Inicialização e geração de iDVVs com Intel SGX / OpenSGX

Rodrigo Masera de Souza[1], **Rodrigo Bisso Machado**[1], **Diego Kreutz**[1]

[1]Ciência da Computação (CC) e Laboratório de Estudos Avançados (LEA)
Universidade Federal do Pampa (UNIPAMPA)

{rodrigomsr2,rodrigobissomachado}@gmail.com, kreutz@computer.org

Abstract. *High quality and low-cost cryptographic material generators (e.g. secret keys, nonces), such as iDVV, by themselves, are not enough to ensure the confidentiality of sensitive data. Malicious insiders and external attackers can still compromise the machine and get access to the sensitive data. In this paper, we propose Intel SGX for ensuring the integrity and confidentiality of iDVVs, which are secret codes used for encryption and message authentication in programmable networks. Using OpenSGX enclaves, we discuss the implementation and evaluation of an iDVV generator. Our findings suggest that SGX is a viable technology to solve the problem.*

Resumo. *Mecanismos de geração de material criptográfico (e.g. chaves secretas, nonces) de alta qualidade e baixo custo, como o iDVV, por si só, não garantem a confidencialidade desses dados sensíveis em caso de usuários internos maliciosos ou comprometimento da máquina por atacantes externos. Este trabalho propõe a utilização da tecnologia Intel® SGX para isolar e garantir a integridade e a confidencialidade de iDVVs, códigos secretos utilizados para cifragem e autenticação de mensagens em redes programáveis. Uma discussão da implementação e avaliação de um inicializador e gerador de iDVVs, utilizando enclaves OpenSGX, é apresentada neste artigo. Os resultados sugerem SGX é uma tecnologia viável para resolver o problema.*

1. Introdução

Em redes programáveis, mais conhecidas como SDN (*Software-Defined Networking*), a segurança dos canais de comunicação entre os dispositivos de encaminhamento e os controladores é de suma importância [Kreutz et al. 2013, Kreutz 2013, Kreutz et al. 2014, Scott-Hayward et al. 2016]. Soluções de baixo custo computacional para geração de material criptográfico de qualidade, como iDVV (*integrated Device Verification Value*) [Kreutz et al. 2018], e bibliotecas de criptografia de alto desempenho, como NaCL, representam uma opção atrativa às soluções tradicionais (e.g. PKI + OpenSSL) devido à limitada capacidade computacional dos dispositivos de encanamentos e a grande demanda de computação dos controladores em ambiente como data centers, onde a demanda pode chegar a 20 milhões de novos fluxos de rede por segundo [Kreutz et al. 2015].

Um iDVV representa um material criptográfico (e.g. chave secreta, nonce) de alta qualidade (valor pseudo-aleatório de boa entropia) e baixo custo computacional. Os geradores de iDVVs podem ser inicializados através de serviços como um centro de distribuição de chaves (KDC) [Kreutz et al. 2017a, Kreutz et al. 2017b] ou de forma independente, entre dois dispositivos quaisquer, utilizando uma solução baseada em

Diffie-Hellamn (DH), funções pseudo-aleatórias (PRF) e funções de derivação de chaves (KDF) [Masera et al. 2018]. Entretanto, ambas as soluções, apesar de garantirem a geração de material criptográfico de alta qualidade, por si só, não garantem a confidencialidade da inicialização, geração e utilização das informações sensíveis em caso de comprometimento do dispositivo de encaminhamento ou do controlador, por exemplo. Como forma de resolver o problema, este trabalho propõe a utilização da tecnologia Intel® SGX (*Software Guard Extensions*) [Costan et al. 2017].

SGX permite que um aplicativo, ou seu sub-componente, seja executado em um ambiente de execução isolado, denominado de *enclave*. O hardware Intel® SGX protege o *enclave* contra qualquer software malicioso, incluindo sistema operacional, hypervisor e firmware de baixo nível, que possa vir a tentar comprometer a integridade do *enclave* ou roubar informações sigilosas como chaves secretas [Costan et al. 2017, Jain et al. 2016, Arnautov et al. 2016]. O código e os dados de enclave residem em uma região de memória física protegida chamada cache de página de enclave (EPC). Enquanto dentro da cache, código e dados são protegidos pelos controles de acesso da CPU. Quando movidos para DRAM, os dados das páginas EPC são protegidos (cifrados) pelo mecanismo de criptografia de memória no chip, também conhecido como *Memory Encryption Engine* (MEE).

O objetivo principal deste trabalho é apresentar, discutir e avaliar um gerador de iDVVs baseado na tecnologia Intel® SGX. Para viabilizar a implementação e avaliação sem a necessidade de aquisição de hardware específico, pode ser utilizado o emulador SGX denominado OpenSGX [Jain et al. 2016]. Este emulador permite a implementação, execução e avaliação de programas em *enclaves* SGX. Os resultados sugerem que a utilização de SGX é uma alternativa viável para cenários onde segurança é uma das prioridades, ou seja, garantir a integridade e confidencialidade da inicialização e/ou geração dos iDVVs.

As principais contribuições do trabalho são: (i) introdução da tecnologia Intel® SGX e OpenSGX no contexto da segurança de material criptográfico em redes programáveis; (ii) implementação e avaliação da inicialização do gerador de iDVVs, baseada no método DH + PRF + KDF [Masera et al. 2018], utilizando OpenSGX; e (iii) discussão dos resultados e apresentação de caminhos futuros de investigação.

O restante deste trabalho está organizado como segue. A implementação da solução proposta é discutida na Seção 2. Os resultados e as considerações finais e trabalhos futuros são apresentados nas Seções 3 e 4, respectivamente. Por fim, vale ressaltar que, devido à limitação de espaço, outras informações, incluindo uma conceituação e discussão detalhada de iDVVs, Intel® SGX e OpenSGX, uma relação e discussão de trabalhos similares que utilizam SGX e/ou outras tecnologias (e.g. TrustLite) em contextos específicos, podem ser encontrados na versão estendida do trabalho [de Souza et al. 2018].

2. Implementação

Recentemente, os autores propuseram um método para inicializar os geradores dos iDVVs que não depende de um KDC [Masera et al. 2018]. Este método permite que duas entidades (e.g. dispositivos de rede) A e B quaisquer, sem intermédio de uma terceira parte, estabeleçam canais de comunicação seguros utilizando iDVVs. Para isso, foram criados e avaliados diferentes métodos, entre os quais foi selecionado o método denominado

DH+PRF+KDF, que é utilizado apenas para inicializar os geradores de iDVVs, mais especificamente, atribuir os valores iniciais da semente (*seed*) e chave (*key*) utilizados no *bootstrap* do gerador.

O porte do método DH+PRF+KDF para OpenSGX, desenvolvido originalmente em Python, exigiu a troca de linguagem de programação, pois SGX e OpenSGX suportam apenas a linguagem de programação C. Vale ressaltar que, para fins de comparação (ver Seção 3), foram implementadas duas versões do método, uma para OpenSGX e outra para um ambiente Linux tradicional.

No caso do OpenSGX, foram utilizadas apenas bibliotecas padrão da linguagem C, APIs OpenSGX e os portes das bibliotecas de criptografia PolarSSL e OpenSSL. Como pode ser observado no trecho de código da Listing 1, há funções específicas do ambiente SGX, como `enclave_main()` (ao invés da tradicional função `main()`), `sgx_write_sock()`, `sgx_read_sock()` e `sgx_exit()` (linhas 4, 8, 11 e 26, respectivamente). As funções `Diffie_Hellman()`, `PRF()` e `KDF()` (linhas 14, 17 e 20) representam implementações destas funções em linguagem C, seguindo as definições apresentadas em [Masera et al. 2018]. Por fim, o gerador de iDVVs é inicializado com a chamada da função `iddv_init()` (linha 24).

Para implementar as funções `PRF()` e `iddv_init()`, foram utilizadas as primitivas criptográficas SHA256 e HMAC-SHA256, através das funções `sha256()` e `sha256_hmac()` da biblioteca PolarSSL. Estas funções são necessárias à segurança da comunicação entre os dispositivos e são utilizadas para gerar valores pseudo-aleatórios (SHA256 e HMAC-SHA256) e códigos de autenticação (HMACs) que asseguram a integridade e autenticidade das mensagens.

```
1  #include "test.h"
2  #include "polarssl/sha256.h"
3  ...
4  void enclave_main()
5  {
6      ...
7      // envia parametros Diffie-Hellman
8      sgx_write_sock(client_fd, buffer, len(buffer));
9      ...
10     // recebe parametros Diffie-Hellman
11     sgx_read_sock(client_fd, buffer, size(buffer));
12     ...
13     // calcula chave secreta pre-mestra
14     preMasterSecret = Diffie_Hellman(clientPubKey, privKey, modulus);
15     ...
16     // calcula chave mestra
17     masterSecret = PRF(preMasterSecret, prf_seed);
18     ...
19     // deriva a chave secreta e a seed da chave mestra
20     idvv_key = KDF(masterSecret, kdf_seed, 3);
21     idvv_seed = KDF(masterSecret, kdf_seed+idvv_key, 3);
22     ...
23     // inicializa o gerador de iDVVs
24     iddv_init(idvv_seed, idvv_key);
25     ...
26     sgx_exit(NULL);
```

```
27 }
```

Listing 1. Implementação utilizando OpenSGX

3. Resultados

Esta seção apresenta uma avaliação e comparação de desempenho das duas novas implementações do método DH+PRF+KDF. As implementações foram realizadas na linguagem de programação C, sendo uma para OpenSGX (`https://github.com/sslab-gatech/opensgx`, versão de setembro de 2018) e outra para rodar em sistemas GNU/Linux.

O objetivo da avaliação foi comparar as duas implementações do método para verificar os eventuais gargalos ou *overheads* da tecnologia SGX, uma vez que uma das principais vantagens dos iDVVs são o baixo custo computacional e o alto desempenho. Os testes foram realizados em uma máquina virtual (MV) VirtualBox, com 1 GB de memória, 1 core e sistema operacional Ubuntu Server 16.04. A MV foi executada num notebook HP 14-d030br, Intel core i5, 2^a geração, 8GB de RAM e 240GB de SSD, rodando Ubuntu 17.10. Foram executados três testes distintos, um na plataforma OpenSGX, um no modo usuário do QEMU e um terceiro nativo GNU/Linux. O modo usuário do QEMU foi utilizado pelo fato de o OpenSGX utilizar o QEMU. Portanto, a execução sobre o QEMU fornece uma base de comparação mais justa.

As ferramentas `perf` e `gcc` foram utilizadas para coletar os dados de código e de execução dos programas. Foram realizadas 1.000 execuções para medir o tempo médio de execução da função `idvv_next()`, da primitiva criptográfica `sha256()` e do `tempo de execução`.

Os resultados estão resumidos na Tabela 1. Foram levados em consideração cinco parâmetros de avaliação e comparação, sendo eles o tempo total de execução, número de ciclos de CPU, número de instruções, tempo da função `idvv_next()`, tempo da função de hash `SHA256()`. Os tempos são todos em milisegundos (ms). Como esperado, a execução **nativa** apresenta o melhor desempenho, pois não carrega consigo a sobrecarga da emulação. Já o cenário **QEMU**, apesar de o número de instruções e ciclos de CPU permanecerem os mesmos, reduz consideravelmente o desempenho uma vez que o mesmo código da execução **nativa**, compilado estaticamente, está agora sendo executado sobre o emulador QEMU. Devido a isso, o tempo total de execução aumenta em 28x. Entretanto, pode-se observar que o tempo de execução da função geradora de iDVVs, `idvv_next()`, aumentou apenas 7x. Isto significa que o maior *overhead* da emulação está na inicialização da execução e não na execução das funções propriamente ditas.

Parâmetro	**Nativo**	**QEMU**	**OpenSGX**
Tempo de execução (ms)	$0,26$	$7,58$	$136,66$
Ciclos de CPU	$1.520.638$	$1.520.638$	$4.812.588$
Número Instruções	825	825	2611
Tempo `idvv_next()`	$0,0098$	$0,0726$	$33,29$
Tempo `SHA256()`	$0,0059$	$0,0297$	$16,1153$

Tabela 1. Comparação das execuções nativa, QEMU e OpenSGX

No terceiro cenário, execução com **OpenSGX**, há um aumento de 18x no tempo total de execução quando comparado com o cenário equivalente, o **QEMU**. Entretanto, vale ressaltar que o código está sendo executado sobre uma emulação OpenSGX em um emulador QEMU, ou seja, há dois níveis de emulação. Além disso, o código OpenSGX possui cerca de 3x mais instruções assembler e ciclos de CPU. De qualquer forma, num hardware Intel® SGX, essa diferença deve cair uma vez que o desempenho de um código rodando em um *enclave* SGX é superior ao desempenho do mesmo código rodando no emulador OpenSGX.

O overhead da tecnologia SGX ocorre por dois motivos [Arnautov et al. 2016]: (1) *enclaves* consideram o kernel do sistema operacional inseguro; e (2) baixo desempenho em memória. O primeiro motivo retrata as consequências de encapsular informação do sistema operacional, que ocorre quando uma thread precisa realizar chamadas de sistema, onde o contexto do *enclave* deve ser salvo antes que a informação possa ser enviada à memória para a realização da chamada de sistema. O baixo desempenho em memória é resultado direto dos mecanismos de segurança do SGX. Quando ocorre um *cache-miss*, as linhas de cache devem ser decifradas antes de realizar uma busca na memória, o que significa uma operação adicional custosa, gerando um *overhead* significativo. Este é um dos principais *trade-offs* entre segurança e desempenho no contexto da tecnologia SGX.

Apesar de existir uma sobrecarga que não deve ser desprezada com SGX, a decisão entre utilizar ou não a tecnologia deve levar em conta o *trade-off* desempenho-segurança. Se o desempenho tem prioridade sobre a segurança, então recomenda-se a utilização da tecnologia SGX somente para a inicialização dos geradores de iDVVs. Neste caso, depois da inicialização, o gerador irá manter o estado interno em memória insegura. Entretanto, um atacante conseguirá comprometer apenas as sessões atual e futuras uma vez que o iDVV evolui de forma irreversível, isto é, o gerador de iDVV fornece a propriedade de segurança conhecida por *perfect forward secrecy* (PFS). Mesmo que o atacante tenha uma cópia (cifrada) de todas as comunicações passadas entre duas entidades A e B, ele não conseguirá comprometer a confidencialidade sem conhecer os dados originais de inicialização, que estão disponíveis somente dentro de um *enclave* SGX.

Por outro lado, se a segurança é a maior prioridade, recomenda-se a utilização de soluções baseadas em SGX, ou seja, deixar tanto a inicialização quanto a geração dos iDVVs dentro de *enclaves* SGX. Por fim, vale lembrar que Intel® SGX é uma tecnologia nova, ou seja, ela tende a evoluir significativamente nos próximos anos. Esta evolução irá, como uma boa probabilidade, levar em conta o desempenho, como tem ocorrido com praticamente todos mecanismos e recursos de segurança (e.g. conjunto de instruções AES) suportadas em hardware.

4. Considerações Finais

A implementação do método de inicialização de geradores de iDVVs, conhecido como DH+PRF+KDF, na plataforma OpenSGX, se mostrou viável na prática. Apesar do *overhead* de execução, identificado nos testes prático, esta solução, baseada na tecnologia SGX, pode ser utilizada na prática considerando dois casos em particular. O primeiro é o caso de cenários onde o desempenho tem prioridade sobre a segurança. Nestes casos, recomenda-se a utilização da tecnologia SGX somente para a inicialização dos geradores de iDVVs, como discutido na seção 3. Por outro lado, se a segurança é a maior prioridade,

recomenda-se a utilização de soluções completamente baseadas em SGX, ou seja, deixar tanto a inicialização quanto a geração dos iDVVs dentro de *enclaves* SGX.

Como trabalhos futuros podem ser mencionados: (1) investigar o overhead da solução em máquinas que oferecem as últimas gerações da tecnologia Intel® SGX; (2) analisar o impacto de diferentes ataques contra os *enclaves* SGX que suportam os geradores de iDVVs; (3) realizar um estudo de viabilidade técnica e comercial de inclusão da tecnologia SGX em dispositivos de rede, como comutadores.

Referências

Arnautov, S., Trach, B., Gregor, F., Knauth, T., Martin, A., Priebe, C., Lind, J., Muthukumaran, D., O'keeffe, D., Stillwell, M., et al. (2016). SCONE: Secure Linux Containers with Intel SGX. In *OSDI*, volume 16, pages 689–703.

Costan, V., Lebedev, I., and Devadas, S. (2017). Secure processors part i: Background, taxonomy for secure enclaves and intel sgx architecture. *Foundations and Trends® in Electronic Design Automation*, 11(1-2):1–248.

de Souza, R. M., Machado, R. B., and Kreutz, D. (2018). Inicialização e geração de iDVVs com Intel SGX / OpenSGX. https://goo.gl/RpR48F.

Jain, P., Desai, S. J., Shih, M.-W., Kim, T., Kim, S. M., Lee, J.-H., Choi, C., Shin, Y., Kang, B. B., and Han, D. (2016). OpenSGX: An open platform for sgx research. In *USENIX NDSS*.

Kreutz, D. (2013). Designing dependable control platforms for software defined networks. In *IEEE SRDS*. https://goo.gl/AjePLJ.

Kreutz, D., Ramos, F., and Verissimo, P. (2014). Anchors of trust for autonomic and secure configuration and assessment in SDN. In *IEEE/IFIP DSN*. https://goo.gl/QjfB5c.

Kreutz, D., Ramos, F. M., and Verissimo, P. (2013). Towards secure and dependable software-defined networks. In *ACM SIGCOMM HotSDN*, pages 55–60, New York, NY, USA. ACM.

Kreutz, D., Ramos, F. M. V., Veríssimo, P. E., Rothenberg, C. E., Azodolmolky, S., and Uhlig, S. (2015). Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76.

Kreutz, D., Yu, J., Esteves-Verissimo, P., Magalhaes, C., and Ramos, F. M. V. (2017a). The KISS principle in Software-Defined Networking: An architecture for Keeping It Simple and Secure. *ArXiv e-prints*.

Kreutz, D., Yu, J., Ramos, F. M. V., and Esteves-Verissimo, P. (2017b). ANCHOR: logically-centralized security for Software-Defined Networks. *ArXiv e-prints*.

Kreutz, D., Yu, J., Ramos, F. M. V., and Esteves-Verissimo, P. (2018). The KISS principle in Software-Defined Networking: a framework for secure communications. *IEEE Security & Privacy*. Accepted. Publication expected by the end of October 2018.

Masera, R., Bisso, R., and Kreutz, D. (2018). Diffie-Hellman (DH), PRF e KDF: Uma alternativa para inicializar iDVVs. SIEPE. https://goo.gl/pb1as3.

Scott-Hayward, S., Natarajan, S., and Sezer, S. (2016). A survey of security in software defined networks. *IEEE Communications Surveys & Tutorials*, 18(1):623–654.