UNIVERSITY OF CALGARY

SECURE LOGGING: NOTIONS OF SECURITY AND CRYPTOGRAPHIC APPROACHES TO SECURITY

SEPIDEH AVIZHEH

SEPIDEH.AVIZHEH1@UCALGARY.CA
UNIVERSITY OF CALGARY, ALBERTA, CANADA

Logging

- Log: a record of the important events in the system
- Logs are composed of log entries
- Each Log entry contain an event

m1	m2	m3	m4	m5	m6	m7	m8	m9	…

- Applications:
  - Troubleshooting and maintanence
  - Intrusion detection: any set of actions that attempt to compromise the integrity, confidentiality or availability of a resource
  - Digital Forensics: investigation after intrusion is detected

Secure Logging

- logs typically contain computer security-related information
  - adversaries want to stay covert → modify and tamper with the log files without being detected
  - Example: some malwares are specifically designed to alter logs to remove any evidence of their installation or execution
- Goal: Ensure Integrity
  - Alteration
  - Deletion
  - Reordering

Road map

- Forward Integrity
  - Prf-chain MAC (Bellare-Yee)
- Forward-secure stream integrity
  - Aggregate authentication (Ma-Tsudik)
- Crash Integrity
  - SLiC (Blass-Noubir)
- Adaptive Crash Integrity
  - Security definition
  - Impossibility result
  - Double evolving key mechanism
  - Comparison with SLiC
  - Implementation and Evaluation

Logging scheme

- **Gen(.):**
  - Takes security parameter
  - outputs initial state
- **Log(.,.):**
  - Takes the current state and a new event
  - Outputs a new state
- **Recover(.,.):**
  - Takes an initial state or the latest state
  - Reconstructs the longest sequence of events that pass the system integrity checks, or outputs "untrusted log"

Secure Logging through MAC

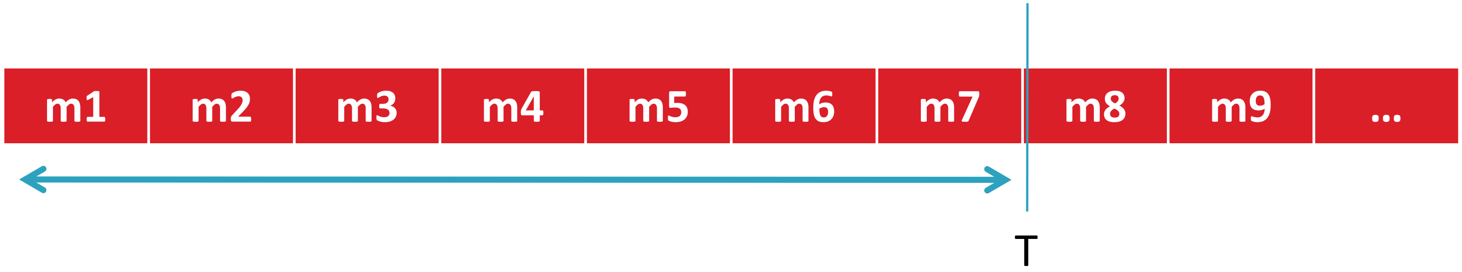
m1|H1	m2|H2	m3|H3	m4|H4	...

$$H_1 = MAC_{K_1}(m_1)$$

- MAC: secure against chosen message attacks
  - HMAC
  - CBC-MAC
- Security relies on the key to be unknown to attacker
  - What about the case that attacker compromises the system?
  - No security will be guaranteed

Forward Integrity

- Attacker compromises the logging device at time T
- Attacker gets access to keys



- Goal: Preserve the integrity of Log entries generated before time T

Forward Integrity

Adversary

1) Issues q events to be logged

Challenger

2) Observes the output of Log()

3) Gets acesses to keys (issues an open request at time T)

4) Keys

•Adversary succeeds if he outputs a false log entry (mj,hj) for an earlier time

Prf –chain Mac (Bellare-Yee)

m1|H1	m2|H2	m3|H3	m4|H4	…

$$H_1 = MAC_{K_1}(m_1) \qquad H_i = MAC_{K_i}(m_i)$$

$$K_1 = PRF_{K_0}(\chi) \rightarrow K_2 = PRF_{K_1}(\chi) \rightarrow \ldots \rightarrow K_i = PRF_{K_{i-1}}(\chi)$$

- K_{i-1} is removed

Truncation atatck

- Attacker may
  - Truncate the log

- Goal: Preserve the integrity of Log files against Truncation

Forward secure stream integrity

- Forward secure sequential aggregate authentication

- Forward security
- Stream security
- Integrity

Forward secure sequential aggregate authentication (Ma-Tsudik)

m1|H1	m2|H2		…

$$H_1 = MAC_{K_1}(m_1)$$
$$T_1 = \mathrm{H}(H_1)$$

$$H_2 = MAC_{K_2}(m_1)$$
$$T_2 = \mathrm{H}(T_1 \mid H_2)$$

$$H_i = MAC_{K_i}(m_i)$$

- Previous Mac is removed from the system

Crash attack Blass-Noubir (CNS' 17)

Operating System (OS)

1) Updates x to x' (in the cache)
2) Stores x'
3) Deletes x

- ~~System crashes~~ before x' is stored

=> System is stateless

Adversary

1) Gets access to the logging device
2) Modifies the log file

   (delete events)

3) Crashes the System

⇒ System is stateless

Normal Crash ≁ **Crash Attack**

Crash Integrity against a non-adaptive attacker

Adversary

- Gen oracle
- Log oracle
- Recover oracle
- Crash oracle

1) Issues log queries for n events →

Challenger

2) Uses Log() on each event

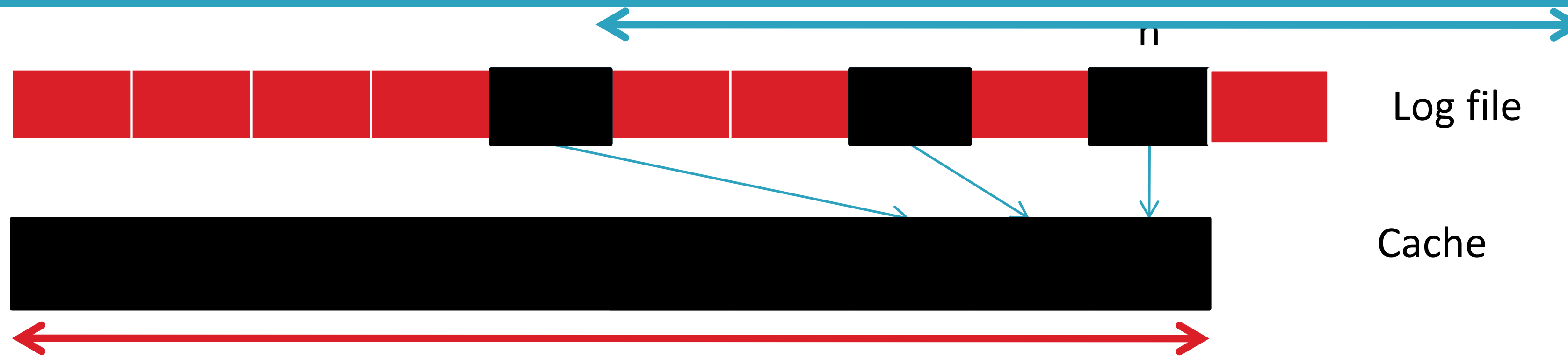
Adversary compromises the device

3) The last state of the log file ←

4) A modified log file, crashed state →

•The goal is to remain undetected
•Adversary succeeds if he can remove/modify an event which is not supposed to be in the cache during the crash (Expendabe set)

Cache



Cache size (cs) = > maximum number of log events that will be lost during a normal crash

- Logging an event generates a set of disk write operations,
  - will add a new entry to the Lstore
  - may update a number of other entries
- If logging device crashes before Log(.,.) completes, all write operations created by Log(.,.) will be lost.
- we consider 2cs events (the interval [n- cs+1, n+cs]) as expendable set

SLiC

$$(c_i, H_i, k_i)$$

$$c_i = Enc_{K_i}(m_i)$$

$$H_i = MAC_{K_i}(Enc_{K_i}(m_i))$$

$$\kappa_i = PRF_{K_i}(i)$$

$$K_i = PRF_{K_{i-1}}(\chi)$$

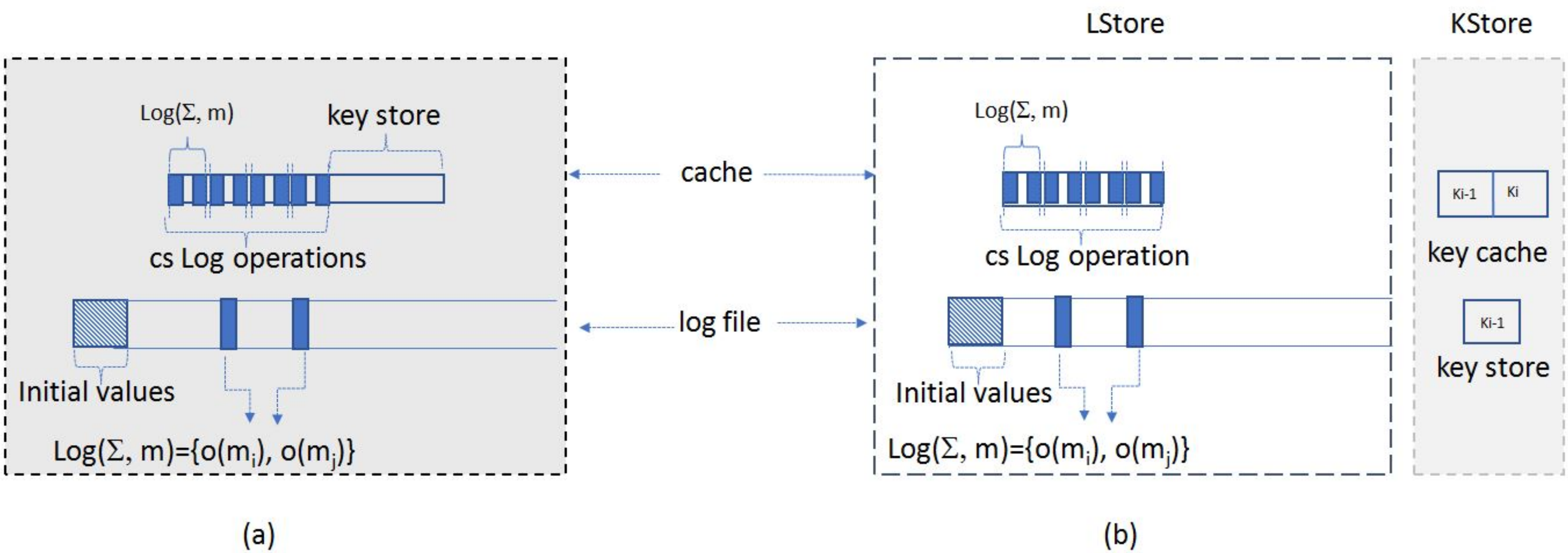
Adaptive crash attack

- An Insider adversary who can observe the log file during the log operation
- Adversary compromises the device
  - can rewind the system to a past state
- Non of the existing schemes are secure in this model

s1	s2			
s3	s2	s1		
s3	s2	s4	s1	
s3	s2	s5	s1	s4
1	**2**	**3**	**4**	**5**

System model

- **Logging device:**
  - runs Gen(.) and Log(.,.)

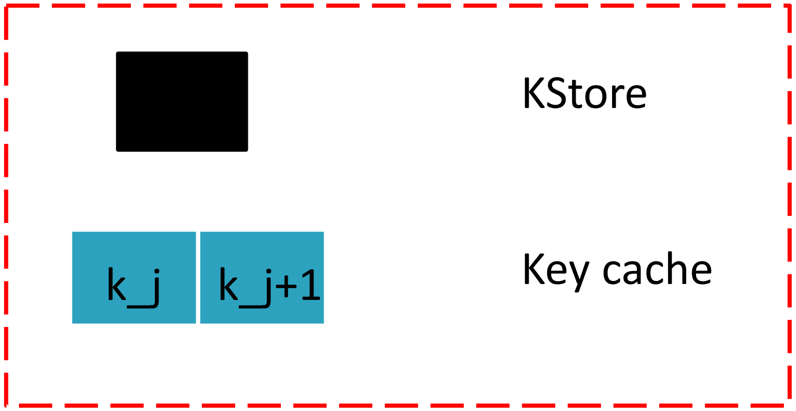


Non-adaptive crash attack

Adaptive crash attack

Key Cache

- The log operation will also update keys
- We assume the KStore stores the key, k_j, which is used in constructing o(m_j) only



- If crash happens, k_j that is being updated will also become unreliable.

Crash Integrity against a non-adaptive attacker

Adversary

Challenger

- Gen oracle
- Log oracle
- Recover oracle
- Crash oracle

n times

1) Issues log queries

2) Uses Log() on each event

3) Observes the state of the Lstore and its cache

4) The last state of the Kstore and its cache

5) A modified log file, crashed state

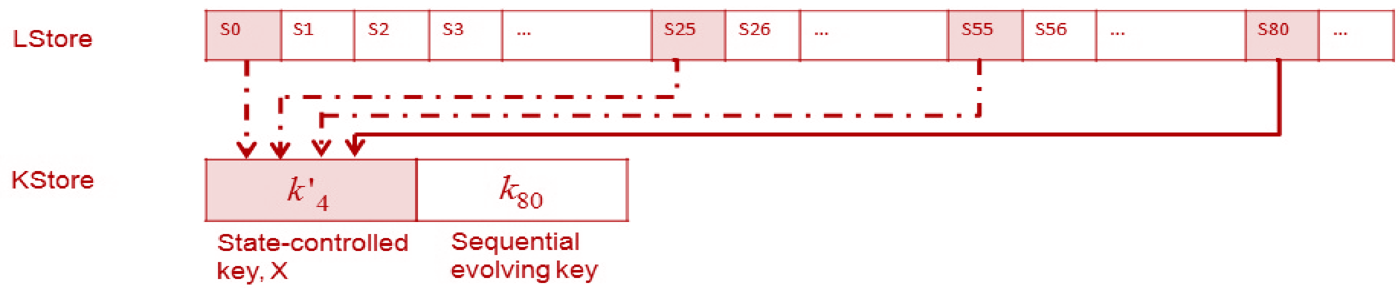
- **The goal is to remain undetected**
- **Adversary succeeds if he can remove/modify an event which is not supposed to be in the expendable set**

Impossibility Result

- All existing schemes are vulnerable to adaptive crash attack
  - Even considering a protected KStore according to our model
  - KStore can be undetectably removed or modified when the system is compromised
- A logging system that cannot reliably protect its state information during logging operation and assuming an adaptive adversary who can see the LStore, is subjective to rewinding

Logging scheme

- Double evolving key mechanism
  - Use two key sequences evolve with different rate
  - State controlled key: updated with probability $\frac{1}{m}$ through the result of a choice function $CF(): H(k'_{j-1}, i) < T$

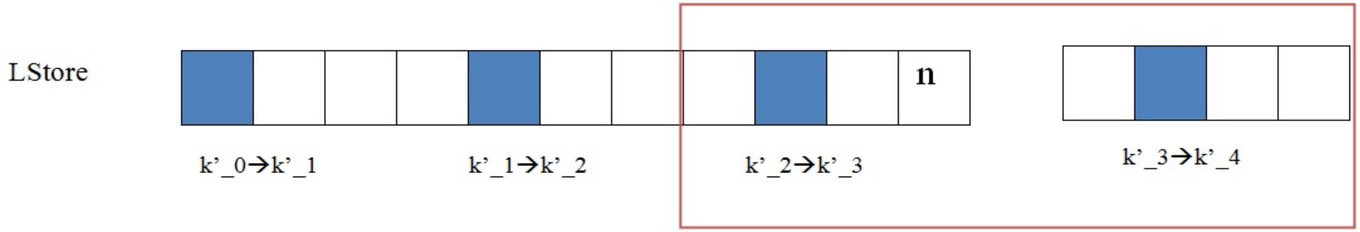


Security (informally)

- The double evolving key mechanism is $\frac{\alpha^2}{m}$ stable
  - α is the probability of a removal in a normal crash
  - if the choice function CF() outputs 1 with probability $\frac{1}{m}$
  - the probability that the key is removed by a normal crash is $\frac{\alpha^2}{m}$
- Use two (or more) independent state-controlled keys
  - different PRFs
  - evolves at different rates
  - probability that all keys are missing will be reduced to a greater extent

Recovery

- Generate the keys
  - All sequential and state controlled keys
  - For evolving state controlled keys we check CF()
- Compute expendable set
  - Captures the LStore entries that are considered unreliable when a crash happens
- Determine the set of all possible keys that may reside in the Kstore during crash



- Output R or "untrusted log"

Acheives Crash Integrity against adaptive attacker

Complexity analysis

- Advantages:
  - our scheme is faster
  - Each log operation in our scheme requires one write operation on disk whereas in SLiC requires two write operations
  - The order of events is preserved in the log file

Algorithm/scheme	Our scheme	SLiC	$SLiC^{Opt}$
Log(.,.)	O(1)	O(1)	O(1)
Recover(.,.)	O(n’)	O(n’log(n’))	O(n’)

n’: number of events

Implementation

- --Windows computer with 3.6 GHz Intel(R) Core(TM) i7-7700 CPU
- --Raspberry Pi 3, Model B with 600 MHz ARM CPU running Raspbian

Logging performance
(total time in seconds)

- # events: 2^{20}

Hardware	Scheme	Exp 1	Exp 2	Exp 3	Exp 4	Exp 5
Windows PC	Our scheme	40.2	40.2	40.4	40.7	40.5
	SLiC	95.2	96.0	95.2	95.4	96.0
	Plain	2.0	2.0	2.0	2.0	2.0
Raspberry Pi 3	Our scheme	330.5	325.4	319.0	324.5	319.6
	SLiC	790.2	792.0	777.9	789.2	796.8
	Plain	18.8	18.7	18.8	19.0	18.9

Conclusion

- We reviewed exsisting notions of secure logging
- We inroduced adaptive crash attack
  - adversary can rewind the system back to one of the past states
- We showed that this attack is strictly stronger than non-adaptive crash attack
  - all existing schemes are subjective to this attack
- We also proposed double evolving key mechanism

Future works

- Ensuring crash integrity against an adaptive attacker without considering a protected memory for keys
- We observed that
  - By using uniform distribution for double evolving key mechanism, adversary can succeed with less probability
- Finding the best probability distribution for evolving the key that it minimizes the success probability of the attacker

Thank you!