Making decryption accountable

Mark Ryan

HP Inc Labs, Bristol
Univ. of Birmingham

Secure Implementation of Cryptographic Software
Lesbos, Greece
August 2017

Twenty-fifth International Workshop on
Security Protocols (SPW'17)
21 March 2017

HotSpot at ETAPS
23 April 2017

Going out tonight?

- Teenager wants privacy
- Parent wants security

Investigatory Powers Act 2016


The Observer

Focus Surveillance and the net

A Bill which is slipping through the House of Lords will allow MI5 access to all our online communications, says **John Naughton**. It could mean we're all guilty until proven innocent. So why don't we care more?

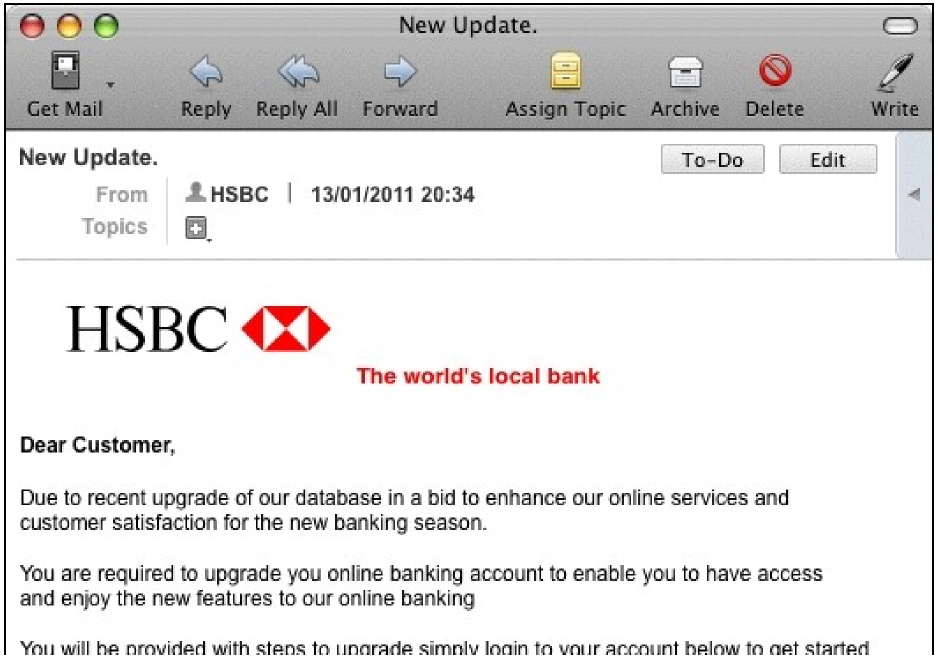
Your privacy ends here

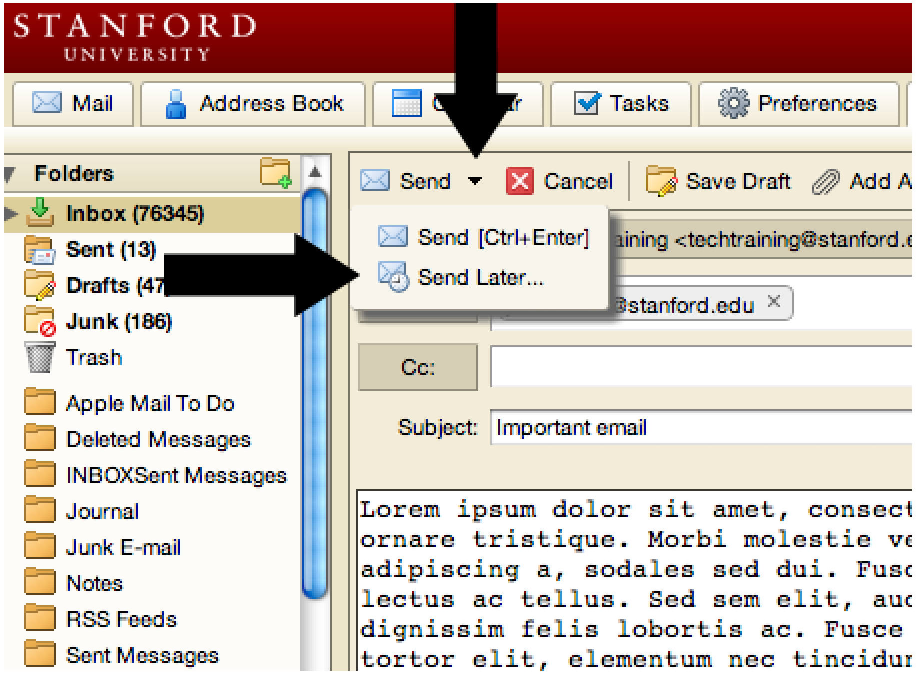
- Gives Gov wide-ranging snooping and interference powers
- Oversight is unverifiable
- Making decryption accountable is potentially a step towards verifiable oversight

Corporate email

- Corporation may need to access employee email

- But employees may expect some transparency





Mobile phone and IoT sensor data

- “Find my iphone” requires you to continuously send your location to Apple
  - You’d get to know when they decrypt it

- More generally, decryption accountability potentially enables detection of policy violations in IoT sensor data.

Electronic voting

- Voter's client software encrypts her vote, using a public key pk, and sends it to server.

- … mix nets … homomorphic combination … verification of zkps …

- The result is decrypted, using the secret key sk corresponding to pk.
  - We'd like to know that individual voters' votes are not decrypted.

Requirements

- Users create ciphertexts using a public key *pk.*
- Decrypting agent *Y* is capable of decrypting the ciphertexts *without any help from the users*.
- When *Y* decrypts ciphertexts, it unavoidably creates evidence *e* that is accessible to users. The evidence cannot be suppressed or discarded without detection.
- By examining *e*, users gain some information about the quantity and nature of the decryptions being performed.

This requires hardware

- If Y has a ciphertext and a decryption key, it is impossible to detect whether she applies the key to to ciphertext or not.
  - The decryption key has to be guarded by a hardware device D that controls its use.
- ***What is a minimal specification for D that will give us the desired properties?***
- Idea of this paper: propose a simple generic design that achieves the desired functionality.

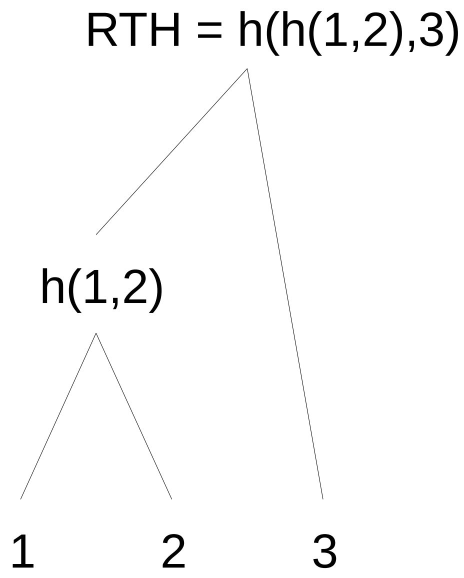
Core idea

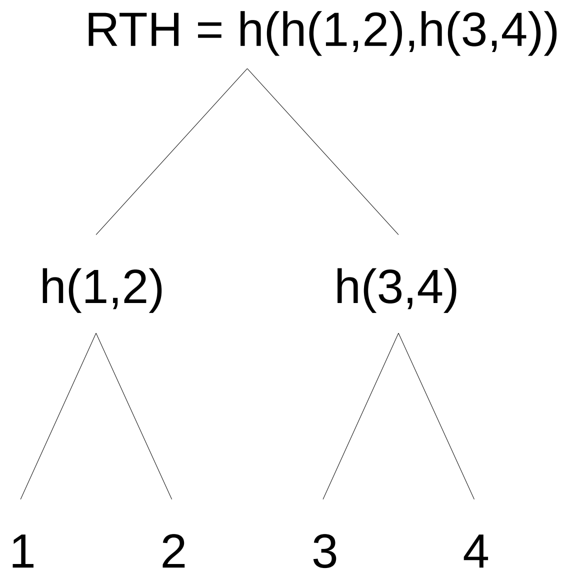
- There is a log L in which all decryption requests are recorded.
  - D will perform a decryption only if the request is accompanied by a proof that it has been entered into L.

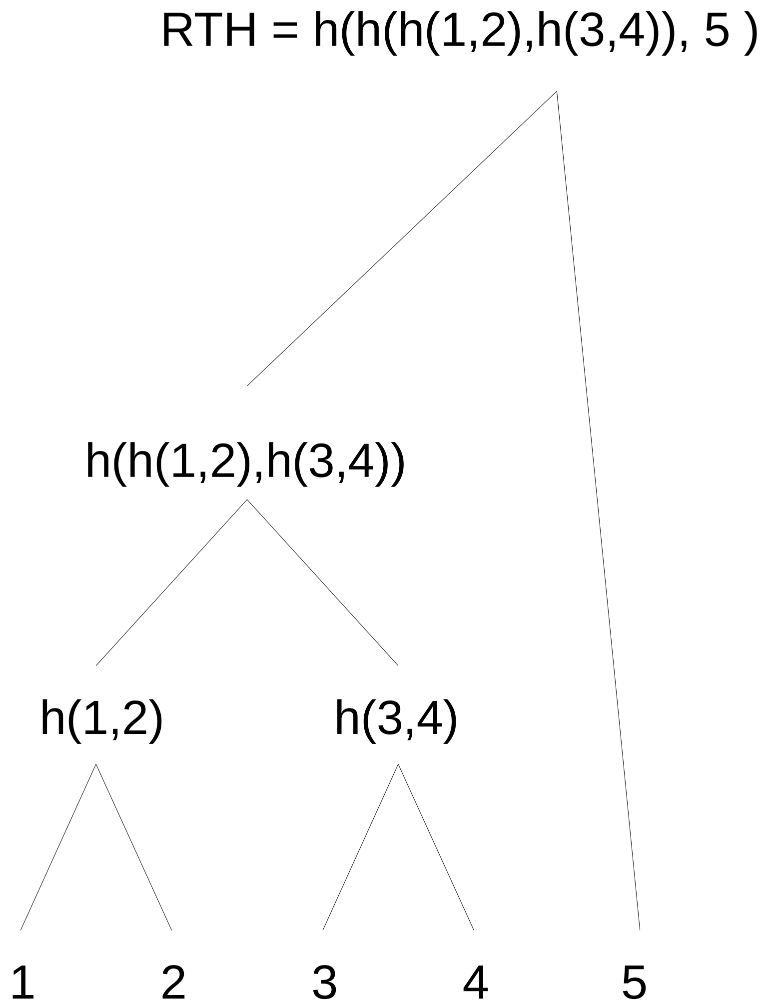
- Someone maintains L, but we minimise the requirement to trust that maintainer.
  - The maintainer of L is not required to be trusted w.r.t. ***integrity*** of L. If the maintainer cheats, e.g. by deleting/modifying entries from L, or by forking L, users can detect that.
  - The maintainer is required to be trusted for ***confidentiality***, so we design L so that confidentiality isn't required.

The log L

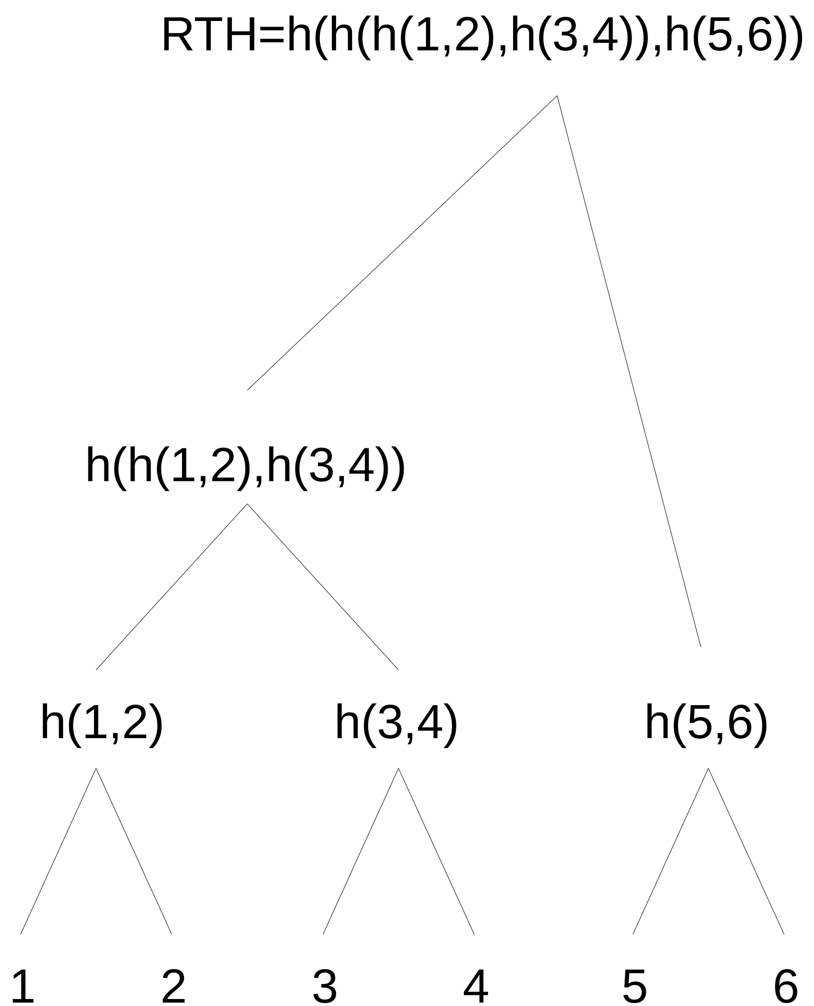
- The log L is organised as an append-only Merkle tree
  - as used in, for example, certificate transparency
- The maintainer periodically publishes the root tree hash (RTH) H of L

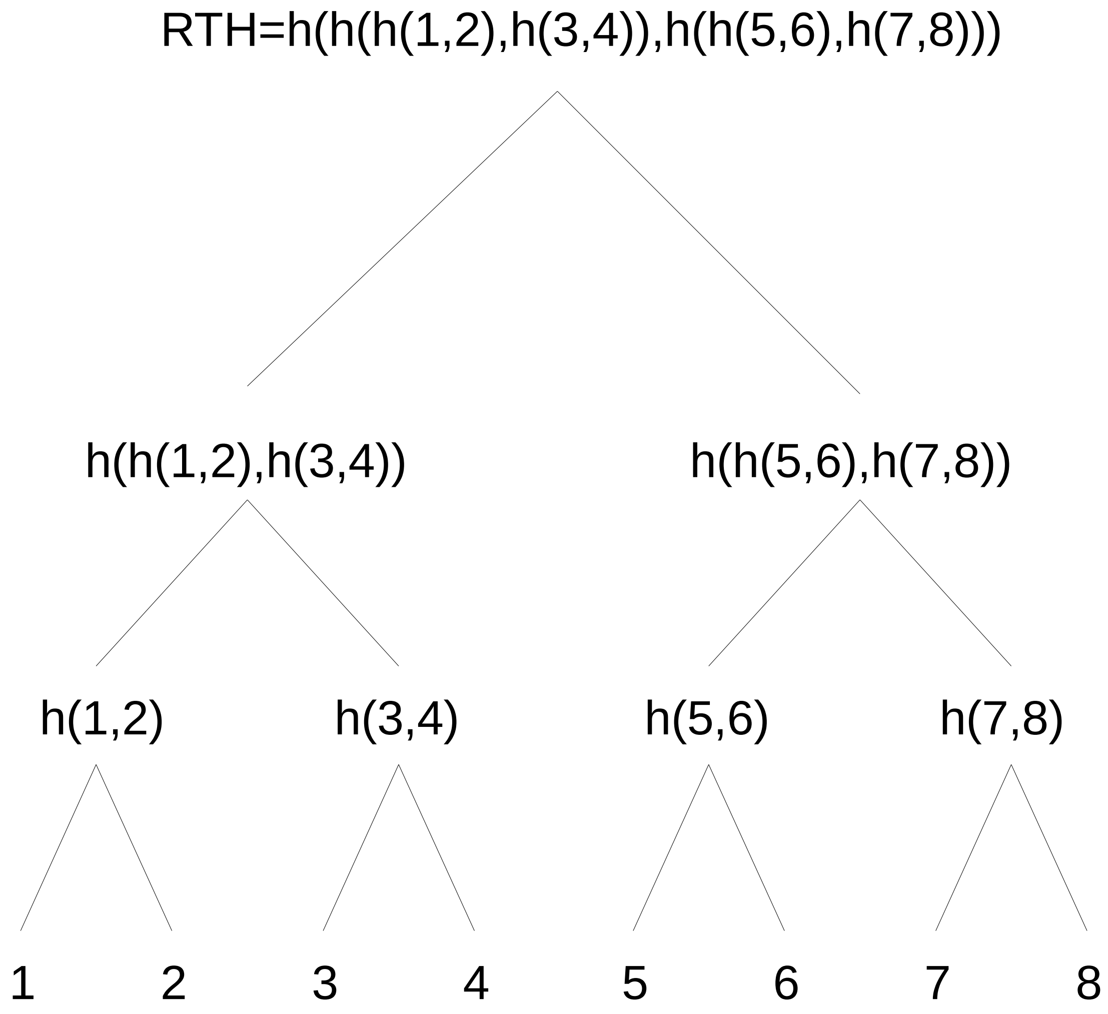

RTH = h(h(1,2),3)
h(1,2)
1
2
3

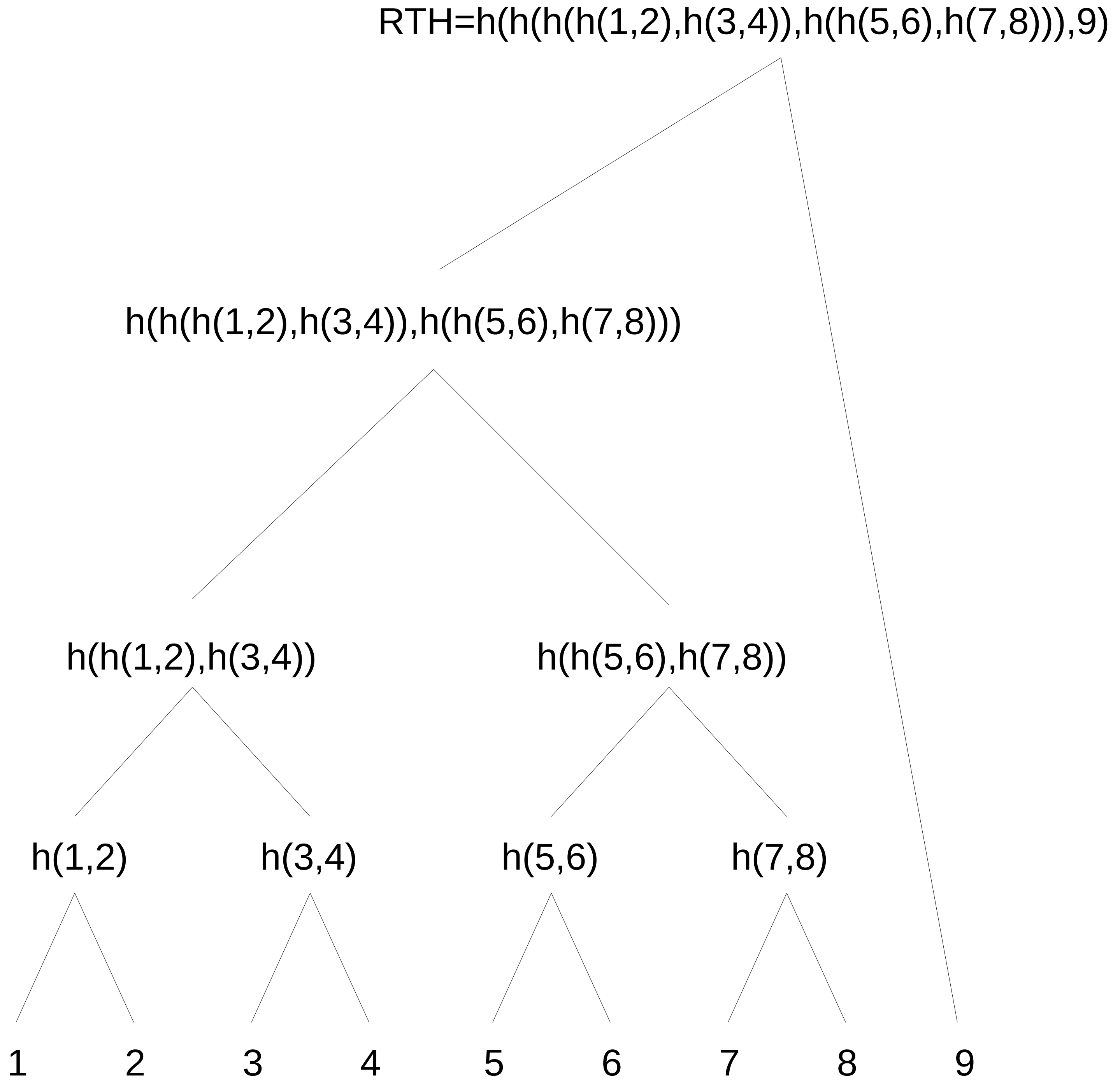

RTH = h(h(1,2),h(3,4))
h(1,2)
h(3,4)
1
2
3
4



RTH = h(h(h(1,2),h(3,4)), 5)
h(h(1,2),h(3,4))
h(1,2)
h(3,4)
1
2
3
4
5

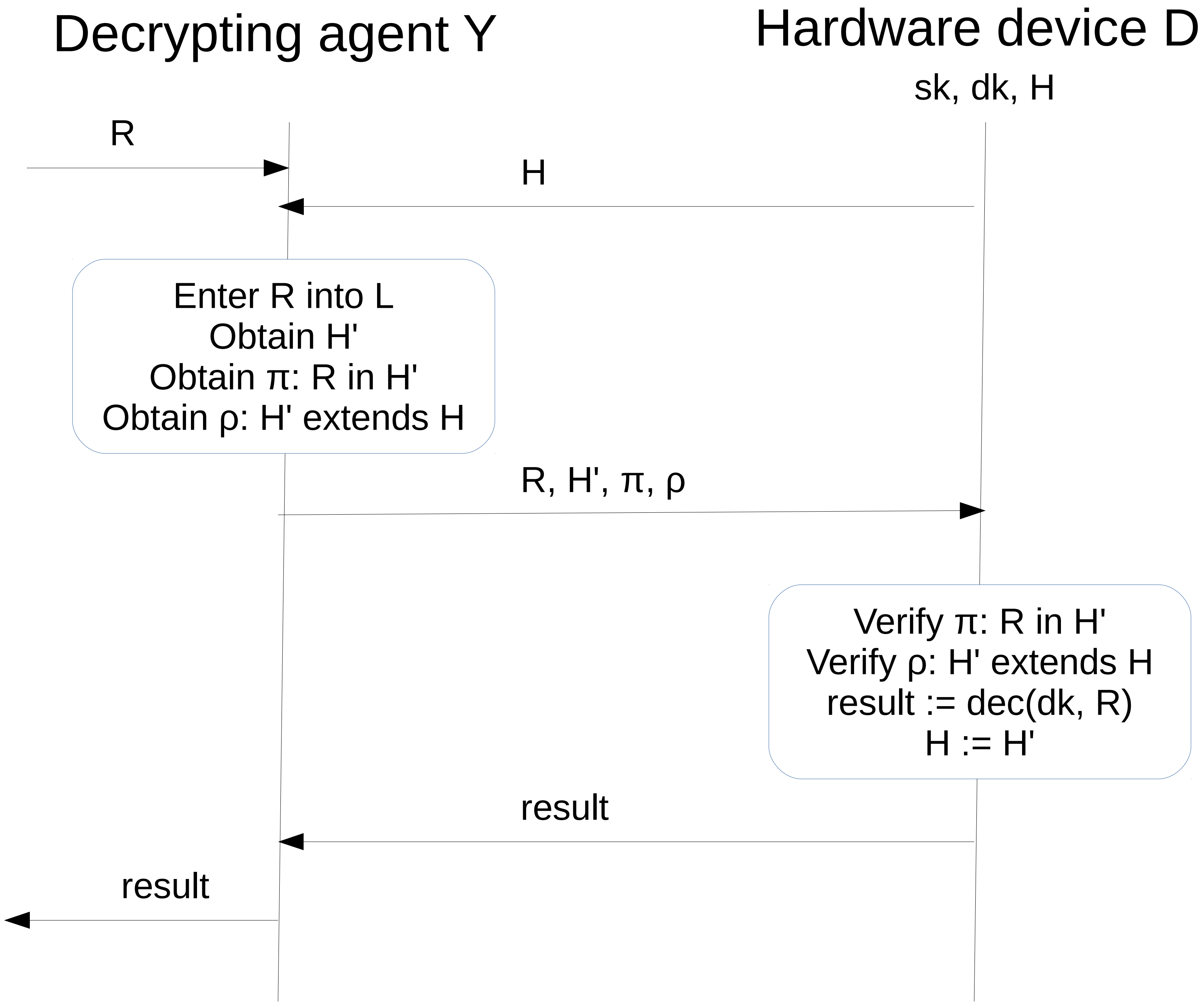

RTH=h(h(h(1,2),h(3,4)),h(5,6))
h(h(1,2),h(3,4))
h(1,2)
h(3,4)
h(5,6)
1
2
3
4
5
6


RTH=h(h(h(1,2),h(3,4)),h(h(5,6),h(7,8)))
h(h(1,2),h(3,4))
h(h(5,6),h(7,8))
h(1,2)
h(3,4)
h(5,6)
h(7,8)
1
2
3
4
5
6
7
8


RTH=h(h(h(h(1,2),h(3,4)),h(h(5,6),h(7,8))),9)
h(h(h(1,2),h(3,4)),h(h(5,6),h(7,8)))
h(h(1,2),h(3,4))
h(h(5,6),h(7,8))
h(1,2)
h(3,4)
h(5,6)
h(7,8)
1
2
3
4
5
6
7
8
9

The log L

- The log L is organised as an append-only Merkle tree
  - as used in, for example, certificate transparency
- The maintainer periodically publishes the root tree hash (RTH) H of L
- The maintainer is capable of generating two kinds of proof about the log's behaviour:
  - A proof π that some data item d is in the tree with RTH H
  - A proof ρ that the tree with RTH H' is an append-only extension of the tree with RTH H
- All the ops, incl gen and verif of proofs, are O(log n)



Decrypting agent Y
Hardware device D
sk, dk, H
R
H
Enter R into L
Obtain H'
Obtain π: R in H'
Obtain ρ: H' extends H
R, H', π, ρ
Verify π: R in H'
Verify ρ: H' extends H
result := dec(dk, R)
H := H'
result
result

Evidence of decryption in L

- Evidence about decryptions is obtained by inspecting L, which contains the decryption requests.
  - Example 1: L contains a hash of the ciphertext that is decrypted. This allows a user U to detect if ciphertexts she produced have been decrypted.
  - Example 2: L contains a unique value representing the decrypted ciphertext, but the value cannot be tied to a particular ciphertext (for example, the value could be the hash of a re-encryption). This allows users to see the number of ciphertexts decrypted, but not which particular ones.

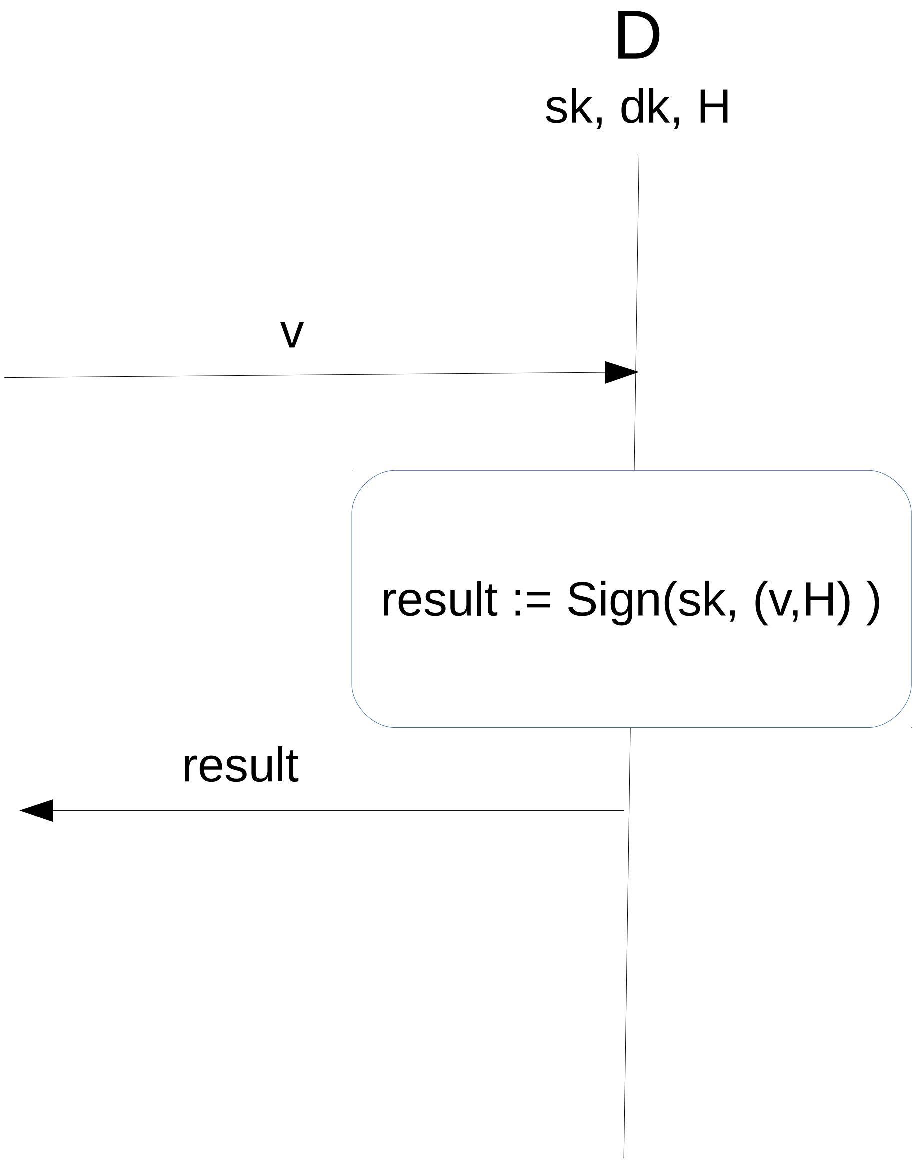
Insecure!

- The log provider could maintain two versions of the log:
  - The one it shows to users: it has no decryption requests in it, so users are happy
  - The one it shows to D: it has lots of decryption requests in it, so D decrypts a lot of data

- The users and D each verify that the version they see is maintained append-only. But they can't detect that they are different versions.

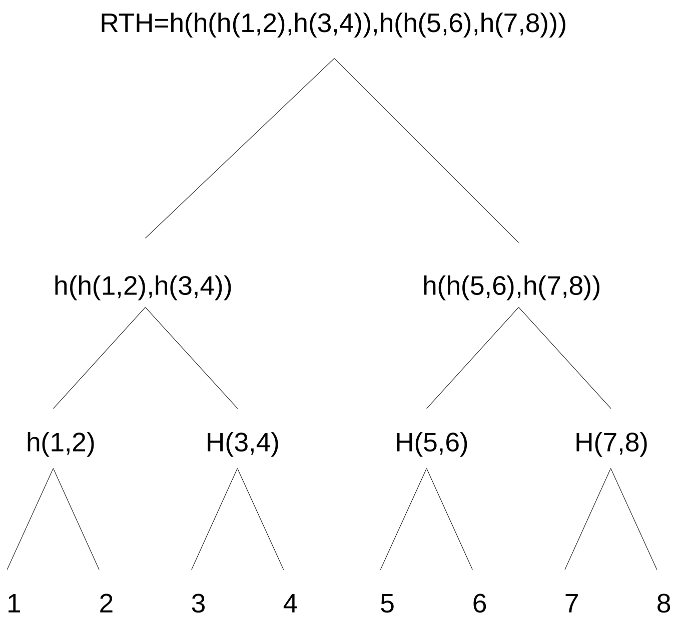
- The usual way of addressing this attack is “gossip protocols”.
  - Doesn't work here.

- To defeat the fork attack, we introduce a second protocol for D
- D periodically signs a *cryptographic beacon* v.
  - A cryptographic beacon is an *unpredictable* but *verifiable* value.
- Sign(sk, (v,H)) assures users that:
  D had RTH H at “time” v



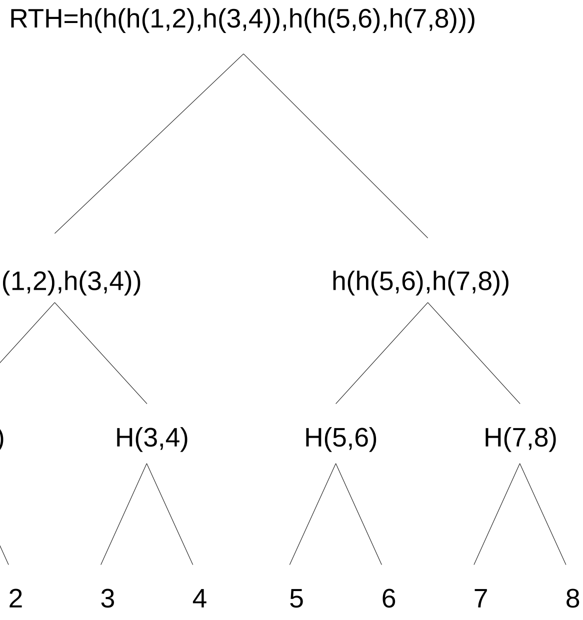
Verifiably unpredictable values

- We want to generate an unpredictable value which can be verified to have been generated after a given timepoint.
- One simple idea: everyone contributes a random value, and we hash all the values.



Verifiably unpredictable values

- We want to generate an unpredictable value which can be verified to have been generated after a given timepoint.
- One simple idea: everyone contributes a random value, and we hash all the values.
- Another idea: *cryptographic beacons* e.g., based on stock market indices



Proposal: a device D with two protocols

D stores: H, dk, sk

- Input: R, H', π, ρ
- Compute:
  - Verify π: R in H'
  - Verify ρ: H' extends H
  - result := dec(dk, R)
  - H := H'
- Output: result

- Input: v
- Compute
  - Result := Sign(sk, (v,H))
- Output result

Conclusion

- The decrypting agent has no way to decrypt data without leaving evidence in the log, unless it can break the hardware device D.
- Who manufactures D?
  - How can the relying parties (both users U1 . . . and decrypting agents Y) be assured that it will behave as specified?
- One idea is that it is jointly manufactured by an international coalition of companies with a reputation they wish to maintain.