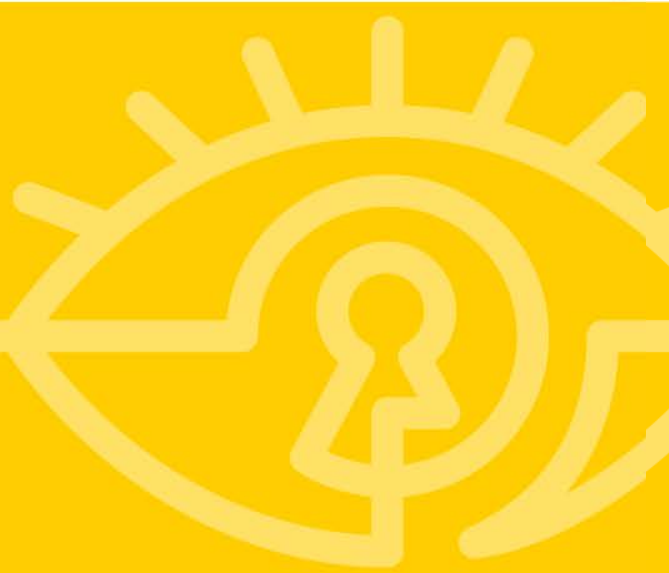
RSA Conference 2016
San Francisco | February 29 – March 4 | Moscone Center

Connect to Protect

SESSION ID: CRYP-W04

Efficient Culpably Sound NIZK Shuffle Argument without Random Oracles

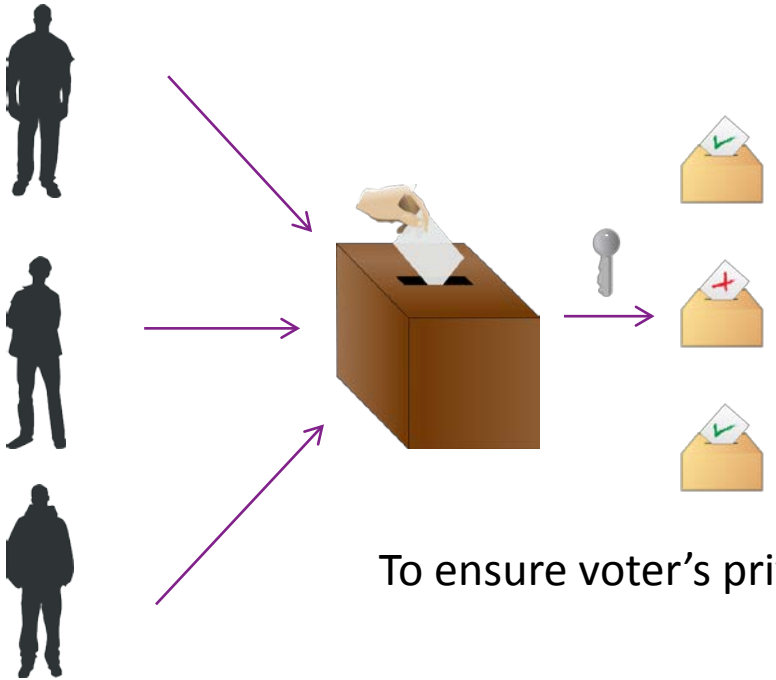
Prastudy Fauzi
University of Tartu
Joint work with Helger Lipmaa



Motivation: Electronic Voting

Motivation: Electronic Voting

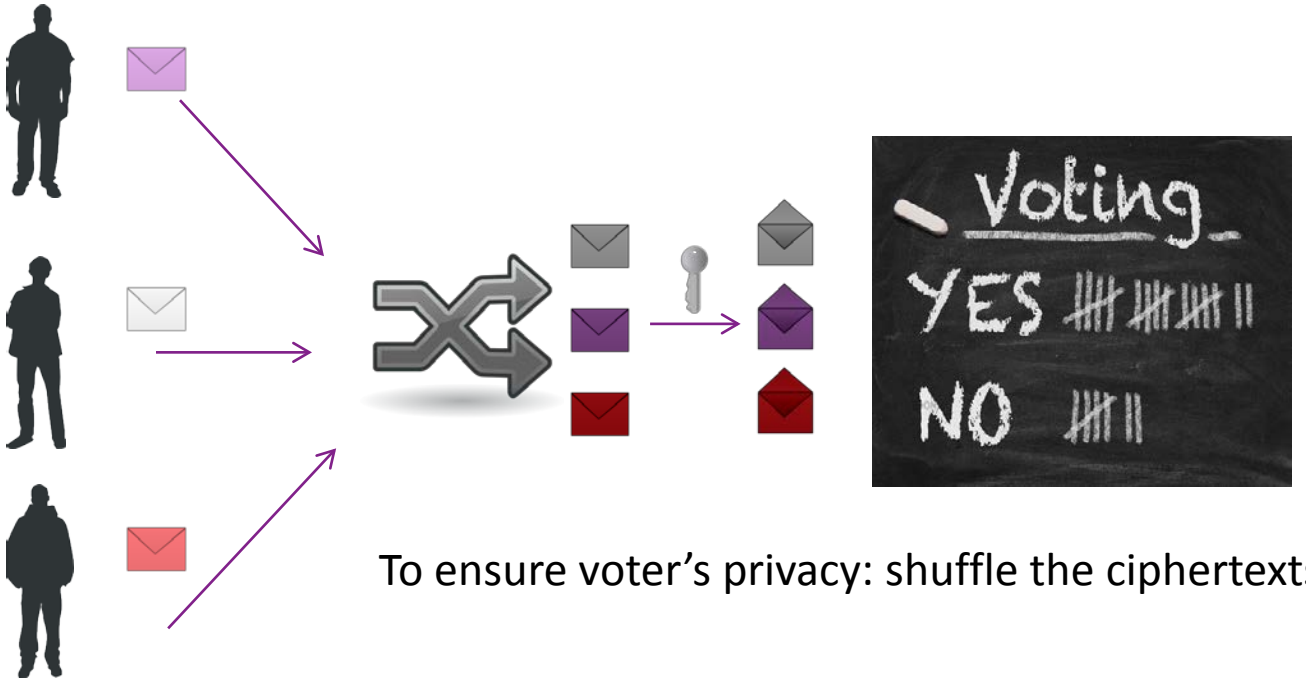
- Traditional voting



To ensure voter's privacy: shake the ballot box

Motivation: Electronic Voting

- Electronic version



To ensure voter's privacy: shuffle the ciphertexts

Motivation: Electronic Voting

- The shuffle is done by a **mix-server**

Original

$C = (c_1, c_2, c_3)$

Permutation

$\psi = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \end{pmatrix}$

Shuffled

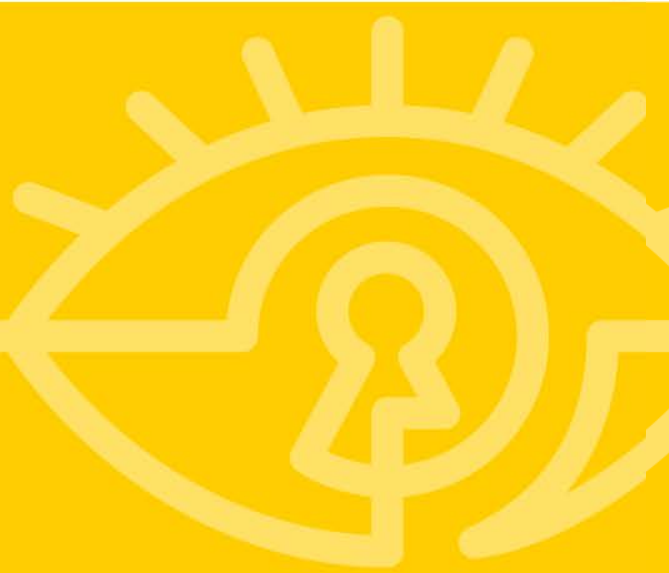
$C' = (c'_1, c'_2, c'_3)$

$$c'_i = c_{\psi(i)} \cdot enc(1; t_i)$$

- Shuffled ciphertexts must be **re-randomized** to hide ψ

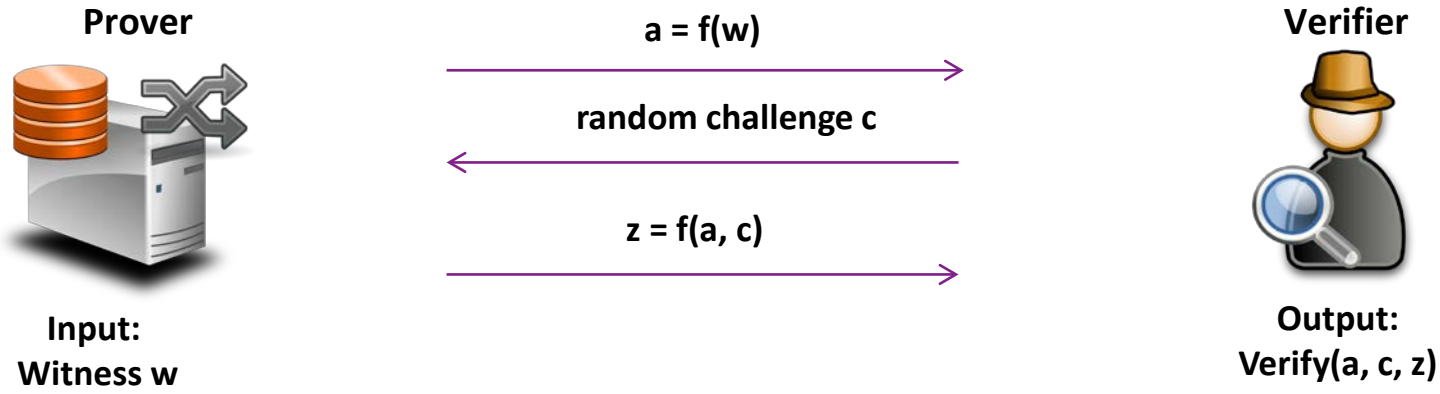
Motivation: Electronic Voting

- Issues to address:
  - You don’t trust the mix-server
    - Mix-server must provide proof that shuffle is done correctly
    - Proof can be verified by anyone, at any time
  - The mix-server doesn’t trust you
    - Proof must not reveal the permutation to any verifier



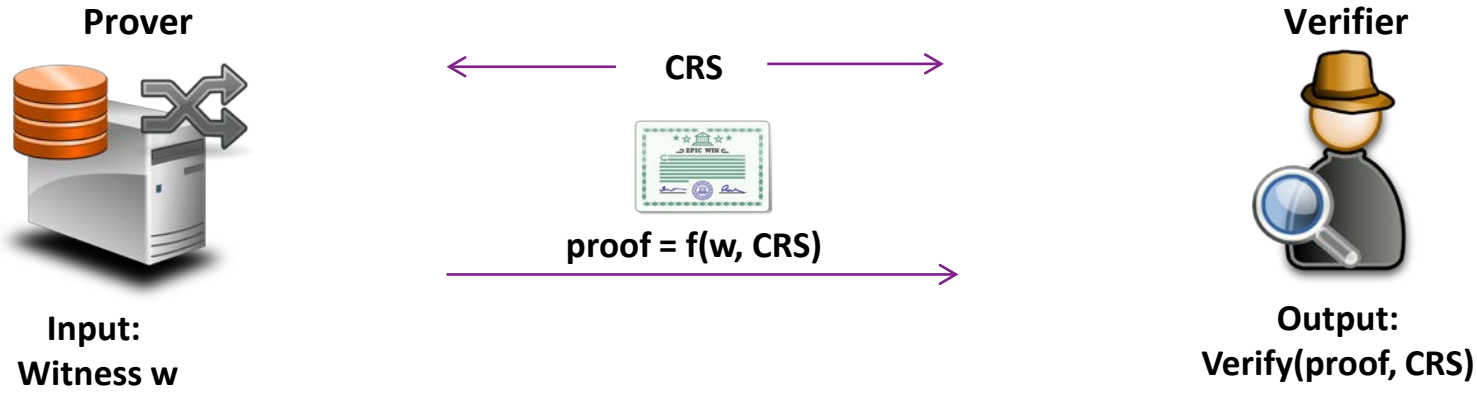
How to Prove?
How to Verify a Proof?

Interactive Zero-Knowledge Proof

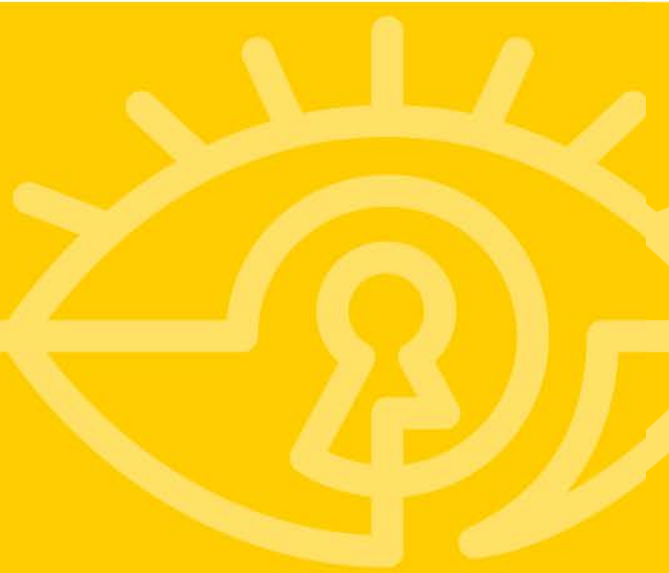


- **Complete**: If Prover is honest, Verifier should always accept
- **Sound**: If Prover is not honest, Verifier should reject proof
- **Zero-knowledge**: Verifier does not learn anything about the permutation from valid proof

Non-Interactive Zero-Knowledge



- Verifier's challenge c is incorporated into a (randomly generated) **reference** string
- This is called the **common reference string** (CRS) model
- Alternative is the **random oracle** (RO) model (challenge c is replaced by a hash function)



Previous Work on NIZK Shuffle

Previous Work on NIZK Shuffle

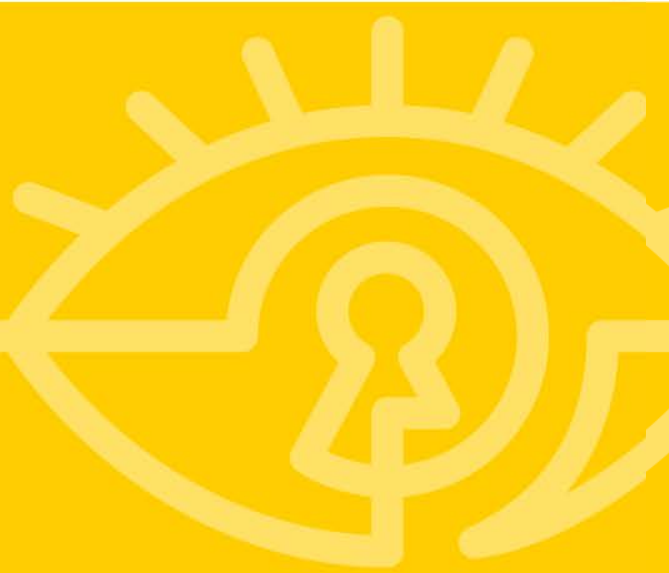
	[GL07]	[LZ13]	Daylight	[Gro10a]
|CRS|	2n + 8	7n + 6		n + 1
Communication	15n + 120 (+3n)	6n + 11 (+6n)		3n
Prover's comp.	51n + 246	22n + 11 (+6n)		6n (+2n)
Verifier's comp.	75n + 282	28n + 18		6n exp.
Model	CRS	CRS		RO + CRS

- |CRS|: number of group elements in the CRS
- Communication: number of group elements sent to the verifier
- Prover's comp.: number of exponentiations
- Verifier's comp.: number of pairings

Previous Work on NIZK Shuffle

	[GL07]	[LZ13]	?	[Gro10a]
Soundness	Culpably sound	White-box sound		Sound
Arg. of knowl.	No	Yes		Yes
PKE (knowledge assumption)	No	Yes		No

- Culpably sound: successful adversary + secret key can break security assumptions
  - Still acceptable, can assume *some* coalition of parties know secret key
- White-box sound: successful adversary + random coins of all encrypters (e.g. voters) can break security assumptions
  - Weaker notion, can not only extract plaintexts, but also randomizers



Our Results

NIZK Shuffle Argument: Overview

- Common input:
  - $C = (c_1, c_2, c_3, \cdots, c_n)$: set of ciphertexts before shuffle
  - $C' = (c'_1, c'_2, c'_3, \cdots, c'_n)$: set of ciphertexts after shuffle
- Private input: permutation ψ, or equivalently, permutation matrix P_ψ

Main idea:

1. Prove that there exists a permutation matrix P_ψ such that $\mathrm{P}_\psi M = M'$, where $M = dec(C),\ M' = dec(C')$
2. Find verification equations that hold only for honestly generated C, C'

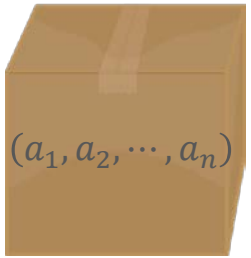
Building Blocks

- Commit to n-tuples using the polynomial commitment scheme [Gro10b, Lip12]

$(a_1, a_2, \cdots, a_n) \in \mathbb{Z}_p^n$

$ck = \left\{(g_1, g_2^{\gamma})^{y_i(\chi)}\right\}_{i=0}^{n}$

commit



$\in \mathbb{G}_1 \times \mathbb{G}_2$

- Bilinear maps, $\hat{e}: \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$
- Elgamal encryption, $enc(m; r) = (g^r, m\, h^r)$

Step 1: Permutation Matrix Argument

P_ψ is permutation matrix iff:

- Every row (transposed) is a unit vector

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

- Every column sums to 1

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Step 1: Permutation Matrix Argument

Permutation Matrix argument for $\mathrm{P} = \mathrm{P}_{\psi}$ is as follows:

A. Commit to every row of P
   - Choose randomness s.t. $\sum_{i=1}^{n} r_i = 0$

B. For each committed row P_i and its commitment:
   - Prove it is a valid commitment to a unit vector

C. It can trivially checked that $\prod_{i=1}^{n} com(P_i; r_i) = com(\vec{1}_n; 0)$

Step 1B: Unit Vector Argument

- $\vec{a}$ is unit vector iff every element is Boolean and sum to 1.

  This holds iff $\mathrm{V}\vec{a} + b \in \{0,2\}^{n+1}$, i.e.

  $$(\mathrm{V}\vec{a} + b - \vec{1}_{n+1}) \circ (\mathrm{V}\vec{a} + b - \vec{1}_{n+1}) = \vec{1}_{n+1},$$

  where $\mathrm{V} = \begin{pmatrix} 2I_{n\times n} \\ \vec{1}_n^T \end{pmatrix}$, $\mathrm{b} = \begin{pmatrix} \vec{0}_n \\ 1 \end{pmatrix}$

- Use square span programs (SSP), similar to [DFGK14]

Step 2: Finding the Equations

- Assume $c_i' = c_{\psi(i)}$. Consider the tautology

$$\prod_{i=1}^{n} \hat{e}(c_i', g_2^{\gamma y_i(\chi)}) = \prod_{i=1}^{n} \hat{e}(c_i, g_2^{\gamma y_{\psi^{-1}(i)}(\chi)})$$

- Need privacy: the term $g_2^{\gamma y_{\psi^{-1}(i)}(\chi)}$ reveals $\psi^{-1}(i)$
  - Commit to $\mathrm{P} = \mathrm{P}_{\psi^{-1}}$, replace term by $com(P_i; r_i)_2$
  - Induces extra "error" term $E = f(CRS, \vec{r}, \vec{t})$

Step 2: Finding the Equations

- Not yet sound!
  - Adversary can create "valid" C, C' but $\forall\psi\exists i: dec(c_i') \neq dec(c_{\psi(i)})$
  - Fix: add similar equations using different polynomials $(\hat{y}_i(\chi))_{i=0}^{n}$
- This is finally culpably sound under a new Power Simultaneous Product (PSP) assumption
  - We prove this assumption holds in the generic bilinear group model

Additional Notes

- Permutation matrix must be committed w.r.t.both sets of polynomials $(y_i(X))_{i=0}^{n}, (\hat{y}_i(X))_{i=0}^{n}$
  - Need to prove they commit to same value
- $(y_i(X))_{i=0}^{n}, (\hat{y}_i(X))_{i=0}^{n}$ must be carefully chosen so the conditions of the PSP assumption are true
  - Such a choice exists

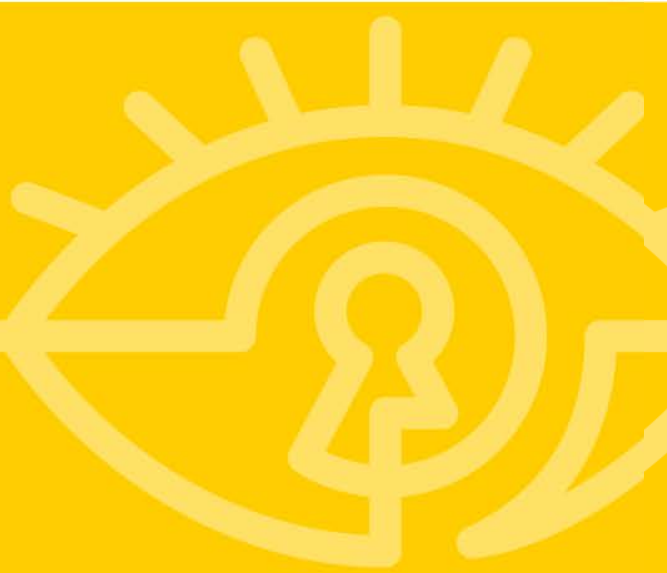
Assumptions:
XDH, PKE, PCDH, TSDH, PSP

Efficiency of NIZK Shuffle

	[GL07]	[LZ13]	This work	[Gro10a]
|CRS|	2n + 8	7n + 6	8n + 17	n + 1
Communication	15n + 120 (+3n)	6n + 11 (+6n)	7n + 2 (+2n)	3n
Prover's comp.	51n + 246	22n + 11 (+6n)	16n + 3 (+2n) [only 2n + 2 online]	6n (+2n)
Verifier's comp.	75n + 282	28n + 18	18 n + 6 [only 8n + 4 online]	6n exp.
Soundness	Culpably sound	White-box sound	Culpably sound	Sound
Arg. of knowl.	No	Yes	Yes	Yes
PKE (knowledge assumption)	No	Yes	Yes	No

Summary

- Created efficient permutation matrix argument
  - From unit vector argument
  - Using techniques from square span programs
- Defined a new assumption (PSP) secure in the (bilinear) GGM
- Used the above to create efficient NIZK shuffle argument
- Future work:
  - Remove large overhead from “calibrating” verification equations
  - Decrease the use of knowledge assumptions



Thank You!

Extended version: http://eprint.iacr.org/2015/1112.
The authors were supported by the EU's Horizon 2020 research and innovation programme under grant agreement No 653497 (project PANORAMIX), and the Estonian Research Council.

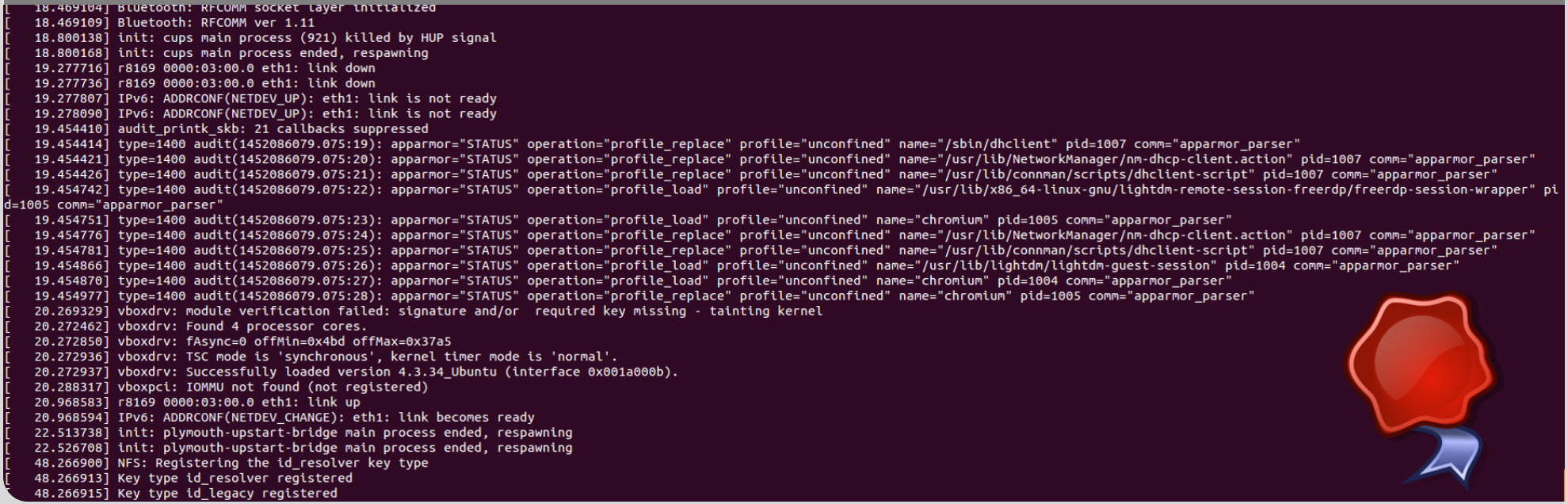
Karlsruhe Institute of Technology

Secure Audit Logs with Verifiable Excerpts

March, 2nd 2016

Gunnar Hartung

INSTITUTE OF THEORETICAL COMPUTER SCIENCE



Outline

(Seal Image Source: CC-0 by OpenIcons)

Introduction

What is Secure Logging?

Securing Log Files against retroactive modifications

Introduction

What is Secure Logging?

Securing Log Files against retroactive modifications

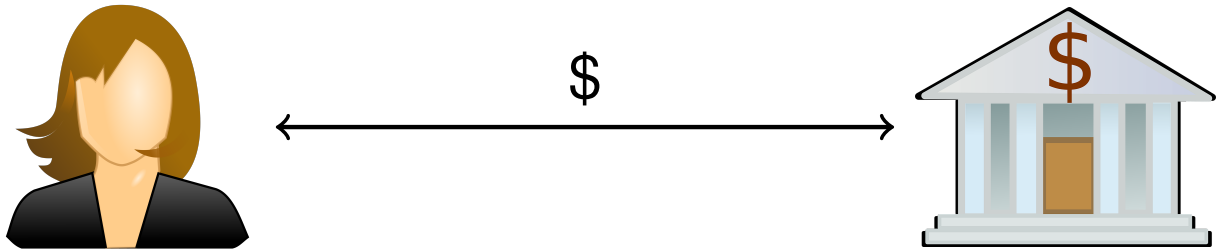
Why care?

- paramount for system debugging/maintenance
- intrusion detection
- forensics after an intrusion
- Attackers cover their traces by editing log files.
- required/recommended by
  - DoD Orange Book [Lat85]
  - NIST Handbook on Computer Security [NIS95]
  - Common Criteria [CC12]

Why Excerpts?

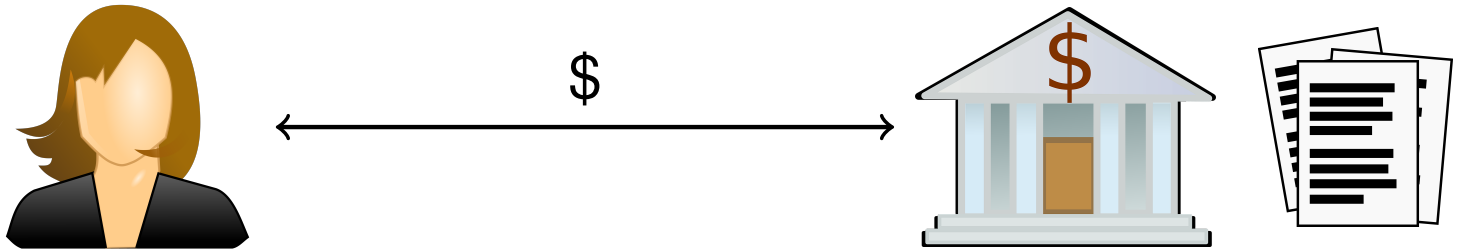
Images: CC-0 by dagobert83, ClkerFreeVectorImages, mireyaqh, sheikh_tuhin

Why Excerpts?



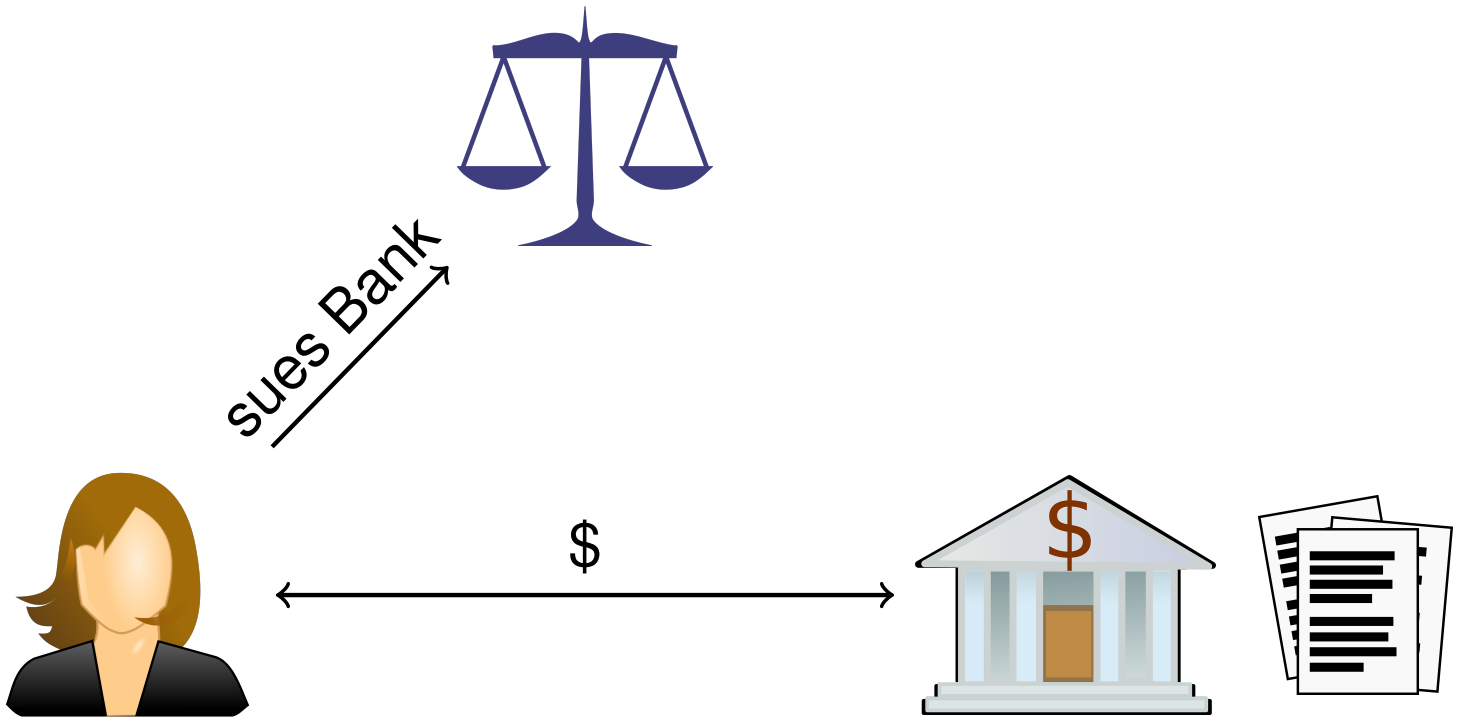
Images: CC-0 by dagobert83, ClkerFreeVectorImages, mireyaqh, sheikh_tuhin

Why Excerpts?



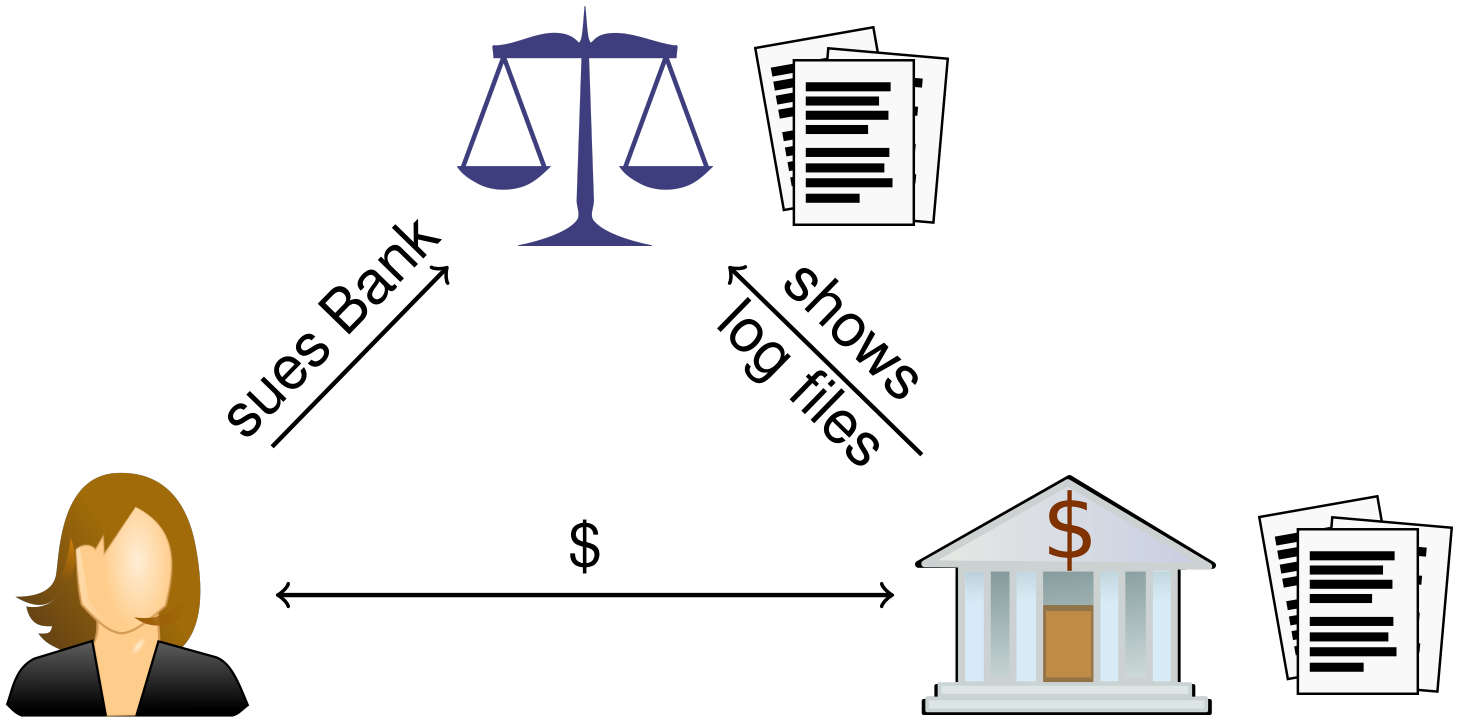
Images: CC-0 by dagobert83, ClkerFreeVectorImages, mireyaqh, sheikh_tuhin

Why Excerpts?



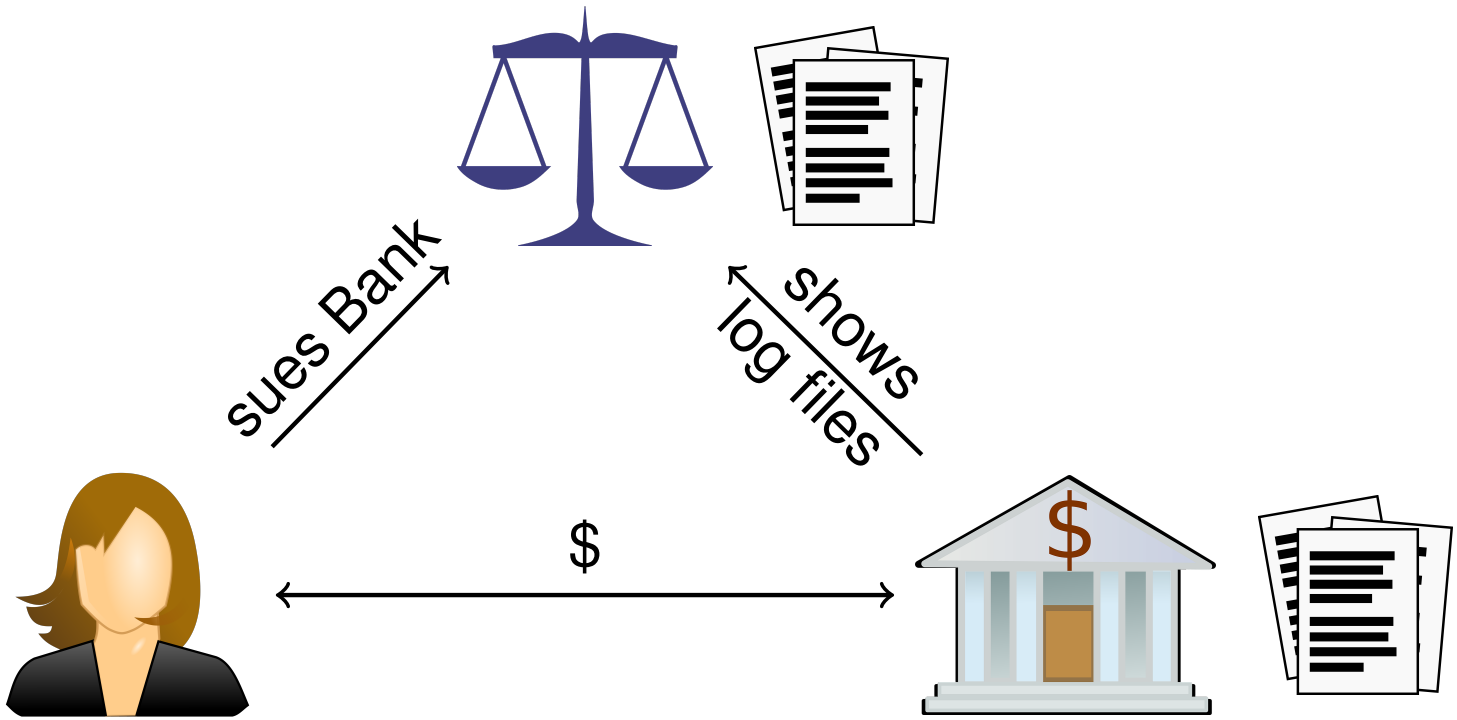
Images: CC-0 by dagobert83, ClkerFreeVectorImages, mireyaqh, sheikh_tuhin

Why Excerpts?



Images: CC-0 by dagobert83, ClkerFreeVectorImages, mireyaqh, sheikh_tuhin

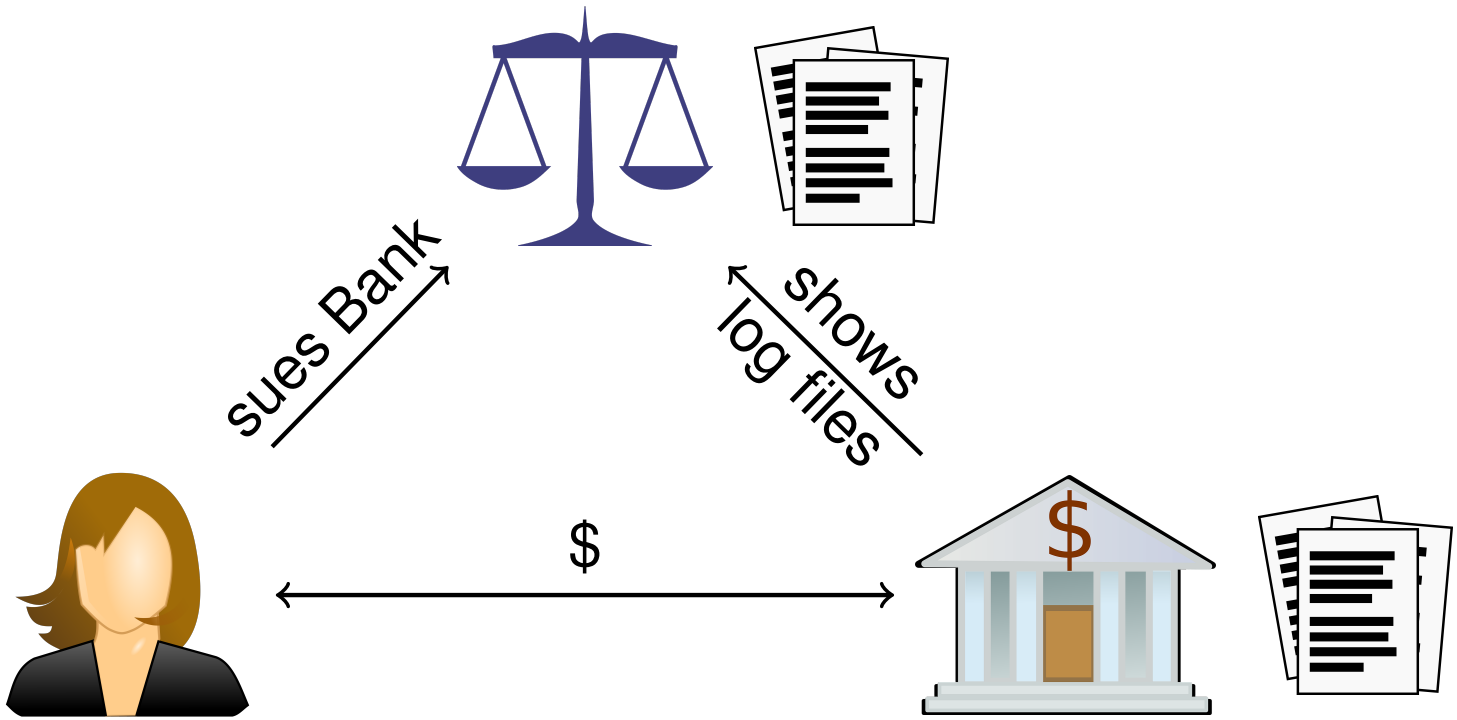
Why Excerpts?



- log files contains lots of confidential information

Images: CC-0 by dagobert83, ClkerFreeVectorImages, mireyaqh, sheikh_tuhin

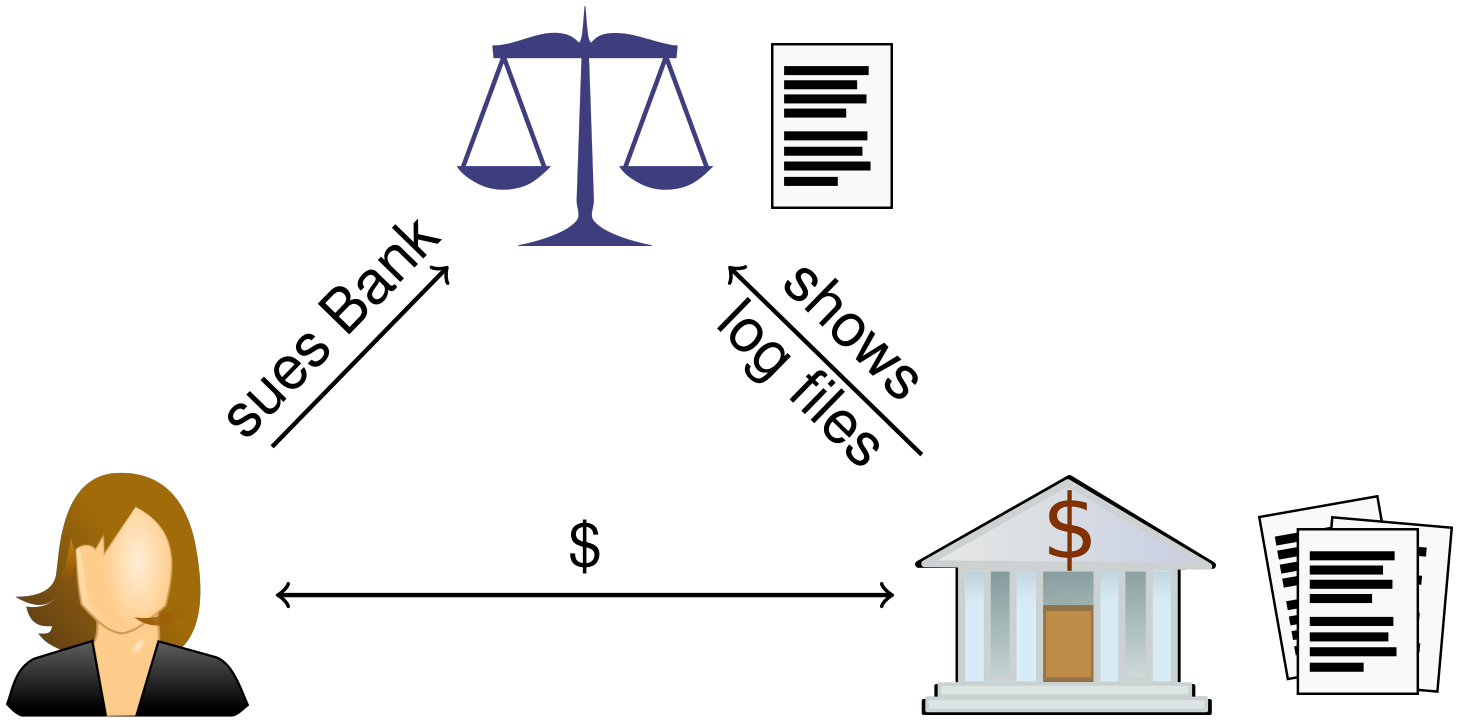
Why Excerpts?



- log files contains lots of confidential information
- very large, hard to analyze

Images: CC-0 by dagobert83, ClkerFreeVectorImages, mireyaqh, sheikh_tuhin

Why Excerpts?



- log files contains lots of confidential information
- very large, hard to analyze

Excerpts solve both problems!

Images: CC-0 by dagobert83, ClkerFreeVectorImages, mireyaqh, sheikh_tuhin

Standard Approaches

WORM Drives:

- 
- Standard drives with custom firmware

Images: CC-BY-2.0 by Till Dettmering, Public Domain via Wikipedia, Ocrho

Standard Approaches

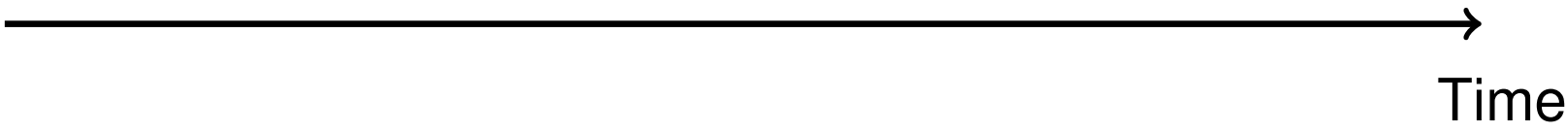
WORM Drives:

- 
- Standard drives with custom firmware

Images: CC-BY-2.0 by Till Dettmering, Public Domain via Wikipedia, Ocrho

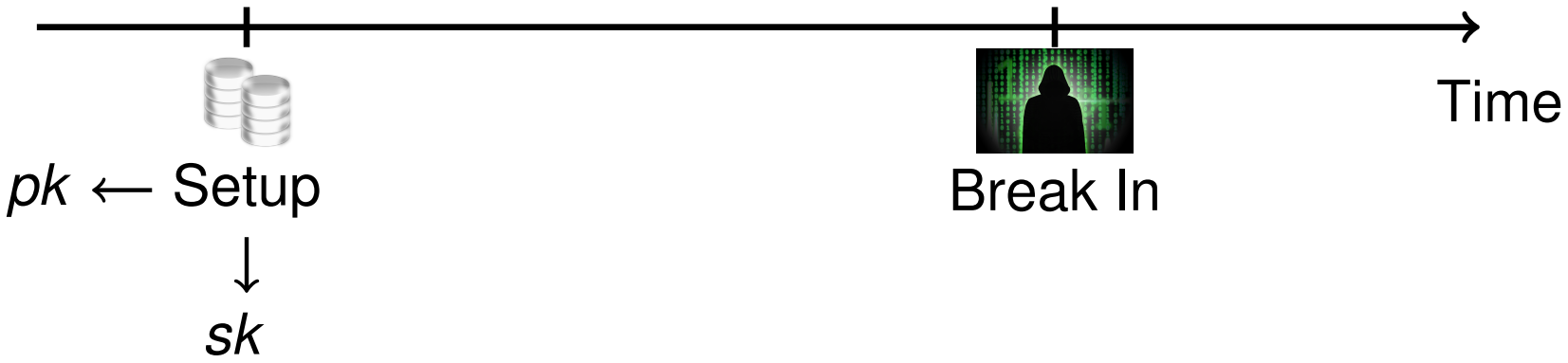
Crypto!

Model



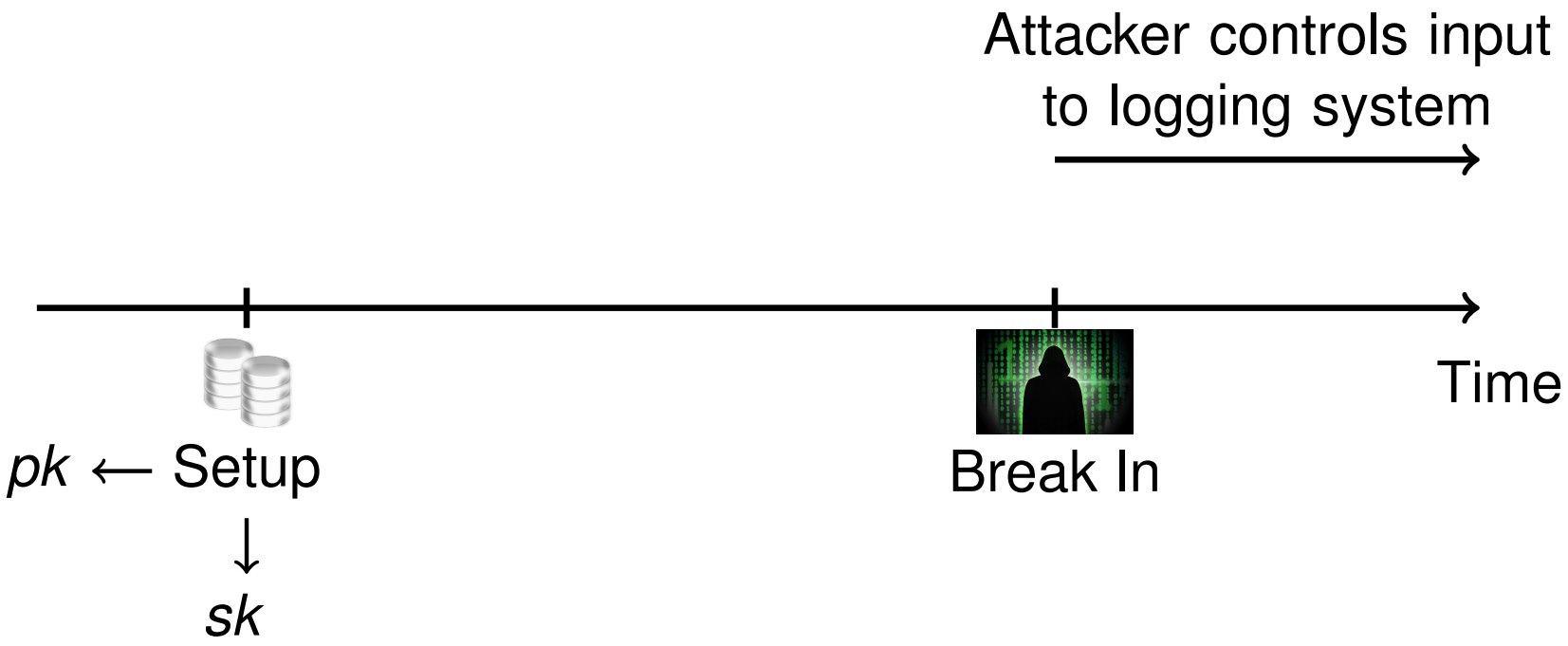
Model

Model



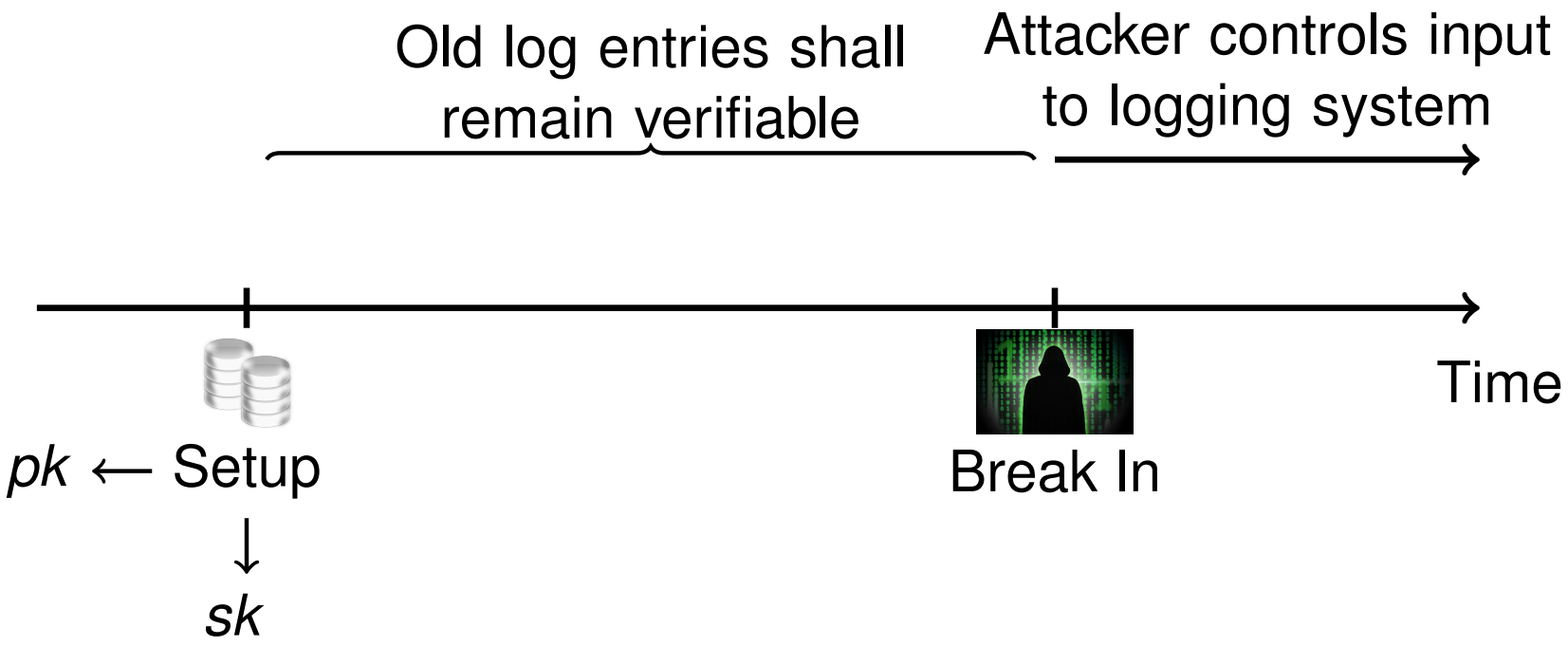
Images: CC-0 by OpenClipArtVectors, CC-BY-SA-4.0 International by www.elbpresse.de

Model



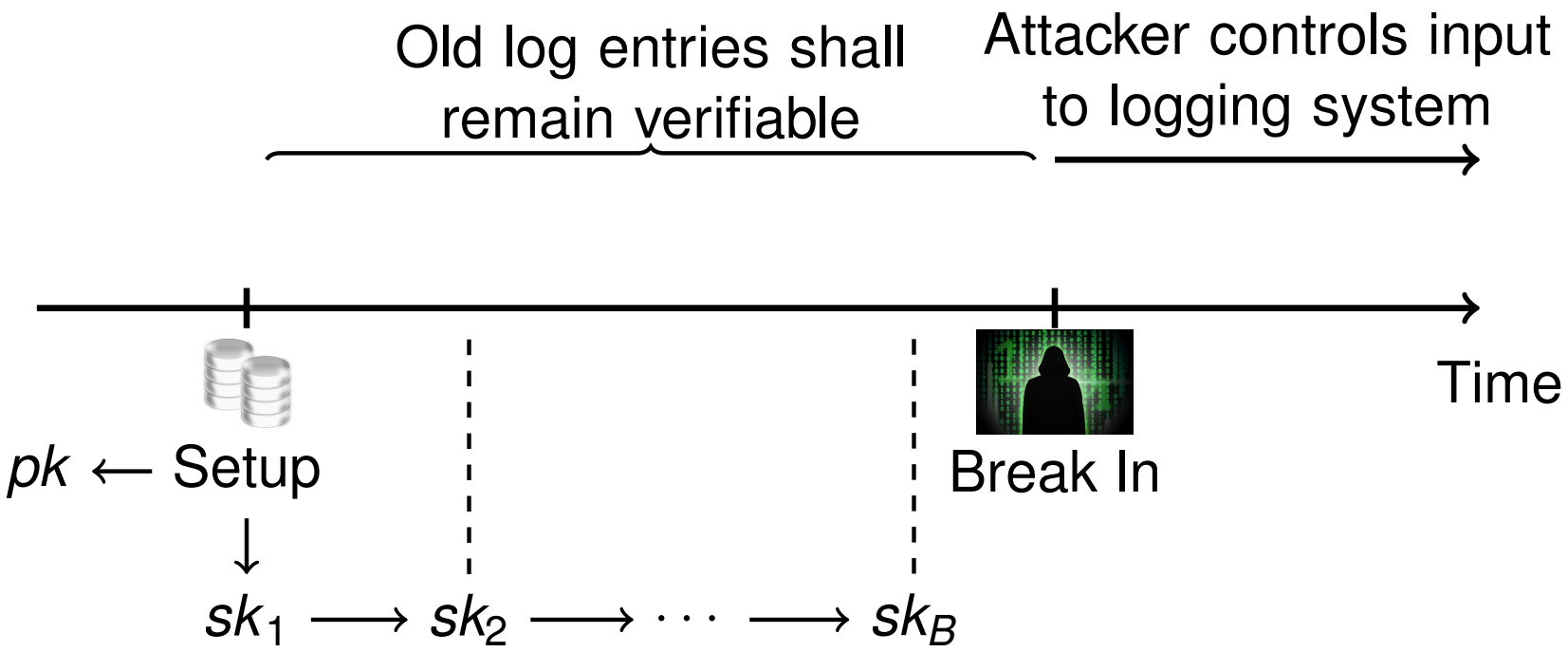
Images: CC-0 by OpenClipArtVectors, CC-BY-SA-4.0 International by www.elbpresse.de

Model



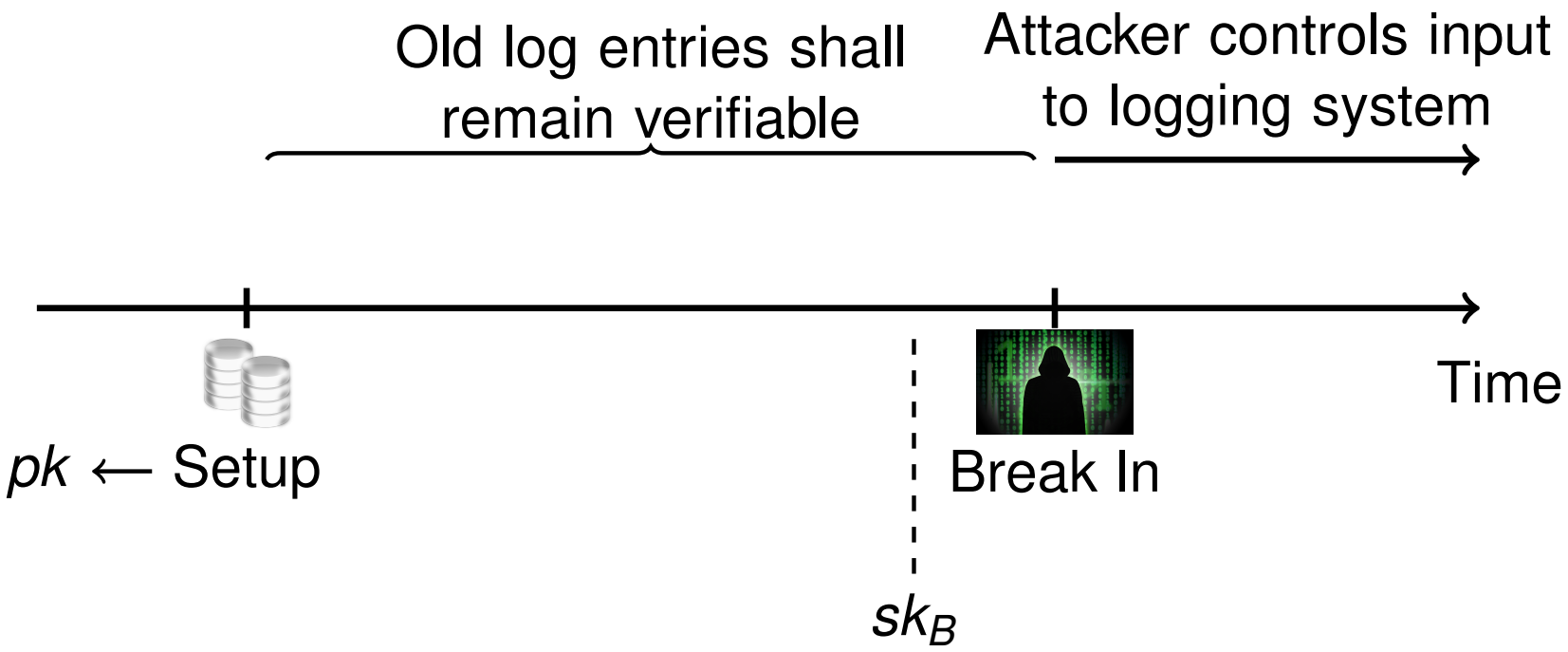
Images: CC-0 by OpenClipArtVectors, CC-BY-SA-4.0 International by www.elbpresse.de

Model

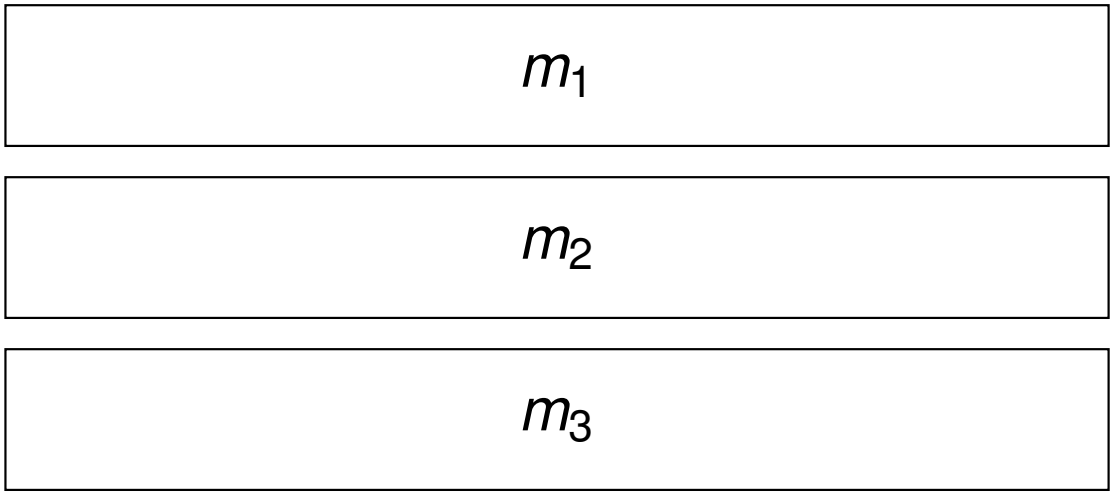


Images: CC-0 by OpenClipArtVectors, CC-BY-SA-4.0 International by www.elbpresse.de

Model



Secure Logging with Crypto



[BY97], [SK98], [BY03], [Hol06]

Secure Logging with Crypto

m_1	σ_1
m_2	σ_2
m_3	σ_3

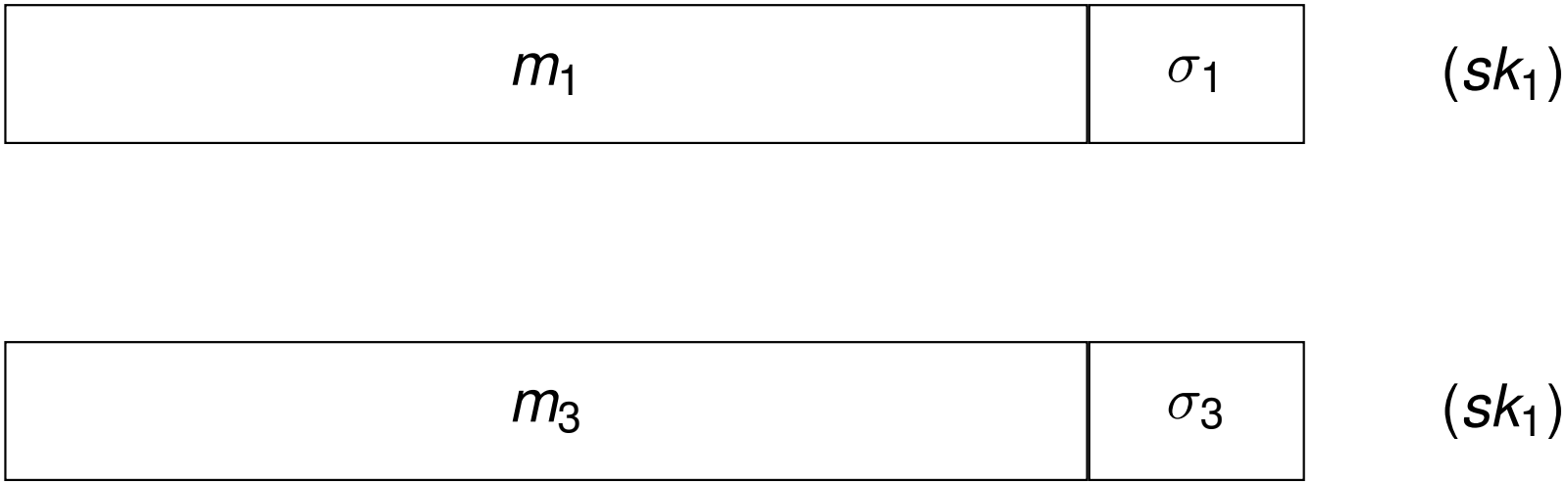
[BY97], [SK98], [BY03], [Hol06]

Secure Logging with Crypto

m_1	σ_1	(sk_1)
m_2	σ_2	(sk_1)
m_3	σ_3	(sk_1)

[BY97], [SK98], [BY03], [Hol06]

Secure Logging with Crypto



[BY97], [SK98], [BY03], [Hol06]

Secure Logging with Crypto

1	m_1	σ_1	(sk_1)
2	m_2	σ_2	(sk_1)
3	m_3	σ_3	(sk_1)

[BY97], [SK98], [BY03], [Hol06]

Secure Logging with Crypto

1	m_1	σ_1	(sk_1)
2	m_2	σ_2	(sk_1)
3	m_3	σ_3	(sk_1)

[BY97], [SK98], [BY03], [Hol06]
don't fully prevent truncation.

Secure Logging with Crypto

1	m_1	σ_1	(sk_1)
2	m_2	σ_2	(sk_1)
3	Switching to sk_2	σ_3	(sk_1)
4	m_3	σ_4	(sk_2)

[BY97], [SK98], [BY03], [Hol06]
don't fully prevent truncation.

Secure Logging with Crypto

1	m_1	σ_1	(sk_1)
2	m_2	σ_2	(sk_1)
3	Switching to sk_2	σ_3	(sk_1)
4	m_3	σ_4	(sk_2)

[BY97], [SK98], [BY03], [Hol06]
don't fully prevent truncation.

(Fully preventing truncation is surprisingly hard.
Solutions: [MT08], [YP09], [YPR12])

Secure Logging with Crypto

1	m_1	σ_1	(sk_1)
2	m_2	σ_2	(sk_1)
3	Switching to sk_2	σ_3	(sk_1)
4	m_3	σ_4	(sk_2)

[BY97], [SK98], [BY03], [Hol06]
don't fully prevent truncation.

(Fully preventing truncation is surprisingly hard.
Solutions: [MT08], [YP09], [YPR12])

Goal here: Prevent truncation to epoch before break-in.

Outline

New Feature: Excerpts

Excerpts should be:

- correct: all messages unchanged
- complete: all relevant log entries present in excerpt

New Feature: Excerpts

Karlsruhe Institute of Technology

Excerpts should be:

- correct: all messages unchanged
- complete: all relevant log entries present in excerpt

Which log entries are “relevant”?

New Feature: Excerpts

Excerpts should be:

- correct: all messages unchanged
- complete: all relevant log entries present in excerpt

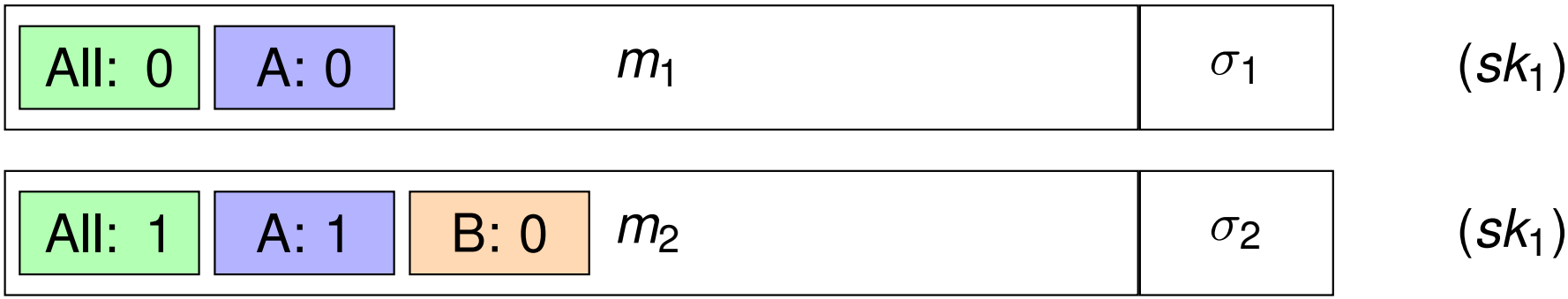
Which log entries are “relevant”?

Defined by application:

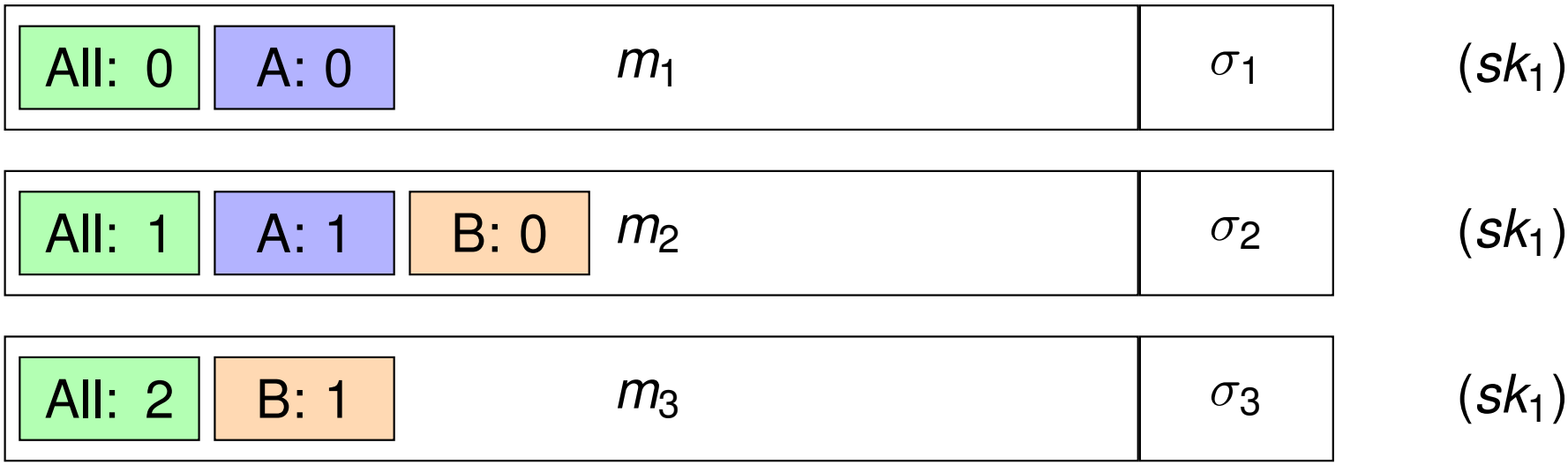
- assign each log entry to ≥ 1 *categories*, identified by name ν
- excerpts for ≥ 1 *entire* categories
- “special” categories:
  - All: contains all log entries
  - EM: contains all epoch markers

Logging with Excerpts

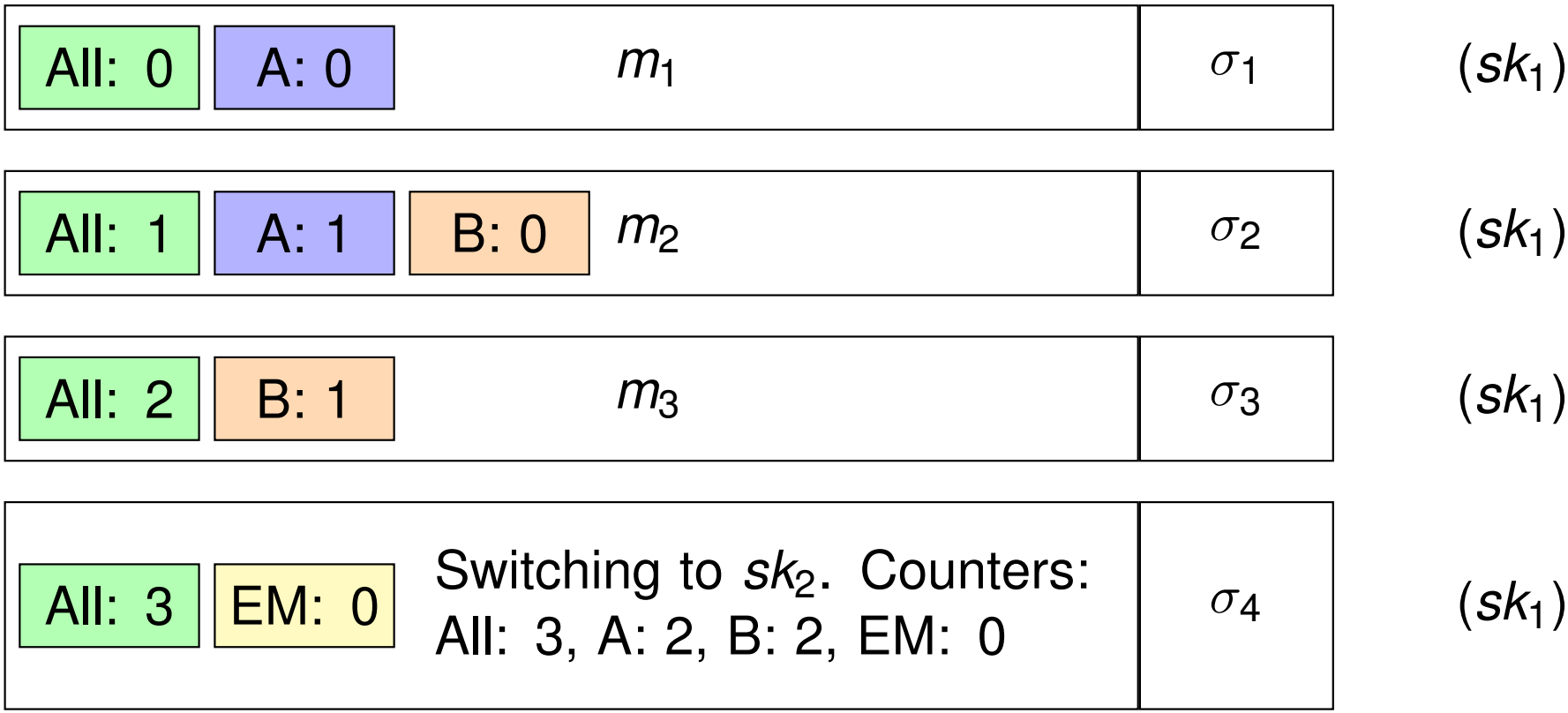
Logging with Excerpts



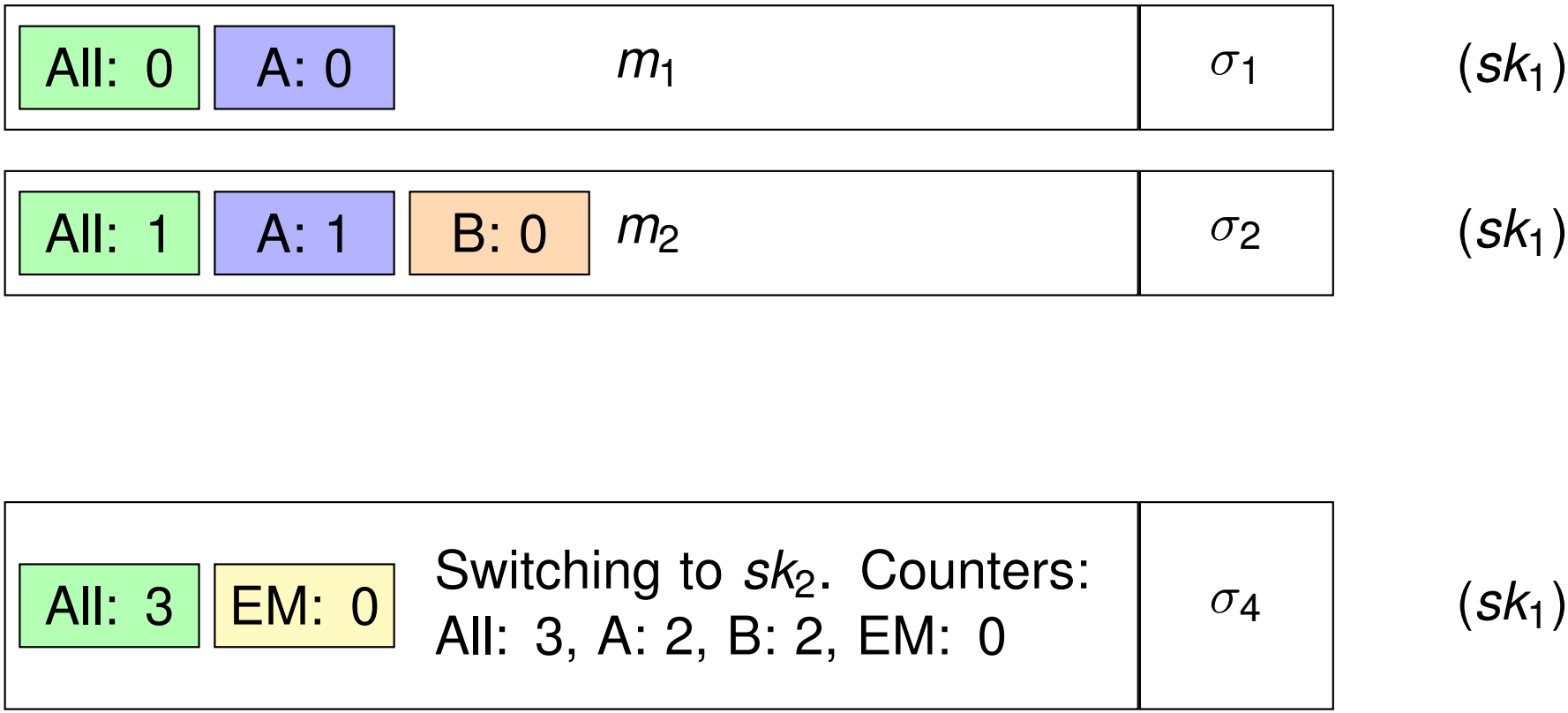
Logging with Excerpts



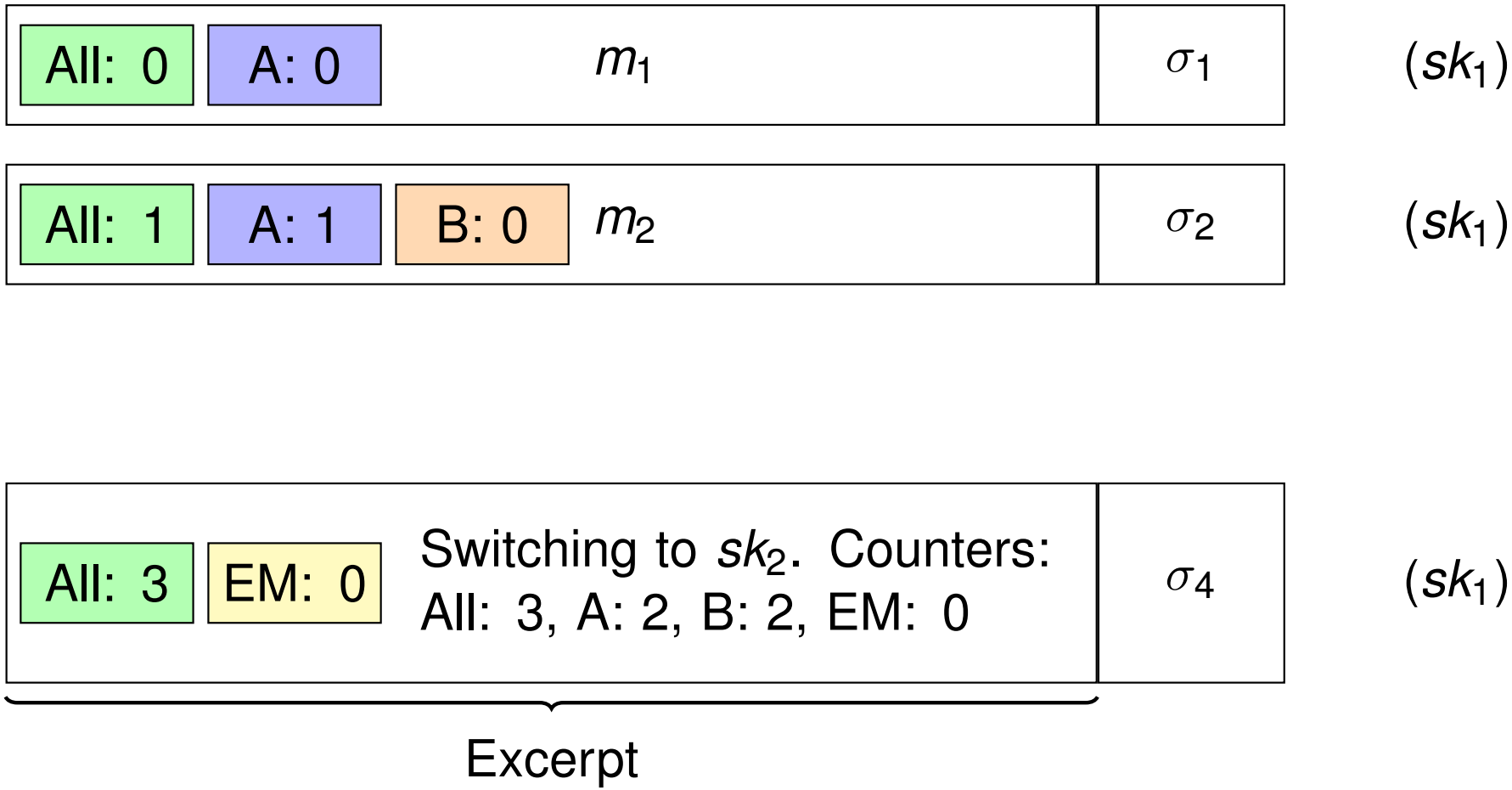
Logging with Excerpts



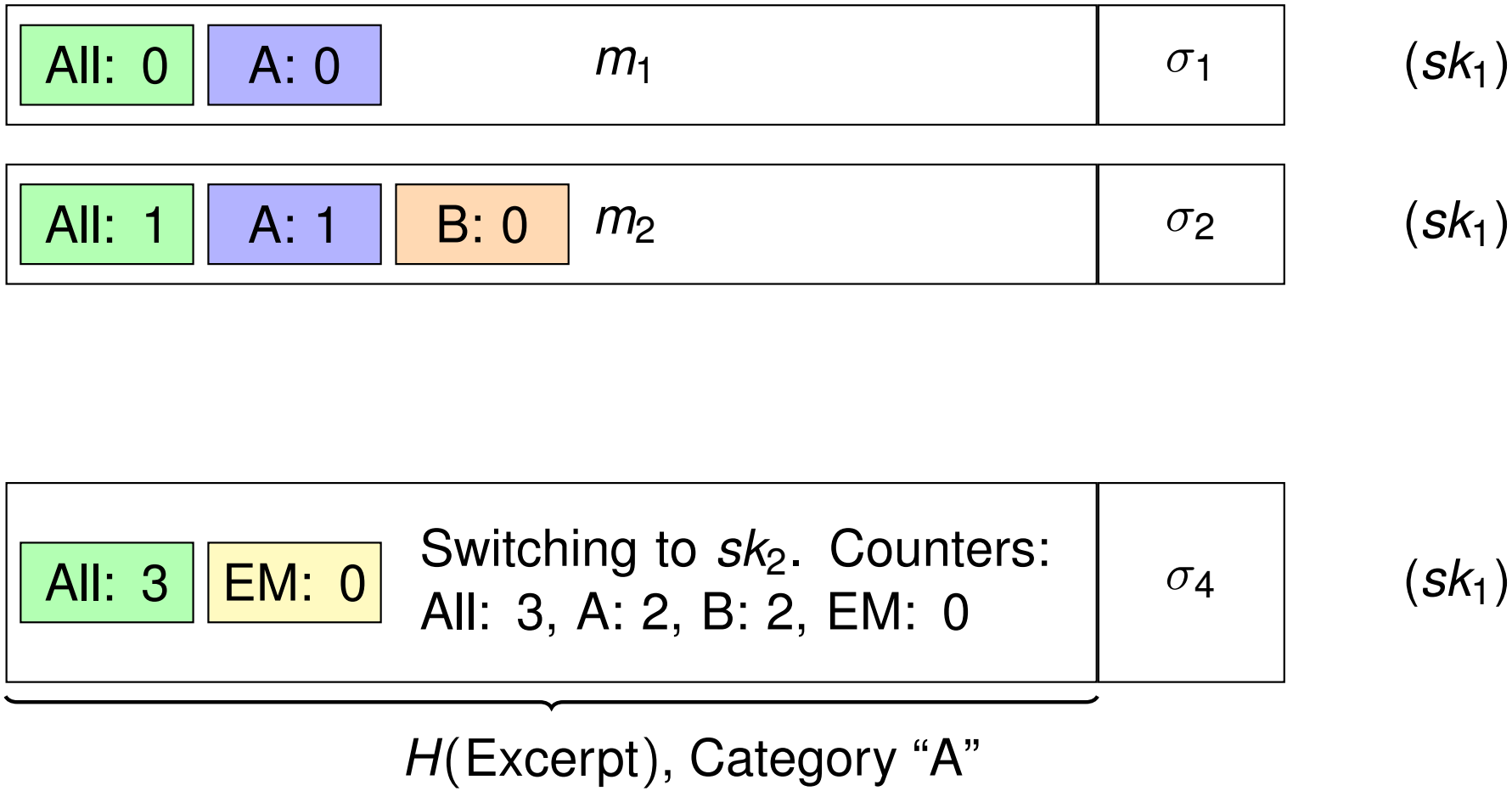
Logging with Excerpts



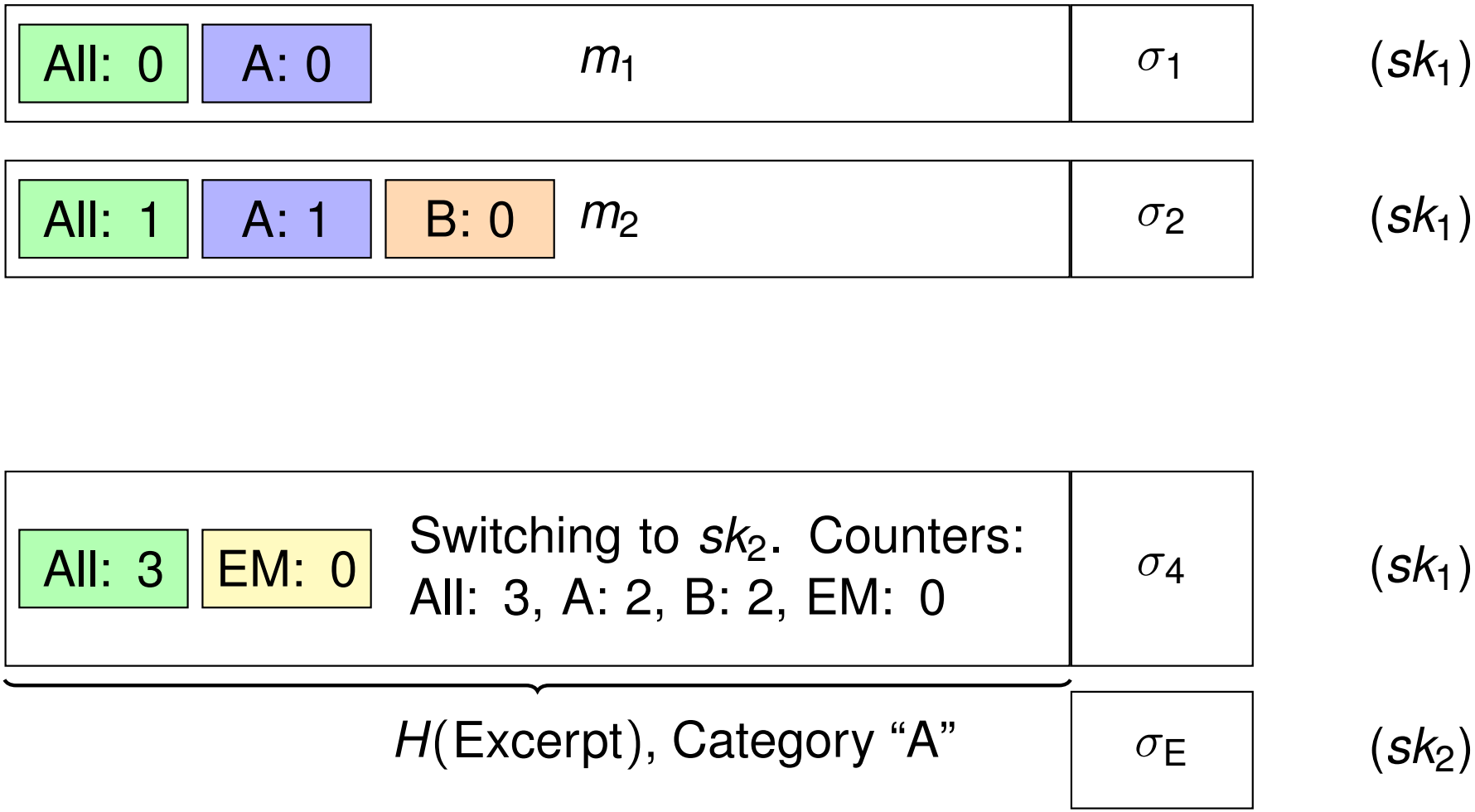
Logging with Excerpts



Logging with Excerpts



Logging with Excerpts



Outline

Security Experiment

Challenger

Images: CC-0 by sheikh_tuhin, barretr, tiothy, CC-BY-SA-4.0 International by www.elbpresse.de

Security Experiment

Challenger

Images: CC-0 by sheikh_tuhin, barretr, tiothy, CC-BY-SA-4.0 International by www.elbpresse.de

Security Experiment

Challenger

Oracles

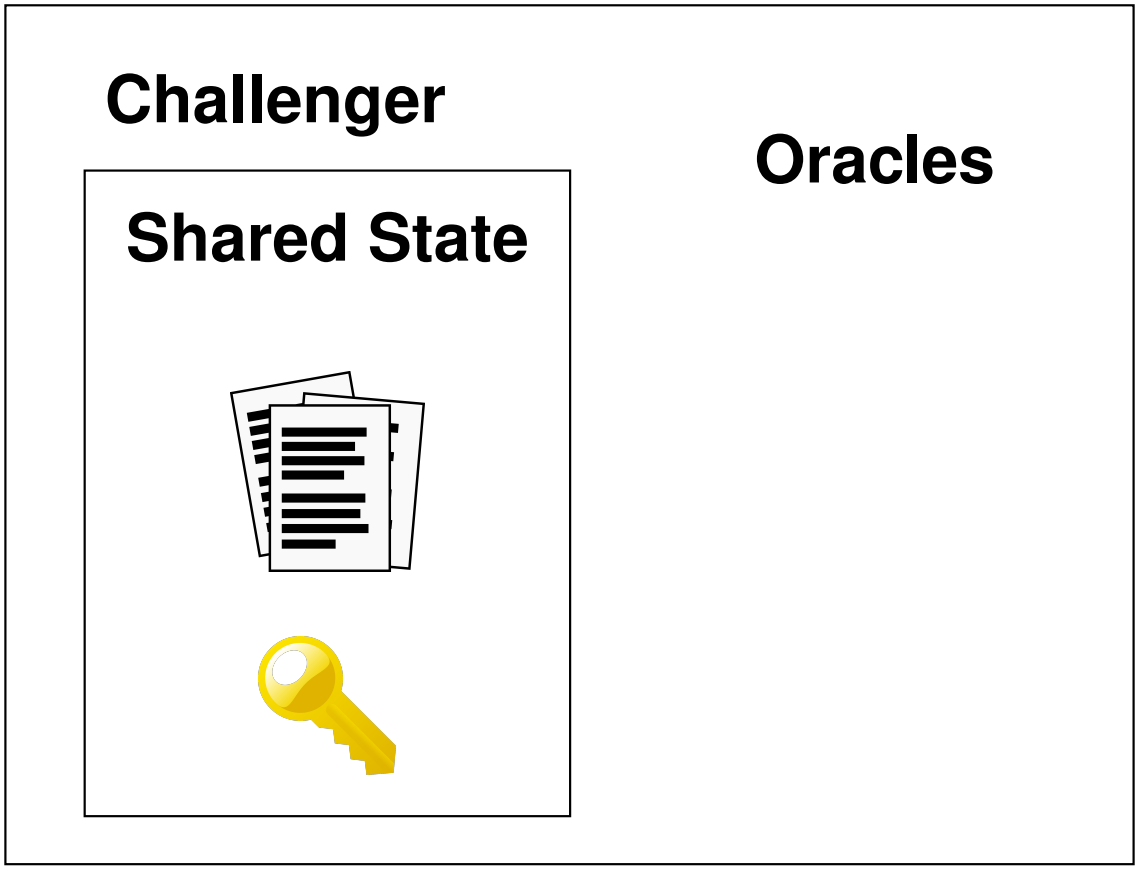
Security Experiment

Challenger

Shared State

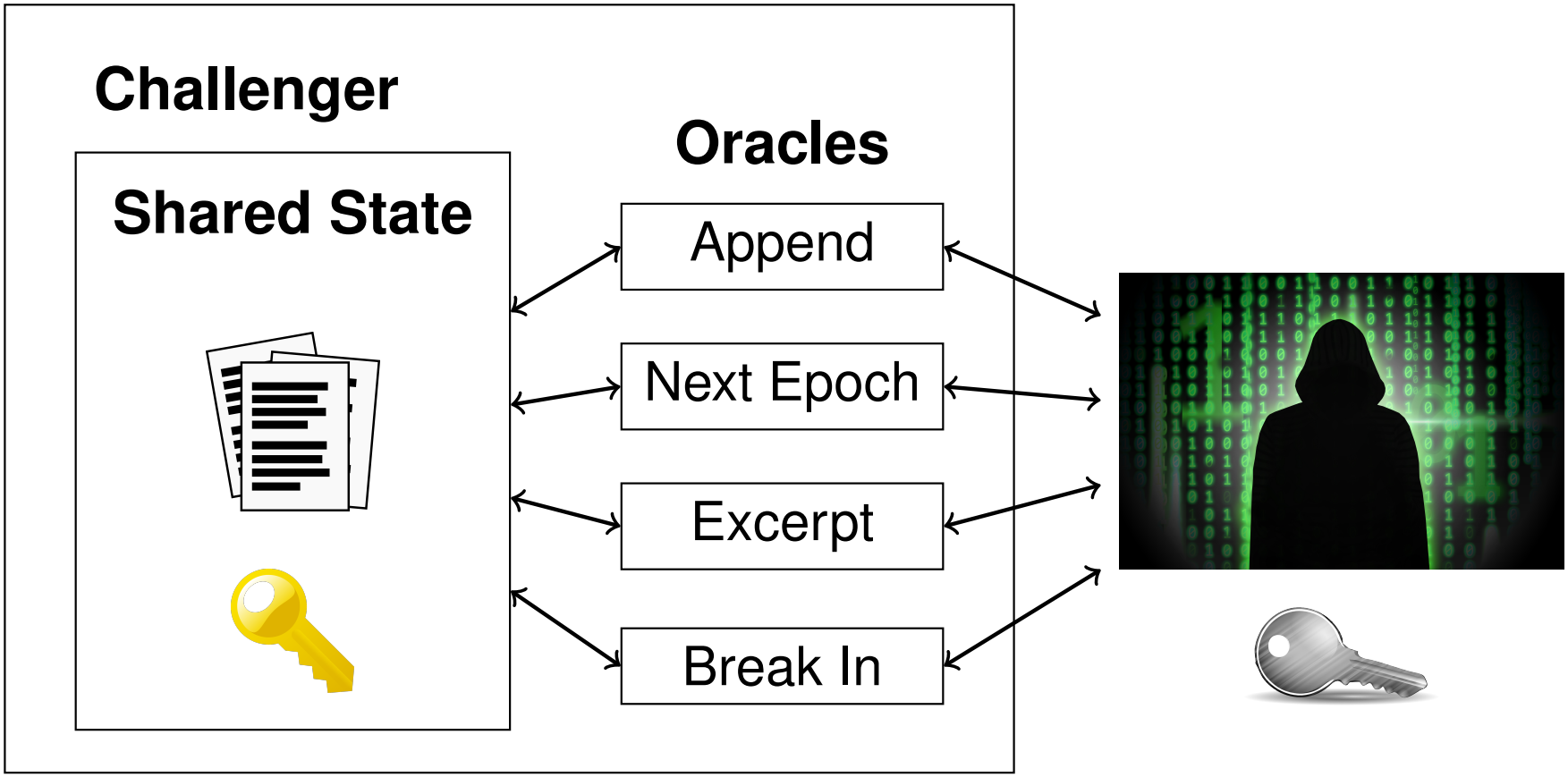
Oracles

Security Experiment



Images: CC-0 by sheikh_tuhin, barretr, tiothy, CC-BY-SA-4.0 International by www.elbpresse.de

Security Experiment



Images: CC-0 by sheikh_tuhin, barretr, tiothy, CC-BY-SA-4.0 International by www.elbpresse.de

Security Definition

Trivial Forgeries:

- excerpts requested from $\mathcal{A}$'s oracle
- if $\mathcal{A}$ got sk_i: pure continuations of the log file state from the most recent epoch switch

Definition (Security, informal)

A logging scheme is *secure* if no PPT adversary has a non-negligible chance of outputting a valid and non-trivial forgery in the above experiment.

Proving Security

Theorem (Informal)

If the above scheme is based on an EUF-CMA-secure signature scheme with forward-security, then it is secure according to the previous definition.

Proof Technique:

- show that attacker must forge ≥ 1 signature if changing any information before last recent epoch switch
- copy that signature and output it as a forgery against the signature scheme
  $\Longrightarrow$ tight

Conclusion

- Secure logging is important.
- Secure logging is hard. Mostly because of truncation.
- Excerpts can be useful.
- Excerpts can be verified securely.

Thank you.
Questions?

Contact: gunnar.hartung@kit.edu
PGP-Key ID: `B1A7 C146`
Fingerprint: `4C39 AC36 6FAD 9E52 3144`
`8352 9E37 381F B1A7 C146`
S/MIME Cert: at
`https://crypto.iti.kit.edu/?id=hartung&L=2`

Backup Slides

- Quotes on Secure Logging ▸ Go
- Forward-Secure Signatures ▸ Go
- Logging Schemes ▸ Go
- Security Proof Sketch ▸ Go
- References ▸ Go
- End ▸ Go

Quotes on Secure Logging

Karlsruhe Institute of Technology

> “Audit data must be protected from modification and unauthorized destruction to permit detection and after-the-fact investigations of security violations.”
> — [Lat85]

> “It is particularly important to ensure the *integrity* of audit trail data against modification. [...] The audit trail files needs to be protected since, for example, intruders may try to ‘cover their tracks’ by modifying audit trail records. ”
> — [NIS95, Section 18.3.1]

> “[A product] shall protect the stored audit records in the audit trail from unauthorised deletion.
> [A product] shall be able to prevent/detect unauthorised modifications to the stored audit records in the audit trail.”
> — [CC12, Section 8.6].

Syntax of Forward-Secure Signatures

KeyGen(T): Create a key pair (sk_0, pk), where sk_0 is the initial secret key. (pk is constant for all epochs.) T is an upper bound on the number of epochs.

Update(sk_i): Compute sk_{i+1} from sk_i. (If $i < T - 1$. sk_i is expected to be erased securely.)

Sign(sk_i, m): Create a signature σ for m with key sk_i.

Verify(pk, i, m, σ): Check if m was signed in epoch i.

- 1^κ is implicit.

Security Experiment

- for a scheme Σ_{FS}, an attacker $\mathcal{A}$, and $T \in \mathrm{poly}(\kappa)$

Setup obtain $(sk_0, pk) \leftarrow \mathrm{KeyGen}(T)$, give pk, T to $\mathcal{A}$.

Queries $\mathcal{A}$ interacts with the challenger:

- may request signature σ for arbitrary messages m
- may force the challenger to update the secret key
- may obtain one secret key sk_i
  - afterwards: no more queries allowed

Forgery $\mathcal{A}$ outputs a message m^*, signature σ^* and epoch number i^*.

Security Definition

Definition (Trivial Forgeries)

A forgery is *trivial* iff:

- $\mathcal{A}$ requested a signature for m^* during epoch i^* or
- $\mathcal{A}$ obtained the secret key for an epoch $i \leq i^*$

Definition (Winning)

$\mathcal{A}$ *wins* an instance of the experiment if it outputs a valid and non-trivial forgery.

Definition (Security)

Σ_{FS} is *secure* if no PPT attacker $\mathcal{A}$ has non-negligible (in κ) chance to win.

◂ Back

Logging Schemes with Excerpts

KeyGen(T): creates key pair sk_0, pk.

Update(sk_i, M, σ): compute sk_{i+1} from sk_i. (If $i < T - 1$. sk_i is expected to be erased securely.) M is the current overall logfile, and σ is the corresponding signature for it.

AppendAndSign($sk_i, (m, N), M, \sigma$): Creates a signature for the message m, which is inserted into the categories N.

Extract(sk_i, M, σ, N): Creates a signature for the excerpt for categories N of M.

Verify(pk, N, E, σ): Checks an excerpt E for completeness and correctness.

New Feature: Excerpts

Definition (Category)

Let M (the log file) be a sequence of log entries (m_i, N_i). The category named ν is the subsequence $C(\nu, M)$ of M that contains all entries with $\nu \in N_i$.

Definition (Excerpt)

Let M (the log file) be a sequence of log entries. An excerpt for categories $N = \{\nu_1, \ldots, \nu_n\}$ is the subsequence

$$E = \bigcup_{\nu \in N} C(\nu, M) \ ,$$

where $C(\nu, M)$ is the category named ν.
(For a proper adaptation of $\cup$ to sequences.)

Proving Security

Theorem (Informal)

If the above scheme is based on an EUF-CMA-secure signature scheme with forward-security, then it is secure according to the previous definition.

Proof Technique:

- show that attacker must forge ≥ 1 signature if changing any information before last recent epoch switch
- copy that signature and output it as a forgery against the signature scheme
  $\Longrightarrow$ tight

Proof (Sketch)

Reduction:

- assume successful attacker $\mathcal{A}$ against logging scheme
- construct attacker $\mathcal{B}$ against Σ_{FS}
- show that $\mathcal{B}$ has non-negligible success probability

Emulation of the Experiment:

- $\mathcal{B}$ must emulate the logging security experiment for $\mathcal{A}$.
- $\mathcal{B}$ plays the forward-secure unforgeability game against Σ_{FS}.

Reduction

$\mathcal{A}$'s information:	emulated by $\mathcal{B}$ through:
input: pk, T	input: pk, T

Image: CC-BY-SA-3.0 Unported by Steschke, via Wikipedia

Reduction

$\mathcal{A}$'s information:	emulated by $\mathcal{B}$ through:
input: pk, T	input: pk, T ✓
signature (logging) oracle	signature oracle

Image: CC-BY-SA-3.0 Unported by Steschke, via Wikipedia

Reduction

$\mathcal{A}$'s information:	emulated by $\mathcal{B}$ through:
input: pk, T	input: pk, T ✓
signature (logging) oracle	signature oracle ✓
epoch switching	epoch switching

Image: CC-BY-SA-3.0 Unported by Steschke, via Wikipedia

Reduction

$\mathcal{A}$'s information:	emulated by $\mathcal{B}$ through:
input: pk, T	input: pk, T ✓
signature (logging) oracle	signature oracle ✓
epoch switching	epoch switching ✓
breaking in	breaking in

Image: CC-BY-SA-3.0 Unported by Steschke, via Wikipedia

Reduction

$\mathcal{A}$'s information:	emulated by $\mathcal{B}$ through:
input: pk, T	input: pk, T ✓
signature (logging) oracle	signature oracle ✓
epoch switching	epoch switching ✓
breaking in	breaking in ✓
excerpt oracle	signatures for individual log entries + signature oracle

Reduction

$\mathcal{A}$'s information:	emulated by $\mathcal{B}$ through:
input: pk, T	input: pk, T ✓
signature (logging) oracle	signature oracle ✓
epoch switching	epoch switching ✓
breaking in	breaking in ✓
excerpt oracle	signatures for individual log entries + signature oracle ✓

$\rightsquigarrow$ Left to Show:
Any valid and non-trivial excerpt forgery contains a valid and non-trivial signature forgery.

Forgeries

Case 1: $i^* < i_{\text{BreakIn}}$ epoch markers in excerpt E:

- signature on (N, E) valid for epoch $i^* < i_{\text{BreakIn}}$.
- $\mathcal{A}$ never queried for a signature for E
  $\Longrightarrow$ $\mathcal{B}$ never queried for (N, E)
  (assuming proper encoding)
  $\Longrightarrow$ valid and non-trivial forgery on (N, E)

Case 2: $\geq i_{\text{BreakIn}}$ epoch markers in excerpt E:

$\Rightarrow$ changed the excerpt wrt. a previous epoch $i^* < i_{\text{BreakIn}}$

$\rightsquigarrow$ restrict the discussion to epochs before break-in

Forgeries

Assume for contradiction:
All messages (including counters!) in forged excerpt were queried at signature oracle before.

Ensured by Verification:

- no messages from other categories
- no duplicates
- message order

$\Longrightarrow$ forged excerpt is subsequence of "real" excerpt.
non-trivial $\Longrightarrow$ *strict* subsequence ($\subsetneq$)

Verification checks for completeness $\Longrightarrow$ excerpt invalid
$\Longrightarrow$ contradiction $\Longrightarrow$ $\mathcal{A}$ forged a signature $\square$

◂ Back

References I

- Mihir Bellare and Bennet S. Yee, *Forward integrity for secure audit logs*, Tech. report, University of California at San Diego, 1997.
- Mihir Bellare and Bennet Yee, *Forward-security in private-key cryptography*, Topics in Cryptology — CT-RSA 2003 (Marc Joye, ed.), Lecture Notes in Computer Science, vol. 2612, Springer Berlin Heidelberg, 2003, pp. 1–18 (English).
- *Common Criteria for Information Technology Security Evaluation, version 3.1 r4, part 2*, September 2012, `https://www.commoncriteriaportal.org/cc/`.

References II

- Jason E. Holt, *Logcrypt: Forward security and public verification for secure audit logs*, Proceedings of the 2006 Australasian Workshops on Grid Computing and e-Research – Volume 54 (Darlinghurst, Australia, Australia), ACSW Frontiers '06, Australian Computer Society, Inc., 2006, pp. 203–211.
- Donald C. Latham (ed.), *Department of defense trusted computer system evaluation criteria*, US Department of Defense, December 1985, `http://csrc.nist.gov/publications/history/dod85.pdf`.

References III

- Di Ma and Gene Tsudik, *A new approach to secure logging*, Data and Applications Security XXII (Vijay Atluri, ed.), Lecture Notes in Computer Science, vol. 5094, Springer Berlin Heidelberg, 2008, pp. 48–63 (English).
- *An Introduction to Computer Security: The NIST handbook*, October 1995, NIST Special Publication 800-12.
- Bruce Schneier and John Kelsey, *Cryptographic support for secure logs on untrusted machines*, The Seventh USENIX Security Symposium Proceedings, 1998.
- Attila A. Yavuz and Ning Peng, *BAF: An efficient publicly verifiable secure audit logging scheme for distributed systems*, Computer Security Applications Conference, 2009. ACSAC '09. Annual, Dec 2009, pp. 219–228.

References IV

Attila A. Yavuz, Ning Peng, and Michael K. Reiter, *Efficient, compromise resilient and append-only cryptographic schemes for secure audit logging*, Financial Cryptography and Data Security (Angelos D. Keromytis, ed.), Lecture Notes in Computer Science, vol. 7397, Springer Berlin Heidelberg, 2012, pp. 148–163 (English).