ASCS ADAPTIVE & SECURE COMPUTING SYSTEMS LABORATORY

Ira A. Fulton Schools of Engineering
Arizona State University

2023 International Cybersecurity Winter (ICW) School

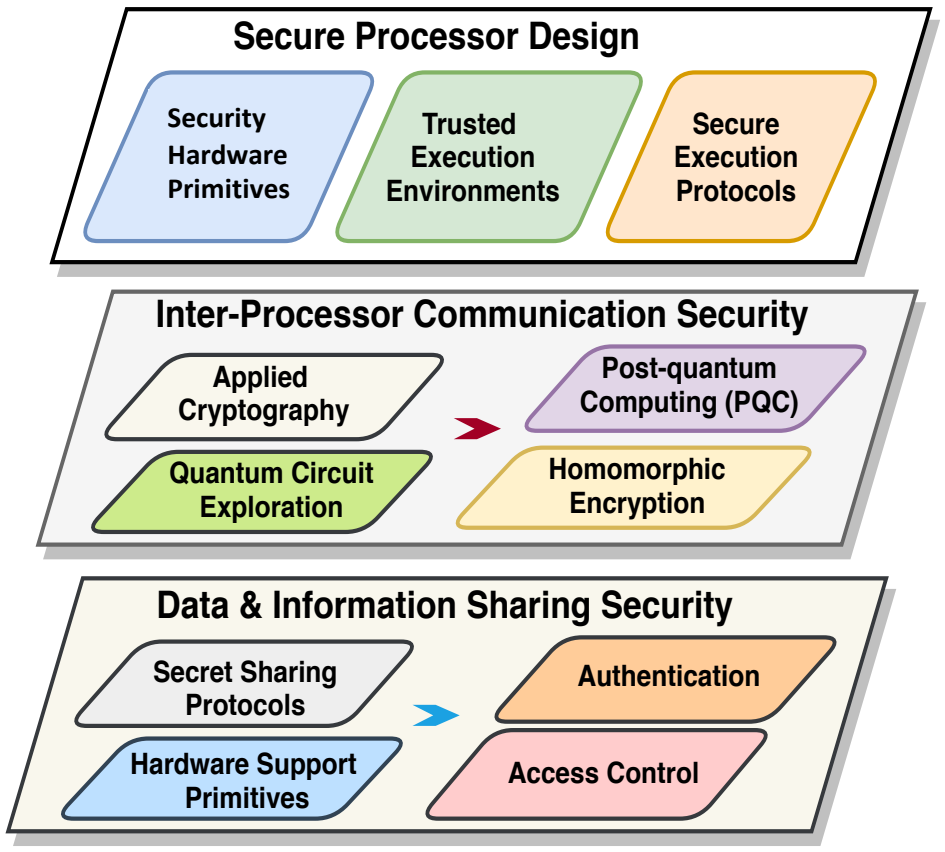
System Cybersecurity Major Topics

Prof. Michel A. Kinsy
Director
Secure, Trusted, and Assured Microelectronics (STAM) Center

STAM Center
SECURE, TRUSTED, AND ASSURED MICROELECTRONICS

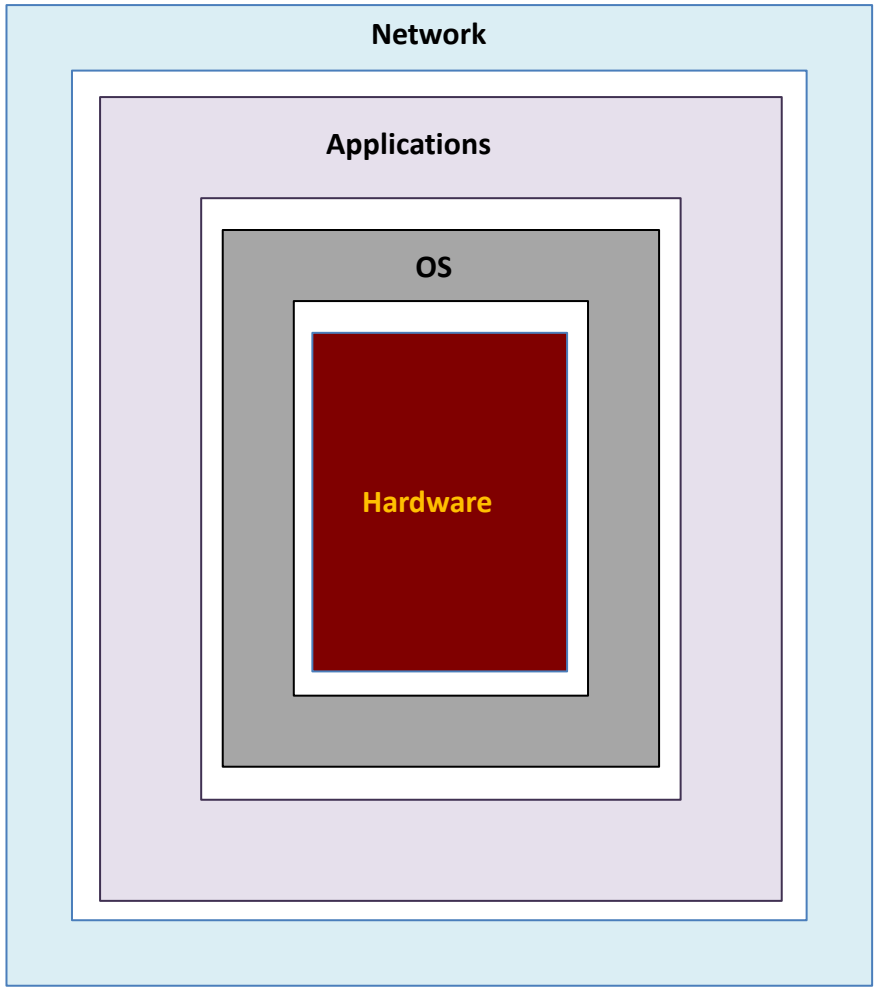
https://stamcenter.asu.edu/

Computer Systems Security



- Data in manipulation (Processing)
- Data in motion (Communication)
- Data at rest (storage)
- Side-Channel
- Supply-Chain Trust Issues

Attacks, Vulnerabilities, and Countermeasures



Security of Modern Computing Systems

Network Security
1. Denial of Service (DoS)
2. Man-in-the-Middle Attacks
3. Spear Phishing Attacks
4. ...

Information Security
1. Corruption
2. Data Leakage
3. Ransomware
4. ...

Network Storage

Phone

Wearable Device

Tablet

Wireless Router

Internet

Desktop

Modem/ Router

Server

Software

+

Hardware (Printed Circuit Board)

Software Security
1. Software Virus
2. File Corruption
3. IP Theft/piracy
4. ...

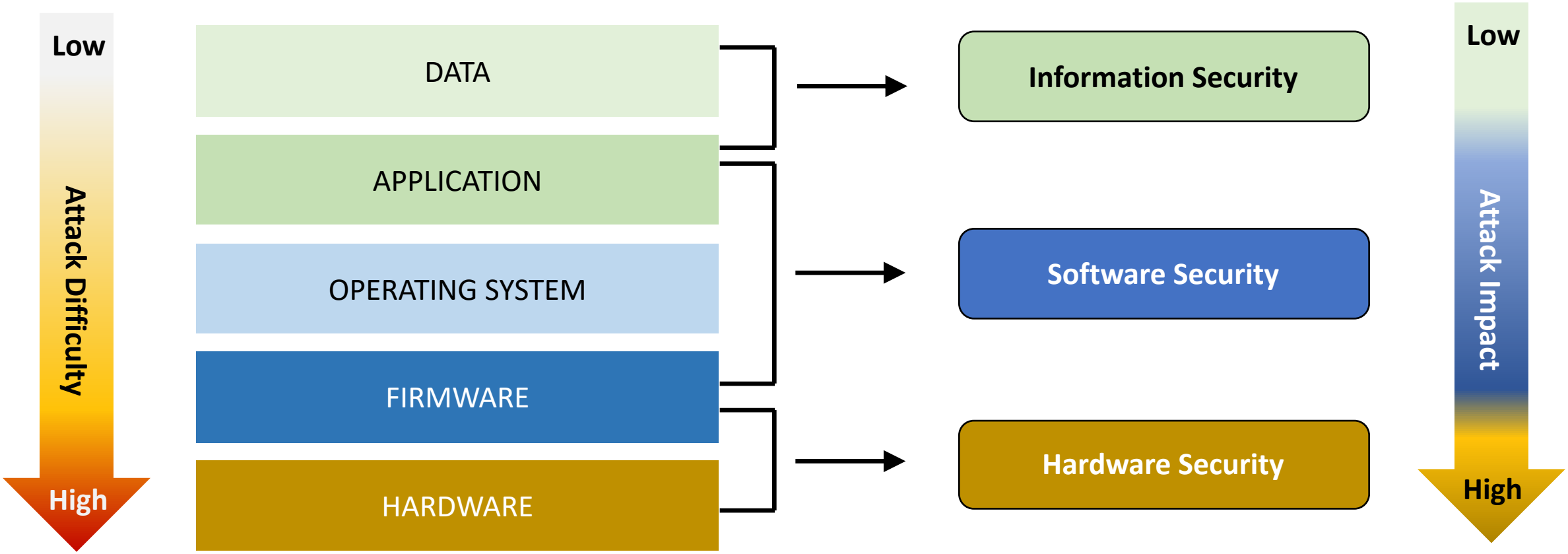
Processor IC

Hardware Security
1. Hardware Trojans
2. Hardware IP Theft
3. Reverse Eng./Cloning/SCA
4. ...

Computer Security Classes

- Software Security
  - Malicious attacks on the software, exploiting different implementation vulnerabilities (e.g., inconsistent error handling, buffer overflow)
  - Techniques to ensure reliable software operation in presence of potential security risks
- Information Security
  - General practice of providing confidentiality, integrity, and availability of information through protection unauthorized access, use, modification, or destruction
- Hardware Security
  - Attacks and protection of hardware and serves as the foundation of system security
  - Provides trust anchor for other security elements of the system

Computing System Layers



Security Policies vs. Security Mechanisms

- Security policies provide the rules for accessing/operating on resources and for protecting the computing system
- Security mechanisms security policies provide the means and technology for enforcing the security policies
- Principles for designing secure systems
  - Construct a thread model
    - Prove that security mechanisms can prevent the associated thread (s)
      - Use formal verification
  - Use auditing to detect unforeseen violations
    - Security log
      - Possible transaction information – actors' id, resource accessed, operation, timestamp
  - Tradeoffs analysis
    - Performance/quality of service versus security

Functional Security Properties of Systems

Confidentiality:

- Secure channels / symmetric cryptography:
  - One-time pads
  - Stream ciphers
  - Block ciphers
    - Modes of operation
- Key exchange / key distribution:
  - Public / private cryptography
  - Forward secrecy
- Obfuscation:
  - Indistinguishability obfuscation
  - Deniable encryption
  - Program obfuscation
    - Opaque predicates
  - Hardware obfuscation
    - Anti-tamper, split manufacturing, …
- Private lookups, private metadata:
  - Mix networks, oblivious RAM, onion routing
- Isolation
  - Virtualization, containerization, sandboxing…
  - Secure architectures,
    - Trusted execution engines, secure enclaves
  - Formal verification
- Zero-knowledge proofs

Integrity:

- Message integrity:
  - Error correction codes
  - (Cryptographic) hash functions
- Privacy-preserving computation:
  - Multi-Party Computation (MPC):
    - Oblivious Transfer
    - Yao's Garbled circuits
    - Universal composability
  - Homomorphic Encryption (HE)
  - Hardware Root-of-Trust (HRoT):
    - Physical unclonable functions, e-fuses
  - Federated Learning
  - Distributed Consensus:
    - Digital currency, private voting
- Software security:
  - Virtual memory, file system permissions
  - App signing, sandboxing
  - Control flow integrity:
    - Shadow stacks
  - Buffer overflow protection:
    - ASLR, stack canaries
  - Malware detection:
    - Antiviruses, malware signatures
    - Hardware performance counters

Authentication:

- Asymmetric cryptography
  - One-way functions, trapdoor functions
  - Key exchange / key distribution algorithms
  - Digital signatures
- Public key infrastructure:
  - Web of trust
  - Certificate authorities, root certificates, self-signed certificates
- Passwords, biometrics, …
- Password-based key derivation

Authorization:

- Access Control Lists
- Role-Based Access Control
- Capability-Based Security

Non-repudiation:

- Digital signatures
- Commitment schemes
- Message authentication codes
- Deniable encryption, undeniable signatures

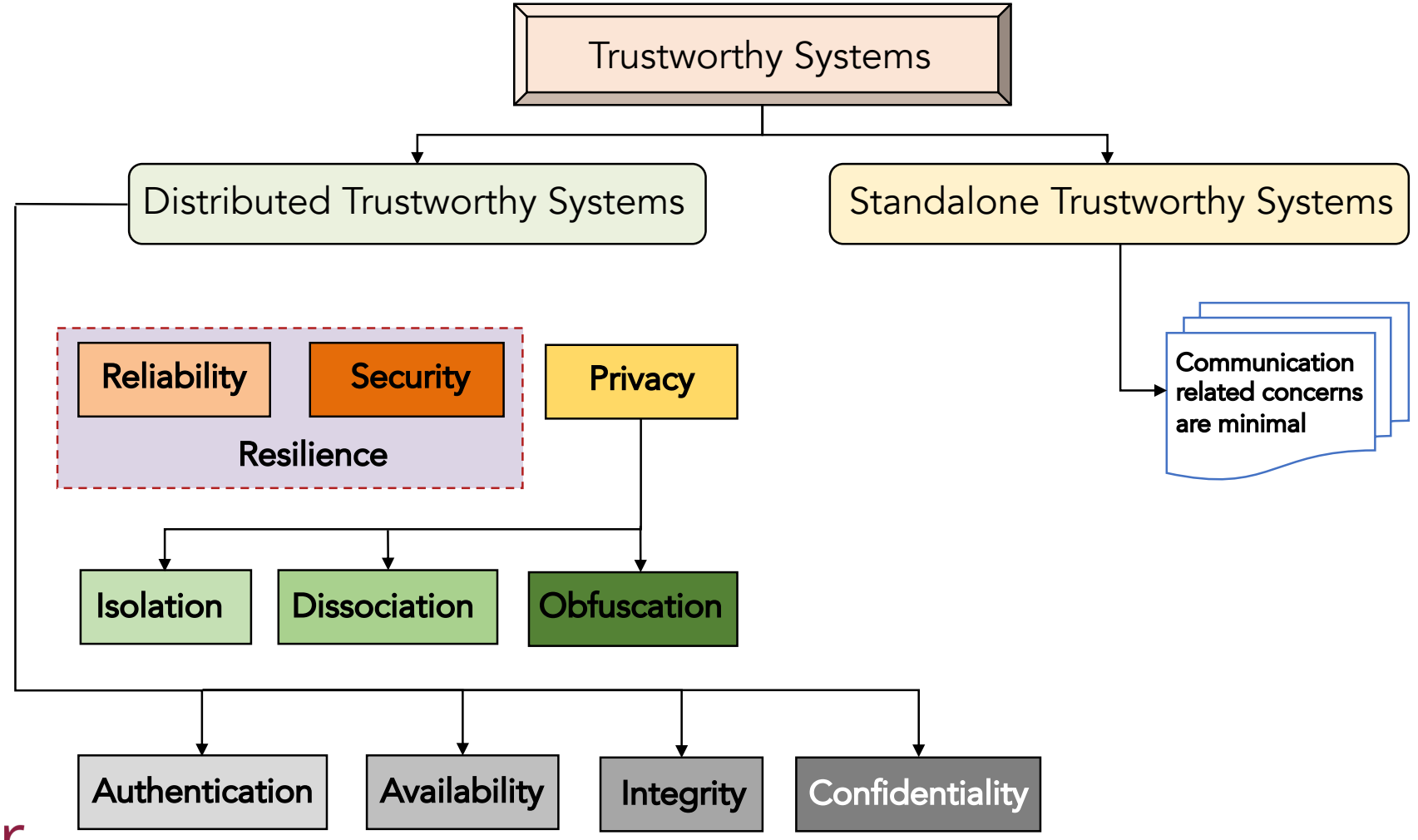
Security vs. Trust

- Security issues arise from its own vulnerability to attacks
- Trust issues arise from involvement of untrusted entities and components in the life cycle of the hardware
  - Untrusted IP and Compute-Aid Design (CAD) tool vendors, untrusted design, fabrication, test, and distribution facilities, and lack of traceable provenance
  - These parties are capable of violating the trustworthiness of the hardware component and system
- Trust issues often lead to security concerns

Trusted vs. Trustworthy

- When a component of a system is trusted
  - Security of the system depends on it
  - Failure of component will compromise the security policy
  - Determined by its role in the system
- When a component is trustworthy
  - Component is deemed to be trusted
    - e.g., It is implemented correctly
  - Determined by intrinsic properties of the component

Trustworthy System Design



Trustworthy Systems

- Security Mechanisms
  - Encryption
    - To implement confidentiality and integrity
  - Authentication
    - To verify participant identities
  - Authorization
    - To validate allowable system operations
  - Auditing
    - To check, associate and time-stamp actions with subjects
  - Trusted Computing Base (TCB) is the set of services/mechanisms within a distributed system required to support a security policy

Trustworthy Systems

- Trusted Computing Base (TCB)
  - Every distributed trustworthy system must have some TCB
  - Hardware and software necessary for enforcing all security rules
- Ideally
  - Rooted in hardware and small
  - Should be isolated from the rest of the computing components
  - Its correctness and runtime state should be easily and independently verifiable

NIST Definitions

- National Institute of Standards and Technology (NIST)
- Trusted Computing Base (TCB)
  - Totality of protection mechanisms within a computer system, including hardware, firmware, and software, the combination responsible for enforcing a security policy.
- Security kernel
  - Hardware, firmware, and software elements of a trusted computing base implementing the reference monitor concept. Security kernel must mediate all accesses, be protected from modification, and be verifiable as correct.

NIST Definitions

- Enclave
  - A set of system resources that operate in the same security domain and that share the protection of a single, common, continuous security perimeter.
- Security life of data
  - The time period during which the security of the data needs to be protected (e.g., its confidentiality, integrity or availability).
- Trusted Execution Environment
  - None
- Secure Computing
  - None

NIST Definitions

- Security architecture
  - A set of physical and logical security-relevant representations (i.e., views) of system architecture that conveys information about how the system is partitioned into security domains and makes use of security-relevant elements to enforce security policies within and between security domains based on how data and information must be protected.
  - Note: The security architecture reflects security domains, the placement of security-relevant elements within the security domains, the interconnections and trust relationships between the security-relevant elements, and the behavior and interactions between the security-relevant elements.
  - The security architecture, similar to the system architecture, may be expressed at different levels of abstraction and with different scopes.

Computing System Security Patterns

- Patterns
  - Procedural
    - Confidentiality
      - Preventing an unauthorized party (users, processes, etc) from accessing services, resources data or discovering the existence of a critical piece of data
      - Access control policies and mechanisms
      - Avoid data leakage
      - Over both main and side channels
    - Integrity
      - Only an authorized party can manipulate data or access resources and do so only in an authorized way
    - Availability
      - A resources is available if it should accessible by an authorized party on an appropriate occasion.
      - I.e., Parties or processes that are authorized to access a piece data or a resource should be able to get it

Computing System Security Patterns

- Patterns
  - Procedural
    - Confidentiality
    - Integrity
    - Availability
    - Nonrepudiation
      - A user or a process cannot deny or hid the fact that they have performed an activity
    - Authentication
      - Being able to verify identity and credential of accessing users or processes
    - Authorization
      - Determine whether a particular user or process has the credential to access a piece of data or resource or being able to carry out a function
    - Authentication and authorization are complementary

Computing System Security Patterns

- Patterns
  - Procedural
    - Confidentiality
    - Integrity
    - Availability
    - Nonrepudiation
    - Authentication
    - Authorization
    - Authentication and authorization are complementary
  - Algorithmic
    - Public-Key encryption
    - Key exchange protocols
    - Hash functions
    - Digital signatures
  - Structural

- Patterns
  - Procedural
  - Algorithmic
  - Structural
  - Horizontal and vertical tracks
    - Software and hardware
    - Virtualization and views
    - Methodologies

Computing System Security Patterns

- Patterns
  - Procedural
  - Algorithmic
  - Structural
  - Horizontal and vertical tracks
    - Software and hardware
      - Application
      - User program code
      - Compiler and utilities
      - Operating systems
      - Firmware
      - Architecture
      - Hardware
    - Virtualization and views
    - Methodologies

- Patterns
  - Procedural
  - Algorithmic
  - Structural
  - Horizontal and vertical tracks
    - Software and hardware
    - Virtualization and views
      - Logical
        - It requires translation from virtual environment to a physical one
        - Virtual vs. Physical Memory
      - Relative
        - It derives or proves the security of some part of the system based on the security of another
        - Trusted Computing Base
      - Physical
        - It is the self-containing security approach
        - Physical memory protection
    - Methodologies

Computing System Security Patterns

- Patterns
  - Procedural
  - Algorithmic
  - Structural
  - Horizontal and vertical views
    - Software and hardware
    - Virtualization
    - Methodologies
      - Design paradigms
        - Speculative execution
        - Hyper-Threading
        - Pipelining
      - Secure by design approach

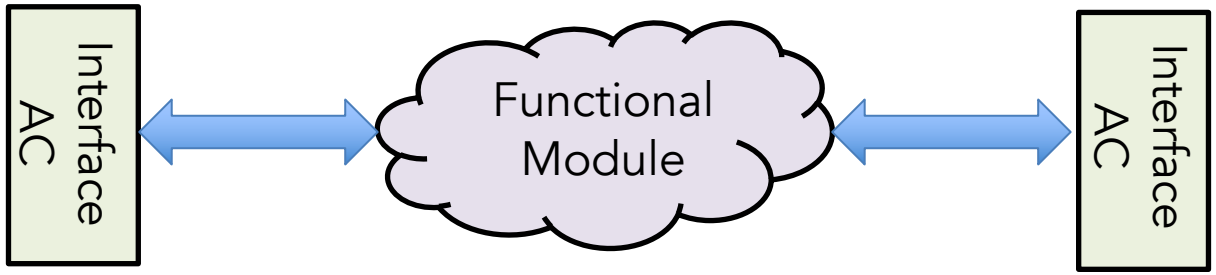
- Patterns
  - Procedural
  - Algorithmic
  - Structural
  - Horizontal and vertical views
    - Software and hardware
    - Methodologies
      - Secure by design approach
        - Methods to check the integrity of computing processes
        - Monitored and controlled access to system resources
        - Predictable computing services and behaviors

Secure by Design

- Some of the well understood concepts are
  - Least Privilege
    - Provide to each component (hardware module or software routine) only the privileges it needs
  - Fail-safe defaults
    - Only allow explicit permission
  - Efficient security mechanism
    - Implement simple security mechanisms to encourage usage
  - Formally Secure
    - Avoid relying on security by secrecy or obscurity
- They implementation is another issue altogether

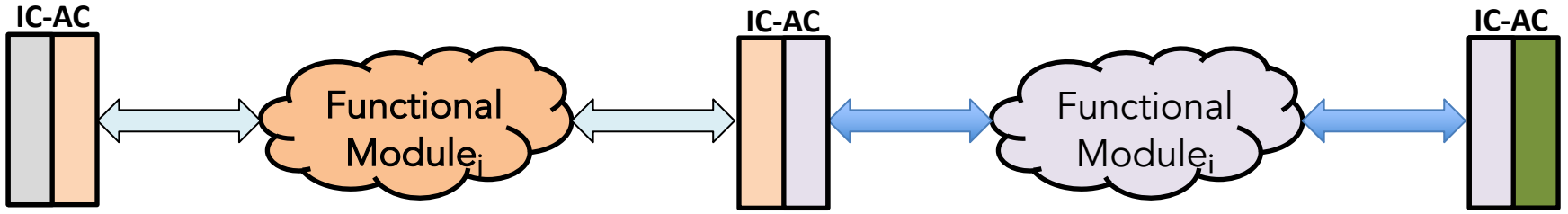
Interface-Centric Access Control

- Decide how to partition the design
  - Both physically and logically to implement IC-AC
- Regulate whether components have sufficient privileges to communicate through certain interfaces
- Provide secure interaction between secure/non-secure components
- Formally verify the composition of IC-AC policies, i.e., proper access privilege propagations
- Route messages according to policies implemented in the IC-ACs



Interface-Centric Access Control

- The module's security needs to encompass its interfaces
- It's each module's responsibility to guarantee its interface's security
- Proper hand-offs




```
Object_xi{
        Element a;
        Element b;
        Attribute p;
        Attribute q;
        Rule k;
}
```

```
Object_i{
        Element b;
        Attribute p;
        Rule f(k);
}
```

```
Object_ij{
        Element e;
        Element f;
        Element g;
        Attribute u;
        Rule r1, r1;
}
```

```
Object_j{
        Element e;
        Element f;
        Attribute v;
        Rule f(r2);
}
```

```
Object_jx{
        …
}
```

Trusted Computing Base

- The set of all hardware, software and procedural components that enforce the security policy
  - To compromise the security of the system, an attacker must subvert one or more of them
- What consists of the conceptual Trusted Computing Based in a Unix/Linux system?
  - Hardware, kernel, system binaries, system configuration files, etc.
- Key Features of Trusted Computing Base
  - Sealed Storage - stored data can only be accessed by certain software/hardware combination
  - Remote Attestation - remote certification that only authorized code is running on a system

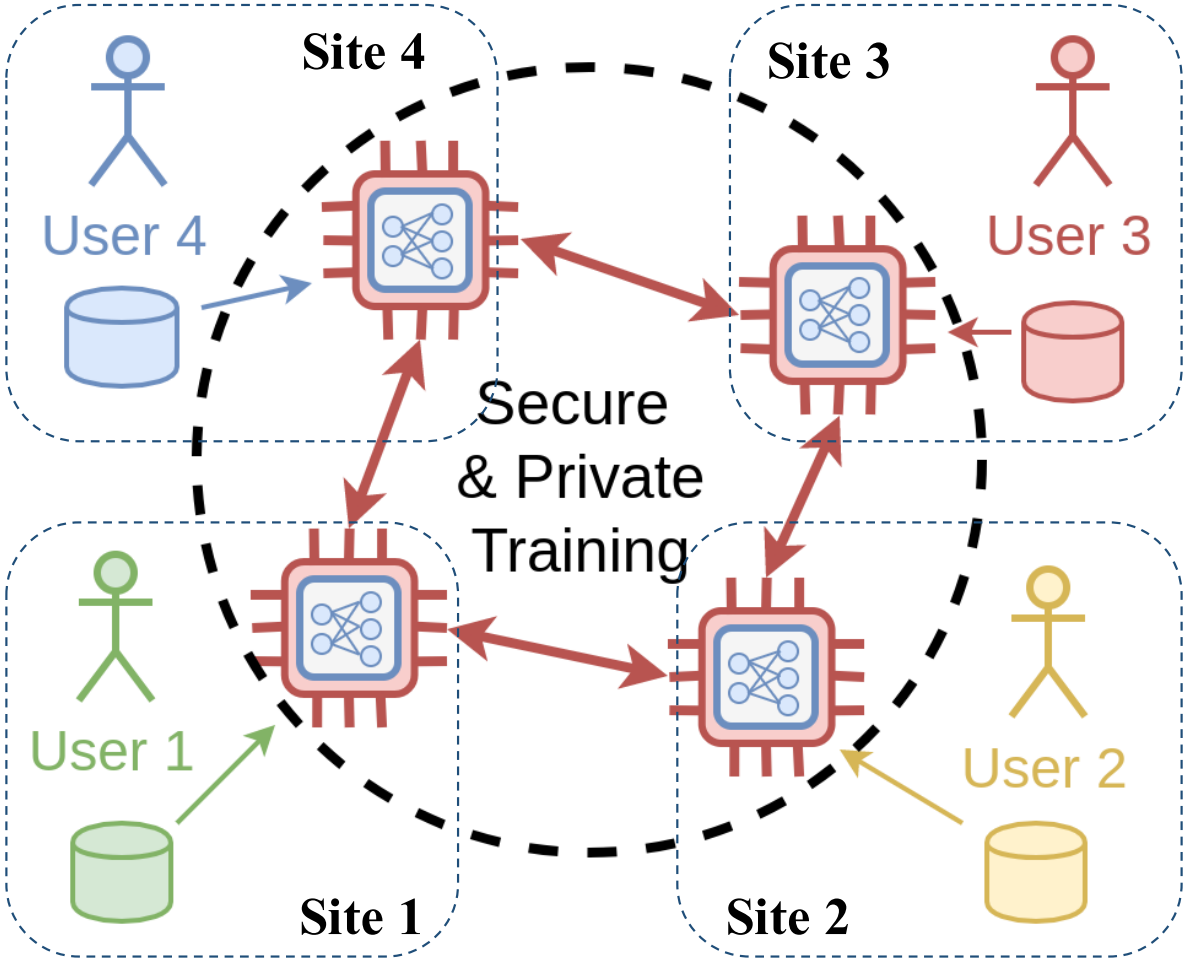
Trusted Platform Module

- What is a Trusted Platform Module
  - A specification by TCP or implementation of the specification
  - A hardware module (integrated circuit) that provides
  - Secure generation of cryptographic keys,
  - Storage of keys that cannot be retrieved
  - A Hardware Random Number Generator.
  - Remote attestation, etc
- Key Features
  - Mandatory access control
    - Object reuse protection
    - Write over old data when file space is allocated
  - Complete mediation
    - Prevent any access that circumvents monitor
  - Auditing
    - Log security-related events and check logs

Problem Statement

- Multiple hospitals may want or need to share data to better understand patients' outcomes for a particular decease or condition for a more effective treatment or care – for example Covid-19 data
- They may be required by law, but not very incentivized to do it, or provide their most up-to-date or complete data or findings
- In parallel, there are some very firm rules on how patients' data can be used or shared
- One can reduce to the problem to one of distributed training and inference among distrustful parties

Distrustful Parties Distributed Training & Inference



Secure Computation Approaches

Multi-Party Computation (MPC)

Pros

- Low compute requirements
- Easy to accelerate
- Provably secure
- Supports multiple threat models
- Easy to map existing algorithms

Cons

- High communication costs
- High latency
- Information theoretic proofs are weaker than PKE ones

Fully Homomorphic Encryption (FHE)

Pros

- Very low communication costs
- Requires a single round of communications, i.e., "fire and forget"
- Useful when one side is limited in compute / memory / storage
- Provably secure – relies on strength of PKE

Cons

- Very high computational requirements
- Harder to accelerate
- Mapping existing algorithms to FHE may be difficult

Trusted Execution Environments (TEE)

Pros

- No communication required
- Trivial to accelerate
- Great support for existing software

Cons

- Weaker security guarantees
- Cannot stop determined adversaries
- Historically plagued by vulnerabilities and breaches
- Long term deployment is difficult – TEE's can 'run out' of entropy / CRP's, etc.

Solution 1: Secure Multiparty Computation

- For the Two-party secure multiparty computation
- Assume
  - Alice has x, Bob has y, and they want to compute two functions $f_A(x,y)$ and $f_B(x,y)$
    - It could be the same function
- The set up for the n-party secure multiparty computation makes the same assumptions
  - Here instead of just Alice and Bob, there are n parties
  - Each party with a private input
  - And they want to jointly compute the function

    $fx_i=(x_1, \ldots, x_n)$

Secure Multiparty Computation

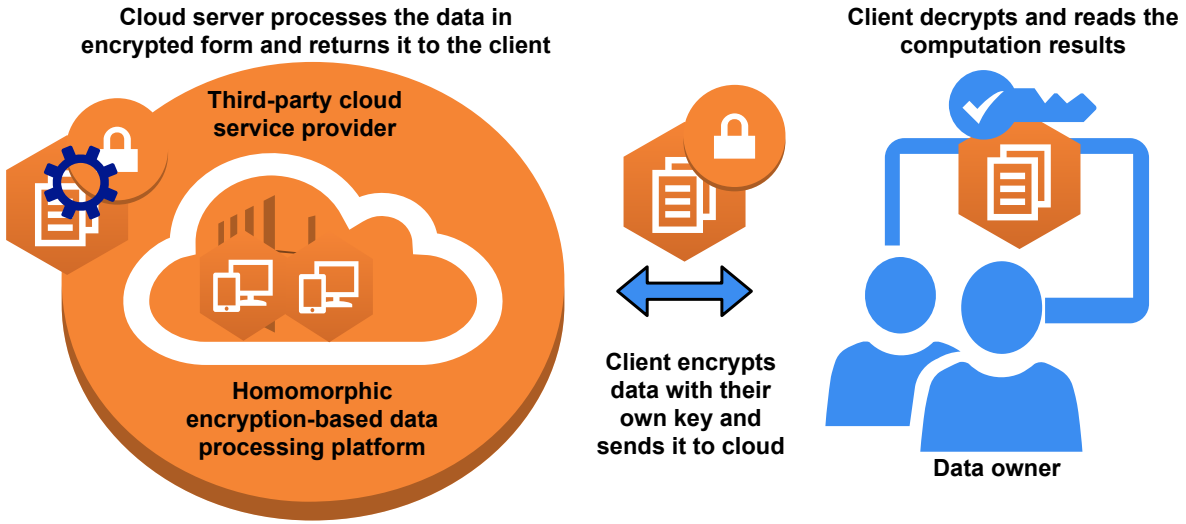
- Validity
  - Secure function evaluation (SFE) system must be able to correctly computed
    - For example, result must be computed with inputs from at least all correct parties
- Privacy
  - P_1 and P_2 cannot know each others input ip_1, ip_2
- Agreement
  - Result must be same for all parties (P_1 and P_2)
- Termination
  - All active parties (P_1 and P_2) eventually receive final result
- Fairness
  - P_1 should not be able to learn the result while denying it to P_2

Secure Multiparty Computation

- There are two major adversary models for secure computation
  - Semi-honest/passive model
    - Follows all required steps
    - Looks for all advantageous information leaked
    - Assumed to be selfish
  - Fully malicious/active model
    - Arbitrarily deviates from the protocol
    - Aborts the protocol at anytime
    - Takes any step that runs counter to the desirable outcome

Solution 2: Privacy-Preserving Computation

- Homomorphic Encryption (HE)
  - Encryption scheme that allow computation on encrypted data without decryption
  - Homomorphic encryption can be used along with cloud services to perform computations on encrypted data, guaranteeing data privacy



Overview Homomorphic Encryption (HE)

- An encryption scheme is called homomorphic over an operation '*' if it supports the following
  - $\forall (m_1, m_2) \in M, Enc(m_1) * Enc(m_2) = Enc(m_1 * m_2)$
  - Where Enc is the encryption algorithm and M is the set of all possible messages
- Supporting addition and multiplication operations is sufficient to create an encryption scheme that the homomorphic evaluation of an arbitrary function
  - Any Boolean circuit can be represented using only XOR (addition) and AND (multiplication) gates

Overview Homomorphic Encryption (HE)

- Techniques to compute on encrypted data can be classified in three (3) categories
  - Partially Homomorphic Encryption (PHE)
    - Allowing only one type of operation with an unlimited number of times
  - Somewhat Homomorphic Encryption (SHE)
    - Allowing some types of operations with a limited number of times
  - Fully Homomorphic Encryption (FHE)
    - Allowing an unlimited number of operations with unlimited number of times

What are TEEs?

- Isolated Execution
  - Isolated data cannot be read or write by other regions
  - Dedicated memory management
- Secure Storage
  - Main memory
  - Optionally non-volatile storage

What are some major TEEs

- ARM Trust Zone
  - Separates rich OS with smaller secure OS
- SGX
  - Software Guard Extension
- Sanctum
  - Builds on top of SGX
- Keystone
  - Open-source Framework, RISCV based
- AMD Platform Security Processor (PSP)
  - A trusted execution environment subsystem incorporated into AMD microprocessors

Distributed Trustworthy Systems

- Make the trusted party distributed
  - Solution share the key/secret among multiple parties
    - A threshold secret sharing scheme is a method of sharing a secret S among a set of n participants such that any subset consisting of k participants can reconstruct the secret S
    - But no subsets smaller than k size reconstruct S
- Shamir threshold secret sharing (SSS) scheme in 1979
  - It is based on Lagrange interpolation polynomial
  - Polynomial of degree k can be specified
    - By k+1 coefficients
    - By k+1 distinct points

Threshold Secret Sharing Scheme

- Select
  - p a large prime number and
  - S as the secret value
  - $s_1,\ldots,s_{k-1}$ a set of randomly numbers from [0, p-1]
- A (k, n) threshold polynomial can be written by

$$s(x) \equiv S+s_1x+s_2x^2+\ldots+s_{k-1}x^{k-1} \pmod p$$

- Send $(x_i,s(x_i))$ to the i-th participant
- Secret sharing in distributed systems provides
  - Fault-tolerant
  - Multi-factor authentication
  - Multi-party authorization

Threshold Secret Sharing Scheme

- Secret Reconstruction
  - To reconstruct the secret S, one needs to collect at least k partial secrets
  - The secret can then be reconstructed using Lagrange interpolation

$$s(x) \equiv \sum_{j=1}^{k} \left[s(x_j) \prod_{i=1, i \neq j}^{k} \frac{x - x_i}{x_j - x_i} \right] \bmod p$$

- The scheme can be extended to support share renewal and share recovery

Oblivious Transfer

- Oblivious Transfer refers to the technique of transferring a specific piece of data based on the receiver's selection

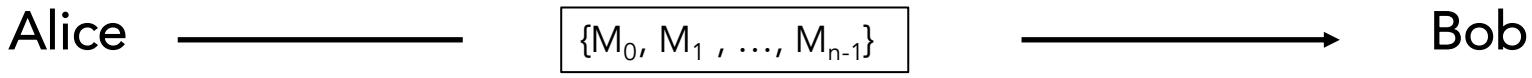
Alice sends two messages to Bob

Bob elects to see one of them and only one M_s with $s \in \{0,1\}$

- Alice does not know which one of the two Bob has selected
- Bob is also oblivious to the content of the non-selected message

Oblivious Transfer

- Oblivious Transfer refers to the technique of transferring a specific piece of data based on the receiver's selection



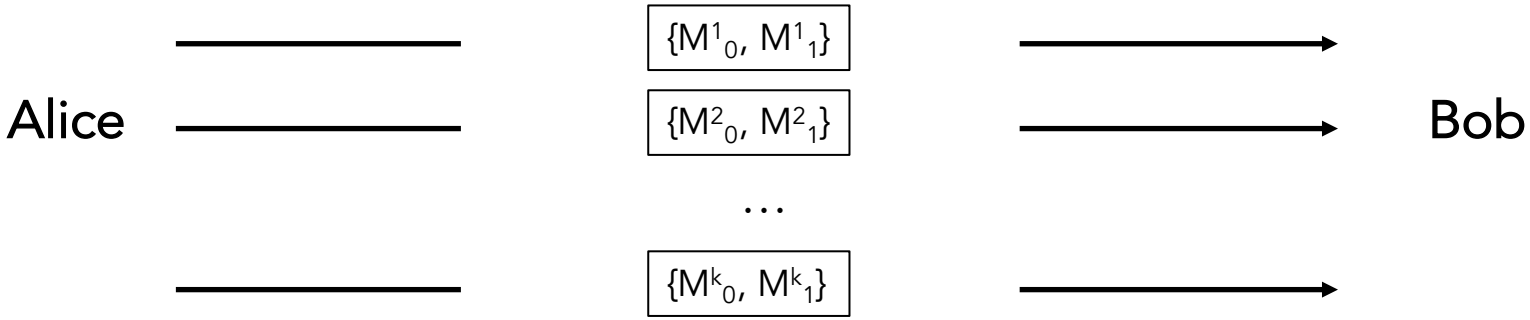
Alice sends two messages to Bob

Bob elects to see one of them and only one M_s with $s \in \{0,1,\ldots,n-1\}$

- Alice does not know which one of the *n* Bob has selected
- Bob is also oblivious to the content of the non-selected message

Oblivious Transfer

- Oblivious Transfer refers to the technique of transferring a specific piece of data based on the receiver's selection



Alice sends two-k messages to Bob

Bob elects to see one-k of them

M^k_s with $s \in \{0,1\}^k$

- There are algorithms for optimizing these straightforward implementations

Oblivious Transfer

- Oblivious transfer is the necessary and sufficient condition for multiparty computation
- How can one practically perform this oblivious transfer?
  - For that let us introduce garbled circuits
    - Garbling is a process by means of which the Boolean gate truth table is obfuscated

Public Key Infrastructure (PKI)

- What is Public Key Infrastructure (PKI)?
  - Enables users to securely and privately exchange data over an unsecured medium without the loss of integrity or confidentiality
  - "A PKI is a set of agreed-upon standards, Certification Authorities (CA), structure between multiple CAs, methods to discover and validate Certification Paths, Operational Protocols, Management Protocols, Interoperable Tools and supporting Legislation"
    - "Digital Certificates" book by J. Feghhi, J. Feghhi, and P. Williams
    - "agreed-upon standards"
      - By who?
    - "Interoperable Tools and supporting Legislation"
      - Interesting
    - "Certification Authorities (CA)"
      - This is a technical nugget that we can work with

Public Key Infrastructure (PKI)

- A PKI is an Infrastructure to support and manage Public Key-based Digital Certificates
  - Couple concepts
    - Public Key
    - Digital Certificates
- What are the functions and components of PKI
  - Certification authority (CA)
  - Registration authority (RA)
  - PKI clients
  - Digital certificates
  - Certificate Distribution System or repository
  - Keys (Public and Private)

Public-Key Cryptography

- A Public-key cryptography (PKC) is a two-key protocol system
  - It uses one key for encryption and another for decryption
  - It also called asymmetric encryption
  - It is primarily used for authentication, non-repudiation, and key exchange
- PKC depends upon the existence of so-called one-way functions or mathematical functions that are easy to compute whereas their inverse function is relatively difficult to compute
- There are three classes of cryptosystems
  - Message Digest
  - Secret Key
  - Public Key

The Three Cryptosystem Classes

- Message Digest
  - Maps variable length plaintext into fixed length ciphertext
  - No key usage, computationally infeasible to recover the plaintext
  - Examples
    - MD2-4-5, SHA, SHA-1, etc.
- Secret Key
  - Encrypt and decrypt messages by using the same Secret Key
  - Examples
    - Blowfish, DES, IDEA, RC2-4-5, Triple-DES, etc.
- Public Key
  - Encrypt and decrypt messages by using two different Keys: Public Key, Private Key (coupled together)
  - Examples
    - DSA, RSA, etc.

Public Key Infrastructure (PKI)

- A PKI is an Infrastructure to support and manage Public Key-based Digital Certificates
  - Couple concepts
    - Public Key
    - Digital Certificates
- What are digital certificates and digital signatures?
  - A Digital Signature is a data item that vouches the origin and the integrity of a Message
    - The originator of a message uses a signing key (Private Key) to sign the message and send the message and its digital signature to a recipient
    - The recipient uses a verification key (Public Key) to verify the origin of the message and that it has not been tampered with while in transit

Public Key Infrastructure (PKI)

- What are digital certificates and digital signatures?
  - A Digital Signature is a data item that vouches the origin and the integrity of a Message
- Problem with Digital Signature is how to linked the "Identity" of the Signer to signature
  - Why should one trusts who the Sender claims to be?
- Digital Certificate
  - A Digital Certificate is a binding between an entity's Public Key and one or more Attributes relating its Identity
  - The entity can be a Person, an Hardware Component, or a Service
  - A Digital Certificate is issued (and signed) by someone
    - Usually the issuer is a Trusted Third Party

Digital Certificates (CA)

- Challenges to resolve with CA
  - How are Digital Certificates Issued?
  - Who is issuing them?
  - Why should one Trust the Certificate Issuer?
  - How can one check if a Certificate is valid?
  - How can one revoke a Certificate?
  - Who is revoking Certificates?

Certification Authorities (CA)

- The basic functions of the CA
  - Key Generation
  - Digital Certificate Generation
  - Certificate Issuance and Distribution
  - Revocation
  - Key Backup and Recovery System
  - Cross-Certification
- Certificate Distribution System
  - Provide a repository or a set of repositories for
    - Digital Certificates
    - Certificate Revocation Lists (CRLs)

Registration Authority (RA)

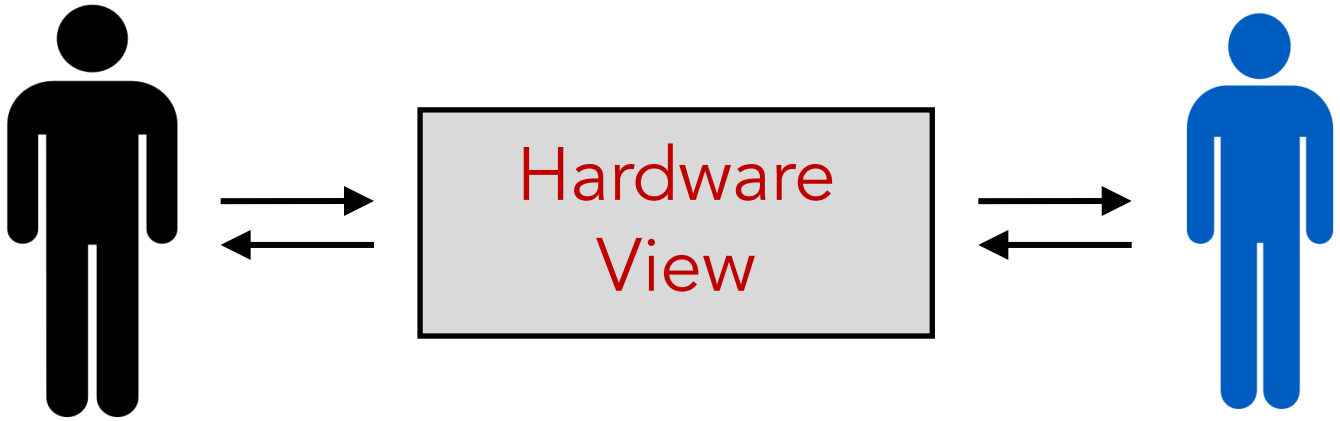
- The basic functions of the RA
  - Registration of Certificate Information
  - Face-to-Face Registration
  - Remote Registration
  - Automatic Registration
  - Revocation
- How should we do these for hardware?
  - Excellent question

Public Key Infrastructure (PKI)

- PKI-enabled Applications
- Functionality required for PKI
  - Cryptographic functionality
  - Secure storage of Personal Information
  - Digital Certificate Handling
  - Directory Access
  - Communication Facilities

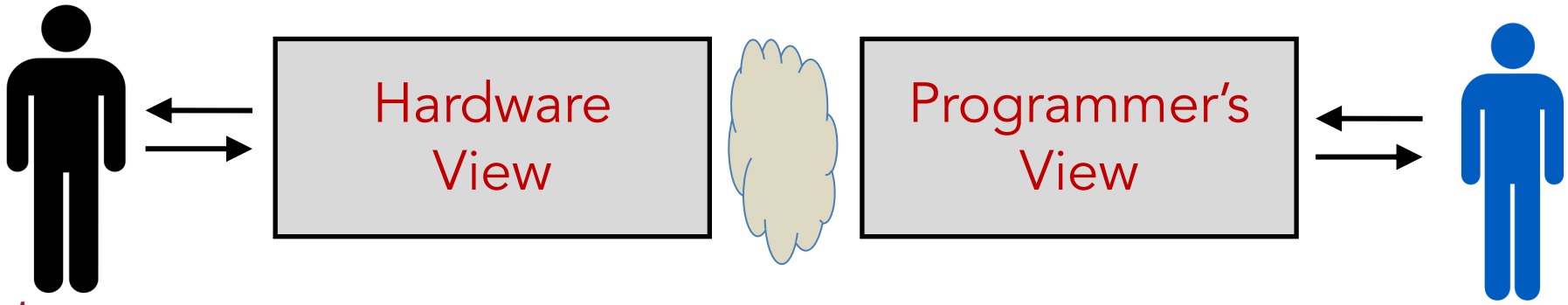
Programming Model

- The underlying programming model does not help either
- In the mid in 50's, the programmer's view of the machine was inseparable from the actual hardware implementation



Programming Model

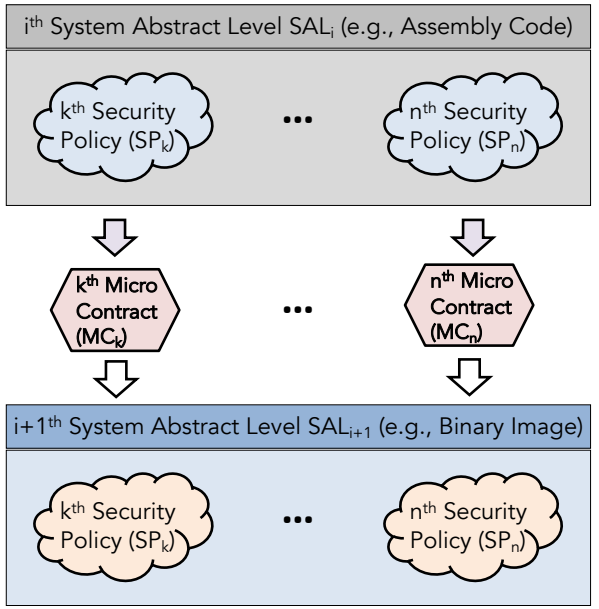
- Over time the programmer's view and the hardware implementation diverged
  - Programmer visible state of the processor (and memory) plays a central role in computer organization for both hardware and software
    - Software must make efficient use of it
- Programmer's machine model is a contract between the hardware and software



What is the Big Security Idea?

- Our answer is *security micro-contracts* or simply *micro-contracts*

Application	**Software**
Compiler/Libraries	
Operating System	
Instruction Set Architecture (ISA)	
Architecture/Microarchitecture	**Hardware**
Circuits (Hardware Functions)	
Semiconductor Technology	

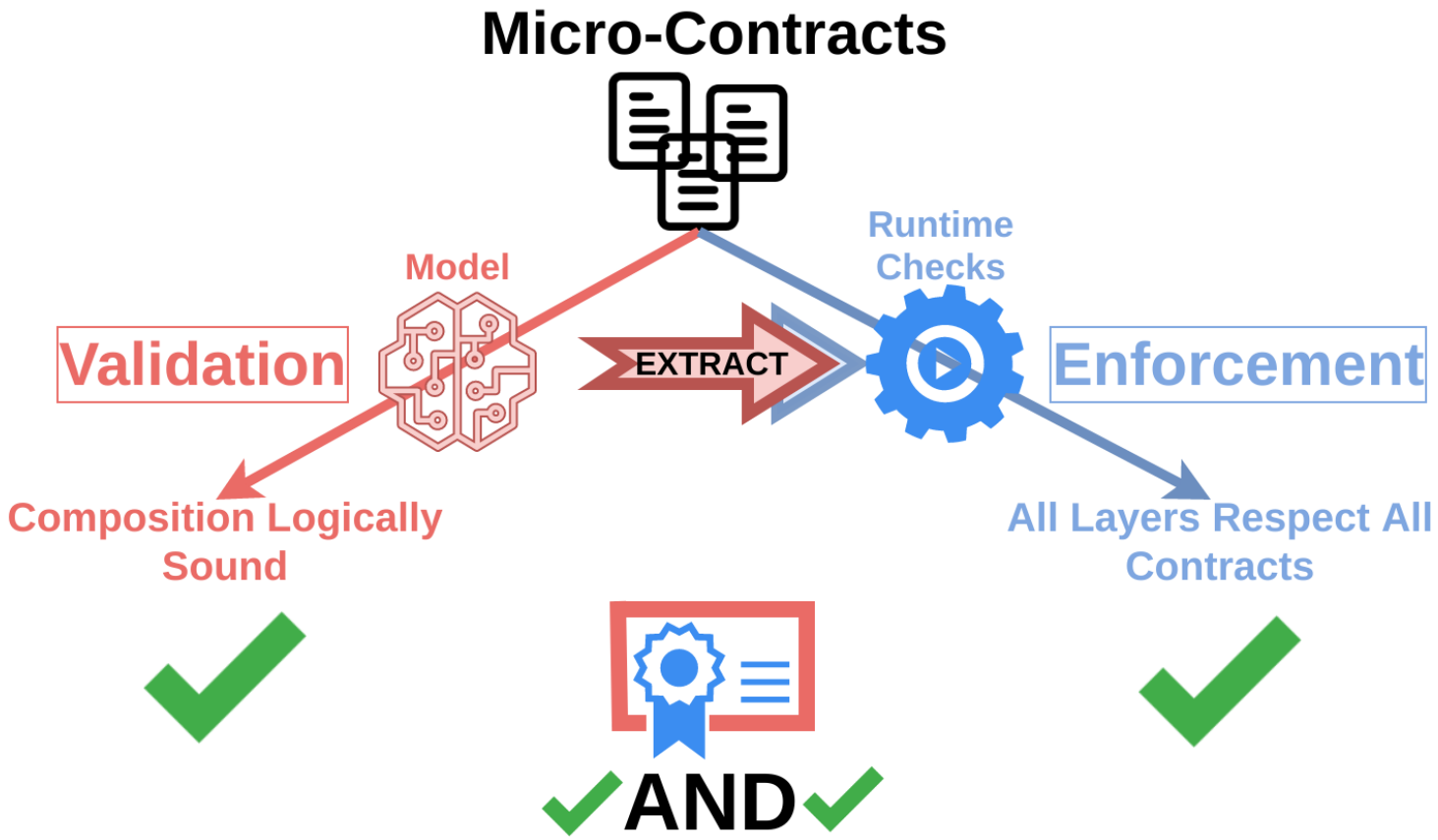


Security Vulnerabilities

- Human Error
  - Software bugs causing privilege escalation
  - OpenSSH library that allows for connecting unauthorized clients to the server
- Misuse of the existing abstractions
  - Compiler optimizes code exposing security vulnerabilities -optimization-unstable code
- Inherent to the Abstraction
  - Adjacent Layers optimizing in their own way without knowledge of the other layer.
  - E.g., Compiler cannot distinguish between load and store that cause a cache miss or a cache hit.
  - Strength Reduction
    - Two equally time intensive branches become to have observable time difference

Security Micro-contracts & Their Guarantees

- Security centric framework that considers security of the complete system
- Strictly Defined, Minimal, Modular
- Policy implemented between adjacent layer



Micro-Contract Motivation

- Study security implications of computing system design decisions
  - Decisions on one layer may impact the decisions on the adjacent layers
- Some of the security issues remain undiscovered when studied on a single layer
  - Cache side-channel attacks
- Some valid optimizations according to realistic objectives may introduce security flaws
  - Performance optimizations introduce security flaws
  - Speculation in microarchitecture or compiler strength reduction
- Model attacks formally for attestation reasons
  - Prove that system is not vulnerable?

Problem statement

- Microarchitectural side-channels are effective - as motivated by these examples
  - Spectre, Meltdown, Microarchitectural Data Sampling (Fallout, RIDL, ZombieLoad)
- What they have in common?
  - Data supposed to be not observable is observable
- Closing these gaps are required in other phases
  - Compiler and OS need to establish contracts over ISA with Microarchitecture
- Formal methods tools that model microarchitecture are available
  - That allow us to be rigorous and to offer guarantees while keeping focus on architecture and security aspects of the research
- Given an attack model, to what extent can we guarantee the computing system's resistance to it?
  - Computing system is considered holistically as the composition of all its parts

Semantic Gap Challenge

- Semantics - the meaning of a program
  - In fact, a program does not do what the author says but what the semantics says.
  - Semantic rules are formally given by the language specification and are implemented by a compiler/runtime.
  - C11 specification is given by ISO/IEC 9899:201
- Denotational
  - The meaning of a program is a function of the meanings of its parts.
  - "Mathematician's viewpoint"
- Operational
  - Step by step algorithm unfolded in time
  - "Engineer's viewpoint"

Semantic Gap Illustration

What is programmer's idea?
- Scrub the memory to make the key unavailable to the attacker as soon as possible.

What is compiler's idea?
- In compiler optimization terminology *key = 0x0;* is a dead store
- Delete this statement
- The code has the same denotational semantics with less assembly instructions, optimization is sound!

What is attacker's idea?
- Find an attack entry (possibly somewhere else), steal the key from memory.

```
crypt() {
    key = 0xC0DE; // read key
    ... // work with the secure key
    key = 0x0; // scrub memory
}
```

CWE-14 and CWE-733

Can We Find Some Simple Solutions?

- Disable all optimizations
  - 5x performance overhead
    - David Molnar et. al., "The Program Counter Security Model: Automatic Detection and Removal of Control-Flow Side Channel Attacks", 2005.
- Use volatile keyword
  - Programmers do not use it properly (number of bugs reported)
  - Compilers do not compile it correctly
    - Eric Eide and John Regehr, "Volatiles Are Miscompiled, and What to Do about It", 2008.

How Much Should We Trust the Compiler?

- Optimizing compilers
  - Compilers that undertake any transformation on source program to make it better
- Optimization-unstable code
  - Code removed by an optimizing compiler due undefined behavior (correct transformation)
  - Incorrect functionality
  - Security issues

Ken Thompson, “Reflections on Trusting Trust”, Turing award speech, 1984.

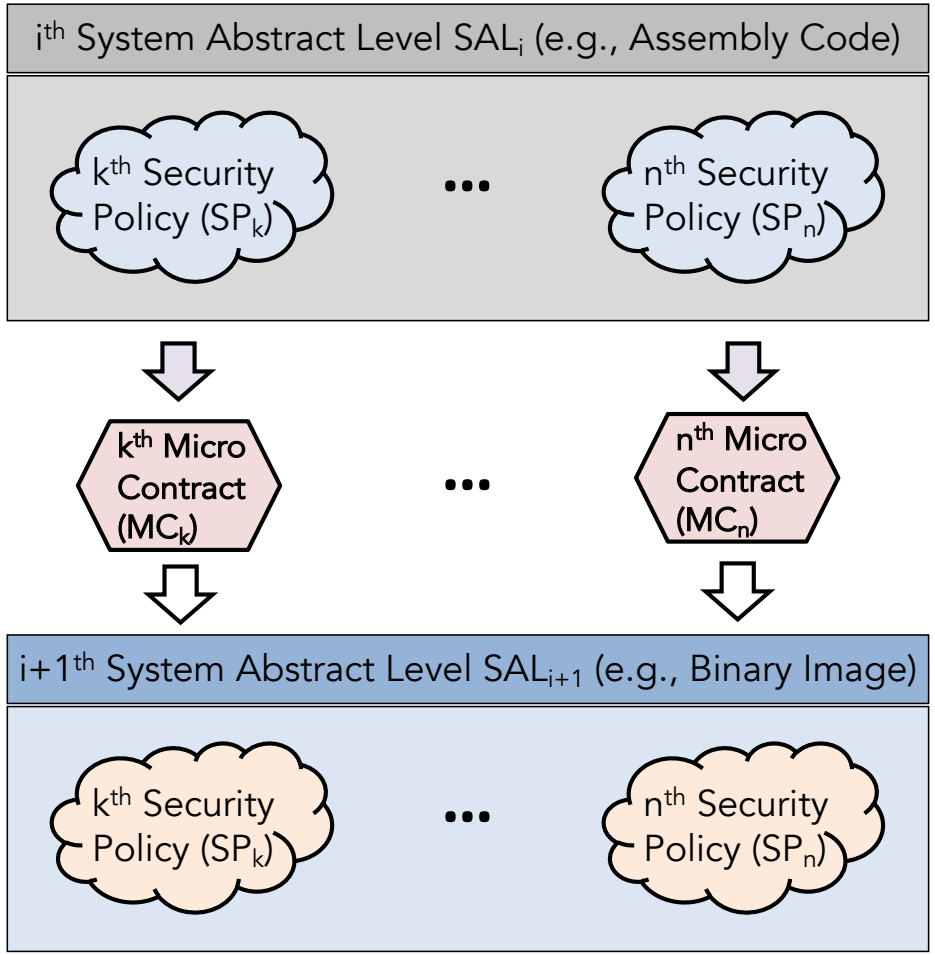
Compiler Optimization Introduced Vulnerabilities

- There are three classes
  - Class I: Information leak through persistent state
    - Dead store elimination
    - Function call inlining
    - Code motion
  - Class II: Elimination of security-relevant code
    - Code elimination due undefined state
  - Class III: Introduction of side channels
    - Common subexpression elimination
    - Strength reduction
    - Peephole optimizations

D'Silva et. al., "The Correctness-Security Gap in Compiler Optimization", 2015.

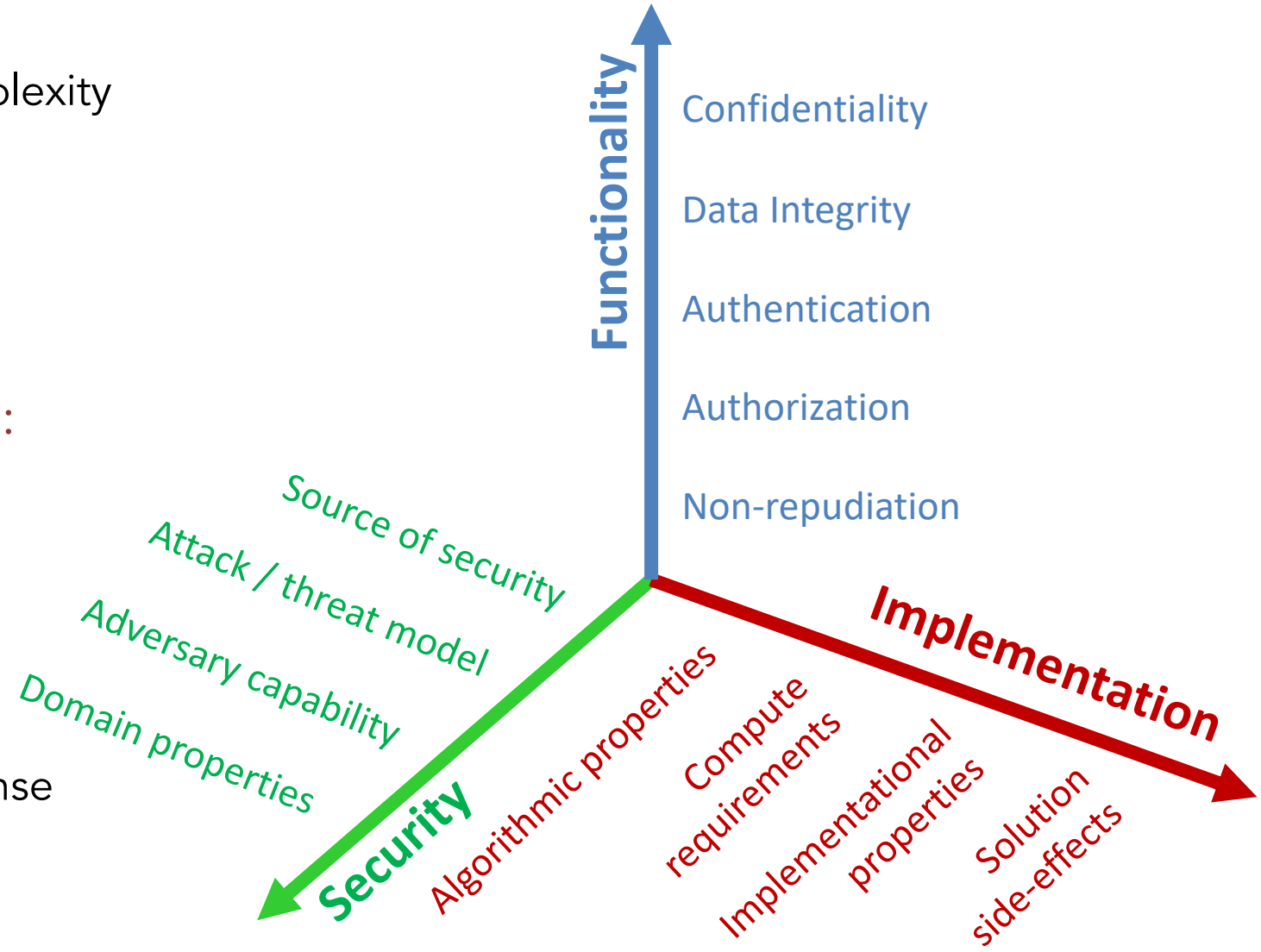
Micro-Contracts Approach

- Should rely on both approaches
  - Adopt abstract machine approach to make models that correspond well to particular real machines
  - Internally model operational semantic of C w.r.t. tuple (attack, architecture/"real machine")
- Light weight - prove only micro-contracts of computing system
  - E.g., timing side channels on LLVM-RISC-V
- An instance of Compiler-ISA Class III Micro-Contracts
  - Bottom up - try to get maximum usability of minimal abstract machines



Security Concepts: Implementational Properties

- Implementational properties:
  - Algorithmic properties:
    - Computational / space complexity
    - Strong / weak scaling
  - Compute requirements:
    - FLOPS
    - Memory
    - Network bandwidth
  - Implementational properties:
    - Throughput
    - Latency
    - Power & area
    - Error correction, noise robustness
  - Solution side-effects:
    - Side-channel attacks & defense



PQC Secure – Post-Quantum Cryptosystems Tool

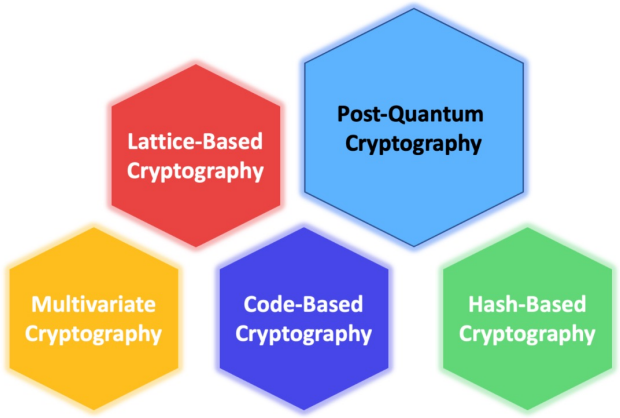
PQC Secure .

Home Research Publications Contact us

We design, implement, and deploy post-quantum secure systems.

The team investigates post-quantum secure computing systems and cryptosystems,

- Post-Quantum Cryptography
- Post-Quantum Hardware Library
- Lattice-Based Cryptography
- Code-Based Cryptography
- Cryptographic Agility
- Homomorphic Encryption Hardware



Foundations of Quantum Resistant Cryptography

Following the NIST (National Institute of Standards and Technology) PQC proposal submissions and rounds, we have been investigating the mathematical foundations of the algorithms, real-time implementation, hardware architecture, open problems, attack vectors, and crypto-agility.

We have been examining their performance, parallelism, security under worst-case intractability assumptions, memory utilization, and latency. Our algorithmic and system work includes lightweight lattice-based cryptography, ultra-low latency, and seamless integration with the existing infrastructure.

Algorithm Design

Research, Development, and Training Portfolio

Algorithms Design

Code-based cryptosystems are still quantum resistant. We advance a new variant of the McEliece cryptosystem that takes advantage of non-binary Orthogonal Latin Square Code to achieve much lower complexity and key size.

High-Performance Designs

A set of FPGA-based post-quantum cryptographic primitives (PQCPs) consisting of four frequently used security components, i.e., public key cryptosystem (PKC), key exchange (KEX), oblivious transfer (OT), and zero-knowledge proof (ZKP).

Learn More

Flexible Hardware Library

An open-source, hardware library with a focus on accelerating the arithmetic operations involved in Ring Learning with Error (RLWE)-based algorithms. Library components include RNS, CRT, NTT-based polynomial multiplication, etc.

Learn More

Noise Sampling Designs

Small error sampling - we provide different noise samplers with the goal of providing concrete recommendations for future use and adoption in various cryptosystems based on sampling efficiency, hardware cost, and throughput.

Learn More

Low-Power Hardware Design

A key challange for PQC cryptosystems is their power consumption. We explore improvements that can enable their effective implementation in low-power portable/IoT devices.

Learn More

https://www.pqcsecure.org/

Tools - Gauge: An Interactive Data-Driven Visualization Tool

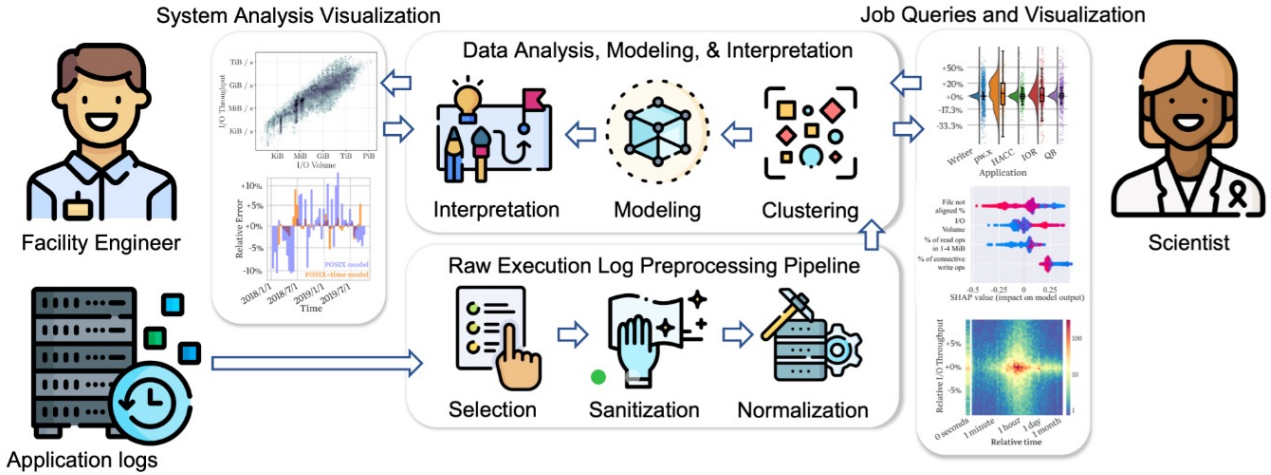
Gauge
Clustering
Modeling
Interpretation

Home Applications Publications Collaborators Contact Us Interactive Gauge

Gauge: Domain-agnostic dataset exploration tool

Gauge is an information analysis and interpretation tool that works on bulk unsorted and unlabeled data to find patterns hidden 'between' data points. Gauge extends analyst capabilities by grouping similar data into clusters where their differences can be highlighted, and an analyst can develop an understanding of cause and effect within a local area.

Live Gauge



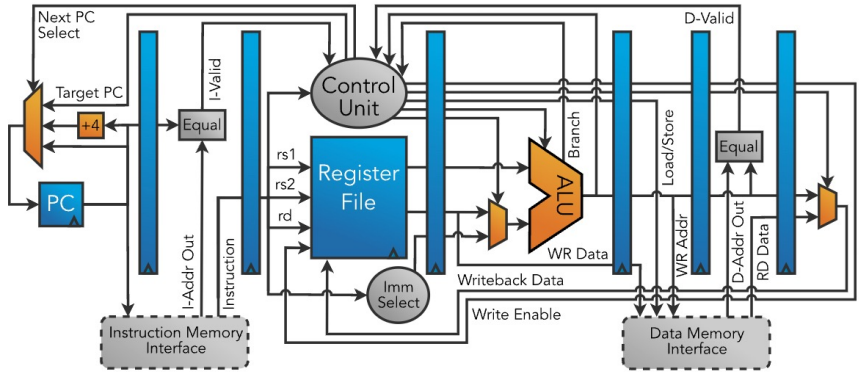
https://www.gaugeviz.org/

RISC-V Infrastructure

Trireme RISC-V Design Platform

Home Features Examples Publications Authors Download Links

A Complete Platform for RISC-V Design Space Exploration



The Trireme Platform includes everything you need to bring up the hardware and software of a custom RISC-V system, from ultra-low-power microcontrollers to high-performance multi-core processors.

Parameterized RTL

Written in Verilog 2001 without any vendor-specific or 3rd party IP.

Build custom hardware systems with a variety of cores, a configurable coherent cache system, memory modules/controllers, and Network-on-Chip support.

Get the RTL ⇨

Toolchain Included

Multiple output binary formats for use on Linux, bare-metal FPGA systems, or RTL simulations.

Simple parameters to customize binary entry point, stack address & size, heap size, and more. Board Support Package (BSP) included.

Get the Toolchain ⇨

Web-Based ISA Simulator

Simulate RISC-V ISA instructions right in your browser. Ideal for beginners and classroom use.

Upload C and assembly, or write it in the editor window. Observe register and memory state after each instruction.

Try the Simulator ⇨

Trireme Explorer

Generate Verilog for a custom core, cache, memory & NoC configuration. Download customized top module.

Select Verilog parameters in a GUI. See a visualization of the configured RISC-V system. Save and export settings for sharing.

Try the Explorer ⇨

Example Systems

From low-power microcontrollers, to Linux ready multi-core processors, the Trireme Platform can do it all. Get started with some of the Trireme Platform's included example systems.

Arroyo Processor
Linux ready single-core system

Basin Processor
Dual-core with DRAM or Flash

Cove Processor
Microcontroller with privilege modes

Delta Processor
Simple Out-of-Order core system

https://www.trireme-riscv.org/

Acknowledgements

- Sincere thanks to the STAM Center Team