CMSC/Math 456: Cryptography (Fall 2023)

Lecture 21

Daniel Gottesman

Administrative

Problem set #8 (a programming assignment) is out, due next Thursday at noon. Problem set #7 was due earlier today.

This class is being recorded

Hash Functions

A hash function H(x) maps the input x to a shorter string. The main cryptographic property of hash functions is collision resistance: it is hard to find a pair x, x' such that H(x) = H(x').

But sometimes we also want to consider weaker or stronger properties of hash functions. One such property is that sometimes we abstract a hash function into a random oracle.

A random oracle is just a random function to which we have only black-box access.

When we say a hash function can be modeled as a random oracle, we are saying that it has no exploitable structure.

This class is being recorded

Hash Functions for MACs

Uses the property of collision resistance:

Given a hash function H(x) and a fixed-size MAC Mac(k,m), we can make a new MAC:

$$\mathrm{Mac}'(k, m) = \mathrm{Mac}(k, H(m))$$

This lets us efficiently authenticate long messages with short tags and keys.

If Eve can't find a collision, to forge a new message she will have to produce a new value of H(m), which in turn requires a new tag.

This class is being recorded

Hash Functions for Key Derivation

Uses random oracle model:

Suppose Alice and Bob do a public key protocol or have some other method to derive a key k which is not uniformly distributed. Or perhaps Eve has learned partial information about the key k (so from Eve's point of view, k is no longer uniformly random).

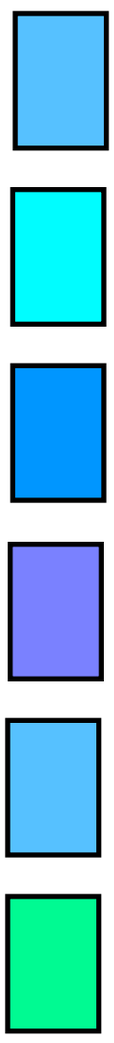
If H is a random oracle, then H(k) is close to uniformly random as long as k has a significant random element to its distribution.

Each possible value of k gives an uncorrelated value of H(k). We can measure randomness via entropy (in this case, min-entropy). The random oracle preserves the entropy but concentrates it into fewer bits.

This class is being recorded

Hash Functions for Fingerprinting

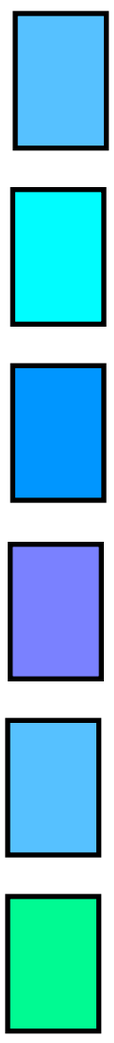
We have a long list of files. How can we determine if two of them are the same?



This class is being recorded

Hash Functions for Fingerprinting

We have a long list of files. How can we determine if two of them are the same?

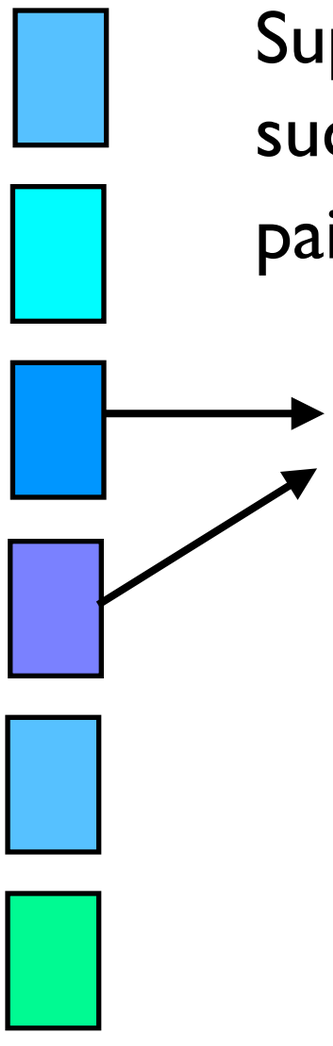


Suppose we have n files A_i each of length L. Find (i,j) such that $A_i = A_j$ or determine that there is no such pairs.

This class is being recorded

Hash Functions for Fingerprinting

We have a long list of files. How can we determine if two of them are the same?



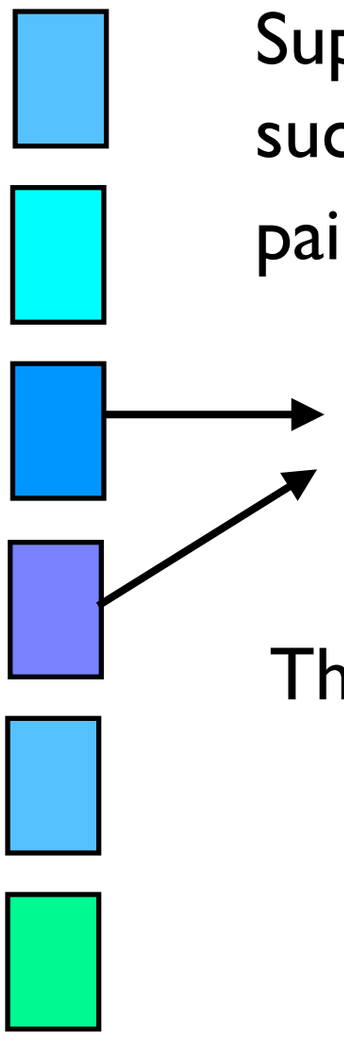
Suppose we have n files A_i each of length L. Find (i,j) such that $A_i = A_j$ or determine that there is no such pairs.

Comparing a single pair of files directly, character by character, takes time 2L.

This class is being recorded

Hash Functions for Fingerprinting

We have a long list of files. How can we determine if two of them are the same?



Suppose we have n files A_i each of length L. Find (i,j) such that $A_i = A_j$ or determine that there is no such pairs.

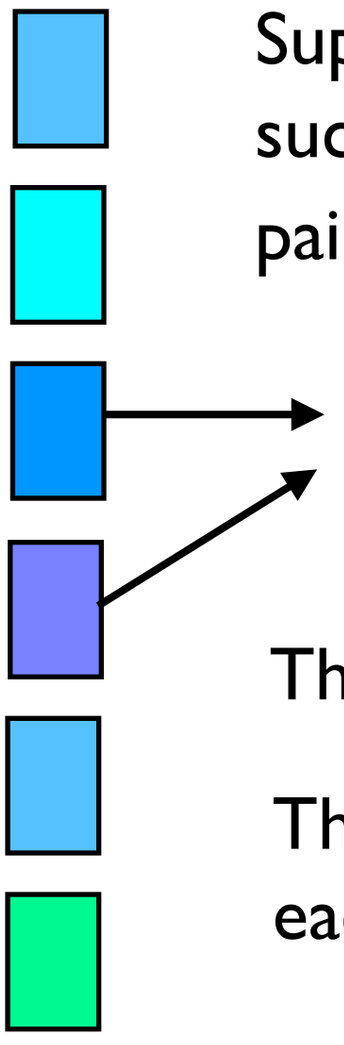
Comparing a single pair of files directly, character by character, takes time 2L.

There are $O(n^2)$ pairs of files to check.

This class is being recorded

Hash Functions for Fingerprinting

We have a long list of files. How can we determine if two of them are the same?



Suppose we have n files A_i each of length L. Find (i,j) such that $A_i = A_j$ or determine that there is no such pairs.

Comparing a single pair of files directly, character by character, takes time 2L.

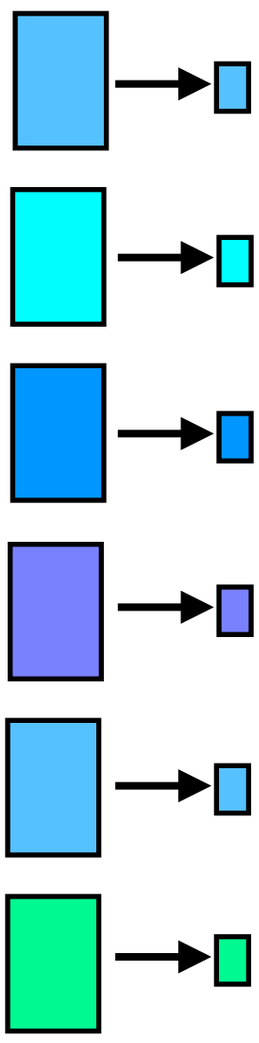
There are $O(n^2)$ pairs of files to check.

Thus, directly comparing every character between each pair of files takes time $O(n^2 L)$.

This class is being recorded

Hash Functions for Fingerprinting

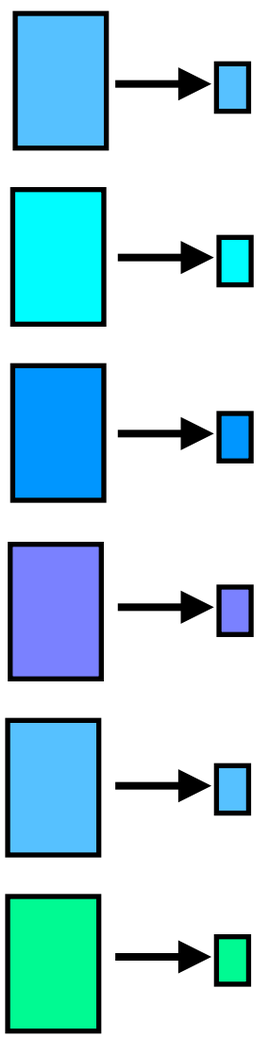
We can do better using hash functions because of the collision resistance property:



This class is being recorded

Hash Functions for Fingerprinting

We can do better using hash functions because of the collision resistance property:

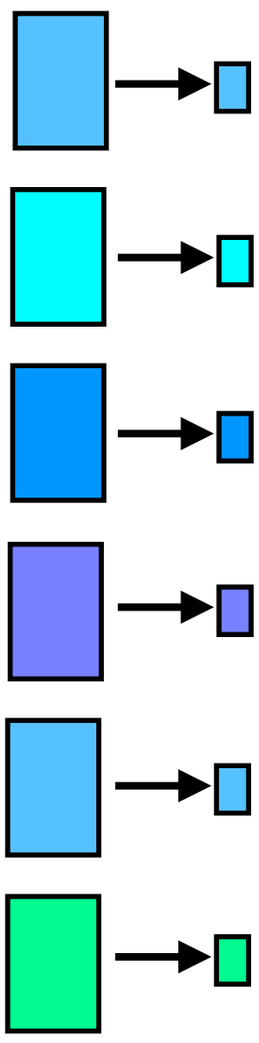


Suppose we first calculate a hash $h_i = H(A_i)$ of each file.
Each hash value h_i has length $s < L$.

This class is being recorded

Hash Functions for Fingerprinting

We can do better using hash functions because of the collision resistance property:



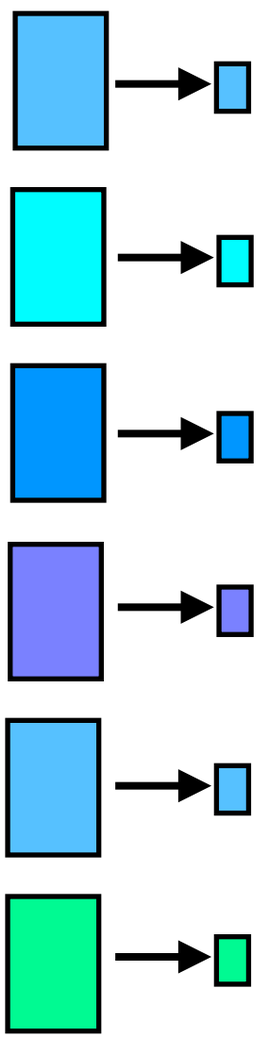
Suppose we first calculate a hash $h_i = H(A_i)$ of each file.
Each hash value h_i has length $s < L$.

Comparing two hash values takes only time 2s.

This class is being recorded

Hash Functions for Fingerprinting

We can do better using hash functions because of the collision resistance property:



Suppose we first calculate a hash $h_i = H(A_i)$ of each file.
Each hash value h_i has length $s < L$.

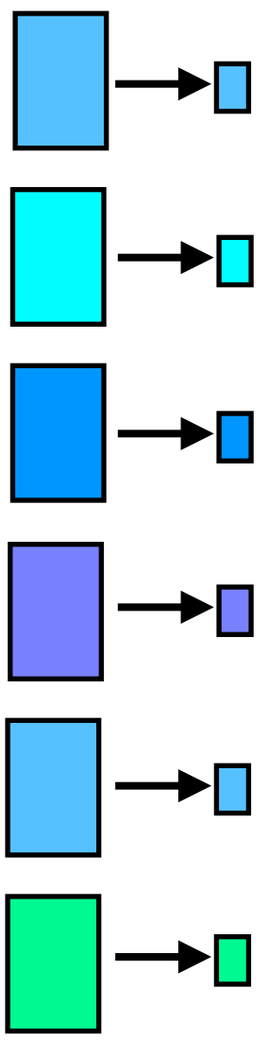
Comparing two hash values takes only time $2s$.

Computing the hash values takes time $O(nL)$ (or possibly $O(nL^a)$ if the hash is not very fast to compute).

This class is being recorded

Hash Functions for Fingerprinting

We can do better using hash functions because of the collision resistance property:



Suppose we first calculate a hash $h_i = H(A_i)$ of each file.
Each hash value h_i has length $s < L$.

Comparing two hash values takes only time 2s.

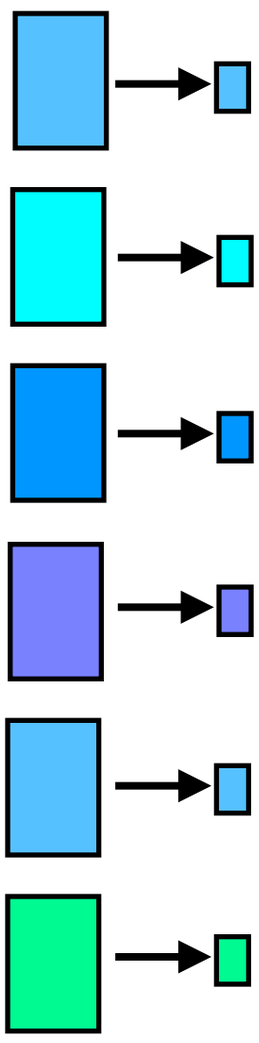
Computing the hash values takes time $O(nL)$ (or possibly $O(nL^a)$ if the hash is not very fast to compute).

The total time is then $O(n^2 s + nL)$, which is less than $O(n^2 L)$ when n is large.

This class is being recorded

Hash Functions for Fingerprinting

We can do better using hash functions because of the collision resistance property:



Suppose we first calculate a hash $h_i = H(A_i)$ of each file. Each hash value h_i has length $s < L$.

Comparing two hash values takes only time 2s.

Computing the hash values takes time $O(nL)$ (or possibly $O(nL^a)$ if the hash is not very fast to compute).

The total time is then $O(n^2 s + nL)$, which is less than $O(n^2 L)$ when n is large.

This algorithm fails if two different documents A_i and A_j have the same hash value $H(A_i) = H(A_j)$ — which would be a collision in the hash function. A collision is automatic if $s < \log n$.

This class is being recorded

Applications of Fingerprinting

File fingerprinting has many applications:

- Data deduplication: Identify extra copies of a file already stored in the system.
- Virus scanning: Identify if newly received files exactly match any known malware.
- Detect copyright violation or plagiarism: Identify an exact copy of anything from a list of works.
- Track sensitive information: Keep track of files containing sensitive information such as medical records to be sure they aren't accidentally sent somewhere insecure.

But it is not very good for the virus scanning or copyright violation applications: *any* change in the file will produce a different hash.

These applications don't usually require cryptographic strength hash functions.

This class is being recorded

Hash Functions for Password Files

Uses random oracle model:

When you enter a password into a computer, how does the computer know if the password is correct or not?

This class is being recorded

Hash Functions for Password Files

Uses random oracle model:

When you enter a password into a computer, how does the computer know if the password is correct or not?

The computer stores a list of everyone's passwords.

This class is being recorded

Hash Functions for Password Files

Uses random oracle model:

When you enter a password into a computer, how does the computer know if the password is correct or not?

The computer stores a list of everyone's passwords.

But if a hacker gets access to this list, everyone's password is compromised.

Password files are a prime hacker target.

This class is being recorded

Hash Functions for Password Files

Uses random oracle model:

When you enter a password into a computer, how does the computer know if the password is correct or not?

The computer stores a list of everyone's passwords.

But if a hacker gets access to this list, everyone's password is compromised.

Password files are a prime hacker target.

Instead, store hashes of the passwords. This:

- Is more efficient
- Conceals the file contents unless the attacker can invert the hash function

So what we need is for the hash function to be a one-way function (easy to compute, hard to invert).

This class is being recorded

Salt for Password Files

Since the hash function itself is known, one attack is to preprocess by making a list of hashes of common passwords.

> Unfortunately, many people's passwords are somewhat weak. With pre-computed hashes, the hacker can compare each hash to every password in the file.

This class is being recorded

Salt for Password Files

Since the hash function itself is known, one attack is to preprocess by making a list of hashes of common passwords.

> Unfortunately, many people's passwords are somewhat weak. With pre-computed hashes, the hacker can compare each hash to every password in the file.

To foil this attack, passwords are normally hashed with a unique random salt:

Password x

Password file stores (username, salt, H(salt || x))

This class is being recorded

Salt for Password Files

Since the hash function itself is known, one attack is to preprocess by making a list of hashes of common passwords.

> Unfortunately, many people's passwords are somewhat weak. With pre-computed hashes, the hacker can compare each hash to every password in the file.

To foil this attack, passwords are normally hashed with a unique random salt:

> Password x
>
> Password file stores (username, salt, H(salt || x))

When the user types in their password, the system retrieves the salt for this specific user and computes the hash H(salt || x) to compare with the password file.

This class is being recorded

Salt for Password Files

Since the hash function itself is known, one attack is to preprocess by making a list of hashes of common passwords.

> Unfortunately, many people's passwords are somewhat weak. With pre-computed hashes, the hacker can compare each hash to every password in the file.

To foil this attack, passwords are normally hashed with a unique random salt:

> Password x
>
> Password file stores (username, salt, H(salt || x))

When the user types in their password, the system retrieves the salt for this specific user and computes the hash H(salt || x) to compare with the password file.

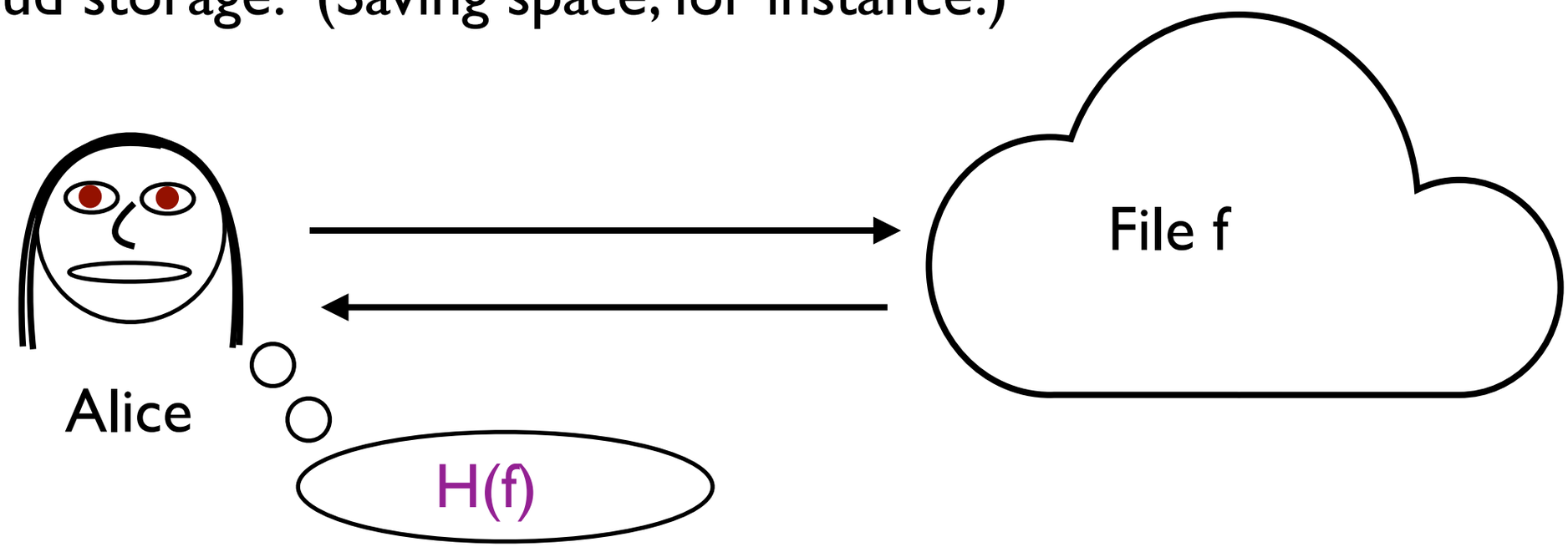
An attacker can do the same thing for common passwords, but has to do so *separately* for each user since their salts are different.

This class is being recorded

Hash Functions for Verifying Files

Suppose you want to store some files on a cloud server but you want to be able to verify that the files haven't been corrupted when you retrieve them.

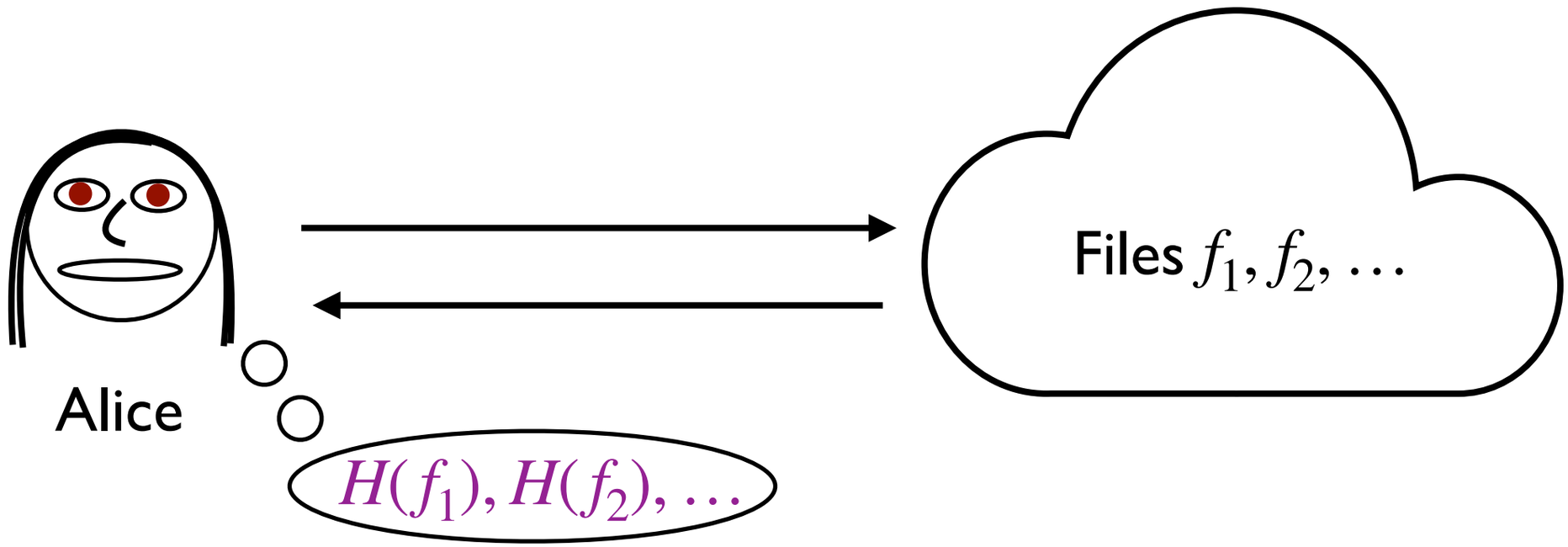
Storing the original file would remove the point of using the cloud storage. (Saving space, for instance.)



But Alice can locally store a fingerprint H(f), which is much shorter, and verify the file easily once it is retrieved.

This class is being recorded

Storing Many Files

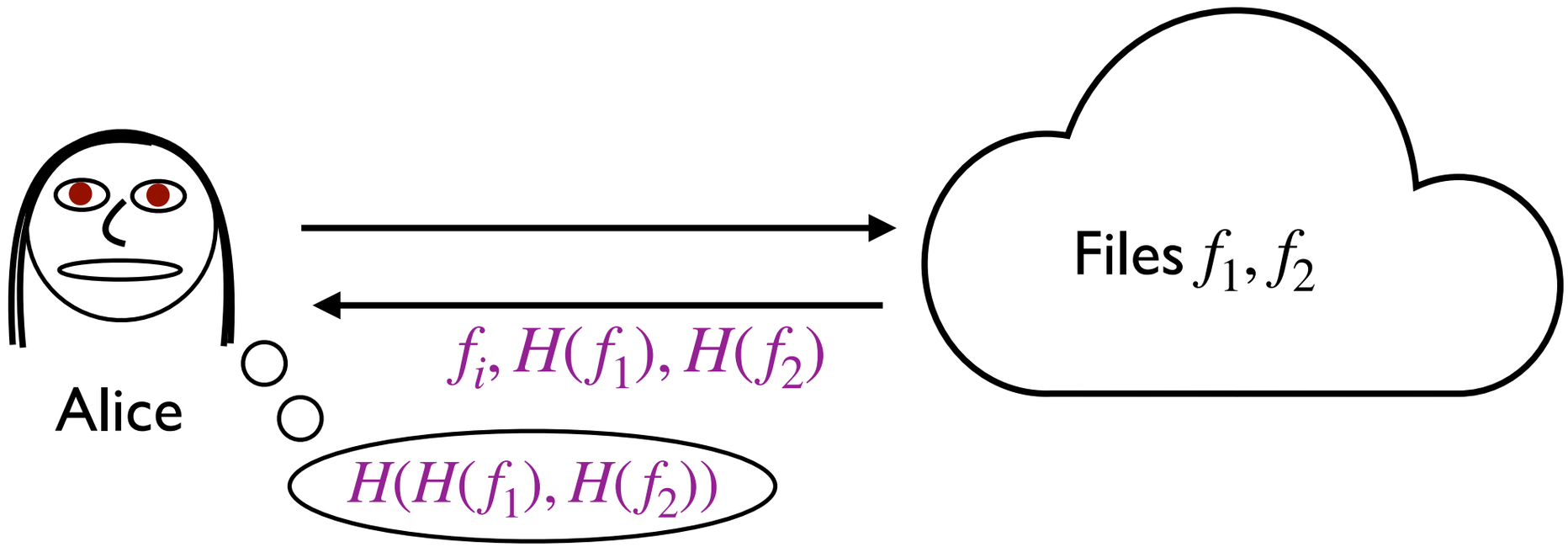


But what if Alice wants to store many files on the cloud and have the ability to check any of them?

She could store the fingerprint of each file, but that starts to get large. If there are n files, she would be storing O(n) bits.

This class is being recorded

Storing Two Files



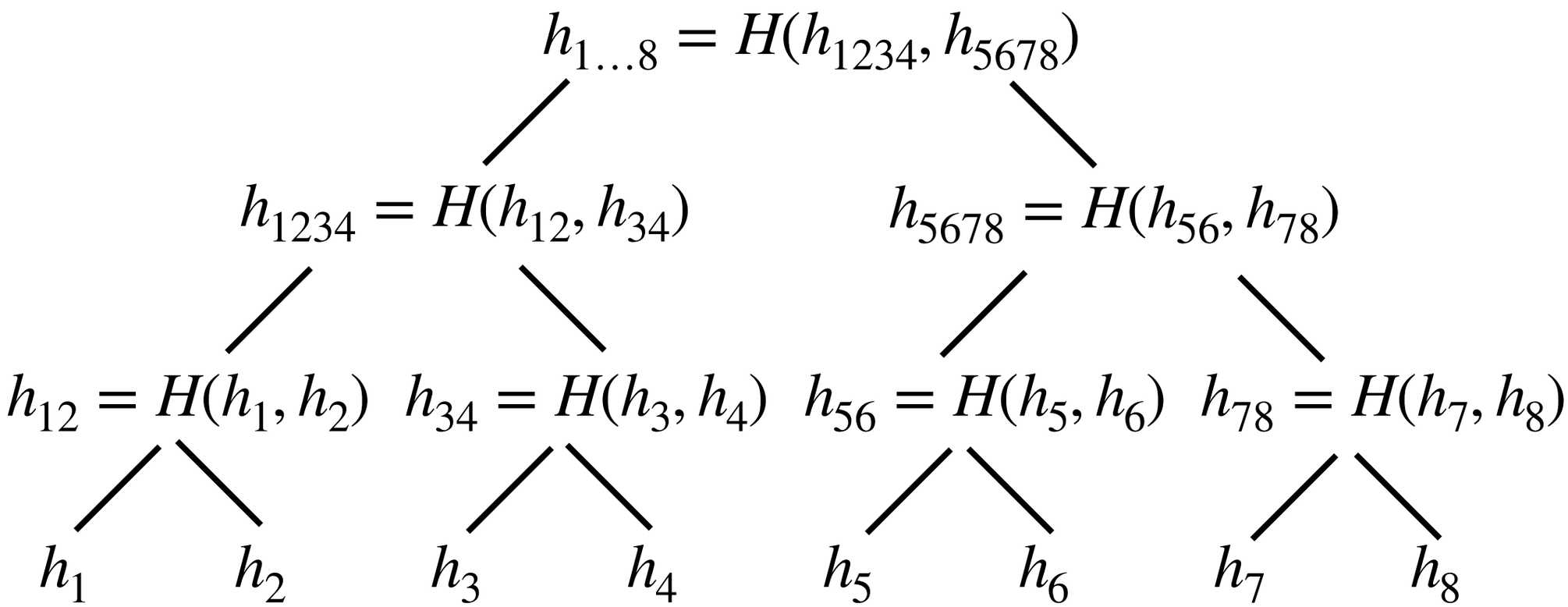
If Alice wants to store just 2 files on the cloud, she can save space by keeping just $H(H(f_1), H(f_2))$. When she retrieves a file f_i, she can also ask the cloud for hashes $H(f_1), H(f_2)$ and whichever file she wanted.

To verify, Alice computes $H(f_i)$ and verifies the hash value. Then she also computes $H(H(f_1), H(f_2))$.

This class is being recorded

Storing Many Files Efficiently

Now suppose Alice is storing n files on the cloud.

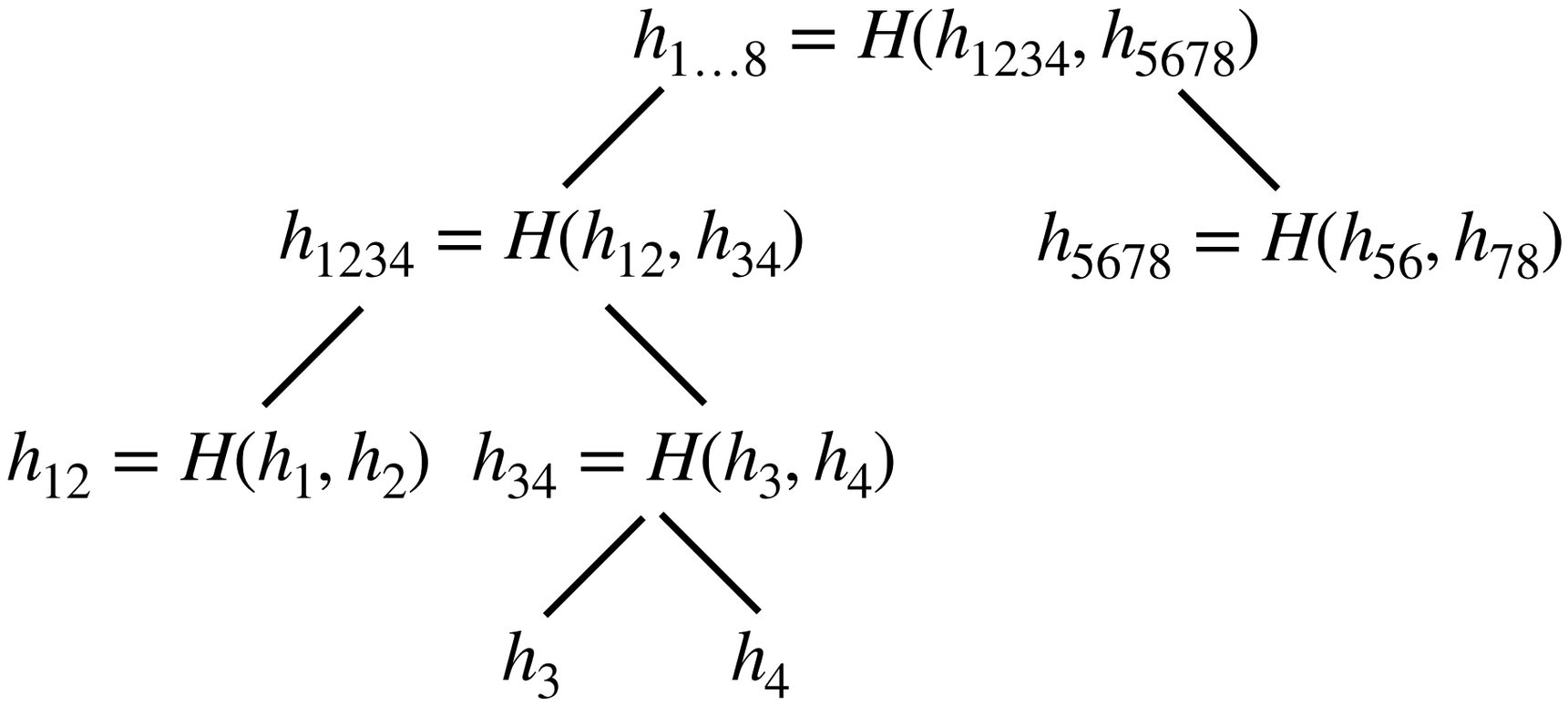


$$h_i = H(f_i)$$

She can initially compute hashes in a tree structure and store only a single hash, the root of the tree ($h_{1\ldots8}$ in the example).

This class is being recorded

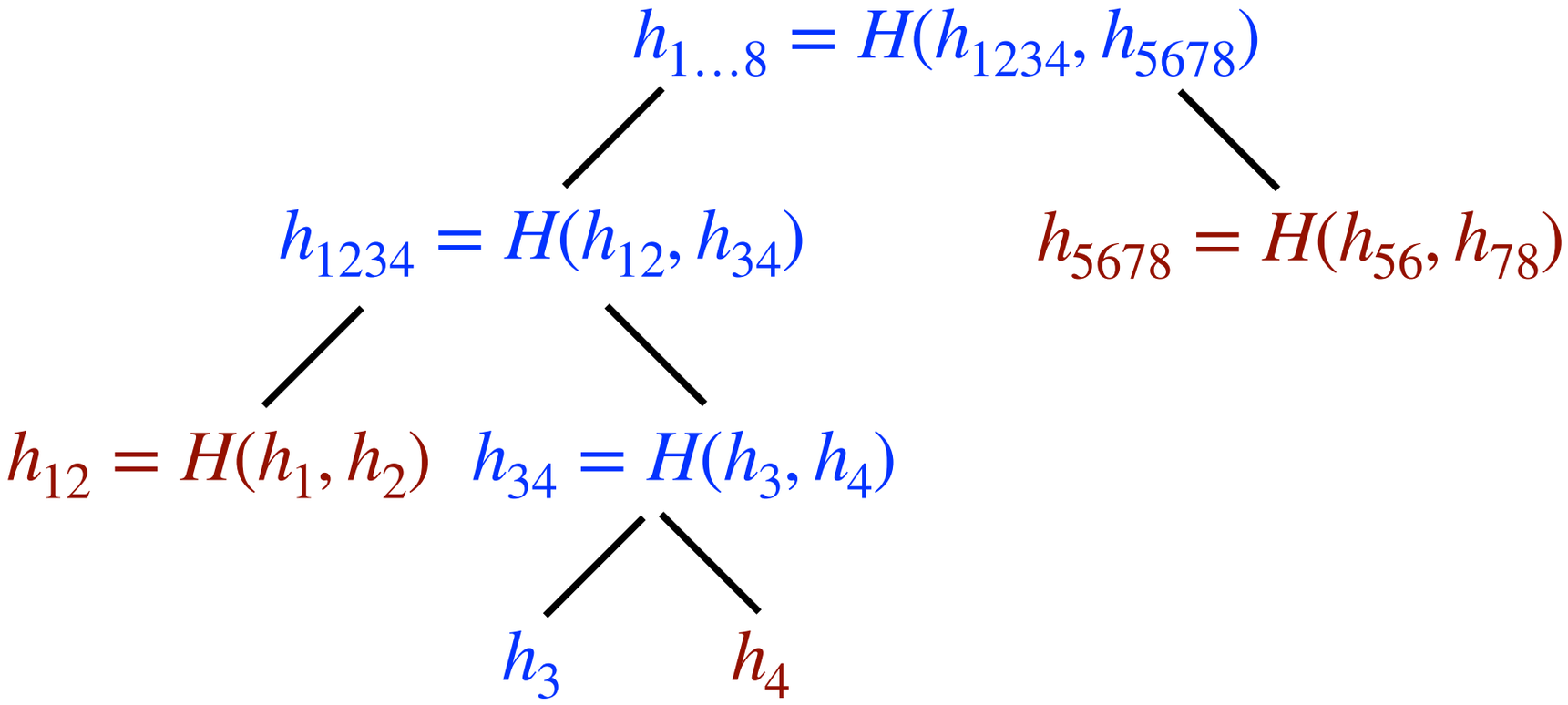
Retrieving One of Many Files



When Alice wishes to retrieve a file, she asks for the file f_i and the cloud server returns it, along with $h_i = h(f_i)$, and all of the hashes above it in the tree. For each of those hashes, the cloud server also returns the other child so that Alice can verify all the needed hashes.

This class is being recorded

Retrieving One of Many Files



That is, in this example, Alice asks for the file f_3, and the server returns

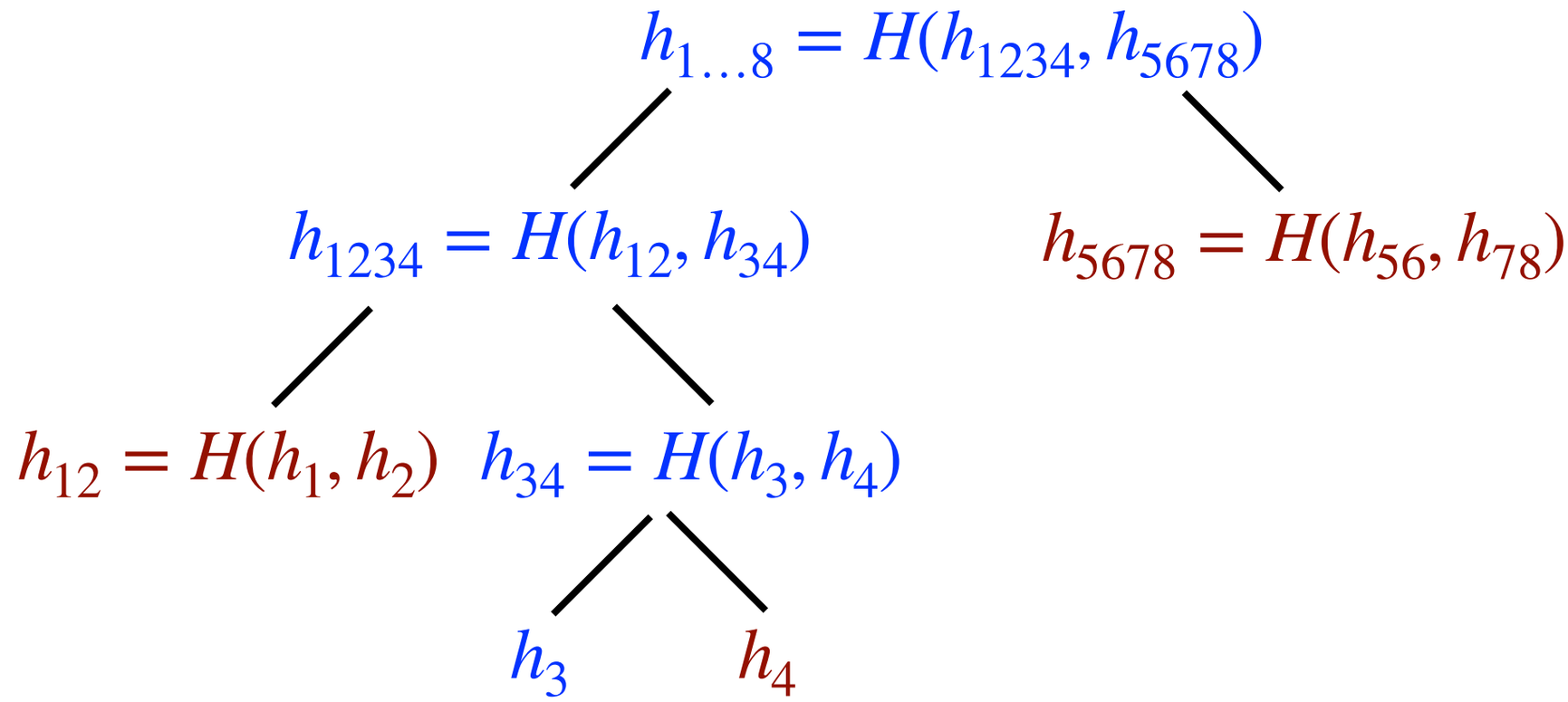
$$(f_3, h_4, h_{12}, h_{5678})$$

Alice can then compute h_3, h_{34}, h_{1234}, and then $h_{1\ldots8}$, which she compares with her stored value for $h_{1\ldots8}$.

This class is being recorded

Security of File Storage

Alice's concern is that the file she retrieves is identical to the one she originally stored.

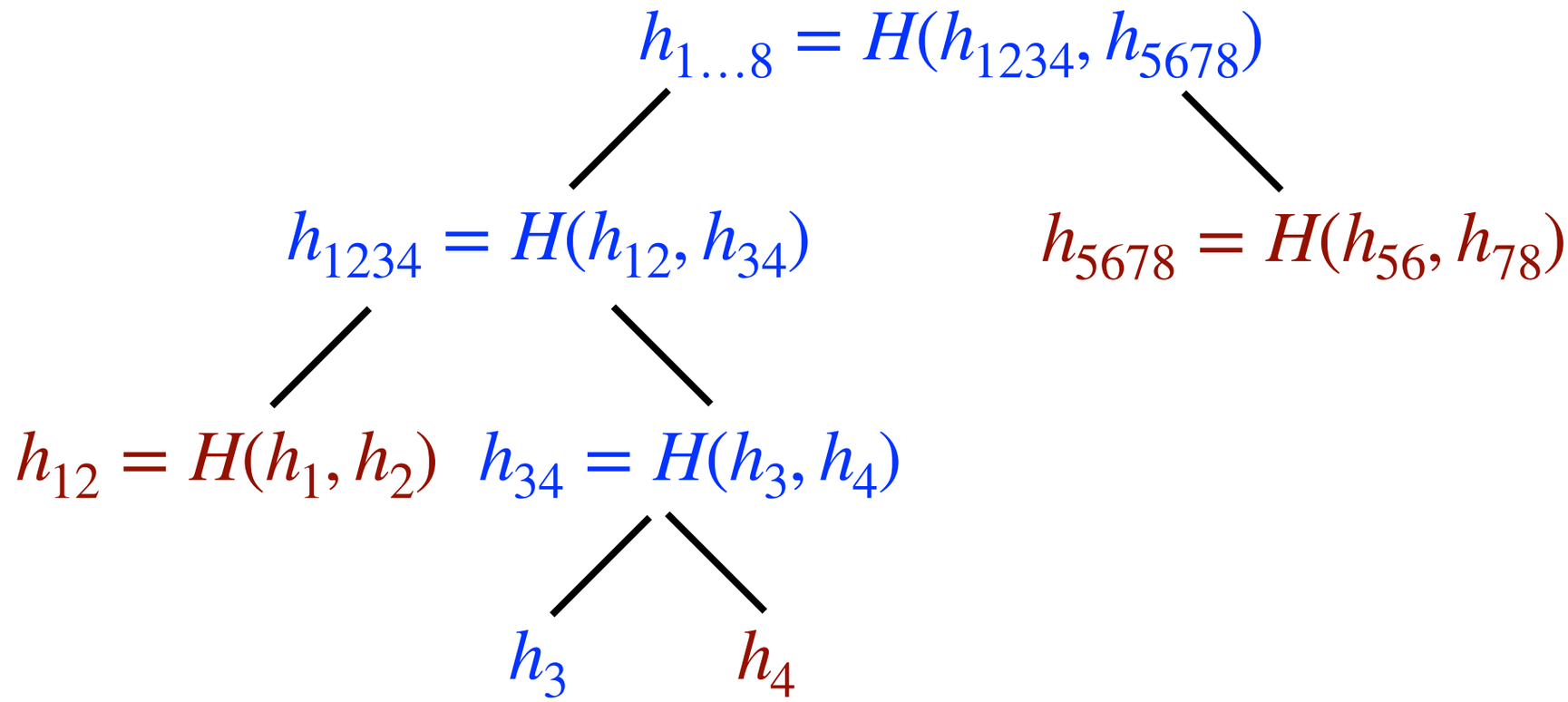


If the cloud server wants to change the file f_3, it will need to find a collision for one of the hash values h_3, h_{34}, h_{1234}, or $h_{1\ldots8}$. Otherwise, the only way to match $h_{1\ldots8}$ is to send the correct h_{5678} and to return values consistent with the correct h_{1234}.

This class is being recorded

Security of File Storage

Alice's concern is that the file she retrieves is identical to the one she originally stored.

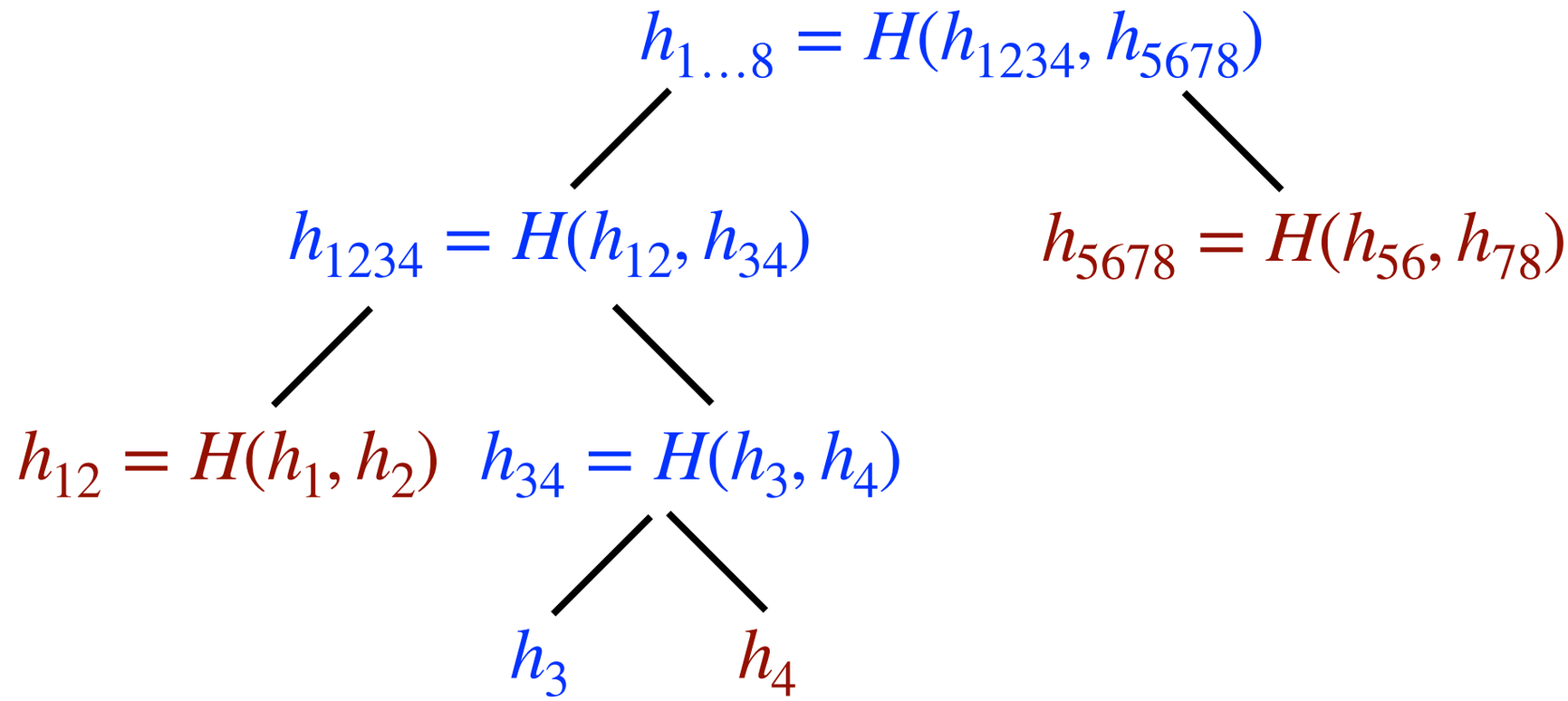


But if h_{1234} is going to have the correct value and there is no collision, h_{12} and h_{34} both must be correct also, which means that h_3 and h_4 must also have the correct values …

This class is being recorded

Security of File Storage

Alice's concern is that the file she retrieves is identical to the one she originally stored.



But if h_{1234} is going to have the correct value and there is no collision, h_{12} and h_{34} both must be correct also, which means that h_3 and h_4 must also have the correct values …

and if h_3 is correct and there is no collision, f_3 must be correct.

This class is being recorded

File Storage Summary

This method is known as a Merkle tree.

Alice only needs to store one hash value.

When she retrieves a single file, she receives an additional $O(\log n)$ hash values.

> This compares to $O(n)$ stored hash values if she tries to store them all.

Note that this is a different problem from a MAC in two ways:

- Alice doesn't have to worry above the authenticity of the hash that she keeps, so she doesn't need to authenticate it.
- Alice is only concerned about verifying *part* of the full set of stored files rather than the whole set.

This class is being recorded

Multiparty Computation

A different type of cryptographic protocol is a secure multiparty computation, in which two or more people are trying to perform some computational task but don't trust each other.

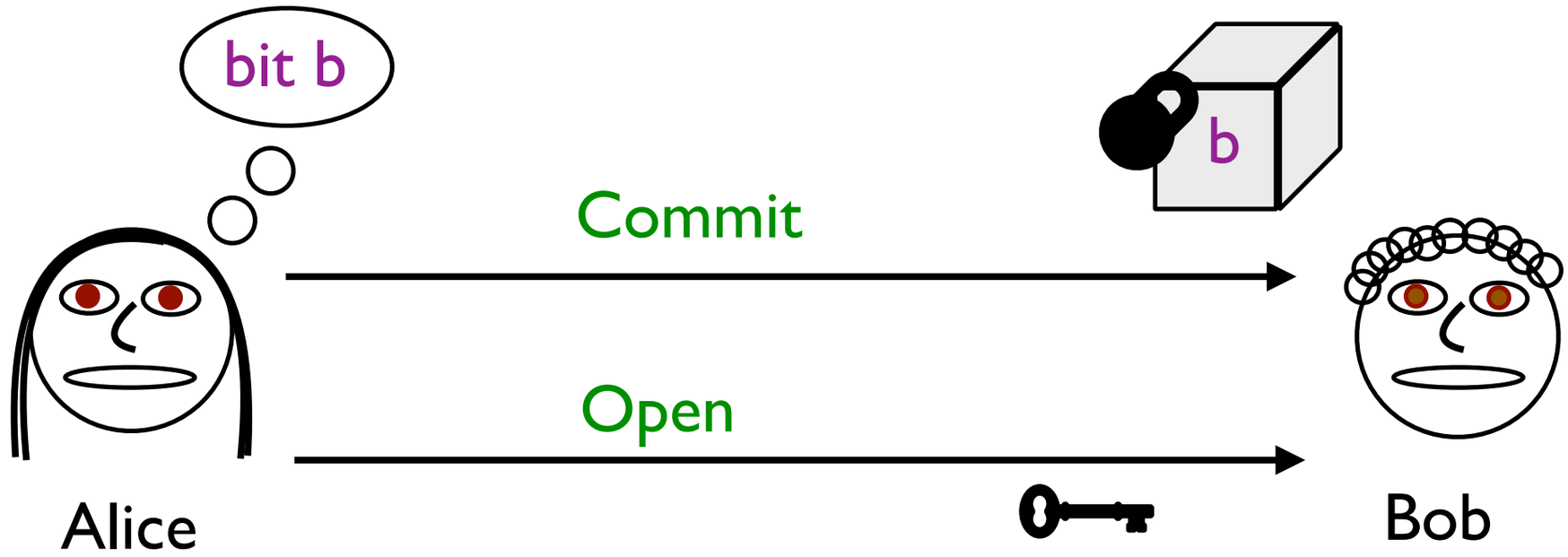
> This differs from the situation in the communication protocols we have seen so far in that the adversary controls one (or more) of the expected users of the protocol.

There are a wide variety of multiparty computation protocols. An example includes zero-knowledge proofs, a method by which it is possible to convince someone of a fact without revealing any information about the proof itself.

This class is being recorded

Bit Commitment

A useful cryptographic primitive for multiparty computations is bit commitment.



In the Commit phase, Alice sends Bob a bit b encoded in some way, in a virtual lockbox which Bob cannot open to learn b.

In the Open phase, Alice sends Bob the virtual key to open the lockbox, revealing b. Alice should not be able to change b from the value she originally put in the lockbox.

This class is being recorded

Why Bit Commitment?

The main application of bit commitment is as a cryptographic primitive useful for building more complicated multiparty computation protocols.

However, it can occasionally be useful by itself.

Example: Suppose I have a good algorithm to predict the stock market. I want to convince you that I can do this but I don't want to reveal what will happen since you haven't paid me.

Solution: I commit to a prediction about what the stock market will do today. You won't be able to see it until the opening phase, which happens after the day ends. But I can't change the value which I committed to before the day started, so once the commitment is opened, you can see that I predicted correctly.

This class is being recorded

Hash Functions for Commitment

Commit Phase: Alice chooses random r and then sends Bob $H(b||r)$ to commit to the bit b.

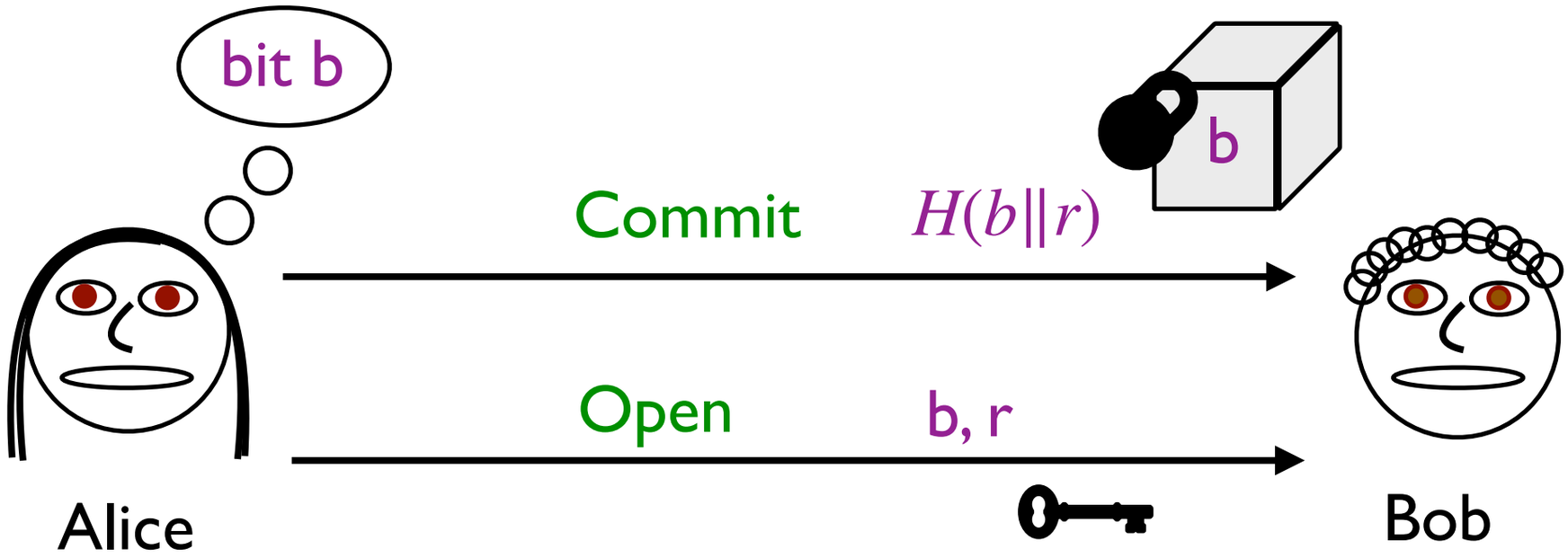
Open Phase: Alice sends to Bob b and r. Then Bob computes $H(b||r)$ to verify the commitment.

The security requires that Bob not be able to determine b before the open phase. (The protocol is hiding or concealing.) Note that it is possible that the commitment is *never* opened, and it should continue to be binding indefinitely.

Security also requires that Alice not be able to open the commitment to more than one possible value. (The protocol is binding.)

This class is being recorded

Security of Commitment with Hash



The hiding property follows if Bob is unable to invert the hash function to learn b. This is certainly true if the hash function is modeled as a random oracle. (But you only need it to be a one-way function.)

The binding property follows from the collision resistance of the hash function: To open the commitment with a different b, Alice would need to find (b', r') with $H(b, r) = H(b', r')$.

This class is being recorded