*ACM Asia Conference on Computer and Communications Security (ASIA CCS '25)*

# BranchGauge: Modeling and Quantifying Side-Channel Leakage in Randomization-Based Secure Branch Predictors

**Quancheng Wang**, Ming Tang, Ke Xu, Han Wang

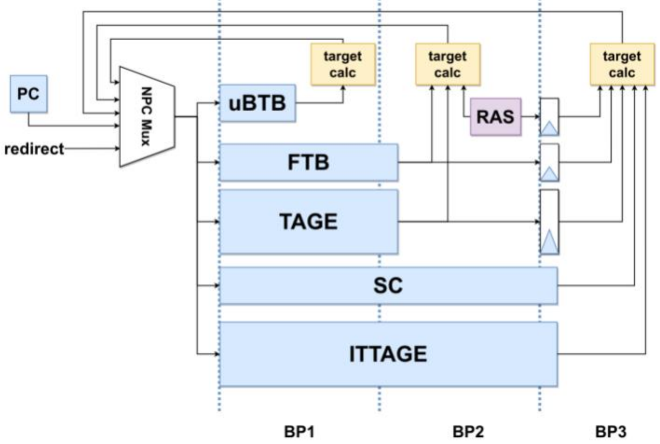
*School of Cyber Science and Engineering*

*Wuhan University*

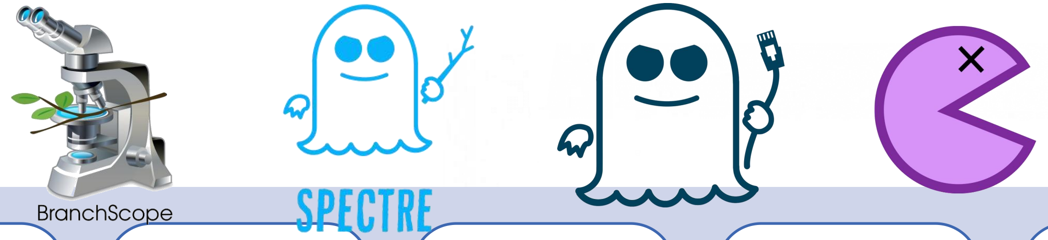
*August 27, 2025*

# Background: Timing and Speculative Attacks

- ➢ The design philosophy of modern CPU is faster speeds and greater efficiency
- ➢ Branch prediction unit (BPU) play a critical role in addressing control hazards
- ➢ However, inherent sharing characteristic introduces side-channel attack surfaces




Branch Predictor
Handle Control Hazards

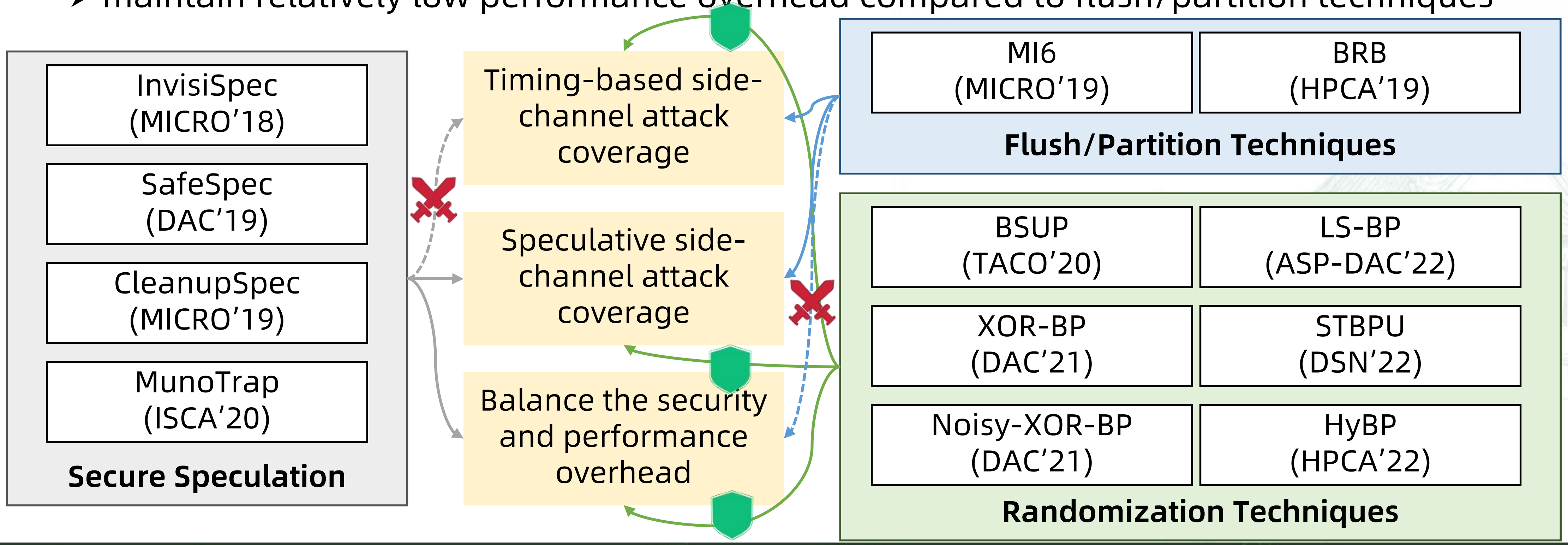



| 2018 | 2019 | 2020 | 2021 | 2022 | 2023--now |
|---|---|---|---|---|---|
| Spectre v1 | NetSpectre | Bluethunder | SPEAR | SpecHammer | GhostRace |
| Spectre v2 | SGXPectre | BlindSide | Speculative | PACMAN | InSpectre |
| BranchScope | SMotherSpectre | SpectreRewind | Interference | BHI | TikTag |
| | | | BranchSpectre | RetBleed | ITS |

intel AMD ARM RISC-V

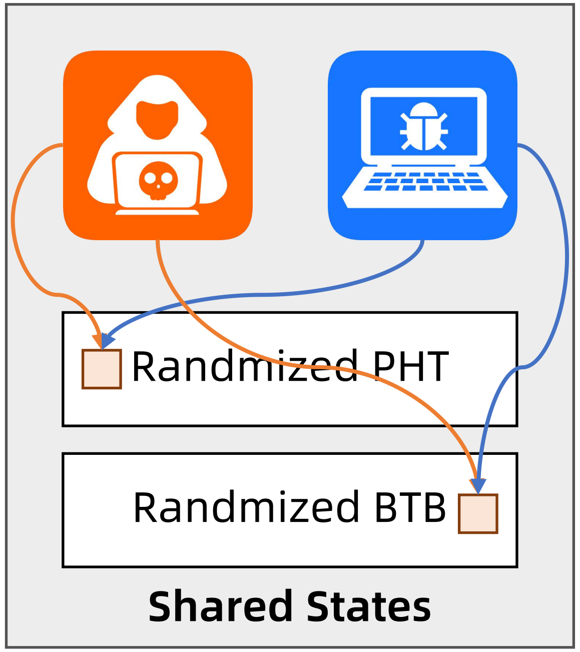
# Background: Secure Branch Prediction Designs

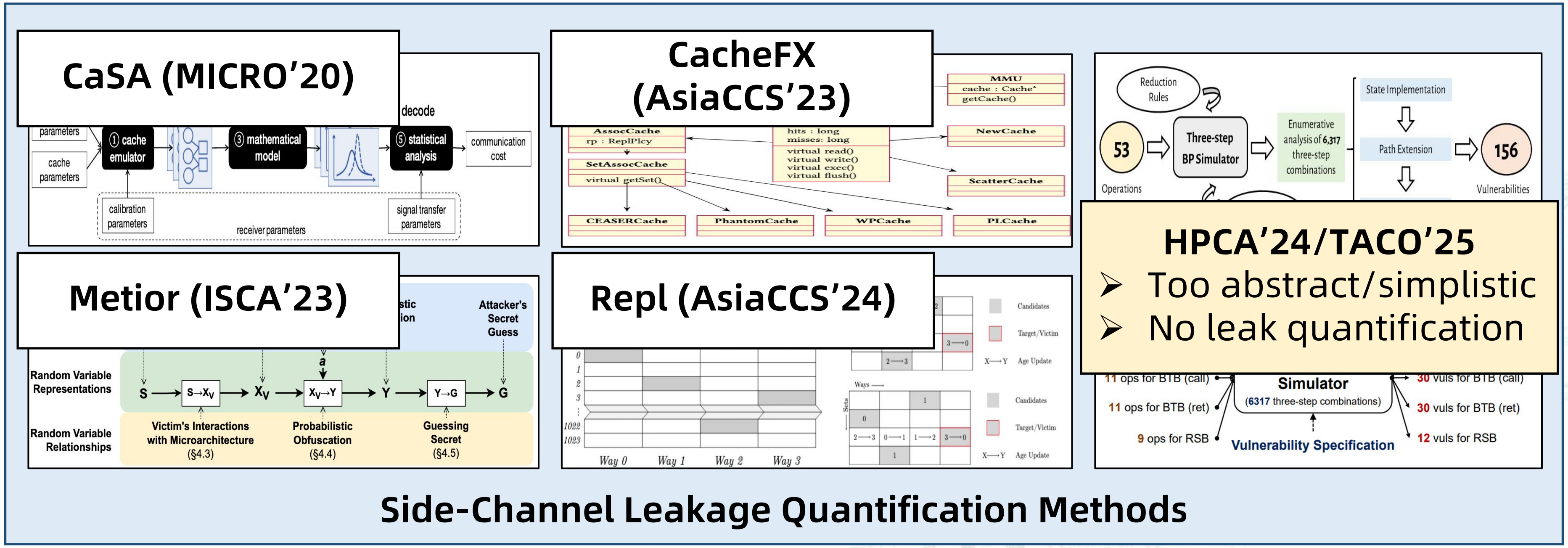
- **Researchers have proposed secure speculation schemes and branch predictors**
- **Randomization-based approaches stand out as more promising**
  - make attacks significantly more challenging and effectively reduce leakage risks
  - maintain relatively low performance overhead compared to flush/partition techniques



# Background: Side-Channel Evaluation Methods

- **However, these randomized approaches often fail to guarantee absolute security**
  - shared states between attacker and victim threads still exist
- **Existing evaluation methods cannot accurately quantify leakage in such designs**
  - CaSA, Metior, CacheFX, ...



# Challenge

➢ **How to build a leakage quantification framework for rand branch predictors?**

**Leakage Quantification Framework**

***Secure Design***

- Index Randomization
- Content Randomization
- Key Management
- Security Domain
- …

***Attack Type***

- PHT Timing Attack
- BTB Timing Attack
- PHT Speculative Attack
- BTB Speculative Attack
- …

***Timing Observation***

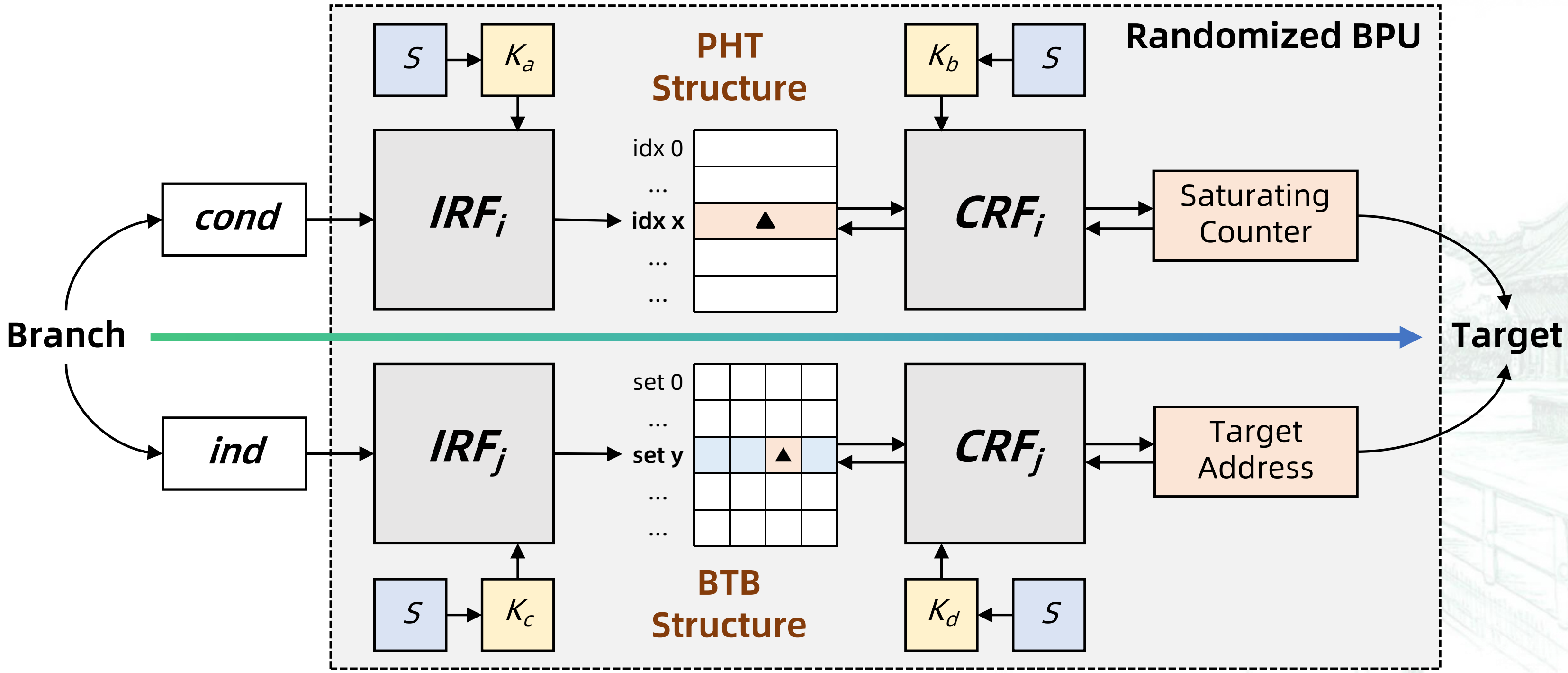
- Branch Hit
- Branch Miss
- Misprediction
- …

***Quantification Metric***

- Attacker’s Cost
- Attacker’s Success
- Victim’s Leakage
- …

# Modeling: Defining Components and Workflow

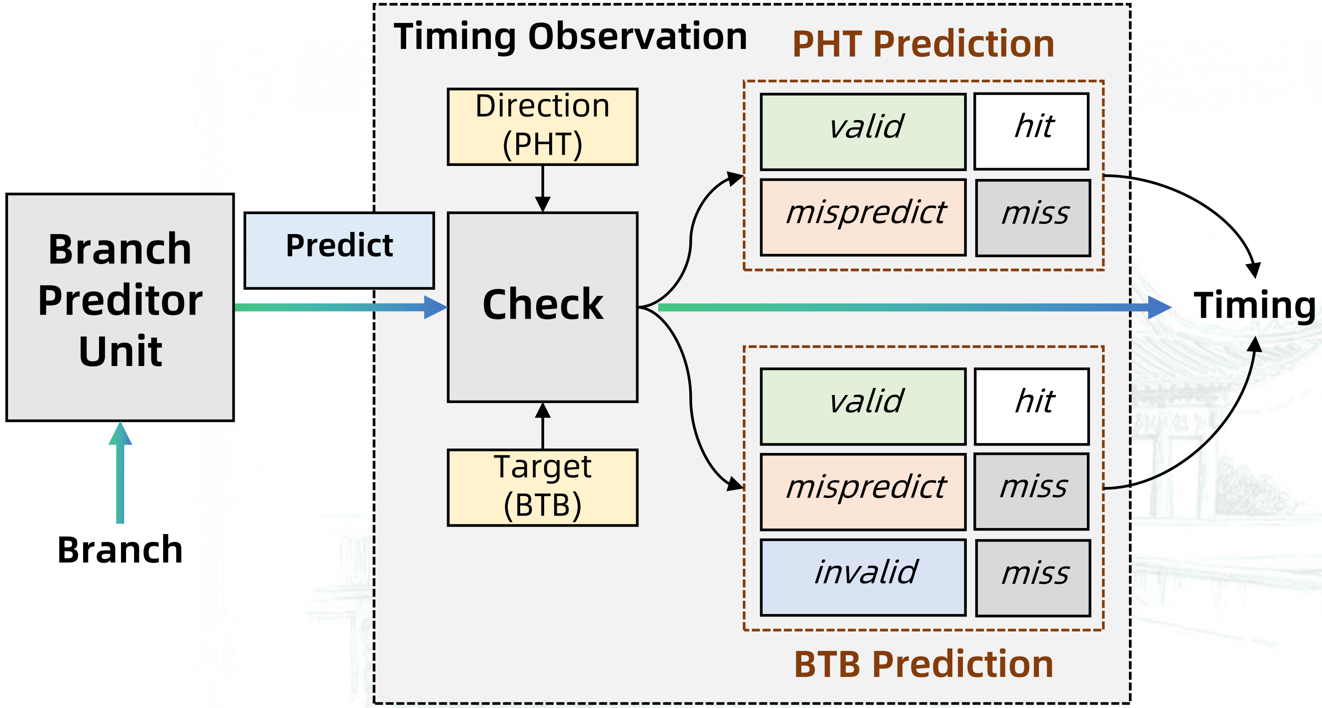
- Our model integrates index and content randomization functions: IRF and CRF
- IRF combines the branch instruction address with a randomization key K
- CRF randomizes PHT counter or BTB content using a randomization key K
- To further enhance security, distinct keys are assigned to different domains S



# Modeling: Constructing Timing Observations

➢ **We define a timing observation model to capture timing and speculative attacks**

| Components | Prediction State | Output Timing | Description |
|---|---|---|---|
| **PHT** | valid | hit | prediction matches the actual direction |
| | mispredict | miss | prediction differs the actual direction |
| **BTB** | valid | hit | prediction matches the actual target |
| | mispredict | miss | prediction differs the actual target |
| | invalid | miss | no matching is found in the target BTB set |

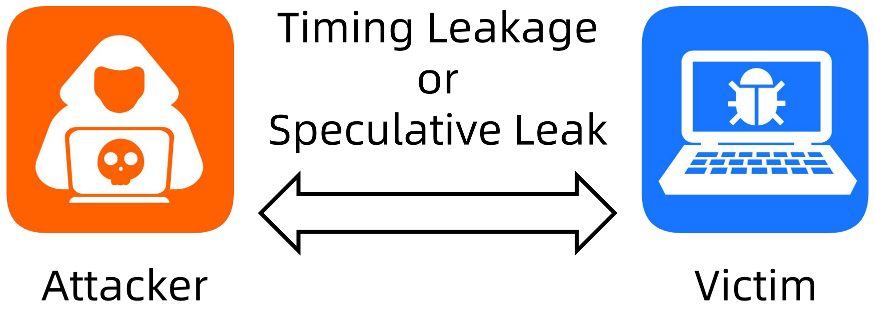


# Modeling: Implementing Existing Secure Designs

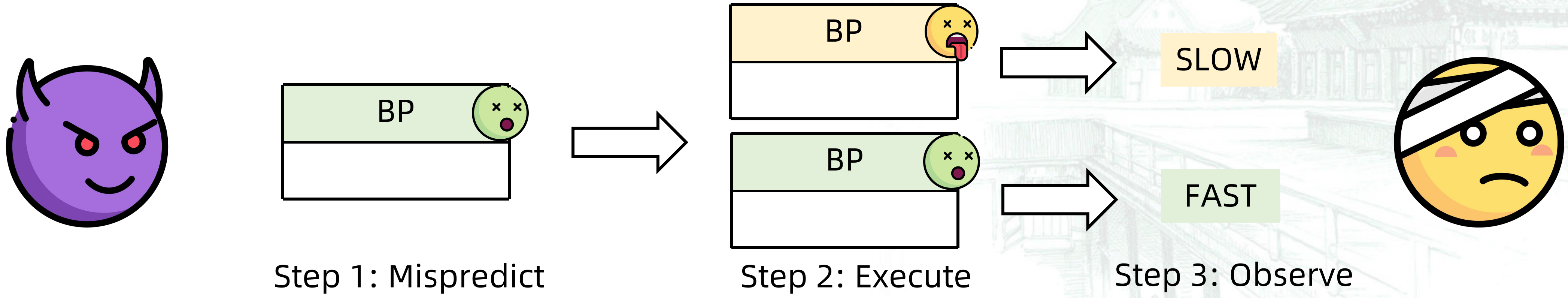
- We instantiate 6 randomization-based secure branch predictor designs
- We associate S and K with the security domain and the corresponding private keys
- Our implementation incorporates IRF for PHT/BTB, CRF for PHT/BTB src/BTB dest

| Branch Predictor | Reference | PHT | | BTB | | |
|---|---|---|---|---|---|---|
| | | IRF | CRF | IRF | CRF (Src) | CRF (Dest) |
| BSUP | TACO’20 | XOR/Key0 | LLBC/Key0 | XOR/Key0 | | LLBC/Key0 |
| XOR-BP | DAC’21 | | XOR/Key0 | | XOR/Key0 | XOR/Key0 |
| Noisy-XOR-BP | DAC’21 | XOR/Key0 | XOR/Key1 | XOR/Key0 | XOR/Key1 | XOR/Key1 |
| LS-BP | ASP-DAC’22 | XOR/PID+Key0 | | XOR/PID+Key0 | | |
| STBPU | DSN’22 | Hash/Key0 | XOR/Key1 | Hash/Key0 | Hash/Key0 | XOR/Key1 |
| HyBP | HPCA’22 | QARMA/Key0 | XOR/Key1 | QARMA/Key0 | QARMA/Key0 | XOR/Key1 |

# Fomulating: Reuse-Based PHT/BTB Attacks

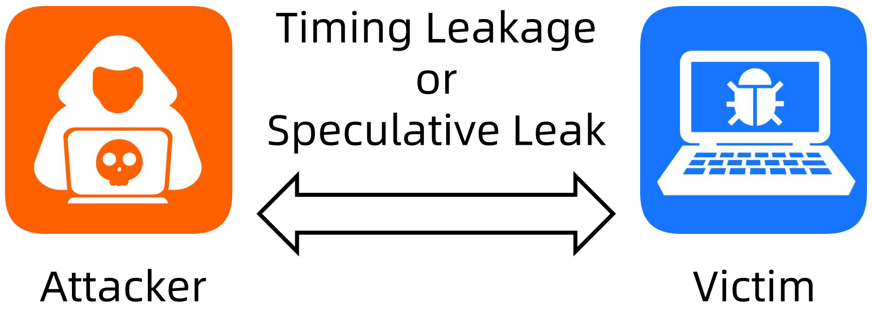


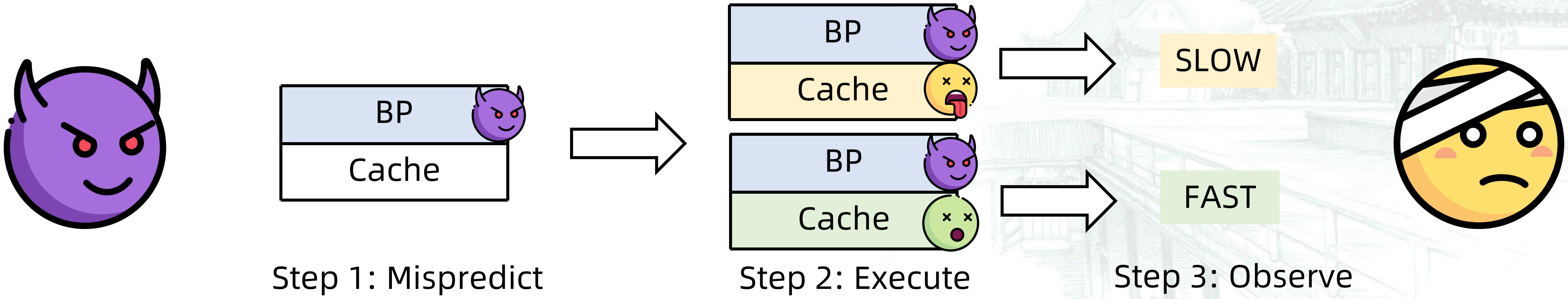

- **Attacker and victim**
  - Attacker: App, OS, VM, etc.
  - Victim: App, OS, VM, TEE, etc.
- **Attacker's strategy**
  - Find the proper branch instruction to mispredict specific PHT/BTB entry
  - Probe timing differences of the target branch instruction
- **High-level of reuse-based attacks (timing leakage)**



# Fomulating: Reuse-Based PHT/BTB Attacks

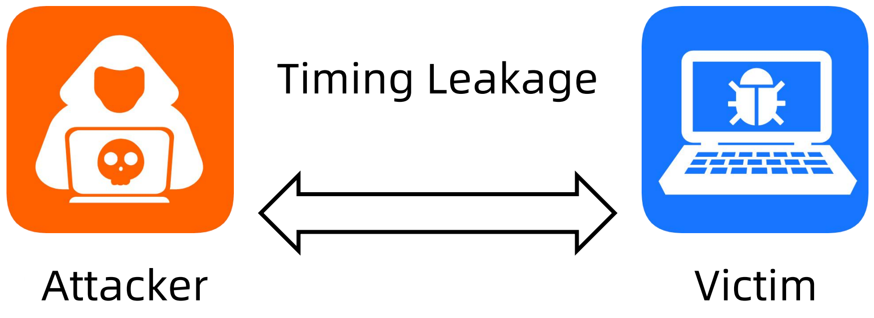
- **Attacker and victim**
  - Attacker: App, OS, VM, etc.
  - Victim: App, OS, VM, TEE, etc.
- **Attacker's strategy**
  - Find the proper branch instruction to mispredict specific PHT/BTB entry
  - Probe timing differences of covert channels due to speculative execution
- **High-level of reuse-based attacks (speculative leakage)**

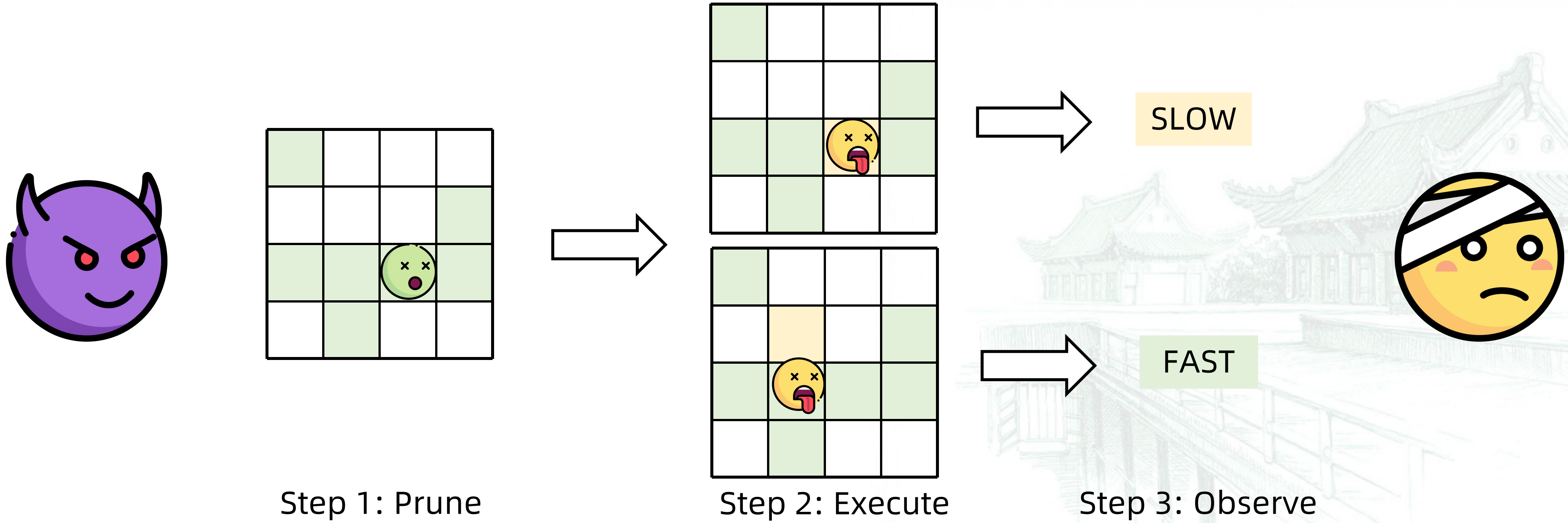


# Fomulating: Prune-Based BTB Attacks

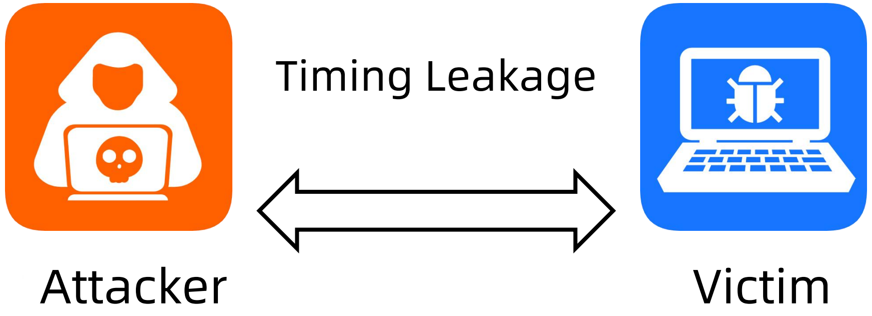
- **Attacker's strategy**
  - Construct eviction set for specific BTB set
  - Probe timing differences of branch instructions
- **High-level of prune-based attacks**

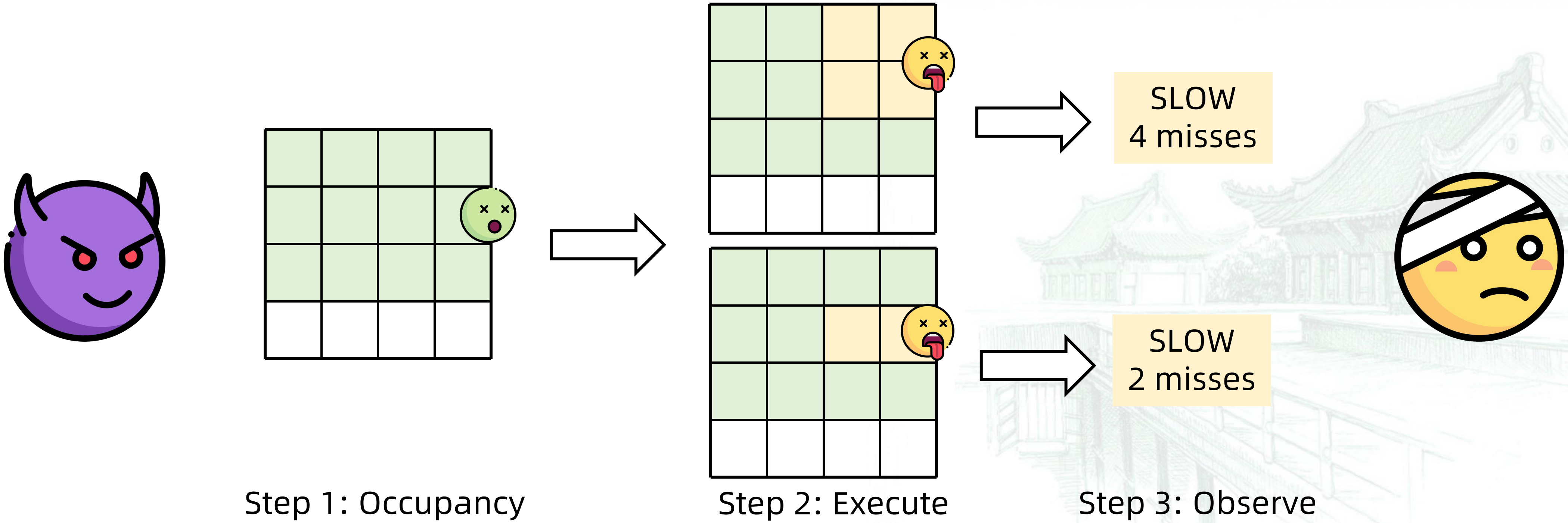


# Fomulating: Occupancy-Based PHT/BTB Attacks

- **Attacker's strategy**
  - Filling the PHT/BTB branch predictor
  - Probe timing differences of branch instructions
- **High-level of occupancy-based attacks**



# Evaluation: Leakage Quantification Metrics

- **Branch Accesses N:** The total number of branch accesses required to achieve the attack
  - Encompassing accesses in both the attacker's space and the victim's space

Branch Accesses N Calculation Formula

$$\mathbf{N}[\mathbb{G}, \mathbb{V}] = \sum_{i=1}^{|\mathbb{G}|} \mathbf{N}[\mathbb{G}_i] + \sum_{j=1}^{|\mathbb{V}|} \mathbf{N}[\mathbb{V}_j]$$

- **Collision Probability Pr:** The probability of a collision between the attacker and victim
  - The leakage of sensitive information across repeated trials

Collision Probability Pr Calculation Formula

$$\mathbf{Pr}[s|\mathbb{G}, \mathbb{V}] = \frac{1}{|R|} \sum_{i=1}^{|R|} (s = [\mathbb{V} \to \mathbb{G}])$$

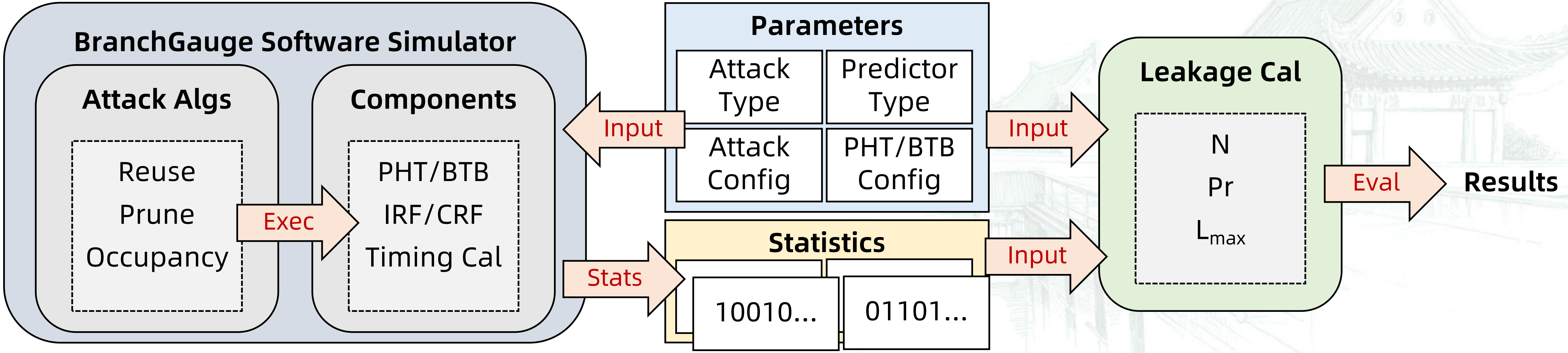
- **Maximal Leakage $L_{max}$:** The maximum amount of information that can be leaked
  - A relative measure of leakage by comparing the actual leakage to random guessing

Maximal Leakage $L_{max}$ Calculation Formula

$$\mathbf{L_{max}}[\mathbb{V} \to \mathbb{G}] = \log_2 \left( \sum_{s \in \mathbb{S}} \max_{\mathbb{G}} [\mathbf{Pr}[s|\mathbb{G}, \mathbb{V}]] \right)$$

# Evaluation: Leakage Quantification Methodology

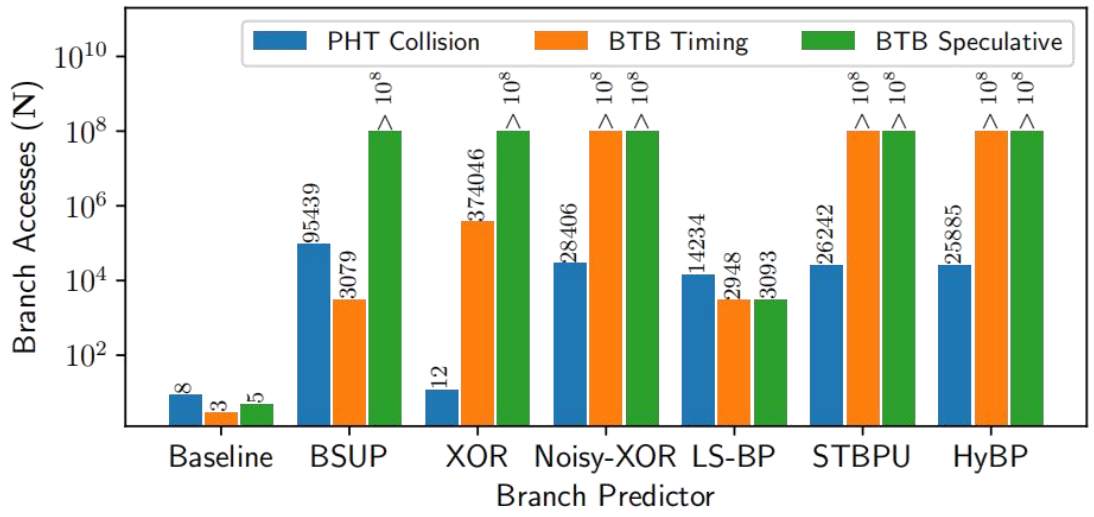
- **We develop a software simulator with the following implementation strategy**
  - **Component structure:** define vector variables, lookup functions and update functions
  - **Timing observation:** set return values of lookup functions, hit (1) or miss (0)
  - **IRF and CRF functions:** implement get tag/counter for PHT, get set/tag/dest for BTB
  - **Attack algorithms:** collect statistics on branch accesses and timing observations
  - **Leakage calculation:** use the collected data and the previously defined formulas

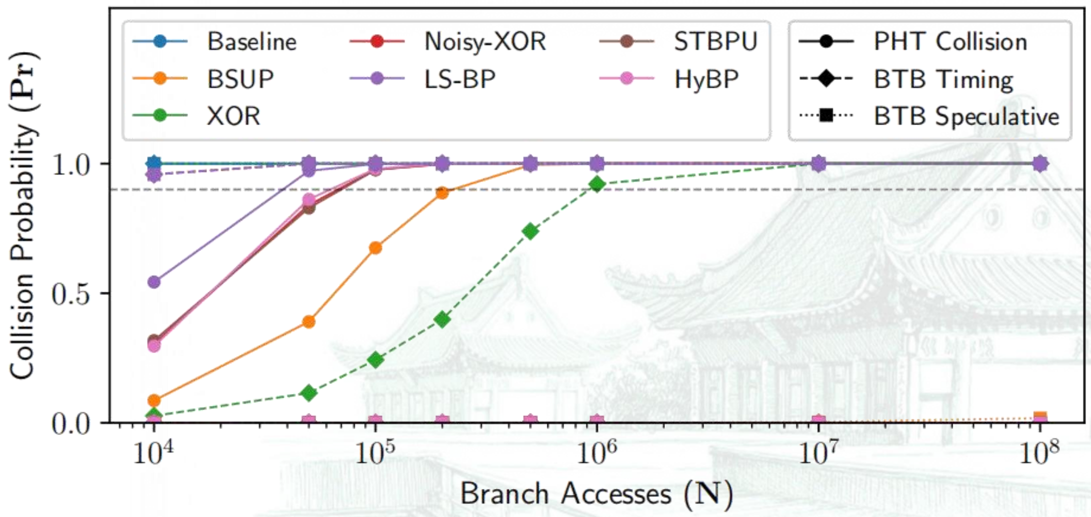


# Evaluation: Reuse-Based Attacks

武漢大學
WUHAN UNIVERSITY

- IRF increases the complexity but the attacker can still generate branch conflicts
- The CRF security relies on bit width, making encrypted BTB more secure than PHT
- PHT reuse attacks (both timing and speculative) remain a major challenge



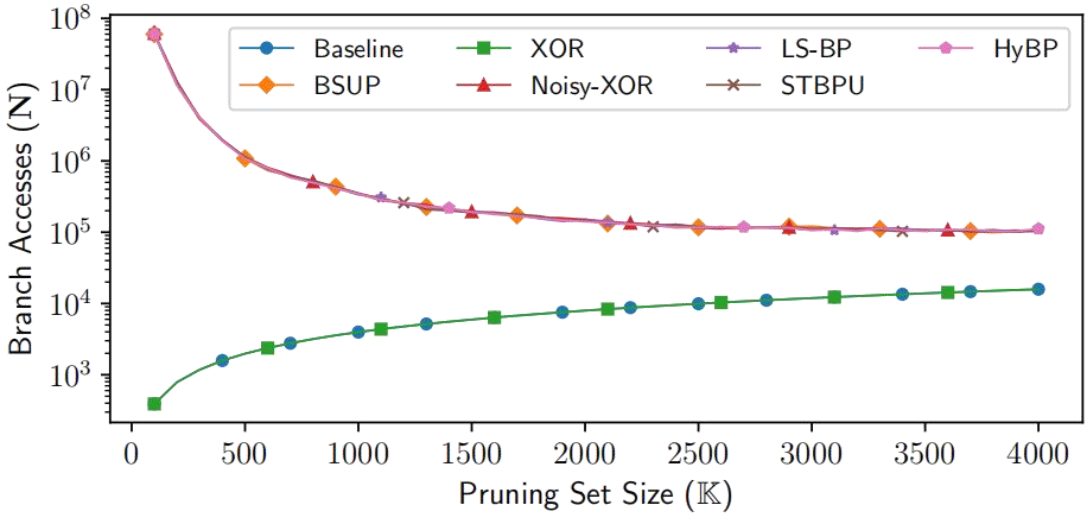




Branch Accesses N
Reuse-Based

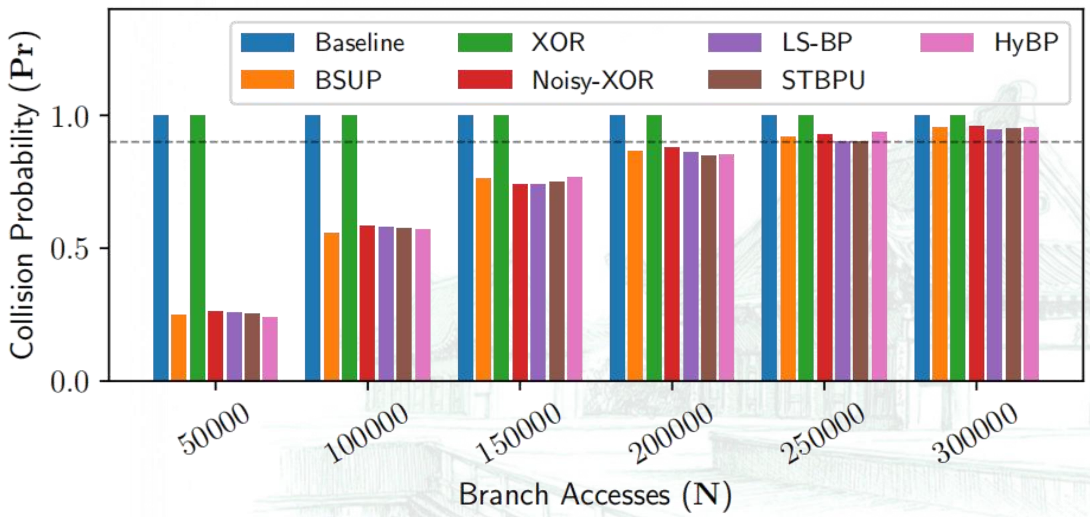
Collision Probability Pr
Reuse-Based

# Evaluation: Prune-Based Attacks

- The goal of IRF randomization is making eviction set construction difficult
- Pruning-based strategies allow attackers to construct eviction sets
- With a reasonable number of branch accesses for all existing randomized BTBs



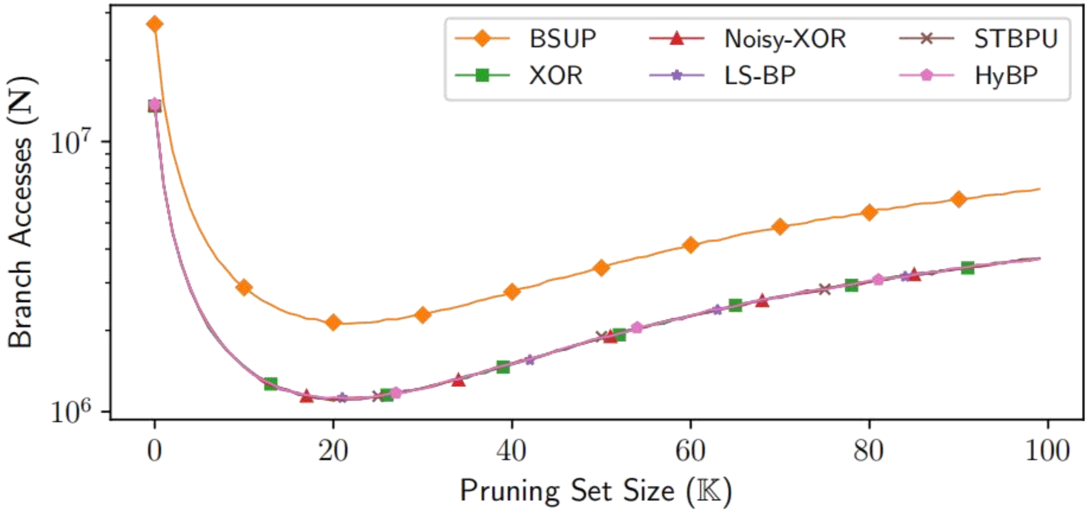

Branch Accesses N
Prune-Based



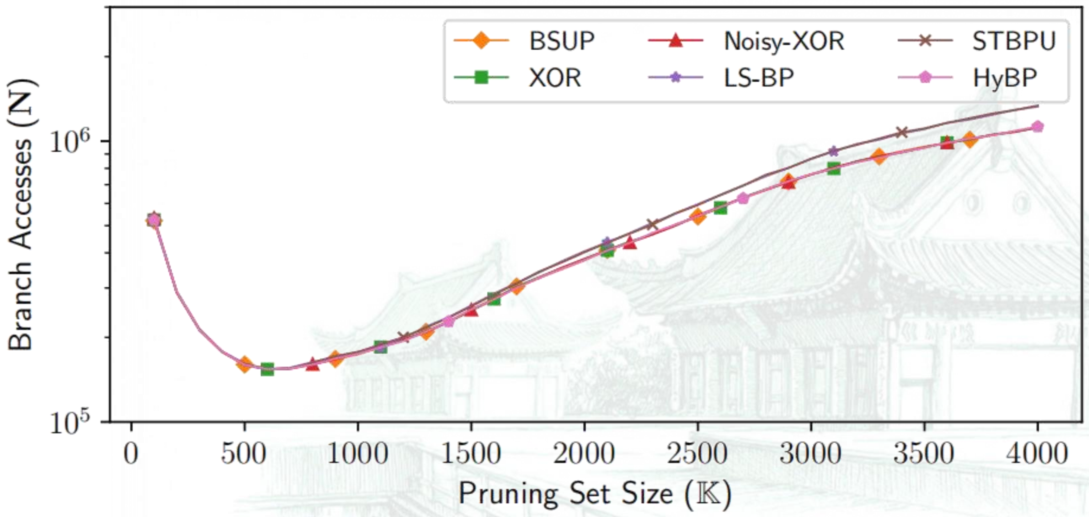

Collision Probability Pr
Prune-Based

# Evaluation: Occupancy-Based Attacks

- The effectiveness of randomization appears to vanish when relaxing constraints
- Despite different designs employing various random mapping or encryption
- The attacker can easily construct occupancy sets that avoid self-conflicts




Branch Accesses N
PHT Occupancy




Branch Accesses N
BTB Occupancy

# Evaluation: Maximal Leakage

- **Multi-bit leakage: covert channel attacks based on occupancy strategies**
- **PHT maximum leakage reaches 1.81/1.56 for 2-bit/3-bit saturating counter (4 iters)**
- **BTB maximum leakage reaches 0.74 (1 iter), 1.22 (2 iters) and 1.87 (4 iters)**
- **All existing designs remain vulnerable to PHT/BTB occupancy attack vectors**

| Design | 1 Iteration | | | 2 Iterations | | | 4 Iterations | | |
|---|---|---|---|---|---|---|---|---|---|
| | $10^4$ | $10^5$ | $5\times10^5$ | $10^4$ | $10^5$ | $5\times10^5$ | $10^4$ | $10^5$ | $5\times10^5$ |
| BSUP | 0.01 | 0.01 | 0.39 | 0.08 | 0.14 | 0.89 | 0.29 | 0.64 | 1.56 |
| XOR | 0.00 | 0.00 | 0.66 | 0.10 | 0.26 | 1.15 | 0.39 | 0.90 | 1.81 |
| Noisy-XOR | 0.00 | 0.00 | 0.66 | 0.11 | 0.27 | 1.15 | 0.38 | 0.90 | 1.81 |
| LS-BP | 0.01 | 0.01 | 0.66 | 0.10 | 0.27 | 1.15 | 0.37 | 0.91 | 1.81 |
| STBPU | 0.00 | 0.00 | 0.65 | 0.11 | 0.25 | 1.14 | 0.38 | 0.89 | 1.80 |
| HyBP | 0.00 | 0.00 | 0.66 | 0.12 | 0.25 | 1.15 | 0.39 | 0.88 | 1.81 |

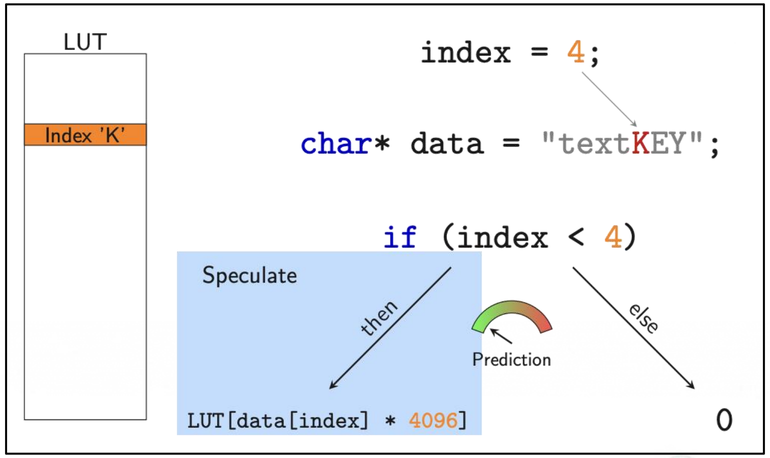
Maximal Leakage $L_{max}$
PHT Occupancy

| Design | 1 Iteration | | | 2 Iterations | | | 4 Iterations | | |
|---|---|---|---|---|---|---|---|---|---|
| | $10^4$ | $10^5$ | $2\times10^5$ | $10^4$ | $10^5$ | $2\times10^5$ | $10^4$ | $10^5$ | $2\times10^5$ |
| BSUP | 0.00 | 0.71 | 0.74 | 0.09 | 1.19 | 1.22 | 0.37 | 1.85 | 1.87 |
| XOR | 0.01 | 0.71 | 0.74 | 0.11 | 1.19 | 1.22 | 0.39 | 1.85 | 1.87 |
| Noisy-XOR | 0.01 | 0.70 | 0.74 | 0.10 | 1.19 | 1.22 | 0.38 | 1.84 | 1.87 |
| LS-BP | 0.01 | 0.71 | 0.74 | 0.11 | 1.19 | 1.22 | 0.39 | 1.85 | 1.87 |
| STBPU | 0.00 | 0.70 | 0.74 | 0.09 | 1.19 | 1.22 | 0.38 | 1.84 | 1.87 |
| HyBP | 0.01 | 0.70 | 0.74 | 0.10 | 1.19 | 1.22 | 0.39 | 1.84 | 1.87 |

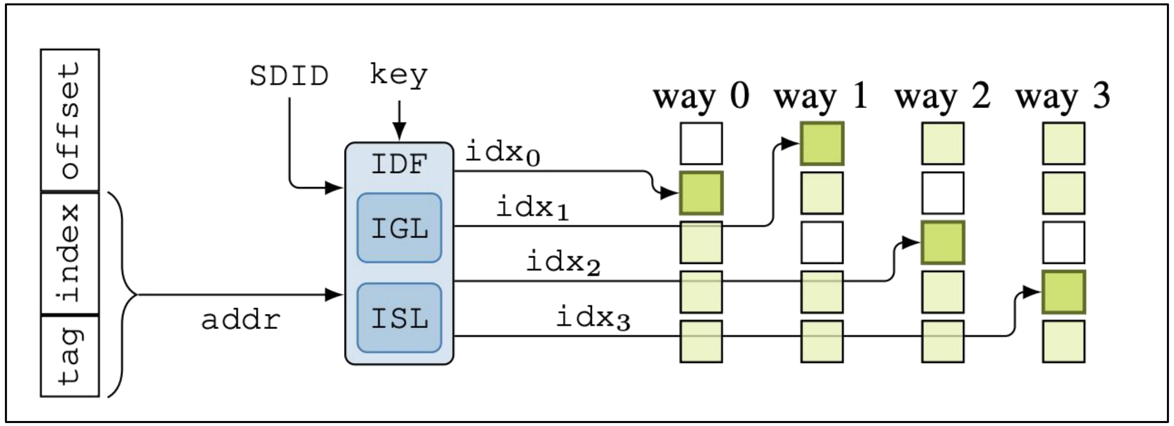
Maximal Leakage $L_{max}$
BTB Occupancy

# Discussion: Future Directions

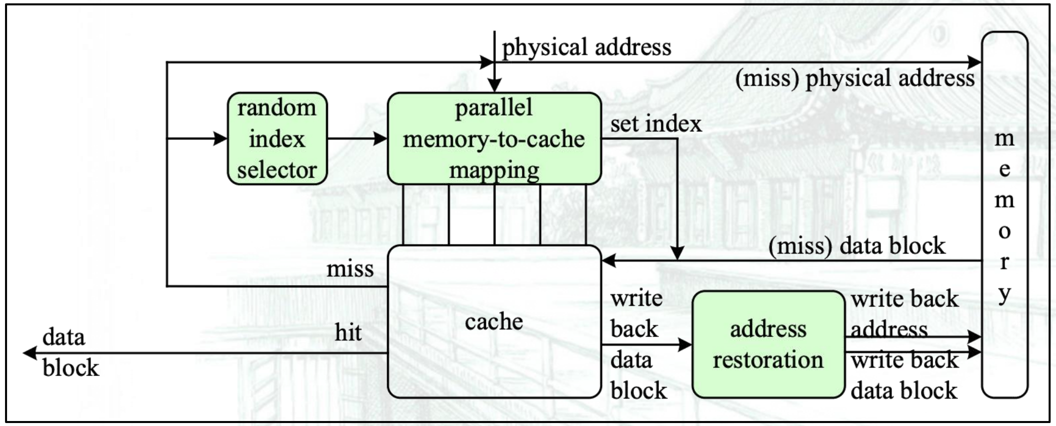
- **Developing more effective countermeasures**
  - Hybrid designs: SassCache (S&P'23)
  - Non-deterministic designs: PhantomCache (NDSS'20)
- **Taking more attention on PHT security issues**
  - Remain the weakest link in current secure designs
  - Particularly vulnerable to Spectre attacks




PHT Security Issues




SassCache Architecture
Giner et al. S&P'23




PhantomCache Architecture
Tan et al. NDSS'20

# Conclusion

- **Modeling: Components and Workflow of Randomized Secure Branch Predictors**
  - Develop a PHT and BTB branch predictor model focused on side-channel security
  - Integrate indexing randomization and content randomization mechanisms
  - Incorporate timing observation mechanisms of timing and speculative attacks
- **Fomulating: Reuse-Based, Prune-Based and Occupancy-Based Attack Strategies**
  - Describe microarchitectural attacks targeting PHT entries, BTB entries, and BTB sets
  - Define reuse-based, prune-based, and occupancy-based attack strategies
  - Effectively evaluate the side-channel security properties within our framework
- **Evaluation: Leakage Quantification Metrics and Empirical Side-Channel Analysis**
  - Define branch access number, collision probability and maximal leakage metrics
  - Demonstrate the effectiveness of our framework in quantifying side-channel leakage
  - Underscore the necessity for stronger countermeasures against side-channel attacks

# Q&A