Building an Encrypted and Searchable Audit Log

Waters, Balfanz, Durfee & Smetters

Presenter: Matthew Green

Talk Outline

- Searches on Encrypted Data: Background & Previous Work
- Secure Audit Logs, The Scheme
- Extensions and Recent Work
- Implementation: Where is it?
- Open Problems

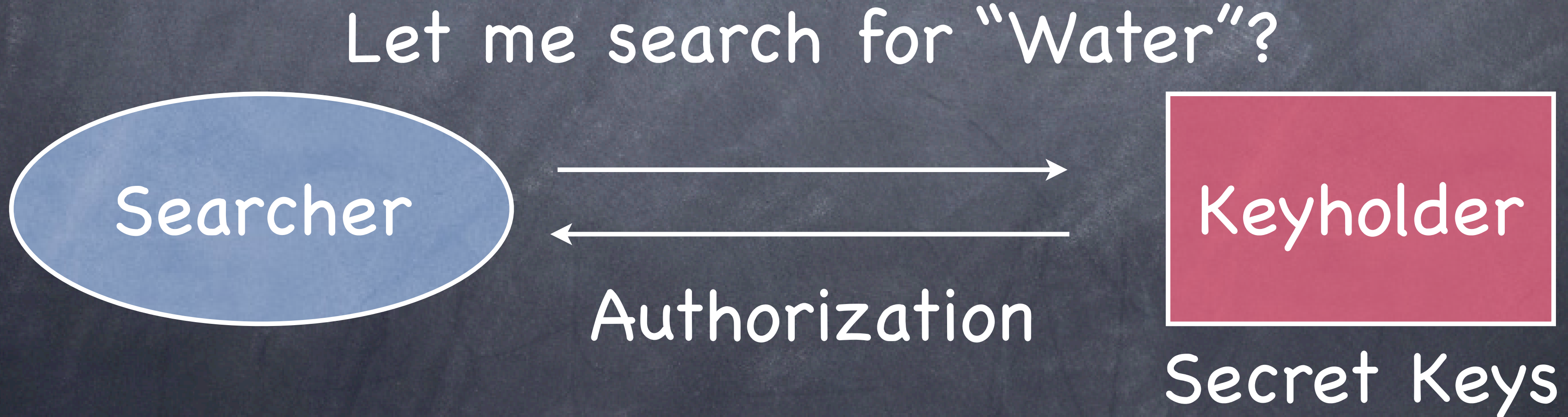
Searching Encrypted Data?

- Search ciphertexts based on contents
- Maintain confidentiality, allow searchers to detect certain elements, e.g. keyword
- Notions of security, Dictionary attacks?

$E_k($ "3100 Wyman Park Drive, Baltimore" $)$

Delegated Searching

- Contact the Keyholder for authorization to search on a particular term

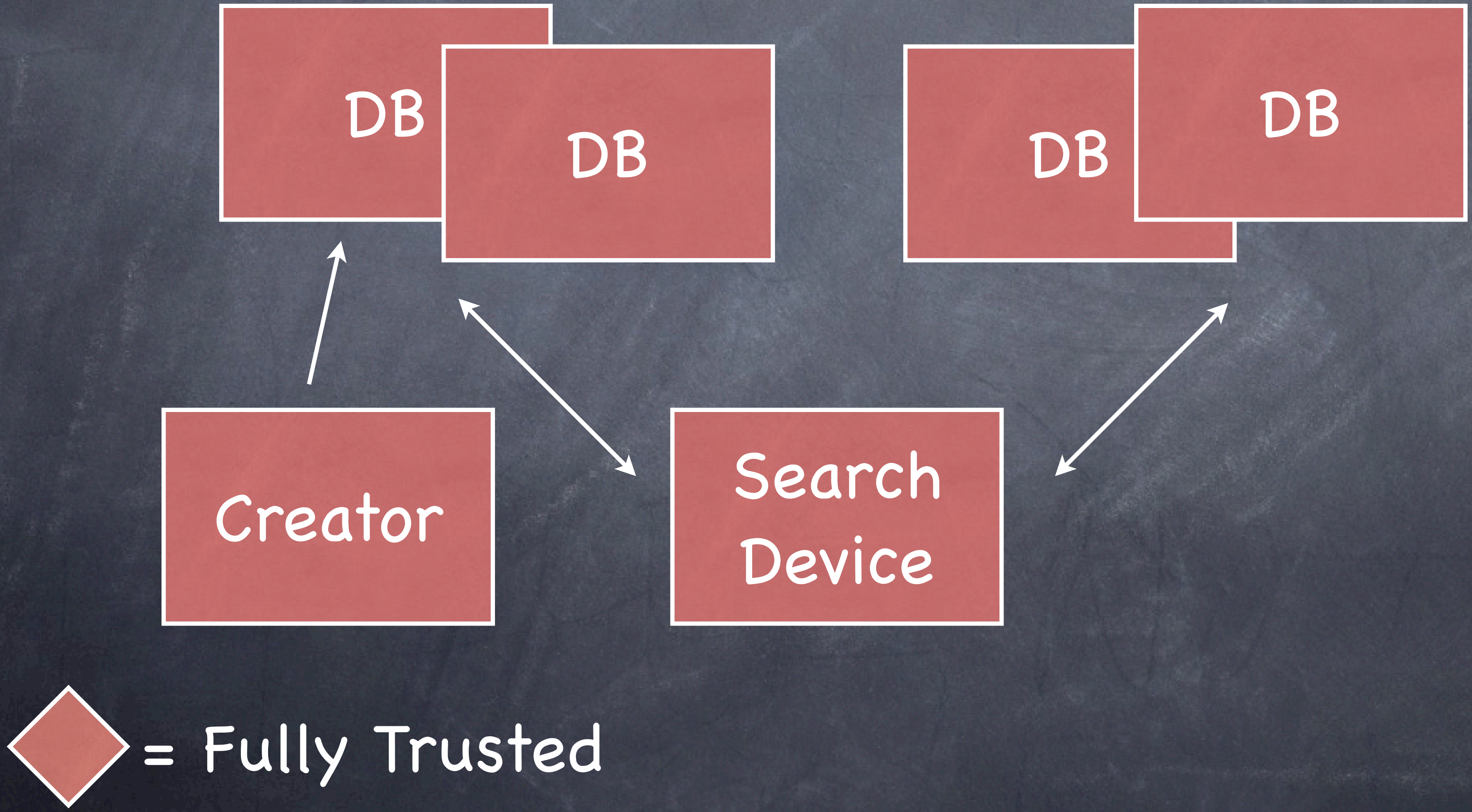


Delegating: Motivation

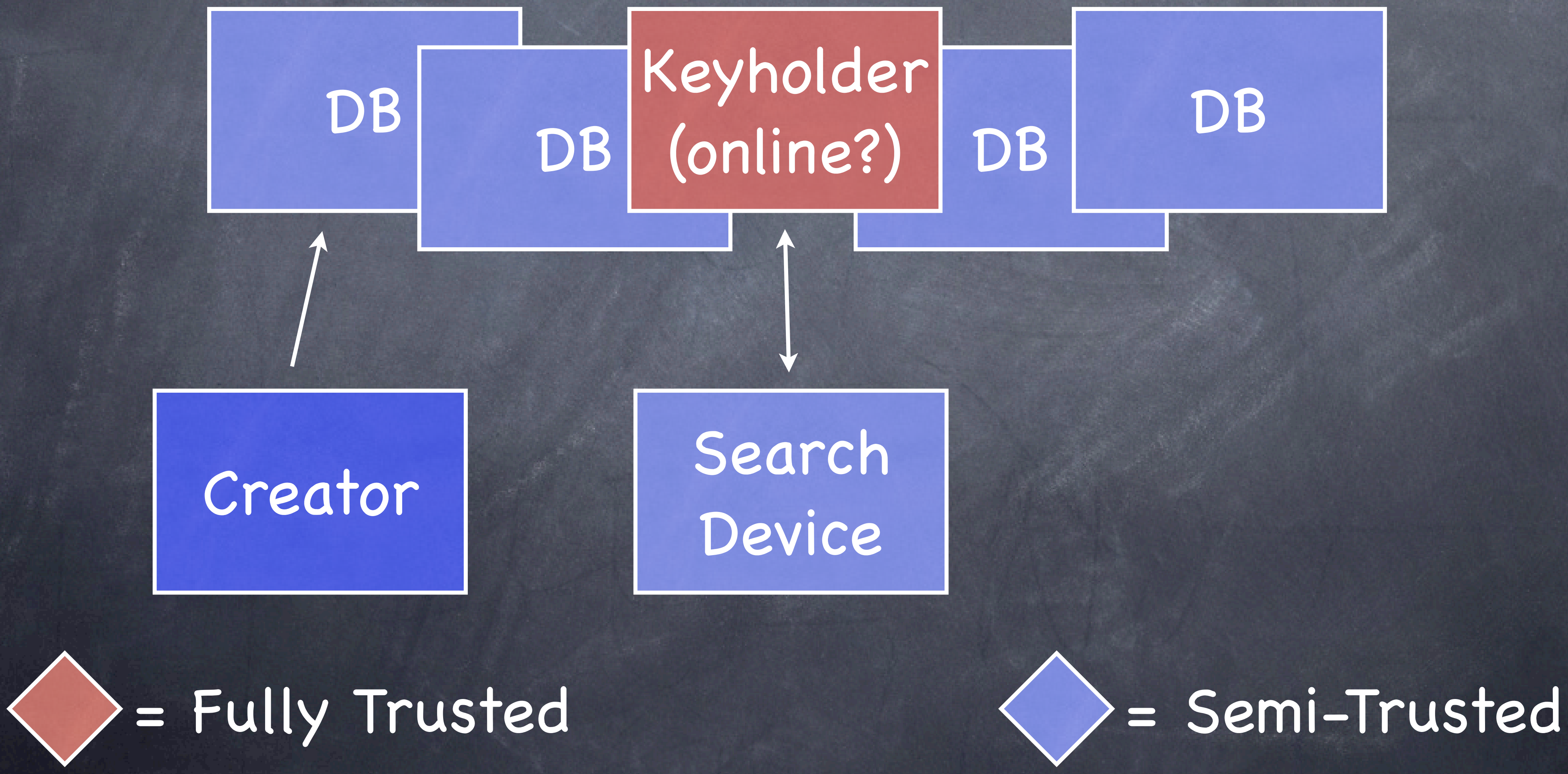
- Motivation is twofold:
  - Efficiency: keyholder can offload search workloads to somebody else, reduce bandwidth
  - Reduce size of Trusted Computing Base

Keyholder

Trusted Computing Base



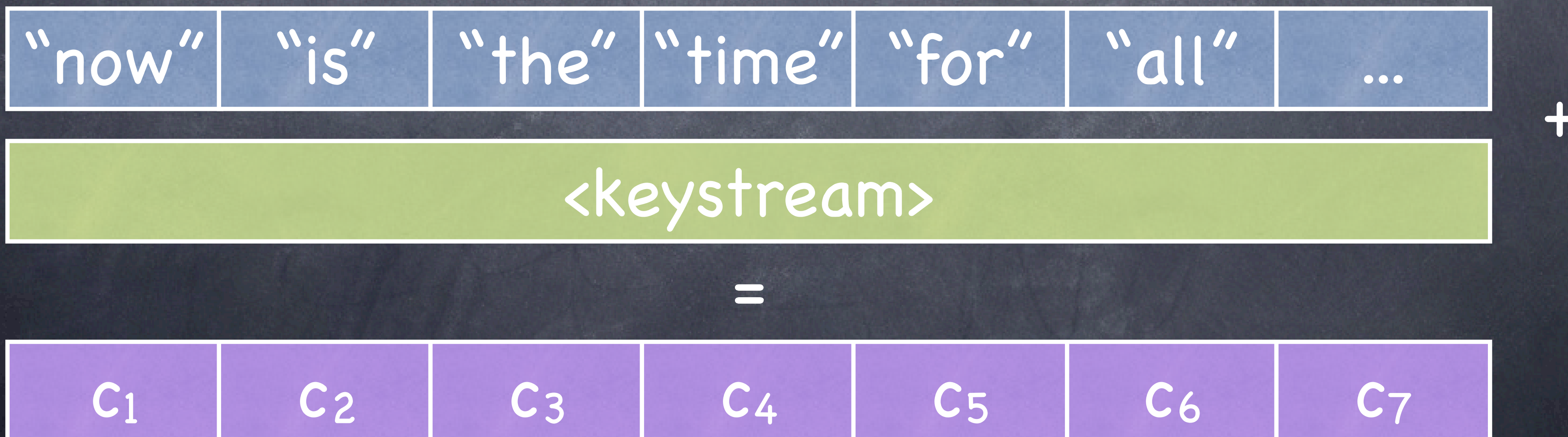
Reducing a Trusted Computing Base



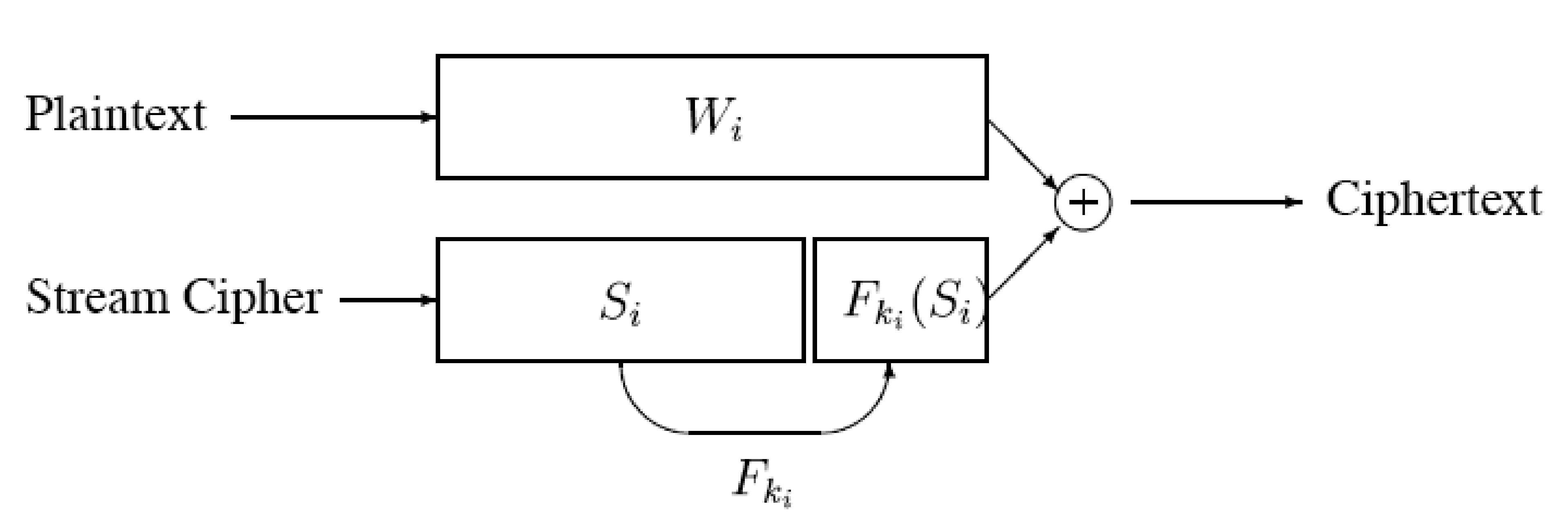
Schemes

Song, Wagner & Perrig

- Plaintext is divided into words, $w_1 \ldots w_n$
- Encrypted with a symmetric-key stream cipher



Song, Wagner & Perrig



SW&P, Searching

(now is the time for all ...)

c_1	c_2	c_3	c_4	c_5	c_6	c_7

c_1	c_4	c_7
XOR	XOR	XOR
"time"	"time"	"time"
=	=	=
<???>	$S_4, f_k(S_4)$	<???>

Search delegation: keyholder reveals k, to allow tests on $\langle S_i, f_k(S_i) \rangle$

Secure Indexes (Goh)

- Goh introduces IND-CKA, IND2-CKA model for ciphertexts
  - IND-CKA: A ciphertext reveals no information unless you search for the precise keyword
  - IND-CKA2: As above, reveals no information about the # of keywords

Audit Logs

- Record activity that takes place on a server/device.
  - Log attacks/unauthorized usage
- Should be efficiently searchable by authorized users (e.g., searches by username or activity type)

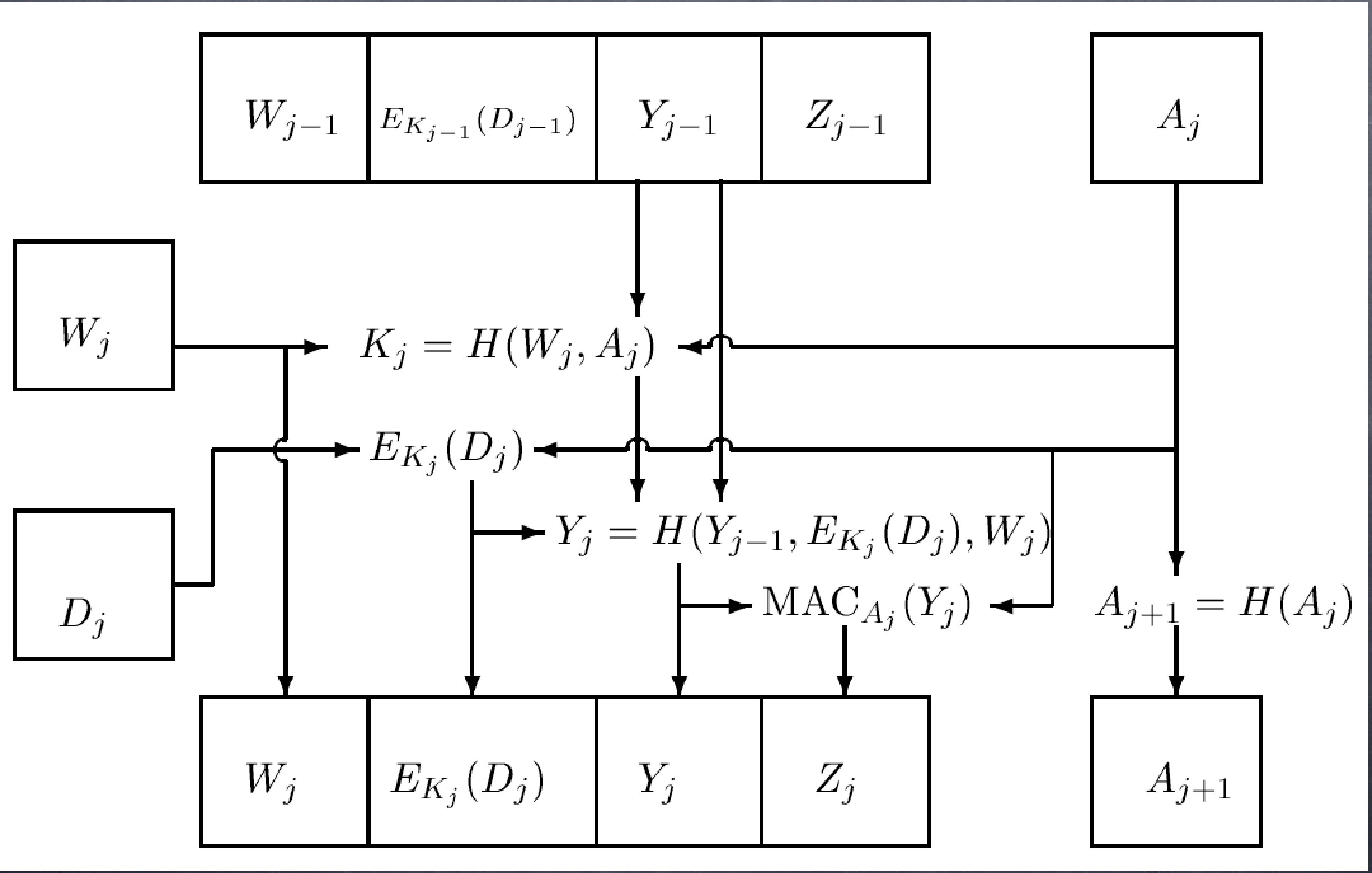
Audit Log Attacks

- Attacker gains total control of machine and all of its secrets. There are three primary threats to the audit log:
  - Destruction (total or selective)
  - Modification, e.g. to cover attack trail
  - Examination, e.g. to recover usage data & other potentially useful information

Protecting Log Integrity

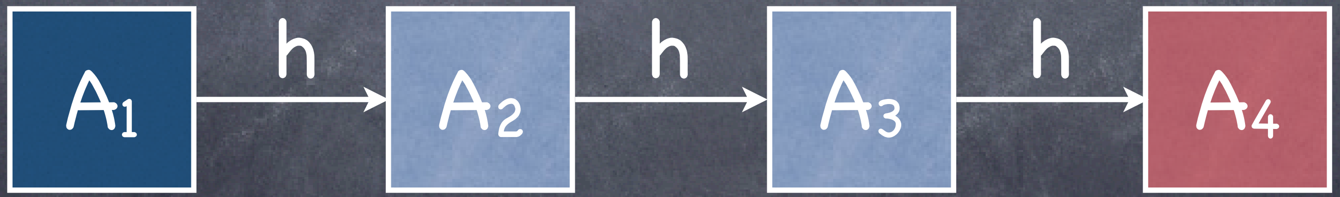
- Schneier & Kelsey: Cryptographic Protection for Audit Logs
- Ensures integrity & privacy of log entries written <u>before</u> compromise
- (can't save entries written afterwards!)

Schneier/Kelsey



Integrity & Privacy

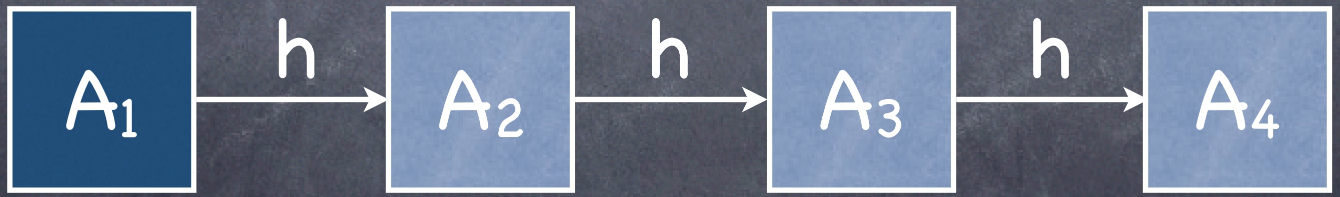
- S&K use a hash-chain to guarantee security/integrity of older log entries
- Forward Secure



$$k_n = f(A_n)$$
$$km_n = f'(A_n)$$

Integrity & Privacy

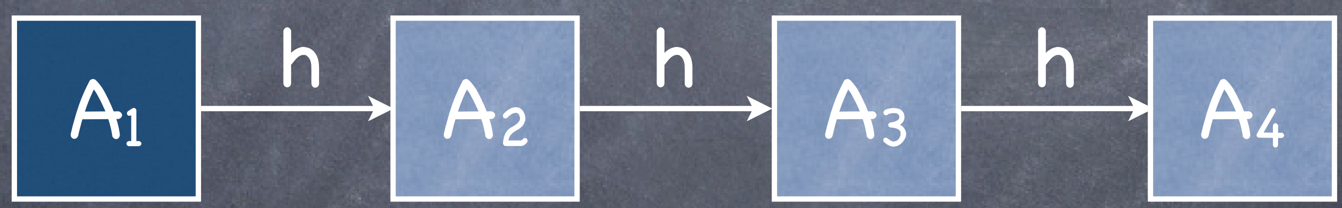
- Decryption requires the original secret (or some intermediate version)
- Search requires full decryption
- Must be absolutely sure A_{n-1} is eradicated



$$k_n = f(A_n)$$
$$km_n = f'(A_n)$$

Selective Record Types

- We can limit which records a user can decrypt, by deriving keys based on public record types



Type = (critical) (routine) (private) (routine)

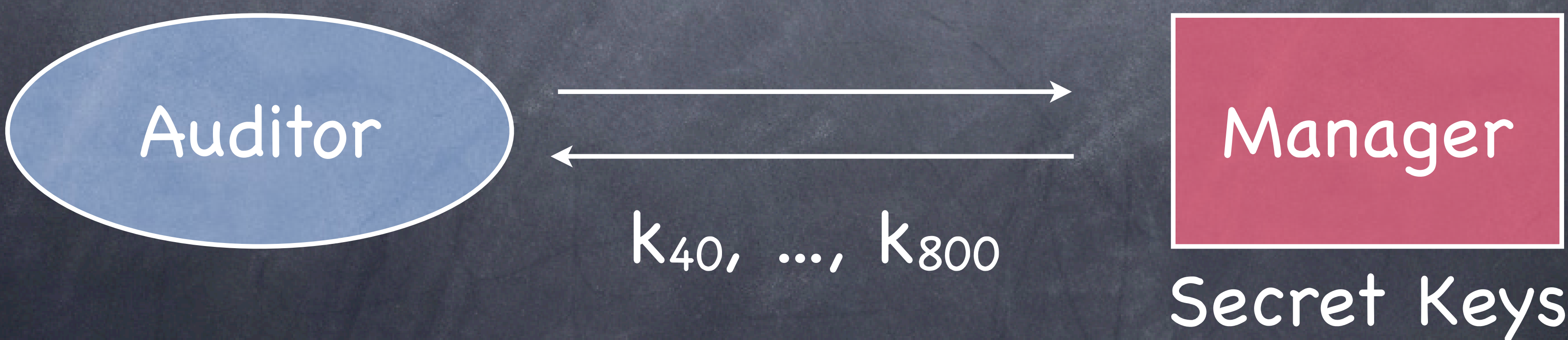
$$k_n = f(\text{Type}, A_n)$$

$$km_n = f'(\text{Type}, A_n)$$

Decrypting a Log

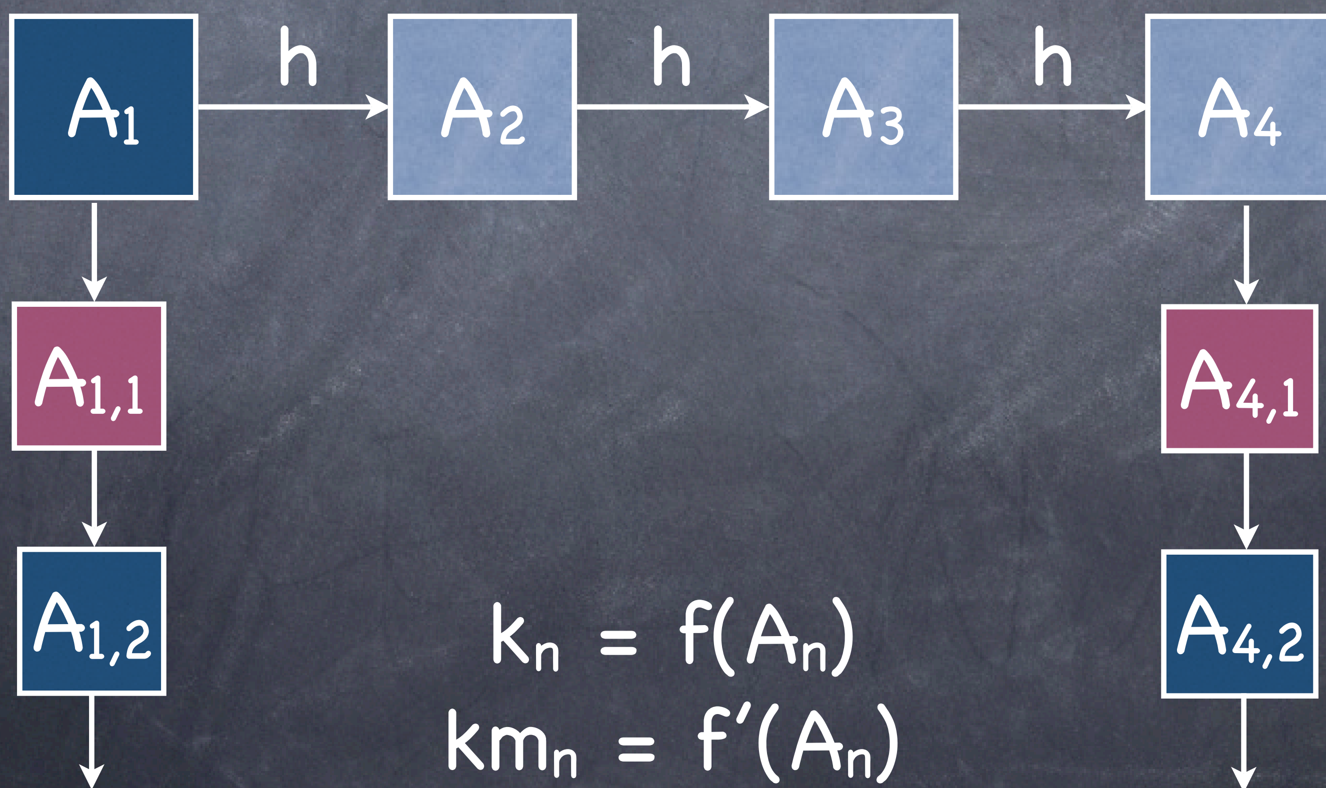
- Contact the Trusted Manager for a decryption key on any log entries you want
- Specify entry types (or keys won't work)

Might I decrypt entries 40-800 of types {....}?



Time-based Access

- Schneier/Kelsey can provide time-based decryptions (or search)



Identity Based Encryption

- First proposed by Shamir in 1984, actual schemes by Cox, then Boneh & Franklin
  - Anyone can compute a Public Key from some public Info + a string
  - PKG can generate a Secret Key from the string + some secret Info

PK= "mgreen@cs.jhu.edu" + PK_M

SK= "mgreen@cs.jhu.edu" + SK_M

PKG

Elliptic Curves

- Based on Curve Points (e.g, P, Q.)
- Point Addition, similar to integer multiplication:
  (P + Q) = (Q + P), (Q + <unity>) = Q
- Scalar Multiplication, similar to exponentiation:
  e.g.: 5 * P = (P + P + P + P + P)
  1 * P = P
  q * P = P (where q is the order)

Cryptographic Assumptions

- Discrete Logarithm Problem:
  Given g^a mod p, find a
- Computational Diffie-Hellman Problem:
  Given g^a & g^b, find g^{ab} (mod p)

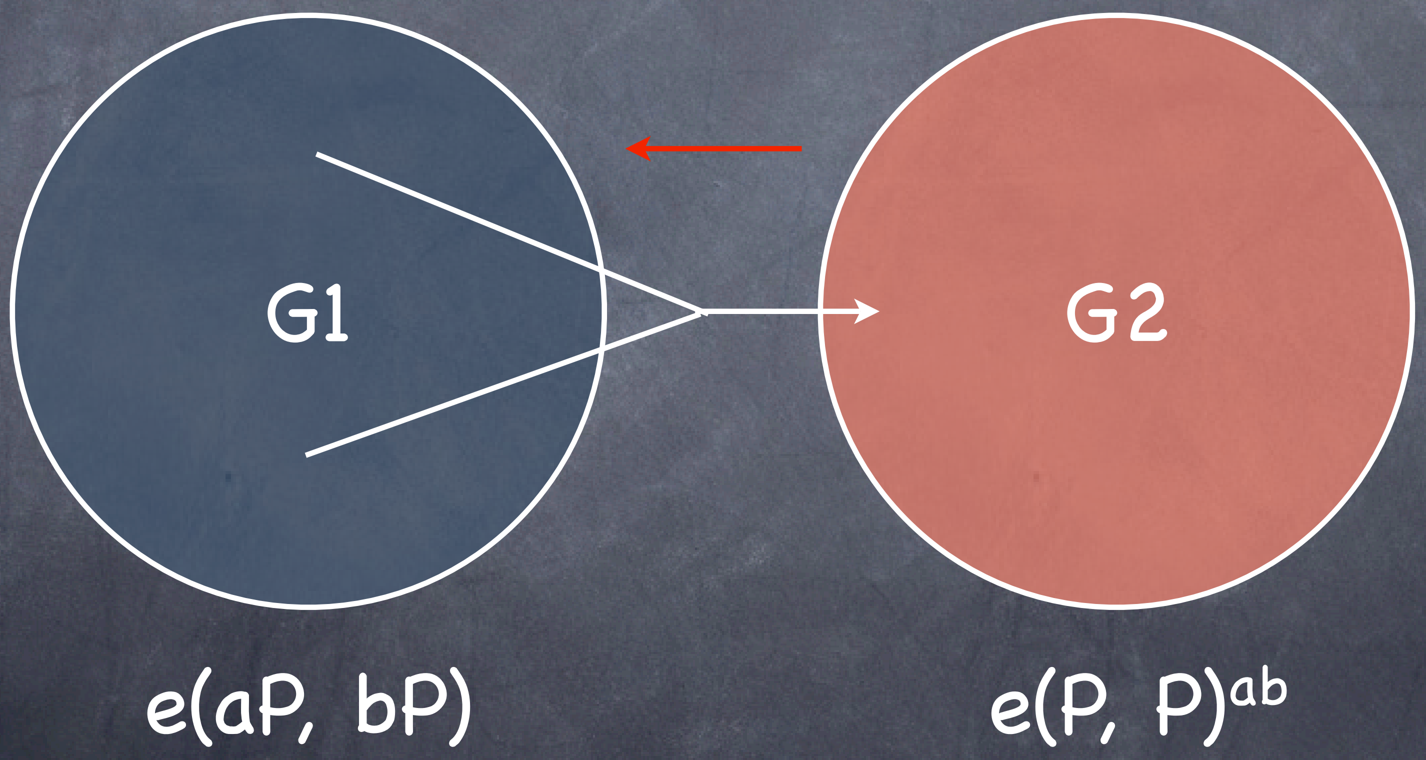
Elliptic Curve Assumptions

- EC-Discrete Logarithm Problem:
  Given aP, find a
- EC-Computational Diffie-Hellman Problem:
  Given aP & bP, find abP

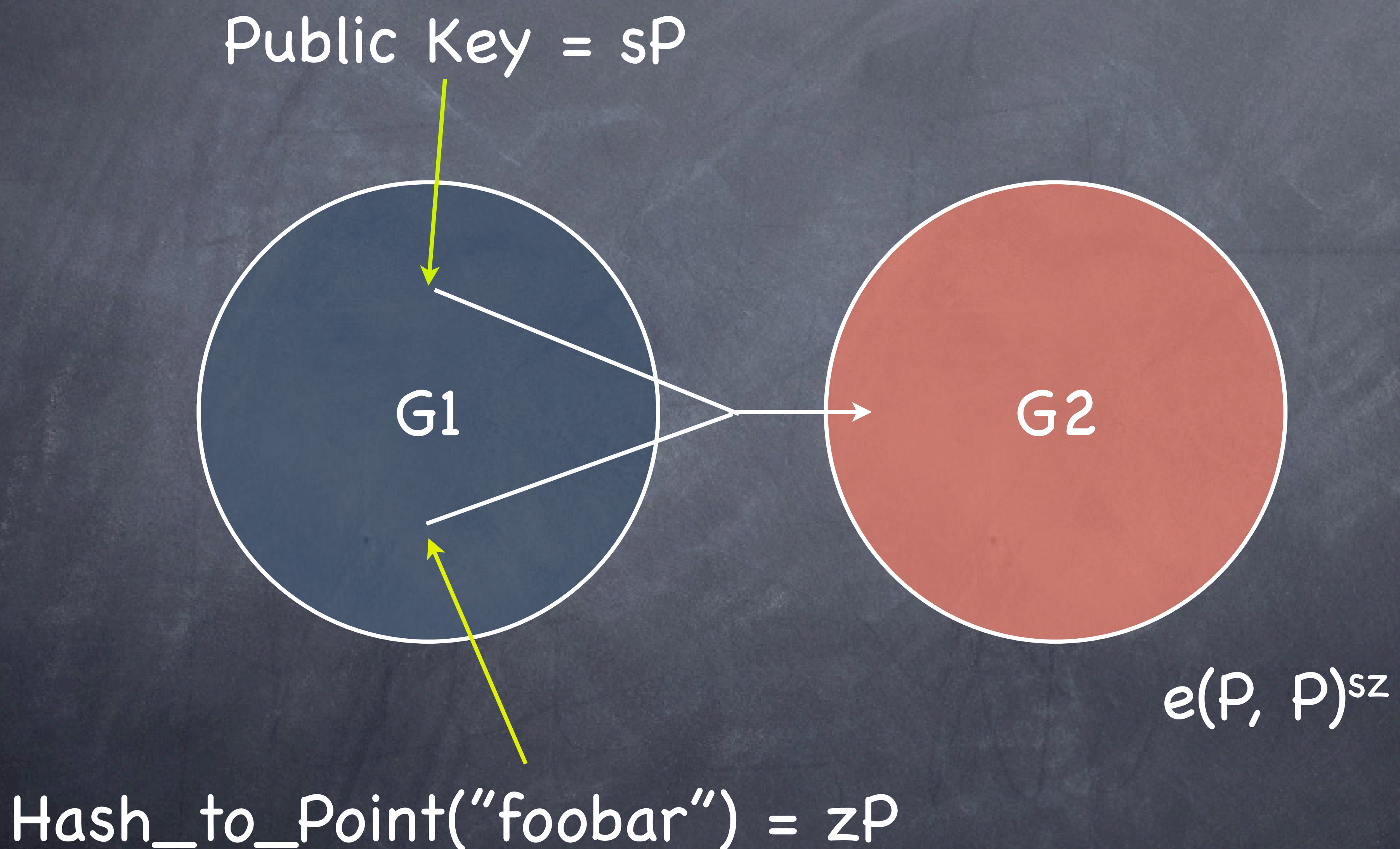
Bilinear Pairings

- A Bilinear Pairing is a function e(G1, G1) -> G2 with the following properties:
  - Non-degeneracy. For generator points <P, Q> in G1, e(P, Q) is a generator of G2
  - Bilinearity. $e(aP, bQ) = e(P, Q)^{ab}$
  - One Way. No way to map back from G2 to G1

Pairings != CDH



Fun With Pairings



Boneh & Franklin's IBE

- A pairing $e(P, Q) \rightarrow Z_q$
  Two hash functions: Hash_to_Point(), H()
- Public Parameters: (curve params, p, q, P)
- $SK_M = s$, $PK_M = sP$

B & F’s IBE Encryption

- GET_PK(PK_M = sP, “<keystring>”):
  PK = e(Hash_to_Point(”<keystring>”, sP)
  = e(zP , sP)
  = $e(P, P)^{sz}$

- GET_SK(SK_M = s, “<keystring>”):
  SK = s * Hash_to_Point(”<keystring>”)
  = s * zP
  = szP

B & F's IBE Decryption

- IBE_ENC(M, PK = $e(P, P)^{sz}$):
  r = random int from Z_q
  C = <rP, M XOR H(PK^r)>
- IBE_DEC(C, SK = szP):
  $e(rP, szP) = e(P, P)^{szr}$
  Hash $e(P, P)^{szr}$, then XOR to recover M

Boneh, Crescenzo, Ostrovsky & Persiano

- Same scheme as Waters (independently discovered)
- Provides a real security model

Creating a Log Entry

E_K("mgreen searched for … 'Gas', 'Electricity', 'Water' … ")

IBE-ENC(PK("Gas"), <flag | K>)

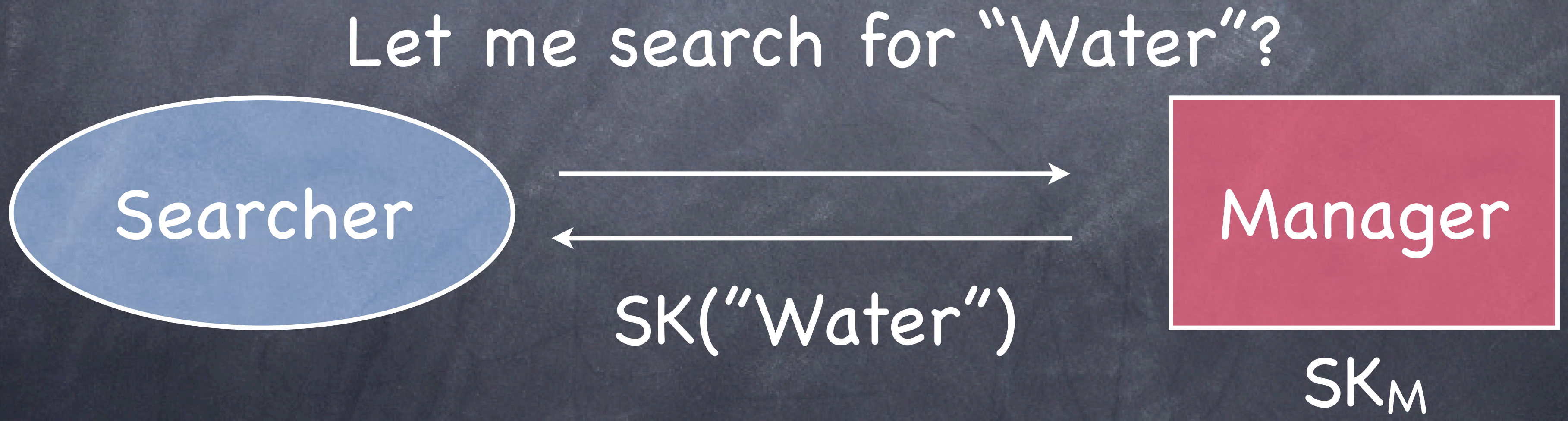
IBE-ENC(PK("Electricity"), <flag | K>)

IBE-ENC(PK("Water"), <flag | K>)

E_{PK}(K), H(this record || H(last record))

Searching, Step 1

- Contact the Trusted Manager for a search key on a particular term



Searching, Step 2

E_K("mgreen searched for ... 'Gas', 'Electricity', 'Water' ... ")

IBE_DEC SK("Water") →

IBE-ENC(PK("Gas"), <flag | K>)

IBE-ENC(PK("Electricity"), <flag | K>)

IBE-ENC(PK("Water"), <flag | K>)

E_{PK}(K) ...

Adding Time

- Simple approach: append a Time period to IBE keystrings, e.g.:

IBE-ENC(PK("Gas || 9-14-04"), <flag | K>)

- Searcher indicates time period when requesting IBE Secret Key
- Must still try all records

Caching IBE Public Keys

- To produce an IBE ciphertext, we generate an IBE Public Key.
  - Key Gen is the most expensive operation, requiring up to 175ms (that's per keyword!)
  - To save time, we could cache these keys for later reuse
- The downside: If an adversary captures this cache, they learn which keywords have been active recently

Batching Keywords

- n * m Keyword Ciphertexts
  n = total log entries
  m = average # of keywords per entry
- Log generation & Search time proportional
- Many common keywords will be repeated, can we be more efficient than?

Does Batching Help?

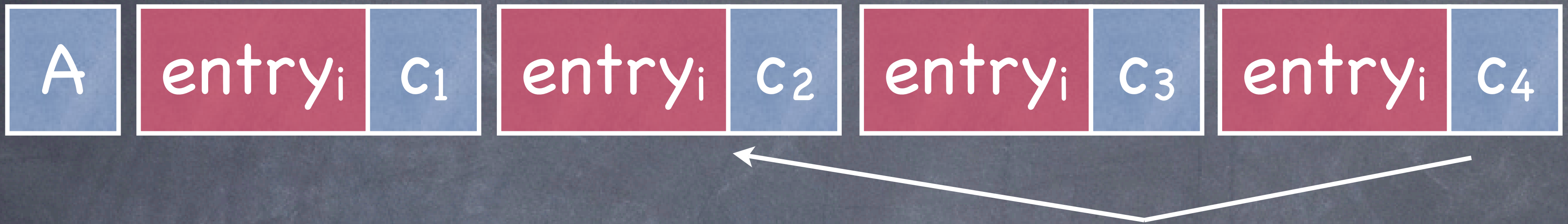
- Batching reduces the number of ciphertexts from (m)n to t, where t is total # of unique keywords in the block
  - Batching reduces waste for the most common keywords, but what about the uncommon ones?
  - Who searches on common words, anyway?

Block Batching Example

Entry 1 ... Entry 50

"water": 1,2,4 | k_1,k_2,k_4,k_{19}
"qas": 14, 20, 27 | k_{14},k_{20},k_{27}
"electricity": 3, 49 | k_3, k_{49}
"snorkles": 24 | k_{24}
"petunia": 4 | k_4
"spork": 33 | k_{33}

Davis, Monrose & Reiter



- Uses "backpointers" to link groups of keywords within a time period
- Advantages of batching, but doesn't keep the log open (unwritten) for long periods

Randomness Re-use

- To search a block of n keywords requires n pairing computations
  $C = \langle rP, M \text{ XOR } h(e(P, P)^{szr}) \rangle$
  $e(rP, SK(\text{"keyword"})) = e(P, P)^{szr}$
- We can reduce this if we re-use the same value r for each keyword in a batch

Randomness Re-use

- We can use <rP> for a group of ciphertexts, and only store the second term:

  c1 = <flag | k> XOR $h(e(P, P)^{rsz})$
  c2 = <flag | k> XOR $h(e(P, P)^{rsz'})$
  c3 = <flag | k> XOR $h(e(P, P)^{rsz''})$

- Only one pairing, but still have to XOR with many ciphertexts

A Slightly Better Approach

- PK("water") = e(sP, Hash_to_Point("water"))
  $= e(P, P)^{sz}$

- SK("water") = s * Hash_to_Point("water")) = szP

rP

$h'(e(P, P)^{szr})$	"water": 1,2,4 | k_1, k_2, k_4, k_{19}
...	"gas": 14, 20, 27 | k_{14}, k_{20}, k_{27}
...	"electricity": 3, 49 | k_3, k_{49}
...	"snorkles": 24 | k_{24}
...	"petunia": 4 | k_4
$h'(e(P, P)^{sz'r})$	"spork": 33 | k_{33}

Waters' Implementation

- Waters et al. implemented the IBE-based scheme to log SQL queries (MySQL Proxy)
- Used Stanford IBE Library, 1024-bit supersingular curves (q=160); AES 128-bit 2.8GHz Pentium IV
- Hash-chain integrity checking

Implementation: Optimizations Used

- IBE Public Key Caching:
  PK generation + encryption = 180ms
  encryption only (cached key) = 5ms
  100MB Cache -> ~800,000 Public Keys

  Webster's Dictionary: 300,000 words

- Randomness Re-use

Implementation: Ok, and...?

- Implementation reveals the pairing computation time, encryption time-- and not much else
- Is it practical? Where are your performance numbers and graphs? What data are you storing? Can we have the source code?

Open Problems

- Reducing storage & computational costs
- Better security models, reduced involvement of keyholder
- New approaches, or incremental improvements?

Other Problems

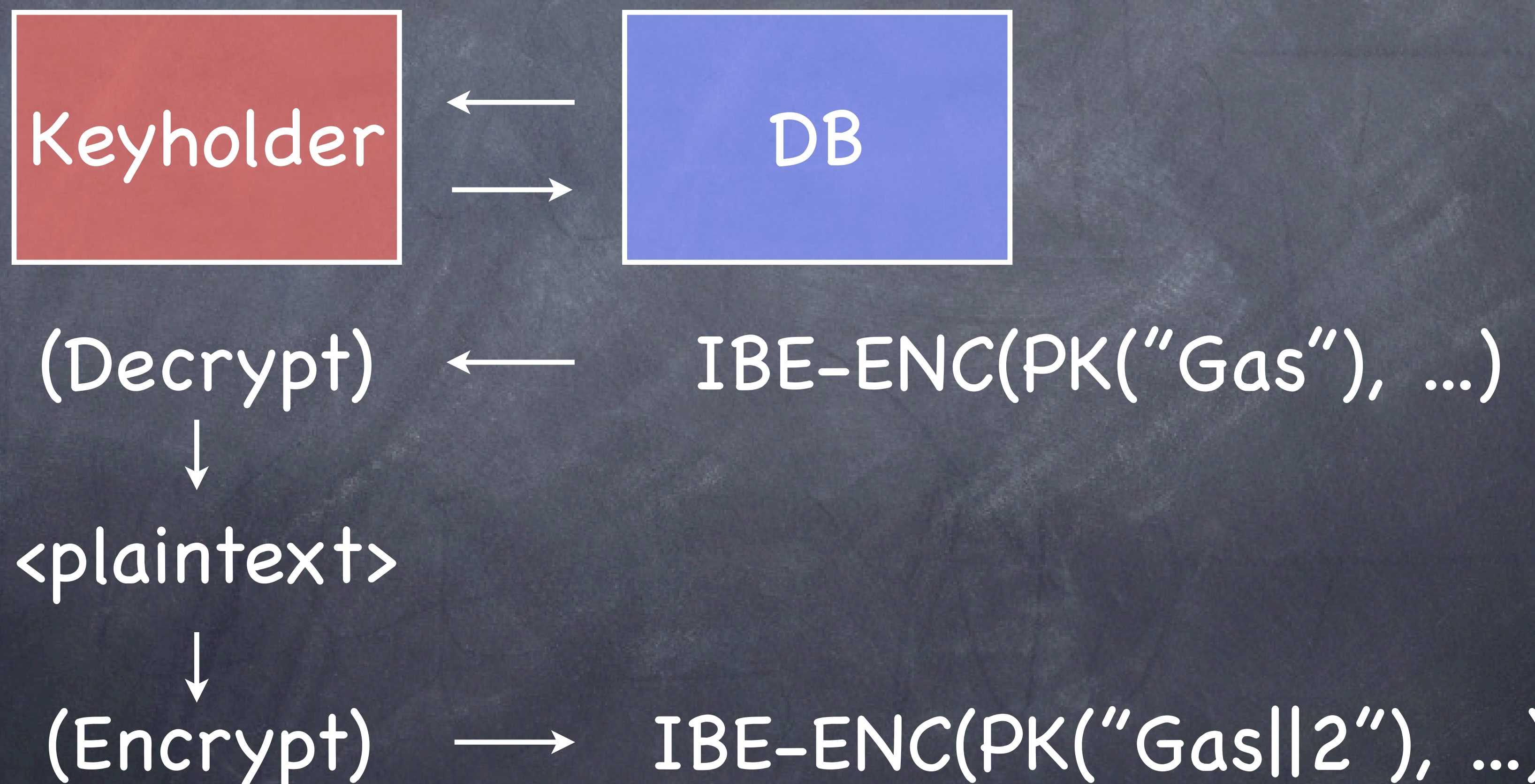
- In the Song scheme, all keywords in the document are searchable
- In the Goh scheme (and many others), relevant keywords chosen by data creator
  - Subtler concerns: What if keywords are not chosen correctly? What if data creator is malicious?

END

Revoking Search Keys

- We might want to revoke a search key after we've given it out
- A possible approach:
  - Re-encrypt all keywords under new IBE keys
  - e.g.: "Gas" -> "Gas || 2"

Revoking through Dumb Re-encryption



Revoking through Proxy Re-encryption?

IBE-ENC(PK("Gas"), ...)

↓

(Re-Encrypt)

↓

IBE-ENC(PK("Gas||2"), ...)

Trusted Computing Base

 = Fully Trusted

Waters et al. Symmetric-Key Scheme

E_K(“mgreen searched for … ‘Gas’, ‘Electricity’, ‘Water’ … ”)

h_S(“Gas”) XOR <flag | K>

h_S(“Electricity”) XOR <flag | K>

h_S(“Water”) XOR <flag | K>

Secret Key = S

Waters et al. Symmetric-Key Scheme

E_K("mgreen searched for … 'Gas', 'Electricity', 'Water' … "), r

$c1 = h_{a1}(r)$ XOR $\langle flag \mid K \rangle$

$c2 = h_{a2}(r)$ XOR $\langle flag \mid K \rangle$

$c3 = h_{a3}(r)$ XOR $\langle flag \mid K \rangle$

$a_1 = h_S$("Gas")

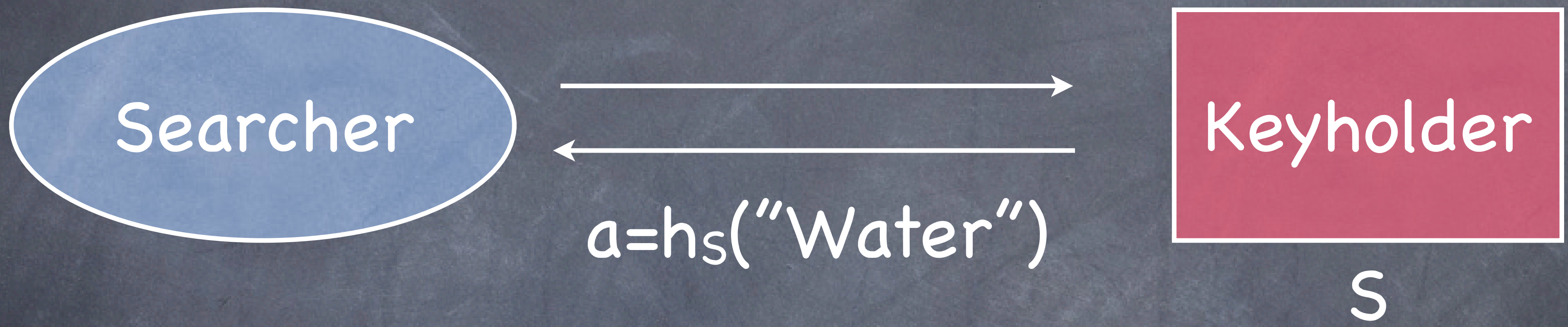
$a_2 = h_S$("Food")

$a_3 = h_S$("Water")

Master Secret = S

Symmetric, Searching

Let me search for "Water"?



c1 XOR a = "???"

c2 XOR a = <flag | key>

c3 XOR a = "???"

Reducing a Trusted Computing Base

Keyholder

Reducing a Trusted Computing Base