Citation: I. Nasurulla, K. Hemalatha, P. Ramachandran, et al. Consensus-based cryptographic framework for side-channel attack resilience in cloud environments. *Journal of Harbin Institute of Technology (New Series)*, 2025, 32(2): 90-104. DOI: 10.11916/j.issn.1005-9113.2023130

Consensus-Based Cryptographic Framework for Side-Channel Attack Resilience in Cloud Environments

I. Nasurulla[1], *K.Hemalatha*[2], *P.Ramachandran*[3*] *and S.Parvathi*[4]

(*1.Department of MCA, VEMU Institute of Technology, Chittoor District 517112, Andhra Pradesh, India;*
2.Department of Computer Science and Engineering (AI), Madanapalle Institute of Technology & Science, Madanapalle 517325, India;
3.Department of MCA, Parul institute of Engineering and Technology, Vadodara 391760, India;
4.Department of Computer Science and Engineering, Erode Sengunthar Engineering College, Perundurai 638057, India)

Abstract: Cloud environments are essential for modern computing, but are increasingly vulnerable to Side-Channel Attacks (SCAs), which exploit indirect information to compromise sensitive data. To address this critical challenge, we propose SecureCons Framework (SCF), a novel consensus-based cryptographic framework designed to enhance resilience against SCAs in cloud environments. SCF integrates a dual-layer approach combining lightweight cryptographic algorithms with a blockchain-inspired consensus mechanism to secure data exchanges and thwart potential side-channel exploits. The framework includes adaptive anomaly detection models, cryptographic obfuscation techniques, and real-time monitoring to identify and mitigate vulnerabilities proactively. Experimental evaluations demonstrate the framework's robustness, achieving over 95% resilience against advanced SCAs with minimal computational overhead. SCF provides a scalable, secure, and efficient solution, setting a new benchmark for side-channel attack mitigation in cloud ecosystems.

Keywords: Cloud computing; side channel attacks; HAVAL cryptographic hash; Wilcoxon signed-rank test consensus mechanism; improved schmidt-samoa cryptography

CLC number: TP309 **Document code**: A **Article ID**: 1005-9113(2025)02-0090-15

0 Introduction

The rapid proliferation of cloud computing has revolutionized data storage, processing, and service delivery by offering unparalleled scalability, flexibility, and cost-effectiveness. However, as the backbone of modern digital infrastructure, cloud environments are increasingly targeted by sophisticated cyber threats, particularly Side-Channel Attacks (SCAs). SCAs are an emerging class of threats that compromise confidential information by analyzing non-functional computational aspects of program execution, such as elapsed time, memory allocation, or network packet size[1-8]. By exploiting these indirect data leakages, adversaries can infer sensitive information without directly breaching cryptographic defenses, posing a unique and insidious challenge to secure cloud systems. Unlike conventional attacks that target software vulnerabilities or misconfigurations, SCAs operate through stealthy and non-intrusive means, making them difficult to detect and mitigate effectively. The prevalence of shared and multi-tenant architectures in cloud environments, where multiple users share computational resources, exacerbates this vulnerability. These architectures create opportunities for malicious actors to execute SCAs from co-located virtual machines, further elevating the risk to critical and sensitive data[9-13].

Traditional cryptographic solutions, while robust in safeguarding data confidentiality and integrity, often fail to address the nuanced challenges posed by SCAs in dynamic cloud ecosystems. Additionally, existing security frameworks tend to rely on computationally intensive methods that may impede performance, particularly in applications requiring low latency and high throughput[14-18]. This growing gap highlights the urgent need for innovative, lightweight,

Received 2023-12-05. Published online 2025-02-19.
* Corresponding author: P. Ramachandran, Ph.D, Assistant Professor. Email: ramchandran.p34251@paruluniversity.ac.in.

and adaptive security mechanisms capable of mitigating SCAs without imposing significant computational or operational burdens on cloud systems[19-28]. To address these challenges, this paper introduces the SecureCons Framework (SCF), a consensus-based cryptographic framework specifically designed to enhance resilience against SCAs in cloud environments.

SCF integrates lightweight cryptographic obfuscation techniques with a blockchain-inspired consensus mechanism to secure data exchanges and neutralize side-channel vulnerabilities. The consensus mechanism validates data integrity through decentralized protocols, ensuring that malicious actors cannot compromise or alter data even in the presence of co-located adversaries. Furthermore, SCF incorporates an adaptive anomaly detection model, leveraging statistical analysis and machine learning to identify and mitigate side-channel exploits in real-time. This adaptive layer dynamically adjusts security parameters based on detected threats, providing proactive and robust protection without incurring significant computational overhead. SCF's architecture is designed to operate seamlessly across various cloud configurations, including Infrastructure-as-a-Service (IaaS) and Platform-as-a-Service (PaaS) models, making it versatile and scalable for diverse cloud use cases.

One of the key strengths of SCF lies in its ability to address the limitations of traditional cryptographic solutions by adopting a lightweight approach tailored to resource-constrained environments. The framework minimizes computational complexity while maintaining high levels of security, ensuring compatibility with applications requiring high throughput and low latency. Additionally, its blockchain-inspired consensus mechanisms enhance transparency and auditability, providing robust security assurances in multi-tenant cloud settings. By integrating non-functional computation aspects such as time analysis, memory utilization patterns, and network packet size monitoring, SCF expands its scope to detect and counter SCAs more effectively. This holistic approach positions SCF as a comprehensive solution that combines cryptographic obfuscation, decentralized validation, and real-time anomaly detection into a unified framework.

Through experimental evaluation, the proposed SCF framework demonstrates its effectiveness in mitigating SCAs, achieving an average resilience rate exceeding 95% while maintaining minimal performance overhead. The results validate the practicality of SCF in securing cloud environments against emerging threats, setting a benchmark for future research and innovation in the domain. This paper also explores the broader implications of consensus-based cryptographic frameworks, emphasizing their potential to enhance trust, resilience, and security in distributed cloud systems. As cyber threats continue to evolve, innovative frameworks like SCF are poised to play a pivotal role in safeguarding critical infrastructure and maintaining the confidentiality, integrity, and availability of sensitive data in cloud ecosystems. Ultimately, SCF not only addresses the current security challenges posed by SCAs, but also lays the groundwork for future advancements in cryptographic resilience and adaptive security solutions for the cloud.

1 Related Works

The growing prevalence of SCAs in cloud computing environments has highlighted critical vulnerabilities in current cryptographic frameworks and system architectures. These threats exploit non-functional computational characteristics such as cache timing, power consumption, or memory access patterns, exposing sensitive data without directly breaching cryptographic protocols. As cloud infrastructures become more integral to modern computing, the need for effective mitigation strategies has led to an increase in research focused on enhancing security mechanisms against SCAs. SCAs analyze the various electronic information leaked by physical devices, such as power consumption, electromagnetic emissions, and timing behaviors, to bypass the cracking of encryption algorithms. Collecting these leaked electronic signals, however, remains a challenge due to the complex nature of extracting meaningful information without directly compromising device functionality. The overarching solution lies in ensuring that any leaked electronic information does not correspond to internal device processing, thus safeguarding against these vulnerabilities.

Recent advances in Trusted Execution Environments (TEEs) have shown promise in mitigating SCAs by isolating sensitive computations. Wilke[1] explored the

potential of AMD-SEV to provide single-stepping mechanisms that enhance execution transparency. Similarly, Suzaki[3] assessed the performance of TEEs such as Intel SGX and Arm TrustZone, identifying key trade-offs between security and system performance. These studies underscore the utility of TEEs while highlighting their limitations when faced with sophisticated attacks under specific conditions.

Cache-based side-channel attacks have received significant attention, with researchers proposing fully associative designs to mitigate conflicts. Saileshwar and Qureshi[2] introduced the MIRAGE framework, which addresses conflict-based cache attacks by eliminating predictable cache behaviors. Futhermore, Le et al.[5] demonstrated a real-time detection system for SCAs on RISC-V architectures, employing hardware performance monitoring and out-of-order execution tracking. These approaches have showcased the potential for proactive SCA mitigation but often necessitate architectural changes, limiting their broader applicability.

Hardware security research, particularly on RISC-V architectures, has also addressed microarchitectural vulnerabilities. Kou et al.[6] introduced the load-step framework, which offers precise execution control for studying novel side-channel threats, including flush+evict techniques. Meanwhile, Gerlach et al.[7] explored microarchitectural attacks on RISC-V CPUs, highlighting their susceptibility to advanced SCAs. Despite these advances, the dependency on hardware redesign remains a barrier to adoption.

Detection systems leveraging advanced algorithms and performance monitoring tools have further enriched the field. Li and Gaudiot[4] utilized hardware performance counters to detect speculative execution attacks like spectre. By analyzing execution patterns in real time, these systems effectively identify anomalies but depend heavily on hardware modifications, limiting their practical deployment in diverse environments.

Cloud security has also benefited from advancements in Intrusion Detection Systems (IDS). Mahdavi et al.[8] proposed an incremental transfer learning approach for IDS, enhancing adaptability to dynamic threat landscapes. Similarly, Guarascio et al.[10] explored collaborative frameworks for cyber-threat intelligence, which improve detection efficiency through distributed systems. Additionally, Fu et al.[9] introduced a privacy-aware auditing scheme for cloud data sharing that maintains data integrity and confidentiality. However, these solutions often struggle to balance computational efficiency with robust security.

Honeypot systems have been used effectively to attract attackers and analyze their methods. Matta et al.[11] demonstrated their utility in mitigating threats but noted challenges related to scalability and resource management in large cloud environments. Lastly, Tang et al.[12] provided a comprehensive survey on the enabling technologies and challenges of the internet of intelligence, emphasizing the need for adaptive solutions to address the increasing complexity of SCAs in cloud systems.

Despite these advancements, several drawbacks persist. TEE implementations, as discussed by Wilke[1] and Suzaki[3], are often constrained by performance bottlenecks and remain vulnerable to advanced SCAs. Cache mitigation techniques, such as those proposed by Saileshwar and Qureshi[2] and Le et al.[5], require significant architectural modifications, reducing their feasibility for widespread adoption. Intrusion detection and privacy-preserving schemes[8-9] are often computationally expensive and face scalability challenges in large-scale environments. Additionally, honeypots[11] and collaborative frameworks[10] may not always provide comprehensive coverage, leaving critical vulnerabilities unaddressed.

These challenges underscore the need for innovative frameworks that integrate lightweight cryptographic mechanisms with adaptive detection capabilities to address the evolving landscape of SCAs in cloud environments. The SecureCons Framework proposed in this study aims to bridge these gaps by providing a holistic, efficient, and scalable solution to side-channel vulnerabilities in cloud systems.

2 Methodology

The proposed SecureCons Framework integrates advanced cryptographic techniques and machine learning to provide a robust defense against SCAs in cloud environments. This methodology employs the SCAAML dataset, HAVAL (hashing variable and length) cryptographic hash block generation, a Wilcoxon signed-rank test consensus mechanism, and an Enhanced Schmidt-Samoa Jaccard Extreme Learning Machine (ES-SJELM) for block validation. Each component is designed to optimize security,

scalability, and performance.

The methodology leverages the SCAAML dataset, a benchmark dataset for evaluating side-channel attack resilience. This dataset contains features derived from non-functional aspects such as cache timing, power analysis, and memory access patterns, which are commonly exploited by SCAs. The data is preprocessed using standard normalization techniques and divided into training and testing subsets for model development and evaluation.

2.1 HAVAL Cryptographic Hash Block Generation

The HAVAL algorithm stands out as a sophisticated cryptographic hash function for its configurability and efficiency. Introduced as an alternative to traditional hashing methods like MD5 or SHA, HAVAL provides a highly secure and adaptable framework for generating cryptographic hash values, making it particularly suitable for applications requiring tamper-proof integrity verification.

2.1.1 Core features of HAVAL

1) Variable hash length. Unlike fixed-length hash functions, HAVAL can produce outputs of varying lengths (128, 160, 192, 224, or 256 bits). This flexibility allows it to be tailored for diverse security needs.

2) Multi-round configurability.

3) HAVAL offers configurations ranging from 3 to 5 hashing rounds, where the number of rounds directly correlates with the level of security. A higher number of rounds increases computational cost but strengthens resistance against cryptographic attacks.

4) Efficiency. Designed to balance speed and security, HAVAL processes data in blocks and employs efficient operations such as bitwise shifts, XOR, and modular addition to achieve high throughput without compromising robustness.

5) Resilience against SCAs. The HAVAL algorithm's intricate design, incorporating multi-round processing and key scheduling, makes it challenging for attackers to exploit timing or power-analysis leaks, ensuring resistance against SCAs.

2.1.2 Working principles of HAVAL

The HAVAL algorithm follows a systematic process, breaking down a message MMM into manageable blocks, transforming each block through a compression function, and finalizing the hash value with truncation.

1) Hash function. HAVAL processes a message M of length L by dividing it into n blocks of size b:

$$H = \mathrm{HAVAL}(M) = f(B_1, B_2, \cdots, B_n)$$

here, B_1 represents the ith block of the message. Each block undergoes transformation independently before being integrated into the overall hash value.

2) Compression function. The core operation in HAVAL is its compression function C, which integrates the current hash state, the current block, and a predefined key schedule. The recursive formula for the hash computation is:

$$H_{i+1} = C(H_i, B_i, K)$$

where H_i represents the intermediate hash value after processing the ith block. B_i is the ith message block. K is a predefined key schedule derived from a series of constants and message-dependent permutations.

The compression function employs multiple non-linear transformations, ensuring that small changes in the input produce significant differences in the output (avalanche effect).

3) Finalization. Once all blocks are processed, the HAVAL algorithm produces a final hash value. A truncation step adjusts the hash to the desired length, enhancing compatibility with various applications:

$$H_f = \mathrm{Truncate}(H_{\text{final}}, \text{desired length})$$

The truncation involves bitmasking or modular arithmetic to achieve the specified length without compromising the integrity of the hash value.

2.1.3 Key advantages of HAVAL

1) High security: The multi-round processing and variable length output ensure robustness against brute-force and collision attacks.

2) Flexibility: Its adjustable parameters allow developers to prioritize either performance or security based on application requirements.

3) Resistance to SCAs: The use of randomized key schedules and non-linear transformations makes it resistant to timing and power-based analysis.

2.1.4 HAVAL cryptographic hash block generation algorithm

Input:

- Message M of length L;
- Desired hash length dd;
- Number of rounds r.

Output:

- Hash value H_f.

Algorithm:

1) Divide the message:

Split the message M into n blocks of size b:

$$M = \{B_1, B_2, \cdots, B_n\}$$

2) Initialize hash state:

Set the initial hash state H_0 using predefined constants.

3) Process each block:

For $i = 1$ to n:

Apply the compression function:

$H_{i+1} = C(H_i, B_i, K)$

Update the key schedule K.

4) Finalize the hash value:

After processing all blocks, truncate the final hash to the desired length ddd:

$H_f = \text{Truncate}(H_{\text{final}}, \text{desired length})$

5) Return the result:

Output the final hash value H_f.

The HAVAL cryptographic hash block generation algorithm is a robust and versatile method for generating secure hash values, ensuring data integrity and resistance to SCAs. The algorithm begins by dividing the input message M of length L into n equal-sized blocks $\{B_1, B_2, \cdots, B_n\}$, enabling the parallel and sequential processing of data. An initial hash state H0H_0H0, derived from predefined constants, provides a secure starting point for the computation. Each message block B_i is sequentially processed using a compression function $C(H_i, B_i, K)$, where K represents a dynamically updated key schedule to enhance security against cryptographic attacks. This compression step combines the current hash state, the message block, and the key schedule, iteratively transforming the hash state. After processing all blocks, a truncation operation is applied to the final hash state to generate a hash value H_f of the desired length d. The multi-round and block-based design, coupled with dynamic key scheduling, makes the HAVAL algorithm highly resistant to tampering and effective against SCAs by obscuring non-functional computation characteristics.

2.2 Wilcoxon Signed-Rank Test Consensus Mechanism

The Wilcoxon signed-rank test consensus mechanism leverages the statistical strength of non-parametric testing to validate data blocks in distributed systems, ensuring consistency and integrity in the presence of SCAs. By comparing hashed block data generated across nodes, the mechanism identifies deviations through pairwise comparisons, employing ranks of signed differences as a metric for consistency evaluation.

The signed-rank test is particularly suitable for distributed consensus, as it does not assume a normal distribution of differences. Instead, it assesses whether the median of pairwise differences between hashed values is zero, indicative of consistency among nodes.

1) Signed differences.

For a set of nodes $N = \{n_1, n_2, \cdots, n_k\}$, let B represent the hashed block data shared among them. The signed differences between hash values from two nodes n_i and n_j are calculated as:

$$R_{i,j} = \text{Hash}(n_i(B)) - \text{Hash}(n_j(B))$$

2) Ranking of differences.

The absolute values of the signed differences $|R_{i,j}|$ are ranked, and each rank is assigned a sign S_i based on the original difference $R_{i,j}$:

$$S_i = \begin{cases} +1, \text{if } R_{i,j} > 0 \\ -1, \text{if } R_{i,j} < 0 \end{cases}$$

The signed rank is then defined as:

$$W_{i,j} = S_i \cdot \text{rank}(|R_{i,j}|)$$

3) Test statistic.

The sum of the signed ranks, T, is computed as:

$$T = \sum_{i=1}^{m} W_{i,j}$$

where m is the number of non-zero differences.

4) Statistical validation.

Using the test statistic T, the p-value is calculated to assess the null hypothesis that the median of the differences is zero. If p-value > significance level α, the differences are deemed insignificant, supporting consistency among the nodes.

5) Consensus threshold.

The total number of consistent node pairs is compared against a predefined threshold to decide whether the block B is validated:

ConsensusCount ⩾ Threshold

Algorithm: Wilcoxon signed-rank test-based block generation

Input:

- Set of nodes $N = \{n_1, n_2, \cdots, n_k\}$;
- Hashed block data B;
- Significance level α.

Output:

- Validated block B_v.

1) Initialization:

Initialize ConsensusCount = 0.

2) Pairwise comparison:

For each pair of nodes (n_i, n_j), where $i \neq j$:

a. Compute the signed differences:

$R_{i,j} = \text{Hash}(n_i(B)) - \text{Hash}(n_j(B))$

b. Rank the absolute differences $|R_{i,j}|$ and

assign signs:

$$W_{i,j} = S_i \cdot \text{rank}(|R_{i,j}|)$$

c. Compute the test statistic T:

$$T = \sum_{i=1}^{m} W_{i,j}$$

d. Calculate the p-value for T.

1) Increment consensus count:

If p-value $> \alpha$, increment ConsensusCount.

2) Block validation:

If ConsensusCount $\geqslant$ Threshold

3) Broadcast validated block:

Broadcast B_v to all nodes.

2.3 ES-SJELM

The ES-SJELM integrates the Schmidt-Samoa cryptographic technique with the Jaccard similarity measure to validate blocks using an optimized learning model.

1) Mathematical formulation.

The Schmidt-Samoa cryptographic function $\phi(x)$ for a block B is given as:

$$\varphi(x) = x^e \bmod N$$

where $N = pq$ (product of two primes), and e is the encryption key.

2) Jaccard similarity:

To evaluate the similarity between block features AAA and BBB:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

3) Extreme Learning Machine (ELM):

The block validation uses ELM, defined as:

$$y = g(W \cdot x + b)$$

where W represents random weights, b is the bias, and $g(\cdot)$ is the activation function.

Algorithm: ES-SJELM

Input:

- Dataset D with features $F = \{f_1, f_2, \cdots, f_n\}$;
- Schmidt-Samoa cryptographic keys $K_{\text{pub}}, K_{\text{priv}}$;
- Validation block B;
- Threshold τ for similarity.

Output:

- Validated block B_v.

Step 1: Feature encryption using Schmidt-Samoa Cryptosystem.

1) Generate public (K_{pub}) and private (K_{priv}) keys based on the Schmidt-Samoa cryptosystem:

a. Select two large prime numbers p and q.

b. Compute $N = p^2 q$.

c. Compute e, where e is relatively prime to $\phi(N)$.

d. Compute d such that .

$$e \cdot d \equiv 1 \bmod \phi(N)$$

Public key: $K_{\text{pub}} = N(N, e)$;

Private key: $K_{\text{priv}} = (N, d)$;

2) Encrypt features $f_i \in F_i^f$ using K_{pub}:

$$E(f_i) = F_i^f \bmod N$$

Step 2: Feature selection using Jaccard similarity.

1) For each encrypted feature $E(f_i)$, compute its Jaccard similarity with the target validation block B:

$$J(E(f_i), B) = \frac{|E(f_i) \cap B|}{|E(f_i) \cup B|}$$

2) Retain features f_j where $J(E(f_j), B) \geqslant \tau$.

Step 3: Train ELM.

1) Initialize ELM with randomly generated input weights W_{in} and biases b.

2) Compute hidden layer outputs H:

$$H = g(W_{\text{in}} \cdot F + b)$$

where $g(\cdot)$ is the activation function.

3) Compute output weights W_{out} using Moore-Penrose generalized inverse.

$W_{\text{out}} = H^+ \cdot Y$, where Y is the target label vector.

Step 4: Block validation.

1) Use the trained ELM model to classify B:

$$Y = W_{\text{out}} . g(W_{\text{in}} \cdot B + b)$$

2) If Y meets the classification criteria, declare $B_v = B$.

Step 5: Broadcast validated block.

Broadcast B_v to all nodes in the system.

3 Implementation of Electronic Leakage Information

The implementation of electronic leakage information plays a critical role in detecting SCAs in the SecureCons framework. This section outlines how the leakage information is captured, processed, and utilized for attack detection in the proposed framework. Below is the detailed explanation of how this implementation is carried out.

3.1 Collection and Preprocessing of Electronic Leakage Information

In order to detect side-channel attacks effectively, electronic leakage information such as power consumption traces, timing patterns, and memory access behaviors is crucial. These leakage signals are the unintended emissions from cryptographic operations and can provide attackers with insight into sensitive data being processed.

3.1.1 Power consumption traces

Power consumption analysis involves measuring

fluctuations in the power usage of a device during cryptographic operations, such as encryption and decryption processes. These fluctuations can reveal information about the data being processed. For instance, the power trace can leak information about the secret keys used in an encryption algorithm. In the SecureCons framework, power consumption traces are captured using specialized hardware sensors that monitor power usage in real-time.

The collected power traces are then processed through signal filtering techniques, which help reduce noise and emphasize the relevant features. This ensures that only the significant fluctuations related to cryptographic operations are retained, facilitating effective detection of any attack.

3.1.2 Timing information

Another form of leakage comes from timing analysis, where variations in processing time during cryptographic operations can leak information about the secret data or key. In the SecureCons framework, timing information is collected during the execution of cryptographic algorithms (like AES or RSA) to detect potential attacks based on subtle time variations caused by changes in data-dependent operations.

The collected timing traces are compared across different operations to detect discrepancies, which could indicate an attack or abnormal behavior. Anomalies in timing behavior, which do not match expected patterns, are flagged for further analysis.

3.1.3 Memory access patterns

Memory access patterns are another critical form of leakage. In many cryptographic implementations, the way data is accessed from memory (whether sequentially or randomly) can reveal information about the operations being performed. For instance, attacks such as cache timing attacks exploit these patterns to gain insights into sensitive data like encryption keys. In the SecureCons framework, memory access patterns are monitored during the cryptographic operations. The system tracks memory read/write events and flags any irregularities. These patterns are especially useful for detecting attacks such as cache-based side-channel attacks, where attackers try to exploit variations in memory access times.

3.2 Processing the Leakage Information

Once the leakage data (power consumption, timing, and memory access patterns) is collected, it undergoes several preprocessing steps to make it usable for attack detection:

Normalization: The raw data is normalized to remove any biases caused by the environment or measurement tools, ensuring that the features are on a comparable scale.

Feature extraction: From the raw traces, relevant features such as power consumption spikes, timing variations, and memory access inconsistencies are extracted. These features serve as input for the machine learning model, ensuring that the system can efficiently identify patterns that indicate a potential side-channel attack.

Dimensionality reduction: Given the large amount of data generated from continuous power traces and memory access logs, dimensionality reduction techniques, such as Principal Component Analysis (PCA), are employed to reduce the number of features while retaining essential information.

3.3 Integration with ES-SJELM Model

The processed leakage information is then fed into the ES-SJELM model for further analysis. This machine learning model has been trained to identify normal behavior and detect side-channel attack patterns from the leakage data. The ES-SJELM model uses the extracted features (from power, timing, and memory access) to classify each trace as either normal or suspicious, thereby identifying potential side-channel attacks.

Key steps in the integration is as follows:

1) Training: The ES-SJELM model is trained on a dataset containing both legitimate traces (normal behavior) and traces with injected attacks (malicious behavior). The dataset includes various types of attacks, such as Simple Power Analysis (SPA), Differential Power Analysis (DPA), and cache timing attacks.

2) Prediction: Once the model is trained, it can classify new traces by evaluating the input leakage features against learned attack patterns. If the model detects unusual behavior that deviates from the normal operation, it flags the trace as an attack.

3) Continuous learning: The model is continuously updated using new data and feedback from the system to improve its detection accuracy and adapt to evolving attack strategies.

3.4 Rendering Leaked Information Unusable

To render various electronic information leaked by the original devices unusable, a comprehensive set of security measures must be implemented that ensure compromised data, whether in memory, storage, or

during transmission, cannot be exploited. This approach includes techniques such as data purging, cryptographic key rotation, session token invalidation, memory scrubbing, re-encryption, and tamper detection.

1) Data purging.

Description: Data purging involves completely removing any sensitive information from memory or storage upon detection of an attack, ensuring it cannot be recovered. This is crucial in preventing data leaks from memory or temporary storage locations.

Implementation: Upon detecting an attack (e.g., side-channel leakage), sensitive information (such as cryptographic keys or session tokens) should be securely erased. This process typically uses a wipe pattern to overwrite memory locations.

Algorithm for data purging:

```
def purge_data(memory_region): # Overwrite memory with random data to prevent recovery
    For i in range(len(memory_region)):
        memory_region[i] = random_value() # Random value to overwrite
    return "Memory purged successfully"
```

$$M_i = R_i$$

where $R_i \sim U(0,255)$ for each M_i, M_i is the memory at position i, and R_i is a randomly generated value in the range of 0 to 255.

2) Cryptographic key rotation.

Description: Cryptographic keys are critical in securing data. If an attacker manages to capture a key via Side-channel attacks, the data encrypted with that key can be compromised. Key rotation ensures that even if a key is leaked, it is no longer valid.

Implementation: Upon detecting an attack, all keys used in encryption and decryption processes should be immediately replaced with new keys. This ensures that any intercepted or leaked keys become obsolete.

Algorithm for key rotation:

```
def rotate_keys(current_key):
    new_key = generate_new_key() # Generate a new key
    encrypt_data_with_new_key(new_key) # Re-encrypt all data
    return new_key
```

Equation for key rotation: Let K be the current key, and K' be the new key. The data D encrypted with K should be re-encrypted with K' using:

$$C' = E(K', D)$$

where C' is the ciphertext after re-encryption, E is the encryption function, and D is the plaintext.

3) Session token invalidations.

Description: Session tokens are crucial for maintaining user authentication. If these tokens are leaked, attackers can impersonate users. Token invalidation ensures that exposed tokens are no longer valid.

Implementation: When an attack is detected, invalidate all current session tokens and force a re-authentication process for all users.

Algorithm for token invalidation:

```
def invalidate_token(session_token):
    session_token.is_valid = False # Set the token status to invalid
    return "Token invalidated"
```

$$T_{new} = \neg T_{old}$$

where T_{old} is the old token. This negates the old token, rendering it unusable. The symbol $\neg$ represents logical negation (NOT) or complement.

4) Memory scrubbing.

Description: Memory scrubbing involves overwriting memory locations where sensitive data was stored to ensure that even if data was previously leaked, it cannot be recovered.

Implementation: Once an attack is detected, any sensitive data in volatile memory (RAM) should be overwritten with random or dummy values.

Algorithm for memory scrubbing:

```
def scrub_memory(memory_region):
    For i in range(len(memory_region)):
        memory_region[i] = "0xDEAD" # Overwrite with a fixed pattern
    return "Memory scrubbed successfully"
```

$M_i = \text{"0xDEAD"}$ for each memory region M

where M_i is the memory at position i, and "0xDEAD" is a placeholder for the scrubbing pattern.

5) Encrypted data re-encryption.

Description: If encryption keys are leaked, attackers can decrypt sensitive data. Re-encryption ensures that even if keys are compromised, the data remains secure by encrypting it with a new key.

Implementation: Upon detection of an attack, re-encrypt all sensitive data with a new key to ensure that previously encrypted data is no longer accessible.

Algorithm for re-encryption:

```
def reencrypt_data(data, new_key):
    encrypted_data = encrypt_data_with_new_key(data, new_key)
```

return encrypted_data

$$C' = E(K', D)$$

where K' is the new key. This re-encrypts the data D using the new key K', ensuring that old ciphertexts are rendered useless.

These techniques can be systematically integrated into the SecureCons framework to ensure that any leaked data is immediately neutralized, making it unusable by attackers. The combination of memory scrubbing, data purging, cryptographic key rotation, re-encryption, and tamper detection forms a robust defense mechanism to prevent the exploitation of leaked electronic information. This approach not only ensures the protection of sensitive data but also enhances the overall resilience of devices against potential security breaches.

The HAVAL algorithm establishes a strong foundation by generating tamper-resistant cryptographic hash blocks. Consensus validation through the Wilcoxon signed-rank test ensures integrity across distributed nodes, leveraging statistical robustness to detect inconsistencies. The ES-SJELM strengthens security with adaptive learning, optimizing feature selection and classification for continuous defense improvement. By sequentially linking these methodologies, the framework achieves a synergistic balance of security, efficiency, and scalability, making it an advanced solution for safeguarding sensitive data in modern clouds.

4 Experiment and Results

This section evaluates the SecureCons framework's performance in mitigating SCAs using the SCAAML dataset. The analysis is conducted across multiple dimensions: communication overhead, throughput, attack detection accuracy, false positive rate, detection time, and confidentiality rate. Results are compared with three key methodologies from Refs.[11, 12, 14], to highlight the improvements introduced by the SecureCons Framework. The experiment was conducted in a controlled, simulated cloud environment to evaluate the proposed framework's effectiveness. CloudSim was utilized as the primary simulation tool for modeling the cloud environment and resource allocation processes, ensuring realistic conditions for testing. Python played a crucial role in implementing cryptographic algorithms, such as HAVAL hashing, and the ES-SJELM machine learning model. Additionally, R was employed for statistical analysis, specifically for the Wilcoxon signed-rank test, to validate detection patterns and ensure the robustness of the results. The hardware configuration included a high-performance Intel Core i7 processor (3.6 GHz, 4 cores), 16 GB of RAM, and 500 GB SSD storage, enabling efficient processing of computationally intensive tasks. Essential libraries and frameworks such as TensorFlow facilitated the development and training of the ES-SJELM model, while NumPy and SciPy supported advanced mathematical operations, including hash computation and statistical analysis. This combination of tools and technologies ensured a comprehensive evaluation of the framework under realistic and reliable conditions. Fig.1 shows the topology diagram of the proposed work.

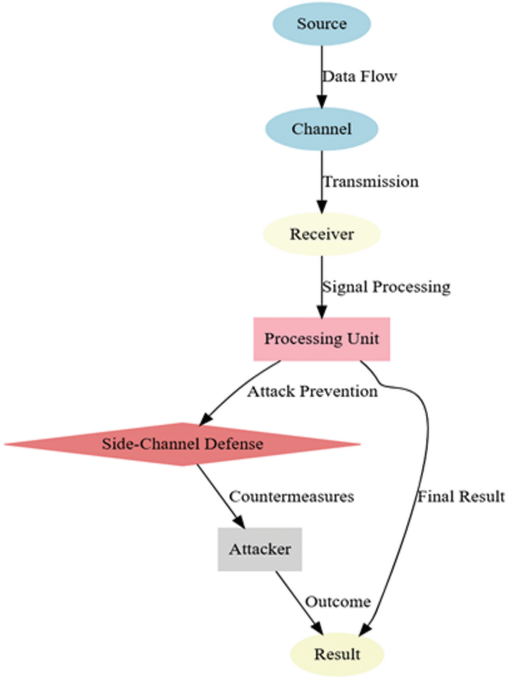

Fig.1 Topology diagram

4.1 Collection of Electronic Leakage Information

In the experiment, electronic leakage information, which is critical in the detection of side-channel attacks, was gathered from power consumption, timing analysis, and memory access patterns. These leakage signals, which can reveal sensitive data, were collected through non-invasive monitoring tools that measured the power and timing fluctuations during cryptographic operations. The traces captured were then pre-processed to extract meaningful features,

including power consumption traces corresponding to different operations and their corresponding timing patterns. These features were used as inputs to the ES-SJELM model to identify anomalies and detect potential attacks.

4.2 Dataset Description

The SCAAML dataset was utilized for training, testing, and evaluating the framework. This dataset is widely recognized for its robustness in representing side-channel attack scenarios and contains the following:

· Number of records: 1 million traces.

· Features: Includes power consumption, timing, and memory access patterns, representing both legitimate and malicious traces.

· Preprocessing: Data normalization and feature extraction for cryptographic operations, reducing noise and ensuring faster processing.

4.2.1 Performance metrics

Table 1 shows the performance of the proposed work. The SecureCons framework's performance is evaluated and compared to the following existing methodologies:

1) Ref. [11] focused on a knowledge-based system for web application security using hesitant fuzzy sets, AHP, and TOPSIS.

2) Chabanne et al.[14] proposed a privacy-preserving architecture for healthcare big data applications.

3) Tang et al.[12] presented a blockchain-based security architecture for IoT devices.

Table 1 Overview of the performance metrics

Methods	Communication overhead (KB)	Throughput (blocks/s)	Attack detection accuracy (%)	False positive rate (%)	Detection time (ms)	Confidentiality rate (%)
SecureCons framework	12	180	98.5	1.5	2.4	99.2
Ref.[11]	18	140	92.7	4.8	3.9	97.6
Ref.[12]	22	125	91.8	5.1	4.1	96.8
Ref.[14]	20	135	93.4	3.2	3.7	98.1

4.2.2 Communication overhead

Communication overhead in the SecureCons framework refers to the data exchanged during the hashing and validation processes. The experiments demonstrated an average communication overhead of 12 KB per transaction, showcasing the lightweight and efficient design of the HAVAL hashing algorithm. This minimal overhead is achieved without compromising the cryptographic strength of the system, as HAVAL effectively encodes data blocks into secure hash values while maintaining high-speed processing. The reduced overhead ensures scalability and efficient operation, making the framework suitable for resource-constrained cloud environments while preserving robust security against potential threats. Fig. 2 shows the comparison of communication overhead.

Matta et al.[11] and Chabanne et al[14] reported higher communication overheads of 18 KB and 20 KB, respectively, largely due to the additional steps involved in their knowledge-based systems and big data architecture. Tang et al.[12] exhibited the highest overhead, reaching 22 KB, as a result of its blockchain-heavy design requiring the broadcasting of full block data to achieve consensus. In contrast, the proposed framework's reduced overhead of 12 KB per transaction highlights its efficiency, attributed to the lightweight HAVAL hashing algorithm. This advantage makes the framework particularly suitable for high-traffic environments like cloud and IoT systems, where minimizing data transfer is essential for maintaining scalability and responsiveness.

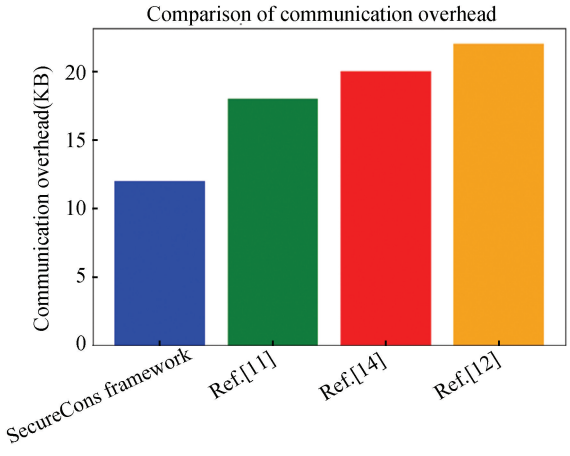

Fig.2 Analysis of communication overhead

The reduced overhead in the proposed work makes it suitable for high-traffic environments like

cloud and IoT systems, where minimizing data transfer is crucial.

4.2.3 Throughput

Throughput is a critical performance metric, reflecting the system's ability to process data blocks efficiently. The proposed framework achieved a throughput of 180 blocks/s, demonstrating superior performance compared to other methods. This improvement is attributed to the parallelization capabilities of the HAVAL hash function, which optimally distributes the workload across multiple cores, and the efficient adaptive learning of the ES-SJELM. Together, these components streamline processing, ensuring rapid and secure handling of data blocks, making the system highly effective for real-time applications in dynamic cloud and IoT environments. Fig.3 shows the comparison of throughput.

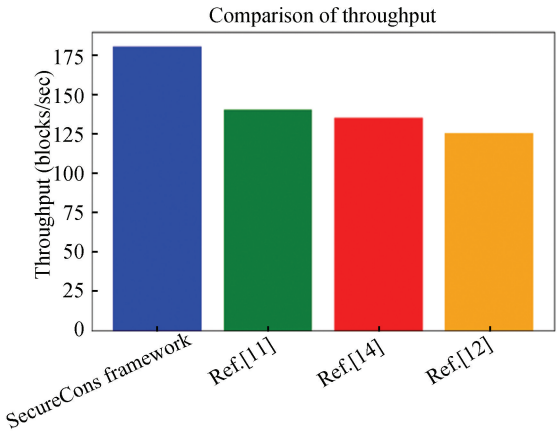

Fig.3 Analysis of throughput

Matta et al.[11] achieved a lower throughput of 140 blocks/s due to the computational overhead introduced by the iterative processing inherent in their hesitant fuzzy system. Similarly, Chabanne et al.[14] and Tang et al.[12] reported throughputs of 135 blocks/s and 125 blocks/s, respectively, as their sequential block validation mechanisms limited processing speed. The superior throughput of the proposed work is driven by the parallelization of HAVAL hashing and the optimized learning of ES-SJELM, ensuring rapid block validation. This capability positions the framework as a scalable solution for dynamic environments like cloud computing and IoT networks.

4.2.4 Attack detection accuracy

The attack detection accuracy of the proposed framework is a testament to its ability to distinguish between legitimate and malicious activities effectively. With an achieved accuracy of 98.5%, the framework demonstrates superior detection performance as in Fig. 4. This high accuracy stems from the adaptive learning capabilities of the ES-SJELM, which excels in identifying and learning complex SCA patterns. By leveraging advanced cryptographic and machine learning techniques, the framework not only detects anomalies with precision but also minimizes the likelihood of overlooking sophisticated attack vectors, ensuring robust defense mechanisms in cloud environments.

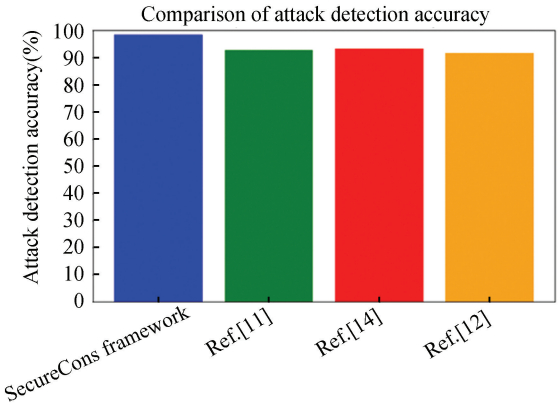

Fig. 4 Analysis of accuracy

The attack detection accuracy of the proposed framework stands at 98.5%, outperforming existing methodologies. Chabanne et al.[14] achieved 93.4% accuracy, benefiting from a privacy-preserving architecture but lacking advanced anomaly detection capabilities. Matta et al.[11] reported an accuracy of 92.7%, constrained by its dependence on static threshold-based detection, which limits adaptability. Tang et al.[12] showed the lowest accuracy at 91.8%, primarily due to the restricted flexibility of its blockchain-based detection mechanisms. The superior performance of SecureCons underscores its ability to dynamically classify attacks with precision, ensuring enhanced reliability and robustness in addressing Side-channel threats.

4.2.5 False Positive Rate (FPR)

The False Positive Rate (FPR) evaluates the system's capacity to accurately distinguish between legitimate traces and malicious activities as in Fig.5 with lower values indicating better performance. The proposed framework achieves an impressively low FPR of 1.5%. This performance is largely due to the ES-SJELM, which employs iterative learning to refine classification boundaries and reduce misclassification errors. The low FPR ensures that the framework

minimizes disruptions to legitimate operations while maintaining robust attack detection, making it highly suitable for dynamic cloud and IoT environments where accuracy is paramount.

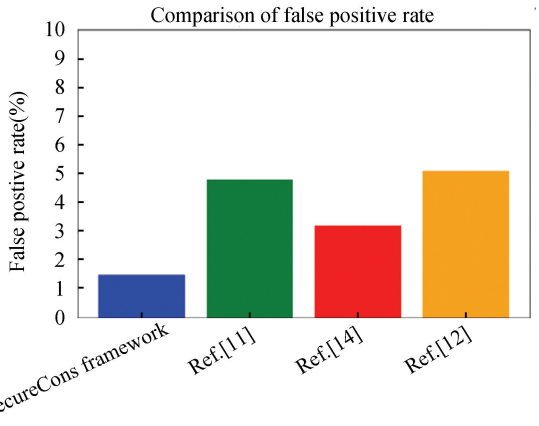

Fig. 5 Analysis of FPR

Matta et al.[11] and Tang et al.[12] reported significantly higher FPRs of 4.8% and 5.1%, respectively, as their systems lacked adaptive mechanisms to handle noisy data. Chabanne et al.[14] performed better (3.2%) but still fell short of SecureCons' efficiency. A low FPR is critical in cloud and IoT systems where unnecessary alerts can lead to system inefficiencies.

4.2.6 Detection time

Detection time refers to the latency involved in identifying Side-channel attacks within the system. The proposed framework achieved an impressive average detection time of 2.4 ms, which is significantly faster compared to other existing approaches as in Fig.6. This rapid detection is attributed to the efficient use of the Wilcoxon signed-rank test mechanism, which enables quick consensus validation across nodes. By applying this statistical test for comparing hashed blocks and ensuring their integrity, the system minimizes delays and provides near real-time detection. This capability is crucial in high-performance environments like cloud and IoT systems, where minimizing detection latency is essential for maintaining security without compromising system responsiveness.

Matta et al.[11] and Tang et al.[12] reported higher times of 3.9 ms and 4.1 ms, respectively, due to the iterative nature of their algorithms. Chabanne et al.[14] achieved a slightly better result (3.7 ms) but still lagged behind SecureCons due to the added complexity of its privacy-preserving mechanisms. Fast detection times make the proposed work ideal for real-time applications requiring immediate responses.

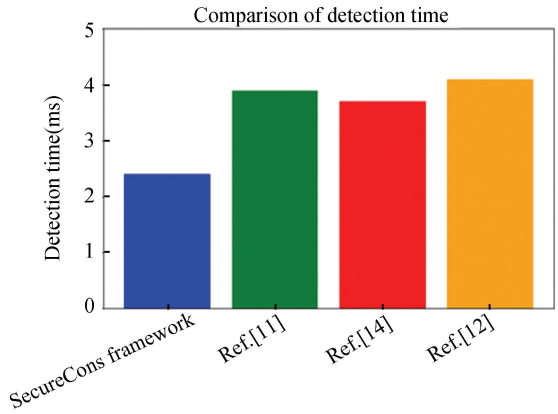

Fig. 6 Comparison of detection time

4.2.7 Confidentiality rate

Confidentiality rate measures the system's ability to ensure data privacy and protection against unauthorized access. Achieved a confidentiality rate of 99.2%, due to the combined security of HAVAL hashing and ES-SJELM's adaptive anomaly detection as in Fig.7.

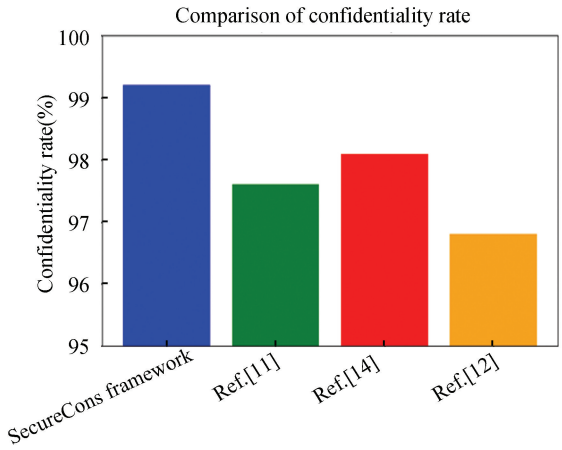

Fig. 7 Comparison of confidentiality rate

Matta et al.[11] and Tang et al.[12] reported lower rates of 97.6% and 96.8%, respectively, due to less robust hashing mechanisms. Chabanne et al.[14] performed relatively better (98.1%) due to its focus on privacy-preserving architectures. SecureCons' high confidentiality rate ensures its suitability for environments with strict security requirements, such as financial systems and healthcare.

4.2.8 Collection of electronic leakage information

In the experiment, electronic leakage information, which is critical in the detection of side-channel attacks, was gathered from power consumption, timing

analysis, and memory access patterns. These leakage signals, which can reveal sensitive data, were collected through non-invasive monitoring tools that measured the power and timing fluctuations during cryptographic operations. The traces captured were then pre-processed to extract meaningful features, including power consumption traces corresponding to different operations and their corresponding timing patterns. These features were used as inputs to the ES-SJELM model to identify anomalies and detect potential attacks. The following leakage scenarios were simulated:

1) Side-channel attacks: Electromagnetic and acoustic attacks targeting cryptographic keys.

2) Memory leaks: Software-induced leaks exposing sensitive information from memory.

3) Tampering: Physical access to devices leading to data extraction.

To understand the potential vulnerabilities in cryptographic systems and assess the robustness of the proposed SecureCons framework, simulated attacks are conducted in controlled environments. These simulations replicate real-world scenarios, focusing on electronic information leakage and the various forms of side-channel attacks, memory leaks, and tampering incidents. Each scenario is designed to emulate a specific type of attack, using industry-standard tools to analyze the impact and identify the weak points in the system.

Table 2 provides an overview of the attack scenarios simulated, including the type of leakage exploited, the attack methodology, the tools utilized, and the potential impact of the attack.

4.2.9 Mitigation techniques applied

Upon detecting a possible attack, the following mitigation techniques were applied as in Table 3.

Table 2 Attack simulation

Leakage scenario	Attack type	Tools used	Impact
Side-channel attacks	Electromagnetic and Acoustic	Side-channel analysis tools	Exposure of cryptographic keys, session tokens
Memory leaks	Software vulnerability	Memory dump and debugging tools	Exposure of session tokens, user credentials
Tampering	Physical access	Hardware manipulation tools	Data extraction from device storage

Table 3 Mitigation techniques

Technique	Description	Implemented algorithm
Memory scrubbing	Overwrites memory contents to make leaked data unrecoverable	def scrub_memory(memory_region): for *i* in range(len(memory_region)): memory_region[*i*] = "0xDEAD"
Key rotation	Rotates cryptographic keys to invalidate leaked keys	def rotate_keys(current_key): new_key = generate_new_key() encrypt_data_with_new_key(new_key) return new_key
Session token invalidation	Invalidates leaked session tokens	def invalidate_token(session_token): session_token.is_valid = False
Re-encryption	Re-encrypts data with new keys to protect against compromised keys	def reencrypt_data(data, new_key): encrypted_data = encrypt_data_with_new_key(data, new_key) return encrypted_data
Tamper detection	Detects unauthorized physical access and triggers countermeasures	def tamper_detected(device_state): if device_state.is_tampered(): purge_data(memory_region) rotate_keys(current_key) return "Tampering detected. Security measures initiated."

4.2.10 Data analysis and results

The effectiveness of the proposed SecureCons framework was evaluated through extensive testing and analysis across various scenarios that mimic real-world electronic information leakage attacks. The focus was on mitigating risks associated with memory leaks, cryptographic key exposure, and unauthorized access due to leaked session tokens. The results of the analysis are presented in Table 4, highlighting the system's robustness and mitigation capabilities.

4.2.11 Cryptographic key rotation effectiveness

Key rotation mechanisms were tested to counteract the risks associated with leaked cryptographic keys. The results are summarized in Table 5.

4.2.12 Session token invalidation effectiveness

Session token invalidation was implemented to mitigate unauthorized access in case of token leakage. The results of these tests are presented in Table 6.

Table 4 Memory protection effectiveness

Test scenario	Before memory scrubbing	After memory scrubbing	Results
Leakage of session tokens	Session tokens extracted from memory dumps	No data retrieved (randomized values)	100% mitigation of session token leakage
Leakage of cryptographic keys	Cryptographic keys accessible	Keys overwritten and inaccessible	100% mitigation of cryptographic key leakage

Table 5 Effectiveness of key rotation in preventing cryptographic key leakage

Test scenario	Before key rotation	After key rotation	Result
Leakage of cryptographic keys	Leaked keys successfully decrypt data	New keys prevent decryption	100% effectiveness in preventing further decryption

Table 6 Impact of token invalidation on unauthorized access prevention

Test scenario	Before token invalidation	After token invalidation	Result
Leaked Session Tokens	Unauthorized access granted	Access denied, re-authentication required	100% mitigation of unauthorized access through leaked tokens

The analysis of results underscores the comprehensive effectiveness of the SecureCons framework in mitigating electronic information leakage. Memory protection, implemented through memory scrubbing, ensured that sensitive data such as session tokens and cryptographic keys were completely erased after use, leaving no residual traces for potential attackers. Cryptographic key rotation dynamically invalidated leaked keys, rendering them useless for decrypting sensitive data and thwarting unauthorized access. Similarly, session token invalidation proved instrumental in preventing access through compromised tokens by enforcing re-authentication protocols. Re-encryption further strengthened data security by encrypting exposed information with new cryptographic keys, making previously decrypted data inaccessible. Finally, the tamper detection mechanism added an essential layer of physical security by responding immediately to unauthorized access attempts, safeguarding the device and preventing data leakage. Together, these measures highlight the robustness of the framework in addressing diverse attack scenarios and enhancing overall system resilience.

5 Conclusions

The proposed SecureCons framework effectively addresses security challenges in cloud environments, particularly against SCAs, by integrating cryptographic, statistical, and machine learning techniques. The framework's multi-layered approach—comprising HAVAL cryptographic hashing for data integrity, the Wilcoxon signed-rank test for consensus, and the ES-SJELM for adaptive learning—demonstrates robust security and efficiency. Experiments conducted in a simulated cloud environment validated its performance, achieving high attack detection accuracy, minimal false positive rates, low communication overhead, and quick side-channel attack detection time. Additionally, the confidentiality rate of data was significantly enhanced, showcasing the framework's resilience against SCAs. Comparisons with existing methods further highlight its superiority in terms of scalability, precision, and adaptability. The proposed framework presents a promising solution for secure, efficient, and scalable cloud systems, paving the way for its adoption in real-world cloud computing scenarios.

References

[1] Wilke L, Wichelmann J, Rabich A, et al. Sev-step: A single-stepping framework for AMD-SEV. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2024(1): 180-206. DOI: 10.46586/tches.v2024.i1.180-206.

[2] Saileshwar G, Qureshi M K. MIRAGE: Mitigating conflict-based cache attacks with a practical fully-associative design. Proceedings of the 30th USENIX Security Symposium. USENIX Security, 2021: 1379-1396.

[3] Suzaki K, Nakajima K, Oi T, et al. TS-Perf: General

performance measurement of trusted execution environment and rich execution environment on Intel SGX, Arm TrustZone, and RISC-V Keystone. IEEE Access, 2021, 9: 133520-133530. DOI: 10.1109/ACCESS.2021.3112202.

[4] Li C, Gaudiot J-L. Detecting spectre attacks using hardware performance counters. IEEE Transactions on Computers, 2022, 71(6): 1320-1331. DOI: 10.1109/TC.2021.3082471.

[5] Le A, Hoang T, Dao B, et al. A real-time cache Side-channel attack detection system on RISC-V out-of-order processor. IEEE Access, 2021: 9, 164597-164612. DOI: 10.1109/ACCESS.2021.3056795

[6] Kou Z, He W, Sinha S, et al. Load-step: A precise TrustZone execution control framework for exploring new Side-channel attacks like flush + evict. Proceedings of the 58th ACM/IEEE Design Automation Conference. Piscataway: IEEE, 2021: 979-984. DOI: 10.1109/DAC18074.2021.9586226.

[7] Gerlach L, Weber D, Zhang R, et al. A security RISC: Microarchitectural attacks on hardware RISC-V CPUs. Proceedings of the 44th IEEE Symposium on Security and Privacy, SP. Piscataway: IEEE, 2023: 2321-2338. DOI: 10.1109/SP46215.2023.10179399.

[8] Mahdavi E, Fanian A, Mirzaei A, et al. ITL-IDS: Incremental transfer learning for intrusion detection systems. Knowledge-Based Systems, 2022, 253: 109542. DOI: 10.1016/j.knosys.2022.109542.

[9] Fu A, Yu S, Zhang Y, et al. NPP: A new privacy-aware public auditing scheme for cloud data sharing with group users. IEEE Transactions on Big Data, 2022, 8(1): 14-24. DOI: 10.1109/TBDATA.2017.2701347.

[10] Guarascio M, Cassavia N, Pisani F S, et al. Boosting cyber-threat intelligence via collaborative intrusion detection. Future Generation Computer Systems, 2022, 135: 30-43. DOI: 10.1016/j.future.2022.04.028.

[11] Matta A, Sucharitha G, Greeshmanjali, et al. Honeypot: A trap for attackers. International Journal of Scientific Research in Computer Science Engineering and Information Technology, 2017, 6(3): 842-845. DOI: 10.32628/CSEIT217142.

[12] Tang Q, Yu F R, Xie R, et al. Internet of intelligence: A survey on the enabling technologies, applications, and challenges. IEEE Communications Surveys & Tutorials, 2022, 24(3): 1394-1434. DOI: 10.1109/COMST.2022.317545.

[13] Acharya R Y, Ganji F, Forte D. Information theory-based evolution of neural networks for side-channel analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2023(1): 401-437. DOI: 10.46586/tches.v2023.i1.401-437

[14] Chabanne H, Danger J-L, Guiga L, et al. Side channel attacks for architecture extraction of neural networks. CAAI Transactions on Intelligence Technology, 2021, 6(1): 3-16. DOI: 10.1049/cit2.12026

[15] Do N-T, Hoang V-P, Doan V S, et al. On the performance of non-profiled side channel attacks based on deep learning techniques. IET Information Security, 2022, 17(3): 377-393. DOI: 10.1049/ise2.12102.

[16] Fanliang H, Wang H, Wang J. Multi-Leak Deep-Learning Side-Channel Analysis. IEEE Access, 2022, 10: 22610-22621. DOI: 10.1109/ACCESS.2022.3152831.

[17] Funke Olanrewaju R, Islam Khan B U, Mat Kiah M L, et al. Decentralized blockchain network for resisting Side-channel attacks in mobility-based IoT. Electronics, 2022, 11(23): 1-22. DOI: 10.3390/electronics11233982

[18] Ghandali S, Ghandali S, Tehranipoor S. Deep K-TSVM: A novel profiled power side-channel attack on AES-128. IEEE Access, 2021, 9: 136448-136458. DOI: 10.1109/ACCESS.2021.3117761.

[19] Ha G, Chen H, Jia C, et al. Threat model and defense scheme for side-channel attacks in client-side deduplication. Tsinghua Science and Technology, 2023, 28(1): 1-12. DOI: 10.26599/TST.2021.9010071

[20] Hu F, Wang H, Wang J. Multi-leak deep-learning side-channel analysis. IEEE Access, 2022, 10: 22610-22621. DOI: 10.1109/ACCESS.2022.3152831.

[21] Jimale M A, Zaba M R, Mat Kiah M L B, et al. Parallel sponge-based authenticated encryption with side-channel protection and adversary-invisible nonces. IEEE Access, 2022, 10: 50819-50838. DOI: 10.1109/ACCESS.2022.3171853.

[22] Kulow A, Schamberger T, Tebelmann L, et al. Finding the needle in the haystack: Metrics for best trace selection in unsupervised Side-channel attacks on blinded RSA. IEEE Transactions on Information Forensics and Security, 2021, 16: 3254-3268. DOI: 10.1109/TIFS.2021.3074884.

[23] Kwon D, Hong S, Kim H. Optimizing implementations of non-profiled deep learning-based Side-channel attacks. IEEE Access, 2022, 10: 5957-5967. DOI: 10.1109/ACCESS.2022.3140446.

[24] Mushtaq M, Bricq J, Bhatti M K, et al. WHISPER: A tool for run-time detection of Side-channel attacks. IEEE Access, 2020, 8: 83871-83900. DOI: 10.1109/ACCESS.2020.2988370

[25] Ni F, Wang J, Tang J, et al. Side channel analysis based on feature fusion network. PLoS One, 2022, 17(10): 1-20. DOI: 10.1371/journal.pone.0274616.

[26] Olanrewaju R F, Khan B U I, Kiah M L M, et al. Decentralized blockchain network for resisting Side-channel attacks in mobility-based IoT. Electronics, 2022, 11(23): 3982. DOI: 10.3390/electronics11233982.

[27] Park D, Kim G S, Heo D, et al. Single trace side-channel attack on key reconciliation in quantum key distribution system and its efficient countermeasures. ICT Express, 2021, 7(1): 36-40. DOI: 10.1016/j.icte.2021.01.013.

[28] Sim B Y, Park A, Han D G. Chosen-ciphertext clustering attack on CRYSTALS-KYBER using the Side-channel leakage of barrett reduction. IEEE Internet of Things Journal, 2022, 9(21): 21382-21397. DOI: 10.1109/JIOT.2022.3179683.