New York University Abu Dhabi
Modern Microprocessor Architectures Lab
nyuad.nyu.edu/momalab

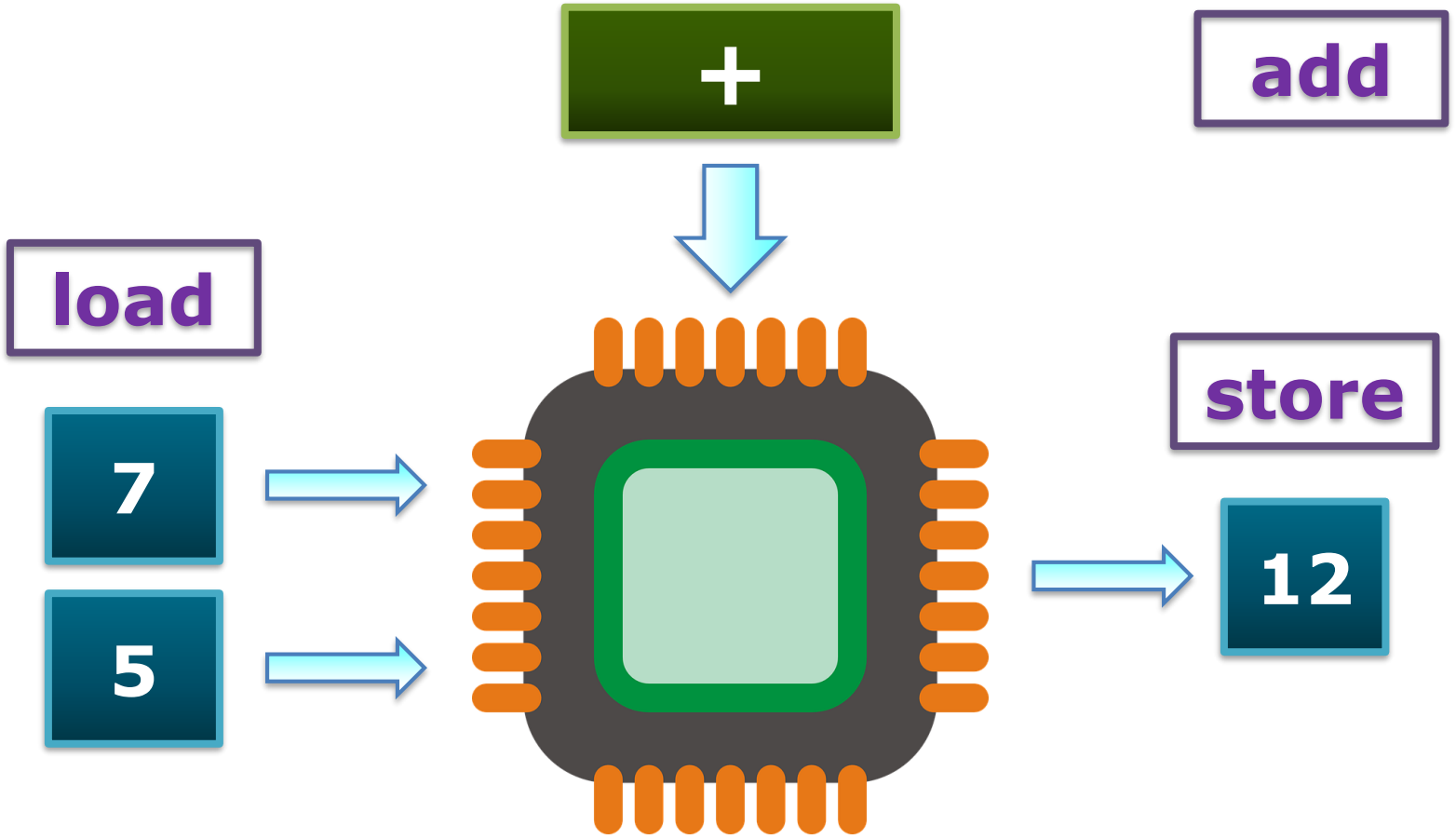
Obfuscating Branch Decisions based on Encrypted Data using MISR and Hash Digests

Nektarios Georgios Tsoutsos
Computer Science and Engineering, New York University

Michail Maniatakos
Electrical and Computer Engineering, NYU Abu Dhabi

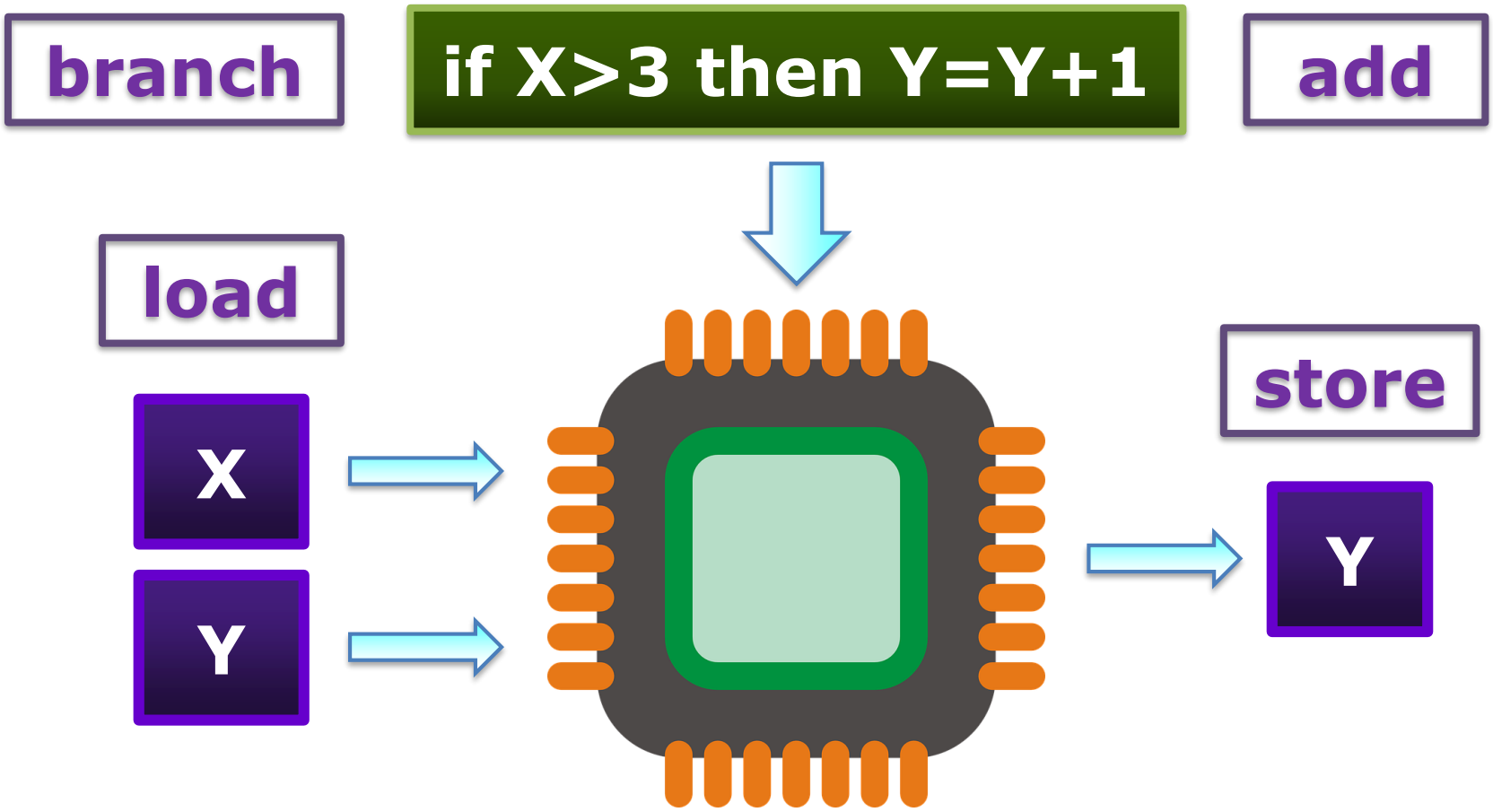
Is this a computer?

Not general-purpose



Is this a computer?

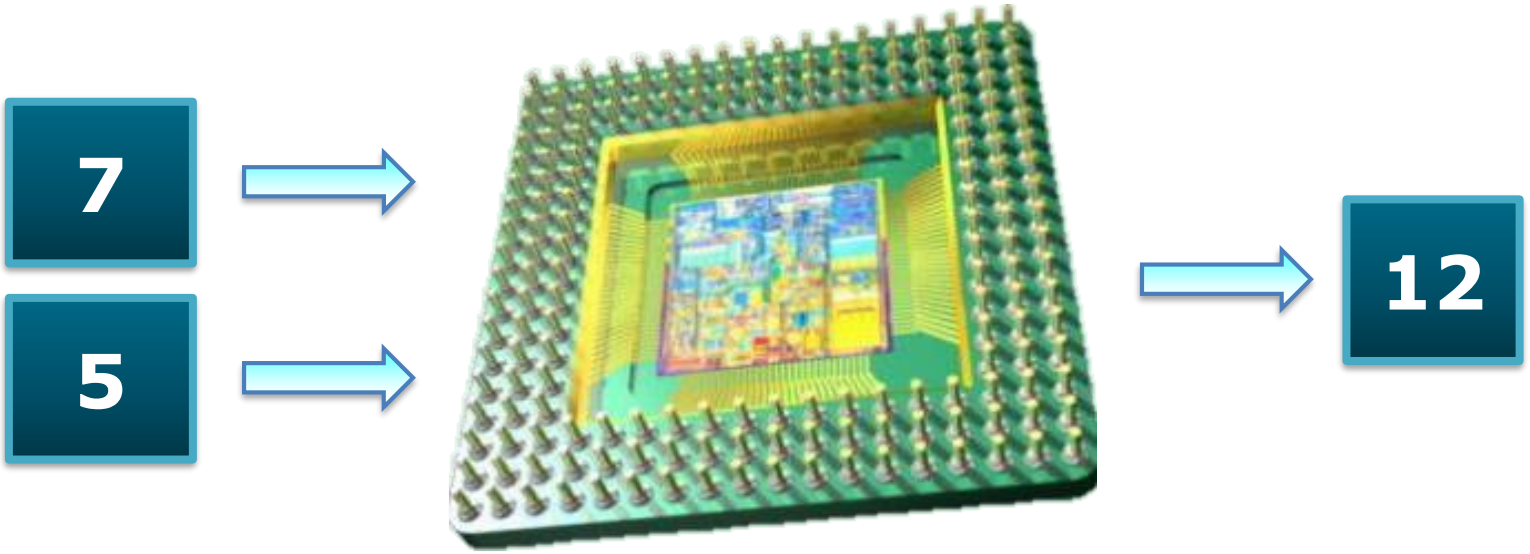
Yes, general-purpose (GP)



Encrypted Computation

The primary objective is data PRIVACY

- Imagine a machine that can process encrypted data natively



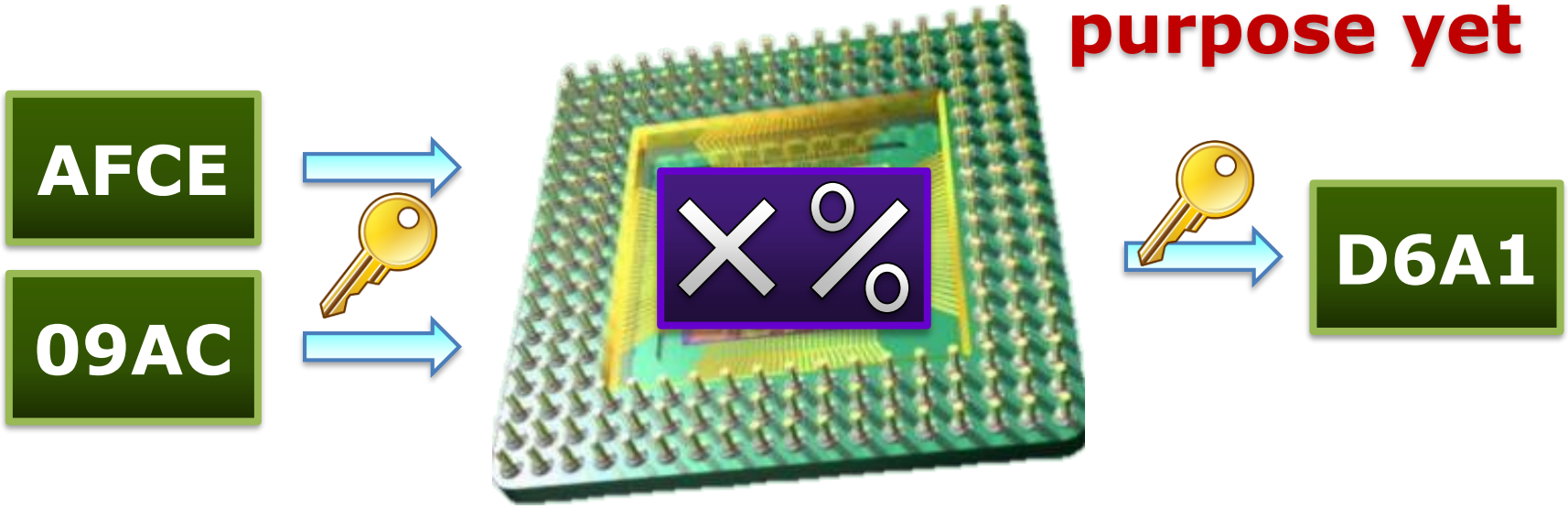
- This is possible by encrypting the data using a homomorphic encryption scheme

Encrypted Computation

The primary objective is data PRIVACY

- Imagine a machine that can process encrypted data natively

Not general-purpose yet



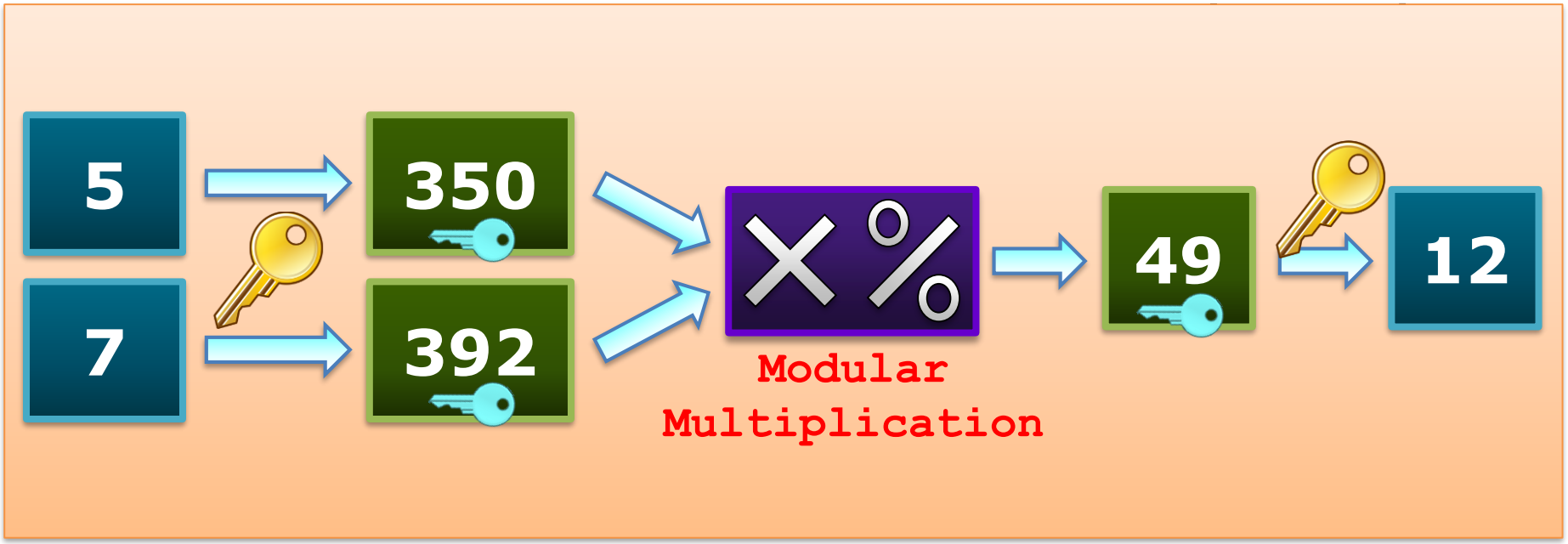
- More complicated machines support **GP** encrypted computation (e.g., Cryptoleq)

Mazonka, Tsoutsos and Maniatakos. IEEE TIFS 11.9 2016

Homomorphic Encryption

- Some asymmetric encryption algorithms allow algebraic operations on ciphertexts
- Consider **Paillier** encryption:
  - $\mathcal{N} = p \cdot q$, for random primes p and q;
  - $\mathcal{E}_{\mathcal{N}}(m, r) = (\mathcal{N} + 1)^m \cdot r^{\mathcal{N}} \bmod \mathcal{N}^2$, for a random parameter $r \in \mathbb{Z}^*_{\mathcal{N}}$ and $m \in \mathbb{Z}_{\mathcal{N}}$
- One can observe that:
  - $\mathcal{E}_{\mathcal{N}}(m_1, r_1) \cdot \mathcal{E}_{\mathcal{N}}(m_2, r_2) = \mathcal{E}_{\mathcal{N}}(m_1 + m_2, r_1 \cdot r_2)$
  - This encryption is additive homomorphic

Homomorphic Encryption



- One can observe that:
  - $\mathcal{E}_{\mathcal{N}}(m_1, r_1) \cdot \mathcal{E}_{\mathcal{N}}(m_2, r_2) = \mathcal{E}_{\mathcal{N}}(m_1 + m_2, r_1 \cdot r_2)$
  - This encryption is additive homomorphic

Problem Formulation

- We want general purpose computation in the encrypted domain
  - We need to make branch decisions using encrypted control values
  - This can create a side-channel and may leak plaintext information
  - Simple LUTs may allow static analysis

- **Our solution**: We obfuscate the connection between control values and the decision outcomes at runtime

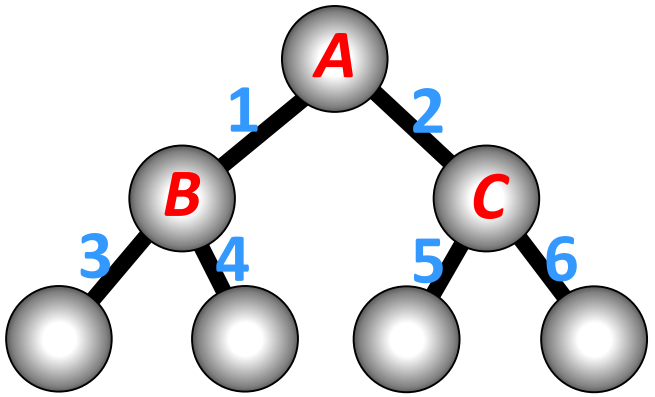
Our approach in a nutshell

- Ignore the control values and execute both branch paths non-deterministically
- Digest the encrypted control values using a 1-way compression function
- Use this digest to index a lookup table and recover the branch information
- Reconcile non-deterministic paths as soon as the branch is resolved correctly
  - Lazily reconcile after multiple branch levels

Control flow graph visual example

Key	Value
…	…
$\mathcal{H}_k(A, B, C)$	$2, 5$
…	…
…	…

Path	Mem Updates
$1, 3$	$X = 5, Y = 3$
$1, 4$	$X = 6, Y = 9$
$2, 5$	$X = 0, Y = 2$
$2, 6$	$X = 1, Y = 8$
…	…



Non-deterministic execution

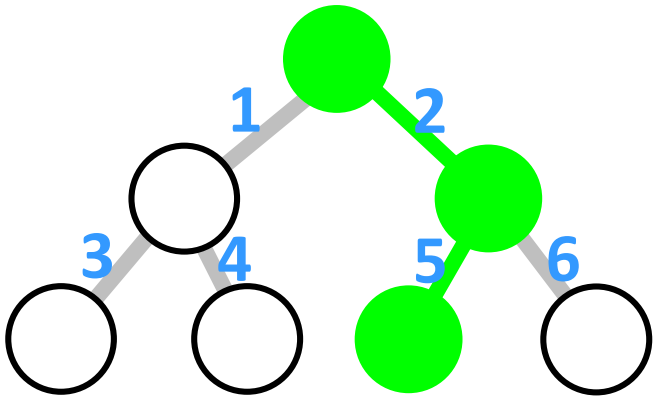
Digest A, B, C

Determine path to commit

Control flow graph visual example

Key	Value
…	…
$\mathcal{H}_k(A, B, C)$	$2, 5$
…	…
…	…

Path	Mem Updates
$1, 3$	$X = 5, Y = 3$
$1, 4$	$X = 6, Y = 9$
$2, 5$	$X = 0, Y = 2$
$2, 6$	$X = 1, Y = 8$
…	…



Non-deterministic execution

Digest A, B, C

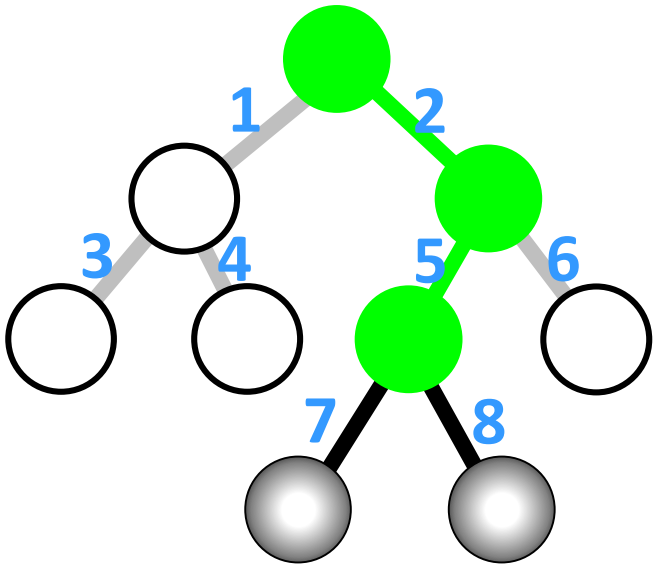
Determine path to commit

Reconcile memory updates

Control flow graph visual example

Key	Value
…	…
$\mathcal{H}_k(A, B, C)$	$2, 5$
…	…
…	…

Path	Mem Updates
$1, 3$	$X = 5, Y = 3$
$1, 4$	$X = 6, Y = 9$
$2, 5$	$X = 0, Y = 2$
$2, 6$	$X = 1, Y = 8$
…	…

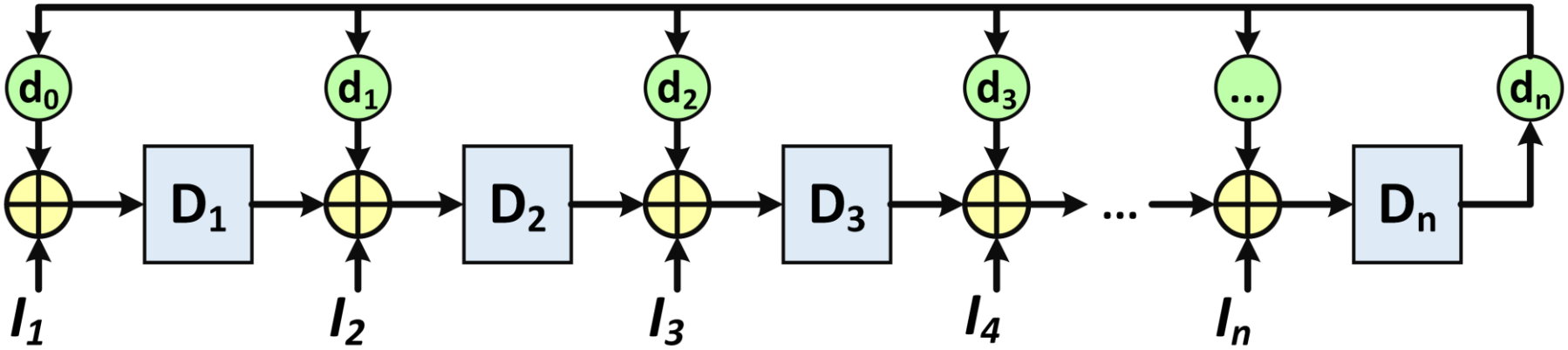


Rinse and repeat

Compression functions

Using polynomial fields

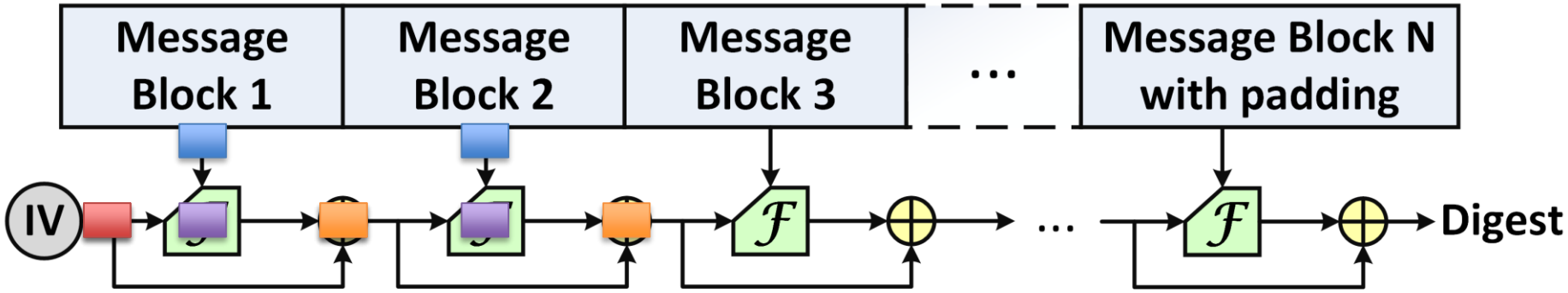
- Multiple Input Signature Register (MISR)



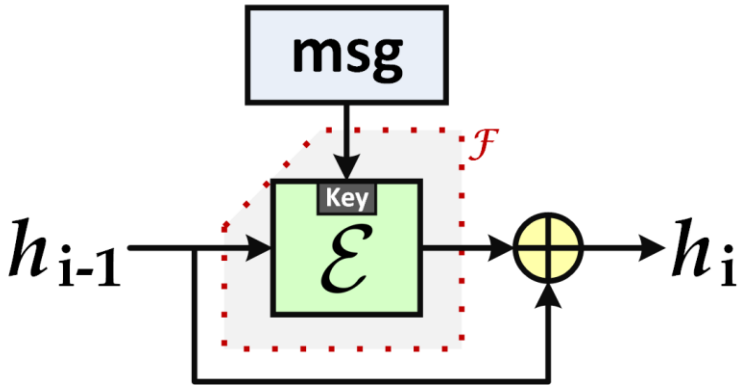
  - Uses a characteristic polynomial $\sum d_i x^i$
  - For example $x^{64} + x^4 + x^3 + x + 1$ is a suitable primitive polynomial
  - Very efficient hardware implementations

Hash functions

- Merkle-Damgard transformation



- Requires a compression function $\mathcal{F}$



Case study: the **`addleq`** machine

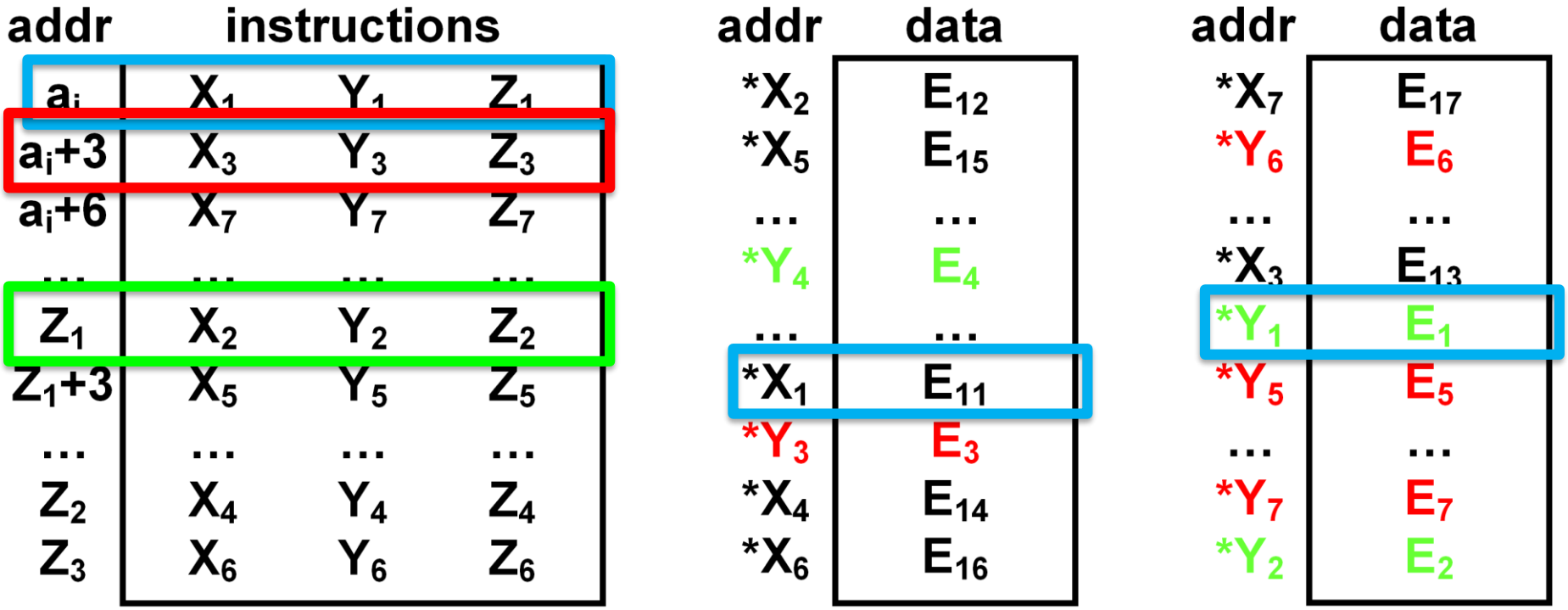
ADD and BRANCH if LESS THAN or EQUAL to ZERO

- Single instruction computer architecture
- Each **`addleq`** instruction uses 3 args
- Fetch X, Y, Z; dereference & fetch *X, *Y
- Add *X to *Y and write back: *Y=*Y+*X
- Branch to Z if *Y≤0
  - Else continue to the next 3-tuple of args
  - Two alternative branch outcomes:

    Path-BRANCH-TAKEN, path-NOT-TAKEN

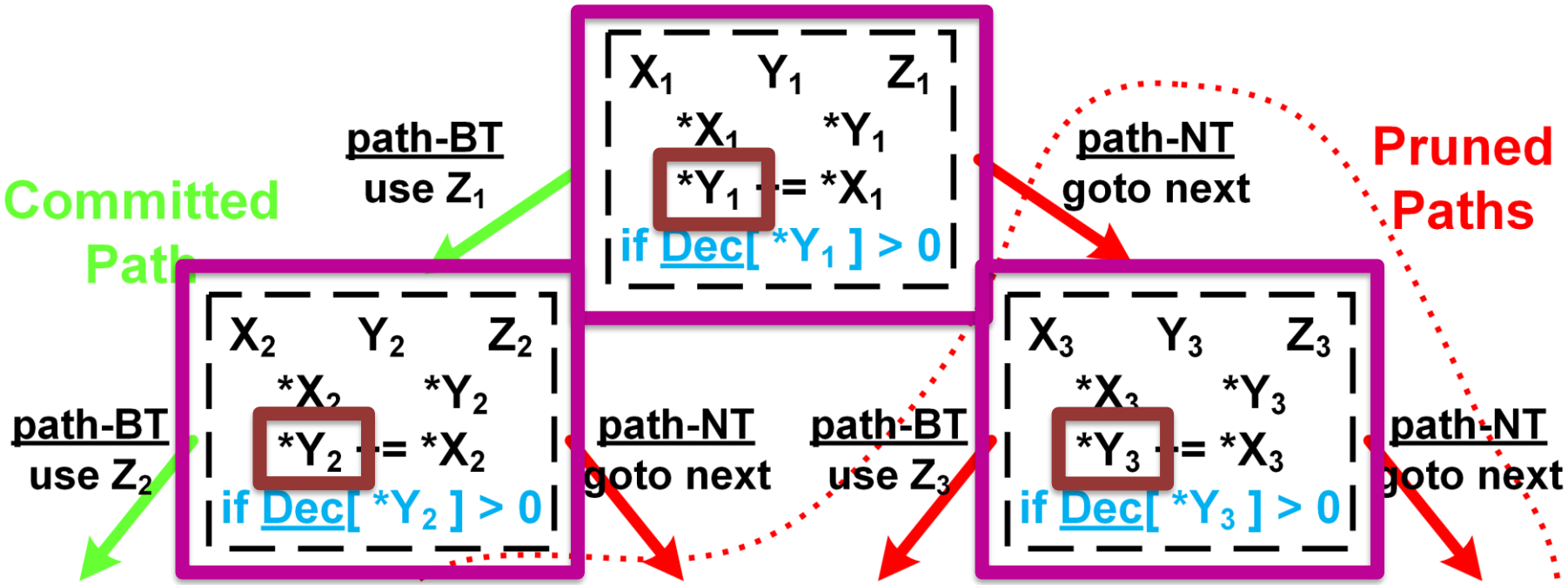
`addleq` memory organization

- Shared memory with instructions & data



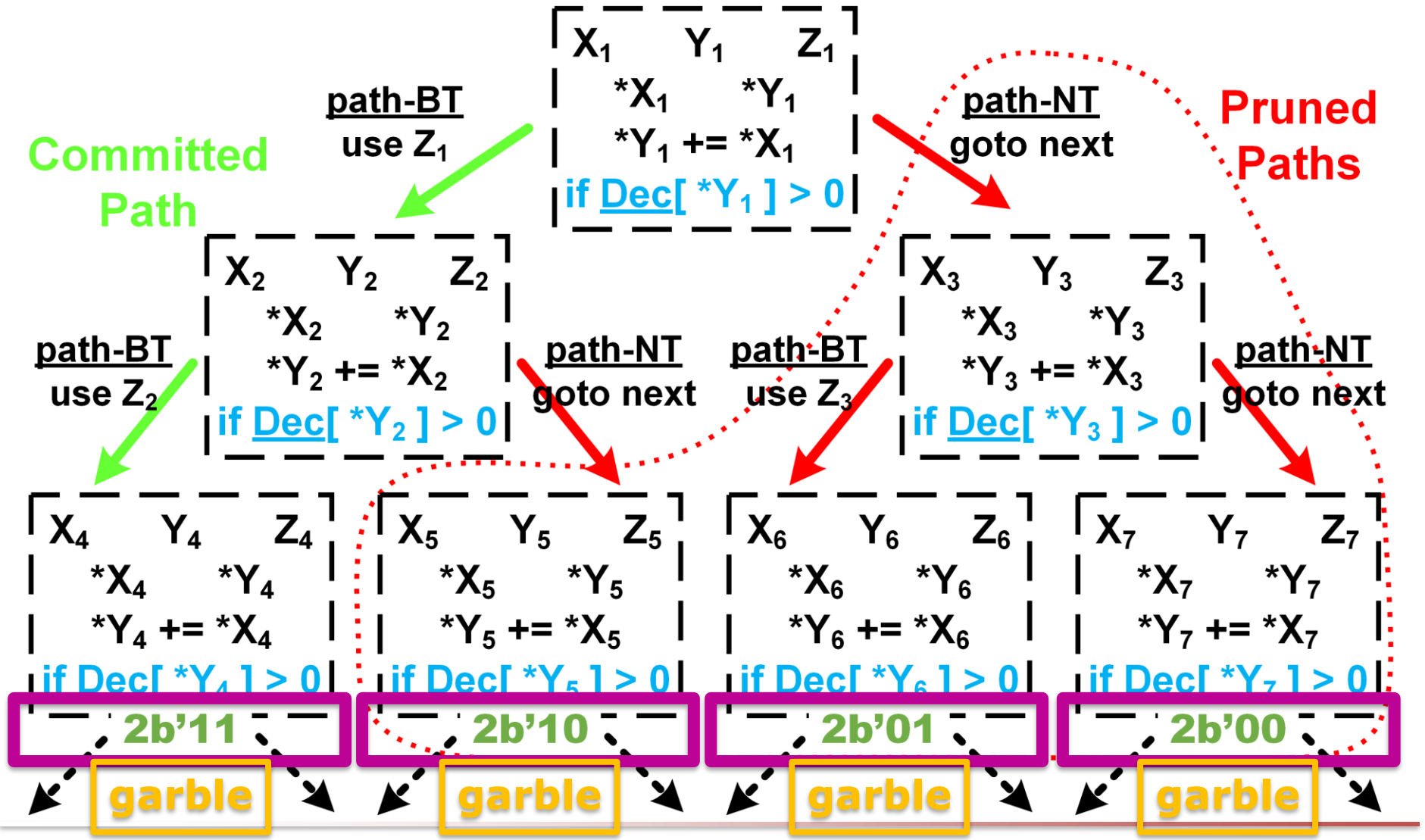
Path-BRANCH-TAKEN, Path-NOT-TAKEN

`addleq` control flow graph



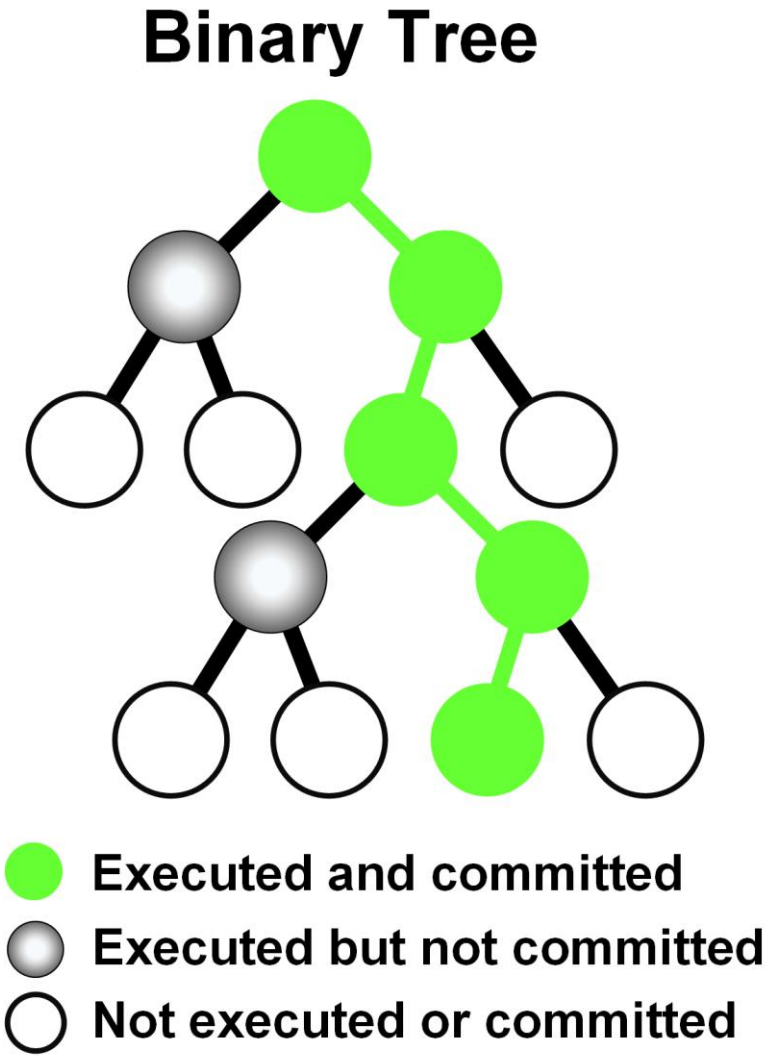
- Control Values $*Y_1$, $*Y_2$, $*Y_3$
- 2 levels are executed (3x overhead)
  - In general: $(2^{h+1} - 2)/h$ for h levels

`addleq` control flow graph



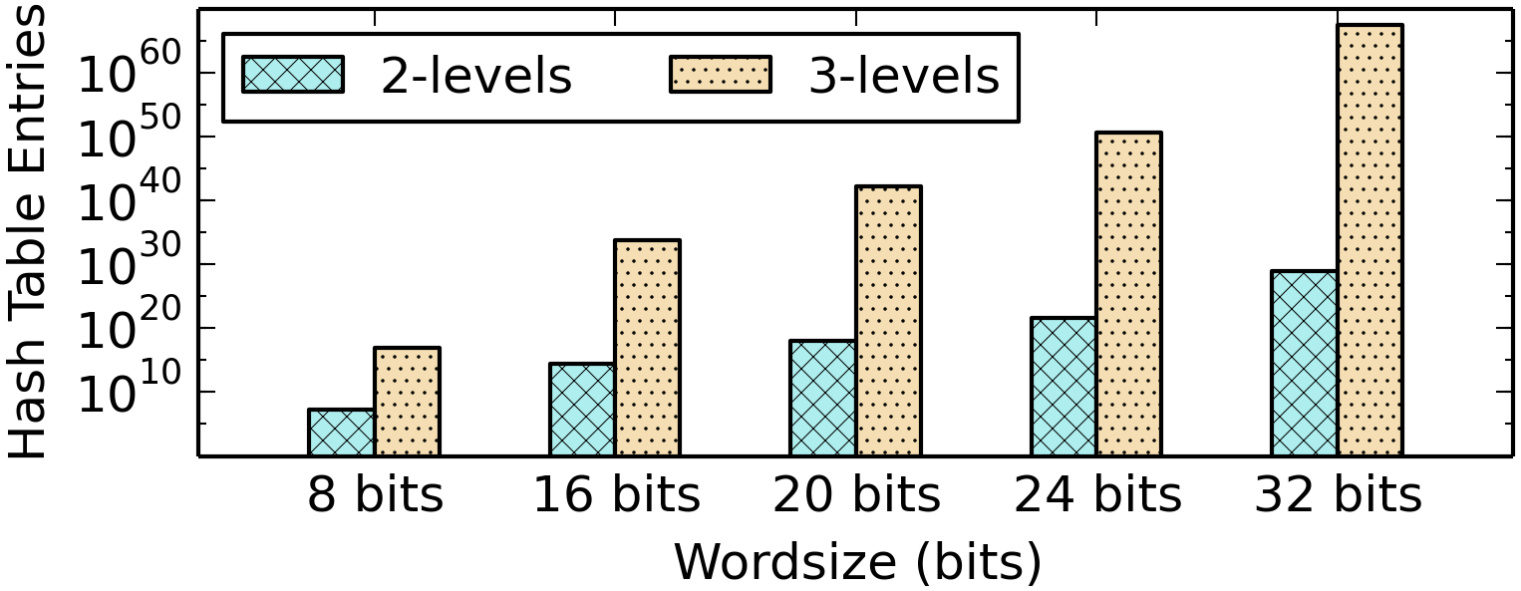
Digesting 2-levels of control values

- Use key sk to prevent probing the table
- Use the hash table to recover the correct 2-bit branch outcome
- Prune speculative paths
- Commit the program state in memory



Evaluation: hash table rows

- For 2 levels and a wordsize of 8 bits we need at most 16.78M hash table entries
  - Each row stores 2 bits for branch outcomes
- Scalability for larger wordsizes



Evaluation: obfuscation metrics

- We classify our obfuscation strategy using existing criteria [Collberg et al. 97]

Metric	MISR	SHA
Resilience	Full	1-way
Potency	Very High	Very High
Cost	Cheap	Costly

- Compared: 64-bit MISR and 256-bit SHA

Evaluation: aliasing considerations

- Control value digests should have adequate bitsize to avoid collisions
- Collision probabilities are negligible for 256-bit hash digests
- For MISR digests, we require at least 64 cells to minimize aliasing
  - More than one shift operations per input
  - Input longer than 64 bits are divided in 64-bit blocks and each block is applied twice

Conclusion

- Branching over encrypted values can cause information leakage
  - Simple LUTs do not prevent static analysis
- We mitigate this problem by obfuscating the correspondence between control values and branch decision outcomes
  - Non-deterministic evaluation of both paths
  - Lazily resolve the execution trace
  - Digest control values and index a hash table

Thank you