



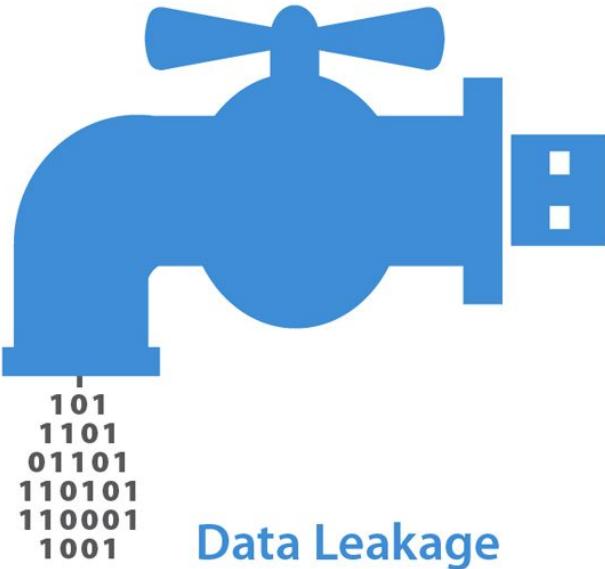
SeguraAí: Confidencialidade de Dados Sensíveis com SGX

Autor1, Autor2, Autor3, Autor4, ...

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

Vazamento de dados

- Bugs de implementação e falhas configuração



Vazamento de dados

- Até 70% das app Web são vulneráveis [Fonte: <http://bit.do/vulnerable-apps>]



Vazamento de dados

- 90% das Crypto Mobile Apps são vulneráveis
[Fonte: <http://bit.do/90-percent-vulnerable>]



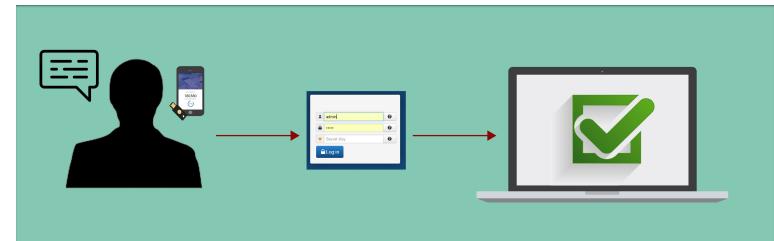
Vazamento de dados

- Vazamentos de larga-escala em grandes infraestruturas (e.g. **nuvens de computação e armazenamento** públicas e privadas)

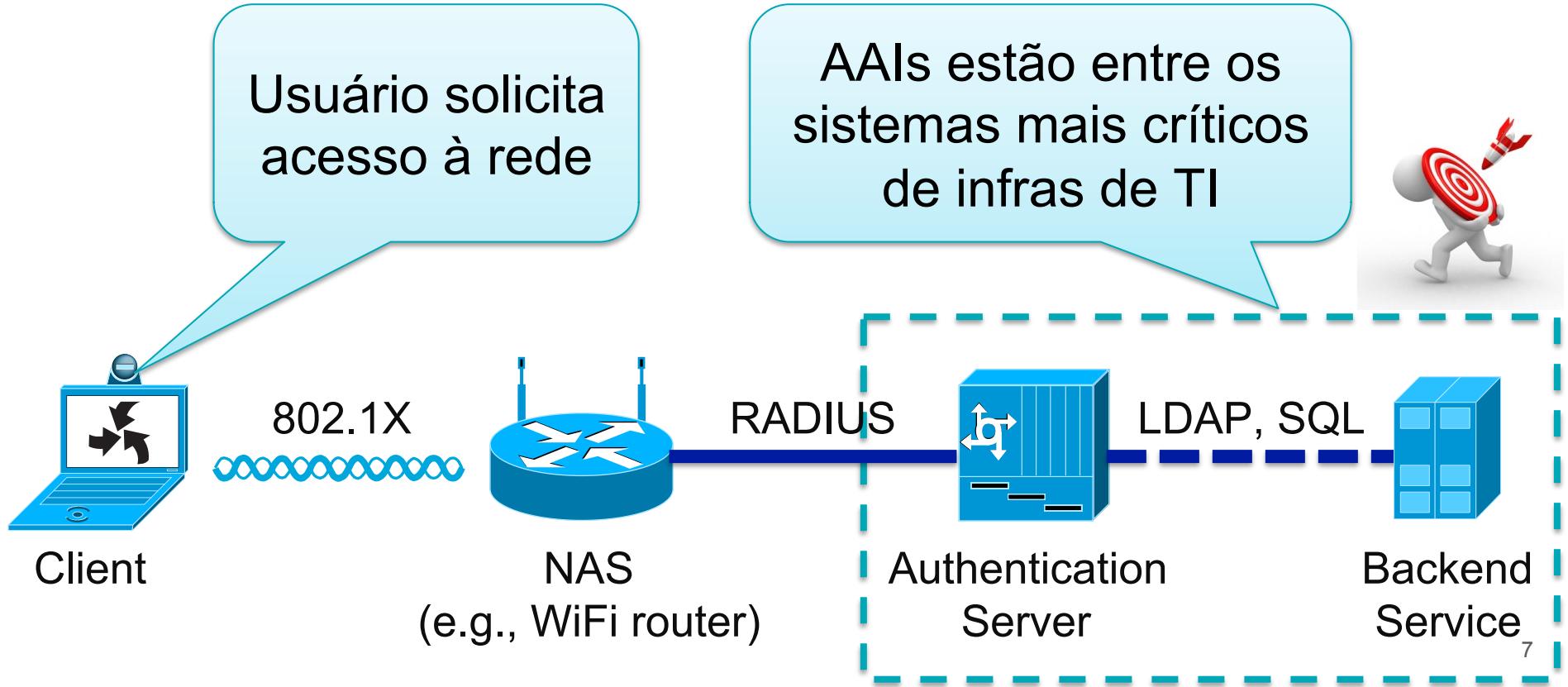


Vazamento de dados

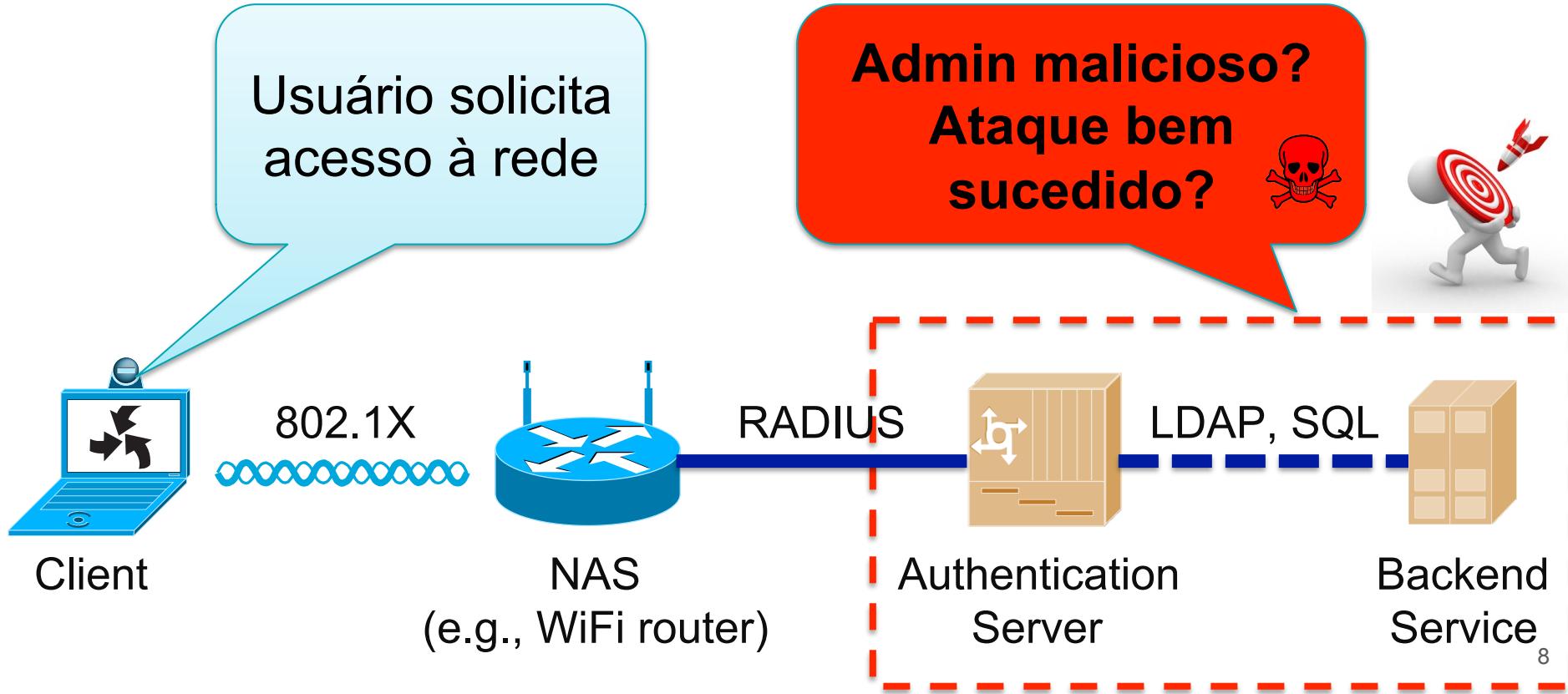
- Dados mais **sensíveis** e **visados**
 - Identificação e autenticação de usuários
 - Autorização de sistemas
 - Dados financeiros



Vazamento de dados



Vazamento de dados



Roteiro

Intel SGX / OpenSGX

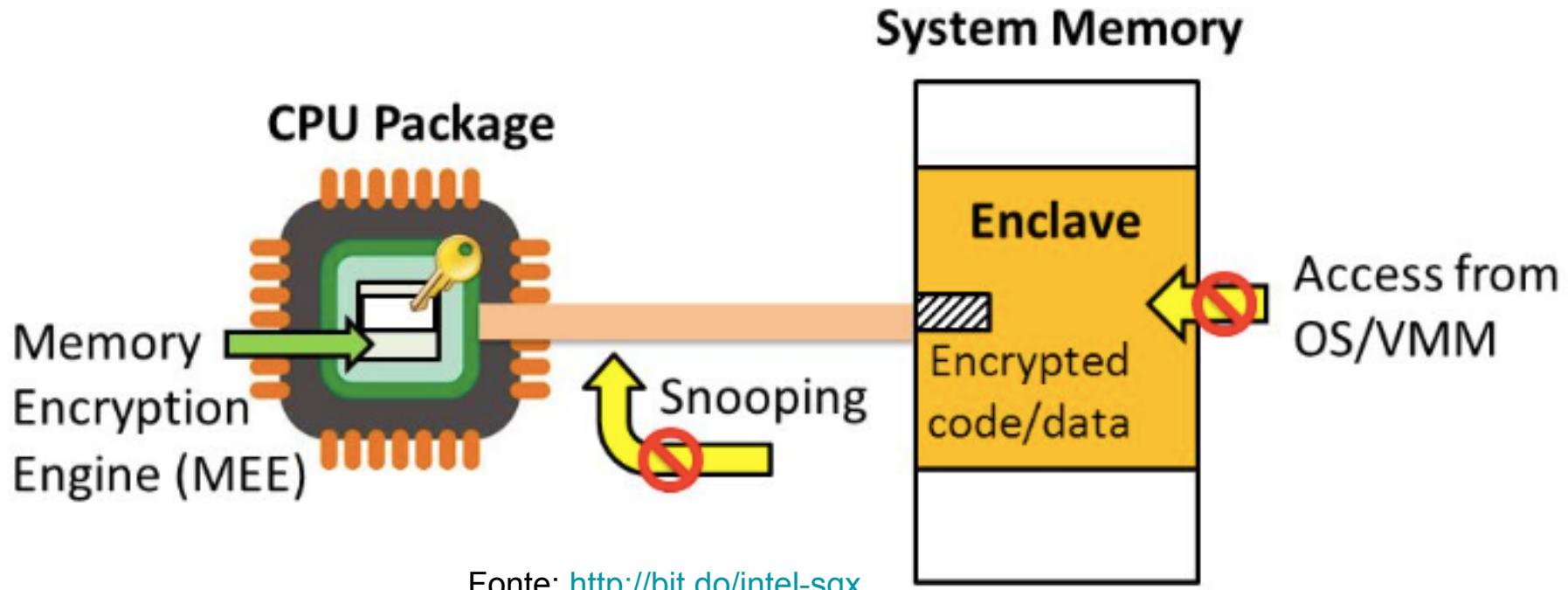
Arquitetura SeguraAí

Implementação e Resultados

Considerações Finais

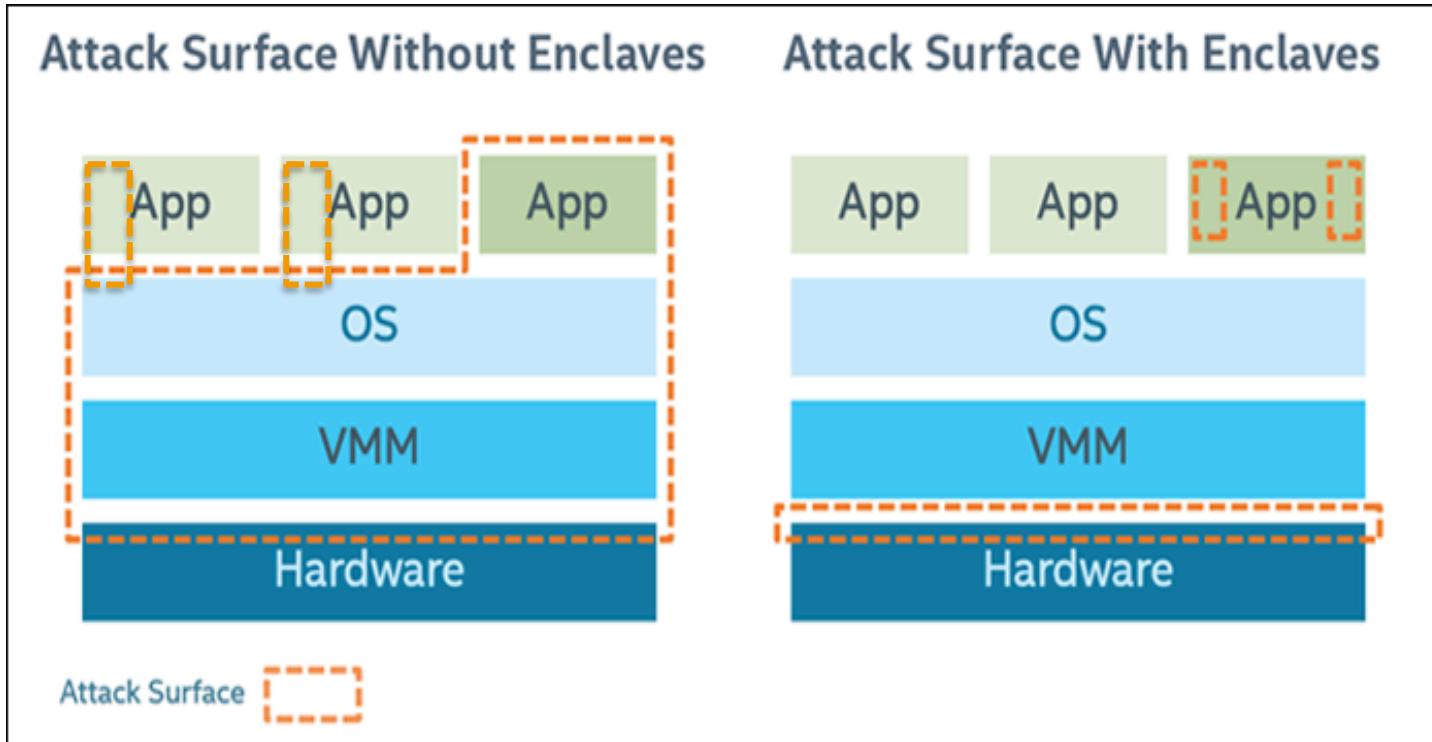
Intel SGX: O que é?

- Execução isolada (dados e código ficam dentro do “enclave”)



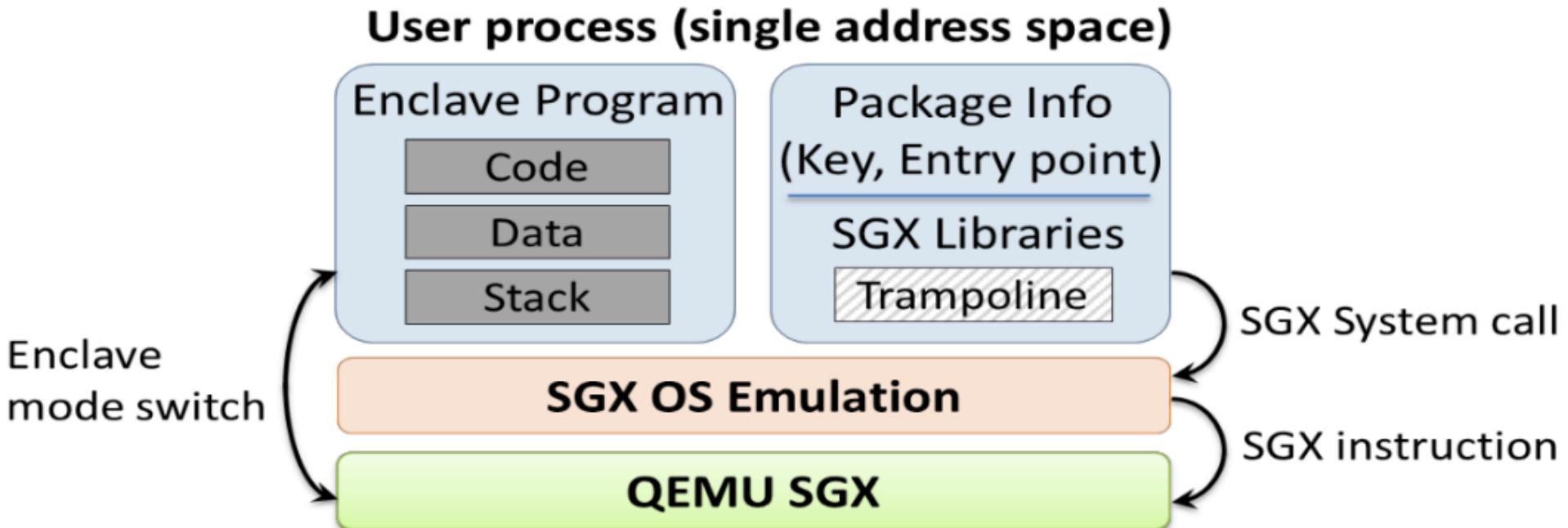
Intel SGX: O que faz?

- Reduz a superfície de ataque (TCB reduzida)



Fonte: <http://bit.do/attack-surface>

OpenSGX: O que é?



Fonte: <https://github.com/sslab-gatech/opensgx>

Roteiro

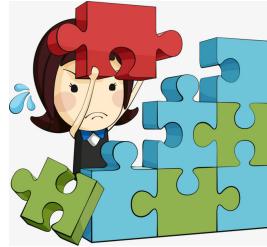
Intel SGX / OpenSGX

Arquitetura SeguraAí

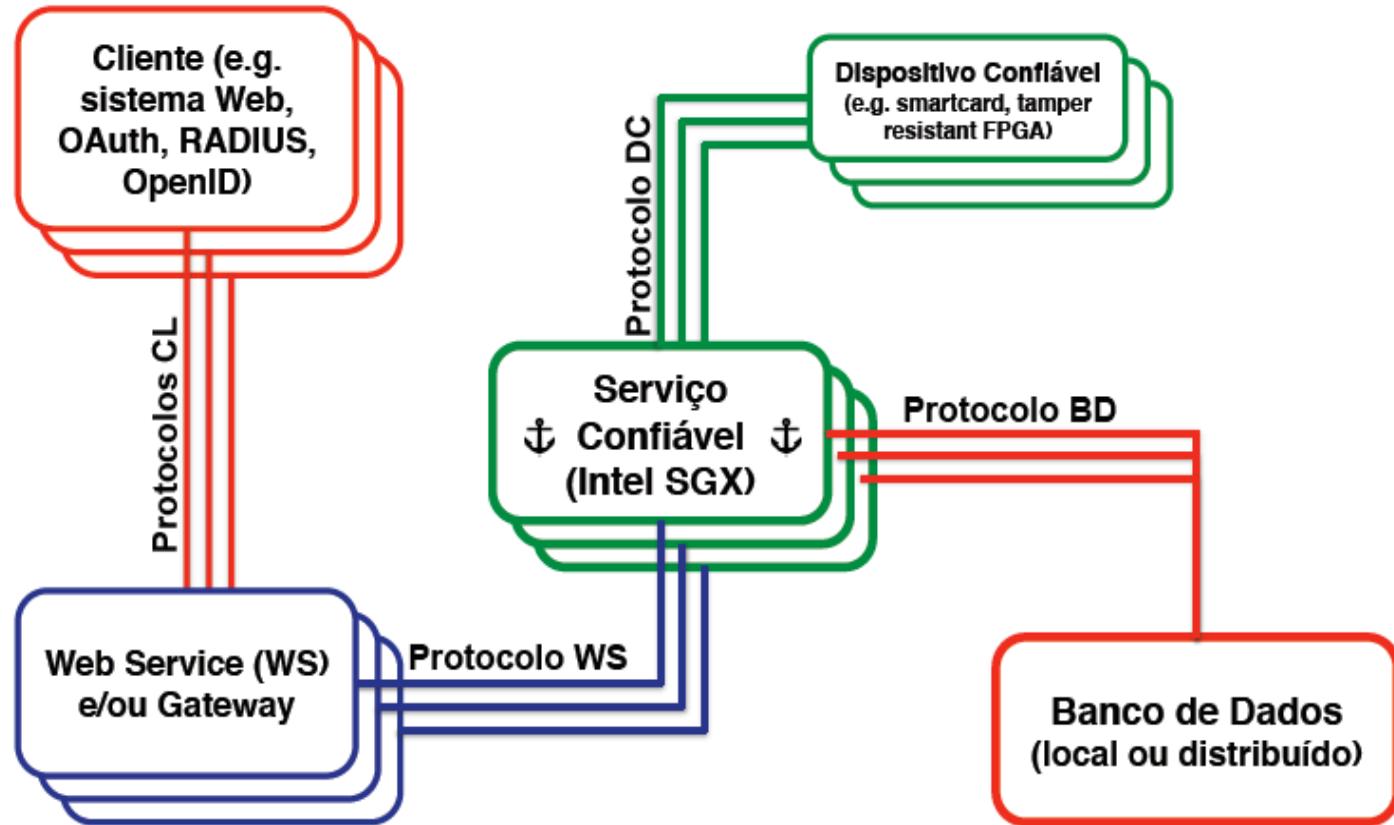
Implementação e Resultados

Considerações Finais

- Interoperabilidade
- Compatibilidade
- Integridade
- Confidencialidade
- Escalabilidade



Arquitetura SeguraAí



Roteiro

Intel SGX / OpenSGX

Arquitetura SeguraAí

Implementação e Resultados

Considerações Finais

Implementação

- Cliente/Servidor em Python/C para OpenSGX
 - Cliente = **Cliente + WS**
 - Servidor = **Serviço Confiável**
- **Protocolo WS** entre Cliente e Servidor
 - <operação,random, $E_k(login,senha)$, $HMAC_k$ >
 - operação = *REGISTRAR* ou *AUTENTICAR*
 - $K = K_{\text{registrar}} \text{ OU } K_{\text{autenticar}}$

Resultados

- Nativo, QEMU e OpenSGX (**Cliente** e **Servidor/Serviço**)

	Total	Médio	StdDev
Cliente nativo	0.248717	0.000124	0.000138
Cliente com QEMU	0.675755	0.000338	0.000144
Cliente com OpenSGX	210.875060	0.105595	0.014282
Autenticação nativa	0.087788	0.000043	0.000021
Autenticação com QEMU	0.467591	0.000234	0.000078
Autenticação com OpenSGX	209.038668	0.104623	0.014071

Resultados

- Tempo de execução OpenSGX = 447x QEMU

	Total	Média	Std Dev
Cliente nativo	0.248	0.000248	0.000018
Cliente com QEMU	0.675	0.000675	0.000044
Cliente com OpenSGX	210.87	0.0021087	0.000062
Autenticação nativa	0.087783	0.000087783	0.0000021
Autenticação com QEMU	0.467591	0.000467591	0.0000078
Autenticação com OpenSGX	209.038668	0.104623	0.014071

OpenSGX é muito mais lento que QEMU
(principal overhead é a **MEE em software**)

Resultados

- Tempo de autenticação = **105ms (0.105s)**

	Total		
Cliente nativo	0.248717	FreeRadius = 100ms [Kreutz et. al, 2014]	
Cliente com QEMU	0.675755	0.000338	0.000144
Cliente com OpenSGX	210.875060	0.105595	0.014282
Autenticação nativa	0.087788	0.000043	0.000021
Autenticação com QEMU	0.467591	0.000234	0.000078
Autenticação com OpenSGX	209.038668	0.104623	0.014071

Roteiro

Intel SGX / OpenSGX

Arquitetura SeguraAí

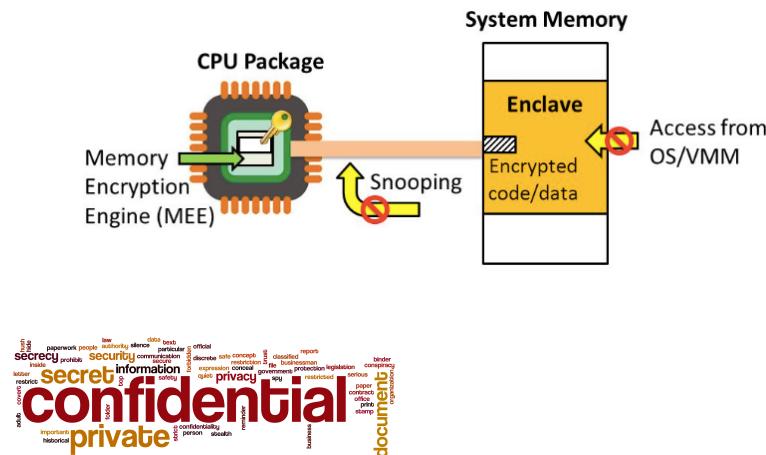
Implementação e Resultados

Considerações Finais

Considerações Finais

Garante integridade e confidencialidade para:

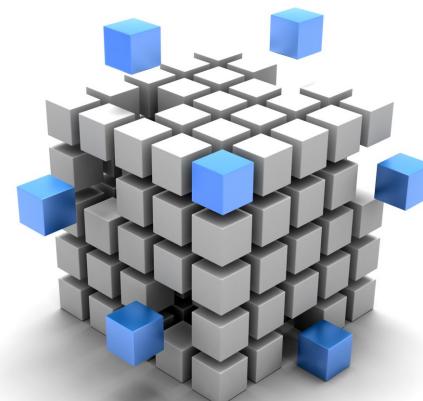
- Serviços de autenticação
 - Serviços de autorização
 - Dados pessoais
 - Dados financeiros
 - Etc.



Considerações Finais

A solução é **modular**, **escalável**, e promove a **interoperabilidade** e **compatibilidade** em:

- Sistemas de autenticação
- Sistemas de autorização
- Sistemas Web
- Outros tipos de sistemas



Trabalhos Futuros

- *Trade-offs* entre segurança e desempenho
- Sobrecarga em máquinas Intel SGX
- Protótipo completo do **SeguraAí**
- Estudo de viabilidade técnica e comercial da solução proposta



Obrigado!

Contatos:

autor1@email.com

autor2@email.com

autor3@email.com