



This work is distributed
under the CC BY-SA license,
unless stated otherwise.

Hardware Security

5. HSMs, TEEs

Ján Jančár

Summer 2026

Today

- HSMs
- TEEs
- Applications and attacks

More acronyms, yay! No more hardware acronyms after this one

Assignment 1: 2FIDO or not 2FIDO?

- Out now
- Questions?

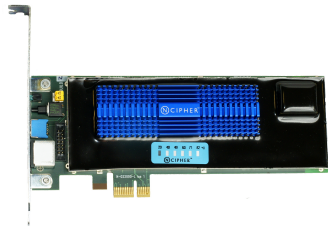
Hardware Security Modules

a big black box

HSM

Hardware Security Module

- Separate physical device
 - *“A beefy smartcard for the datacenter”*
- Fixed API, usually not programmable
- PCI or networked connection to a **host**
- One or more crypto-processors
 - Performs cryptographic operations
 - Generates keys
 - Keys never leave the device
 - Access control
- Anti-tampering measures
- **Trusted path**
 - Channel between a user and the HSM
 - Physically separate, can not be sniffed by the host

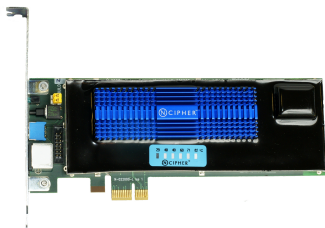


https://commons.wikimedia.org/wiki/File:NCipher_nShield_F3_Hardware_Security_Module.jpg

HSM

Devices

- Gemalto, Thales, Utimaco, SafeNet, nCipher, ...
- YubiHSM, Nitrokey HSM (small form factor)
- Also **smartcard HSM**
 - Not *really* a true HSM
- **SoftHSM**
 - A simulator



https://commons.wikimedia.org/wiki/File:NCipher_nShield_F3_Hardware_Security_Module.jpg



<https://www.yubico.com/ch/product/yubihsm-2/>



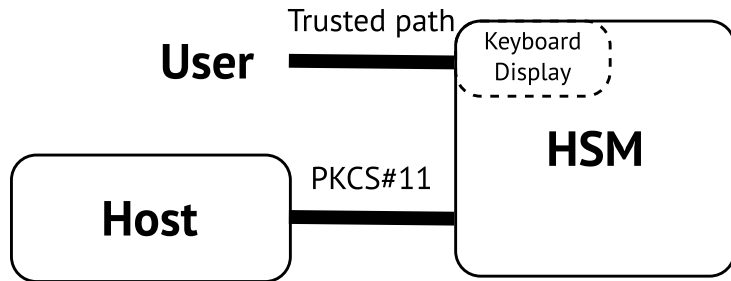
<https://shop.nitrokey.com/shop/nkhs2-nitrokey-hsm-2-7>

- 👁 Tamper evidence
 - Evidence is left when tampering occurs
 - Chemical (coatings) or mechanical (stickers)
- 🛡 Tamper resistance
 - It is hard to tamper with (and not destroy the whole thing)
 - Chemically resistant material, epoxy, shielding
- 🔍 Tamper detection
 - Tampering is detected by the device
 - Sensors (light, temperature), meshes, monitors (voltage)
- 🔊 Tamper response
 - ❓ You detected tampering, what do you do?
 - Maybe just log/warn? But who and how?
 - Destroy all sensitive data / [Lockdown](#)
 - [Burn it all button](#)

- **PKCS#11** (Cryptoki) (Public-Key Cryptography Standard #11)
 - Block cipher operations (encrypt/decrypt)
 - Key generation and wrapping
 - Digital signatures (RSA, ECDSA)
 - Session management (open/close)
 - Object management (find objects, get attributes)
- **KMIP** (Key Management Interoperability Protocol)
 - Key lifecycle management
 - Server-client architecture
- Vendor-specific APIs
 - IBM CCA (Banking)
- Some cloud-vendor-specific APIs
 - AWS: OpenSSL engine/JCE provider

HSM

Setup



- Slots and tokens
 - Originally developed with smartcards in mind
 - Slot is an empty space
 - Token can be installed in a slot
 - Card reader is a slot, a smartcard is a token
 - A HSM has multiple logical slots with tokens
 - ▶ Separation between slots
- Objects
 - Keys, Certificates, Data
 - Have handles
 - Have attributes
 - Live on a token
- Roles
 - User: User of the token
 - ▶ Has a PIN
 - Security Officer (SO): Admin of the token
 - ▶ Has a PIN
 - ▶ Can init/reset the token
- PINs
 - May not be **just** PINs
 - Longer secrets, can use secret sharing
- Mechanisms
 - Cryptographic algorithm + use
 - RSA_PKCS_OAEP is RSA encryption/decryption

PKCS#11 is a non-enforcing standard, you are free to shoot your own foot with it.

- APIs started simple and small to prevent bugs
 - Feature creep
 - Now quite complex and bug prone
- Logical attacks (protocol/API attacks)
 - API-only
 - Manipulate features & allowed command sequences
 - **Wrap-Decrypt** attack
 - ▶ Assumes attacker has host control
 - ▶ Sensitive key (CKA_SENSITIVE = True, CKA_EXTRACTABLE = True)
 - ▶ Storage key (CKA_WRAP = True, CKA_DECRYPT = True)
 - ▶ Attacker exports sensitive key wrapped under storage key
 - ▶ Attacker requests decryption of “data” using the storage key
 - ▶ HSM does not know “data” is the sensitive wrapped key

- Implementation bugs
 - Sometimes HSMs actually run Linux
 - [Hackable Security Modules: Reversing and Exploiting a FIPS 140-2 lvl 3 HSM firmware](#)
 - ▶ Reverse-engineered custom HSM firmware
 - [Everybody Be Cool, This Is A Robbery](#)
 - ▶ Used fuzzing through PKCS#11
 - ▶ Stack & heap overflows, memory corruption bugs

- PKI
 - Certificate Authorities (CA)
 - Sign certificates
- Banking
 - Verify PINs
 - Validate transaction cryptograms
- TLS termination
 - Serve as the endpoint for SSL/TLS
- DNSSEC
 - Sign zone files (DNS assignments)
 - Recent [issue](#) at DENIC caused an outage of .de for a few hours
 - ▶ Cloudflare temporarily suspended DNSSEC validation for .de

- HSMs promise no key leaks
- Yet, keys have to be used by the host
- ? What do they actually bring security-wise?
 - Host compromise: key extraction → key usage
 - Key usage can be
 - ▶ restricted
 - ▶ rate-limited
 - ▶ additionally authorized
 - ▶ logged
 - Attacker needs to maintain host compromise

HSMs are not magic

HSM

Comparison to TPMs

HSMs

- Expensive (1 000\$ – 30 000\$)
- Big & external
- High performance
- PKCS#11
- Multi-tenant
- Secures the *system*

TPMs

- Cheap (5\$ – 10\$)
- Small & internal
- Low performance
- TSS
- Single-tenant
- Secures the device

Trusted Execution Environment

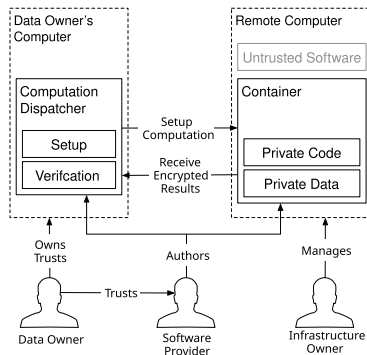
and now for something completely different

TEE

Trusted Execution Environment

Oddly, despite the abundant research on TEEs and the growing number of commercially available TEE solutions, there is no single, widely-accepted definition for a TEE. – Trusted Execution Environments

- How do we safely process high-risk data in cloud-based environments?
- How do we securely store payment credentials on a rooted mobile device?
- Can we prevent malicious smartphone applications from stealing users' fingerprint data?
- Can we protect secrets from a rogue system administrator with extensive access to a given platform?



<https://eprint.iacr.org/2016/086.pdf>

Like a secure VM no one can touch but you, not even the person that owns the CPU

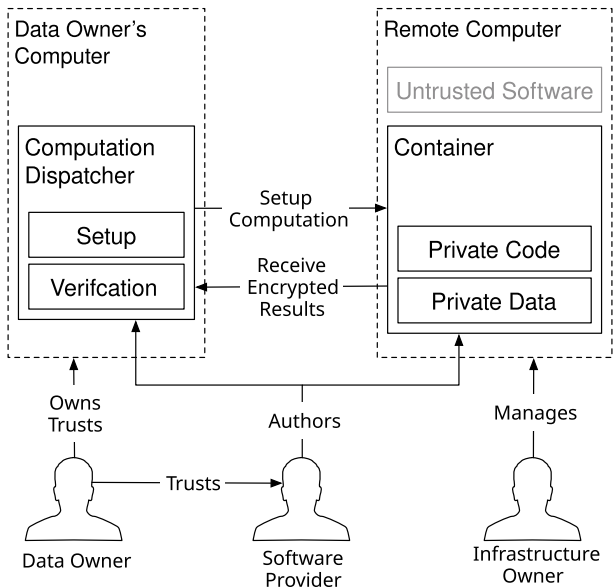
- Confidential computing
- Secure environment within a general-purpose CPU
 - Executes general purpose code (not only cryptography)
- Runs *enclaves*
 - Trust Domains, Trustlets, Secure Encrypted VMs, Realms, ...
- Protected even from the host kernel
 - Sometimes also attackers with physical access (memory reading)
 - But often physical side-channel attacks out-of-scope
- Core features
 - Data confidentiality and integrity during computation
 - Code integrity
 - Remote attestation

- Intel [SGX](#), Intel [TDX](#)
- AMD [SEV-SNP](#)
- ARM [TrustZone](#) (not a full TEE)
 - Used to build full TEEs
 - Splits into “*normal world*” and “*secure world*”
- ARM [Realms](#)
- Many more, both commercial and academic
 - [OP-TEE](#), [Google Trusty](#), [Qualcomm TEE](#)
- Kind of: Apple [Secure Enclave](#)
 - Separate co-processor
 - Only programmable by Apple

- User, Provider, Cloud
- User
 - Has sensitive data
 - Wants some computation done on the data by the provider
 - Fears data misuse by the provider or infrastructure
- Provider
 - Offers a service computing on sensitive data
 - Does not want to leak their proprietary reference data
 - Distrusts the infrastructure
- Cloud
 - Provides the hardware resources
 - Should not learn anything

TEE

Motivation



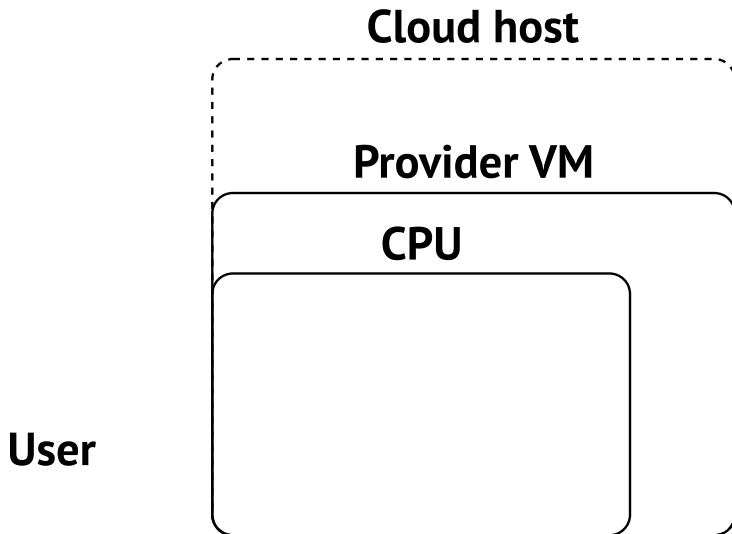
Avoiding the 23andMe data leak

DNA Analysis service

- User submits DNA records
- Provider computes on them and outputs diagnosis
- Goals
 - Provider only learns output diagnosis (maybe not even)
 - User learns output diagnosis (no proprietary data leak)
 - Cloud learns nothing
- ? How do we do this?
 - Cryptographic solution: **FHE** (Fully Homomorphic Encryption) or **MPC** (Multi Party Computation)
 - ▶ Computation on encrypted data, or
 - ▶ Computation split between multiple parties
 - Hardware solution: **TEE**
- ? So how would you build this TEE?

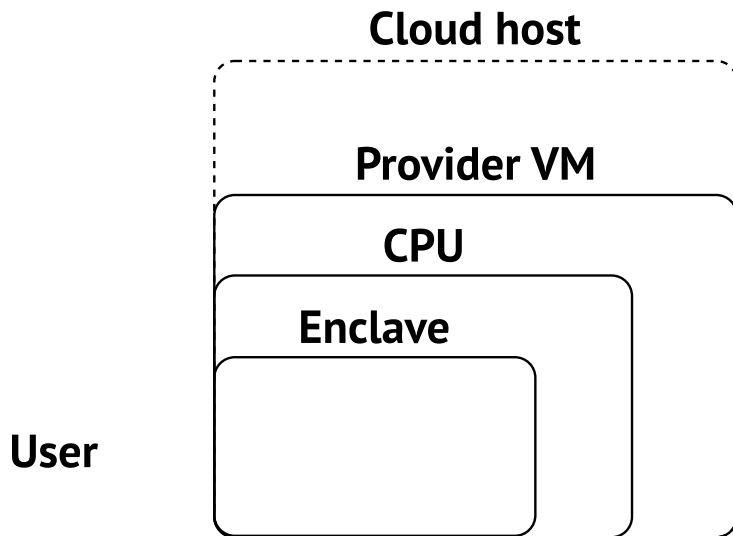
TEE

Let's build it (1)



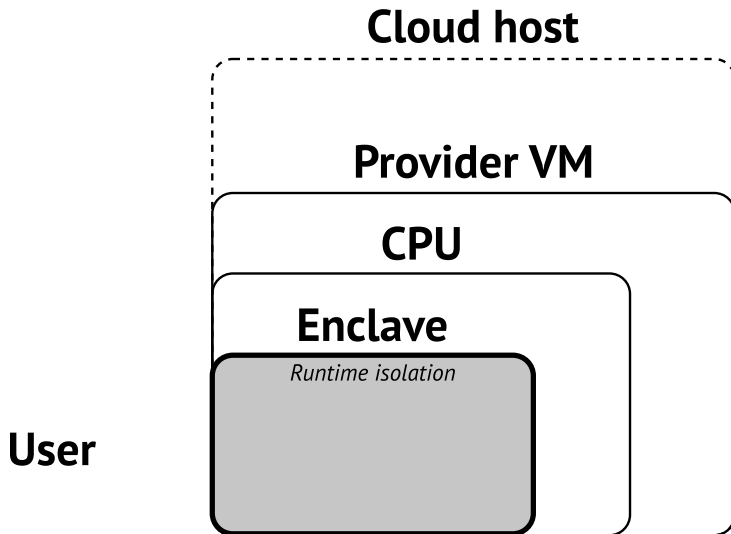
TEE

Let's build it (2)



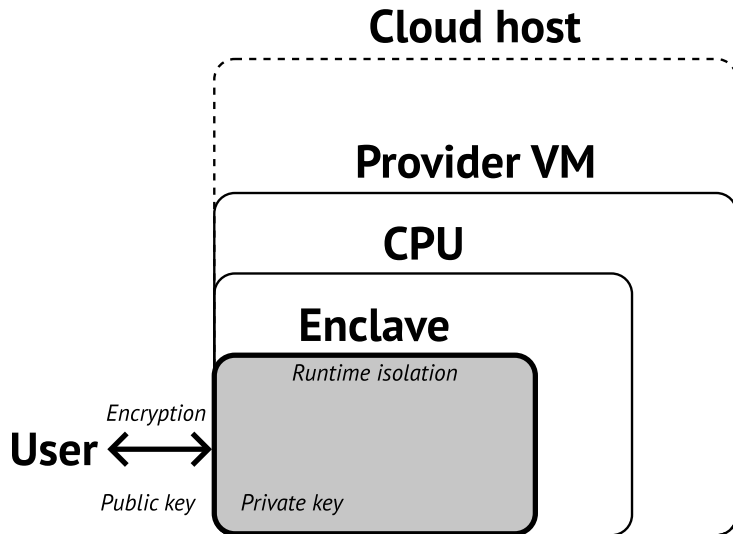
TEE

Let's build it (3)



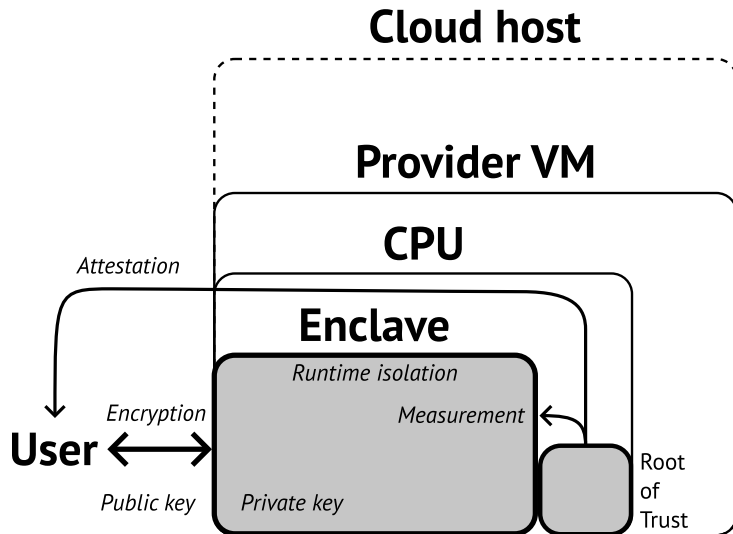
TEE

Let's build it (4)



TEE

Let's build it (5)



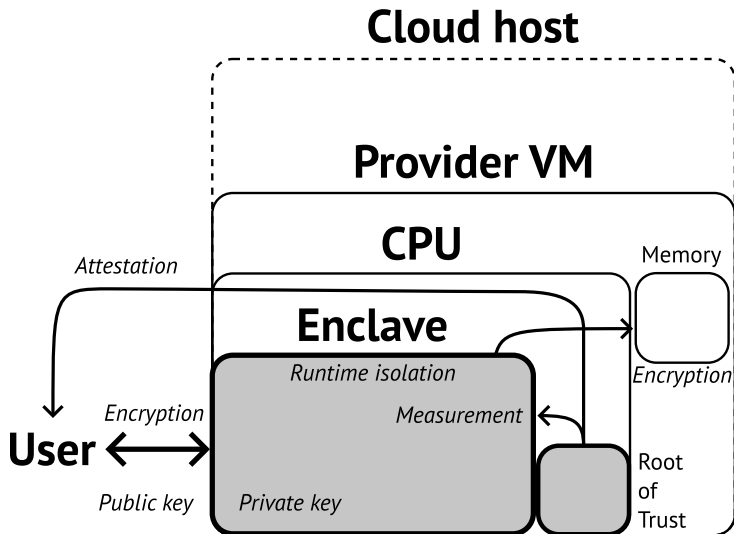
Intermezzo

Reproducible builds

- Provider releases code publicly
 - Not going to happen with proprietary code
- Allows audits by users (or third parties)
- **Deterministic** build process
 - Users can replicate the exact same hashes

TEE

Let's build it (6)



Servers

- The same model (User, Provider, Cloud)
- Confidential Computing Consortium
- Intel SGX, Intel TDX, AMD SEV-SNP, ...
- *How do we safely process high-risk data in cloud-based environments?*
- *Can we protect secrets from a rogue system administrator with extensive access to a given platform?*

Mobile

- The model is inverted
 - Cloud: Your phone
 - User = Provider: Wants to compute something on your phone
 - Your bank wants to securely store payment credentials in your phone
- GlobalPlatform TEE
- ARM TrustZone-based, OP-TEE, QTEE, Google Trusty, ...
- *How do we securely store payment credentials on a rooted mobile device?*
- *Can we prevent malicious smartphone applications from stealing users' fingerprint data?*

Servers

- Signal: SGX for [Private Contact Discovery](#)
 - *Here are all my contacts, which of them are Signal users?*
- Facebook: TEE for [Advanced Browsing Protection](#)
 - *How do you detect malicious links in end-to-end encrypted chat?*

Mobile

- Android: Vendor's TEE for [Hardware-backed Keystore](#)
 - Secure key storage and usage
- Android: Vendor's TEE for [DRM](#)
 - Protecting access to copyrighted content
- Android: Vendor's TEE for [Biometric storage](#)

- Intel Software Guard Extensions
- Now deprecated, still relevant
- Enclave has private memory
 - CPU prevents access outside enclave
 - Encrypted in memory, decrypted inside CPU
 - Single system-wide key, generated at boot
- MAC for memory integrity, complicated structure

- Intel Trust Domain Extensions
- Like SGX, but VMs not enclaves
- Memory encryption, individual keys
- MAC for memory integrity

- Splits into “*normal world*” and “*secure world*”
 - Needs more infrastructure to be a TEE: Secure OS, application management
- No memory encryption

- Secure Encrypted Virtualization - Secure Nested Paging
- Memory encryption, individual keys

- Code running may have bugs
 - TEEs will not help you if you mess up
- Usually fail miserably against physical attackers
 - General-purpose CPUs not designed against side-channel attacks
 - Also timing, cache and micro-architectural attacks
- Troubled story with micro-architectural attacks
 - General-purpose CPUs have them, so TEEs inherit them

Attack	Developer	Intel	Leakage	Mitigation
Branch predictors [45, 55, 80]	X	X	Branches (code)	Constant-time code
Caches [24, 34, 42, 51, 90, 107]	X	X	64B accesses (code + data)	Constant-time code
Memory dependencies [91]	X	X	4B accesses (code + data)	Constant-time code
Port contention [13]	X	X	μ-ops (code)	Constant-time code
Page faults [136]	X	X	4K accesses (code + data)	Constant-time code
A/D bit monitoring [119]	X	X	4K accesses (code + data)	Constant-time code
DRAM channel [131]	X	X	1K - 8K accesses (code + data)	Constant-time code
FLUSH +RELOAD on PTEs [119]	X	X	4K accesses (code + data)	Constant-time code
IA32 segmentation faults [50]	X	X	1B accesses (code + data)*	Mitigated
Interrupts [80, 90, 118, 121]	X	X	Instructions	Constant-time code
CopyCat [92]	X	X	Instructions	Constant-time code
MicroScope [114]	X	X	Instructions	Constant-time code
ROP gadgets [20, 79]	X	X	Gadget dependent [†]	Memory safety
SmashEx [41]	X	X	Gadget dependent [†]	Memory safety
Synchronization [128, 132]	X	X	Gadget dependent [†]	Thread safety
SgxPectre [32]	X	X	Gadget dependent [†]	lfence & retpoline
Load Value Injection [102, 122]	X	X	Gadget dependent [†]	lfence
Foreshadow [120]	X	X	Enclave memory, CPU registers	No Hyper-Threading
SA-00219 [63]	X	X	Enclave memory [‡] , CPU registers	No iGPU
MDS [31, 108, 123]	X	X	In-flight loads/stores, vector registers	No Hyper-Threading
CacheOut [127]	X	X	Enclave memory, CPU registers	
Crosstalk [103]	X	X	MSRs, egetkey, rdrand	
MMIO Stale Data [68]	X	X	MSRs, egetkey, rdrand	No Hyper-Threading
ÆPIC Leak [12, 21]	X	X	Enclave memory [§] , CPU registers	No Hyper-Threading
PlunderVolt/VOLTpwn [74, 94]	X	X	AES-NI keys	No voltage scaling MSRs
PLATYPUS [86]	X	X	AES-NI keys, control flow, etc.	No RAPL

sgx.fail

TEEs have very ambitious goals, but generally do not deliver under attacks.

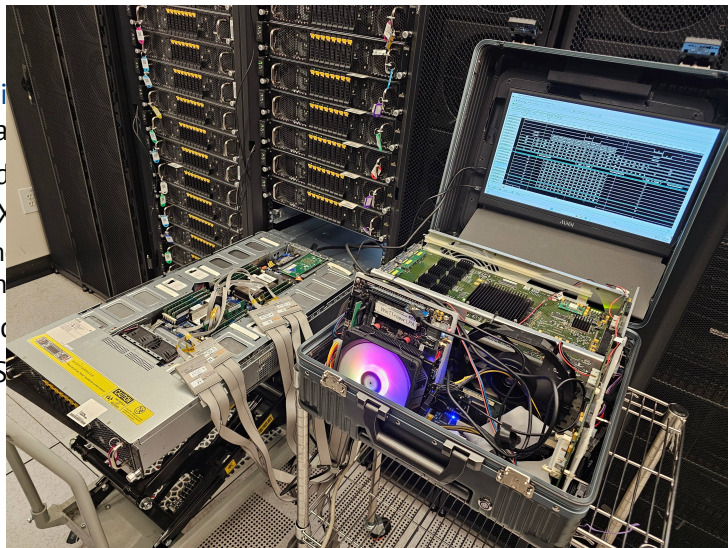
- [TEE.fail: Breaking Trusted Execution Environments via DDR5 Memory Bus Interposition](#); Jalen Chuang, Alex Seto, Nicolas Berrios, Stephan van Schaik, Christina Garman, Daniel Genkin
- Captured RAM Bus traffic using a logic analyzer
- Intel SGX & TDX encryption is **deterministic** (AES-XTS)
 - Same plaintext + address → same ciphertext
 - Ciphertext equality oracle
- Extracted attestation private key
- Forged SGX/TDX attestations

- TEE.fail: Breaking Trusted Execution Environments via DDR5 Memory Bus Interposition; Jalen Chuang, Alex Seto, Nicolas Berrios, Stephan van Schaik, Christina Garman, Daniel Genkin
- Captured RAM Bus traffic using a logic analyzer
- Intel SGX & TDX encryption is **deterministic** (AES-XTS)
 - Same plaintext + address $\rightarrow$ same ciphertext
 - Ciphertext equality oracle
- Extracted attestation private key
- Forge



- TEE.fail: Interposition
- Christina
- Captured
- Intel SGX
 - Sam
 - Ciph
- Extracted
- Forged S

ry Bus
Schaik,



Takeaways

- HSMs secure keys and control their usage
 - but they do not prevent host compromise
- TEEs offer a very strong capability: confidential computation
 - but they often fail in presence of physical/microarchitectural attacks
- Attestation without knowing what is behind the hashes has little value
 - proves code is running in a genuine TEE
 - proves provider put it there
 - does not prove the code is not malicious/wrong

Recommended reading

- For **this** lecture
 - Read [SoK: Hardware-supported Trusted Execution Environments](#)
 - ▶ A very thorough overview of TEEs
- For the **next** lecture
 - Read Embedded Cryptography 1 - Chapter 1 - Timing Attacks - Section 1.2. Example attacks
 - ▶ Nice examples of timing attacks on cryptography

References

HSM

- [Everybody Be Cool, This Is A Robbery](#)
- [Hackable Security Modules: Reversing and Exploiting a FIPS 140-2 lvl 3 HSM firmware](#)
- [YubiSecure? Formal Security Analysis Results for the Yubikey and YubiHSM](#)

TEE

- [Trusted Execution Environments](#)
- [SoK: Hardware-supported Trusted Execution Environments](#)
- [Hardware-assisted Trusted Execution Environments: Look Back, Look Ahead](#)
- [Intel SGX Explained](#)
- [Android Security Reference: TEE](#)
- [TEE.fail: Breaking Trusted Execution Environments via DDR5 Memory Bus Interposition](#)