Inicialização e geração de iDVVs com Intel SGX / OpenSGX

Rodrigo Masera, Rodrigo Machado, Diego Kreutz

3o. Workshop Regional de Segurança da Informação (2018)

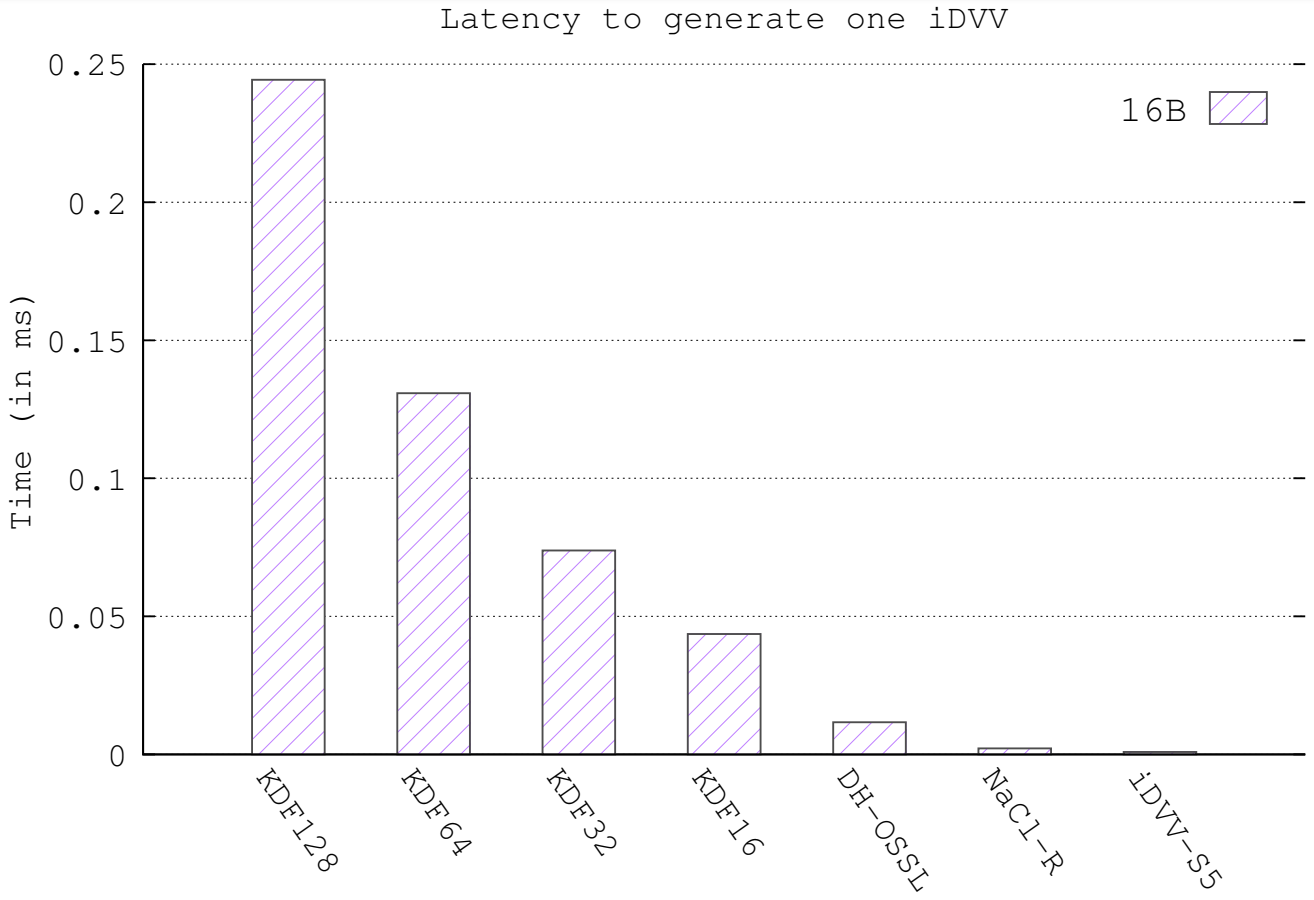
iDVV: O que é?

- integrated Device Verification Value (**iDVV**)
  - integrated Card Verification Value (**iCVV**)
- iDVV = material criptográfico de baixo custo

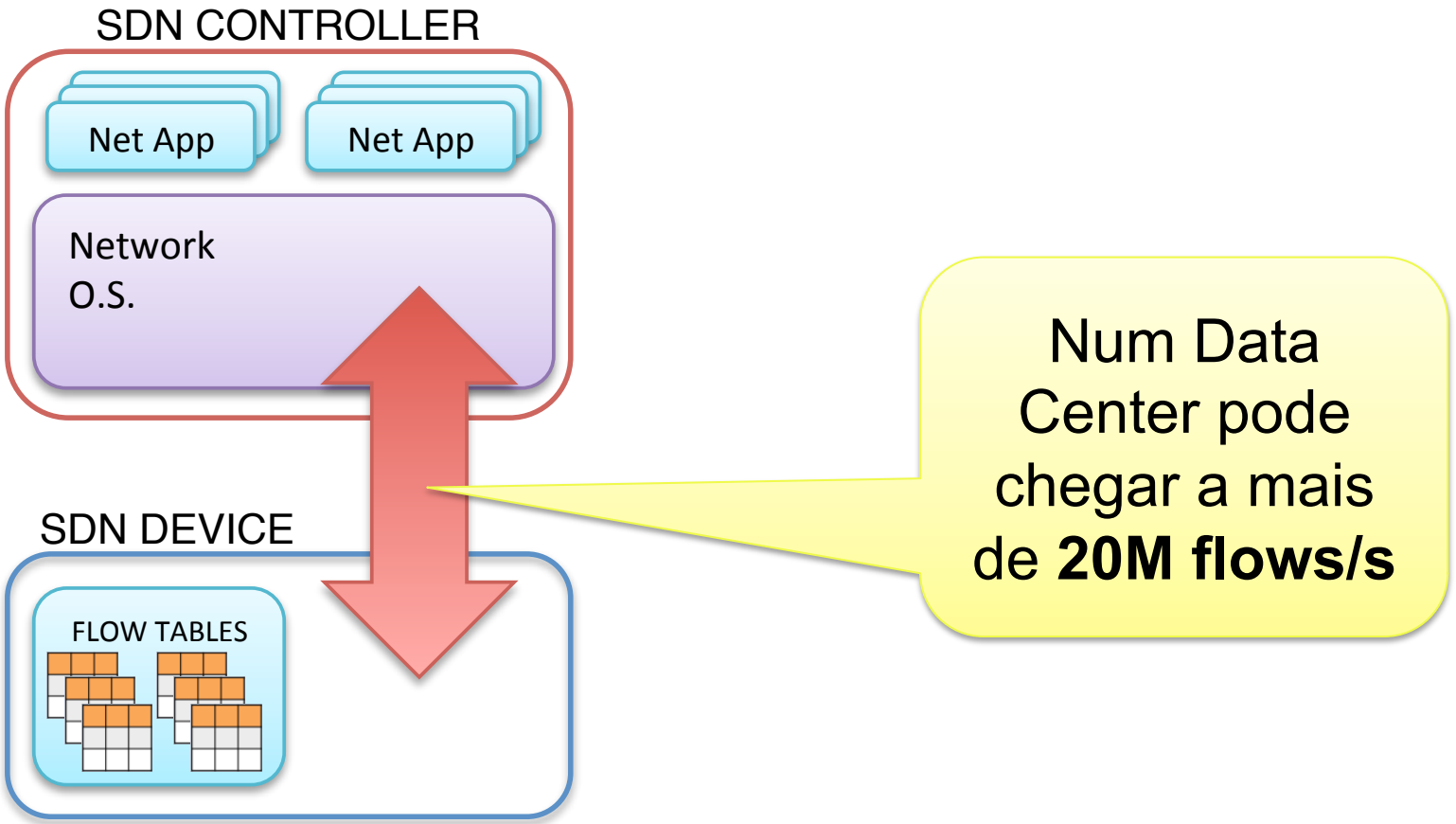
Algorithm 2: iDVV generation

```
1: idvv_next()
2:     seed ← H(seed || idvv)
3:     idvv ← H(seed || key)
```

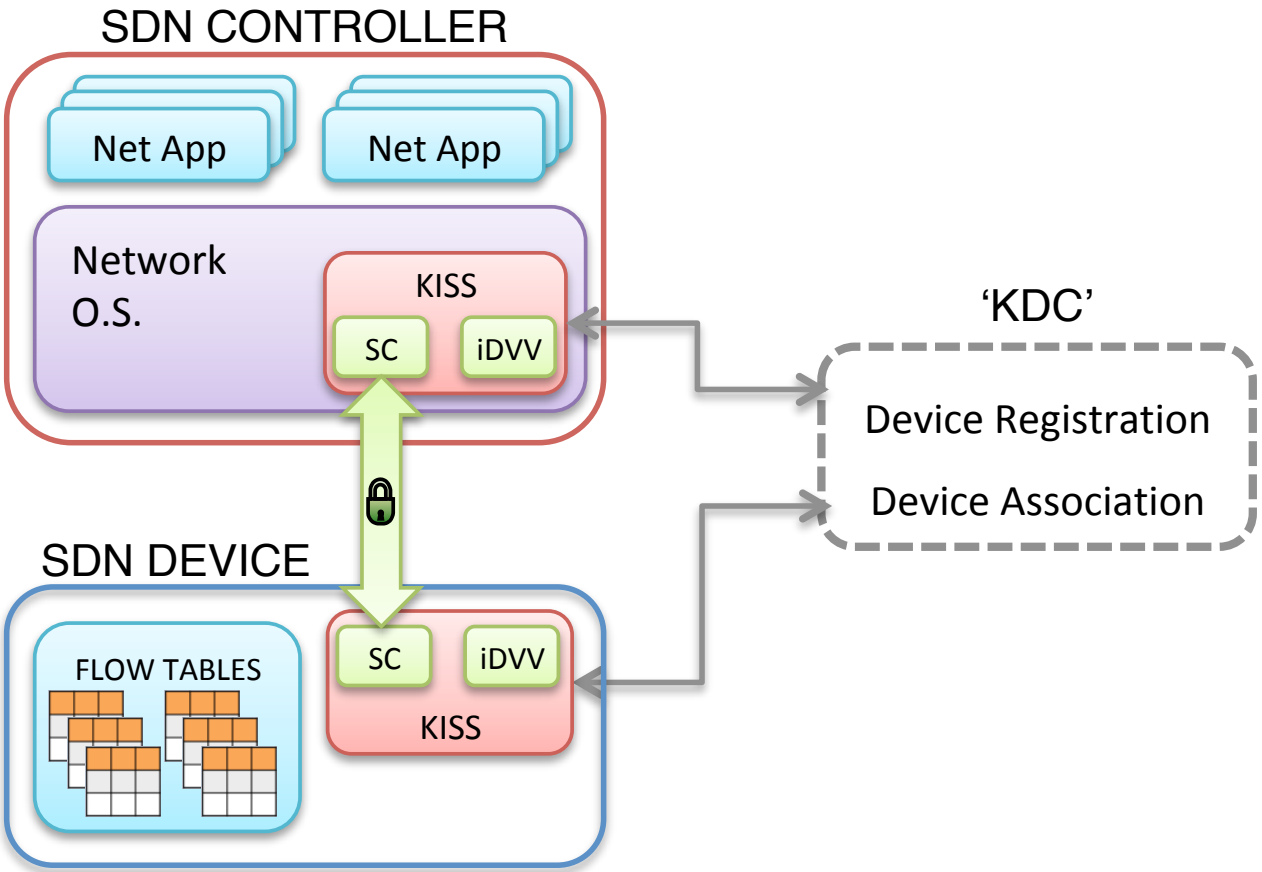
iDVV: O que é?


Latency to generate one iDVV
Time (in ms)
0.25
0.2
0.15
0.1
0.05
0
16B
KDF128
KDF64
KDF32
KDF16
DH-OSSL
NaCl-R
iDVV-S5

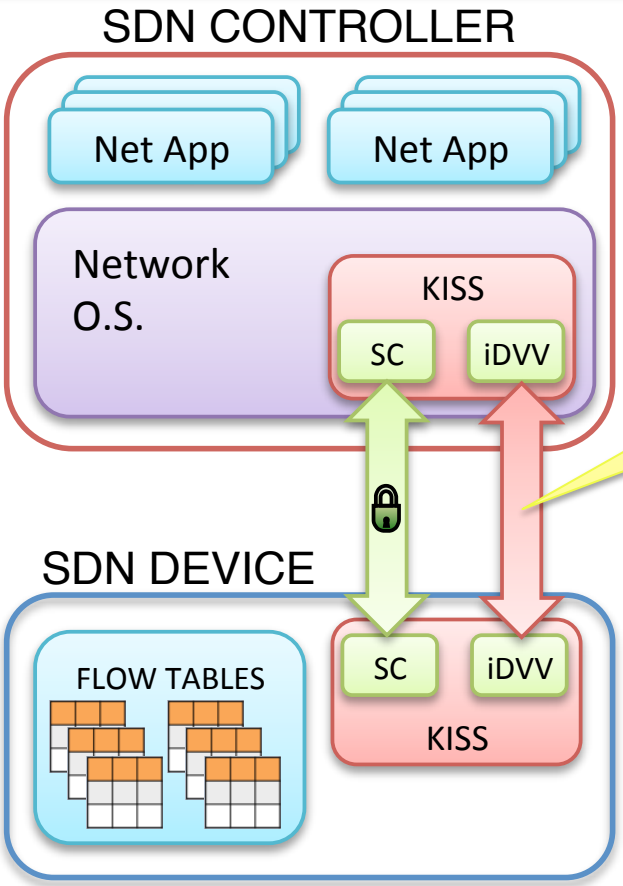
iDVV: Aplicação (caso de uso)



iDVV: Inicialização



iDVV: Inicialização

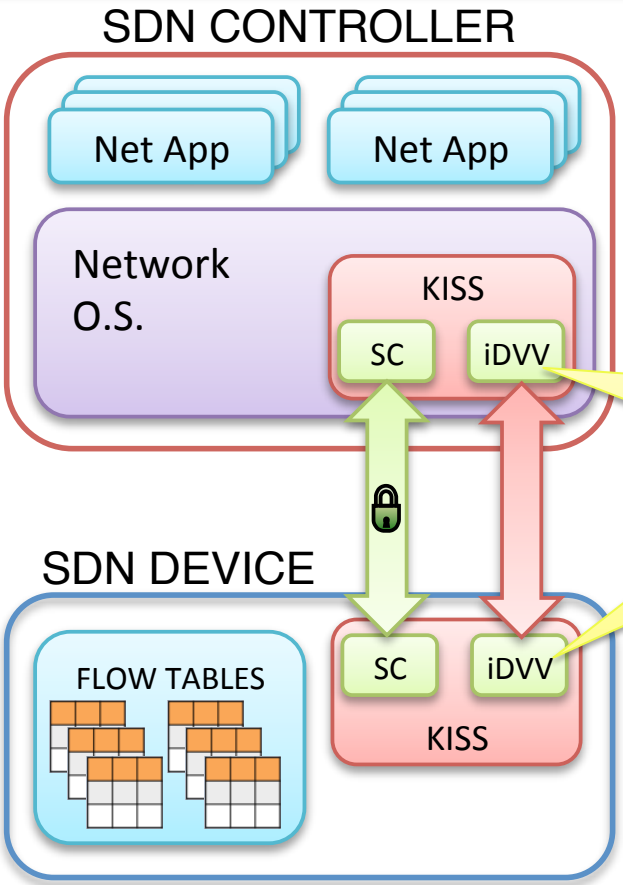


DH+PRF+KDF
(sem KDC)

Algorithm 1: iDVV set-up

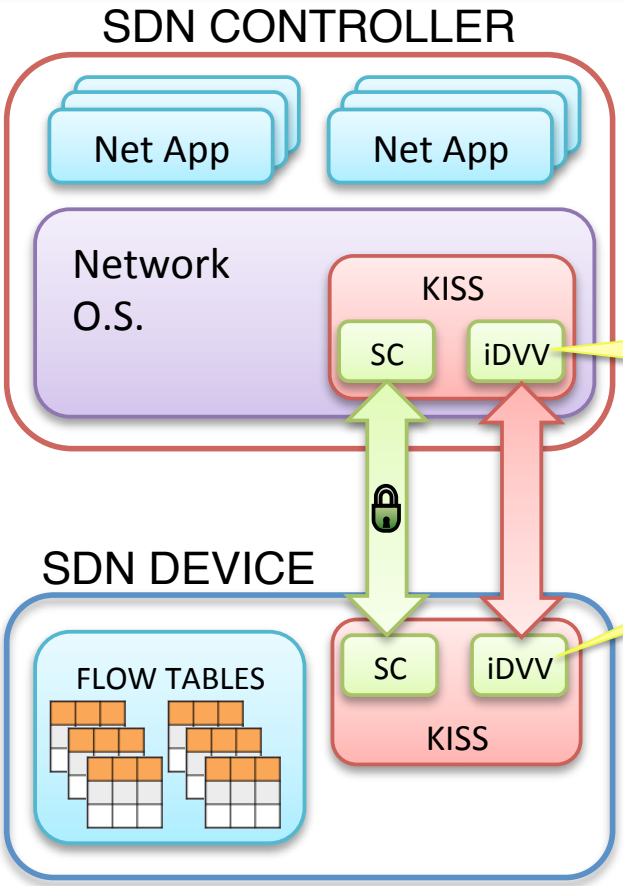
```
1: idvv_init()
2:     idvv ← H(seed || key)
```

iDVV: Qual o problema?

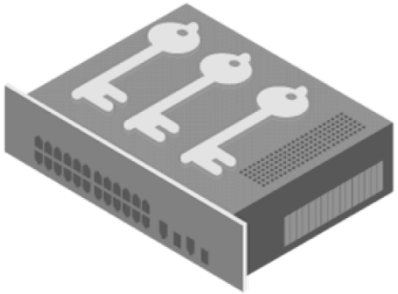


Integridade e **confidencialidade** da inicialização e geração de iDVVs

iDVV: Qual a solução?



Segurança assistida através de tecnologia de **hardware**

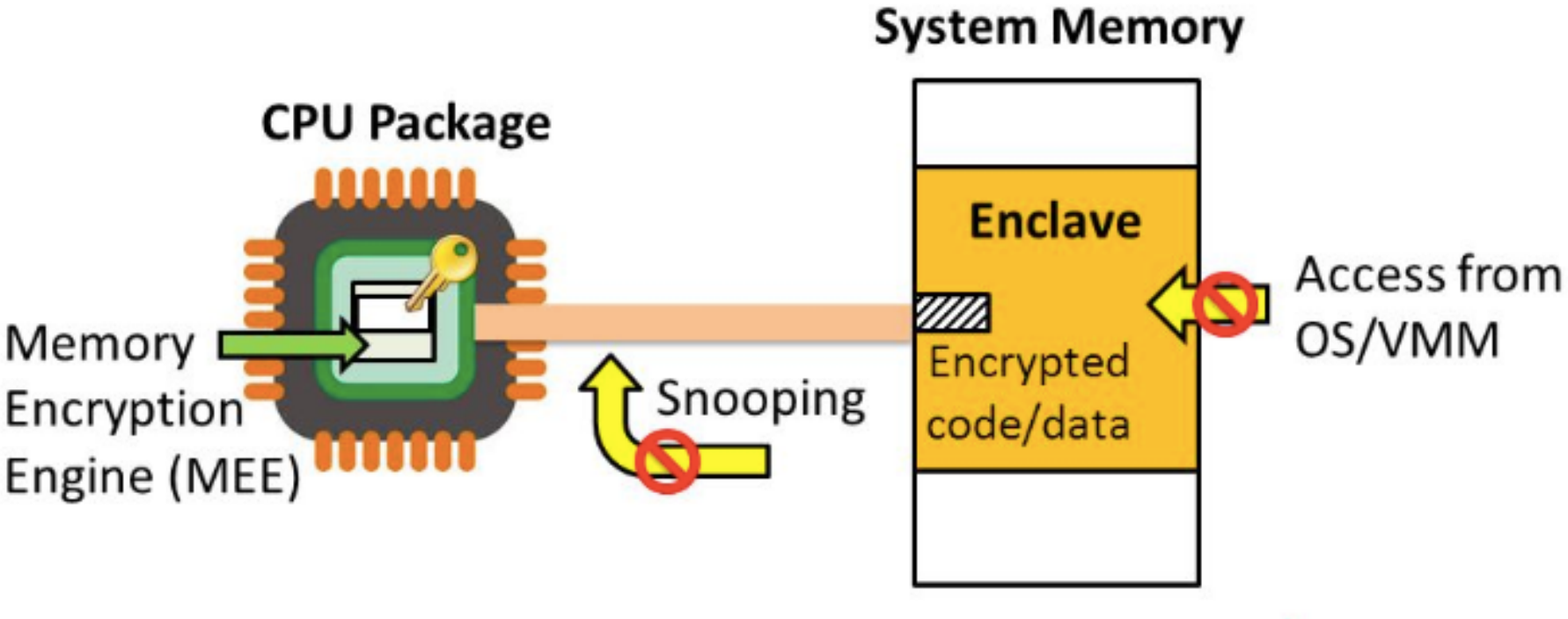


Roteiro

- **Intel SGX / OpenSGX**
- Implementação e Resultados
- Consideração Finais
- Trabalhos Futuros

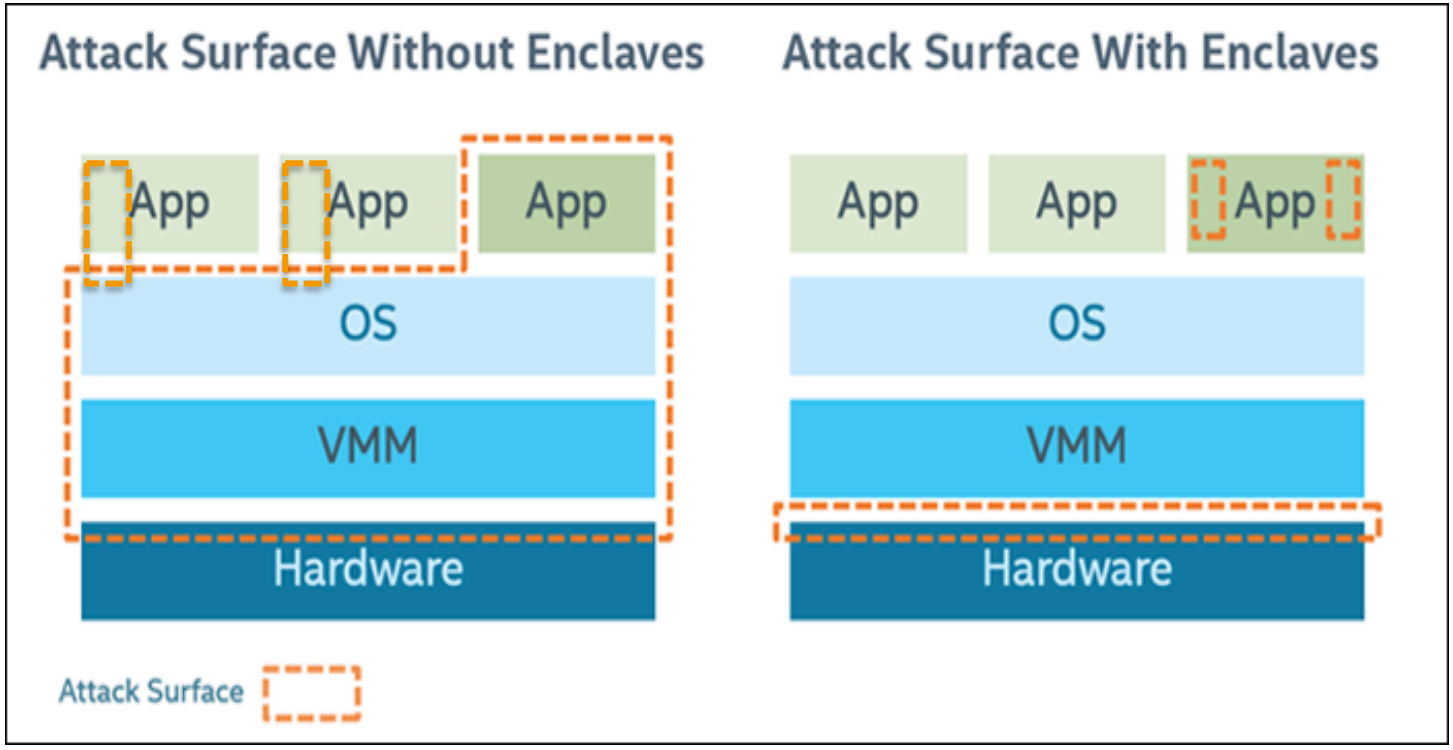
Intel SGX: O que é?

- Execução isolada (dados e código ficam dentro do "**enclave**")



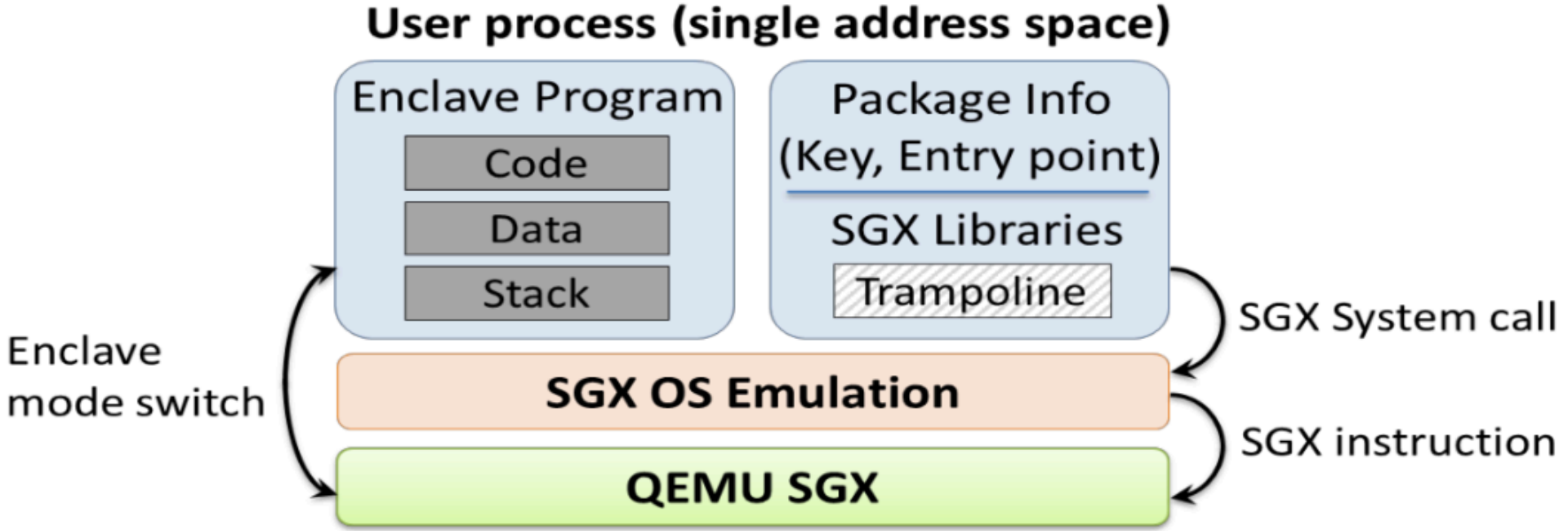
Intel SGX: O que faz?

- Reduz a superfície de ataque (TCB reduzida)



OpenSGX: O que é?

- https://github.com/sslab-gatech/opensgx



Roteiro

Intel SGX / OpenSGX

Implementação e Resultados

Consideração Finais

Trabalhos Futuros

Ambiente de desenvolvimento e testes

- Hardware
  - HP 14-d030br
  - Intel core i5, 2nd Gen
  - 8GB de RAM
  - 240GB de SSD
  - Ubuntu 17.10
- VirtualBox
  - 1GB de RAM
  - 1 core
  - Ubuntu Server 16.04

Implementação (OpenSGX e nativa)

- Migração de Python para C
- Funções nativas do OpenSGX
- Bibliotecas portadas, como PolarSSL

```
#include "test.h"
#include "polarssl/sha256.h"
...
void enclave_main()
{
    ...
    // envia parametros Diffie-Hellman
    sgx_write_sock(client_fd, buffer, len(buffer));
    ...
    // recebe parametros Diffie-Hellman
    sgx_read_sock(client_fd, buffer, size(buffer));
    ...
    // calcula chave secreta pre-mestra
    preMasterSecret = Diffie_Hellman(clientPubKey, privKey, modulus);
    ...
    // calcula chave mestra
    masterSecret = PRF(preMasterSecret, prf_seed);
    ...
    // deriva a chave secreta e a seed da chave mestra
    idvv_key = KDF(masterSecret, kdf_seed, 3);
    idvv_seed = KDF(masterSecret, kdf_seed+idvv_key, 3);
    ...
    // inicializa o gerador de iDVVs
    iddv_init(idvv_seed, idvv_key);
    ...
    sgx_exit(NULL);
}
```

Testes

- Três plataformas
  - Nativo (GNU/Linux)
  - QEMU (modo usuário)
  - OpenSGX
- Ferramentas
  - perf
  - gcc
- Principal métrica: média de 1.000 execuções

Resultados

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

Resultados

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

Resultados

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

Resultados

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

Resultados

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

Resultados

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

Execução nativa x QEMU

- Nativo x Emulado
- Tempo execução: Diferença de 28x
- Overhead na inicialização!

QEMU x OpenSGX

- Tempo execução: Diferença de 18x
- OpenSGX: 3x + instruções assembler
- OpenSGX: 3x + ciclos de CPU

Discussão do overhead

- Com SGX **SO** é considerado **inseguro**
  - Chamadas de sistema
  - Salvar/carregar contexto do **enclave**
- Baixo desempenho na memória
  - Cache-miss
  - **MEE** (*Memory Encryption Engine*)
  - Cifrar/decifrar dados da/para a memória

Roteiro

Intel SGX / OpenSGX

Implementação e Resultados

Consideração Finais

Trabalhos Futuros

Considerações finais

- *Trade-off* desempenho-segurança
- Se o **desempenho** é prioridade
  - Somente inicialização (**idvv_init**) com SGX
- Se a **segurança** é prioridade
  - Tanto inicizalização quanto geração (**idvv_init** + **idvv_next**) com SGX

Roteiro

Intel SGX / OpenSGX

Implementação e Resultados

Consideração Finais

Trabalhos Futuros

Trabalhos Futuros

- Overhead em máquinas Intel SGX
- Impacto de diferentes ataques
- Estudo de viabilidade técnica e comercial para dispositivos de rede

unipampa

Universidade Federal do Pampa

Obrigado!

Contatos:

rodrigomsr2@gmai.com

rodrigobissomachado@gmail.com

kreutz@unipampa.edu.br