



BUILDING SECURE (CLOUD) APPLICATIONS USING INTEL'S SGX

FLORIAN KERSCHBAUM, UNIVERSITY OF WATERLOO

JOINT WORK WITH BENNY FUHRY (SAP), ANDREAS FISCHER (SAP)
AND MANY OTHERS

DO YOU TRUST YOUR CLOUD SERVICE PROVIDER?

- Frequent break-ins
 - Equifax, Yahoo, ...
- High-profile target leakages
 - Jennifer Lawrence, Game of Thrones, ...
- Mass surveillance
 - Snowden revelations, ...

ENCRYPTION?

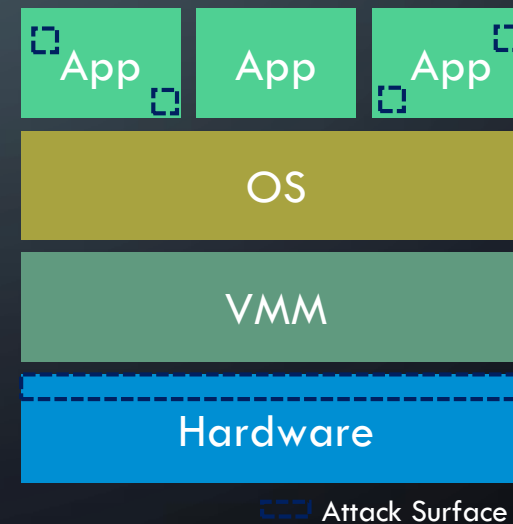
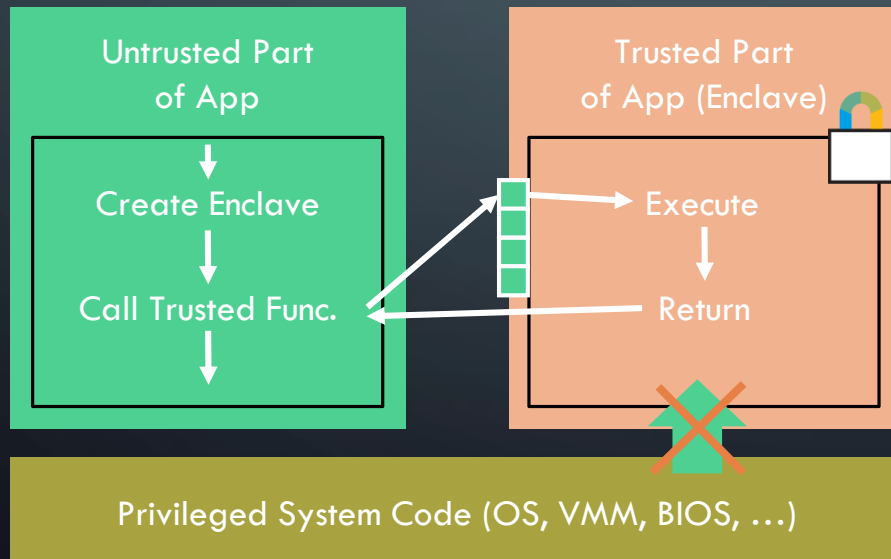
- Encryption during computation still difficult
 - Fully homomorphic encryption is too slow and too space-inefficient
 - Multi-party computation is still slow and communication-inefficient, also doesn't fit service provider model (needs many service providers)
- Intel developed a hardware-supported encryption mechanism: Software Guard Extensions (SGX)

The background is a dark blue gradient. In the corners, there are white, stylized circuit board traces with circular nodes, resembling a network or data flow diagram. These traces are located in the top-left, top-right, bottom-left, and bottom-right corners.

INTEL SGX – WHAT IS IT?

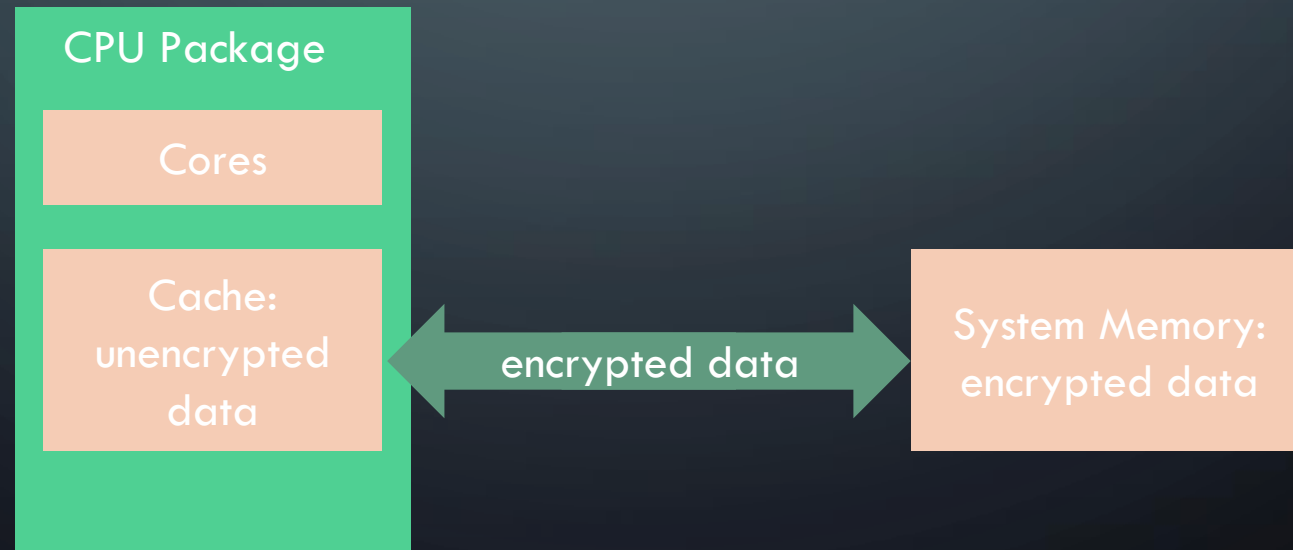
INTEL SGX – WHAT IS IT?

- Extension to CPU instructions introduced with Intel Skylake
- Isolated execution of specific parts: trusted (**enclave**)
- Small attack surface: processor + enclave



INTEL SGX – MEMORY PROTECTION

- Enclave memory is encrypted and protected against replay attacks
- Security perimeter is the CPU package boundary
- Tapping memory or BUS reveals only encrypted data



INTEL SGX – REMOTE ATTESTATION

- Enclave proves that the loaded code is as expected
 - Enclave measured during creation (code is cryptographically hashed)
 - Enclave requests REPORT (measurement signed by processor's private key)
 - REPORT is transformed to QUOTE
 - QUOTE is sent to remote party
 - Remote party checks validity of QUOTE

INTEL SGX – BOOTSTRAPPING ENCRYPTED COMPUTATION

- Load code into enclave
- Remotely attest integrity of code
- Create public-/private-key pair in encrypted memory
- Send public key as part of QUOTE to client
- Perform TLS authentication with endpoint in enclave

(SECURITY) CHALLENGES WHEN USING SGX

- Side channels
 - Memory access pattern
 - Shared Cache
- Resource constraints
 - 96MB enclave (without paging)
- Software Vulnerabilities
 - Hardware protection can be broken by malicious code inside the enclave

HOW TO IMPLEMENT SGX-SUPPORTED APPLICATIONS

- Carefully decide which parts to put into an enclave
 - Resource efficient design
 - Small code base (→ small number of errors)
 - Useful in many application contexts

IN THIS TALK

- **SGX-supported implementation of a database index**
 - Benny Fuhry, Benny Fuhry, Raad Bahmani, Ferdinand Brasser, Florian Hahn, Florian Kerschbaum, Ahmad-Reza Sadeghi. *HardIDX: Practical and Secure Index with SGX* (DBSEC 2017 Best Paper Award)
- **SGX-supported implementation of generic encrypted computation**
 - Andreas Fischer, Benny Fuhry, Florian Kerschbaum, Eric Bodden. *Computation on Encrypted Data using Data Flow Authentication*

The background is a dark blue gradient. In the four corners, there are white, stylized circuit board traces. These traces consist of straight lines that turn at right angles, ending in small circles that represent components or connection points. The traces are more densely packed in the corners and become sparser towards the center.

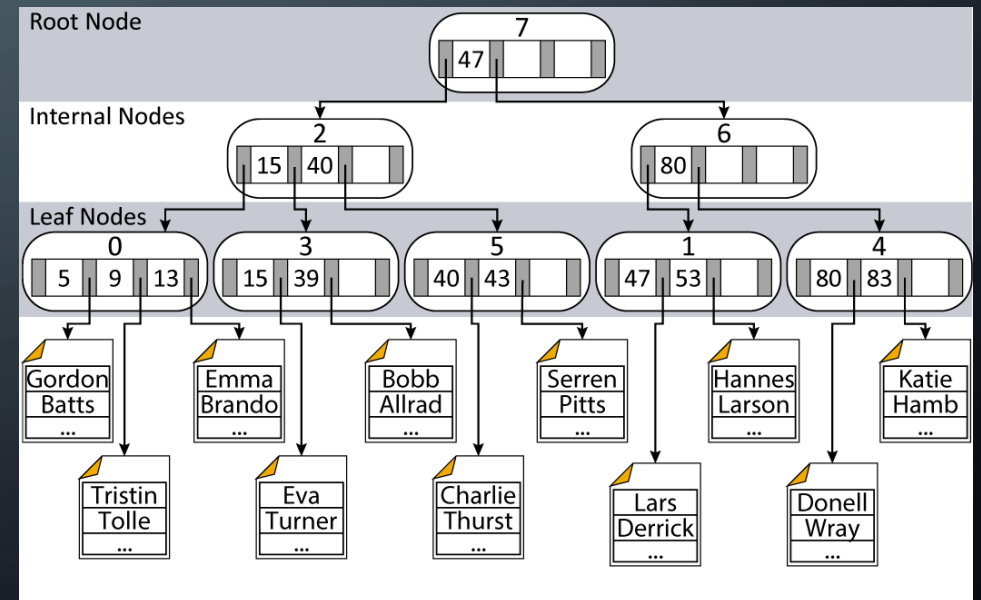
HARDIDX: A SECURE INDEX WITH SGX

OBJECTIVES

- **Search for values and ranges in in untrusted environment**
 - Much faster, but less secure than Fully Homomorphic Encryption
 - Faster and as secure as Searchable Encryption
 - As fast as, but more secure than Order Preserving Encryption
 - Small code base in Trusted Execution Environment

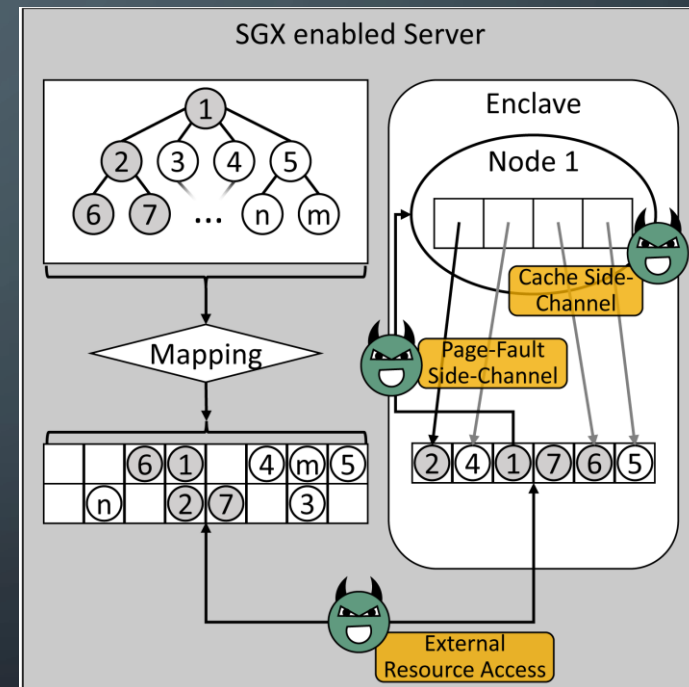
BUILD SECURE INDEX – SECURE B⁺-TREE

- Supports point and range queries
- Contains key-value pairs
- Typical index structure for databases and file systems
- Branching factor: maximal number of children
- Special case: unchained leaf nodes



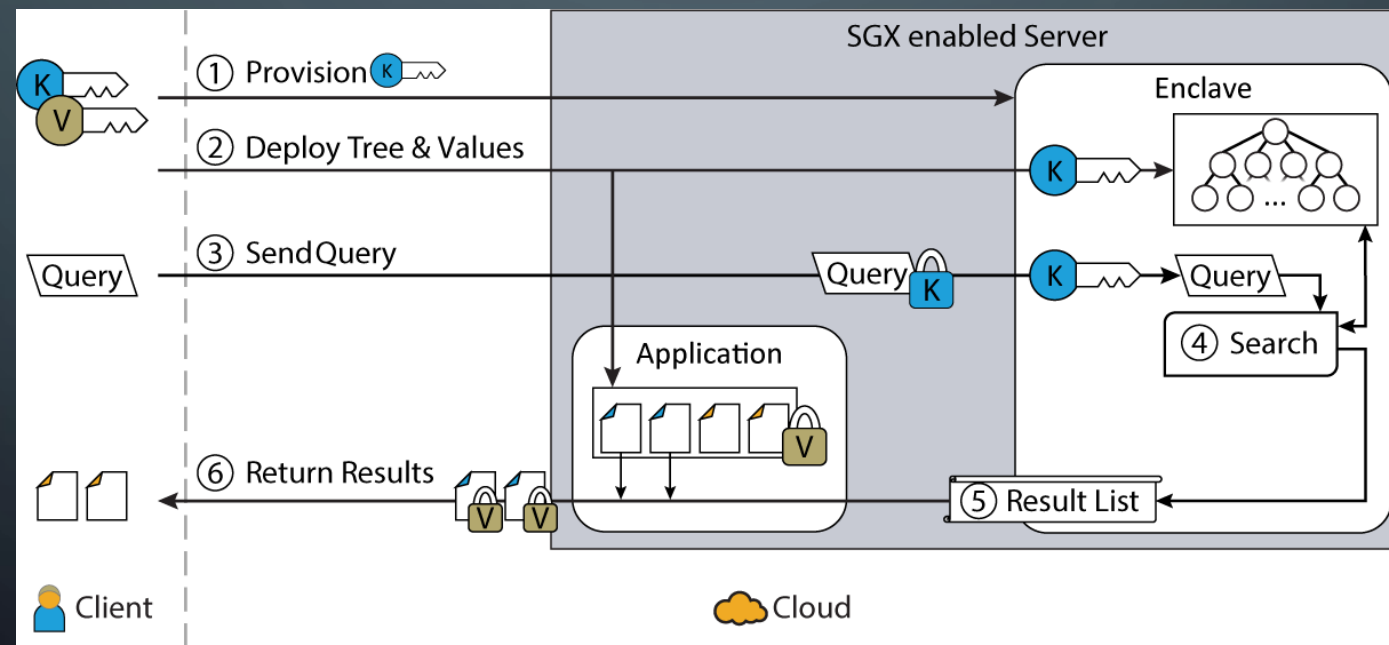
ATTACKER MODEL

- Attacker
 - controls all software (e.g., OS, firmware, BIOS)
 - observes all interactions between untrusted and trusted part
 - uses page-fault side channel to observe access pattern
 - uses cache side channel to learn about code path and data access



CONSTRUCTION 1

- Idea: data stored and processed completely inside SGX enclave



CONSTRUCTION 1 - PROBLEM

- Usable enclave memory limited to 96MB
 - Paging leads to performance problems
- B+-Tree only small part of full DBMS

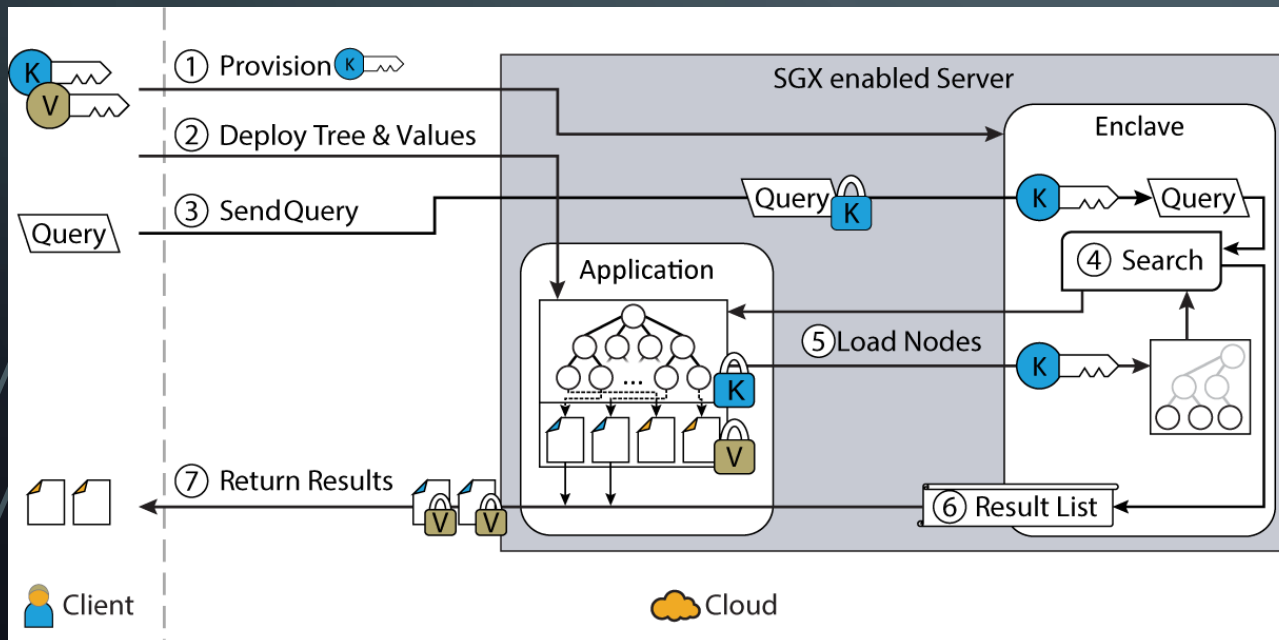


CONSTRUCTION 2

- Idea: only load required nodes into enclave

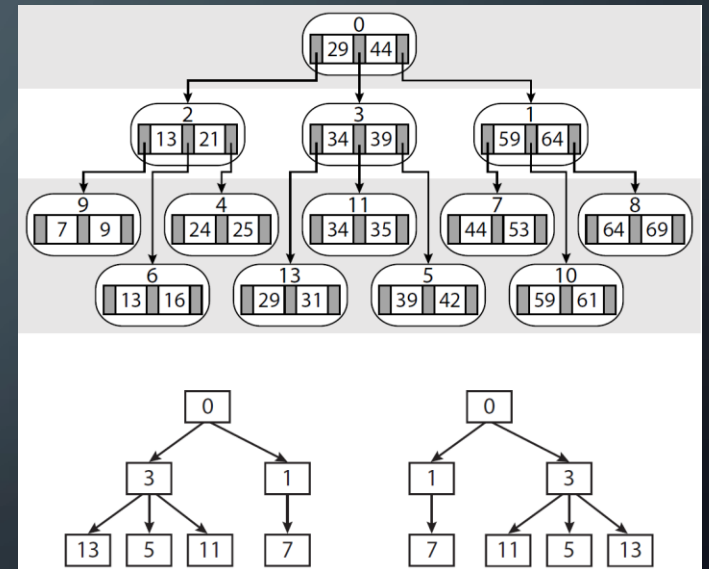
- Space-Time Trade-Off

- Constant enclave memory for arbitrary tree size
- Additional decryption step in enclave



CONSTRUCTION 2 – IDEA OF SECURITY PROOF

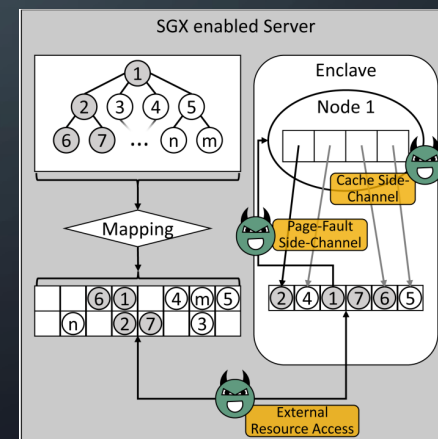
- CKA2-HW-security:
 - Define leakage of the construction
 - Simulator is able to simulate the adversary's view of the enclave-(untrusted) application interaction given only leakage
 - Our leakage:
 - Amount of values, size of each value, amount of nodes in tree
 - Storage position of all nodes that get accessed at traversal
 - Pointers to result values together with the leaf nodes in which these pointers are stored



Example tree access pattern for two snapshots
($R = [33, 55]$)

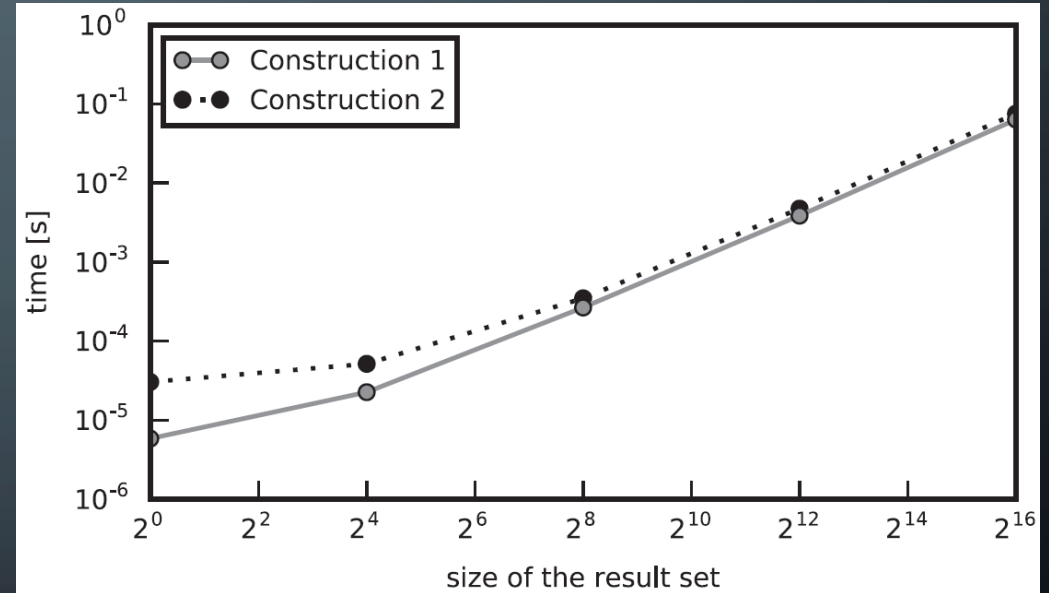
SGX SIDE-CHANNELS

- Proof considers side channels
- External resource access: proof considers interactions between untrusted and trusted part
- Page-fault side channel: construction does not reveal additional information (nodes smaller than one page)
- Cache side channel: nodes are completely traversed and implementation is data independent



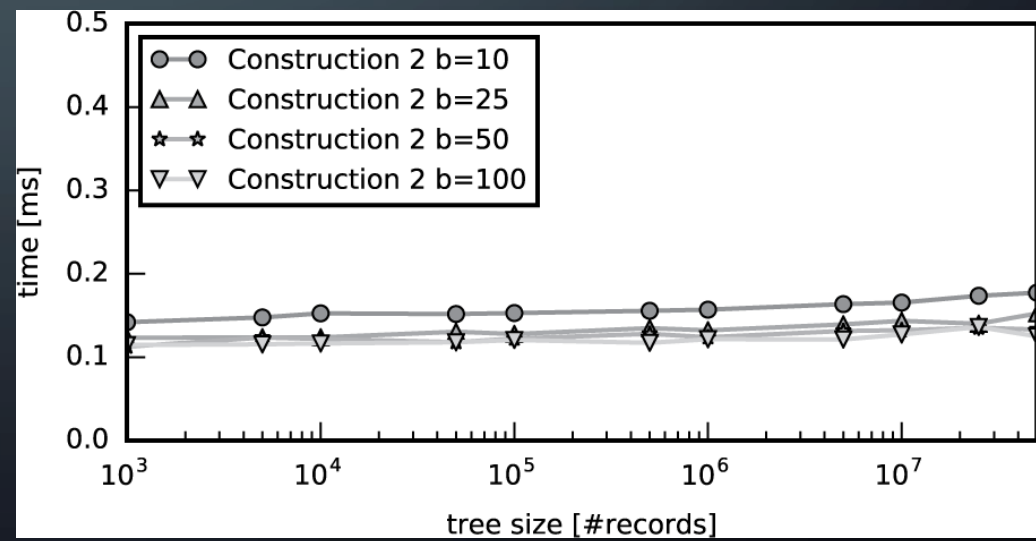
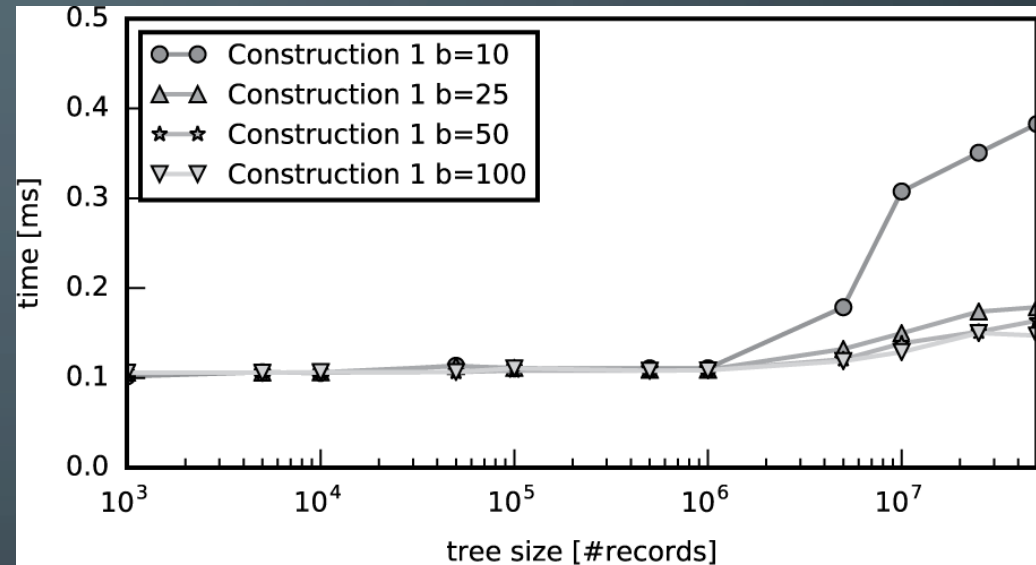
MICRO BENCHMARKS – COMPARISON OF CONSTRUCTIONS

- Setting:
 - 1.000.000 key-value pairs
 - 1000 randomly selected ranges
 - Different result set sizes
- Performance factors:
 - Processor mode switch: Construction 1 < Construction 2
 - Data transfer: result set and query at Construction 1, additionally nodes at Construction 2
 - Access to plain data: one time effort at Construction 1, per node and query at Construction 2
- Effects diminish compared to search time of algorithm



MICRO-BENCHMARKS – MEMORY MANAGEMENT

- Setting
 - 1.000 randomly chosen queries with result set size of 100
 - Different branching factors, different tree sizes
- Main finding:
 - Sharp increase above a tree size of $(10)^6$ at Construction 1
- Explanation:
 - The lower the branching factor, the higher the number of accesses to different memory pages
 - Swapping is very costly



WHAT WE ACHIEVED

- An encrypted index
 - That is even faster than naïve full-memory SGX encryption for large databases
 - Security roughly comparable to searchable encryption that considers side-channels in a formal model

The background is a dark blue gradient. In the four corners, there are decorative white line-art patterns resembling circuit traces or neural network connections. These patterns consist of straight lines of varying lengths and angles, ending in small white circles.

DFAUTH: COMPUTATION ON ENCRYPTED DATA WITH DATA FLOW AUTHENTICATION

THERE IS A THIRD APPROACH

- Fully Homomorphic Encryption
- Partial Encryption
 - MRCrypt
 - JCrypt
 - AutoCrypt
- Hardware Support (SGX)

IDEA OF PARTIAL ENCRYPTION

- Encrypt using scheme that supports required operation
 - Addition \Rightarrow partially homomorphic encryption
 - Multiplication \Rightarrow (other) partially homomorphic encryption
 - Comparison \Rightarrow searchable encryption
 - Etc.
- Partially leak information about execution
 - In our case: Control-flow

PROBLEM

- Use of different encryption scheme requires conversion
 - SGX enables conversion without communication with client
- Main application remains untrusted
 - SGX can be used as a conversion oracle (calls to SGX cannot be authenticated)
 - Partial leakage can be quickly amplified into full leakage (binary search) by active adversary that controls program execution

OBJECTIVE

- Limit leakage to only intended leakage (control flow)
- Approach: Data Flow Authentication

HASE: HOMOMORPHIC AUTHENTICATED SYMMETRIC ENCRYPTION

- **KeyGen()**: evaluation key ek , secret key sk
 - Executed at client
- **Encrypt**(sk , Message m , Identifier i): ciphertext c
 - Executed at client and inside enclave
- **Evaluate**(ek , $\{c_1, c_2, \dots, c_n\}$): ciphertext c
 - Execute at server by untrusted application
- **Derive**(sk , $\{i_1, i_2, \dots, i_n\}$): label l
 - Executed at compile time
- **Decrypt**(sk , Ciphertext c , Label l): message m or error symbol
 - Executed inside enclave and at client

UNFORGEABILITY

- HASE-UF-CPA: Unforgeability

- Loosely speaking: The adversary cannot decrypt unless the label was computed by Derive()

$$\frac{\text{ExpHASE}_{\mathcal{A}, \Pi}^{\text{UF-CPA}}(\lambda)}{\quad}$$
$$S := \{\}$$
$$\langle ek, sk \rangle \leftarrow \Pi.\text{Gen}(1^\lambda)$$
$$(c, I) \leftarrow \mathcal{A}^{E_{sk}}(1^\lambda, ek)$$
$$l := \Pi.\text{Der}(sk, I)$$
$$m := \Pi.\text{Dec}(sk, c, l)$$
$$\tilde{m} := \bigoplus_{(m', i) \in S, i \in I} m'$$
$$\text{return } m \neq \perp \wedge m \neq \tilde{m}$$
$$\frac{E_{sk}(m, i)}{\quad}$$
$$\text{if } i \in \pi_2(S) \text{ then}$$
$$\quad \text{return } \perp$$
$$\text{else}$$
$$\quad S := S \cup \{(m, i)\}$$
$$\quad c \leftarrow \Pi.\text{Enc}(sk, m, i)$$
$$\quad \text{return } c$$

DATA FLOW AUTHENTICATION: EXAMPLE

1. Conversion to single static assignment (SSA) form

- Assigned variable is unique label

2. Type inference

- add: Addition using HASE
- mul: Multiplication using HASE
- cmp: Comparison in trusted module

3. Insertion of type conversions (trusted module calls)

```
1  a = b + c;  
2  d = a * e;  
3  if (d > 42)  
4      f = 1;  
5  else  
6      f = 0;
```

Example Code Before Step 1

STEP 1

1. Conversion to single static assignment (SSA) form ✓

- Assigned variable is unique label

2. Type inference

- add: Addition using HASE
- mul: Multiplication using HASE
- cmp: Comparison in trusted module

3. Insertion of type conversions (trusted module calls)

```
1  a = b + c;  
2  d = a * e;  
3  d1 = d > 42;  
4  if (d1)  
5      f1 = 1;  
6  else  
7      f2 = 0;  
8  f = phi(f1, f2);
```

Example Code After Step 1

phi: Specially interpreted merge function combining the values of two assignments.

STEP 2 & 3

1. Conversion to single static assignment (SSA) form ✓

- Assigned variable is unique label

2. Type inference ✓

- add: Addition using HASE
- mul: Multiplication using HASE
- cmp: Comparison in trusted module

3. Insertion of type conversions ✓ (trusted module calls)

```
1  a = b + c;  
2  a1 = convertToMul(a, "a1");  
3  d = a1 * e;  
4  d1 = convertToCmpGT42(d, "d1");  
5  if (d1)  
6      f1 = 1;  
7  else  
8      f2 = 0;  
9  f = phi(f1, f2);
```

Example Code After Step 3

phi: Specially interpreted merge function combining the values of two assignments.

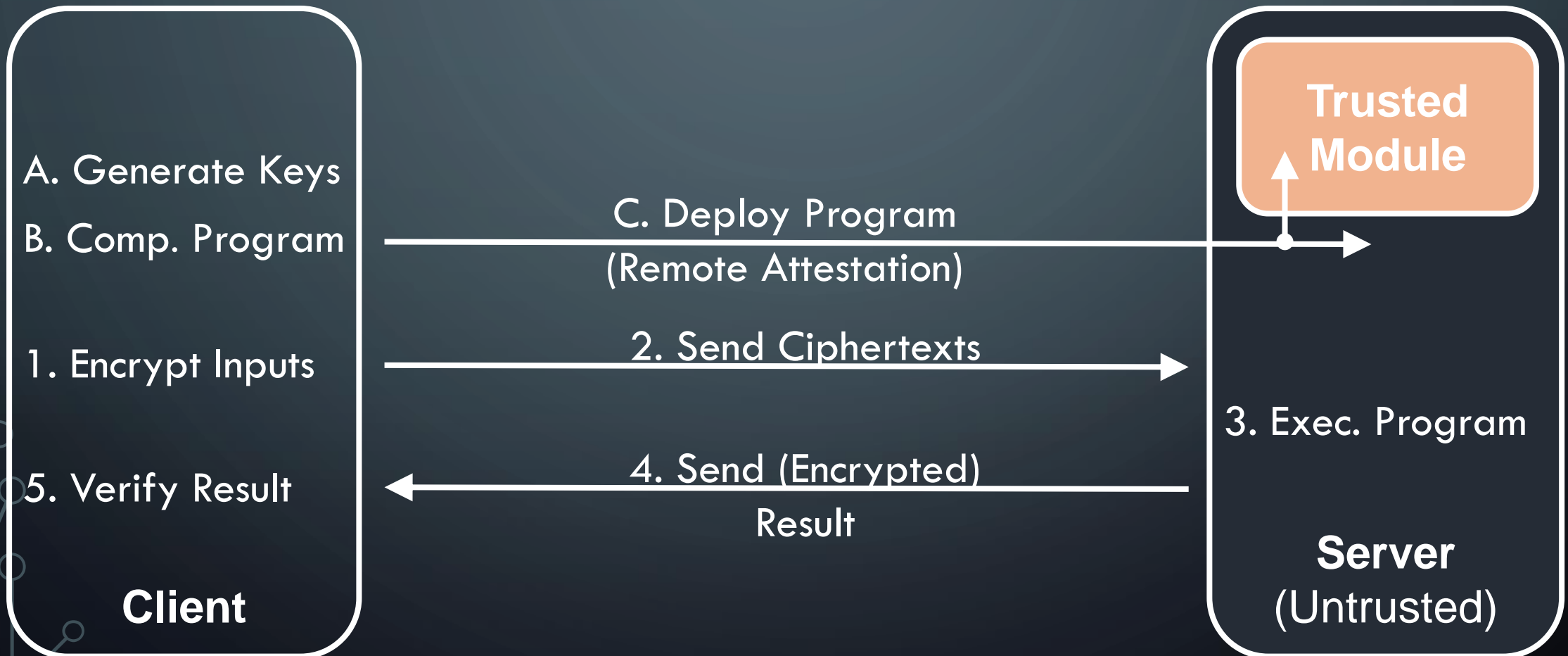
CONVERSION ROUTINE IN SGX

- Client sends to secure enclave
 - Secret keys sk
 - Mapping: assigned variable \Rightarrow HASE label
- On conversion request for variable x
 - Retrieves HASE label
 - Decrypts using HASE label
 - On decryption error halts
 - Returns ciphertext in new encryption scheme (or comparison result)

SECURITY

- Given
 - Program P
 - Memory locations M_1 and M_2
 - Classified memory locations C
- Non-interference under declassification
 - $P(M_1 \downarrow C) = P(M_2 \downarrow C)$

ENCRYPTED COMPUTATION



CASE STUDY: DEEP LEARNING

- Application of DFAuth technique to the Neuroph framework (Java, floating point numbers)
- Breast cancer neural network (image recognition)
- Encryption ensures protection of
 - Input neurons (image being classified)
 - Network weights (intellectual property)
 - Output neurons (result of classification)
- Implementation of trusted module using Intel SGX
- Time to perform evaluation of network on encrypted inputs: 0.86s (slowdown: ca. 675x)

WHAT WE ACHIEVED

- Performance
 - Much faster than Fully Homomorphic Encryption
 - Slower than SGX-only encryption
- Security
 - More leakage than Fully Homomorphic Encryption
 - Compared to SGX-only encryption **small and program-independent trusted code base**

CONCLUSION: ENCRYPTED COMPUTATION IN SGX

- Small (and trustworthy) code base is key
 - Program-independence is achievable
- Careful design for ALL side-channel is needed
- Resource optimization offers intriguing trade-offs