The Security Challenges & Issues From SGX Practice

Xiaoning Li
Chief Security Architect
Alibaba Cloud

Agenda

- Secure Computing Introduction
- Intel® SGX Applications and Challenges
- Secure Computing Environment and Architecture Challenges

Secure Computing

- Providing data computation securely
- Data in encrypted mode beyond secure computing
- Secure computing is isolated and protected by hardware

For Data Security, Secure Computing Provides The Foundation!

Intel® SGX Applications and Challenges

Intel® SGX

- Intel CPU supports Intel® SGX from Skylake CPU
- Available on desktop and server machines
- Trusted execution environment in CPU
- New CPU instructions including both ring 0 and ring 3 instructions
- Intel provides software SDK
  - ECALL/OCALL
  - Enclave Definition Language(EDL)
  - Enclave Code

Security Challenges on Intel® SGX

- Traditional vulnerability/exploit issues in enclave
    - Compatible programing model with traditional vulnerabilities
    - Compatible with existing exploit techniques, such as ROP
- Side Channel Attacks
    - Cache/TLB
    - PageFault
    - Branch Target Buffer
- Secure SDK usages
- Denial of Service

Secure SDK usages

- Secure signing key protection
- Use enclave as release version
- Correct ECALL definition

SGX Signing Key Protection

- Utilizing HSM to protect signing key
- Self-protected signing key enclave

Use Enclave as Release Version

- Disable debug

```
<EnclaveConfiguration>
  <ProdID>0</ProdID>
  <ISVSVN>0</ISVSVN>
  <StackMaxSize>0x40000</StackMaxSize>
  <HeapMaxSize>0x100000</HeapMaxSize>
  <TCSNum>10</TCSNum>
  <TCSPolicy>1</TCSPolicy>
  <!-- Recommend changing 'DisableDebug' to 1 to make the enclave undebuggable for enclave release -->
  <DisableDebug>0</DisableDebug>
  <MiscSelect>0</MiscSelect>
  <MiscMask>0xFFFFFFFF</MiscMask>
</EnclaveConfiguration>
```

- Create enclave with debug mode

```
sgx_status_t sgx_create_enclave(
    const char *file_name,
    const int debug,
    sgx_launch_token_t *launch_token,
    int *launch_token_updated,
    sgx_enclave_id_t *enclave_id,
    sgx_misc_attribute_t *misc_attr
);
```

Correct ECALL Definition

Data Types					
char	short	int	float	double	void
int8_t	int16_t	int32_t	int64_t	size_t	wchar_t
uint8_t	uint16_t	uint32_t	uint64_t	unsigned	struct
union	enum	long			
Pointer Parameter Handling					
in	out	user_check	count	size	readonly
isptr	sizefunc	string	wstring		
Others					
enclave	from	import	trusted	untrusted	include
public	allow	isary	const	propagate_errno	
Function Calling Convention					
cdecl	stdcall	fastcall	dllimport		

Dangerous Pointer Parameter Handling

Pointer Parameter Handling					
in	out	user_check	count	size	readonly
isptr	sizefunc	string	wstring		

Example in untrusted code

```
void ecall_test_functions(void)
{
    int ret = 0;
    char str1[10];
    char str2[10];

    strncpy(str1,"1234",4);
    strncpy(str2,"4321",4);

    printf("before:str1:%s\nstr2:%s\n",str1,str2);
    ret = ecall_sgx_test(global_eid, str1,str2);
    if (ret != SGX_SUCCESS)
        abort();
    printf("after:str1:%s\nstr2:%s\n",str1,str2);


}
```

Example in EDL file

```
enclave {



    trusted {
        public void ecall_sgx_test([user_check] char* str1, [user_check] char* str2);
    };
};
```

Example in SDK code

```
typedef struct ms_ecall_sgx_test_t {
        char* ms_str1;
        char* ms_str2;
} ms_ecall_sgx_test_t;
```

```
#define CHECK_REF_POINTER(ptr, siz) do {          \
        if (!(ptr) || ! sgx_is_outside_enclave((ptr), (siz)))    \
                return SGX_ERROR_INVALID_PARAMETER;\
} while (0)
```

```
static sgx_status_t SGX_CDECL sgx_ecall_sgx_test(void* pms)
{
        ms_ecall_sgx_test_t* ms = SGX_CAST(ms_ecall_sgx_test_t*, pms);
        sgx_status_t status = SGX_SUCCESS;
        char* _tmp_str1 = ms->ms_str1;
        char* _tmp_str2 = ms->ms_str2;

        CHECK_REF_POINTER(pms, sizeof(ms_ecall_sgx_test_t));

        ecall_sgx_test(_tmp_str1, _tmp_str2);


        return status;
}
```

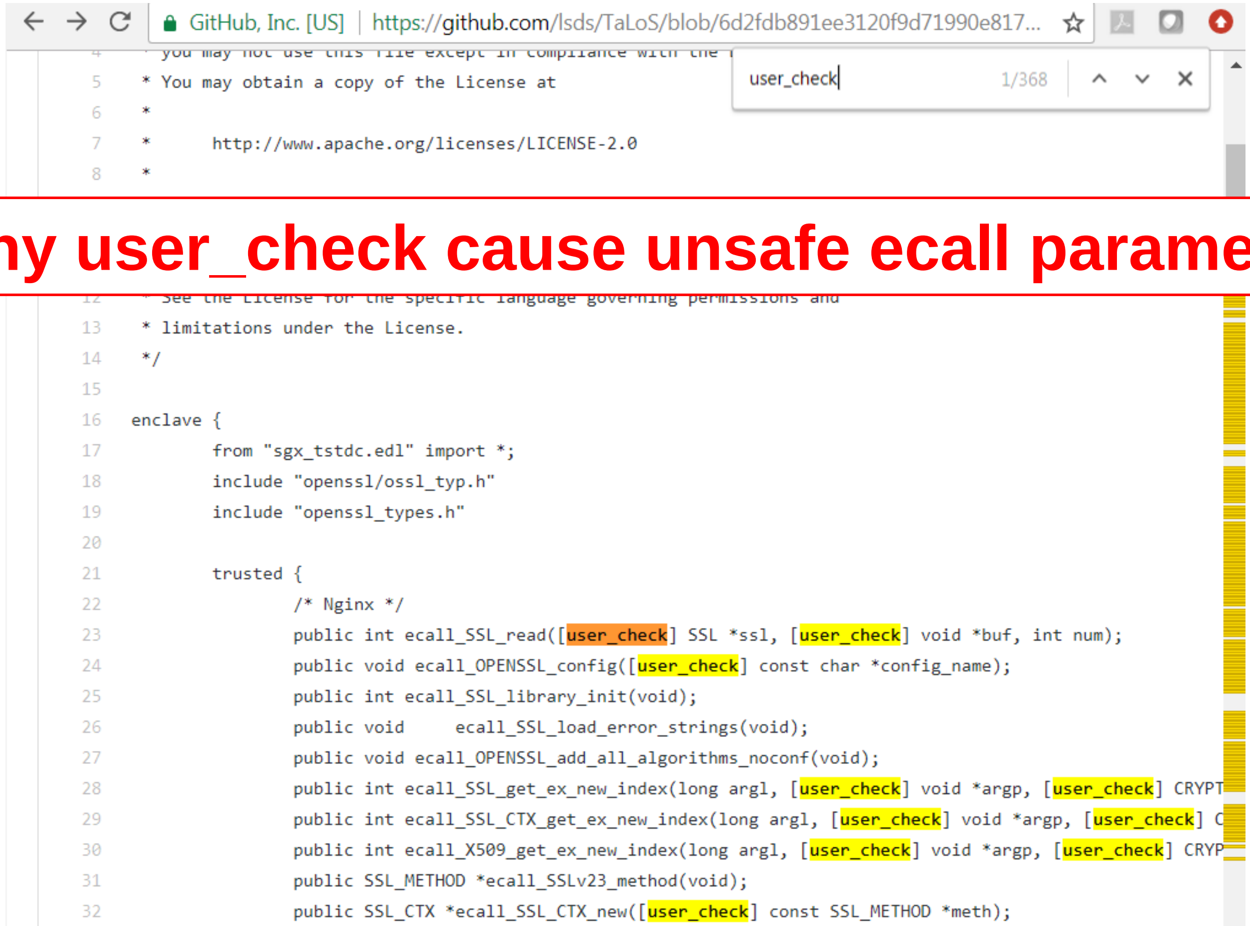
Example in trusted code

```
void ecall_sgx_test(char* str1, char* str2)
{
    strncpy(str2,str1,4);

}
```

Str1 could be in enclave range without boundary checking

Real world Cases - TaLos




```
 * you may not use this file except in compliance with the L
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

enclave {
        from "sgx_tstdc.edl" import *;
        include "openssl/ossl_typ.h"
        include "openssl_types.h"

        trusted {
                /* Nginx */
                public int ecall_SSL_read([user_check] SSL *ssl, [user_check] void *buf, int num);
                public void ecall_OPENSSL_config([user_check] const char *config_name);
                public int ecall_SSL_library_init(void);
                public void     ecall_SSL_load_error_strings(void);
                public void ecall_OPENSSL_add_all_algorithms_noconf(void);
                public int ecall_SSL_get_ex_new_index(long argl, [user_check] void *argp, [user_check] CRYPT
                public int ecall_SSL_CTX_get_ex_new_index(long argl, [user_check] void *argp, [user_check] C
                public int ecall_X509_get_ex_new_index(long argl, [user_check] void *argp, [user_check] CRYP
                public SSL_METHOD *ecall_SSLv23_method(void);
                public SSL_CTX *ecall_SSL_CTX_new([user_check] const SSL_METHOD *meth);
```

Many user_check cause unsafe ecall parameters

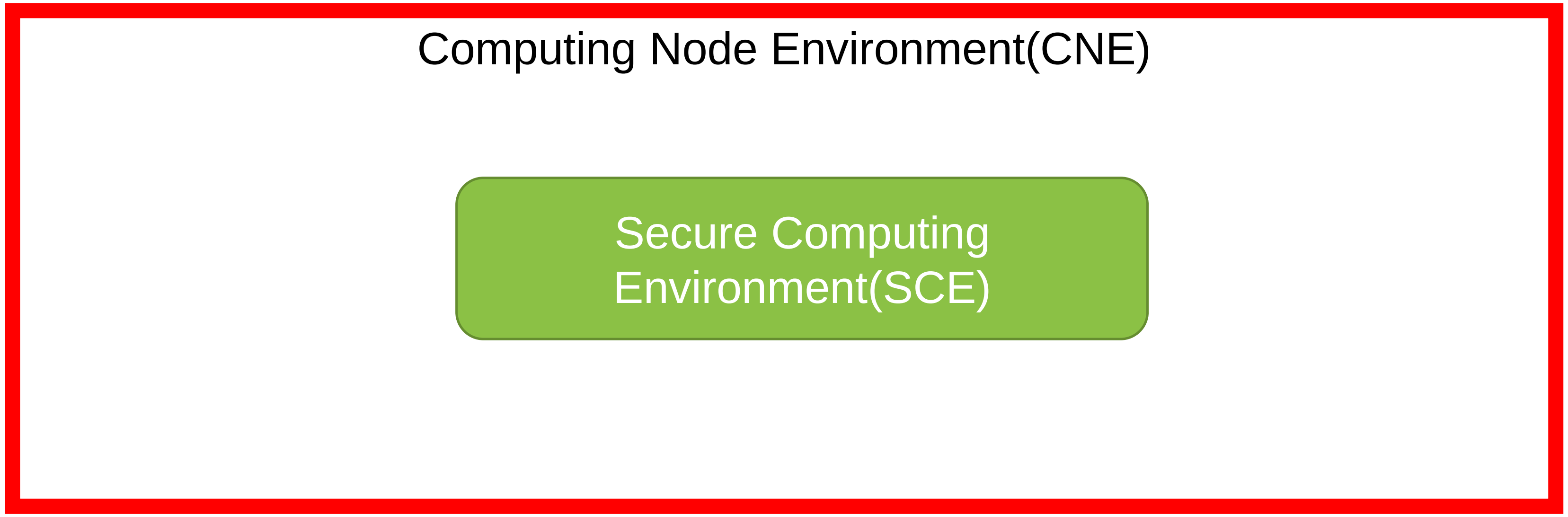
https://github.com/lsds/TaLoS/blob/6d2fdb891ee3120f9d71990e817fc7794317b903/src/talos/enclaveshim/enclave.edl

Denial of Service

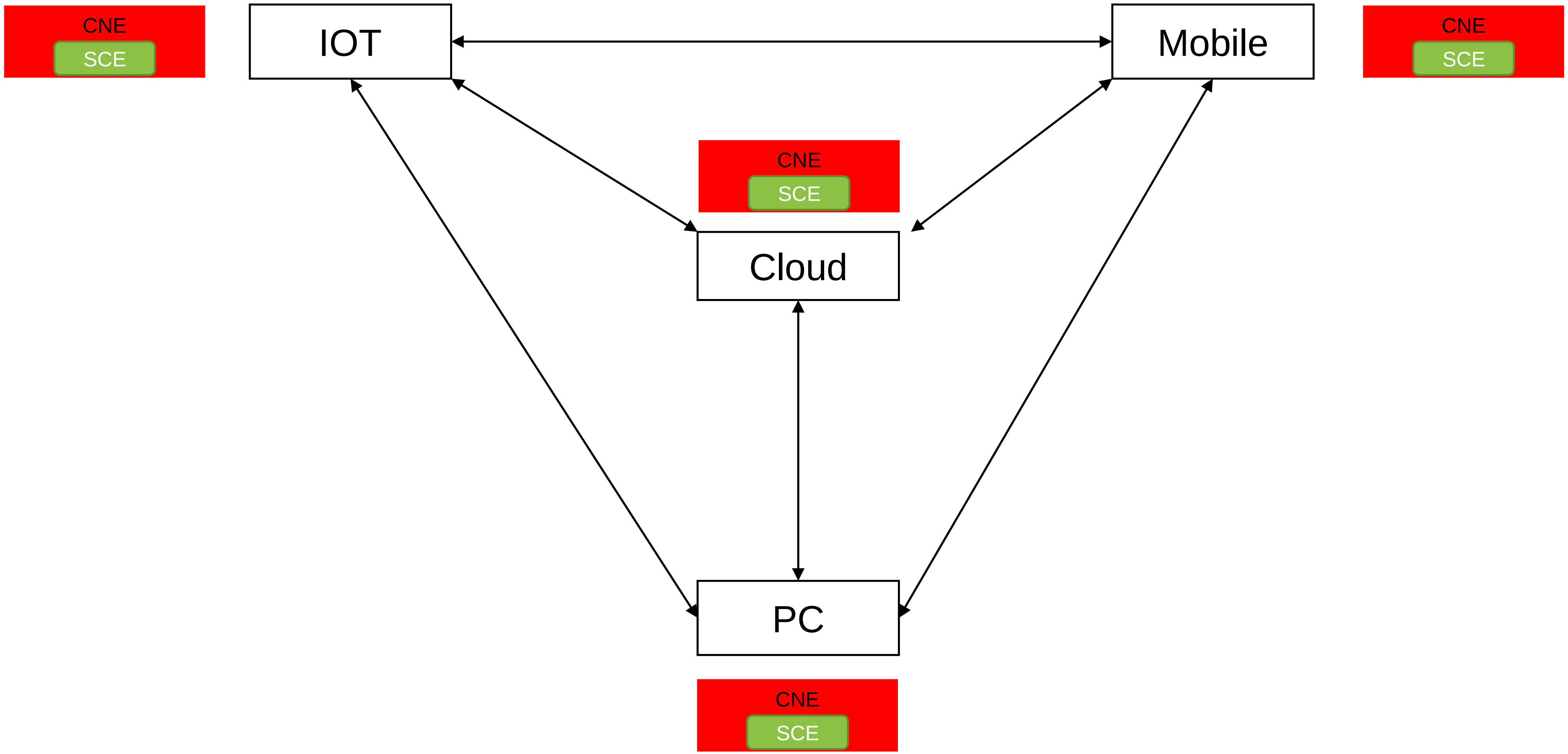
- SGX disabled
- Limited EPC memory
- Shared EPC cross debug and release enclave

Secure Computing Environment

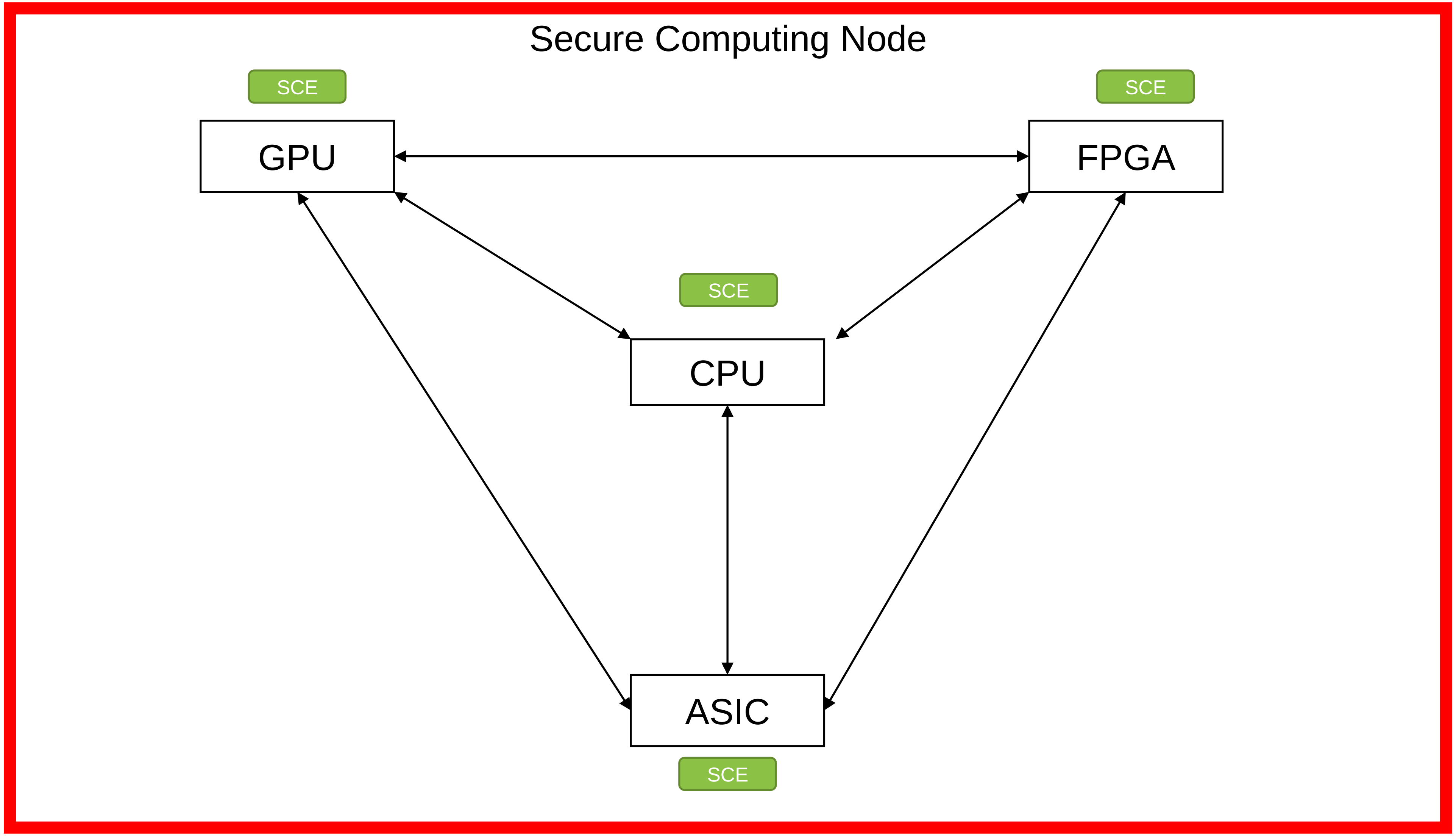
Secure Computing Framework



Secure Computing Framework

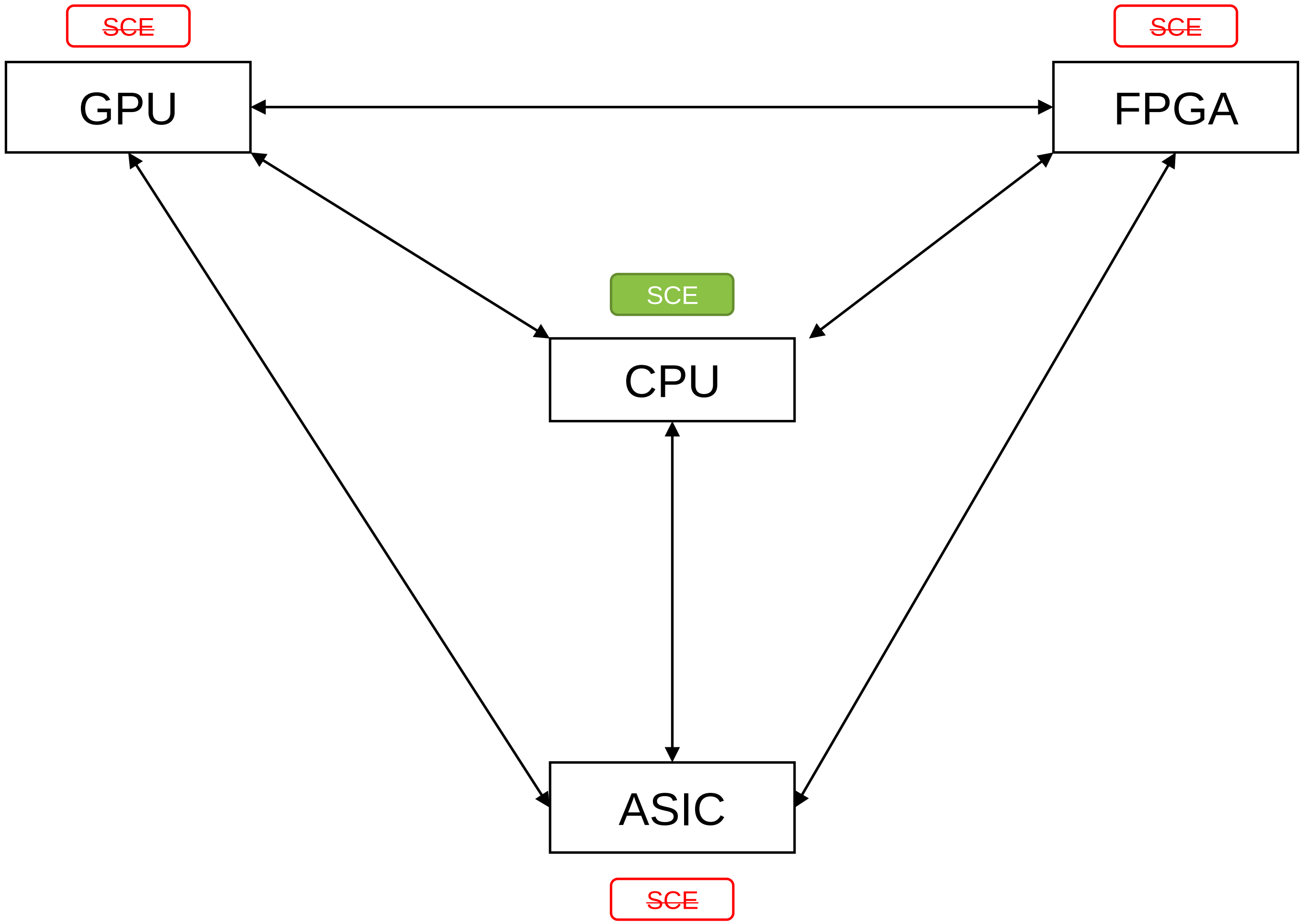


Secure Computing Framework



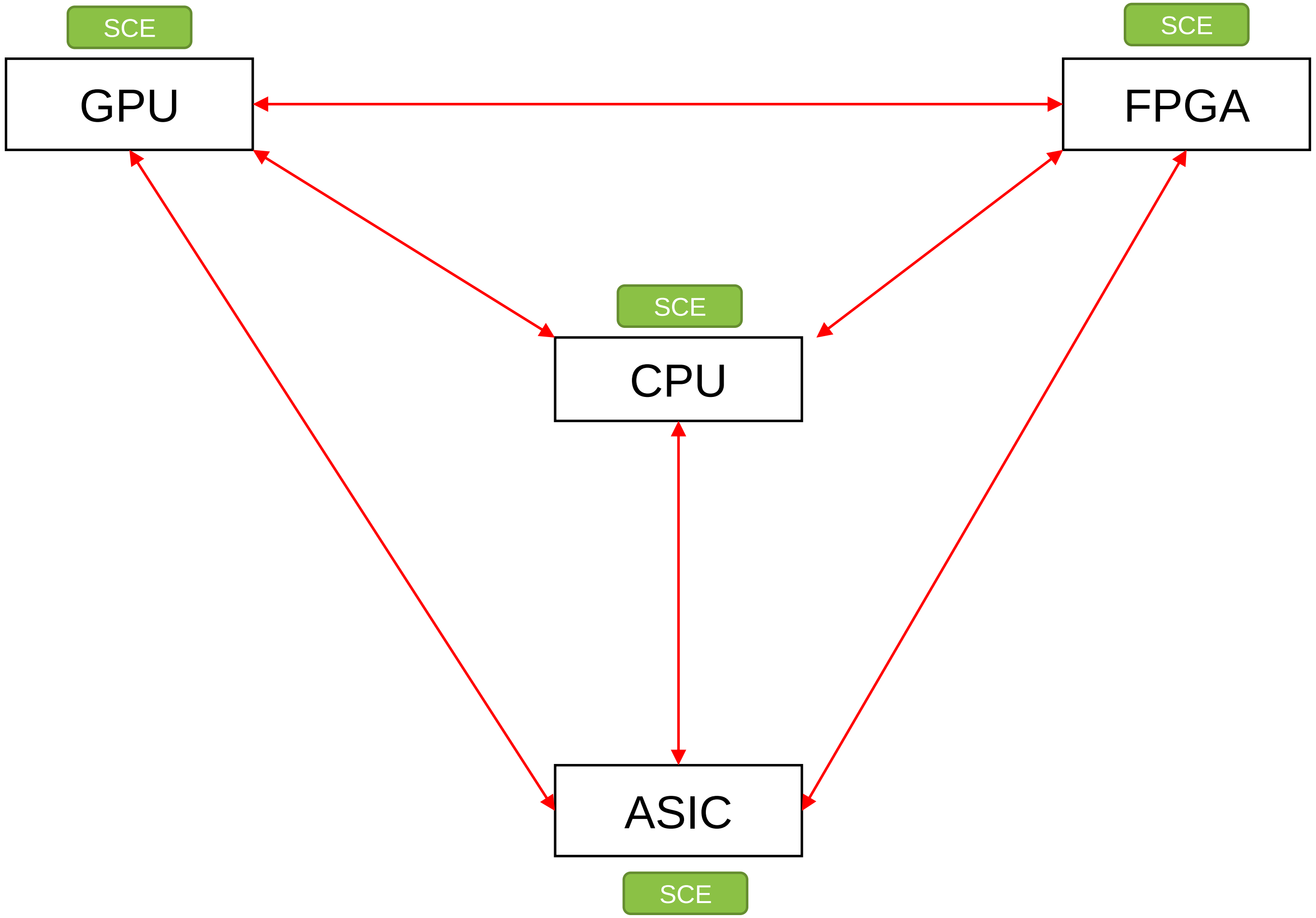
Architecture Challenges

Architecture Challenges 1



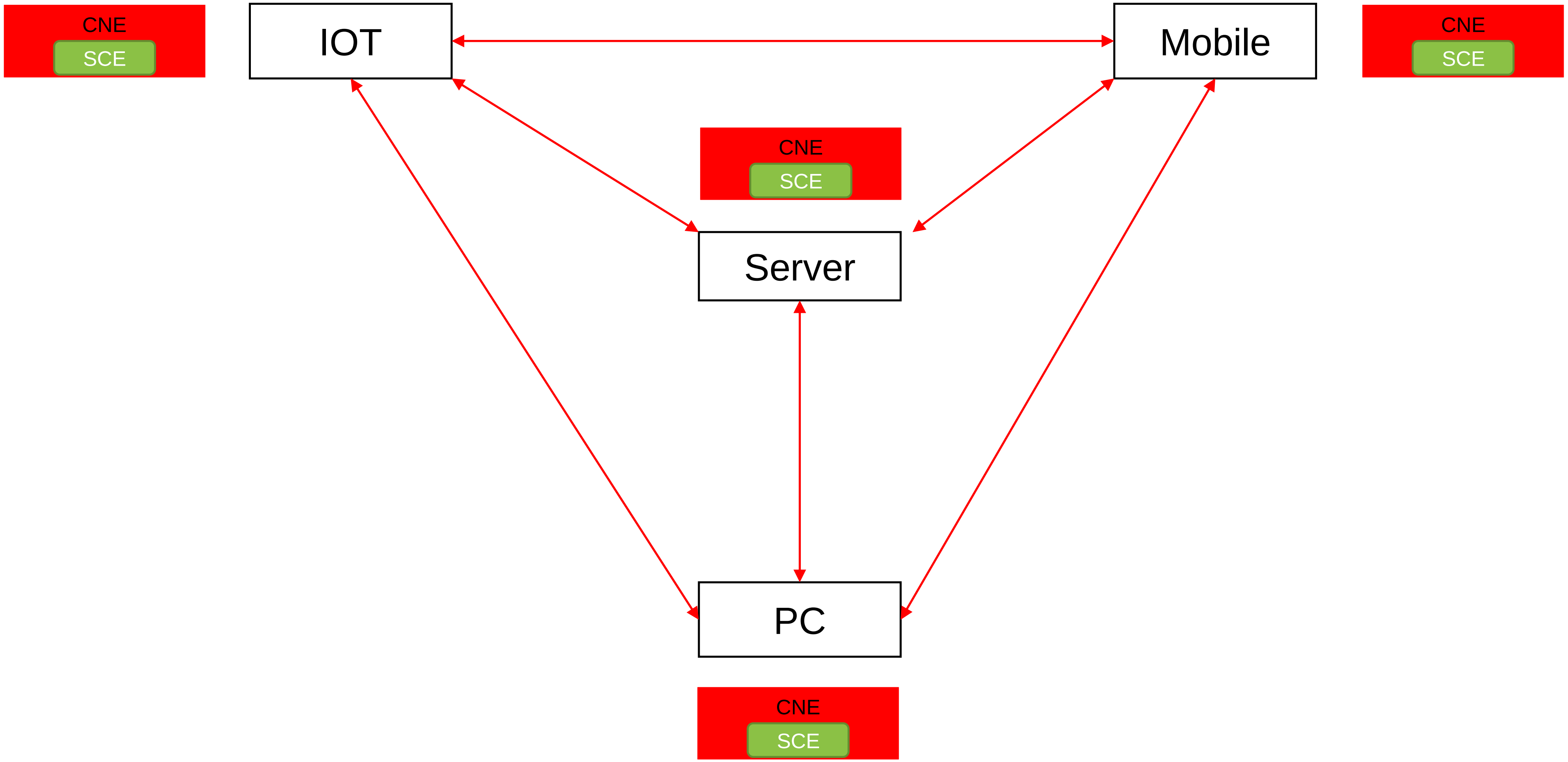
No Existing Secure Computing Environment in GPU/FPGA/ASIC!

Architecture Challenges 2



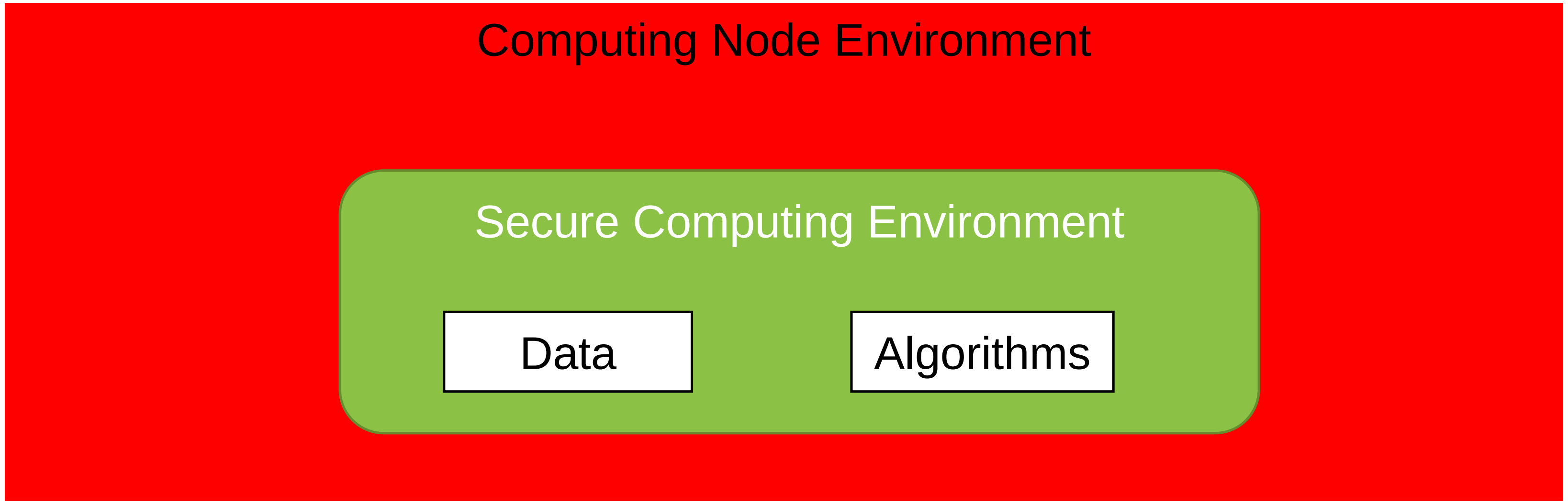
No General Attestation Capability Cross Secure Computing Environments

Architecture Challenges 3



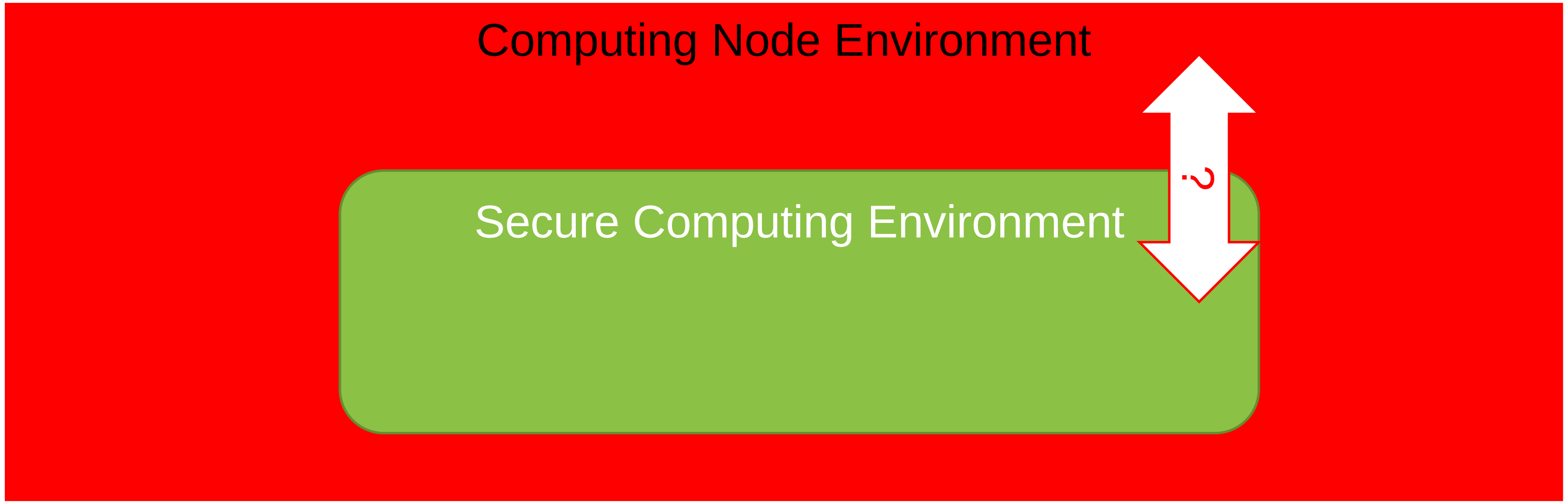
No General Attestation Capability Cross Secure Computing Nodes

Architecture Challenges 4



Secure Computing Algorithms Cloud Be Vulnerable

Architecture Challenges 5



Proving secure computing environment is secure as expected

Summary

- Intel® SGX provides the foundation for secure computing in CPU
- Intel® SGX applications should be implemented correctly to avoid potential attack vectors
- Secure computing has big architecture gap if we want to apply it cross computing devices/nodes

Reference

- [1] Intel® Software Guard Extensions (Intel® SGX), https://software.intel.com/en-us/sgx
- [2] AI and Security Keynotes, Dawn Song, Microsoft Research Faculty Summit 2017
- [3] Stacco: Differentially Analyzing Side-Channel Traces for Detecting SSL/TLS Vulnerabilities in Secure Enclaves , Yuan Xiao, Mengyuan Li, Sanchuan Chen, Yinqian Zhang,CCS2017
- [4] Leaky Cauldron on the Dark Land: Understanding Memory Side-Channel Hazards in SGX, Wenhao Wang, Guoxing Chen, Xiaorui Pan, Yinqian Zhang, XiaoFeng Wang, Vincent Bindschaedler, Haixu Tang, Carl A. Gunter,CCS2017