption.

Figura 4. A typical figure

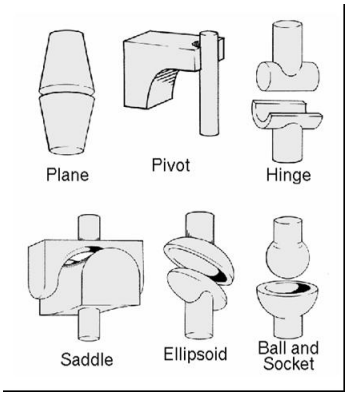

Figura 5. This figure is an example of a figure caption taking more than one line and justified considering margins mentioned in Section 10.

11. Images

All images and illustrations should be in black-and-white, or gray tones, excepting for the papers that will be electronically available (on CD-ROMs, internet, etc.). The image resolution on paper should be about 600 dpi for black-and-white images, and 150-300 dpi for grayscale images. Do not include images with excessive resolution, as they may take hours to print, without any visible difference in the result.

12. References

Bibliographic references must be unambiguous and uniform. We recommend giving the author names references in brackets, e.g. [Knuth 1984], [Boulic and Renault 1991], and [Smith and Jones 1999].

The references must be listed using 12 point font size, with 6 points of space before each reference. The first line of each reference should not be indented, while the subsequent should be indented by 0.5 cm.

Tabela 2. Variables to be considered on the evaluation of interaction techniques

	Chessboard top view	Chessboard perspective view
Selection with side movements	6.02 ± 5.22	7.01±6.84
Selection with in-depth movements	6.29±4.99	12.22±11.33
Manipulation with side movements	4.66± 4.94	3.47±2.20
Manipulation with in-depth movements	5.71 ±4.55	5.37 ±3.28

Referências

Boulic, R. and Renault, O. (1991). 3d hierarchies for animation. In Magnenat-Thalmann, N. and Thalmann, D., editors, *New Trends in Animation and Visualization*. John Wiley & Sons ltd.

Knuth, D. E. (1984). *The TEX Book*. Addison-Wesley, 15th edition.

Smith, A. and Jones, B. (1999). On the complexity of computing. In Smith-Jones, A. B., editor, *Advances in Computer Science*, pages 555–566. Publishing Press.