CLOUD COMPUTING

SECURITY

Lorenzo Leonini
lorenzo.leonini@unine.ch

Lecture objectives

- Introduce some of the specific security concerns introduced by Cloud computing
- Overview solutions for protecting data and computation integrity
- Describe novel approaches for confidential data processing in the Cloud

Introduction (1)

- Cloud computing platforms and applications = tempting attack target
  - Large amount of data in the same place
    - Financial gain from selling/trading sensitive data
  - Resources are accessible online, remotely
  - Multi-tenancy offers larger attack surface
    - One service provider holds data from multiple companies
    - Co-location of resources offer new attack opportunities
    - Vulnerability in one service can affect others
    - Less control over software stack

Introduction (2)

- Trust towards Cloud provider must be very high
  - Is the provider worth this trust?
  - Will the provider need to abide by local privacy-threatening regulations? (e.g. PATRIOT act)
  - What if the provider gets corrupted?
- Focus on security aspects that are specific to the context of Cloud computing systems
  - Application-level security (authentication, etc.) and OS-level security (patches, best practices, policies) obviously still important!

Why is Cloud security important?

Yahoo hack: 1bn accounts compromised by biggest data breach in history

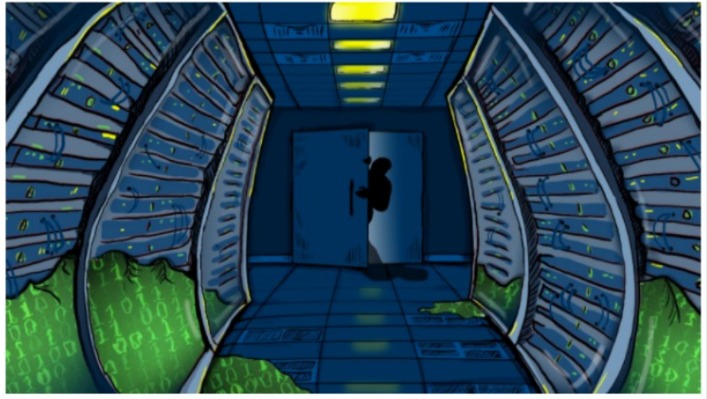
The latest incident to emerge
from the breach of 500m use

Yahoo have said the stolen user accou
Photograph: Dado Ruvic/Reuters

Yahoo said on Wednesday it l
data from more than 1bn use
making it the largest such bre

Another Day, Another Hack: 117 Million LinkedIn Emails And Passwords

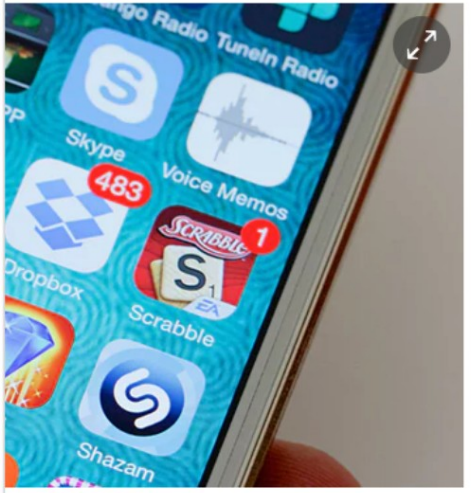
LORENZO FRANCESCHI-BICCHIERAI
May 18 2016, 10:00am



Four years later, the 2012 LinkedIn breach just got way worse.

Dropbox hack leads to leaking of 68m
n the internet

ng encrypted passwords and details of
stomers, has been leaked

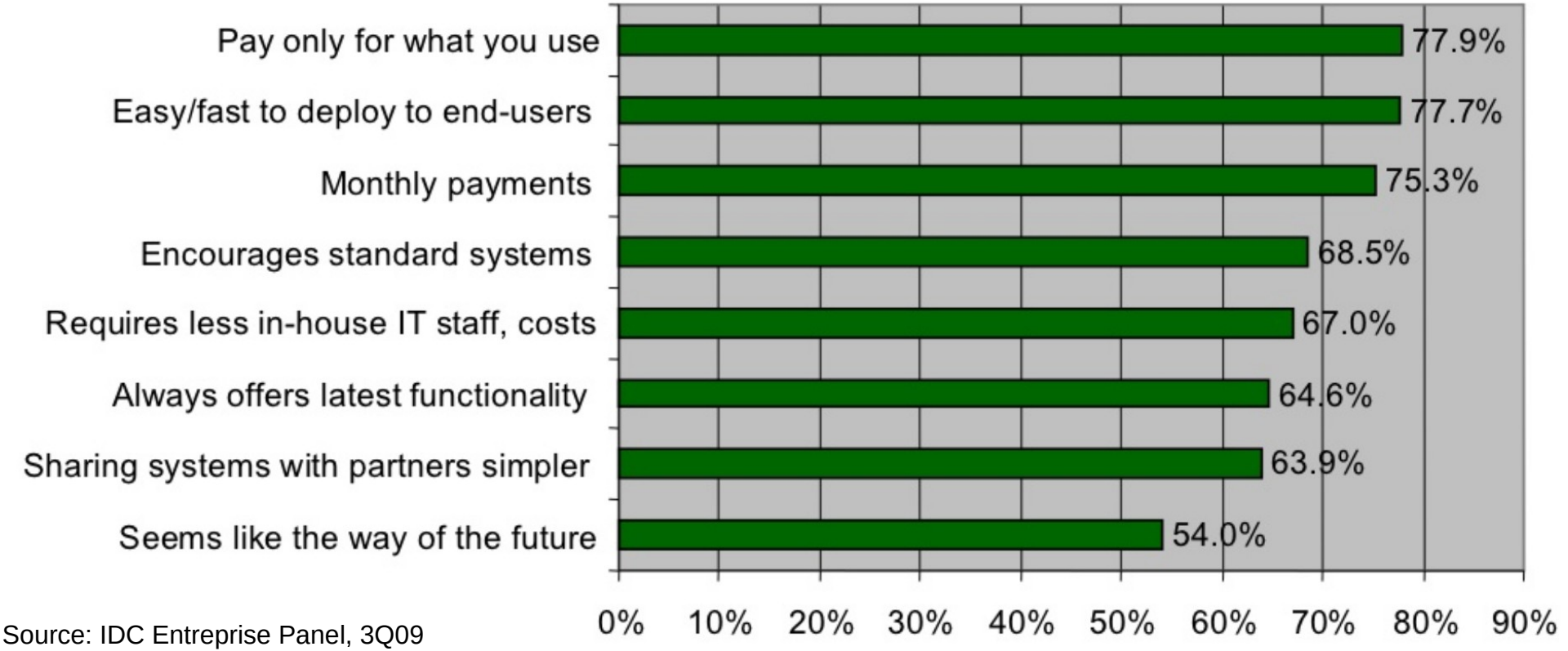


roblem of password reuse. Photograph: Alamy

has been hacked, with over 68m users' email
to the internet.

Cloud benefits

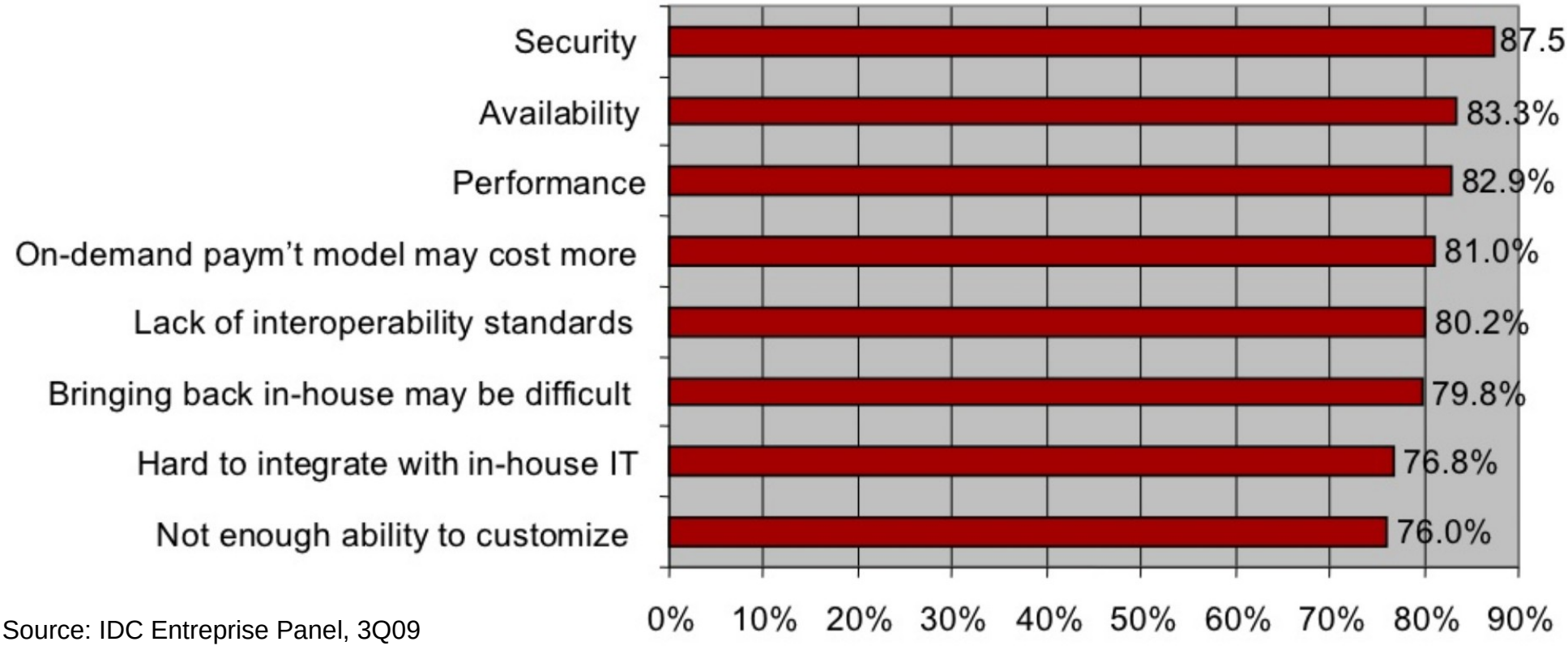
Q: Rate the benefits commonly ascribed to the 'cloud'/on-demand model



Source: IDC Entreprise Panel, 3Q09

Cloud challenges

Q: Rate the challenges/issues of the 'cloud'/on-demand model



Source: IDC Entreprise Panel, 3Q09

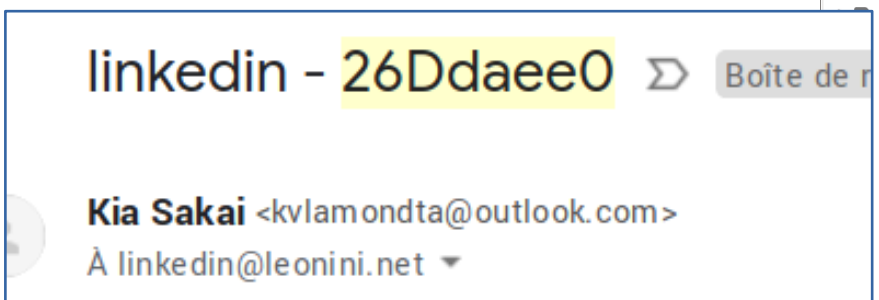
Top security threats for the Cloud

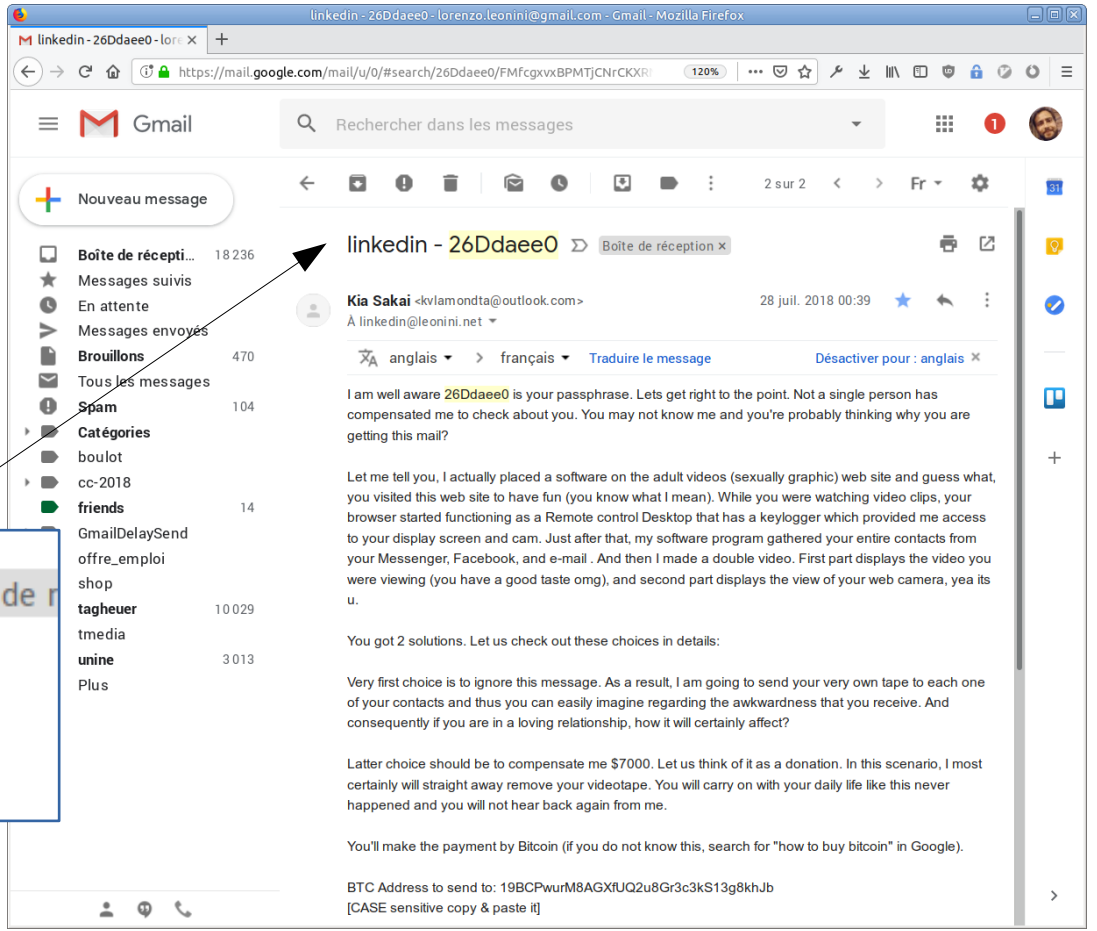
- Survey conducted by Cloud Security Alliance with 271 experts, classified 12 top threats for Cloud security (in decreasing order of importance):
  - 1. Data breaches
  - 2. Insufficient Identity, Credential and Access Management
  - 3. Insecure Interfaces and APIs
  - 4. System Vulnerabilities
  - 5. Account Hijacking
  - 6. Malicious Insiders
  - 7. Advanced Persistent Threats
  - 8. Data Loss
  - 9. Insufficient Due Diligence
  - 10. Abuse and Nefarious Use of Cloud Services
  - 11. Denial of Service
  - 12. Shared Technology Vulnerabilities

http://www.storm-clouds.eu/services/2017/04/the-treacherous-12-cloud-computing-top-threats-in-2016/

Example: Linkedin 2012 data breach

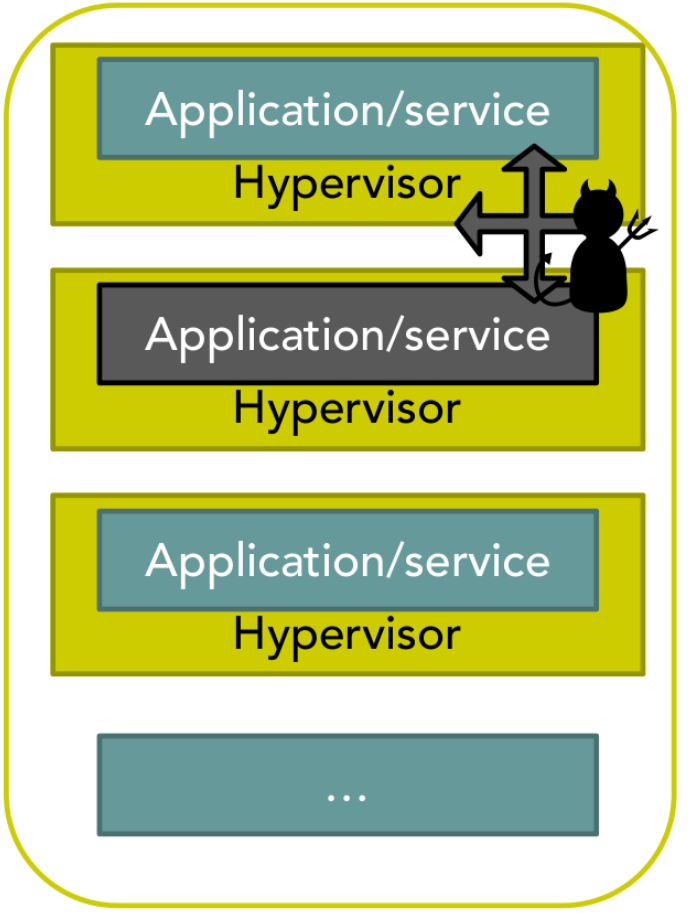
- 100+ millions of stolen accounts
  - Passwords not protected





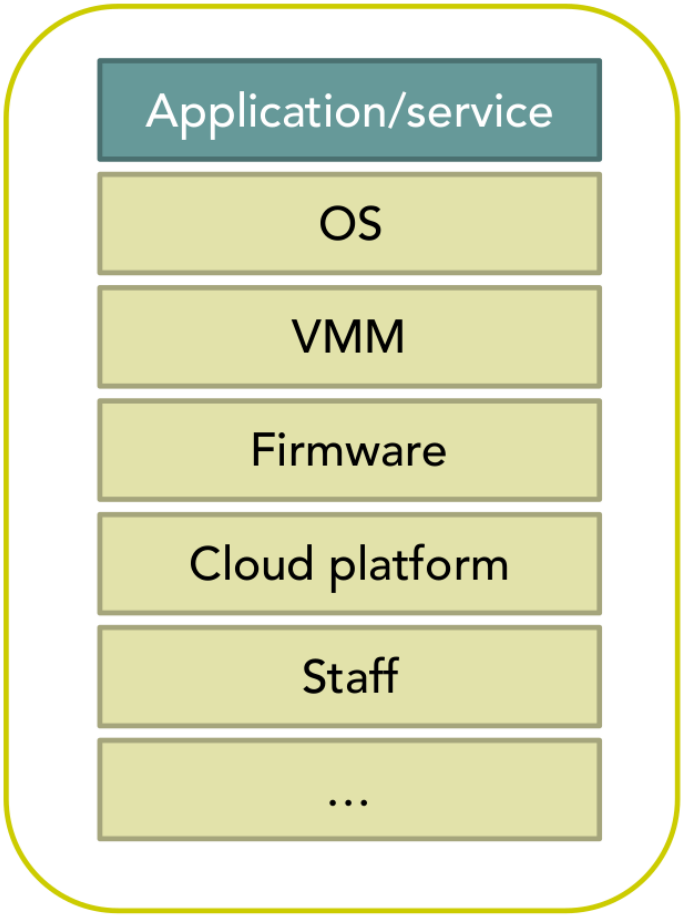
The Provider perspective

- Cloud provider needs to protect against malicious customers
  - Hypervisor-based isolation
- One-way protection
  - Does not protect customer against provider



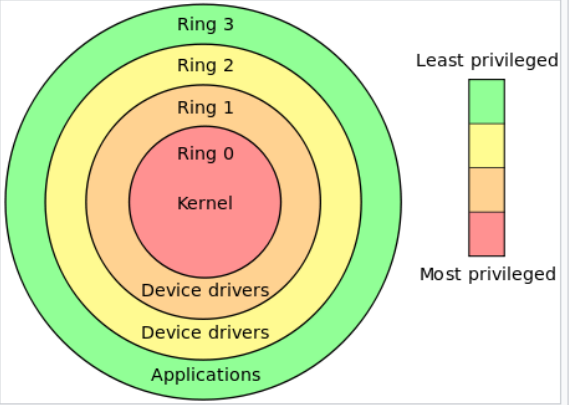
The Client perspective

- Cloud tenant is forced to trust the provider...
  - Including personnel
  - Including every software component
- Ideally, client would like to trust only her/his service



Additionnaly…

- Tenants and providers have to trust hardware manufacturers
  - CPU manufacturer
    - Intel Management Engine
      - On chip operating system
      - Able to communicate via the network !
      - Works even when the PC is powered-off (if power cord attached…)
      - Considered as a backdoor by Electronic Frontier Foundation (EFF)
  - Network infrastructure
- Several contries refuses to use foreign hardware in their critical infrastructures
  - E.g. China with CISCO routers



Privilege rings for the x86. The ME is colloquially categorized as ring -3, below System Management Mode (ring -2) and the hypervisor (ring -1), all running at a higher privilege level than the kernel (ring 0)

wikipedia

Multi-tenancy and security

- Cost saving and efficiency thanks to resource sharing
  - Virtualized machines over same physical hosts
  - Sharing of the network
  - Sharing of services (VM image storage, container repository, ...)
- Virtualization = new security threats
  - VMM/hypervisor can be attacked to get information about, or access to, virtual resources/machines
    - Very difficult to detect from inside a VM!
    - Access to the hypervisor = access to all of the VMs memory, including page tables, kernel memory, etc.
- On the other hand, cloud platforms also likely to be administered by dedicated security experts and follow security updates and best practices more diligently

Protected mode not sufficient

- Protected mode (rings) protects OS from applications, and applications from one another...
  - Until a malicious applications exploits a flaw to gain full privileges and then tampers with the OS or other applications
  - Applications are not protected from privileged code attacks
- The attack surface is the whole software stack
  - Applications, OS, VMM, drivers, BIOS...

Software attacks in the Cloud

- Performed by executing software on the victim computer
  - Can be done remotely
- Vast majority of attacks exploit vulnerabilities in software components
  - E.g., memory safety violations in C/C++
  - E.g., bogus APIs to services or IaaS infrastructure management interface
  - Now also attack hardware: Spectre and Meltdown for Intel CPUs

The software stack

- Cloud platforms contain enormous amounts of code that must be trusted
  - Linux: ~20 MLOC
  - KVM: ~13 MLOC
  - OpenStack: ~2 MLOC
- Cloud platforms are effectively a trusted computing base (TCB): all components of the system are critical to security
  - Software, hardware

Bugs are a reality

- More code → more bugs
  - Exploited vulnerability may lead to complete disclosure of confidential data
- Xen hypervisor
  - 184 vulnerabilities (2012-2016)
    - http://www.cvedetails.com/product/23463/XEN-XEN.html?vendor_id=6276
- Linux kernel
  - 721 vulnerabilities (2012-2016)
    - http://www.cvedetails.com/product/47/Linux-Linux-Kernel.html?vendor_id=33
- Especially bad in privileged software as it may result in unrestricted access to the system

AWS dedicated instances/hosts

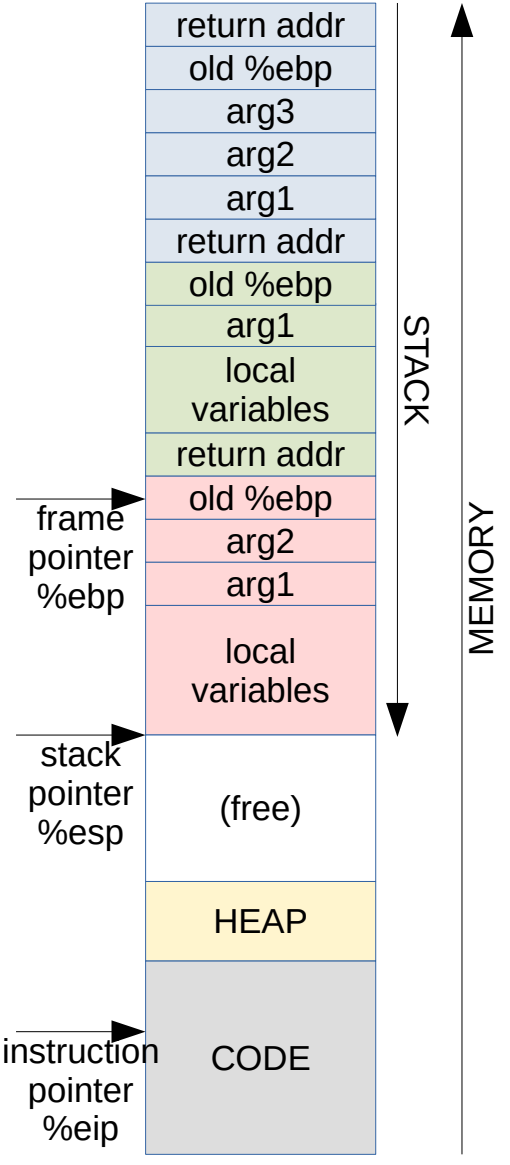
- Avoid the danger of multi-tenants sharing the same physical host
- You can avoid this particular risk by requesting your (underlying) computing instances (EC2) to run on dedicated hardware
  - AWS Dedicated Instances
- AWS Dedicated Hosts
  - Instance placement control
    - Instances affinity
  - Visibility of sockets and physical cores
    - Can be useful to manage software licensing

Example of software attacks

- Control-flow hijacking
  - Goal: execute arbitrary code on the target machine
  - Modify application's control flow
    - Overwrite return address by writing beyond allocated buffer on the stack (instruction pointer)
- Classical attacks
  - Buffer overflow
    - Inject code (in the stack) and point to it
  - Return-into-libc
    - Inject function parameters (in the stack), call function in libc (exec)
  - Return-oriented programming (ROP)
    - Jump to sequences of instructions (gadgets) already present in memory (e.g., libc) ending with a return
    - Chain gadgets to execute arbitrary code

Calling functions

- Call function
  - Create a new stack frame
    - Pointer to previous stack frame (old %ebp)
      - Move %ebp to this address
    - Set function arguments
    - Set local variables
    - Save current instruction pointer (%eip) into return address
    - Move stack pointer (%esp) to the top of the stack
  - Move instruction pointer (%eip) to the code of the function
- Function returns
  - Deallocate stack frame
    - %ebp = old %ebp
    - %eip = return addr



Example: Heartbleed bug

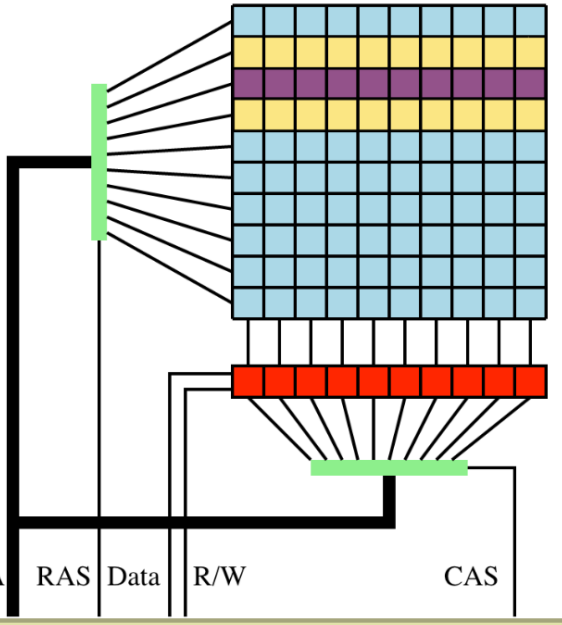
- Serious vulnerability in the popular OpenSSL cryptographic software library
  - Very widely used: apache/nginx (66% of Web servers), email servers, chat servers, VPN, etc.
- Buffer overrun when replying to a heartbeat message
- Can be exploited to get read access to memory
  - The attacker can obtain sensitive data from server's memory: passwords, private keys, …
- https://xkcd.com/1354/

Hardware attacks in the Cloud

- May require physical access to the machine
- Bus snooping
  - Dump CPU ↔ memory communication
- Cold boot attacks
  - DRAM retains its state for a short period of time
  - Power cycle the machine, boot to a lightweightOS, dump memory contents...
  - Or remove memory modules, plug into another machine, dump memory contents

Example: "Row hammer" attack

- Attack the system by causing bit-flips in memory
  - Bits are capacitors
  - Accessing physical bits causes neighboring bits to flip
  - Carefully chosen addresses can result in privilege escalation
  - Reliability problems often evolve in security issues
- Effect
  - Sandbox escape
  - Corrupted page table

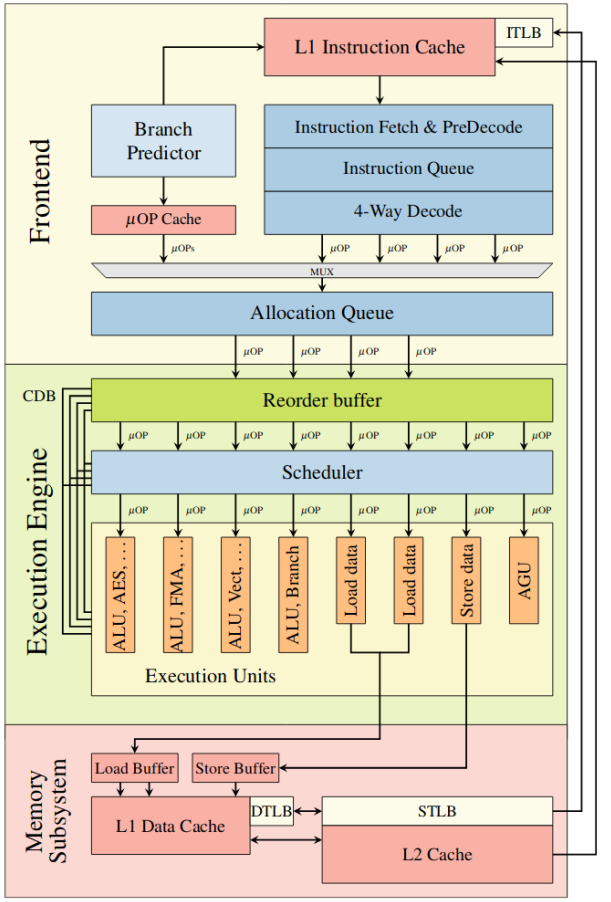


Rapid row activations (yellow rows) may change the values of bits stored in victim row (purple row). [wikipedia]

```
code1a:
  mov (X), %eax // read from address X
  mov (Y), %ebx // read from address Y
  clflush (X)   // flush cache for address X
  clflush (Y)   // flush cache for address Y
  jmp code1a
```

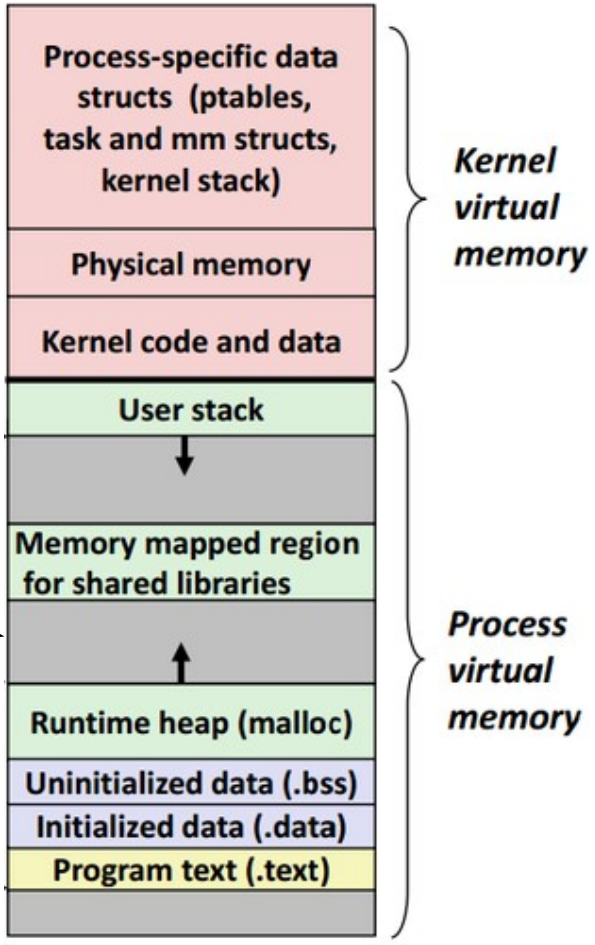
Example: "Meltdown attack" (1)

- Modern CPUs have multiple execution pipelines
  - They can execute instructions that have no dependencies
  - Out of order execution in CPUs
    - Many instructions are executed *before* the real execution point
- At some point instructions are re-ordered
  - And sanity checks are done
    - If checks fails → empty pipeline
- Memory
  - RAM
  - L1/L2/L3 CPU memory caches - for faster access



Example: "Meltdown attack" (2)

- Processes virtual memory (32 bits)
  - 0 - 3Go: Process virtual memory
  - 3 - 4Go: Kernel virtual memory
- Side channel attack - pseudo code
  - 1. `m = read_byte(<kernel_address>)`
    - → Will crash (segfault) the app...
  - 2. `read_byte(<free_memory> + m * 4096)`
    - → ...but this instruction will already have been executed
    - → The byte will be in L1 cache
  - Check all bytes (<free_memory> + [0-256] * 4096)
    - Measure access time
      - Fast access → in cache → **this is the value of the kernel byte**
  - Repeat the process to read all kernel memory!

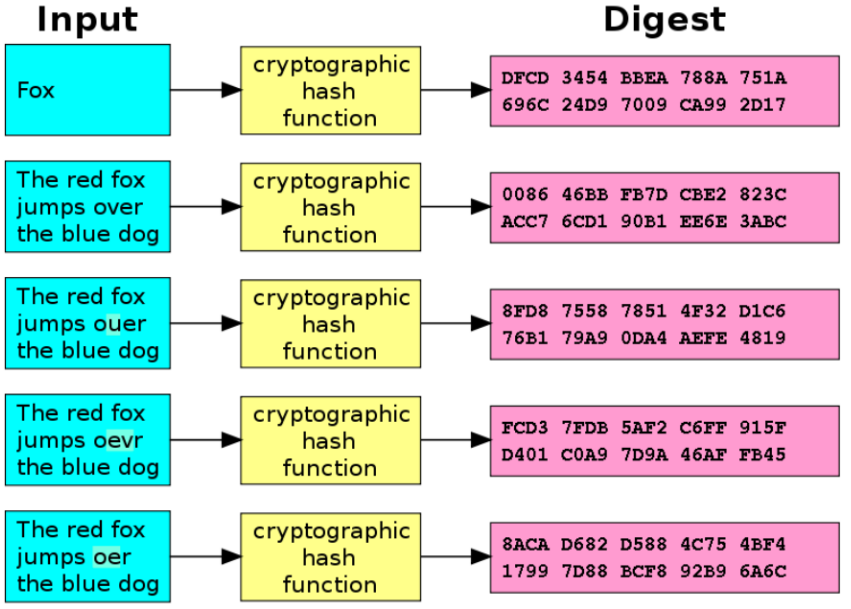


Goals: security in the Cloud

- Confidentiality
  - Information is not made available or disclosed to unauthorized individuals, entities, or processes
  - Encryption
- Integrity
  - Data cannot be modified in an undetected manner
  - MAC, digital signature

Tools: cryptographic hash function

- Algorithm that maps arbitrary input string into a fixed-size output (hash)
  - Infeasible to invert (one-way function)
  - Deterministic
  - Collision resistant
    - Probability of two messages having the same hash is negligible
    - “Infeasible” to find collisions



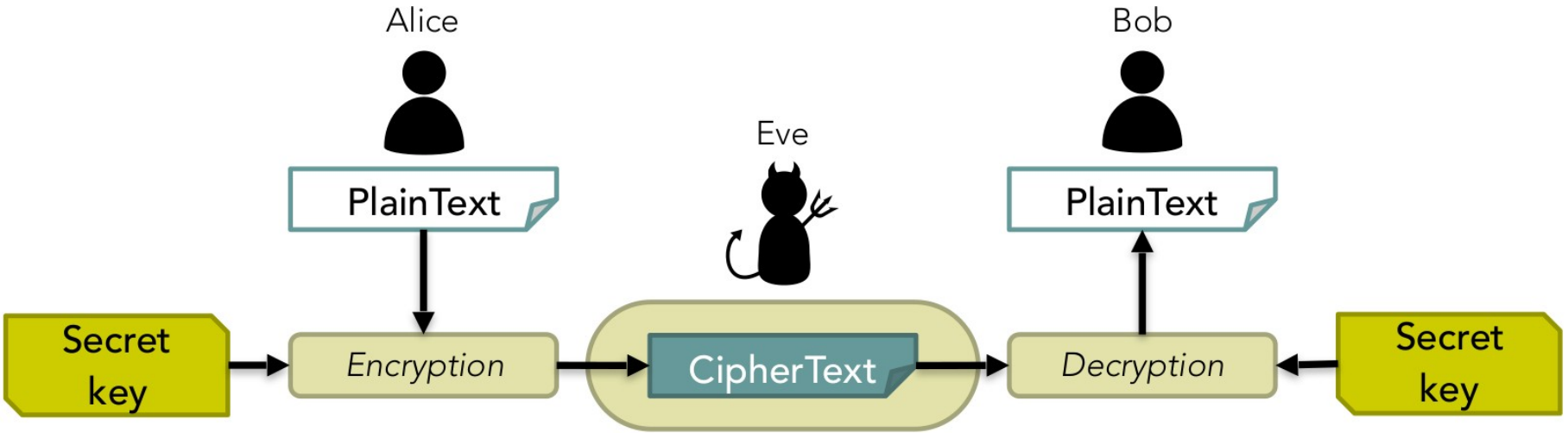
Even small changes in the source input (here in the word “over”) drastically change the resulting output, by the so-called avalanche effect. [wikipedia]

Password hashing

- password → concat(salt, hash(concat(password, salt))
  - Passwords must never be stored cleartext or not salted (linkedin)
    - To check password validity, hashes are compared
  - A salt denies building a database of hashes
  - Hash functions should be slow (to limit feasibility brute force attempts on the DB)
    - Should also be difficult to build specialized hardware to speed up the process
  - A popular function today is *scrypt*
    - `scrypt(pass, salt, time_cost, memory_cost)`
  - Linux default: 5000 rounds of SHA-512, salt: 128 bits
    - `$ mkpasswd -m sha-512 -S SALTSALT -s <<< YourPassword`
      - `$6$SALTSALT$HxfmrQmCf1Urlf5H0HHeNx.B040XjhbJQZeq6ipg8QVxk51T2FkTLBiDU6m9NiAu4.GDiPjMPjP64DYVi8OhN1`

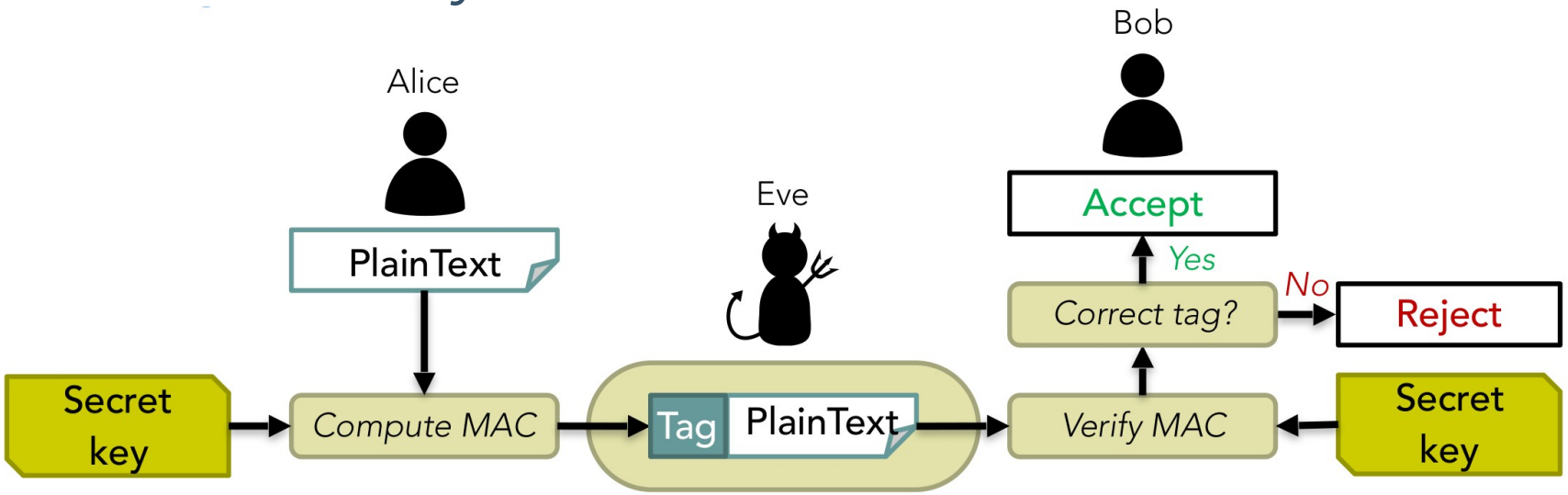
Tools: symmetric cryptography

- All parties have a shared secret key
  - Same key for encryption and decryption
- Example: AES (advanced encryption standard)



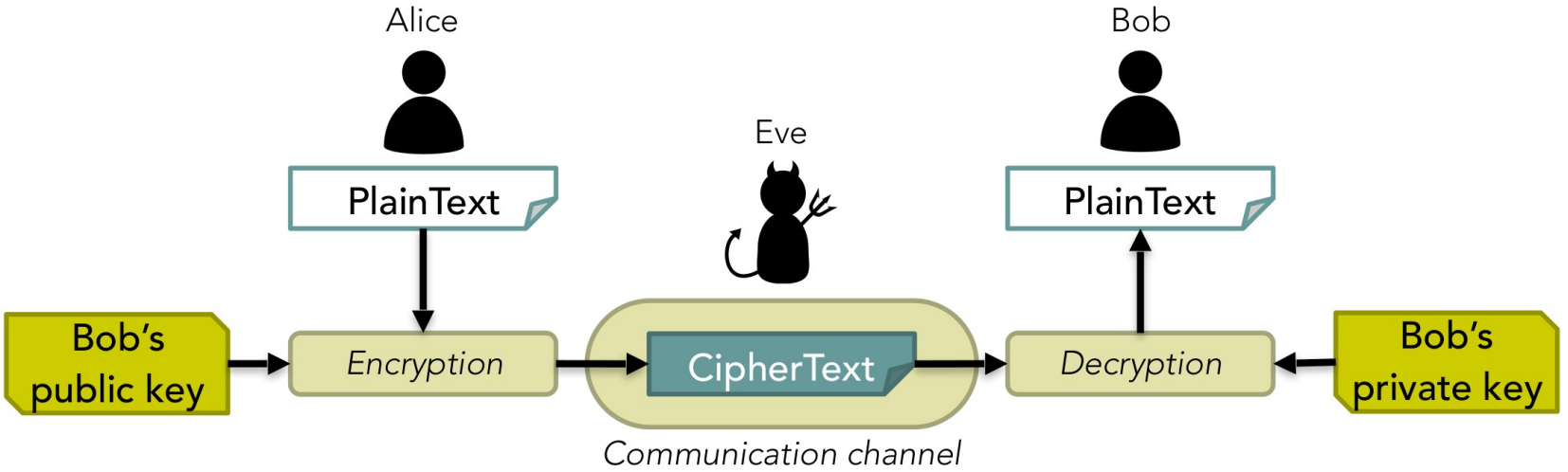
Tools: symmetric cryptography

- Message authentication code (MAC)
  - Tag to check integrity and authenticity of the message
  - The key is required to produce correct MAC
  - No confidentiality



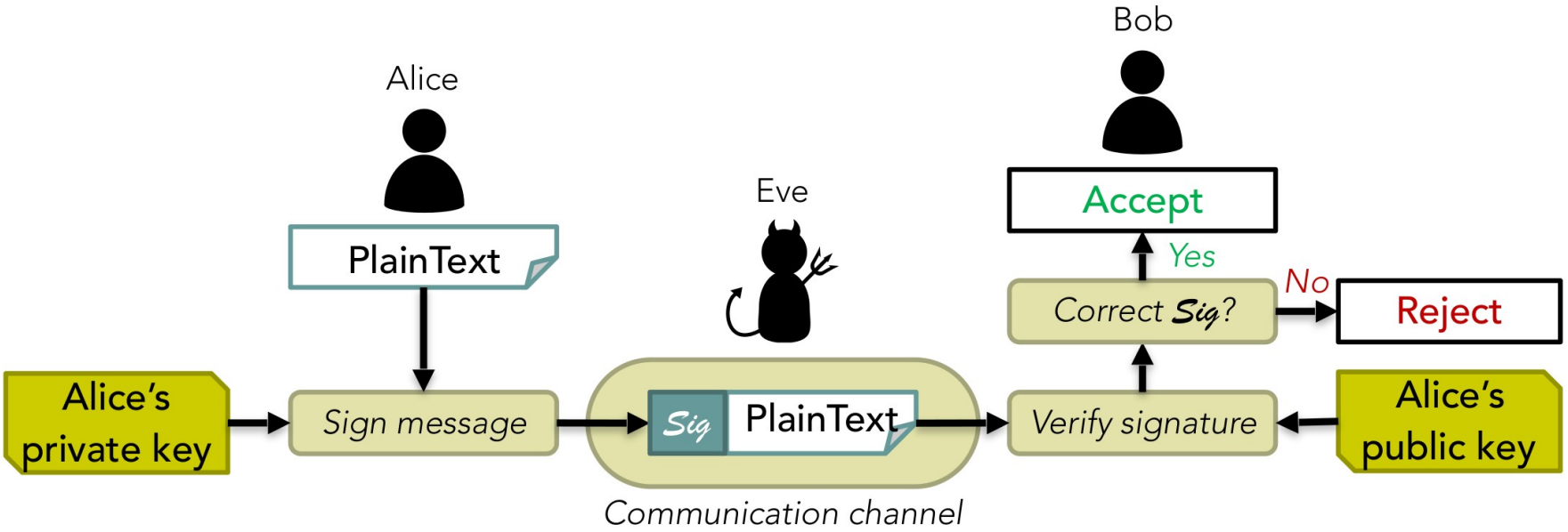
Tools: asymmetric cryptography

- Each party has a key pair
  - Public key: not secret, used for encryption
  - Private key: secret, used for decryption
- Example: RSA (Rivest-Shamir-Adleman)
- Typically used in SSL, SSH, PGP, ...



Tools: asymmetric cryptography

- MAC in asymmetric world: digital signature
  - Sign with sender's private key
  - Verify with sender's public key



Debian SSH 2008

- Debian SSH package maintener big mistake!
  - He removes all the randomness of the private key generation!

Debian OpenSSL Predictable PRNG

Links

Original URL: http://metasploit.com/users/hdm/tools/debian-openssl/ (**Mirror**)

Exploit:

- https://www.exploit-db.com/exploits/5622/ (Perl)
- https://www.exploit-db.com/exploits/5720/ (Python)
- https://www.exploit-db.com/exploits/5632/ (Ruby)

Recommend Tool: **Crowbar** (able to brute force SSH keys)

Testing Method: ssh-vulnkey & **dowkd.pl**

The Bug

On May 13th, 2008 the Debian project announced that Luciano Bello found an interesting vulnerability in the OpenSSL package they were distributing. The bug in question was caused by the removal of the following line of code from *md_rand.c*

```
MD_Update(&m,buf,j);
[ .. ]
MD_Update(&m,buf,j); /* purify complains */
```

These lines were removed because they caused the Valgrind and Purify tools to produce warnings about the use of uninitialized data in any code that was linked to OpenSSL. You can see one such report to the OpenSSL team here. Removing this code has the side effect of crippling the seeding process for the OpenSSL PRNG. Instead of mixing in random data for the initial seed, the only "random" value that was used was the current process ID. On the Linux platform, the default maximum process ID is 32,768, resulting in a very small number of seed values being used for all PRNG operations.

Ensuring data confidentiality

- Data-at-rest protection
  - Encrypt data before storing on disk
  - Encrypted file systems, full-disk encryption (FDE)
- Communication protection
  - Well established end-to-end encryption mechanisms
  - Transport layer security (TLS)
- Trusted platform module (TPM)
  - Tamper-resistant chip external to the CPU
  - Facilities for secure generation of cryptographic keys, remote attestation, sealed storage
  - Limited protection, susceptible to physical attacks

Ensuring data confidentiality

- But how to ensure confidentiality during computations?
  - Need to decrypt data before processing
  - Encryption keys/plaintext data in main memory/ registers
- Memory dump will reveal all secrets

Securing data during computation

- Challenges
  - Data should be searchable (e.g., ability to perform range queries)
  - Must not leak information (e.g., statistical attack knowing the distribution of values)
  - Privacy-utility-performance tradeoffs
- Tools
  - **Software**-level encryption
    - Deterministic or non-deterministic, order-preserving, homomorphic, ...
    - *Specific* vs. *generic* encryption mechanisms
  - **Hardware** support for trusted computing (e.g., SGX)

Encrypted data processing

- Homomorphic encryption
  - "a form of encryption which allows specific types of computations to be carried out on ciphertext and generate an encrypted result which, when decrypted, matches the result of operations performed on the plaintext" [wikipedia]
- Fully homomorphic encryption [Gentry 2010]
  - Generic: supports arbitrary functions on encrypted data
  - Addition, multiplication, binary operations

Homomorphic encryption

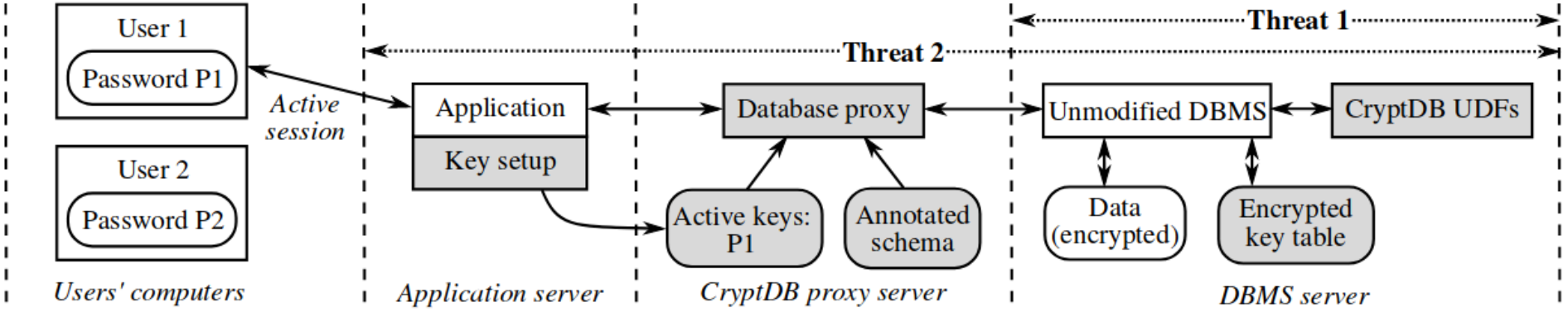
- **HELib**: open-source homormophic encryption library in C++ by IBM [Shoup and Halevi, 2012]
  - Many optimizations to make HE "practical", i.e., make homomorphic evaluation run faster
  - Low-level routines (set, add, multiply, shift, etc.)
- Still far from being practical
  - Addition: ~1+ ms
  - Multiplication: ~10/100+ ms
  - Evaluated the AES-128 circuit in 36 hours in 2012 (vs. 2 ms in the clear)
    - https://mpclounge.files.wordpress.com/2013/04/hespeed.pdf
    - → 8 to 9 orders of magnitude slower !

Specific encryption schemes

- Homomorphic encryption generic but costly
- Specific encryption schemes target limited operations on data
  - Exact-match search
  - Nearest-match search
  - Range queries
  - Equalities
- Example
  - CryptDB: encrypted SQL database

CryptDB: encrypted SQL database

- Proxy between client running in trusted zone and database running in untrusted zone (Cloud)
  - Query rewriting and data encryption
  - Decryption of received query results
  - Handling of keys
  - Untrusted zone only sees encrypted data



CryptDB: encrypted SQL database

- Standard encryption (AES) prevents from performing any type of query
  - Ciphertext for same value encrypted twice is different: not possible to do even equality queries
  - If e(x) is encrypted value of x: even if x == y, e(x) ≠e(y) with high probability
  - This randomization is a good thing in general: prevents Known-Plaintext Attacks
    - Not possible to determine y even if x and e(x) known
- Specific encryptions allowing queries
  - Various privacy-query expressiveness tradeoffs

CryptDB: encryption levels (1)

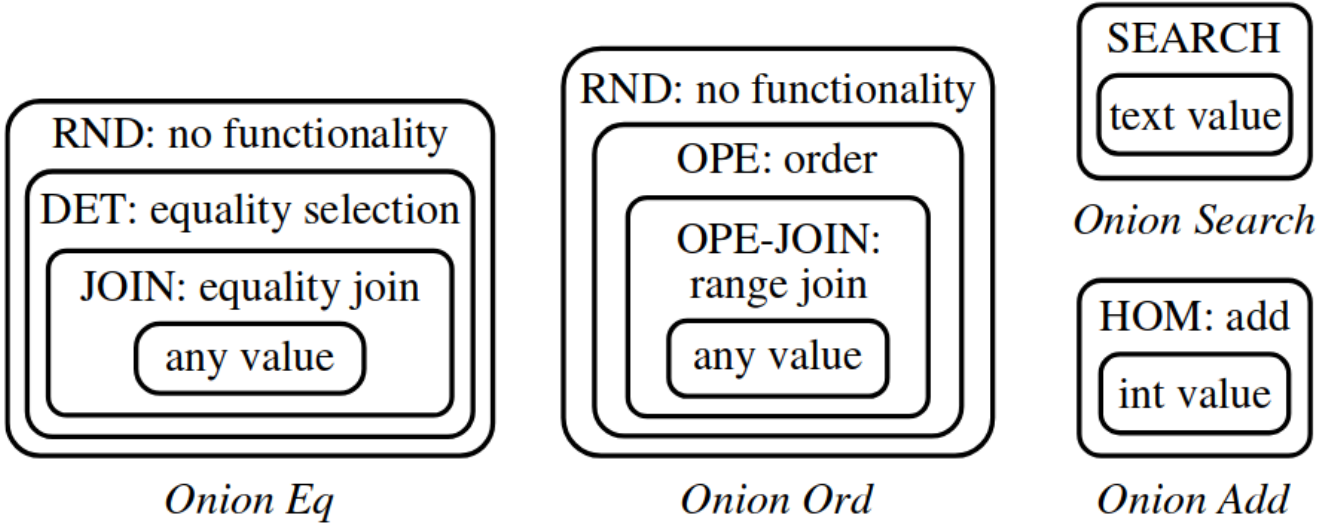
Encryption	Security	Query expressiveness
Random (e.g. AES)	Two equal values map to different ciphertexts w.h.p.: indistinguishability under adaptive chosen-plaintext attack (IND-CPA)	No query possible
Deterministic (e.g. Pseudo-random permutation)	Two equal values map to same ciphertext, but ciphertexts do not preserve ordering	+ Equality search SELECT * FROM t WHERE v=x
Order-preserving (random order preserving mapping)	For two values *x* and *y,* guarantees that (1) ciphertext for *x* is always *e(x)* and (2) if $x < y$ then $e(x) < e(y)$.	+ Range queries ORDER BY, MIN, MAX, SORT

CryptDB: encryption levels (2)

Encryption	Security	Query expressiveness
Summation-homomorphic (Paillier)	For two values *x* and *y,* guarantees that (1) ciphertext for *x* is always *e(x)* and (2) *e(x) * e(y) = e(x+y).*	+ Summations SUM(col), COUNT(col), AVG(col), etc.
Join	Provided with a transformation T1 and T2, can compare the values in two columns C1 and C2 encrypted with K1 and K2. T1 transforms values in C1 to values encrypted with K1 then K2, and vice versa for T2.	Allow joins between two columns *even if encryption key used was different*
Word search (specific enc. scheme)	Vulnerable to statistics attacks if word frequency is known by attacker	Limited full-word version of LIKE

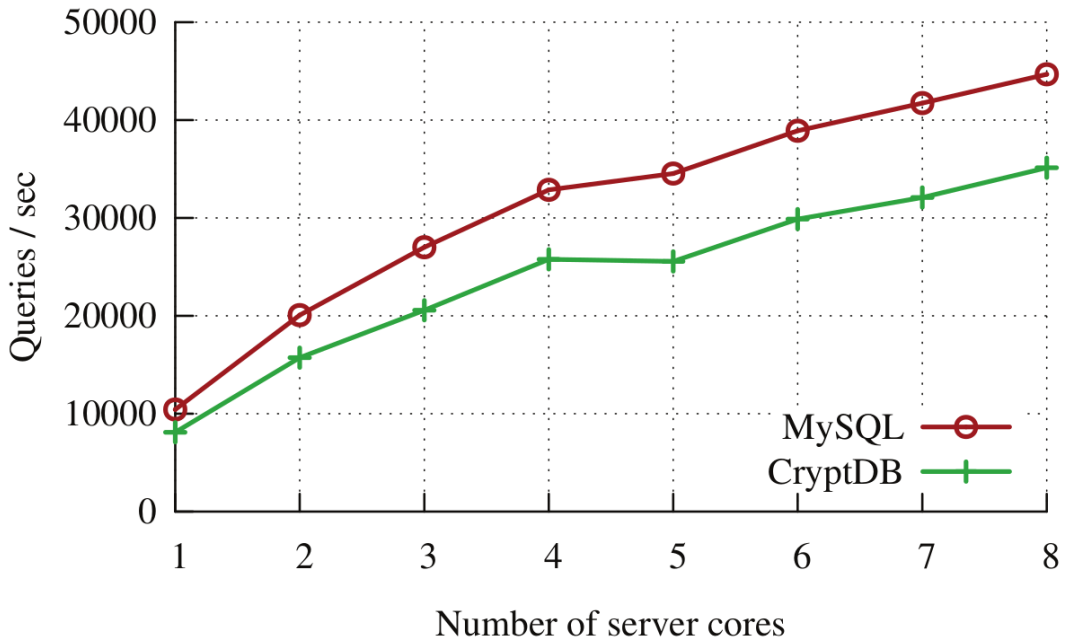
CryptDB: onion layers encryption

- CryptDB can store data encrypted multipletimes with increasingly strong ciphers
- If support needed for more expressiveness, must give up a layer of security and provide servier with decryption key for his layer
  - e.g. provide key for RND layer to allow exact match search



CryptDB performance

- Throughput for TPC-C queries (standard benchmark)
  - Varying number of cores on the underlying MySQL DBMS server
  - CryptDB requires an additional machine for the proxy!



Towards hardware support

- Software-based confidentiality or data protection mechanisms are either:
  - Of limited applicability (only target some specific problem/computation)
  - With limited performance
  - And often both!
- Trusted computing base remains the entire hardware and software stack
- Can we do better with some help from the hardware?

Intel SGX

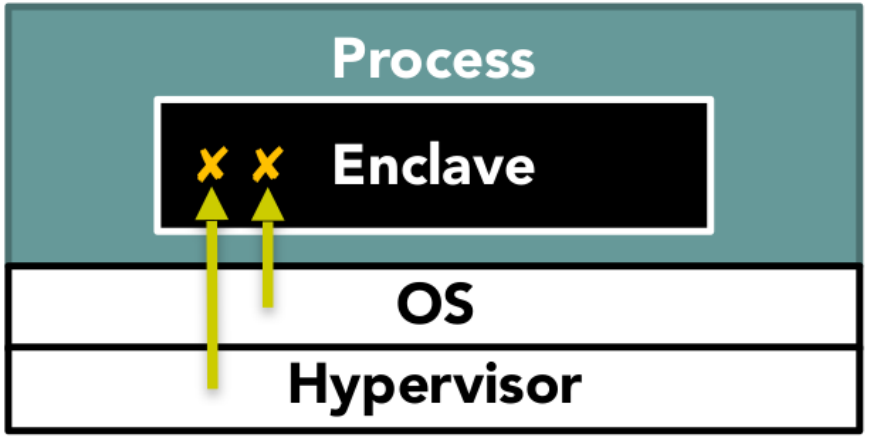
- “Software guard extensions”
- Hardware extension in recent Intel CPUs
  - Skylake (2015), Kaby lake (2016)
- Protects confidentiality and integrity of code and data in untrusted environments
  - Platform owner is considered malicious
  - Only the CPU chip and the isolated region are trusted

Enclaves

- SGX introduces the notion of “enclave”
  - Isolated memory region for code and data
  - New CPU instructions to manipulate enclaves and a new enclave execution mode
- Enclave memory is encrypted and integrity-protected by the hardware
  - Memory encryption engine (MEE)
  - No plaintext secrets in main memory
- Enclave memory can be accessed only by the enclave code
  - Protection from privileged code (OS, hypervisor)

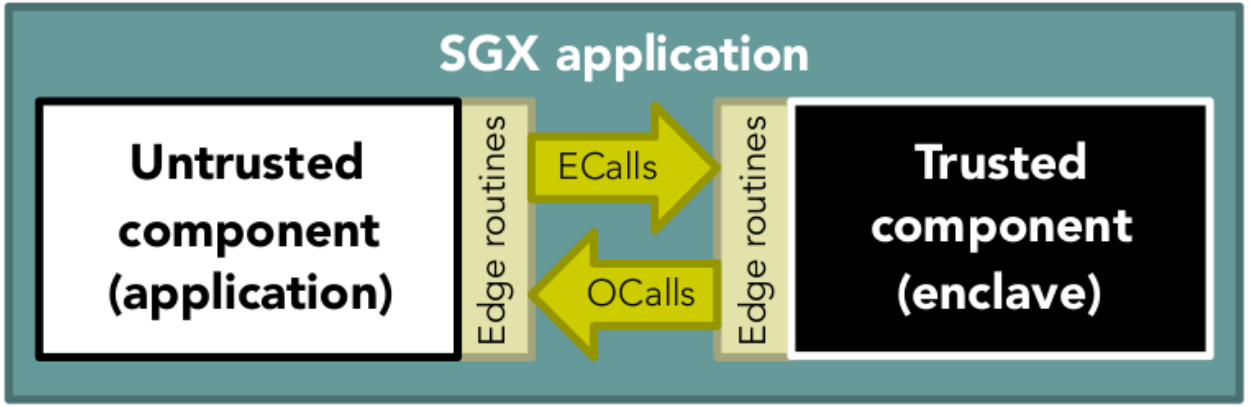
Enclave memory

- Enclave memory is not accessible to other software
  - But enclave can access memory within its process
- Application has ability to defend its secrets
  - Attack surface reduced to just enclaves and CPU
  - Compromised software cannot steal application secrets



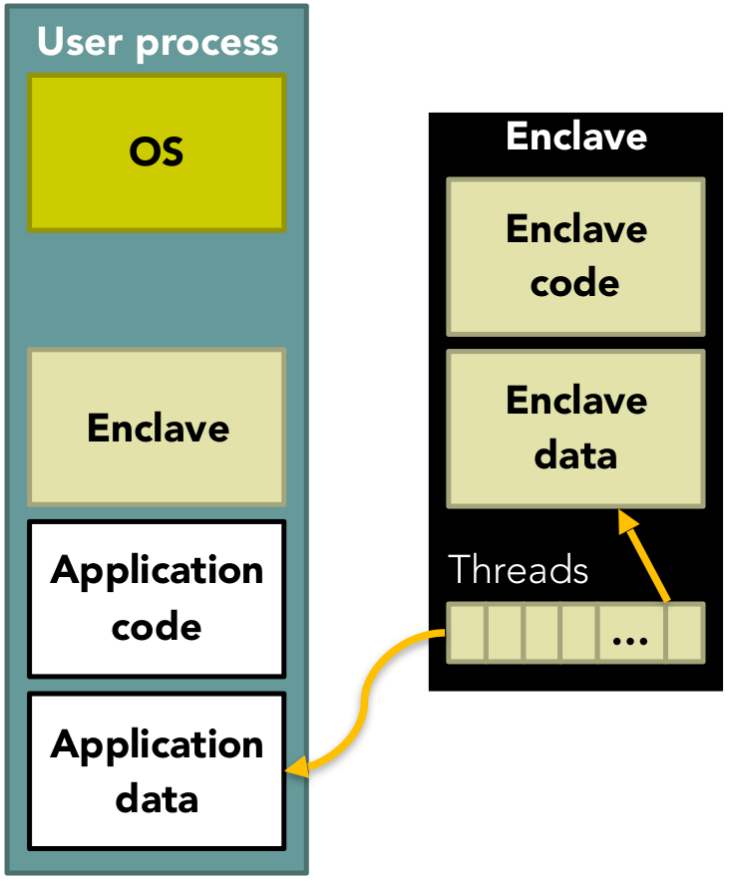
Enclave memory

- Enclave define APIs
  - Enclave interface functions: ECalls to provide input data to the enclave
  - Calls outside the enclave: OCalls to return results from the enclave
  - Constitute the enclave boundary interface



SGX execution model

- Trusted execution environment in a process
  - With its own code and data
  - With controlled entry points
  - Provides confidentiality
  - Provides integrity
  - Supporting multiple threads
  - With full access to application memory



Enclave page cache (EPC)

- Physical memory region protected by the MEE
  - EPC holds enclave contents
- Shared resource between all enclaves running on a platform
  - Currently only 128MB
  - ~96MB available to the user, the rest is for metadata
  - Likely to increase in future versions
- Content encrypted while in DRAM, decrypted when brought to CPU
  - Plaintext in CPU caches

Example

SGX application: unstrusted code

```
char request_buf[BUFFER_SIZE];
char response_buf[BUFFER_SIZE];

int main()
{
  ...
  while(1)
  {
    receive(request_buf);
    ret = EENTER(request_buf, response_buf);
    if (ret < 0)
      fprintf(stderr, "Corrupted message\n");
    else
      send(response_buf);
  }
  ...
}
```

Enclave: trusted code

```
char input_buf[BUFFER_SIZE];
char output_buf[BUFFER_SIZE];

int process_request(char *in, char *out)
{
  copy_msg(in, input_buf);
  if(verify_MAC(input_buf))
  {
    decrypt_msg(input_buf);
    process_msg(input_buf, output_buf);
    encrypt_msg(output_buf);
    copy_msg(output_buf, out);
    EEXIT(0);
  } else
    EEXIT(-1);
}
```

Server:

- **Receives encrypted requests**
- **Processes them in enclave**
- **Sends encrypted responses**

Example

SGX application: unstrusted code

```
char request_buf[BUFFER_SIZE];
char response_buf[BUFFER_SIZE];

int main()
{
  ...
  while(1)
  {
1   receive(request_buf);
2   ret = EENTER(request_buf, response_buf);
    if (ret < 0)
3     fprintf(stderr, "Corrupted message\n");
    else
4     send(response_buf);
  }
  ...
}
```

Enclave: trusted code

```
char input_buf[BUFFER_SIZE];
char output_buf[BUFFER_SIZE];

int process_request(char *in, char *out)
{
  copy_msg(in, input_buf);
  if(verify_MAC(input_buf))
  {
    decrypt_msg(input_buf);
    process_msg(input_buf, output_buf);
    encrypt_msg(output_buf);
    copy_msg(output_buf, out);
    EEXIT(0);
  } else
    EEXIT(-1);
}
```

1. **Receive a requests**
2. **Enter the enclave with two arguments: pointer to a buffer with encrypted request and pointer to a response buffer**
   (EENTER instruction switches CPU to the enclave mode and transfers control to predefined location in enclave)
3. **Print error message if enclave returns an error**
4. **Send the response provided by the enclave**

Example

SGX application: unstrusted code

```
char request_buf[BUFFER_SIZE];
char response_buf[BUFFER_SIZE];

int main()
{
  ...
  while(1)
  {
    receive(request_buf);
    ret = EENTER(request_buf, response_buf);
    if (ret < 0)
      fprintf(stderr, "Corrupted message\n");
    else
      send(response_buf);
  }
  ...
}
```

Enclave: trusted code

```
char input_buf[BUFFER_SIZE];
char output_buf[BUFFER_SIZE];
1
int process_request(char *in, char *out)
{
2 copy_msg(in, input_buf);
3 if(verify_MAC(input_buf))
  {
4   decrypt_msg(input_buf);
    process_msg(input_buf, output_buf);
    encrypt_msg(output_buf);
    copy_msg(output_buf, out);
    EEXIT(0);
  } else
    EEXIT(-1);
}
```

1. **Enclave entry point**
2. **Copy request into enclave memory (enclave can access `in` buffer in untrusted memory while untrusted part cannot access `input_buf` buffer in trusted memory)**
3. **Check the MAC (assuming keys were already exchanged)**
4. **Decrypt request**

Example

SGX application: unstrusted code

```
char request_buf[BUFFER_SIZE];
char response_buf[BUFFER_SIZE];

int main()
{
  ...
  while(1)
  {
    receive(request_buf);
    ret = EENTER(request_buf, response_buf);
    if (ret < 0)
      fprintf(stderr, "Corrupted message\n");
    else
      send(response_buf);
  }
  ...
}
```

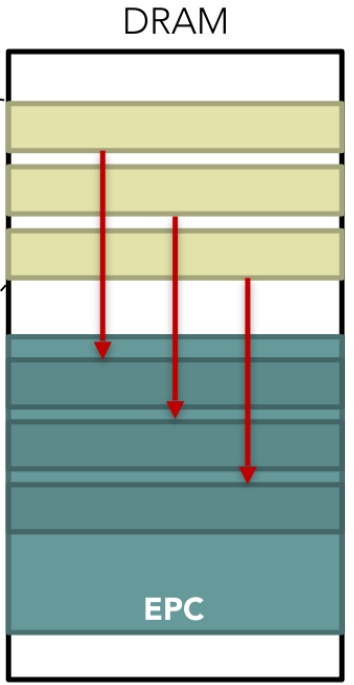
Enclave: trusted code

```
char input_buf[BUFFER_SIZE];
char output_buf[BUFFER_SIZE];

int process_request(char *in, char *out)
{
  copy_msg(in, input_buf);
  if(verify_MAC(input_buf))
  {
    decrypt_msg(input_buf);
5   process_msg(input_buf, output_buf);
6   encrypt_msg(output_buf);
7   copy_msg(output_buf, out);
8   EEXIT(0);
  } else
    EEXIT(-1);
}
```

5. **Process request and store result in output buffer**
6. **Encrypt result and add MAC**
7. **Write result to untrusted memory for access from outside**
8. **Exit the enclave (`EEXIT` instruction switches from enclave to normal mode and transfer control to the next location after `EENTER`, similar to regular return)**

Enclave construction

```
1 { char input_buf[BUFFER_SIZE];
2 { char output_buf[BUFFER_SIZE];

  int process_request(char *in, char *out)
  {
    copy_msg(in, input_buf);
    if(verify_MAC(input_buf))
    {
3     decrypt_msg(input_buf);
      process_msg(input_buf, output_buf);
      encrypt_msg(output_buf);
      copy_msg(output_buf, out);
      EEXIT(0);
    } else
      EEXIT(-1);
  }
```



Enclave is populated using a special instruction (EADD)

- **Contents are initially in untrusted memory**
- **Copied into the EPC in 4KB pages**

Both data and code are copied before starting execution in enclave

Enclave construction

- Enclave contents are distributed in plaintext
  - Can be inspected
  - Must not contain any (plaintext) confidential data
- Secrets are provisioned after the enclave was constructed and its integrity verified
- Problem: what if someone tampers with the enclave?
  - Contents are initially in untrusted memory

Enclave construction

- Someone may tamper with the enclave by modifying the code

```
int process_request(char *in, char *out)
{
  copy_msg(in, input_buf);
  if(verify_MAC(input_buf))
  {
    decrypt_msg(input_buf);
    process_msg(input_buf, output_buf);
    encrypt_msg(output_buf);
    copy_msg(output_buf, out);
    EEXIT(0);
  } else
    EEXIT(-1);
}
```

```
int process_request(char *in, char *out)
{
  copy_msg(in, input_buf);
  if(verify_MAC(input_buf))
  {
    decrypt_msg(input_buf);
    process_msg(input_buf, output_buf);
    copy_msg(output_buf, external_buf);
    encrypt_msg(output_buf);
    copy_msg(output_buf, out);
    EEXIT(0);
  } else
    EEXIT(-1);
}
```

Write unencrypted response to outside memory

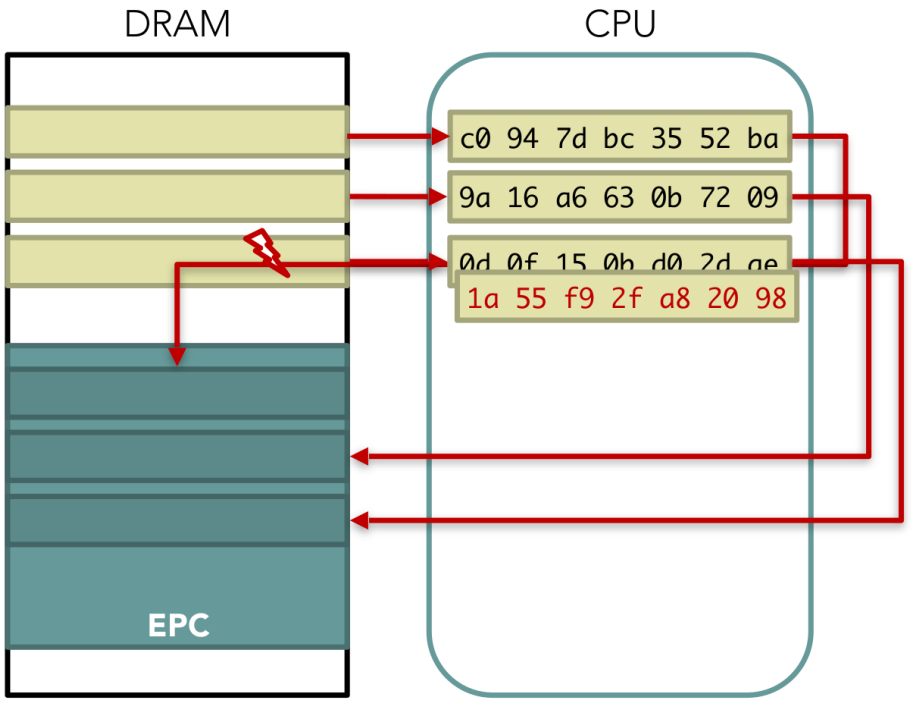
Enclave construction

- CPU calculates enclave's measurement hash during enclave construction
  - Each new page extends the hash with the page content and attributes (read/write/execute)
  - Hash computed with SHA-256
- Measurement can then be used to attest the enclave to a local or remote entity

Enclave attestation

- Is my code running on remote machine intact?
- Is code really running inside an SGX enclave?
- Local attestation
  - Prove enclave's identity (=measurement) to another enclave on the same CPU
- Remote attestation
  - Prove enclave's identity to a remote party
- Once attested, an enclave can be trusted with secrets

Enclave construction

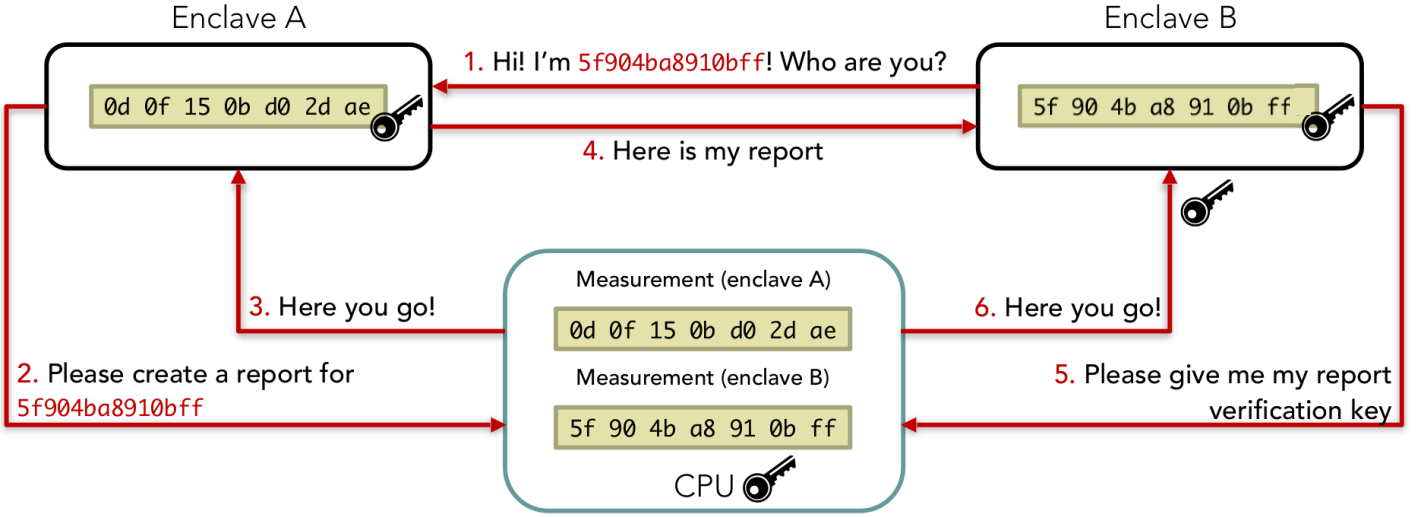


CPU calculates enclave's measurement hash during enclave construction

Different measurement if enclave is modified

Local attestation

- Prove identity of A to a local enclave B



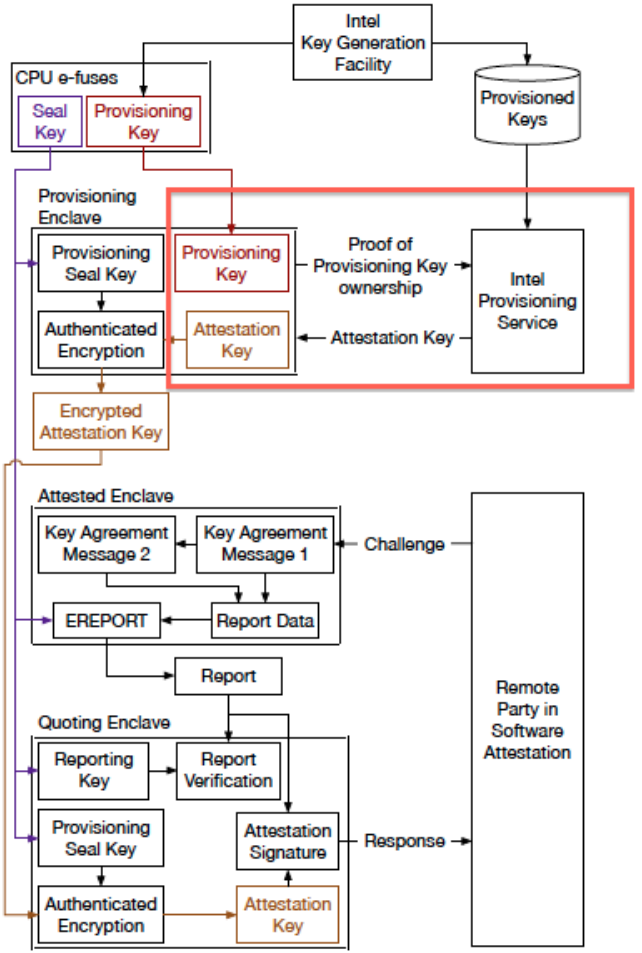
1. **Target enclave B measurement is required for key generation**
2. **Report contains information about target enclave B, including its measurement**
3. **CPU fills in the report and creates a MAC using the report key, which depends on random CPU fuses and the target enclave B measurement**
4. **Report sent back to target enclave B**
5. **Verify report by CPU to check that it was generated on the same platform, i.e., its MAC was created with the same report key (available only on the same CPU)**
6. **Check the MAC received with the report and do not trust A upon mismatch**

Remote attestation

- Transform a local report to a remotely verifiable "quote"
- Based on provisioning enclave (PE) and quoting enclave (QE)
  - Architectural enclaves provided by Intel
  - Execute locally on user platform
- Each SGX-enabled CPU has **two unique keys** fused during manufacturing
  - **Provisioning key**
    - Intel maintains a database of these keys
  - **Seal key**
  - Those keys are not used directly → derived keys

Remote attestation

- PE communicates with Intel attestation service
  - Proves it has a key installed by Intel
  - Receives asymmetric attestation key
- QE performs local attestation for enclave
  - QE verifies report and signs it using attestation key
  - Creates a quote that can be verified outside platform
- The quote and signature are sent to the remote attester, which communicates with Intel attestation service to verify quote validity



SGX limitations

- Amount of memory the enclave can use needs to be known in advance
  - Dynamic memory support in SGX v2
- Security is not perfect
  - Vulnerabilities within the enclave can still be exploited
  - Side-channel attacks are possible
- Performance bottlenecks
  - Enclave entry/exit is costly
  - Paging is very expensive
- Application partitioning? Legacy code? ...

Cloud SGX Presentation

- SGX-Aware Container Orchestration for Heterogeneous Clusters

- Stress-SGX: Load and Stress your Enclaves for Fun and Profit

- Presented by Sébastien Vaucher, PhD student at UNINE

Conclusion

- Security of applications in the Cloud require
  - Using good practices as for in-premises IT
  - Together with new techniques for Cloud-specific attacks
- Colocation and increased trusted code based
  - More attack opportunities
  - Must trust Cloud provider and large code base
- Store only encrypted data in the Cloud: yes but...
  - ... encrypted processing OK in some applications, too costly in others — security balance with expressiveness
- Trusted Execution Environment = HW support for reducing trusted code base to application enclaves

References

- Top Threats to Cloud Computing: Deep Dive - CSA
- The Treacherous 12 - Cloud Computing Top Threats in 2016 - CSA
- Can Homomorphic Encryption be Practical, Lauter, Naehrig, Vaikuntanathan
- All Your Clouds are Belong to us – Security Analysis of Cloud Management Interfaces, Somorovsky et al. - 2011
- Exploiting the DRAM rowhammer bug to gain kernel privileges, Seaboard, Dullien - 2015
- Meltdown: Reading Kernel Memory from User Space, Lipp et al. - 2018
- Intel SGX Explained, Costan, Devadas - 2016