15. Hardware Security

Hardware Features Designed for Security

SMEP/SMAP: Supervisor Mode Execution Prevention/Supervisor Mode Access Prevention

Return-to-user Attack: 内核代码中某些函数指针被设为NULL，dereference时跳转到用户代码。通过SMEP/SMAP禁止内核访问/执行用户代码。

- SMEP: Allows pages to be protected from supervisor-mode instruction fetches. If SMEP = 1, OS cannot fetch instructions from application
- SMAP: Allows pages to be protected from supervisor-mode data accesses. If SMAP = 1, OS cannot access data at linear addresses of application
- ARM的类似技术：PAN, Privileged Access Never; PXN, Privileged eXecute Never; UAO, User Access Only

应用：使用SMAP做进程内隔离，将用户代码放在内核态，需要保护的数据放在用户态

ret2dir Attack: 每一页物理内存都同时拥有kernel和user两个虚拟地址，虽然禁止内核执行用户代码，但通过跳转到恶意代码对应的kernel地址仍可攻击。

MPX: Memory Protection eXtension

Background: C/C++ bounds error (可通过gcc的-fcheck-pointer-bounds flag防止)

Intel introduces MPX since Skylake

- Specified by two 64-bit addresses specifying the beginning and the end of a range
- New instructions are introduced to efficiently compare a given value against the bounds, raising an exception when the value does not fall within the permitted range
    - bndmov: Fetch the bounds information (upper and lower) out of memory and put it in a bounds register.
    - bndcl: Check the lower bounds against an argument (%rax)
    - bndcu: Check the upper bounds against an argument (%rax)
    - bnd retq: Not a "true" Intel MPX instruction
- usage: make CFLAGS="-mmpx -fcheck-pointer-bounds -lmpx" LDFLAGS="-lmpxwrappers -lmpx"
- 将bounds存在bounds table(a two-level radix tree)中，最坏情况可多消耗400%内存，且使用较多bound时性能降低

MPK: Memory Protection Keys

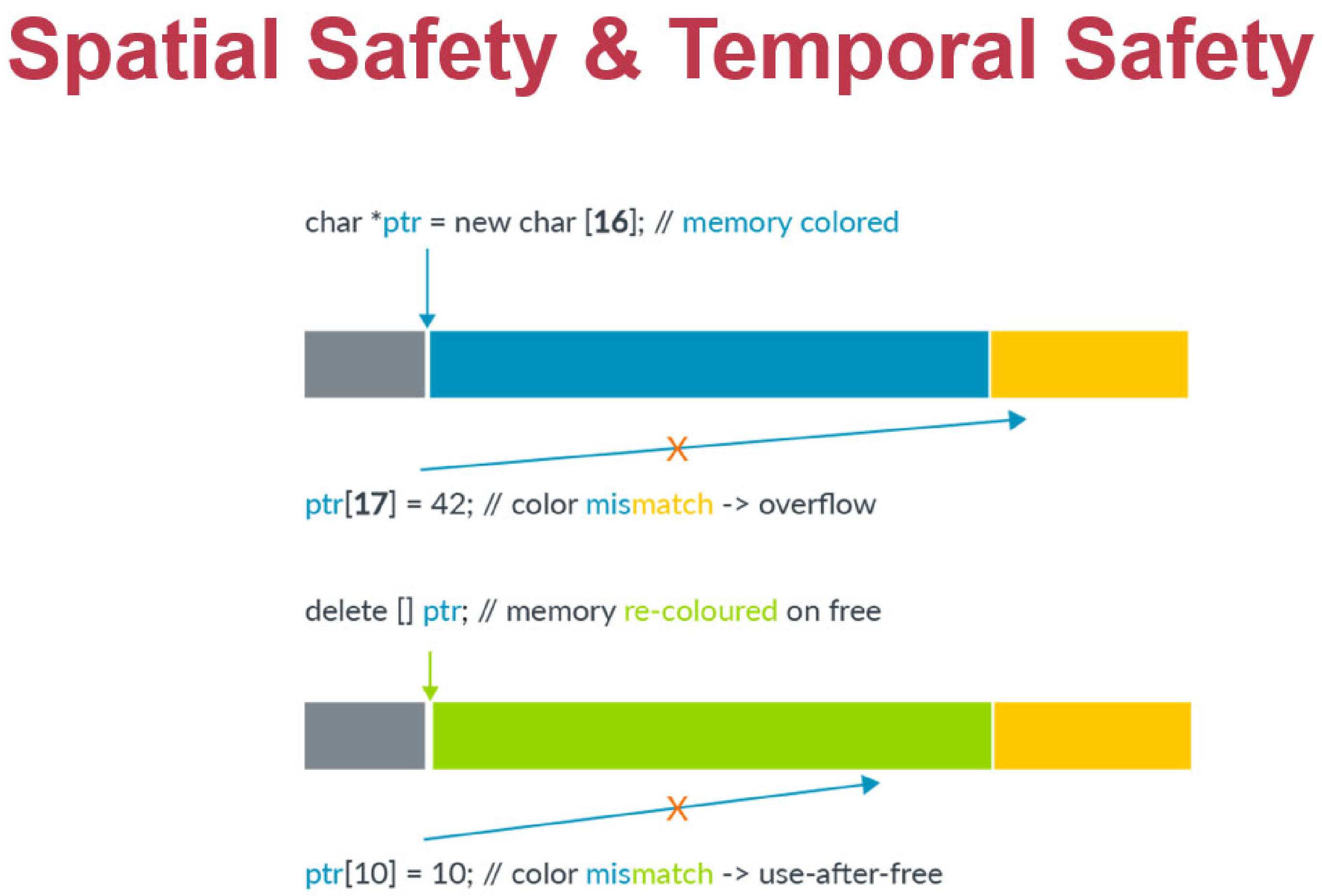
- With MPK, every page belongs to one of 16 domains. A domain is determined by 4 bits in every page-table entry (referred to as the protection key)
- For every domain, there are two bits in a special register (pkru), which denote whether pages associated with that key can be read or written
- Only the kernel can change the key of a page, application can read and write the pkru register using the rdpkru and wrpkru instructions respectively
    - use case 1: protect critical data within one address space
    - use case 2: prevent data corruption (In-memory database prevents writes most of the time, only enable changing data when needs to change)

ARM PA: Pointer Authentication

- ARM只使用了64位中的40位，可以用key对这40位地址进行加密，保存在前24位，从而作为验证，防止地址被篡改
- PA defines five keys
    - Four keys for PAC* and AUT* instructions (combination of instruction/data and A/B keys),
    - One key for use with the general purpose PACGA instruction

- Keys are stored in internal registers and are not accessible by EL0 (user mode)
- New instructions:
    - PAC value creation: Write the value to the uppermost bits in a destination register alongside an address pointer value
    - Authentication: Validate a PAC and update the destination register with a correct or corrupt address pointer. If the authentication fails, an indirect branch or load that uses the authenticated, and corrupt, address will cause an exception

ARM MTE: Memory Tag Extension



- 引入新的内存类型：Normal Tagged Memory，只允许指针访问相同tag的内存
- tag占4位，即0~15
- 不是所有内存访问都需要tag checking，如instruction fetches,translation table walks等
- MTE and PA可结合使用

Intel CET: Control-flow Enforcement Technology

- Shadow stack: a second stack for the program
    - 该栈只记录控制数据(return address), 与原有的stack同时push和pop，如果RET时两个地址不一样，则引发control protection exception
    - 被页表保护
- Indirect Branch Tracking: New instruction: ENDBRANCH
    - call/jmp跳转到的指令必须以ENDBRANCH开头，否则无效
    - cpu维护了一个状态机跟踪indirect call/jmp, 当出现这些指令，状态由IDLE转为WAIT_FOR_ENDBRANCH, 该状态下下一条指令必须为ENDBRANCH，否则报错

Trusted Execution Environment

XOM: eXecute-Only Memory

- 代码和数据在内存中加密
- 存储加密值的哈希

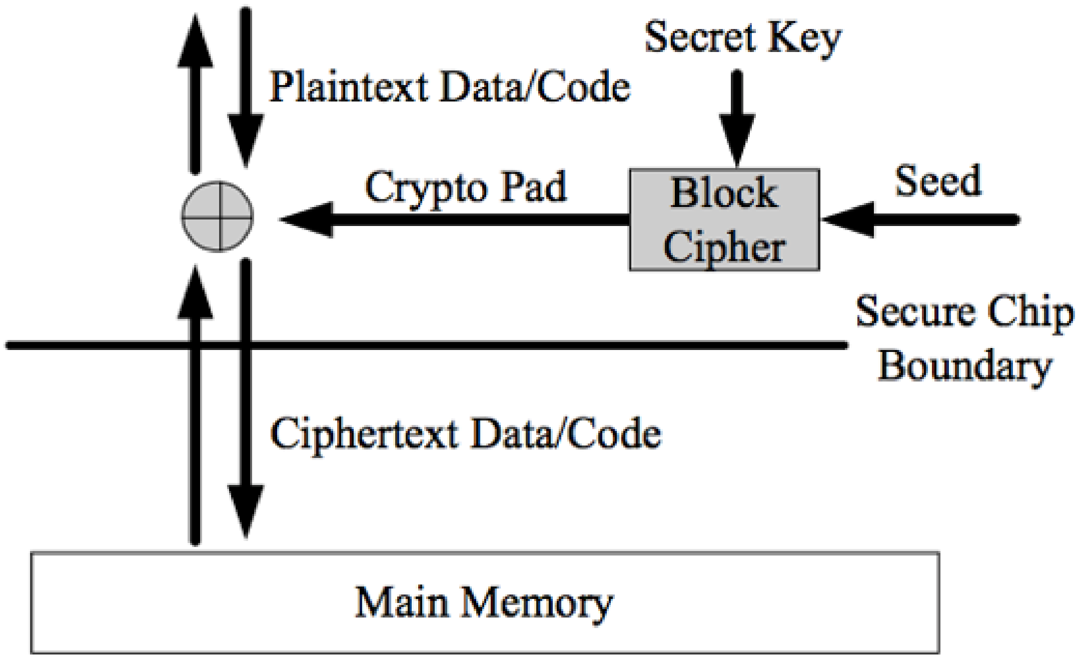
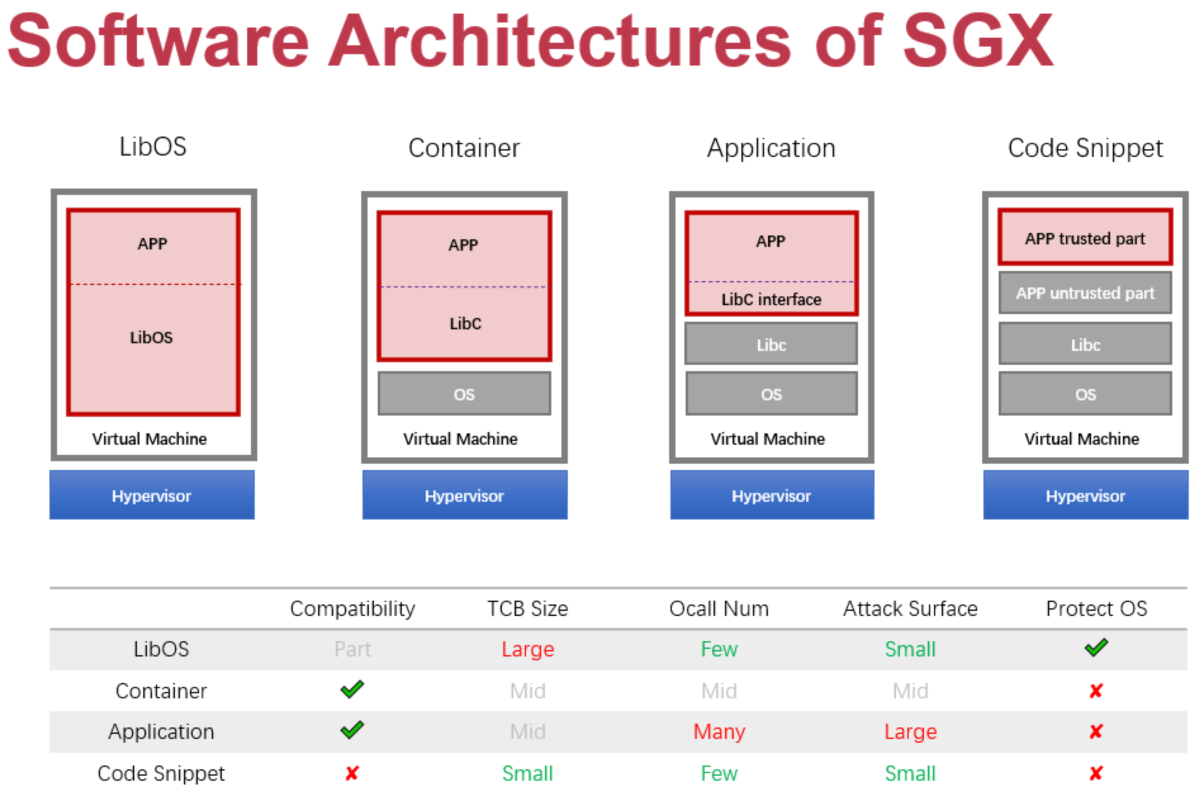


Figure 1. Counter-mode based memory encryption.

-

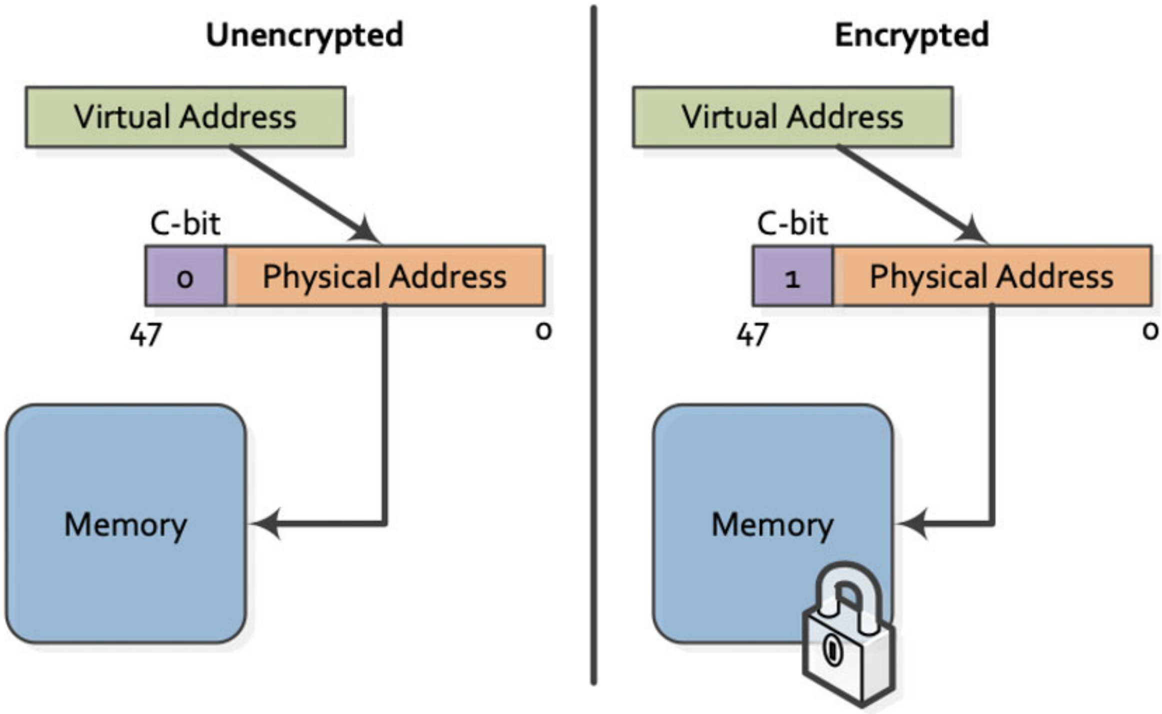
Intel SGX

- SGX Execution Flow
  - App built with trusted and untrusted parts
  - App runs & creates the enclave which is placed in trusted memory
  - Trusted function is called, execution transitioned to the enclave
  - Enclave sees all process data in clear; external access to enclave data is denied
  - Trusted function returns; enclave data remains in trusted memory
  - Application continues normal execution

- 

	Compatibility	TCB Size	Ocall Num	Attack Surface	Protect OS
LibOS	Part	Large	Few	Small	✔
Container	✔	Mid	Mid	Mid	✘
Application	✔	Mid	Many	Large	✘
Code Snippet	✘	Small	Few	Small	✘

AMD's SME/SEV: Secure Memory Encryption/Secure Encrypted Virtualization

- Features
  - 

  - Hardware AES engine located in the memory controller performs inline encryption and decryption of DRAM
  - Minimal performance impact: Extra latency only taken for encrypted pages
  - No application changes required
  - Encryption keys are managed by the AMD Secure Processor and are hardware isolated, not known to any software on the CPU
- Comparing with Intel SGX
  - SME不会防范内核，用来防御cold-boot attacks，以及非易失内存的数据泄露

    - SEV专注于虚拟机，可以防御其他虚拟机以及宿主机
- 类似技术: Intel MKTME: Multi-Key Total Memory Encryption, 支持多个key加密

ARM TrustZone

- Two modes: Normal world (REE, rich execution environment) and secure world(TEE, trusted execution environment), SMC instruction to switch
- 总线上增加1位，外设可以区分请求来自哪个world
- 应用: 手机指纹识别、交通工具、无人机禁飞区

RISC-V PMP/sPMP: Physical Memory Protection

- 通过一组PMP registers将物理内存划分为互相隔离的区域，每一段属于一个enclave
- 由于PMP registers数量有限，enclave数量也受限

Penglai

- Enclave on RISC-V ISA
- 为了实现细粒度内存划分，在DRAM增加一个bitmap，每一位表示一个页是否安全
- 所有不安全的页存储在一个隔离的内存区域PT_AREA
- 使用cache partition防御侧信道攻击
- 通过签名证明确实运行在enclave

Hardware Features Not Designed for Security

Intel TSX: Transactional Synchronization eXtensions

- Programming with RTM(restricted transactional memory)
    - If transaction starts successfully, do work protected by RTM, and then try to commit
    - If abort, system rollback to _xbegin, return an abort code
    - Manually abort inside a transaction
- 使用HTM保护数据: 将数据放在transaction中，利用HTM保证的原子性防止其他并发访问
- 利用HTM攻击KASLR: KASLR技术用来随机化内核地址。用户态随机访问一个地址，有两种情况，一是未映射，二是内核地址空间，两者返回segmentation fault有时间差。将这种试探代码放在transaction中，abort速度极快，可以快速试探出内核地址

Intel CAT: Cache Allocation Technology

The "Noisy Neighbor" Problem: 某些应用使用大量内存，但本身对cache需求不高（例如流媒体播放，之前的帧没有用处），反而占据了其他需要利用cache的应用的cache容量，影响性能。

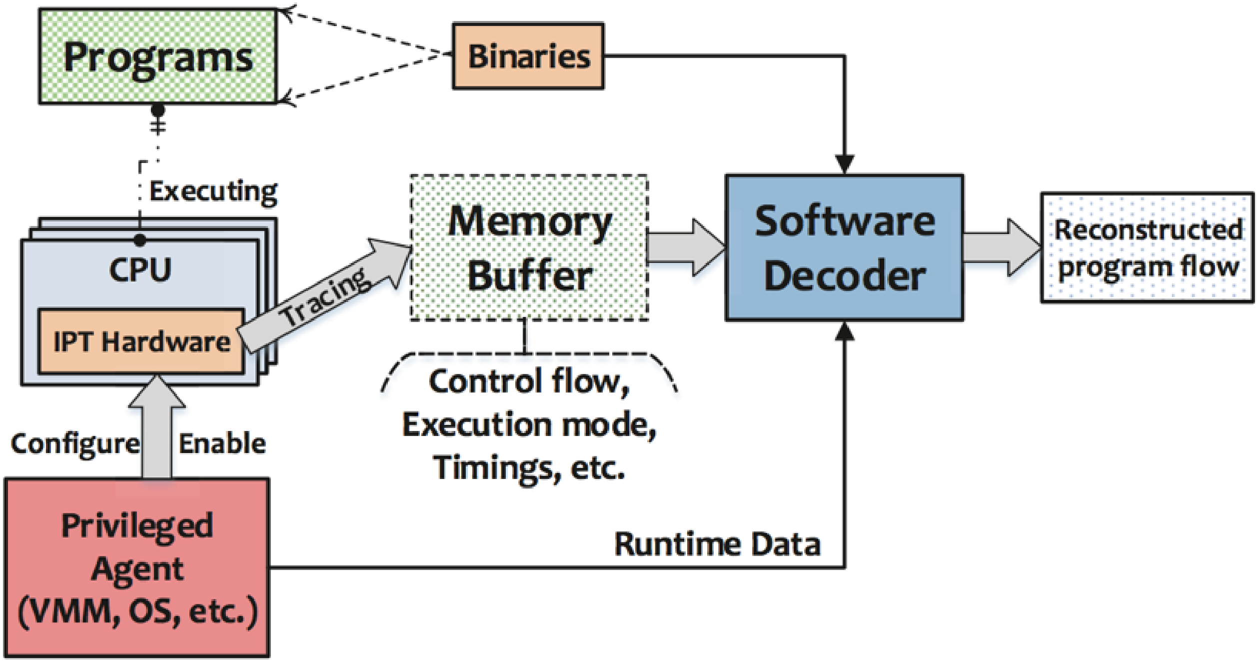
- CAT将thread / app / VM / container通过Class of Service (CLOS)分组，每个CLOS有相应的resource capacity bitmasks (CBMs)，表示可以使用cache的哪一部分
- 通过CAT防御基于cache的侧信道攻击（The PRIME+PROBE Attack，先用随机数据将cache填满，然后触发加密程序，最后重新访问原数据，通过cache miss情况推算加密行为），将不可信应用隔离，只能使用部分cache

PMU: Performance Monitor Unit

- BTS: Branch Trace Store, 记录程序所有地址跳转
- Motivation: Code Injection Attack, Code Reuse Attack
- 做法：利用PMU监测CFI
    - Offline phase: 记录所有可能的分支跳转
    - Online phase: 将实际跳转与合法跳转对比，发现恶意行为
    - 记录3种合法跳转
        - ret_set: all the addresses next to a call
        - call_set: all the first addresses of a function

  - train_sets: all the target addresses that once happened

Intel PT: Intel Processor Tracing

- 

- 增加了硬件对Trace进行压缩，使tracing很快，但decode很慢
- FlowGuard: 将压缩的实际trace与压缩的可信trace直接对比，如果无法判断再解压缩

16. Data Privacy

ZKP: Zero-Knowledge Proof

Problem: Alice想向Bob证明她has the answer A of the problem P，如果直接发送A，则Bob也知道了A

Zero-Knowledge Proof:

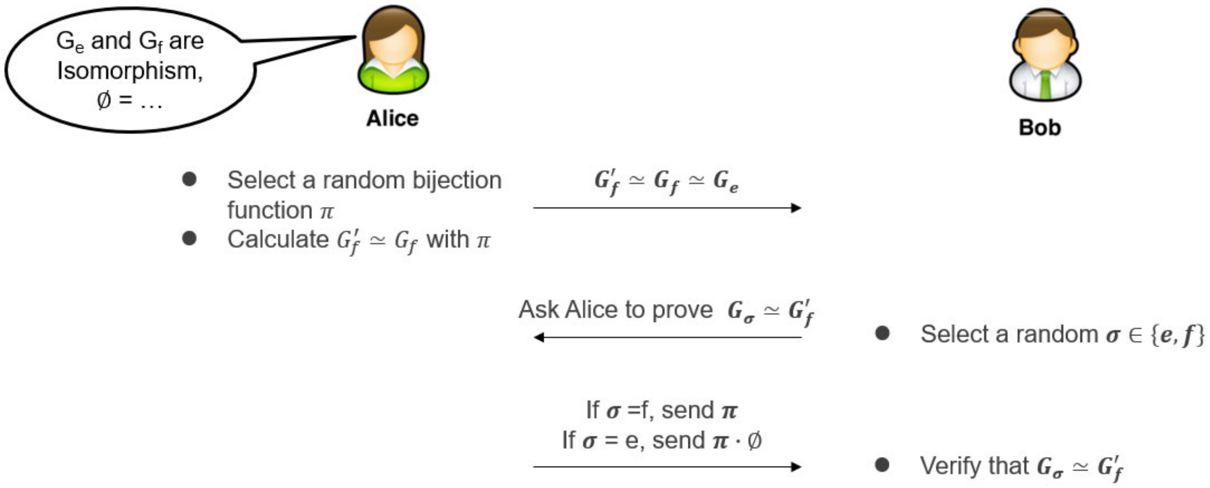
- Completeness: Alice can construct the proof if she has A
- Soundness: Alice cannot construct the proof if she doesn’t have A
- Zeroknowledge: Bob knows nothing about A

Interactive Zero-Knowledge Proof: P has answer $\boldsymbol{x}$ of a problem $\boldsymbol{L}$, and tries to prove it with > 1 iterations:

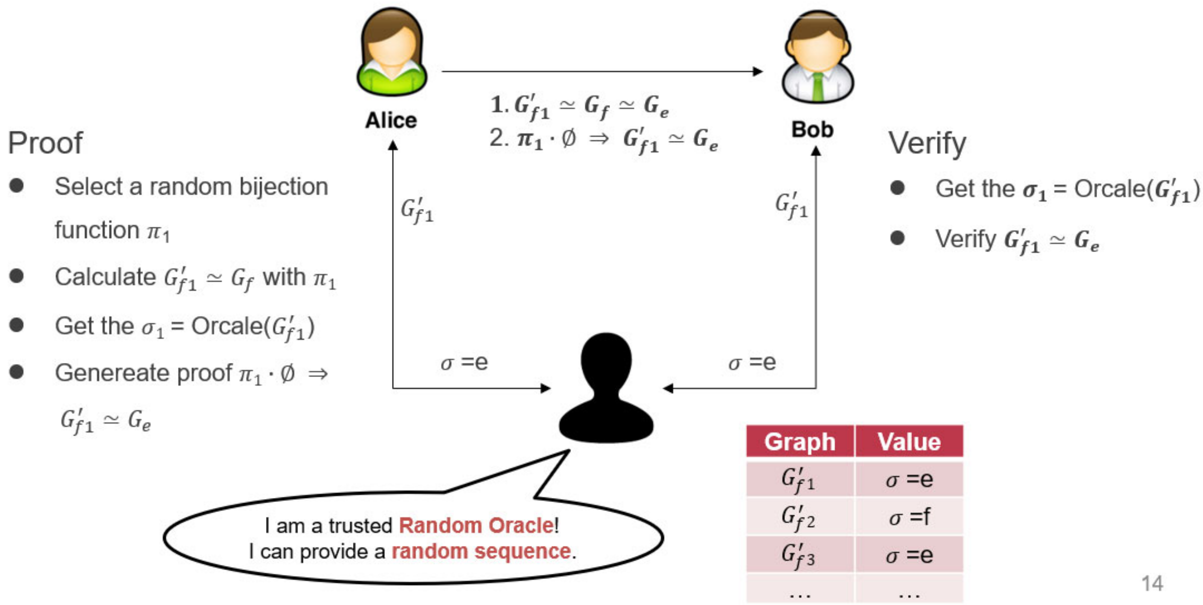
- Step-1: P transfers $\boldsymbol{L}$ to $\boldsymbol{L'}$, and promises that $\boldsymbol{L'}$ is transferred from $\boldsymbol{L}$ and she has the answer $\boldsymbol{x'}$
- Step-2: V challenges P
- Step-3: P shows the proof of the answer $\boldsymbol{x'}$, which will not leak $\boldsymbol{x}$
- V trusts that P has $\boldsymbol{x}$ when P always meets the challenge

图的同构：If G1 = (V1, E1) and G2 = (V2, E2) are isomorphic, there exist a bijection function（双射，即一一对应函数）$\boldsymbol{\phi}$, that for any $(\boldsymbol{u}, \boldsymbol{v}) \in \boldsymbol{E\_1}$, exist $\boldsymbol{\phi}(\boldsymbol{u}, \boldsymbol{v}) \in \boldsymbol{E\_2}$，即两张图的每条边都一一对应

- 基于图同构的Interactive ZKP
  - Alice生成一个与两张图都同构的新图，反复询问迭代

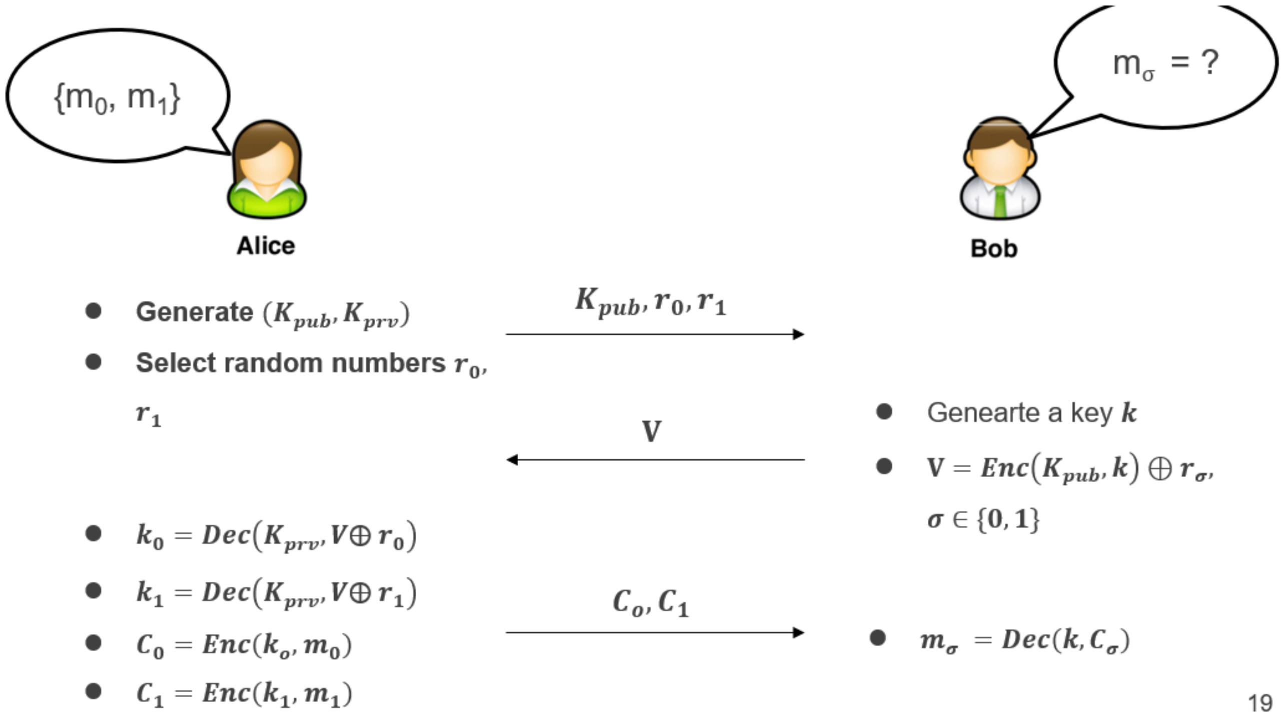
  -
- Non-Interactive ZKP
  - 有一个可信的第三方生成随机序列，让Alice一次性证明

- 

OP: Oblivious Transfer 不经意传输

Problem: sender不想让receiver拿到所有数据，receiver不想让sender知道自己想要哪个数据

1-out-of-2 OT:

- 

- Bob把自己的key用Alice的公钥加密，并与r_i作异或
- Alice分别用两个解密结果加密她的两份消息，将加密结果送给Bob（只有Bob选择的i才能解密出Bob的key，且Alice不知道Bob选择了哪个）
- Bob用k解密两份密文，得到需要的消息，而另一份无法解密

HE: Homomorphic Encryption 同态加密

Problem: 想把数据放在云端计算，但想加密

Solution: 密文可以直接运算，返回结果后用户解密即可。但目前算法难以支持所有密文运算，例如RSA只支持乘法

SMPC: Secure Multi-Party Computing

Problem: 多方拥有不同数据，想用这些数据共同计算而又不将数据泄露给其他方

Yao's Protocol: GC(Garbled Circuits)+OT(Oblivious Transfer), 姚期智的混淆电路+不经意传输算法

TEE: Trusted Execution Environmnet

- Software TEE
  - VM-based TEE
  - Same privilege protection
- ARM TrustZone
- Intel SGX
- AMD SME/SEV
- Penglai, SANCTUM

DP: Differential privacy 差分隐私

Problem: 想要拒绝用户访问数据库单个条目，但不能简单限制，否则可以通过sum(*) - sum(where != xx)反推出

Solution: （若两个数据集有且仅有一条数据不一样，则称此二者为相邻数据集）如果某算法作用于任何相邻数据集，得到一个特定输出的概率应差不多，那么我们就说这个算法能达到差分隐私的效果。也就是说，观察者通过观察输出结果很难察觉出数据集一点微小的变化，从而达到保护隐私的目的。差分度越低，安全性越高。

- 实现方式：向计算函数中加噪声

FL: Federated Learning 联邦学习

Problem: 多方训练同一个模型，需要保证各方数据隐私

- 横向：一方拥有一个样本的全部数据
  - 服务器分发子模型给每个用户，用户提交update，会泄露个人隐私
  - 用户向update中添加噪声，并保证总体对称，相互抵消，不影响总模型更新
  - 如果有用户掉线，服务器必须向其他用户询问该用户的噪声偏移量；则服务器可以伪造用户掉线，通过询问计算出该用户原本的update
  - Secret Sharing & Double Masking算法
    - Share a secret S among N nodes, T nodes can reconstruct the secret，基于k次函数方程求解需要k以上个点的原理
    - Each user u generates a_u and $S_{u,v}$ (for each user pair (u,v))
    - Each user u updates y_u，是关于a和S的函数
    - Server计算$\sum y_i$
      - For online node u, server asks other nodes to get the secret share of the a_u
      - For offline node u, server asks other nodes to get the secret share of the $S_{u,i}$
      - 所有节点防止server同时得到a_u and $S_{u,i}$
- 纵向：每一方拥有同一个样本的部分数据
  - 每方分别单独训练，在全连接层汇总，由有label的一方算出最终模型

后面介绍了3篇论文，是对以上技术的具体应用

- Oblivious Multi-Party Machine Learning on Trusted Processors (Security’2016)
- BatchCrypt: Efficient Homomorphic Encryption for Cross-Silo Federated Learning (ATC’20)
- Privacy Accounting and Quality Control in the Sage Differentially Private ML Platform (SOSP’19)