

Resilient Cloud Services

By

Hemayamini Kurra, Glynis Dsouza, Youssif
Al Nasshif, Salim Hariri

University of Arizona

First Franco-American Workshop on Cybersecurity

18th October, 2013



Presentation Outline





-  Motivations and Background
-  Resilient Cloud Services – Architecture
 - Moving Target Defense
-  Resilient Cloud Application Services-
Architecture
-  Resilient Cloud Data Storage Services-
Architecture
-  Experimental Results
-  Quantification of Security and Resiliency
-  Conclusions and Future Work

Motivations and Background




Cybersecurity Challenges

- Current cybersecurity technologies failed to secure and protect our cyberspace resources and services
 - They are mainly signature based, manual intensive and ad-hoc;
- According to the future of Cloud Computing Survey 2011, the main **inhibitor to cloud adoption is security**.
- 43% of companies globally currently using a cloud computing service reported a **data security lapse or issue with the cloud service** their company is using within the last 12 months
- 15% of the data centers don't have **data backup and recovery plans**.
- Cyber attacks can get costly if not resolved quickly. **The average time to resolve a cyber attack is 18 days**, with an **average cost to participating organizations of \$415,748**. Results show that malicious insider attacks can take more than 45 days on average to contain.
- The cost of the data center outage is calculated as average of \$505502 per incident




Cloud Security Challenges

-  Challenging research problem due to many interdependent tasks
-  Concerns
 - Securing Data-in-Transit
 - Using the same Parameters (Key, Encryption length) for encryption of data
 - Lack of randomness
 - Software Monoculture
-  Organizations give control to cloud provider
-  Security is of major concern for the adoption of cloud computing



Software Monoculture

-  Current software systems are static
-  Easy for attacker to study behavior of system and generate attacks
-  Vulnerabilities in one software can propagate to a great extent






Security of Data-in-Transit

-  The top threats to cloud computing given by CSA (Cloud Security Alliance) states that Insecure interfaces and API's is one of the top threats to cloud computing.
-  Ensuring strong authentication and access controls in addition to encrypted transmission is one of the remedies.
-  Data which is in transit is more vulnerable to attacks when compared to the data which is at rest.

Randomness

-  Information gathering about the system to attack it is the first step in any attack
-  Randomness in the system will not let attacker have enough time to gather information about the system.

Need for Resilience

-  Resilience
-  We cannot build systems that will not be attacked
-  Attack efforts will always be present
-  Cyber resilient techniques are most promising
-  There is a need to change the game to advantage the defender over the attacker

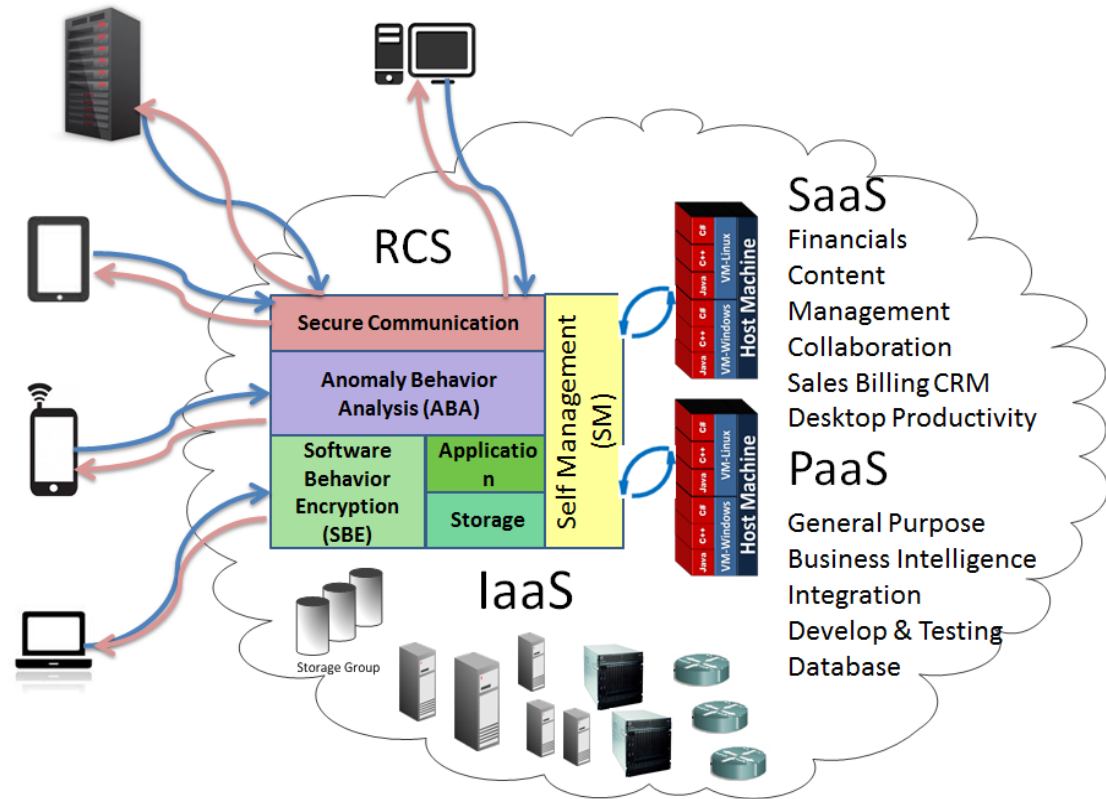
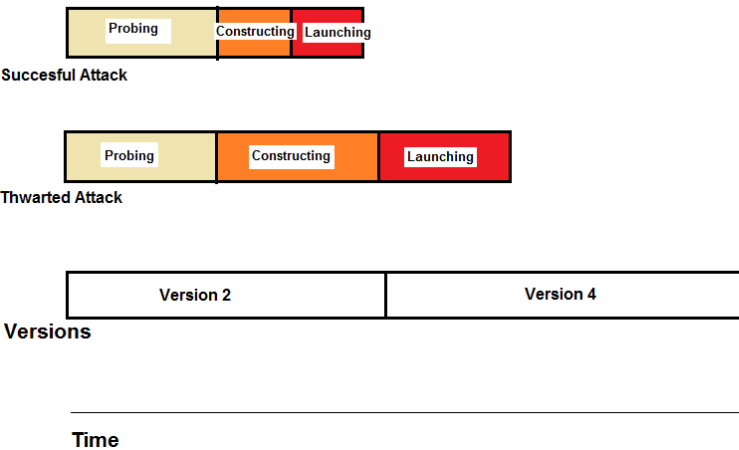
Moving Target Defense (MTD)

Vision

- Create, evaluate and deploy mechanisms and strategies that are diverse, continually shift, and change over time to increase complexity and costs for attackers, limit the exposure of vulnerabilities and opportunities for attack, and increase system resiliency (Source: "CyberSecurity Game-Change Research and Development Recommendations")

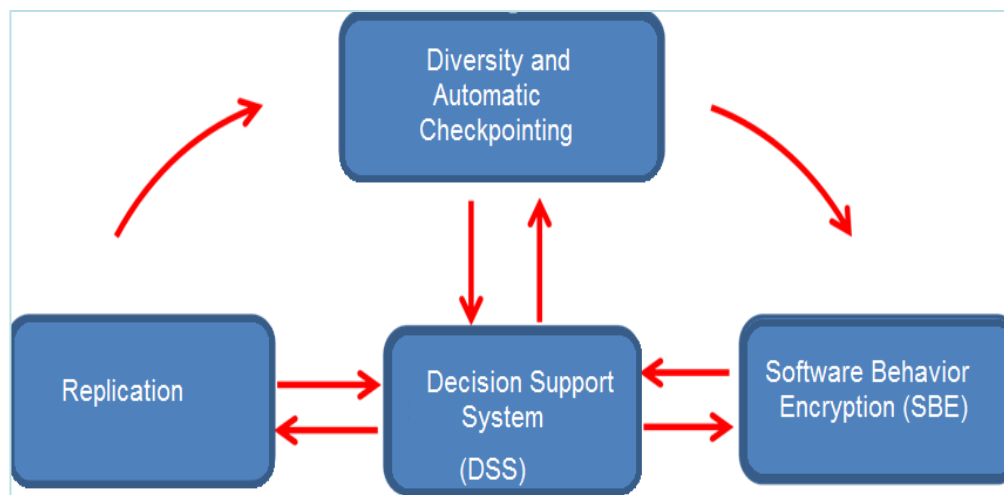
Resilient Cloud Services

Moving Target Defense (MTD)



Resilient Cloud Application Services-Architecture

Architectural Components



Architectural Components



Closed loop architecture



Continuous feedback

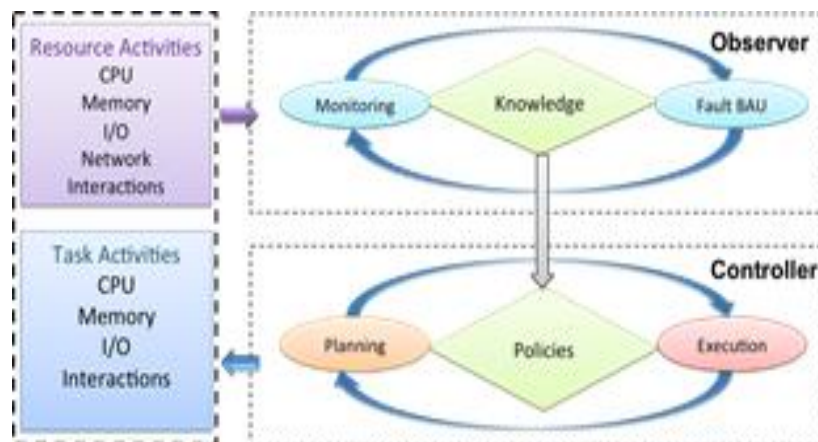
Self- Management

Observer

- Monitoring
- Analysis of current state

Controller

- Management of cyber operations
- Enforcement of resilient operational policies

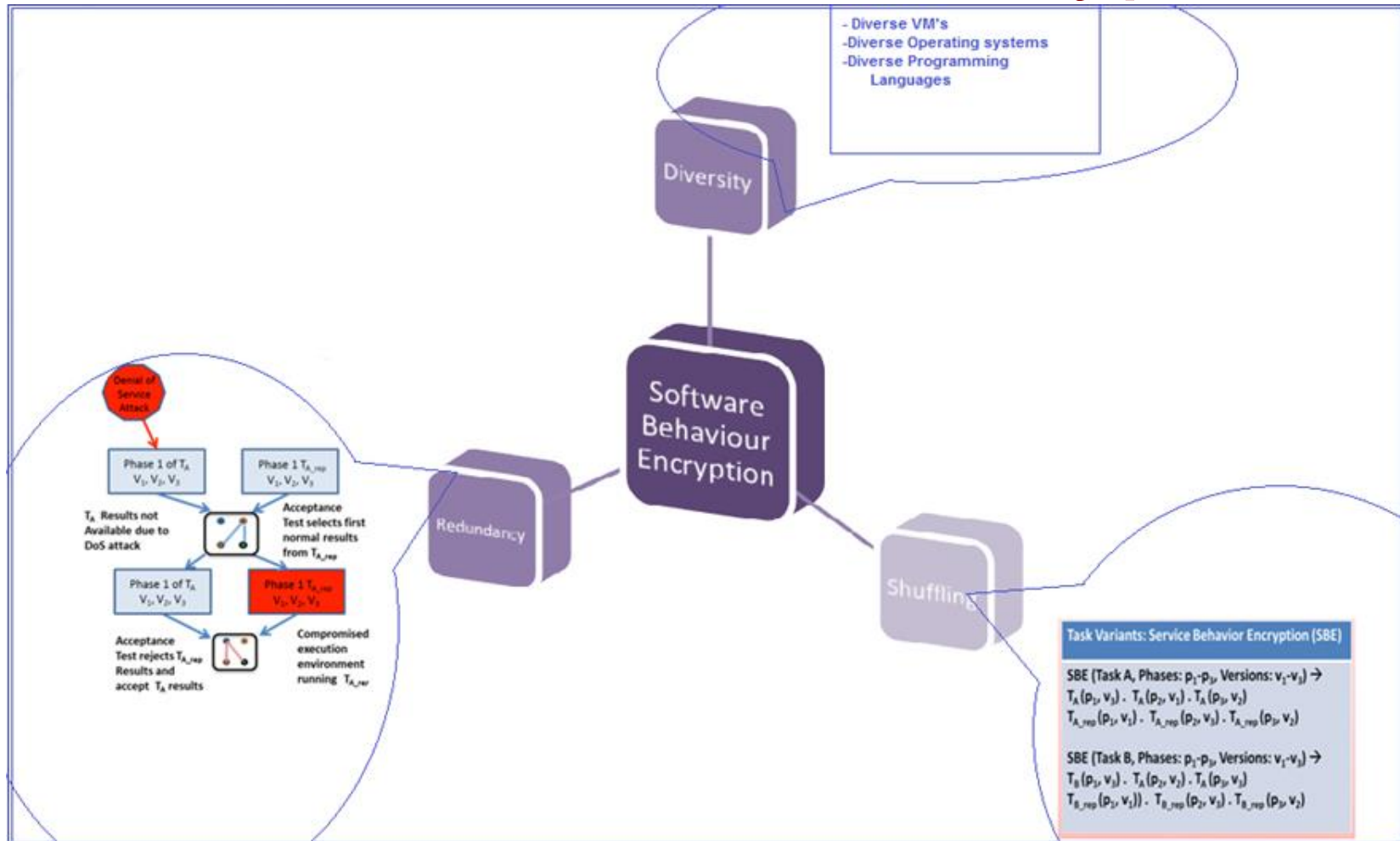


Self-Management architecture

Source:

S. Hariri, L. Xue, H. Chen, M. Zhang, S. Pavuluri and S. Rao, "AUTONOMIA: An Autonomic Computing Environment," in International Performance Computing and Communications Conference, 2003.

Software Behavior Encryption



Software Behavior Encryption



Diversity

- Hot Shuffling software variants at runtime
- Variants are functionally equivalent, behaviorally different



Redundancy

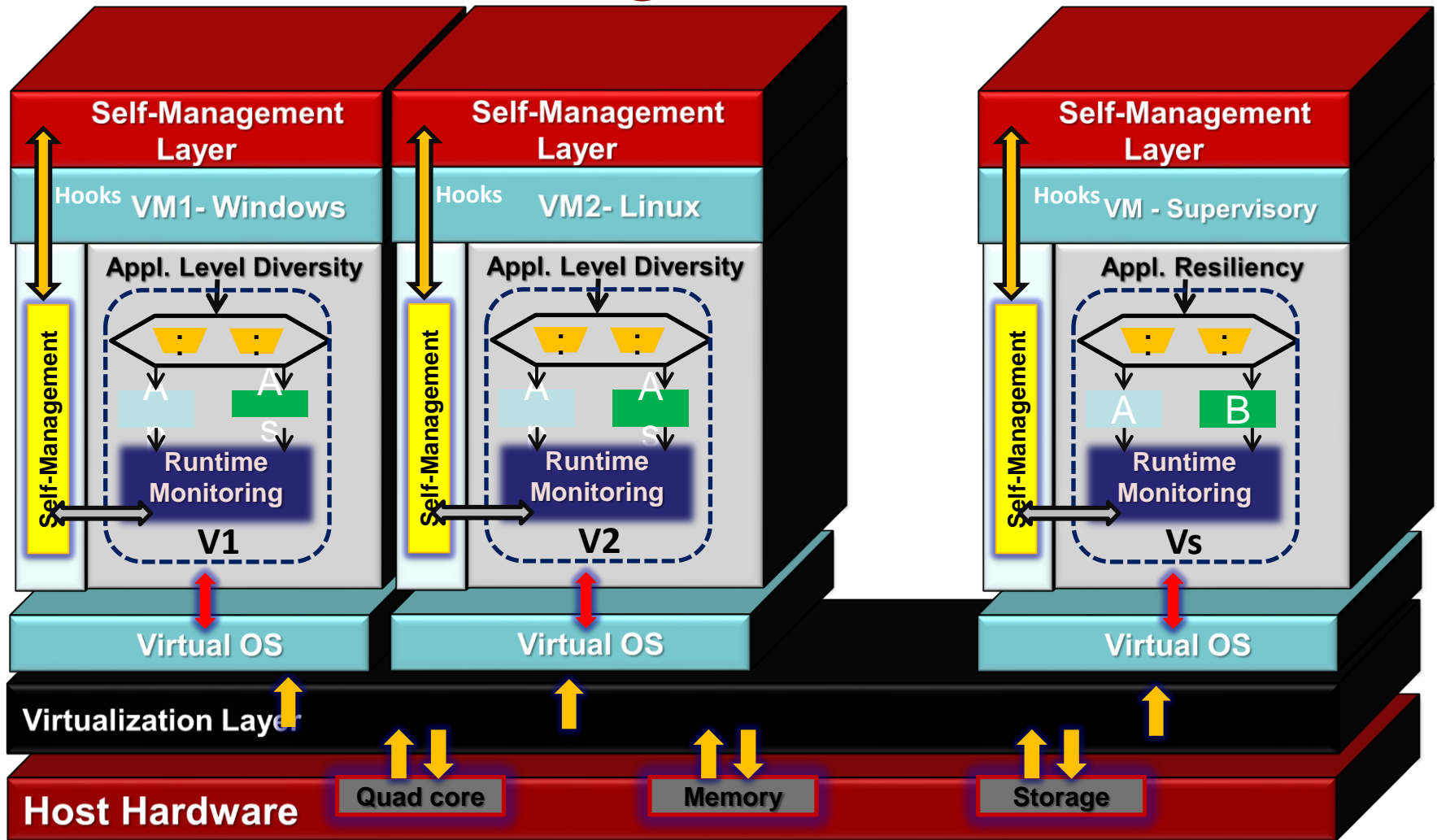
- Multiple replicas on different physical hardware



Shuffling

Experimental Results




Testbed Configuration



Experimental Environment

- 🐾 IBM BladeCenter HS22 based Private Cloud
- 🐾 University of Arizona's Autonomic Computing Lab
- 🐾 Evaluated on a three node cluster
- 🐾 Each node has multiple versions
- 🐾 Version consists of combination of:
 - Operating System
 - Programming Language
 - E.g. <Linux, C++>, <Windows, Java>

Application 1 – MapReduce (MR)

-  Large-Scale Data Processing
-  MapReduce provides
 - Automatic parallelization & distribution
-  MapReduce Wordcount program

Case 1: Resilience against Dos Attacks

Denial of Service attack on Windows VM-6

	Response Time (in seconds)	
	Without DoS attack	With DoS attack
Without MTD	95	615
With MTD	105	105

Case 2: Resilience against Insider Attacks

Compromise attack on Linux VM-1




	Response Time (in seconds)	
	Without Insider attack	With Insider attack
Without MTD	95	No response
With MTD	105	105
% increase in response time with MTD		11%

Checkpointing and Portability

 Need for Automated checkpointing

 Need for transferring state between diverse environments

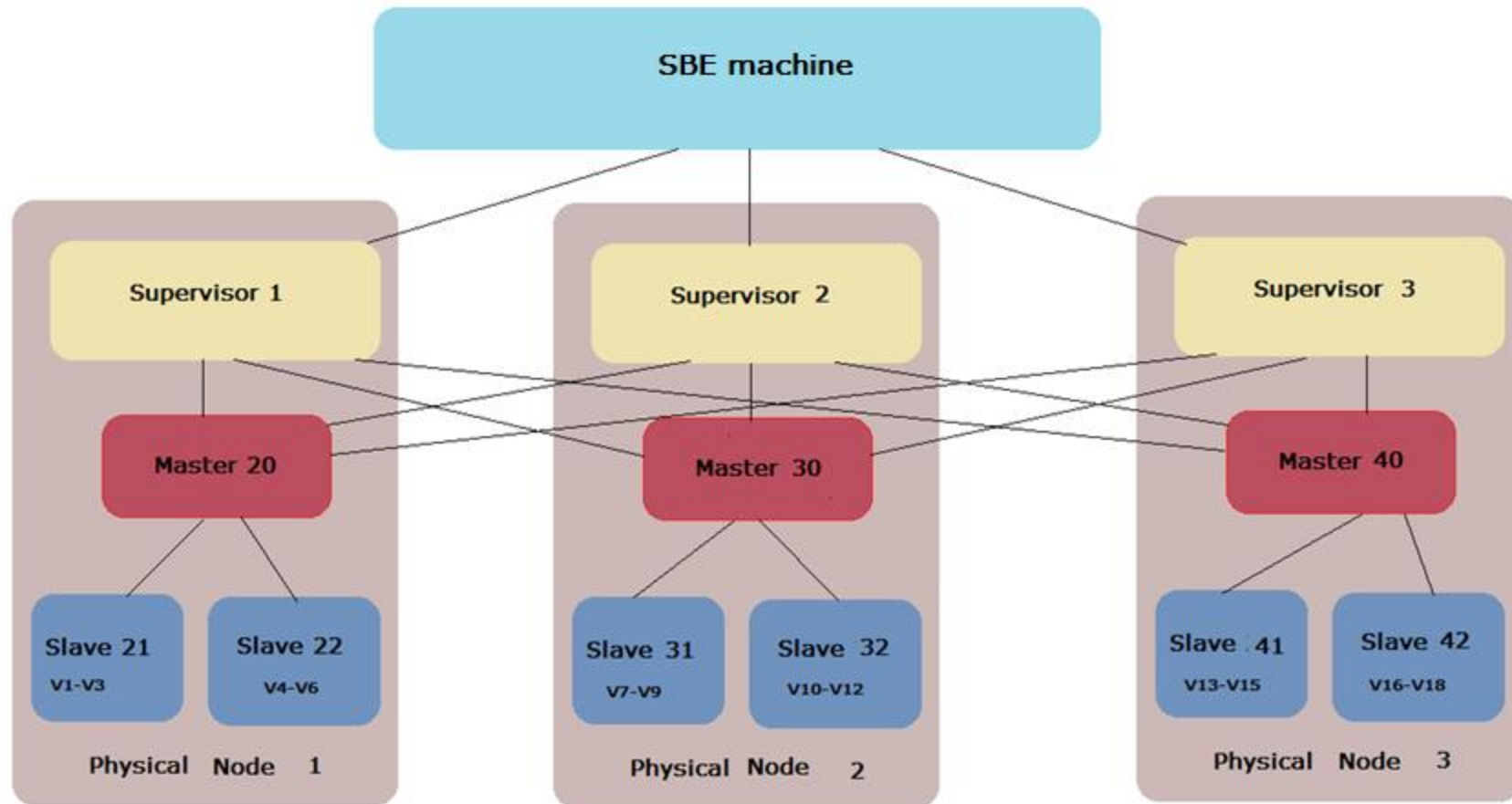
Compiler for Portable Checkpointing (CPPC)

-  Periodically saves computation state to stable storage
-  Automated checkpoint insertion in C, C++, Fortran codes
-  Ability to resume application execution by resuming state on different operating systems and programming languages

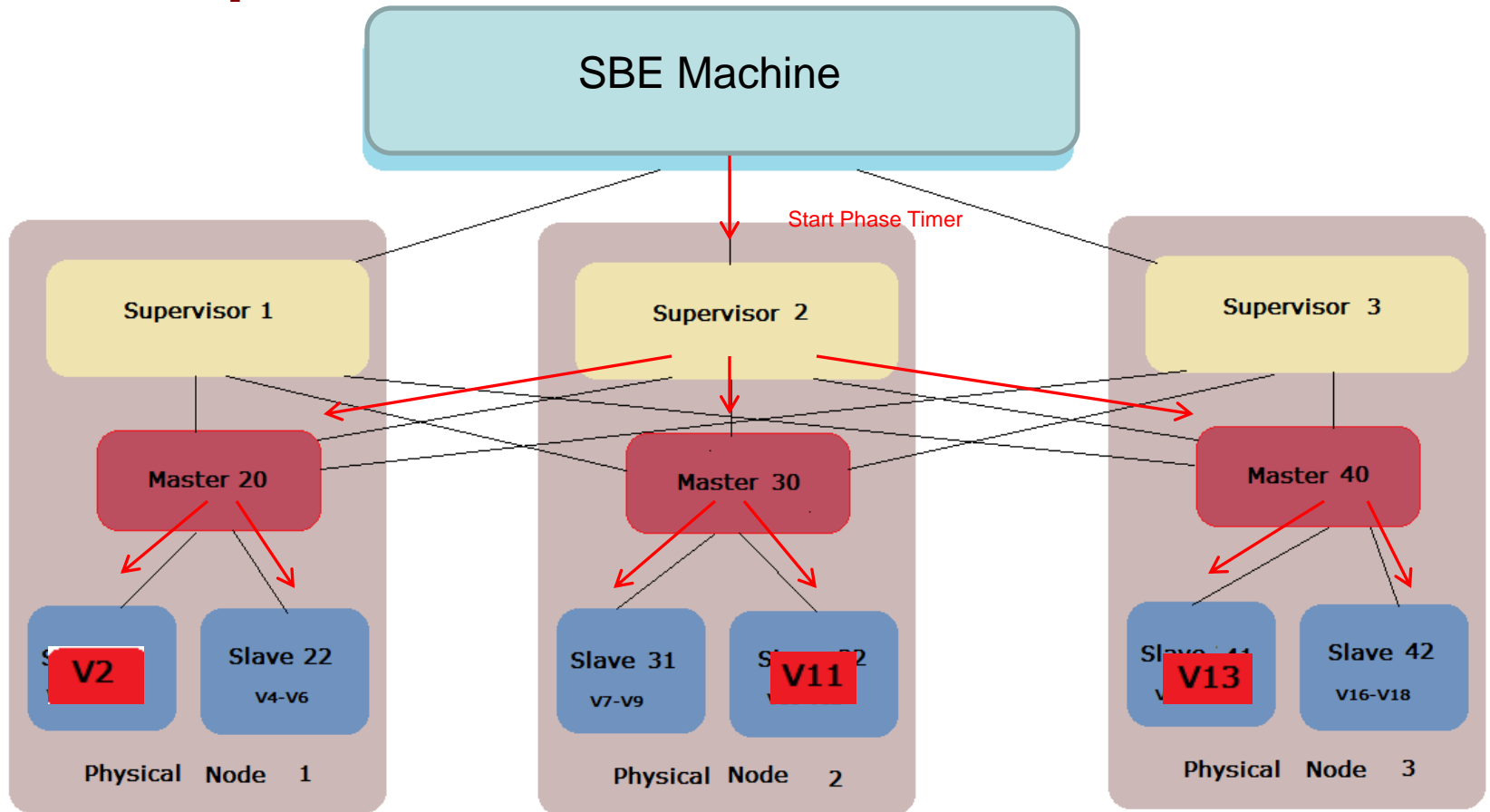
Source:

G. Rodríguez, M. Martín, P. González, J. Touriño and R. Doallo, "CPPC: A compiler-assisted tool for portable checkpointing of message-passing applications," *Concurrency and Computation: Practice & Experience*, vol. 22, no. 6, pp. 749-766, 2010.

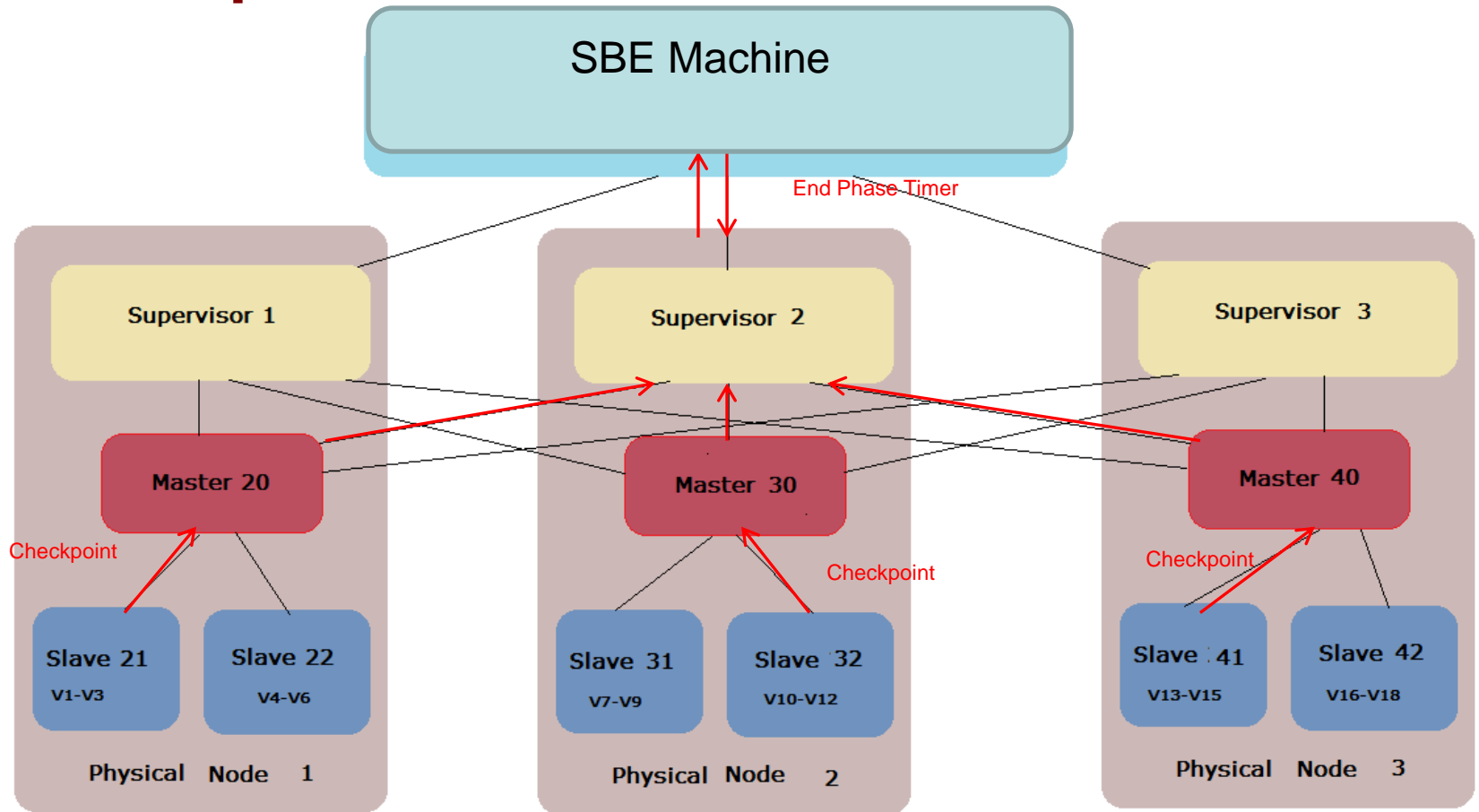
Application 2 - Jacobi's Iterative Linear Equation Solver



Application 2 - Flowchart for each phase



Application 2 - Flowchart for each phase



Application 2 - Overhead






Execution Time in seconds without MTD	Execution time with MTD in seconds		
	2 phases	3 phases	4 phases
200	218	248	276
400	430	486	510
600	642	678	729
800	838	890	988
1001	1082	1114	1122

Execution Time in seconds without MTD	Overhead Percentage (Time)		
	2 phases	3 phases	4 phases
200	8%	19%	28%
400	7%	18%	22%
600	7%	12%	18%
800	5%	11%	24%
1001	8%	11%	12%

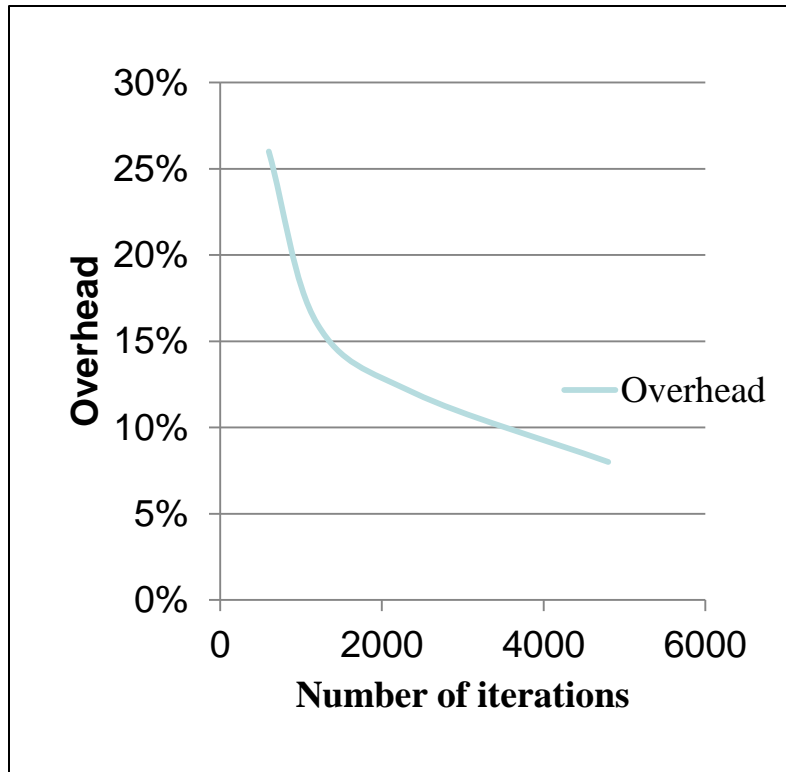
Application 2 - Overhead

Execution Time in seconds without SBE	Execution time with SBE in seconds					
	2 phases		3 phases		4 phases	
	Time	OH	Time	OH	Time	OH
200	218	9%	248	24%	276	38%
800	838	5%	890	11%	988	24%
1500	1568	5%	1624	8%	1663	11%
3600	3671	2%	3847	7%	3890	8%

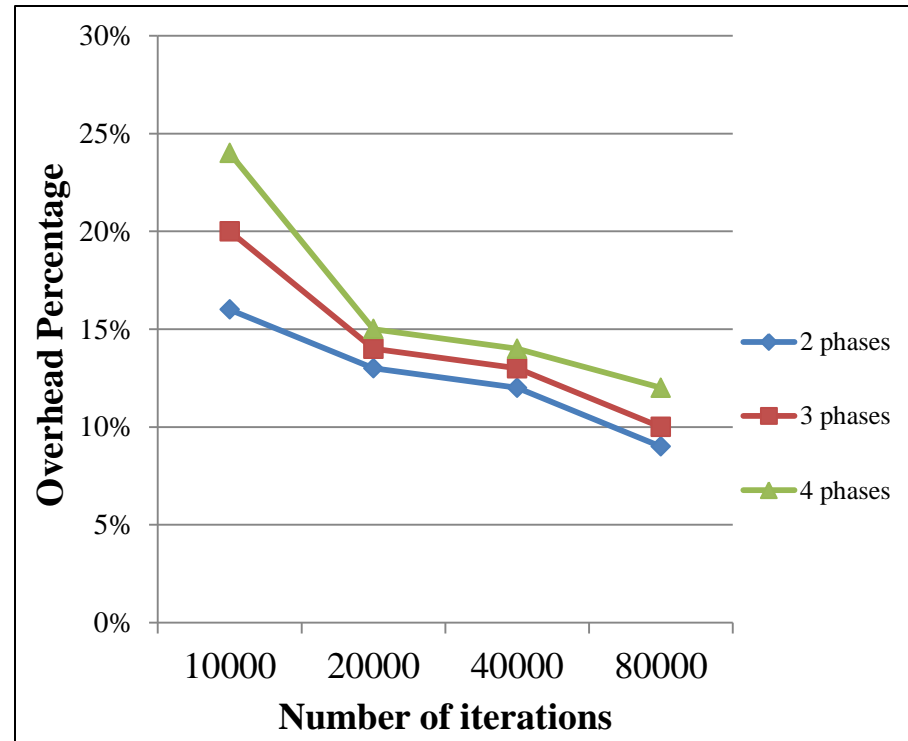
Application 3 – MiBench

-  C programs from six categories
-  Each category targets a specific area of the embedded market
-  Programs used for testing
 - Basicmath (Automotive and Industrial category)
 - Dijkstra's algorithm (Network category)
-  Setup is the same as Application 2
-  Diversity in the form of operating systems

Application 3 - Overhead





Basicmath

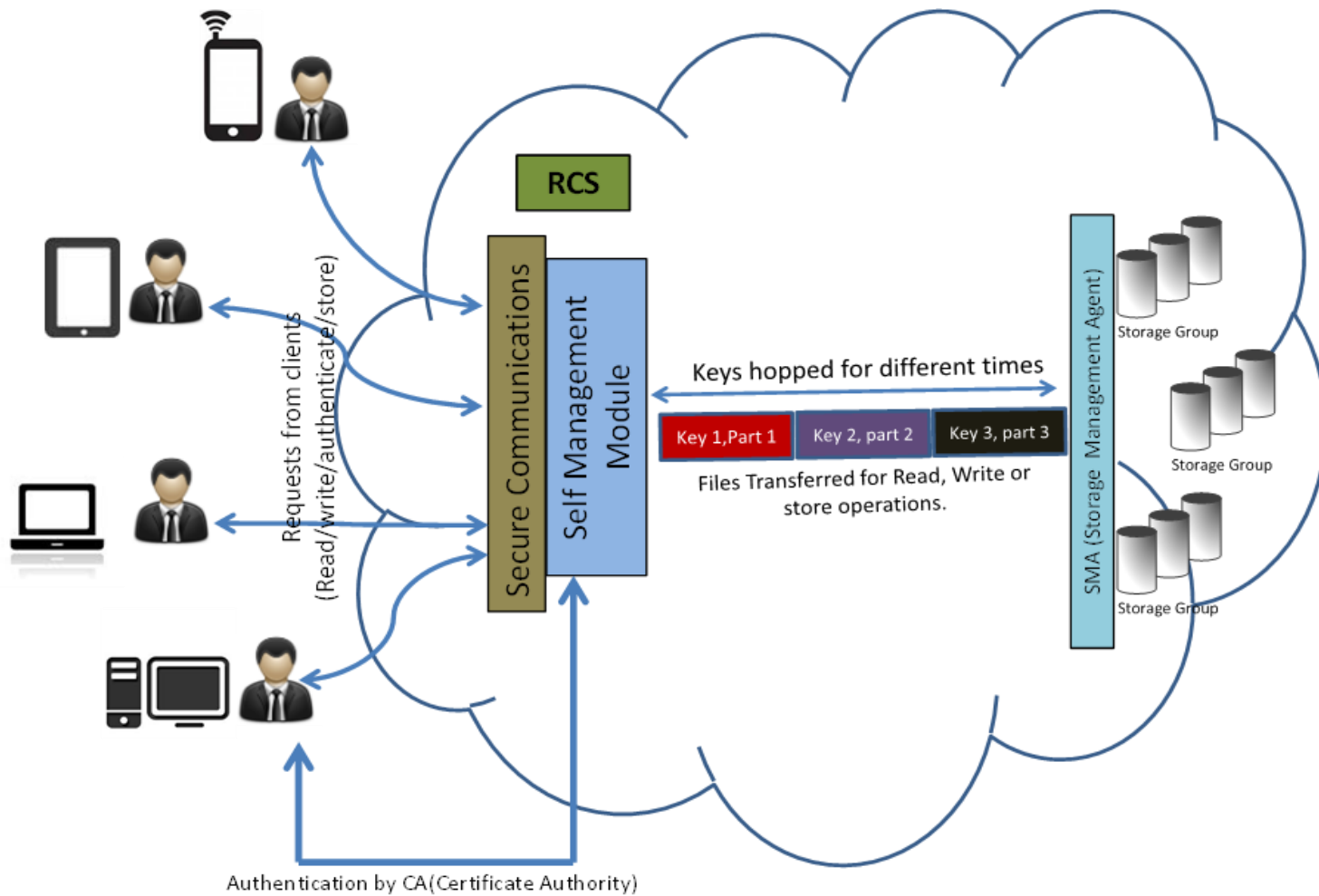


Dijkstra's algorithm

Resilient Cloud Data Storage Services-Architecture




Storage Dynamic Encryption

-  A resilient approach to secure the communications between client and storage server using Key Hopping technique with file partitions.
-  By using shorter keys and hopping them in time we can achieve better performance and security than that traditional method that uses a long key with no hopping.








Architecture

Components

-  Secure Communications
-  Access Control
-  Self-Management

Main functions

-  Key Generation
-  Key Distribution to clients
-  File Partitioning
-  Key Hopping
-  Encryption and Decryption

SDE Approach

$SDE\{\{FilePartA, DESkey1 + B, 2 + C, 3\}, RSAKey1\} \rightarrow TimeWindowA$
 $SDE\{\{FilePartA, DESkey1+B, 2+C, 3\}, RSAkey2\} \rightarrow TimeWindowB$





Long key for long time
→ Security Breach + less Performance
Short Key + Hopping
→ Resiliency + High Performance

- Diverse Keys
- Diverse File Parts
- Diverse Layers of Authentication

Secure Communications

- 🐱 Client – Server communication initiated by **SMM** (Self-Management Module) with **DH key exchange protocol**
- 🐱 Key is generated between SMM and SMA (Storage Management Agent)
- 🐱 It is then distributed to client after **certificate verification** (useful to avoid Man-In-The-Middle attack)
- 🐱 The access control list is then updated and communication starts between the client and cloud system.
- 🐱 The key generated once will be valid only for that particular time window and whenever the key time expires the SMM (Self-Management Module) will again launch the DH key protocol.




Access Control

-  The cloud service provider should limit the access of user account to only authorized users.
-  So in RCDS the authorization of the client is verified using CA certificate authority and is then added permanently to the access control list.
-  If any client that tried to request the server and failed to prove its authenticity, it is added to the block list by SMM and is unlisted only after it proves its authenticity.
-  Self-signed CA certificates are generated and are checked before key distribution.

Key Hopping






- 🐾 Using the same key for a long time is not secure and incurs high overhead. To overcome this problem, we use shorter keys to reduce the time it takes to encrypt data, but change them randomly as it is done in frequency hopping in order to increase the security of the storage service.
- 🐾 The SM module keeps track of the time window and triggers the client and the server at the starting of the time window and when the time window ends. Thus the client and server follow the time window provided by the SM module
- 🐾 Once the time window ends, the keys that are used during that period will expire and SM initiates the generation of keys and distribution of them to various Storage Management Agents.

File Partitioning:







-  In May 2011, a popular file sharing service Dropbox was accused in a complaint to the Federal Trade Commission of using “a single encryption key for all the user data the company stores.”
-  The concern is that if a hacker was able to break into Dropbox’s servers and obtain the key, it could gain access to all of the Dropbox’s user data.
-  So to improve the resiliency of stored data, it is important to partition data into several parts and use different keys for each data partition.

Experimental Results


RCSS approach:

-  DH key exchange protocol
-  Self-Signed CA certificates
-  OpenSSL library – C programmed
-  DES(Data Encryption Standard) in CFB mode (Cipher Feedback Mode) for Data encryption
-  RSA algorithm for encrypting DES keys

Experimental Environment

-  IBM BladeCenter HS22 based Private Cloud
-  University of Arizona's Autonomic Computing Lab
-  Evaluated on a three node cluster
-  Programmed in C
-  OpenSSL library
-  Certificates programmed in OpenSSL library

Performance Improvement Factor

-  To quantify the performance gain from using key hopping, we introduce the Performance Improvement Factor (PIF), which can be computed as:

$$PIF = \frac{RT_{ssl} - RT_{RCS}}{RT_{ssl}}$$

RT_{ssl} = No of sessions * Time taken for SSL protocol

RT_{RCS} = (TDH protocol + Tkeydistribution)*No of hops + (No of sessions * Time for DESen+decryption)


Where,

RT_{ssl} is the response time for system only with SSL.

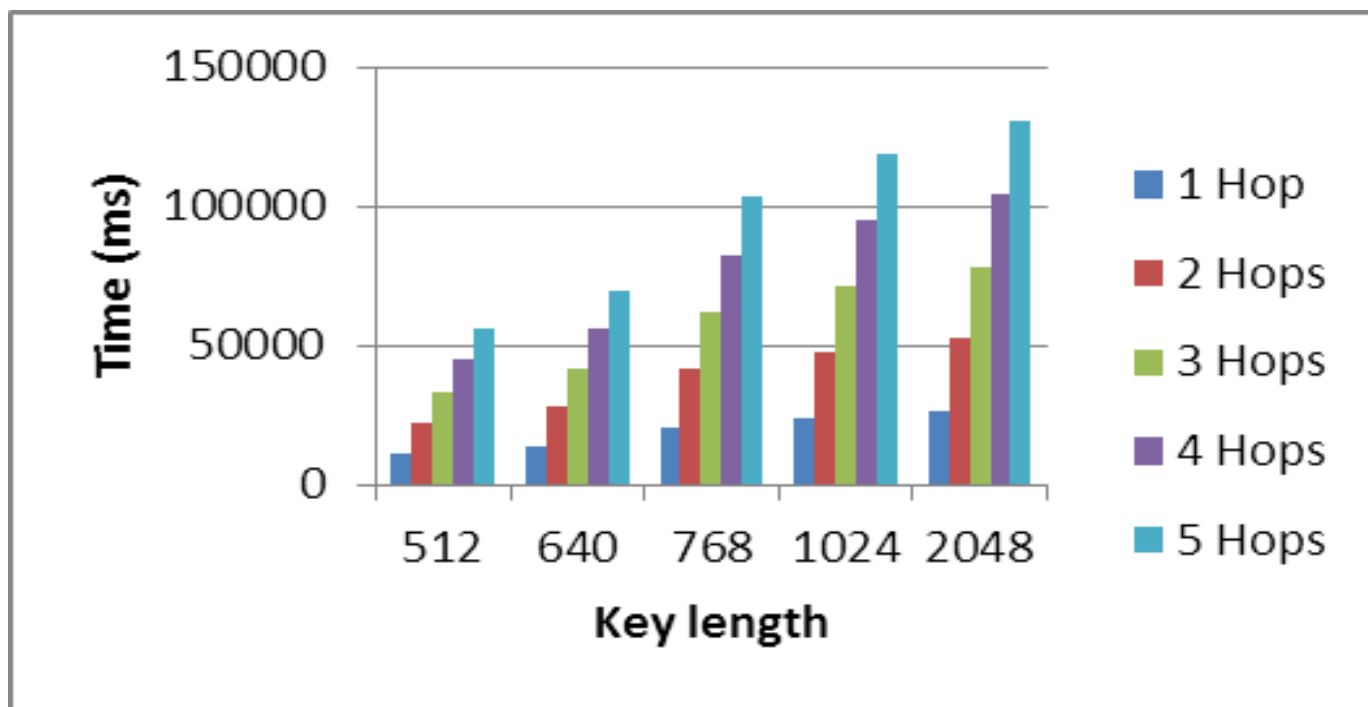
RT_{RCS} is the response time for the system with RCS implementation.

TDH protocol is time of execution for DH protocol.

Tkeydistribution is the time taken for client key distribution.

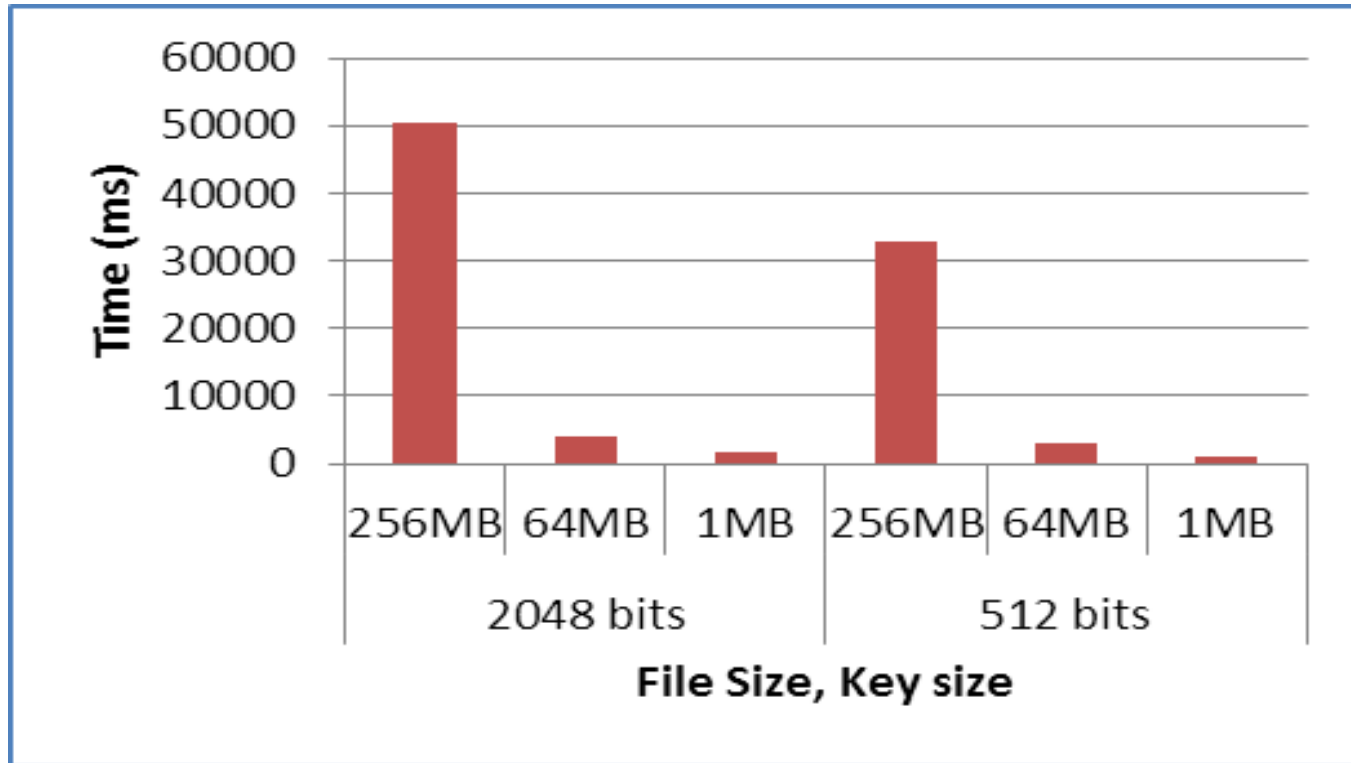
-  Based on our assumption in terms of number of sessions, the performance improvement (PIF) is at 74%.

Results:



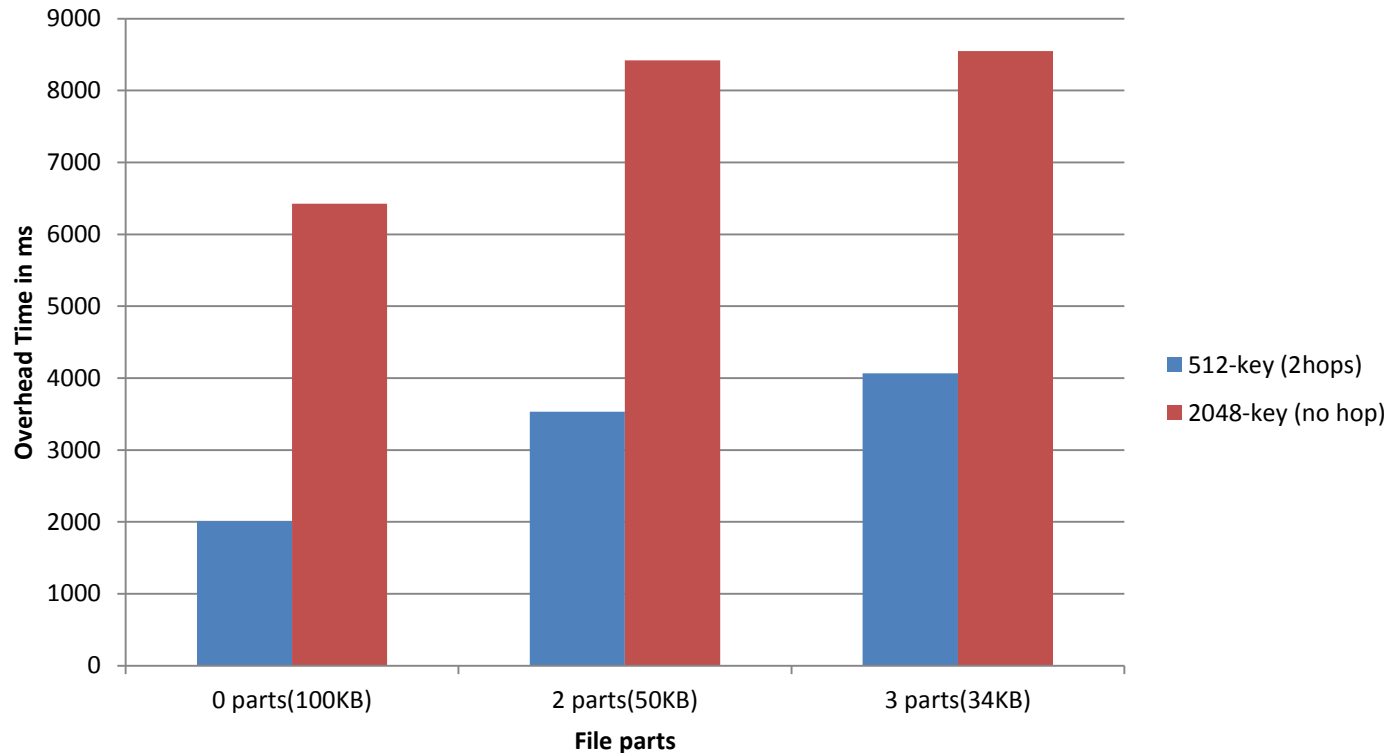
Performance overhead Vs (Key size and number of Hops)

File Partitioning Results:






File Size Vs overhead time for different keys.

Results 2:



File parts Vs overhead time for different keys.

-  What could be the maximum extent in providing the security?
-  Does a normal system requires high resiliency level?
-  How do we recommend users about the security they should take for their system?

Quantification

Quantification of Security and Resiliency



STEP 1: System Self Assessment

- This step is to find the attack vectors of both application and the system by using multiple source code analysis tools like Cppcheck, Flawfinder etc., Checking the vulnerabilities using multiple tools will reduce the error possibilities by collecting all types of vulnerabilities



STEP 2:

- Identifying the attack vectors for applications with respect to the system attack vectors.



STEP 3 Vulnerability Quantification:

- There are many methods specified to estimate the effort required by the attacker to exploit the vulnerabilities in the system
- Here in our approach, we have considered the impact values given by CVE database using CVSS (Common Vulnerability Scoring System)






SBE based Mitigation:

- In SBE the execution time of the application is divided in to phases and multiple operating systems are used in each stage.
- Thus the attacker will have less time to exploit the vulnerabilities. By the time he succeeds exploiting the application is taken for execution on another system.
- Also by using diverse systems which has mutually exclusive attack vectors, the attack surface is reduced by SBE.

Conclusions and Future Work

Conclusions

-  We cannot build perfect cloud security systems
-  RCS architecture can overcome most of the security challenges with less overhead.
-  RCS implements MTD architecture which makes it extremely difficult for an attacker to succeed in attacking the system

Thank You