第四届格与全同态加密国际前沿论坛

可信硬件与密码学的一些思考

王文浩

2021年1月

自我介绍

中国科学院信息工程研究所
INSTITUTE OF INFORMATION ENGINEERING,CAS

王文浩

信息工程研究所，副研究员，硕士生导师

研究方向：

系统安全/隐私计算/侧信道攻防/密码学

主页：https://heartever.github.io

邮箱：wangwenhao@iie.ac.cn

欢迎联系！

隐私计算技术的门派之别

密码学

- 安全多方计算
- 同态加密
- 零知识证明

其它

- 联邦学习
- 差分隐私

可信执行环境(TEE)

- **Intel SGX**
- **AMD SEV/SEV-ES/SEV-SNP**
- **ARM TrustZone/**
- **Intel TDX**
- **RISC-V Keystone/Penglai**

隐私计算技术的门派之别

- 密码学技术
  - 基于困难问题和假设的严格数学证明
  - 性能差（计算、网络传输能力要求高）

- 联邦学习和差分隐私
  - 允许部分信息泄露
  - 性能优

- 可信执行环境技术（TEE）
  - 基于特殊的硬件隔离机制
  - 性能优
  - 微体系结构侧信道
  - 远程认证过程依赖可信第三方（Intel/AMD/ARM）

本报告的目的

- 抛弃门派之见

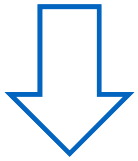
- 隐私计算 千秋万载 一统江湖

从容易认可的“公理”出发

- 信任应当建立在尽可能小的假设之上
  - 安全性规约
  - 信任链
- 安全是相对的，绝对的安全往往是没有实际意义的
  - 安全不能脱离现实的应用场景
  - 威胁模型的定义需要真实刻画现实场景
- 设计和实现是分离的
  - 实现的问题往往比较容易修复
  - 设计的问题往往是根本性问题
- 可证明安全非常重要

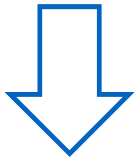
讨论1：信任应当建立在尽可能小的假设之上

- 密码学的信任基础
  - Hard problems
  - 严格的形式化定义和安全性规约



- 后量子密码算法
- 量子计算机下安全的困难性问题

- TEE的信任基础
  - Root of trust
  - 安全启动、度量和信任链
  - Remote attestation



- 如何避免远程认证过程芯片制造商成为唯一信任源？
  - Threshold cryptography
  - [1] Y. Lindell et al. Fast Secure Multiparty ECDSA with Practical Distributed Key Generation and Applications to Cryptocurrency Custody

讨论2：威胁模型的定义需要真实刻画现实场景

- 本地环境
  - 无物理接触
  - 远程网络连接
  - MPC的参与方，HE的加密、解密
  - 使用云端TEE的用户的本地环境（可以没有TEE）

- 远程环境
  - 例如公有云
  - （有限制的）物理接触
  - 远程网络连接
  - HE的计算
  - 云端TEE

- 未经过（形式化）验证的系统软件不可信
  - 例如Rich OS、Linux、Xen等
- 验证过的、内存安全的系统软件或固件
  - 例如运行在trustzone/hypervisor里的seL4

讨论2：威胁模型的定义需要真实刻画现实场景（cont.）

- 远程环境的威胁模型
- Intel SGX等TEE方案
  - 有限制的物理攻击
    - 软件电压、频率控制，冷启动，总线窃听
    - 不考虑操纵电压的物理攻击方式
  - Malicious威胁模型
  - 暂时没有考虑侧信道

- 密码学方案
  - 不信任远程环境硬件和系统软件
  - 往往假定semi-honest或non-colluding威胁模型

讨论3：设计 *vs.* 实现

- 设计
  - 文献[2]证明了SGX的语义满足机密性、完整性和安全度量属性
  - 基于SIGMA协议的EPID远程认证协议
- 实现
  - 传统微体系结构侧信道
    - 页表、cache等[3,4]
    - 文献[5-6]表明侧信道问题是可以以不高的代价进行弥补的
  - 瞬态执行侧信道：Foreshadow、RIDL等
    - 通过微码或下一代硬件修复
  - Enclave ECALL/OCALL接口[7,8]
  - 内存安全问题[9-11]

[2] A Formal Foundation for Secure Remote Execution of Enclaves. Subramanyan et al. CCS 2017
[3] Leaky Cauldron on the Dark Land: Understanding Memory Side-Channel Hazards in SGX. Wang et al. CCS 2017
[4] Bluethunder: A 2-level Directional Predictor Based Side-Channel Attack against SGX. Huo et al. CHES 2020
[5] Racing in Hyperspace: Closing Hyper-Threading Side Channels on SGX with Contrived Data Races. Chen et al. S&P 2018
[6] Randomized Last-Level Caches Are Still Vulnerable to Cache Side-Channel Attacks! But We Can Fix It. Song et al. S&P 2021
[7] A Tale of Two Worlds: Assessing the Vulnerability of Enclave Shielding Runtimes. Bulck et al. CCS 2019
[8] COIN Attacks: On Insecurity of Enclave Untrusted Interfaces in SGX. Khandaker et al. ASPLOS 2020

讨论4：可证明安全、形式化验证

- TEE设计和实现的形式化验证
  - 指令集、语义的形式化验证[2]
  - RA的形式化验证[12]
  - 接口的形式化验证[13]
  - TCB的形式化验证[14]
  - 微体系结构的形式化定义[15]
  - 实现代码的形式化验证[16]
  - 内存安全语言[17]
- 形式化验证方法
  - Theorem proving
  - Model checking
  - Abstract interpretation
- 形式化验证工具
  - Proof assistant: Coq/Isabelle
  - Automatic theorem provers: SAT
  - SPIN
  - *etc.*

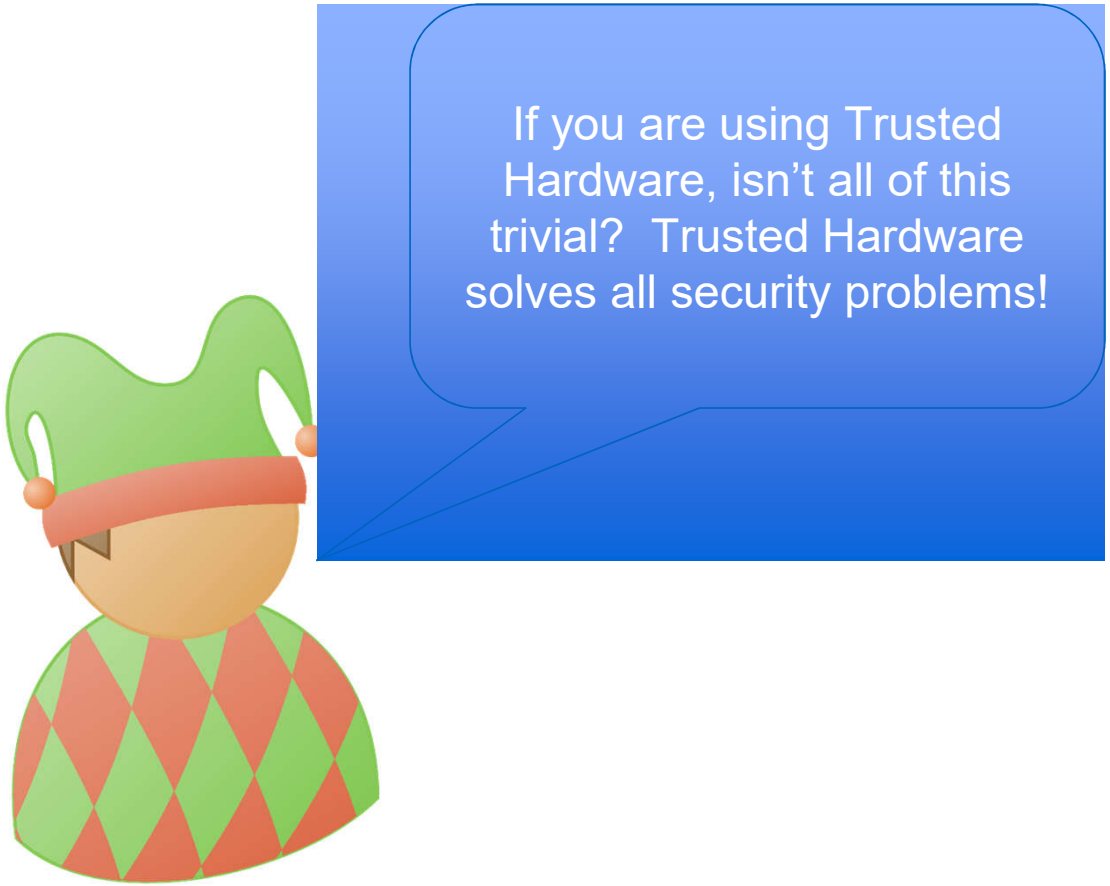
[12] VRASED: A Verified Hardware/Software Co-Design for Remote Attestation
[13] BesFS: A POSIX Filesystem for Enclaves with a Mechanized Safety Proof. Shinde et al. Usenix Security 2020
[14] The seL4® Microkernel
[15] A Formal Approach to Secure Speculation. Cheang et al. CSF
[16] Moat: Verifying Confidentiality of Enclave Programs. Sinha et al. CCS 2015
[17] Towards Memory Safe Enclave Programming with Rust-SGX. Want et al. CCS 2019

硬件机制能否与密码学等技术结合？

- 多种安全计算技术的融合
- 密码技术性能的瓶颈
  - *MPC中Multiplication Triple的生成*[18]
  - *GC中Garbled Table的传输*[19]
  - *HE中的bootstrapping和除法/指数/Sigmoid等非线性计算*[20]
  - *Simi-honest模型 → Malicious模型*[21]
  - *基于可信硬件的fair multiparty computation*[22]
- 问题：如何避免因引入可信硬件造成的信任弱化
  - *清晰的场景定义和威胁模型定义*

[18] Ad-Hoc Secure Two-Party Computation on Mobile Devices using Hardware Tokens. Demmler et al. Usenix Security 2014
[19] Silent Two-party Computation Assisted by Semi-trusted Hardware. Lu et al. Arxiv. 2020
[20] Toward Scalable Fully Homomorphic Encryption Through Light Trusted Computing Assistance. Wang et al. Arxiv. 2020
[21] CRYPTFLOW: Secure TensorFlow Inference. Kumar et al. eprint. 2019
[22] Fairness in an Unfair World: Fair Multiparty Computation from Public Bulletin Boards. Choudhuri et al. CCS 2017

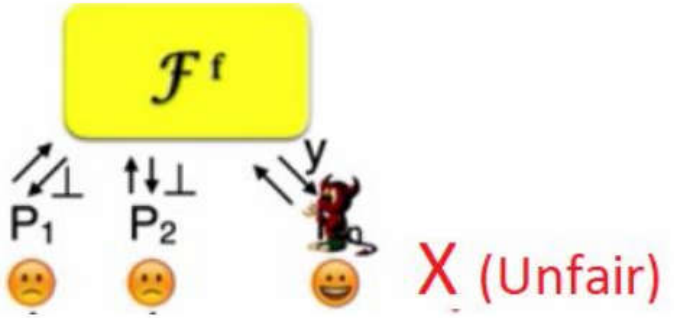
可信硬件?

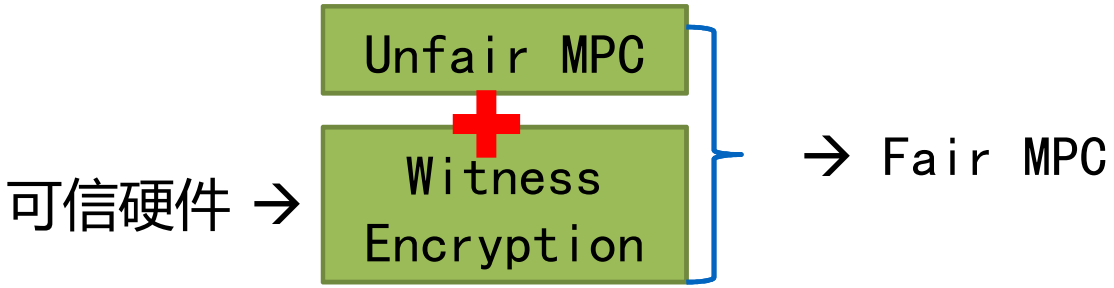

If you are using Trusted Hardware, isn’t all of this trivial? Trusted Hardware solves all security problems!

Fairness in an Unfair World: Fair Multiparty Computation from Public Bulletin Boards （CCS 2017）

Security Property of MPC: Fairness

Definition (Informal)
An adversary can receive their output only if all honest parties receive output.





Ad-Hoc Secure Two-Party Computation on Mobile Devices using Hardware Tokens (Usenix Security 2014)

GMW Protocol

TECHNISCHE UNIVERSITÄT DARMSTADT

Goldreich-Micali-Wigderson, 1987

XOR sharing to mask values: $x = x_A \oplus x_B$

Local evaluation of XOR gates

Interactive evaluation of AND gates

Using pre-computed Multiplication Triples

HOW TO PLAY ANY MENTAL GAME

or

A Completeness Theorem for Protocols with Honest Majority

(Extended Abstract)

Oded Goldreich — Dept. of Computer Sc., Technion, Haifa, Israel

Silvio Micali — Lab. for Computer Sc., MIT, Cambridge, MA 02139

Avi Wigderson — Inst. of Math. and CS, Hebrew University, Jerusalem, Israel

Abstract

We present a polynomial-time algorithm that, given as a input the description of a game with incomplete information and any number of players, produces a protocol for playing the game that leaks no partial information, provided the majority of the players is honest.

Our algorithm automatically solves all the multi-party protocol problems addressed in complexity-based cryptography during the last 10 years. It actually is a completeness theorem for the class of distributed protocols with honest majority. Such completeness theorem is optimal in the sense that, if the majority of the players is not honest, some protocol problems have no efficient solution[3].

1. Introduction

Before discussing how to "make playable" a general game with incomplete information (which we do in section 6) let us address the problem of making playable a special class of games, the *Turing machine games* (*Tm-games* for short).

Informally, n parties, respectively and individually owning secret inputs $x_1,\ldots,x_n$, would like to *correctly* run a given Turing machine M on these x_i's while keeping the maximum possible privacy about them. That is, they want to compute $y=M(x_1,\ldots,x_n)$ without revealing more about the x_i's than it is already contained in the value y itself. For instance, if M computes the sum of the x_i's, every single player should not be able to learn more than the sum of the inputs of the other parties. Here M may very well be a probabilistic Turing machine. In this case, all players want to agree on a single string y, selected with the right probability distribution, as M's output.

The correctness and privacy constraints of a Tm-game can be easily met with the help of an extra, trusted party P. Each player i simply gives his secret input x_i to P. P will privately run the prescribed Turing machine, M, on these inputs and publicly announce M's output. Making a Tm-game playable essentially means that the correctness and privacy constraints can be satisfied by the n players themselves, without invoking any extra party. Proving that Tm-games are playable retains most of the flavor and difficulties of our general theorem.

2. Preliminary Definitions

2.1 Notation and Conventions for Probabilistic Algorithms.

We emphasize the number of inputs received by an algorithm as follows. If algorithm A receives only one input we write "$A(\ )$", if it receives two inputs we write $A(\ ,\ )$ and so on.

RV will stand for "random variable"; in this paper we only consider RVs that assume values in

Work partially supported by NSF grants DCR-8509916 and DCR-8411917, an IBM post-doctoral fellowship and an IBM faculty development award. The work was done when the first author was at the Laboratory for Computer Science at MIT; and the second author at the mathematical Sciences Research Institute at UC-Berkeley.

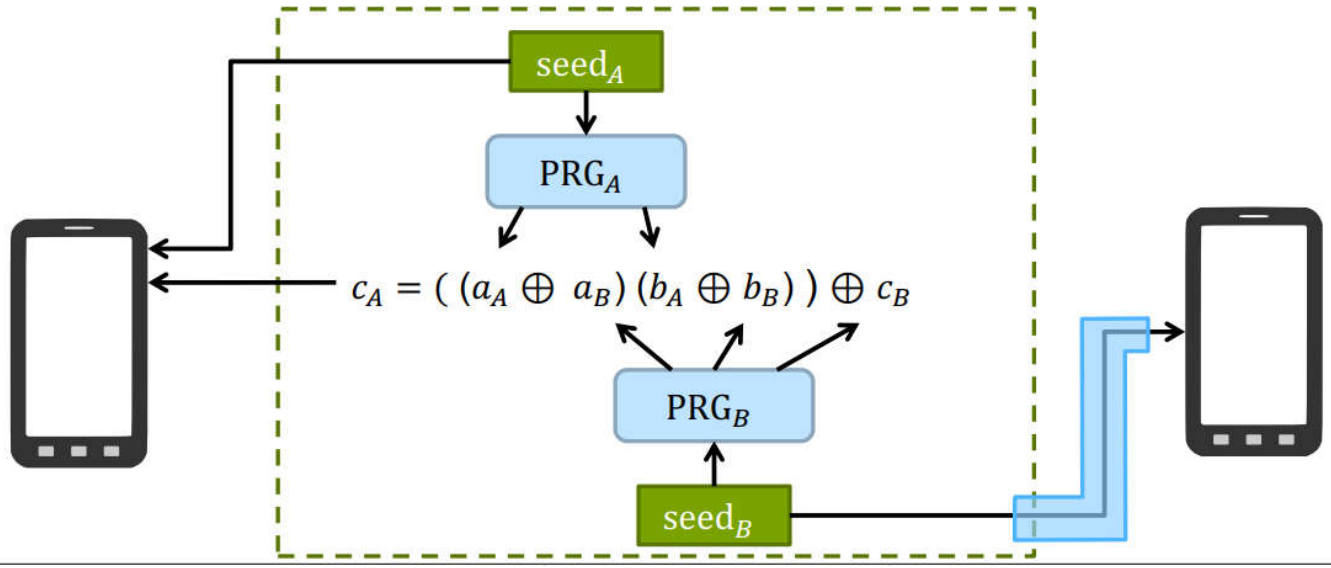
Ad-Hoc Secure Two-Party Computation on Mobile Devices using Hardware Tokens (Usenix Security 2014)

Multiplication Triple Generation

TECHNISCHE UNIVERSITÄT DARMSTADT

Multiplication Triple (MT): $(c_A \oplus c_B)=(a_A \oplus a_B)(b_A \oplus b_B)$

Shares intended for only one party: $a_A, b_A, c_A \Leftrightarrow a_B, b_B, c_B$



我们的一个初步尝试

- 基于现有硬件加速同态加密中的bootstrapping计算

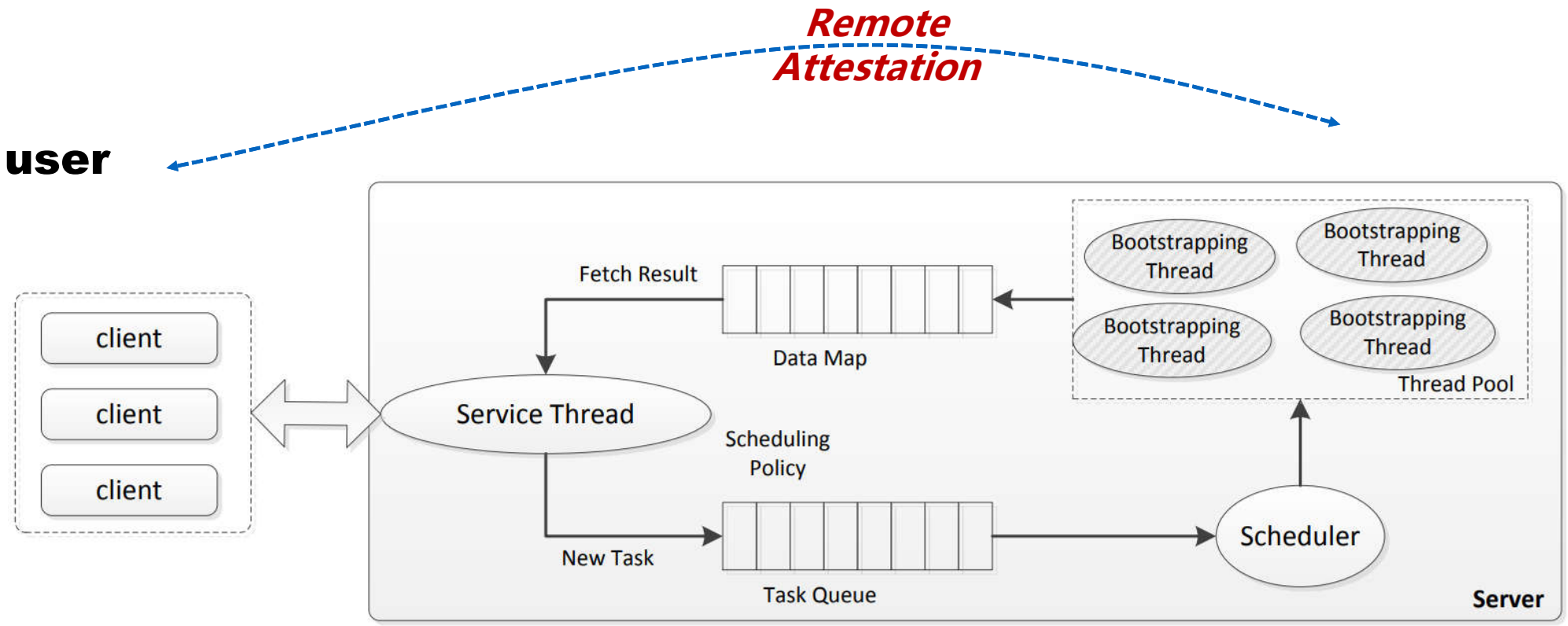
Toward Scalable Fully Homomorphic Encryption Through Light Trusted Computing Assistance

Wenhao Wang, Yichen Jiang, Qintao Shen, Weihao Huang, Hao Chen, Shuang Wang, XiaoFeng Wang, Haixu Tang, Kai Chen, Kristin Lauter and Dongdai Lin

Abstract—It has been a long standing problem to securely outsource computation tasks to an untrusted party with integrity and confidentiality guarantees. While fully homomorphic encryption (FHE) is a promising technique that allows computations performed on the encrypted data, it suffers from a significant slow down to the computation. In this paper we propose a hybrid solution that uses the latest hardware Trusted Execution Environments (TEEs) to assist FHE by moving the bootstrapping step, which is one of the major obstacles in designing practical FHE schemes, to a secured SGX enclave. TEEFHE, the hybrid system we designed, makes it possible for homomorphic computations to be performed on smaller ciphertext and secret key, providing better performance and lower memory consumption. We make an effort to mitigate side channel leakages within SGX by making the memory access patterns totally independent from the secret information. The evaluation shows that TEEFHE effectively improves the software only FHE schemes in terms of both time and space.

我们的一个初步尝试

- 基于现有硬件加速同态加密中的bootstrapping计算



我们的一个初步尝试

- 基于现有硬件加速同态加密中的bootstrapping计算
  - *Intel Xeon E3-1280 v5, 3.7 GHz, 64 GB 内存*
  - *SEAL v2.3*

n	2048	4096	8192	16384	32768
encryption	2323	4704	12809	36322	119422
decryption	419	878	3802	13431	50933
addition	17	62	330	1126	4258
multiplication	3839	7886	38088	154495	694098
square	2818	5743	28571	113490	522905
relinearization	465	989	9012	51738	348991
software only bootstrapping	-	-	-	2.09×10^9	5.20×10^{10}
SGX bootstrapping	5454	11254	43750	1.43×10^5	1.44×10^6

我们的一个初步尝试

- 避免侧信道泄露
  - 秘密依赖的访问模式

File	Function	Changes
util/uintarithsmallmod.cpp; util/uintarithmod.cpp	exponentiate_uint_mod	rewrite `if` statement with `cmov` instructions
encryptor.cpp; rnsencryptor.cpp	Encryptor::preencrypt; RNSEncryptor::rns_preencrypt	rewrite `if` statement with `cmov` instructions
decryptor.cpp; rnsdecryptor.cpp	Decryptor::decrypt; RNSDecryptor::rns_decrypt	rewrite `if` statement with `cmov` instructions
smallntt.h	inverse_ntt_negacyclic_harvey	rewrite `if` statement with `cmov` instructions
encryptor.cpp	Encryptor::preencrypt	resize the destination to size `coeff_count`

我们的一个初步尝试

- 避免侧信道泄露
  - 秘密依赖的访问模式

```
1  // size: number of qwords (8 bytes)
2  void __attribute__ ((noinline)) conditional_mov(void *
       source, void *dest, uint64_t size, uint64_t cond) {
3      __asm__ __volatile__ (
4          "movq %1, %%rax\n"
5          "movq %2, %%rbx\n"
6          "movq $0, %%rdx\n"
7          "loop:\n"
8          "cmpq %3, %%rdx\n"
9          "jge exit\n"
10         "movq (%%rbx), %%rcx\n"
11         "cmp $1, %0\n"
12         "cmove (%%rax), %%rcx\n"
13         "movq %%rcx, (%%rbx)\n"
14         "addq $8, %%rax\n"
15         "addq $8, %%rbx\n"
16         "inc %%rdx\n"
17         "jmp loop\n"
18         "exit:\n"
19         :: "r" (cond), "r" (source), "r" (dest), "r" (
               size)
20         : "rax", "rbx", "rcx", "rdx", "memory");
21 }
```

```
1  uint64_t exponentiate_uint_mod(uint64_t operand, uint64_t
       exponent, const SmallModulus &modulus) {
2  -    ... // fast cases
3
4        uint64_t power = operand;
5        uint64_t product = 0;
6        uint64_t intermediate = 1;
7       // Initially: power = operand and intermediate = 1,
            product is irrelevant.
8
9        while (true) {
10 -          if (exponent & 1) {
11 -              product = multiply_uint_uint_mod(power,
       intermediate, modulus);
12 -              swap(product, intermediate);
13 -          }
14 +          product = multiply_uint_uint_mod(power,
       intermediate, modulus);
15 +          conditional_mov(&product, &intermediate, 1,
       exponent & 1);
16
17           exponent >>= 1;
18           if (exponent == 0) {
19               break;
20           }
21           product = multiply_uint_uint_mod(power, power,
                 modulus);
22           swap(product, power);
23       }
24       return intermediate;
25 }
```

我们的一个初步尝试

- 避免侧信道泄露
  - 秘密依赖的访问模式
  - 指令延时引起的timing difference

Instruction	Appeared in function	Secret dependent?
SQRTPD; SQRTSD; SQRTPS; SQRTSS	Encryptor::set_poly_coeffs_normal	no
DIV; IDIV; DIVPD; DIVSD; DIVPS; DIVSS	Encryptor::Encryptor; SealContext::validate	no
VPMASKMOVD/Q	not found	-
RDRAND	sgx_rdrand	no
CLFLUSH; CLFLUSHOPT	not found	-

采用现有TEE的问题是什么？

- TEE是为通用计算设计的
  - 设计复杂
  - 攻击面大
  - 侧信道难以避免

- 未来工作
  - 避免可信任的第三方
    - 密码学
    - 开源和可审计
  - 如何设计TEE硬件
    - 简单、专用、容易验证
    - 避免资源共享

总结

- 信任应当建立在尽可能小的假设之上
  - 安全性规约
  - 信任链
- 安全是相对的，绝对的安全往往是没有实际意义的
  - 安全不能脱离现实的应用场景
  - 威胁模型的定义需要真实刻画现实场景
- 设计和实现是分离的
  - 实现的问题往往比较容易修复
  - 设计的问题往往是根本性问题
- 可证明安全非常重要