Hardware Security, FTW!

Todd Austin

University of Michigan / Agita Labs
austin@umich.edu

CSE

AGITA LABS

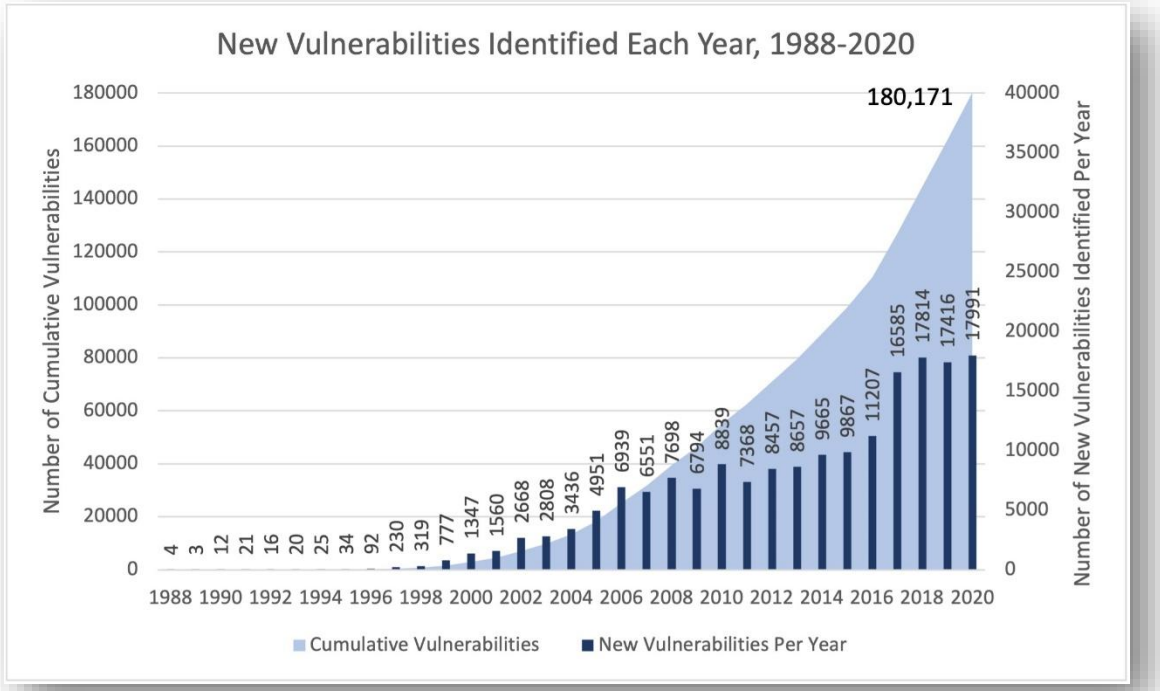
Presentation Agenda

- Part I: Taking Security from Good to Great
- Part II: A Vision for Privacy-Oriented Programming
- Part III: TrustForge: A Cryptographically Secure Enclave

Part I: Taking Security from Good to Great

Why Security Isn't Very Secure

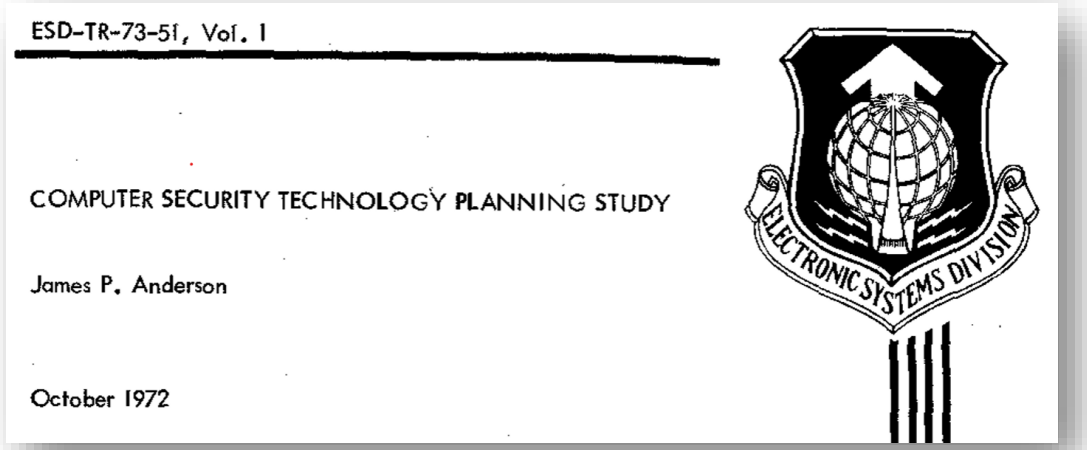
- Endless software vulnerabilities
  - Typically, bugs in software
  - Finding/fixing vulnerabilities doesn't scale
- Side channels abound
  - Observable properties reveal secrets
  - Control, memory, timing, cache, speculative
  - Performance-centric design creates side channels
- ***Case in point:*** 70%+ of Americans have had their SSN stolen in the last decade



Source: IBM X-Force Red

Patching Our Way to Security

- When attacked, we patch…
  - We lack the technology to find all vulnerabilities
  - Due to new code and new vulnerabilities
  - So, patching continues ***forever***
- Creates an asymmetric battlefield
  - Attacks only require ***one vulnerability***
  - Protection requires fixing ***all vulnerabilities***
  - *"Perfection is the enemy of progress"*
- Lacks protection for emergent attacks
  - Zero protection for zero-day attacks


ESD-TR-73-51, Vol. I

COMPUTER SECURITY TECHNOLOGY PLANNING STUDY

James P. Anderson

October 1972

ELECTRONIC SYSTEMS DIVISION


CVE-2025-24993: Windows NTFS Remote Code Execution Vulnerability

The heap-based buffer overflow vulnerability in Windows NTFS may allow an authorized attacker to execute code locally. An attacker may trick a local user on a vulnerable system into mounting a specially crafted VHD to trigger the vulnerability.

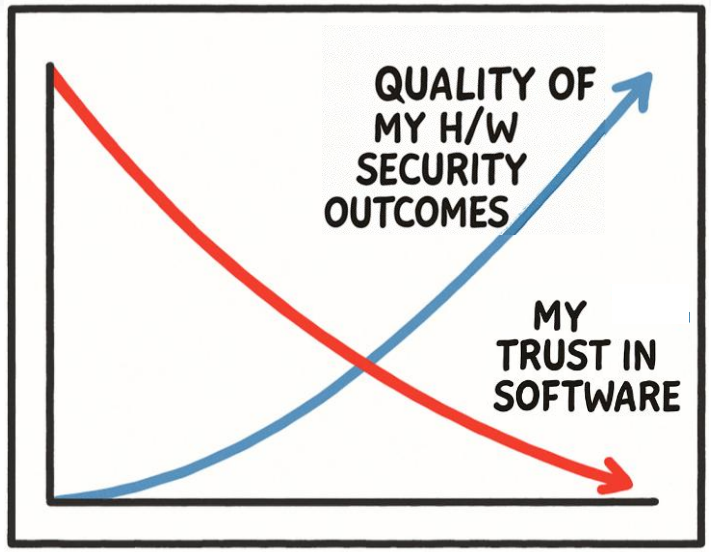
CISA added the CVE-2025-24993 to its Known Exploited Vulnerabilities Catalog and requested users to patch it before April 1, 2025.

Software, the Root-of-Distrust

- What is trust?
  - Knowing ***precisely and for certain*** what something can and cannot do
  - Deciding if “can do” is a threat
- What can software do?
  - Only what it has been ***attested*** to do
  - Unless hacked, then attested S/W can do anything
  - So, verify the software and use safe languages
- Spectre undermines all safety in, and verification of, software
  - Spectre creates ***new function in mispeculation***
  - Defenses do not consider transient functionality
  - E.g., safe Rust Spectre V1 buffer overreads
  - E.g., PACMAN silent PAC pointer checks
- Thus, software, not just bugs in software, is the root of distrust

Safe Rust Spectre V1 Gadget

```rust
#[inline(never)]
pub fn fetch_function(arr1: &Vec<u8>,
                      arr1_len: &mut usize,
                      arr2: &[u8], idx: usize) -> u8
{
    let mut val: usize = 0;
    if idx < *arr1_len
    {
      val = arr1[idx] as usize;
      return arr2[val * 512]
    }
    0
}
```



Taking Security from Good to Great…

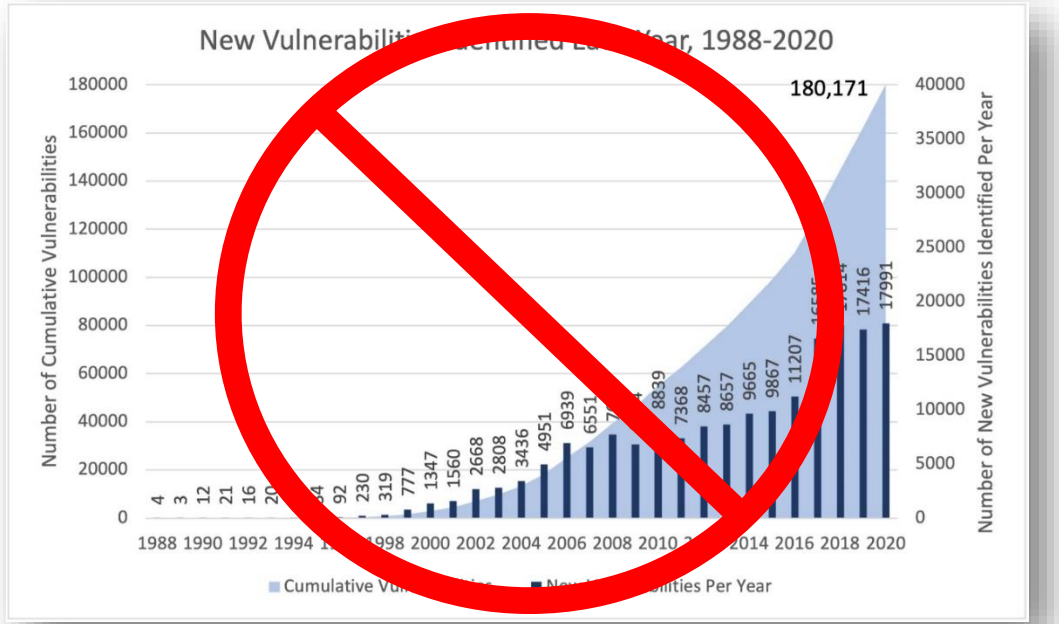
1. ***Concentrate trust*** into a small component
   - Provides separation of concerns and isolation
   - Approach: ***build a trusted secure enclave***

2. Eliminate ***all*** software vulnerabilities
   - But all software is (eventually) hackable!
   - Approach: ***no S/W inside the trusted enclave***

3. Silence ***digital*** side channels
   - Control, memory, timing, and microarchitecture
   - Approach: ***provably side-channel free enclave***

4. Cryptographically secure data everywhere else
   - Eliminates trust for all remaining S/W and H/W
   - Approach: ***encrypt w/ authenticated high-entropy cipher***

- ***TrustForge Enclave:*** a H/W-only enclave without software or digital side channels that computes directly on encrypted data



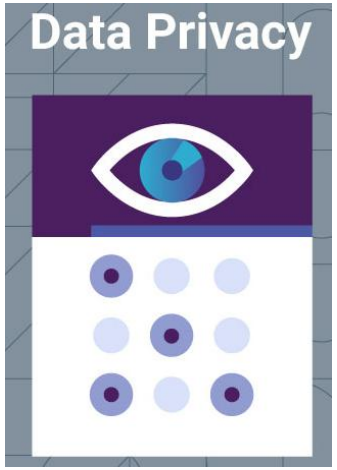
Source: IBM X-Force Red

Part II: A Vision for Privacy-Oriented Programming

Security vs. Privacy

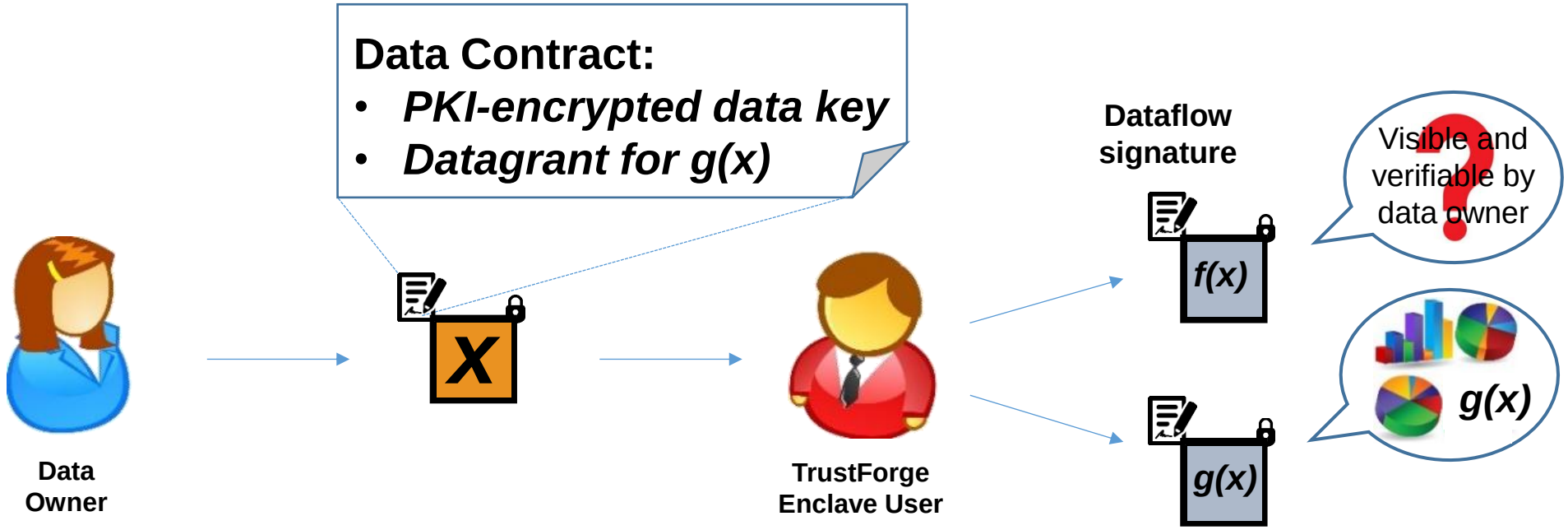
- Are they the same? No!
  - Security – protect ***confidentiality***, ***integrity*** and ***availability*** (CIA)
  - Privacy – ***give data owners control*** over use and visibility of their data
  - Partially overlapping relationship is shared by these two endeavors

- Questions to consider:
  - Is it possible to have strong privacy without strong security?
    - No, strong privacy ***demands*** strong security
  - It is possible to have strong security without strong privacy?
    - Unfortunately, yes, strong security has ***no implication*** on privacy



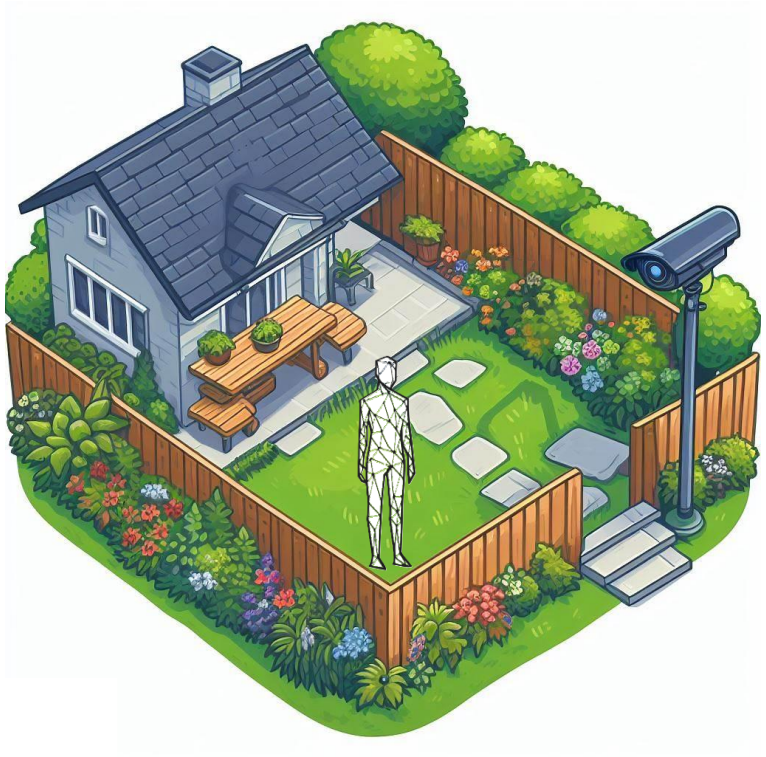
- **Bonus:** privacy applications add value to your security research
  - IRL, H/W security is often view as a ***tax on system cost***

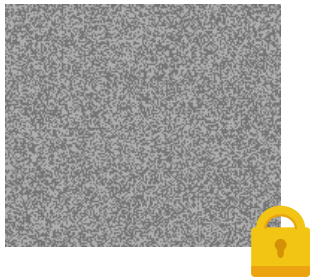
TrustForge Privacy Programming Model



- ***Encrypted verifiable computation*** is performed on sensitive data
  - Data owner sends data contract and encrypted source data
  - Computation produces encrypted results + proof-of-computation
  - Only data owner can decrypt general computation results and verify dataflow
- ***Safe disclosures*** permit enclave user to see specific values
  - Enabled by encrypted datagrants provided in advance by data owner
  - Decryption possible if-and-only-if computation proof matches datagrant

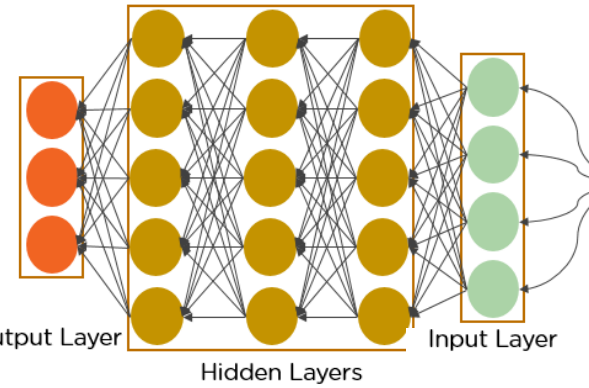
Privacy-Enhanced Surveillance

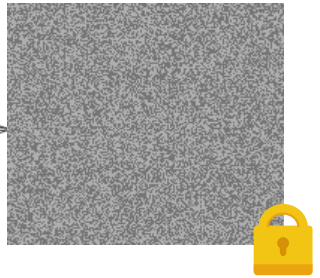




TrustForge

Encrypted computation





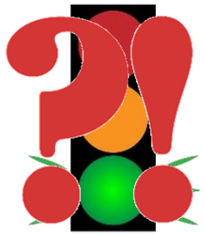
data release

Resident

Intruder

Animal

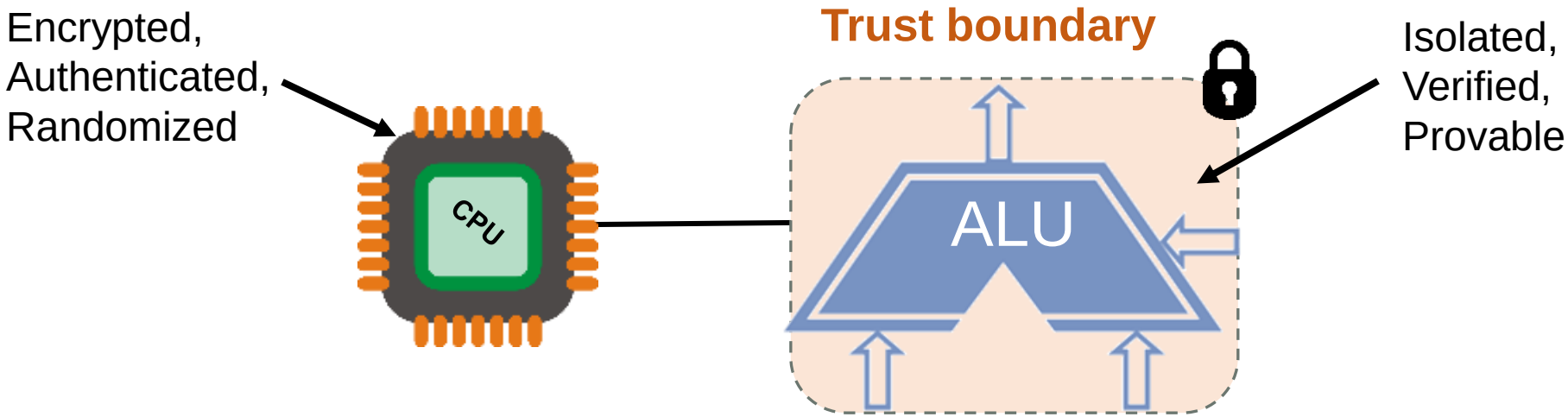
Return verified resu



Part III: TrustForge: A Cryptographically Secure Enclave

This is My Central Claim…

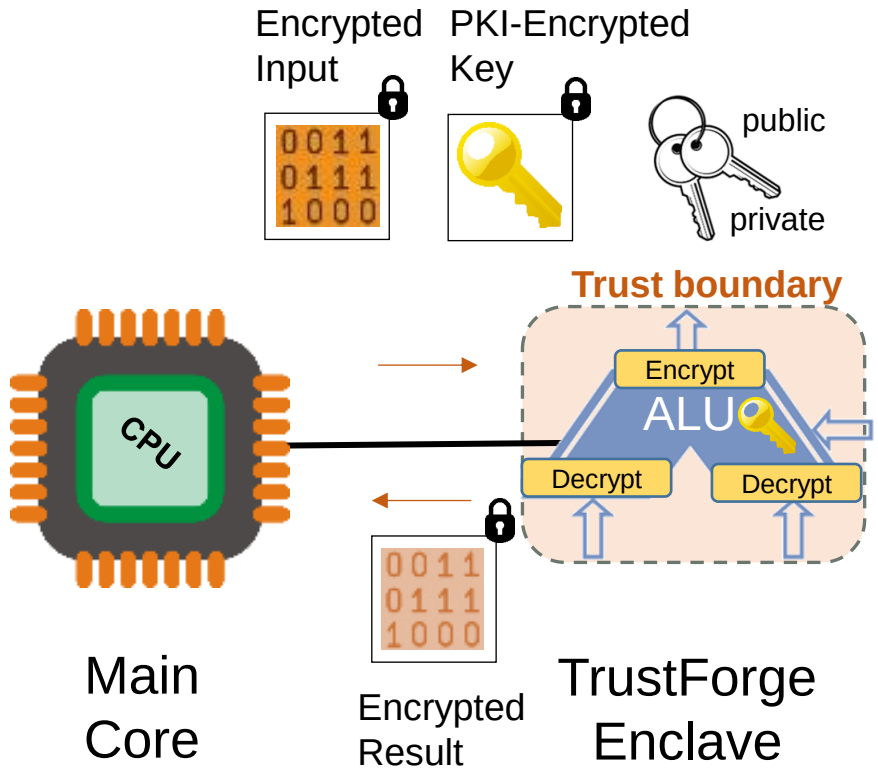
- The key to addressing security/privacy challenges is simply a privatized ***functional unit***…

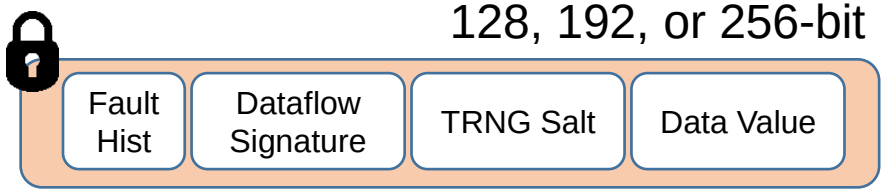


- Denying programmers the ability to see the data they are processing reduces the utility of S/W and H/W hacking and significantly advance data privacy!
- NOTE: This is not homomorphic encryption, it is just privacy-oriented microarchitecture!

TrustForge Encrypted Computation

- TrustForge enclave ***exports a public key***, PKI key exchange occurs inside enclave
- Enclave ***supports RISC operations directly on always-encrypted data***
- Enclave is decrypting, processing, checking, and re-encrypting secret data ***without S/W vulnerabilities or digital side channels***
- Enclave is protected from physical attacks***, no other S/W or H/W is trusted!***

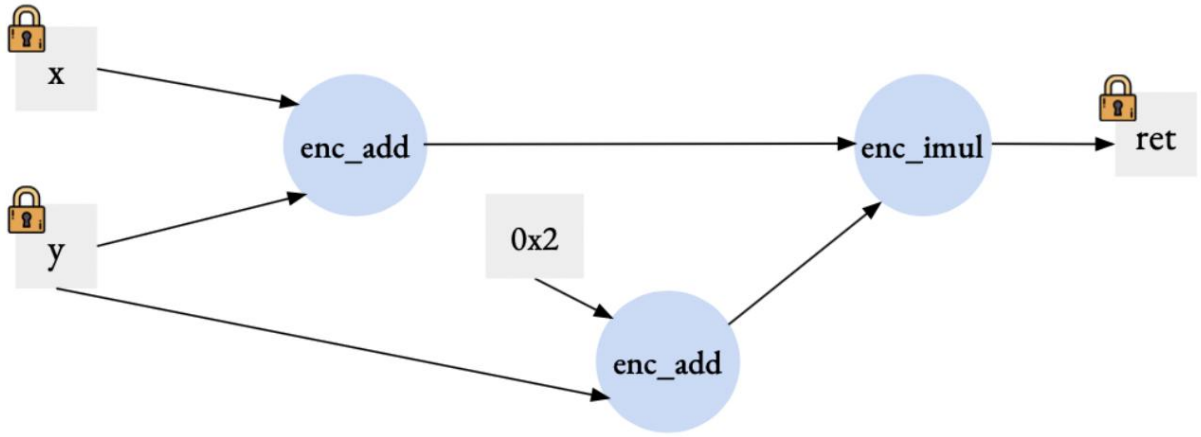




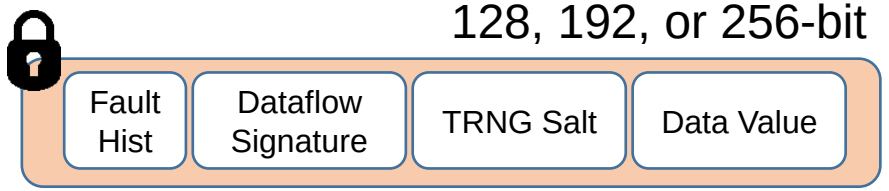
TrustForge Verified Computation

- ***Dataflow signature*** describes dataflow used to create a value
  - Inputs parameters to the computation
  - Order of ops and their dependencies
  - Implemented as a 1-way Merkle hash
- Kept inside encrypted packet
  - Any manipulation is detected
- Crypto-strength integrity defenses
  - Swapping in alternate inputs/outputs
  - Adding/deleting instructions
  - Reordering of operands
  - Changing of dependencies/dataflow
  - Manipulation of regs or memory
  - Replay of insts, regs, or memory

```
enc_add   %ecx, %edx        //t1   = x + y
enc_add   %edx, 0x2         //t2   = y + 2
enc_imul  %ecx, %edx        //ret  = t1 * t2
```



$$H_{t1} = \text{hash}(I_x, I_y, \text{opcode\_enc\_add})$$
$$H_{t2} = \text{hash}(I_y, 0x2, \text{opcode\_enc\_add})$$
$$H_{ret} = \text{hash}(H_{t1}, H_{t2}, \text{opcode\_enc\_imul})$$



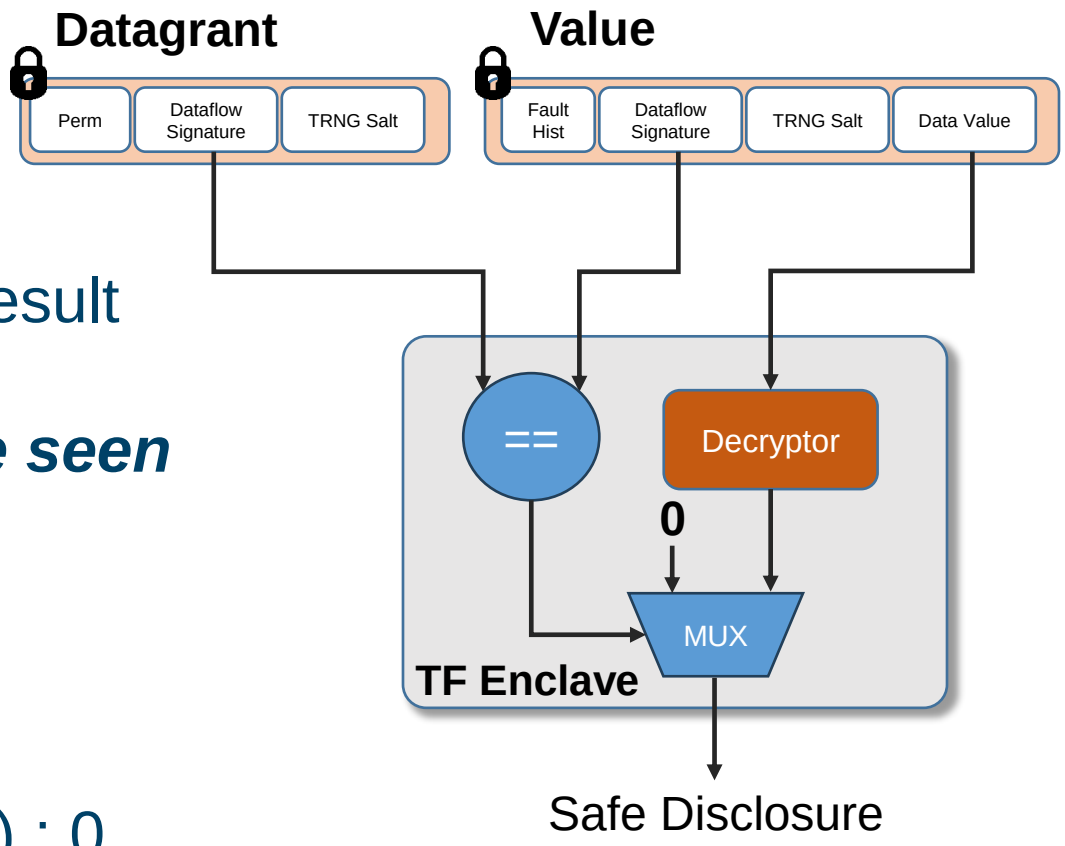
TrustForge Safe Disclosures

- Built on verified computation
  - Disclose if key signature reproduced
- Data owner provides a ***datagrant***
  - Encrypted dataflow hash of disclosable result
  - Permission: decrypt, hash, re-encrypt
  - ***Ensures only pre-approved results are seen***
- Datagrant disclosure semantics:

$hash_{dg} \leftarrow dec_k(datagrant)$

$hash_{val} \leftarrow dec_k(value)$

$value_{grant} \leftarrow hash_{dg} == hash_{val} \; ? \; dec_k(value) : 0$



Programming for TrustForge

- Extends language with encrypted variables
- ***Bit-for-bit compatible*** to unprotected computation, but encrypted
- Built-in support for ***integers, floating-point, Booleans,*** and ***strings***
- Operators on encrypted variables ***return an encrypted result and encapsulate faults***
- Decision processing on encrypted variables implemented with ***CMOV primitive***
- Secret-dependent array indexing implemented with ***ORAM primitives***

Type Class	C++ Data Types
Integer	tfInt32, tfUInt32, tfInt64, tfUInt64
Floating Point	tfFloat, tfDouble
Boolean	tfBool
String/Char	tfChar, tfUChar, tfString

```
1   x = enc_cmov( secret, x+1, x);
2   y = enc_cmov(!secret, y+1, y);
```

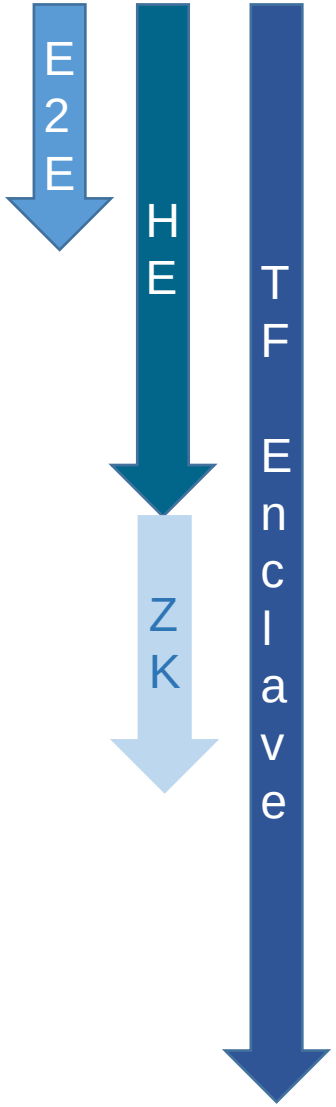
```
1  for(int i=0;  i<len(arr);  i++;)
2     ret = enc_cmov(i==secret, arr[i], ret);
```

Newton-Raphson Solver on TrustForge

```
// calculate fn value using Newton-Raphson method
tfDouble
rn_solver(tfBool& converged, double maxerr,
          unsigned maxiter, fn_type f, fn_type df)
{
  unsigned iter;
  tfDouble guess = 1.0;

  converged = false;
  for (iter = 0; iter < maxiter; iter++)
  {
      converged = tfFabs(f(guess)) <= maxerr;
      seDouble newGuess = guess - f(guess)/df(guess);
      guess = tfCMOV(converged, guess, newGuess);
  }
  return guess;
}
```

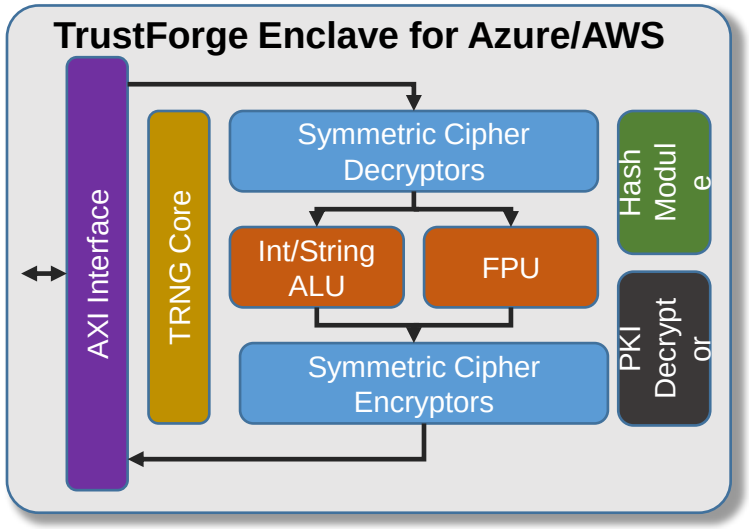
TrustForge Enclave Capabilities



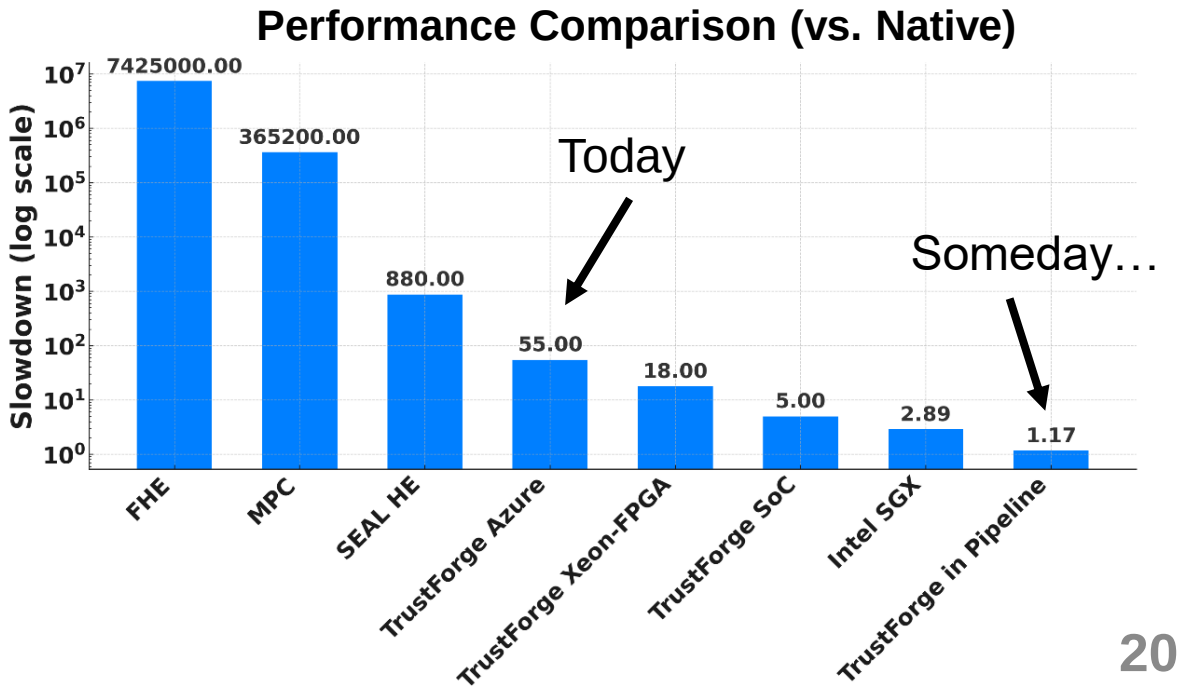
- ***Data encryption*** ensures that sensitive data is protected everywhere in the system with high-entropy ciphers
- ***Encrypted computation*** ensures that hackers and programmers cannot observe sensitive computation
- ***Verified computation*** gives end-to-end integrity receipts to ensure that hackers/programmers cannot change dataflow
- ***Safe disclosures*** allow programmers and data owners to cooperatively decrypt specific computation results in a way that hackers or programmers cannot exploit

TrustForge Enclave for Azure/AWS

- Deployed on AWS/Azure FPGA nodes
  - 6% of total UltraScale+ FPGA, ***~190k gates***
  - ***Logic locked, watermarked,*** and supporting ***forward secrecy***
  - ***2-person years*** to build, fuzz, and verify

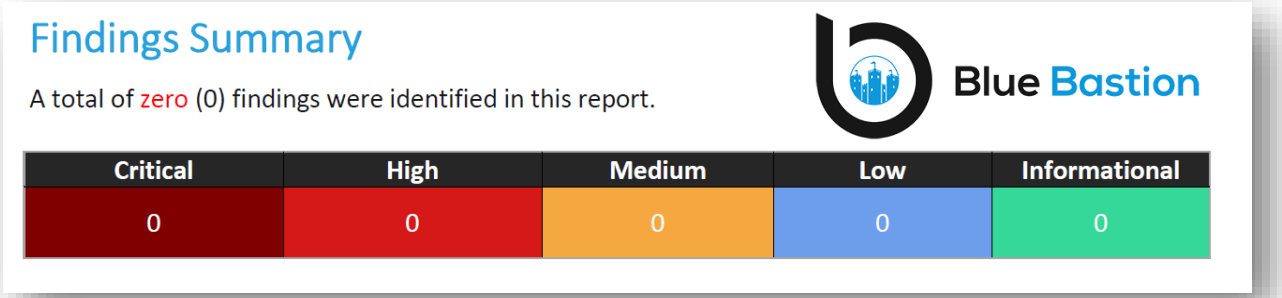


- TrustForge competes with
  - Traditional TEEs
  - Homomorphic encryption (HE/FHE)
  - Multiparty computation (MPC)
  - **Faster than Integer HE/FHE, more secure than TEEs**
  - **1Mx+ faster than Integer FHE**



TrustForge Security Analysis

- Worked with In-Q-Tel & Blue Bastion, pen tested for 3 months
  - Red team had full access to all IP
  - ***Zero vulnerabilities found***
- Formal verification with Princeton
  - Secure for ***any program*** on our ***TrustForge implementation***
  - ***Zero vulnerabilities found***
  - Won award at ACM CCS 2023



Findings Summary

A total of zero (0) findings were identified in this report.

Blue Bastion

Critical	High	Medium	Low	Informational
0	0	0	0	0

Security Verification of Low-Trust Architectures

Qinhan Tan*
qinhant@princeton.edu
Princeton University
Princeton, New Jersey, USA

Yonathan Fisseha*
yonathan@umich.edu
University of Michigan
Ann Arbor, Michigan, USA

Shibo Chen*
chshibo@umich.edu
University of Michigan
Ann Arbor, Michigan, USA

Lauren Biernacki
biernacl@lafayette.edu
Lafayette College
Easton, Pennsylvania, USA

Jean-Baptiste Jeannin
jeannin@umich.edu
University of Michigan
Ann Arbor, Michigan, USA

Sharad Malik
sharad@princeton.edu
Princeton University
Princeton, New Jersey, USA

Todd Austin
austin@umich.edu
University of Michigan
Ann Arbor, Michigan, USA

ABSTRACT

Low-trust architectures work on, from the viewpoint of software, always-encrypted data, and significantly reduce the amount of hardware trust to a small software-free enclave component. In this paper, we perform a complete formal verification of a specific low-trust architecture, the Sequestered Encryption (SE) architecture, to show that the design is secure against direct data disclosures and digital side channels for all possible programs. We first define the security requirements of the ISA of SE low-trust architecture. Looking upwards, this ISA serves as an abstraction of the hardware for the

ACM Reference Format:
Qinhan Tan, Yonathan Fisseha, Shibo Chen, Lauren Biernacki, Jean-Baptiste Jeannin, Sharad Malik, and Todd Austin. 2023. Security Verification of Low-Trust Architectures. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23), November 26–30, 2023, Copenhagen, Denmark.* ACM, New York, NY, USA, 19 pages. https://doi.org/10.1145/3576915.3616643

1 INTRODUCTION

Parting Thoughts…

- What this work has taught me…
  1. We will ***never*** stop software hacking, so ***remove trust in S/W***
  2. We will ***never*** stop uArch side channels, so ***minimize trust in H/W***
  3. We will ***only*** stop side channels if we ***isolate*** and ***formally verify*** trusted H/W
  4. Since ***cryptography is the only superpower in security***, use it elsewhere

- What an opportune moment it is to be a H/W security architect!
  - Durable security can only come from ***H/W security*** or ***advanced cryptography***
  - H/W security is more programmable and performant than alternatives
  - Thus, we can create powerful solutions to ***address privacy challenges today***
  - In the form of ***efficient cryptographic-strength enclaves*** that support ***programmer-friendly privacy-programming paradigms***

To Learn More about TrustForge…

- "Sequestered Encryption: A Hardware Technique for Comprehensive Data Privacy", 2022 IEEE SEED Conference
- "Security Verification of Low-Trust Architectures", 2023 IEEE CCS
- "Privacy-Enhanced Computation via Sequestered Encryption", US Patent 12,105,855
- "Privacy-Enhanced Computation via Sequestered Encryption", US Patent 11,748,521
- Agita Labs, https://agitalabs.com
- Hardware Security Tutorial, https://www.youtube.com/playlist?list=PLPokM2qEmTDClgPTX_GOLeMZkuf7o38yS