Lab #4 - September 27, 2017

Due Dec 31 at 11:59pm **Points** 13 **Questions** 13
Available Sep 27 at 9:20am - Dec 31 at 11:59pm 3 months **Time Limit** None
Allowed Attempts Unlimited

Instructions

[1] Good morning and welcome to the fourth lab session!

Lab sessions provide the opportunity for recitation and more in-depth understanding of the materials covered in class, as well as preparation for upcoming homework assignments.

Your attendance only of a given lab session (and, thus, your participation in the assignments and/or discussions) gives you full credit. You are expected to stay in

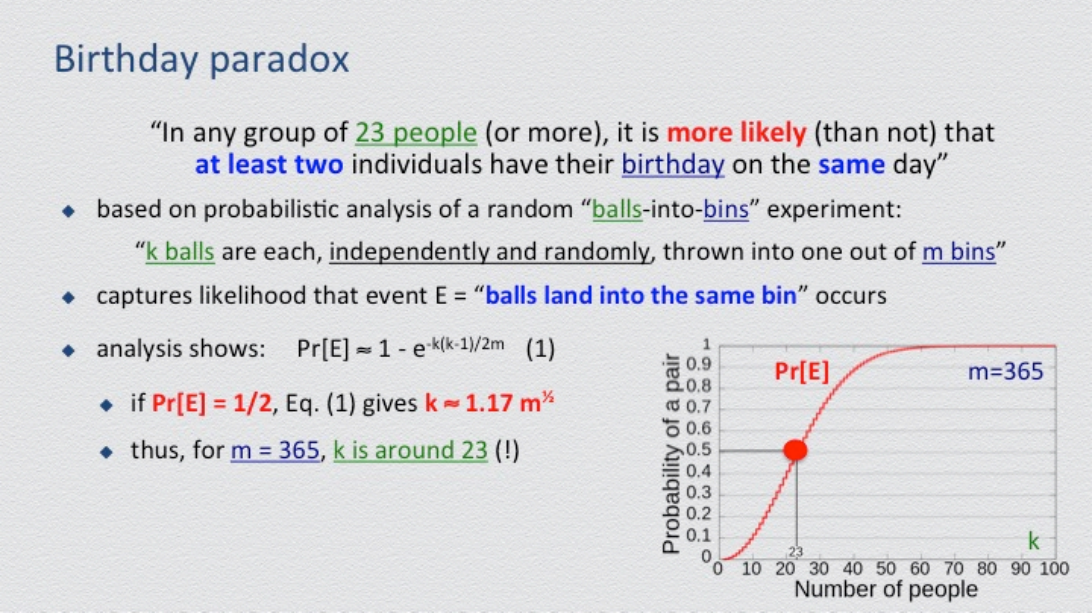
Take the Quiz Again

Attempt History

	Attempt	Time	Score
KEPT	Attempt 1	19 minutes	4 out of 13
LATEST	Attempt 2	2 minutes	2 out of 13
	Attempt 1	19 minutes	4 out of 13

Submitted Sep 27 at 2:01pm

Question 1 0 / 1 pts



Recall the birthday paradox explained above.

If B_1, B_2, ..., B_m denote the m bins, which event E does Equation (1) capture the likelihood of occurring?

A given bin B_i receives two or more balls.

orrect Answer

There exists an i (between 1 and m) so that two or more balls land in bin B_i.

'ou Answered

All balls land at half of the balls. That is, half of the balls receive no ball.

No. The event defined by this (incorrect) answer is not related to event E.

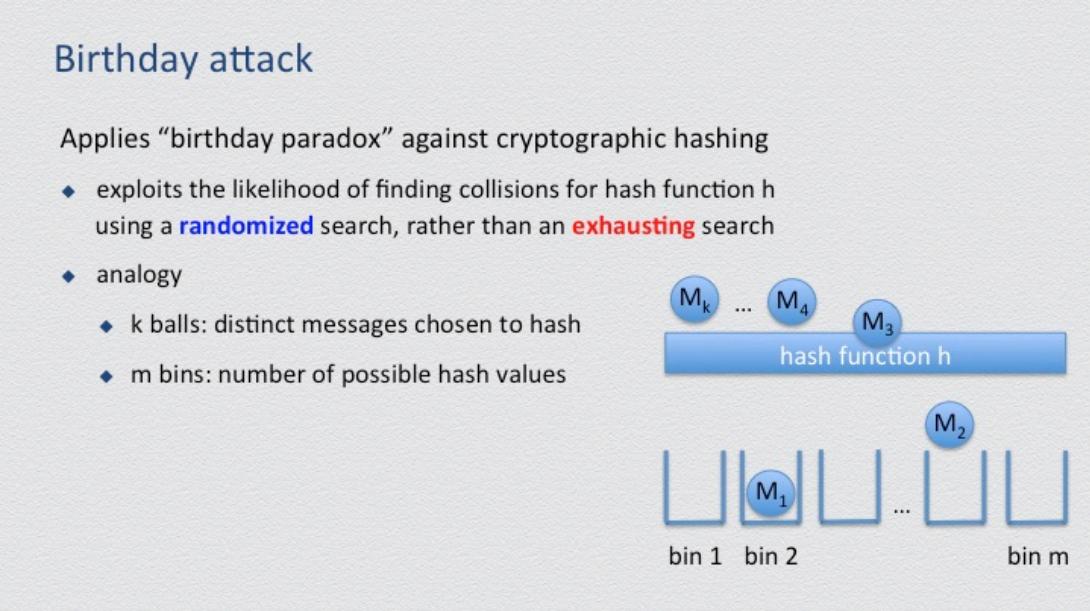
Two balls that are thrown one after the other land in the same bin.

Event E is the complement of event NE = {No two balls land in the same bin} = {Each ball lands in a distinct bin}. This means that there exists at least one pair of balls landing in the same bin.

The assumption, here, is that balls are thrown into bins independently of each other and according to the uniform distribution. That is, any ball will equal-probably land in a specific bin B_i (with probability 1/m). Of course, neither of the two assumptions really hold in the case of people's birth.

Question 2

0 / 1 pts



The birthday paradox can be applied to define successful birthday attacks against cryptographic hash functions: The hash function defines the mapping ("throws") of messages ("balls") into possible hash values ("bins").

The probabilistic analysis of the birthday paradox can be applied to the problem of finding collisions in cryptographic hashing, provided that:

The hash function h maps the message space M uniformly over the hash range H.

'ou Answered

The attacker chooses messages M_1, M_2, M_3, ..., M_k randomly from message space M.

This is a necessary but not sufficient condition. We also need that the hash function operates as an idealized random function.

If the hash function maps the message space M unevenly (i.e., with a non-uniform distribution) over range H., e.g., higher hash values are more often produced, then hash collisions are generally more likely to be found, meaning that equation (1) is only a lower bound to the actual collision-finding probability.

orrect Answer

The attacker chooses messages M_1, M_2, M_3, ..., M_k randomly from the message space M and, also, that the hash function h maps the message space M uniformly over the hash range H.

The hash function h maps the message space M uniformly over the hash range H and M is much much larger than H, that is, many messages are hashed to any possible hash value.

We need two conditions:

1) The attacker chooses messages M_1, M_2, M_3, ..., M_k randomly from the message space M; and,

2) The hash function h maps the message space M uniformly over the hash range H.

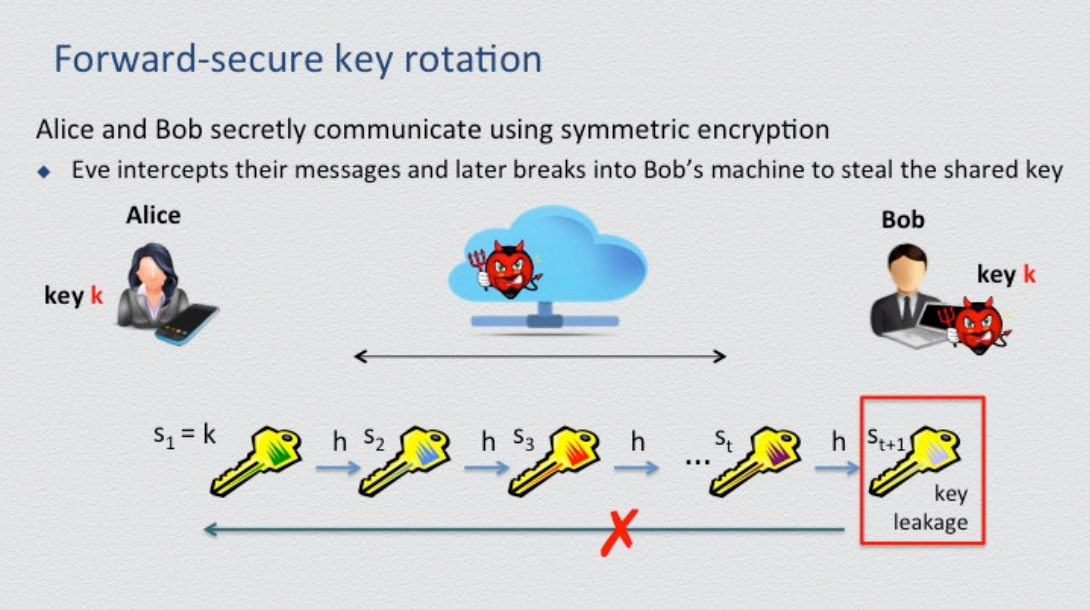
Indeed, both conditions are needed in order for the birthday-paradox analysis to accurately* measure the likelihood of finding hash collisions: We need that h(x) yields any of the |H|=m hash values with equal probability 1/m. Since function h is deterministic, this means that a randomly chosen message x is mapped to the i-th possible hash value h_i with probability 1/m. Thus, we need that messages are chosen at random and that hash values are distributed evenly.

Also, please note that because the hash function h is "stateless" (i.e., inputs are mapped to outputs independently of previously performed hash evaluations), the independence condition on ball throws is already satisfied.

* Of course, Equation (1) is an approximation needed to derive a closed-form formula of the studied probability.

Question 3

0 / 1 pts



Consider the above attack scenario, where an attacker, who controls the communication channel between Alice and Bob which is protected using symmetric cryptography, breaks in Bob's laptop and steals his key k.

As a countermeasure, and starting with their initial key $k = s_0$, Alice and Bob can have their secret key periodically evolve over time (e.g., every midnight) in an unpredictable and irreversible manner, in particular, using a cryptographic hash function to derive the new key s_{i+1} as $h(s_i)$, i = 0, 1, ..., that is, as the hash of the current key s_i.

This key "chaining" via cryptographic hashing is called to provide "forward security" (shown in the bottom of the figure). What does "forward security" mean in this context?

That the attacker cannot forward any of the intercepted messages as they are securely chained with each other and linked to the underlying communication channel.

You Answered

As the name suggests, if an attacker steals the key on January 1st, 2019, she can only break the communications occurring in 2018 or earlier, but not any future communications occurring in 2019.

No. The hash function is known so he can evolve the key himself...

orrect Answer

Counterintuitively, if an attacker will steal the key on January 1st, 2019, she can only break the communications occurring in 2019 and on, but not any communications occurred in 2018 or earlier.

It means that any key leakage does not affect past communications.

Question 4

0 / 1 pts

Hash values as file identifiers

Consider a cryptographic hash function H applied on a file F

- the hash (or digest) H(M) of F serves as a **unique** identifier for F
  - "uniqueness"
    - if another file F' has the same identifier, this contradicts the security of H
  - thus
    - the hash H(F) of F is like a fingerprint
    - one can check whether two files are equal by comparing their digests

Many real-life applications employ this simple idea!

Which property or properties are relevant to the "hashes-as-identifiers" approach, as described above?

Onewayness.

orrect Answer

Succinctness and collision resistance.

'ou Answered

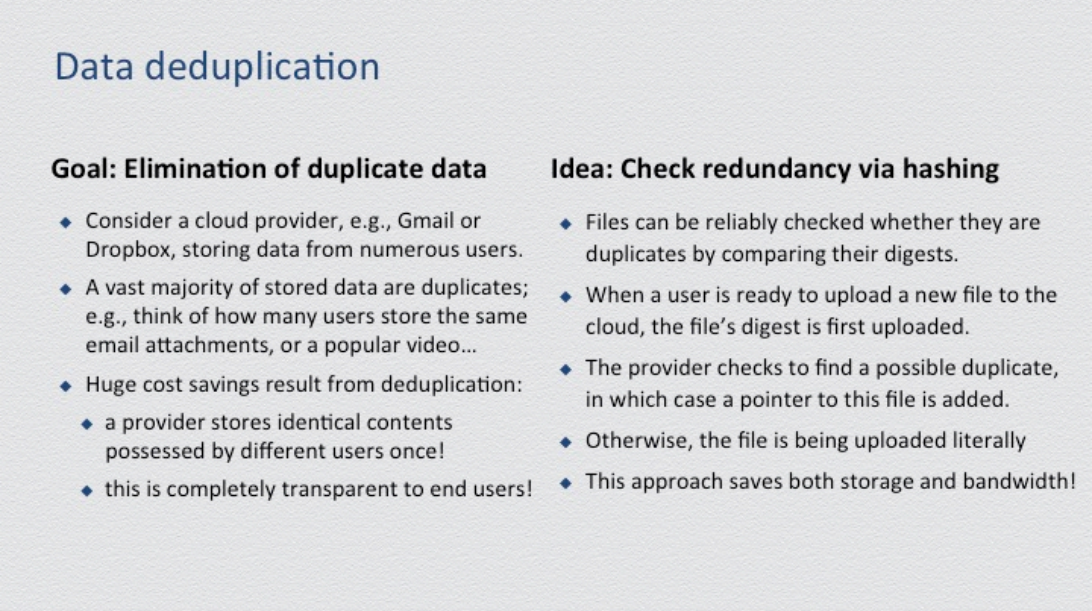
Collision resistance.

Not only; succinctness is also important to ensure the space efficiency of the data object identifiers.

Succinctness.

Question 5

0 / 1 pts



Suppose that a movie studio is about to release a blockbuster (very popular) movie M. How can it test if the movie has been pirated to Dropbox - a provider that uses the above data deduplication technique for reducing storage and bandwidth costs?

'ou Answered

By hashing their movie and uploading to Dropbox the resulting hash value.

No, no, no...

By hacking into Dropbox and searching to find their movie.

By hashing their movie and checking wheather the last bit of the resulted digest is 0.

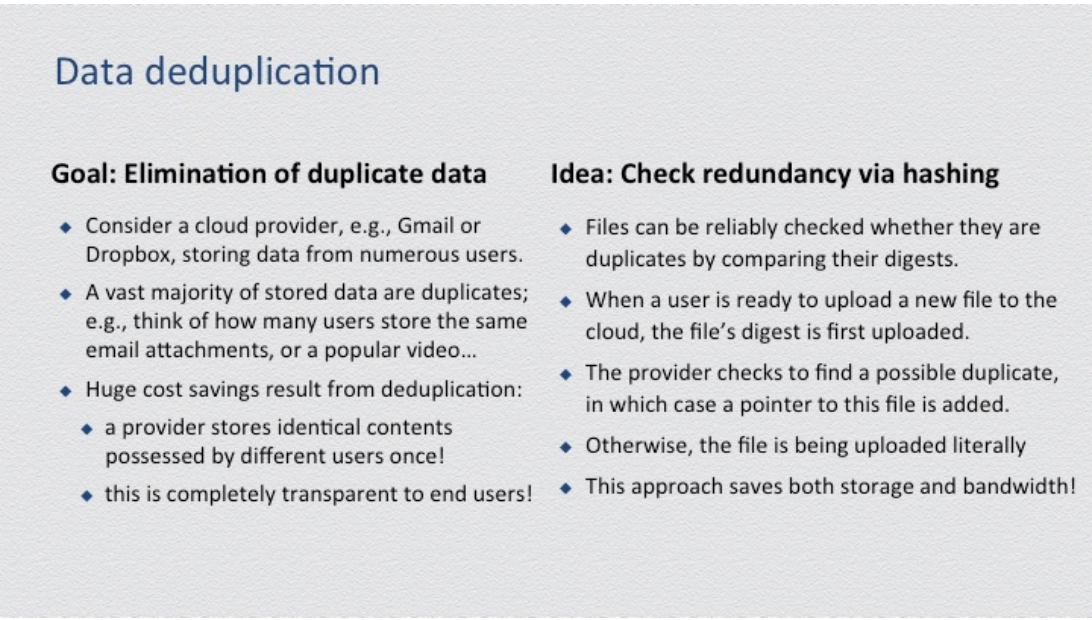
orrect Answer

By checking if they can themselves successfully upload the movie to Dropbox.

The Dropbox client application - in order to save bandwidth - initially only send the file digest to the back-end server, and if the file exists, no file upload is initiated by the client application. Thus, the movie studio can test whether the movie has been leaked to Dropbox simply by monitoring whether the client application ends up uploading the movie.

Question 6

1 / 1 pts



Are there any negative consequences with respect to data confidentiality that are introduced by the above approach for data deduplication?

Correct!

- (•) Yes, in principle.

  Indeed. Anyone can hash a given file to compute its digest and, as long as, digests' equality essentially implies duplicate object, in certain settings sensitive information can be leaked. For instance, if a Dropbox user stores in the cloud a personal file containing sensitive information, then an attacker may guess the format and content of this file and check easily if it is indeed stored in Dropbox: Since this file is highly personalized, and no one else would ever store such a file, it is possible that the attacker successfully concludes with high confidence that the victim user has a certain sensitive property (e.g., she is a patient with a rare disease).

- ( ) No - hashing is not related to confidentiality in any way.

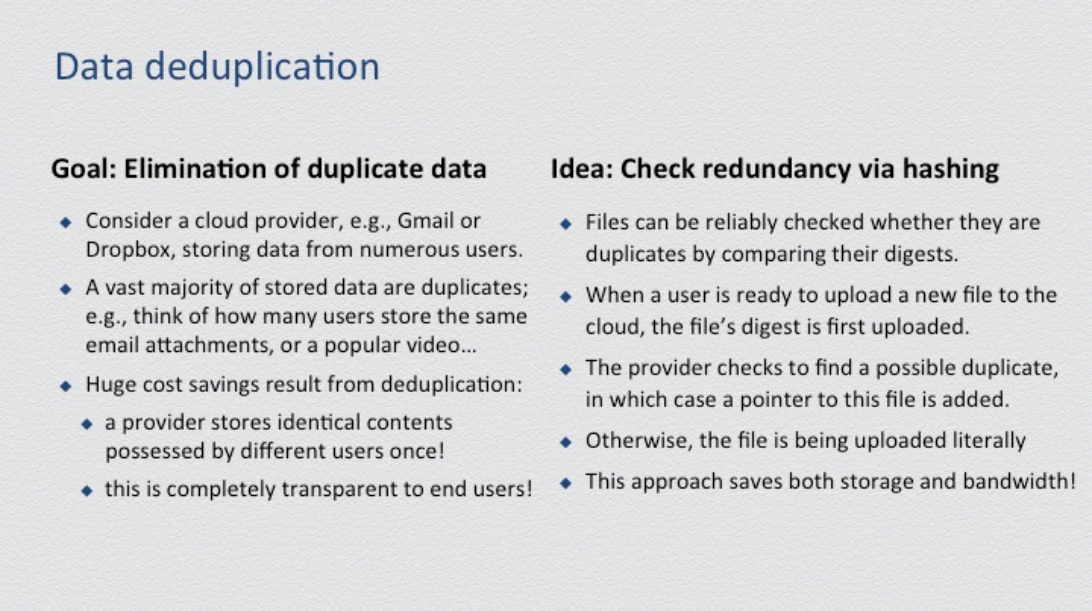
- ( ) No - the onewayness of the underlying cryptographic hash guarantees data confidentiality.

Deduplication affects, in principle, data confidentiality.

Anyone can hash a given file to compute its digest and, as long as, digests' equality essentially implies duplicate object, in certain settings sensitive information can be leaked. For instance, if a Dropbox user stores in the cloud a personal file containing sensitive information, then an attacker may guess the format and content of this file and check easily if it is indeed stored in Dropbox: Since this file is highly personalized, and no one else would ever store such a file, it is possible that the attacker successfully concludes with high confidence that the victim user has a certain sensitive property (e.g., she is a patient with a rare disease).

Question 7

0 / 1 pts



To protect against possible leakage of sensitive information, Dropbox is about to release a new version of its client application, where before being uploaded to the server, files are being encrypted using a CPA-secure symmetric-key encryption under a user-specific key. The whole file-encryption process will be completely transparent to end users.

You are the product manager responsible for the product release and have the final "go/no-go" saying. What is your decision?

'ou Answered

- (•) It depends on the strength of the encryption scheme that is being used, in particular, the key length: if the key length is at least 2n bits long, where n is the desired security strength, typically at least 80, then it's a go, otherwise a no-go!

  No. The analysis does not make any sense. A CPA-secure cipher is randomized, therefore it destroys any benefits coming from deduplication.

orrect Answer

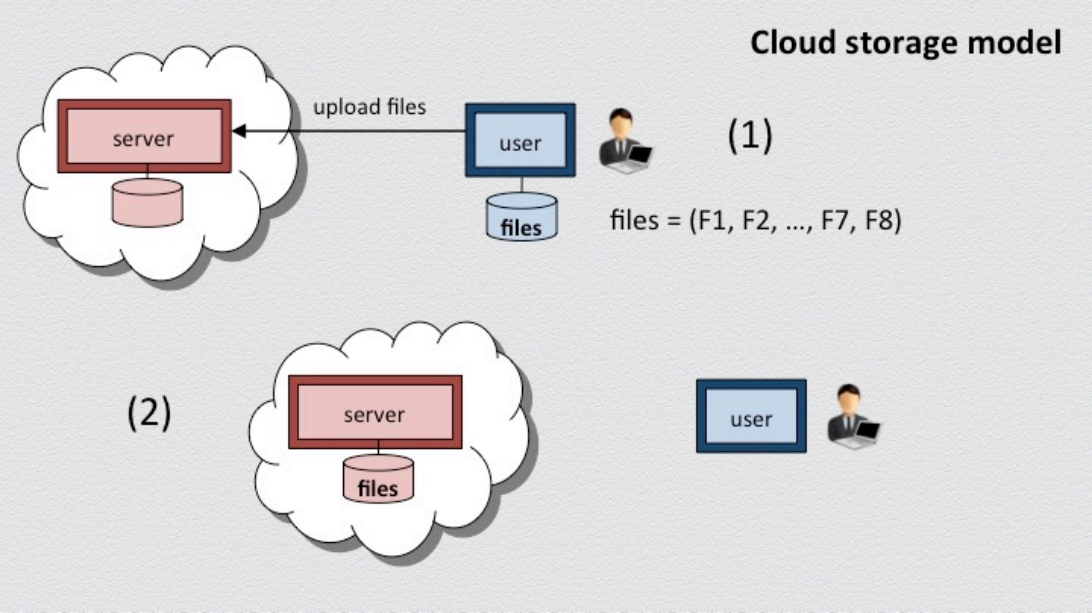
- ( ) No go.
- ( ) Go.

Don't do this...

CPA-encryption is necessarily randomized. Then, such file encryption and file deduplication conflict each other. A CPA-secure encryption scheme would remove any regularities in data objects thus making them "distinct" in the ciphertext domain for all practical purposes (namely, any ciphertext will almost always be unique)! Therefore, almost no data duplicates will be found and the company will lose millions of dollars in increased costs...

Question 8

1 / 1 pts



Let's now explore the application of cryptographic hashing to secure cloud storage.

Figure (1) depicts the uploading of a user's data to the cloud.

Figure (2) depicts the state of the system after the data uploading. We assume that the user does not keep a local copy of his data.

What justifies this assumption in practice?

Users need absolutely keep local copies of their data, or else no security can be achieved.

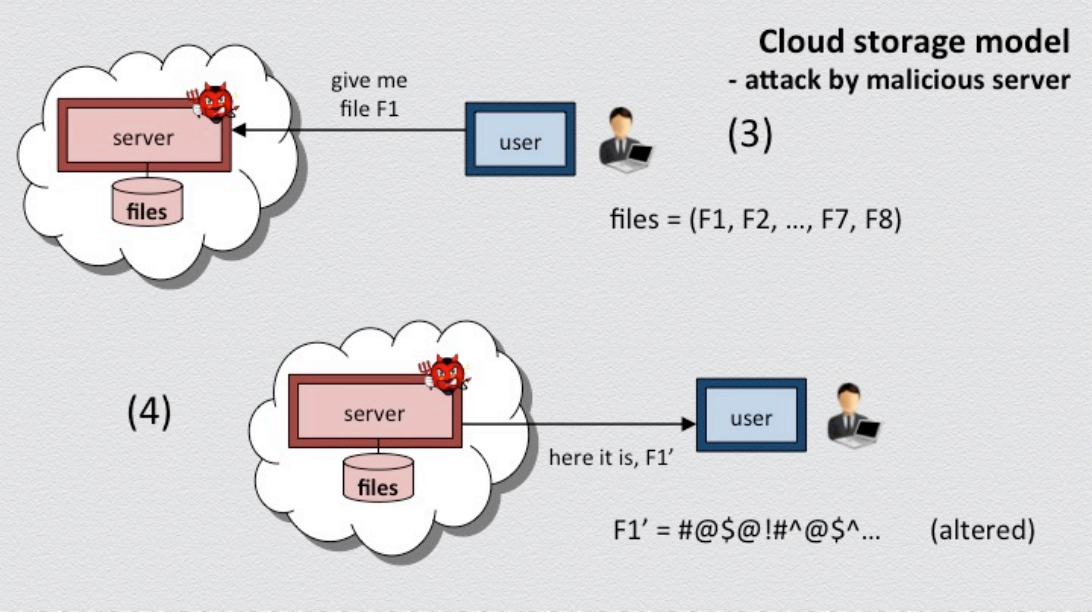
Correct!

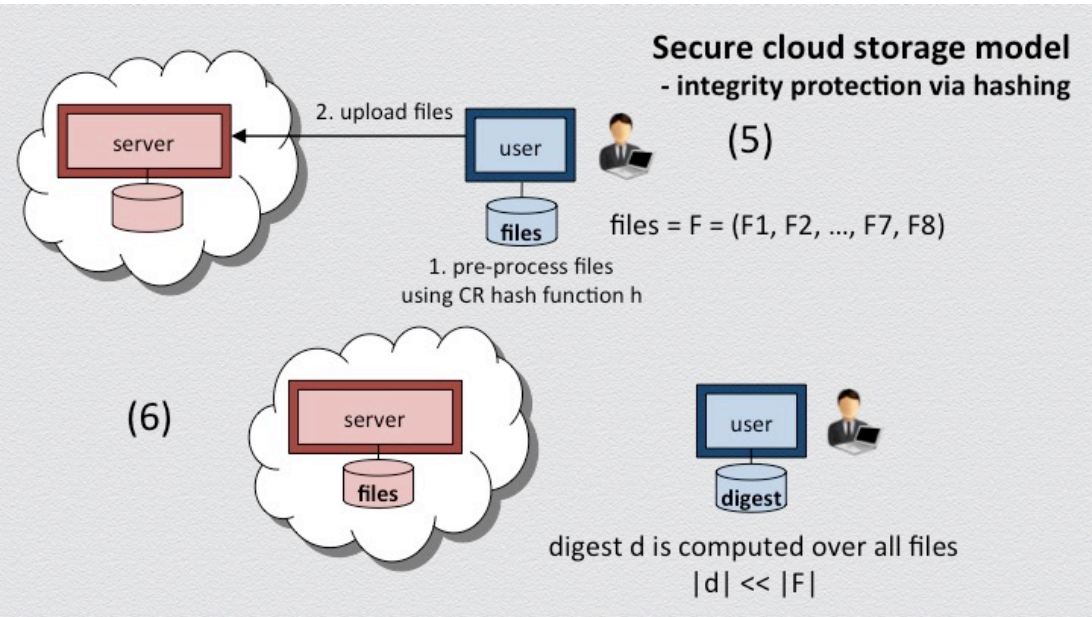
Two reasons are: 1) User outsource their data to be able to transparently and consistently access it by many personal devices, so keeping local copies in these devices introduces the risk of inconsistencies between older and newer versions of data. 2) Keeping a local copy defeats the purpose of data outsourcing.

Right. There may be more reasons for completely outsourcing their data (not keeping a local copy_. This does not mean that users have to necessary do so, but this will be one of our design goals.

Question 9

0 / 1 pts





But things can go wrong if the cloud provider is malicious, as seen in Figures (3) and (4).

Cryptographic hashing can be employed to protect data integrity - better, to enable detection of malicious data manipulation.

Specifically, a collision resistant hash function h is used by the user to first pre-compute a digest d of his data, which the user can locally store, prior to its upload to the cloud, as Figures (5) and (6) show - and as we saw also in class, but with one difference: in the above setting, the digest is a **single hash value** over the entire file collection and thus it can be stored locally (i.e., it is not outsourced to the cloud). Digest d is what will allow the user to verify the integrity of his outsourced files whenever he retrieves a file.

In this setting, what is the advantage of employing a locally-stored single-hash-value digest over the entire file collection (as above) over employing cloud-stored file-specific digests (as we saw in class)?

orrect Answer

- The solution based on a locally-stored single-hash-value digest does not require the client to keep a secret state.
- The solution based on a locally-stored single-hash-value digest is compatible with data outsourcing.

'ou Answered

The solution based on cloud-stored file-specific digests does not allow accessing the file system from different devices.

The two approaches are equally compatible with transparent and consistent data access across different devices, as long as the user has a means by which the user's locally-stored state (the digest d, above, or the randomness r, in the solution that we saw in class) can be migrated to another device.

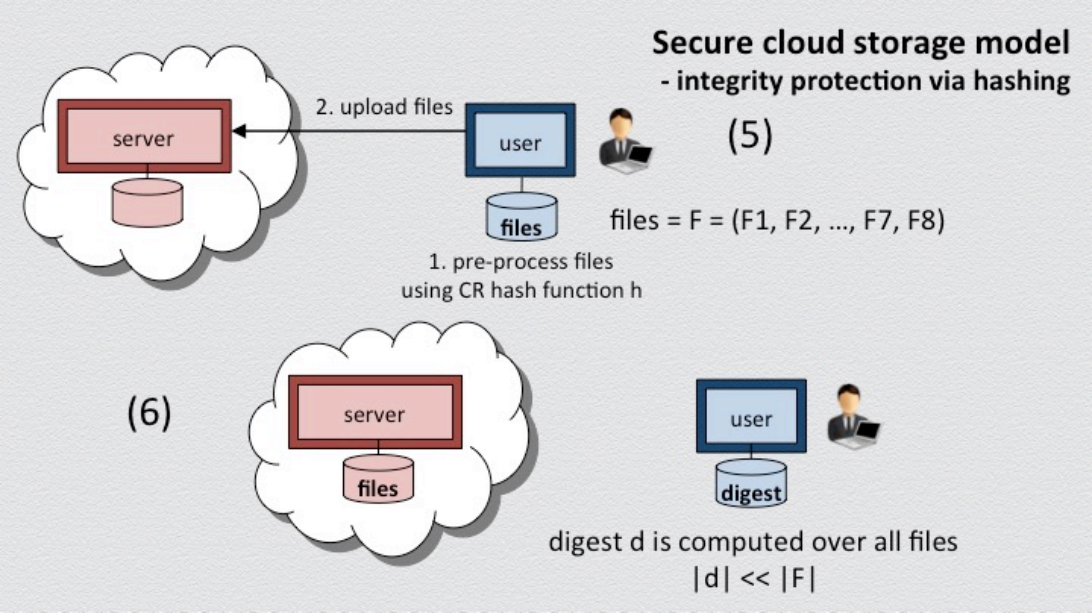
The two approaches are equally compatible with:

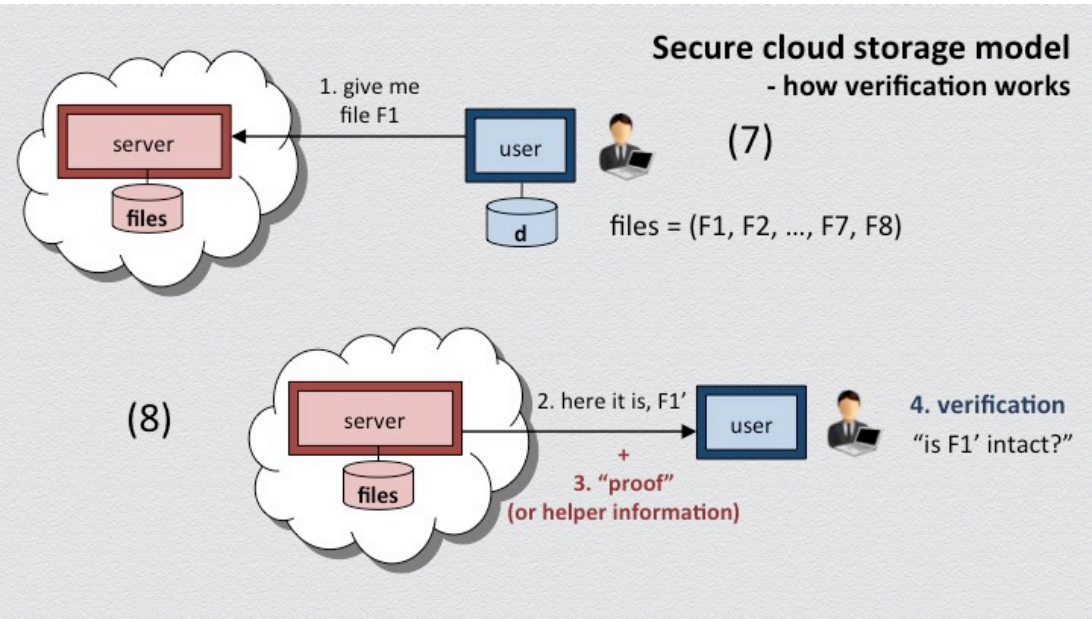
- data outsourcing, as they minimally require the user to keep local state, namely, the digest d in the above solution and the randomness r in the solution that we saw in class;
- transparent and consistent data access across different devices, provided that user's state can be migrated from one device to another device.

However, a file-system's digest d is much less sensitive information than the randomness r which must be kept secret, and this is the main advantage of the first solution over the second one. Indeed, the hash value of any reasonably-large file system can be safely become public, practically leaking no information. Note that because of this a digest is much easier to be migrated to other devices (one can simply send an email to herself), as opposed to a secret r which must migrated securely!

Question 10

0 / 1 pts





And, here is how things work with the verification (or integrity checking). Figure (7) depicts a request for file F1. Figure (8) depicts what the server is supposed to respond for this request: The server returns a file F1' along with a "proof" or helper information, **which should allow the use to recompute the digest**, in order to perform a canonical verification of the form: recompute the digest and compare with the stored original digest version.

Say that in Figure (5), the digest is produced as d = h(h(F1)||h(F2)||...||h(F8)) and the user asks for F2. What will be the answer and the proof in this case when the server is not malicious?

'ou Answered

In this case (F1, F2, ..., F8) is the answer to the request and the proof is nothing.

No, the answer is F2 and the hashes of all other files are needed to be provided as proof to the client in order for the client to perform canonical verification.

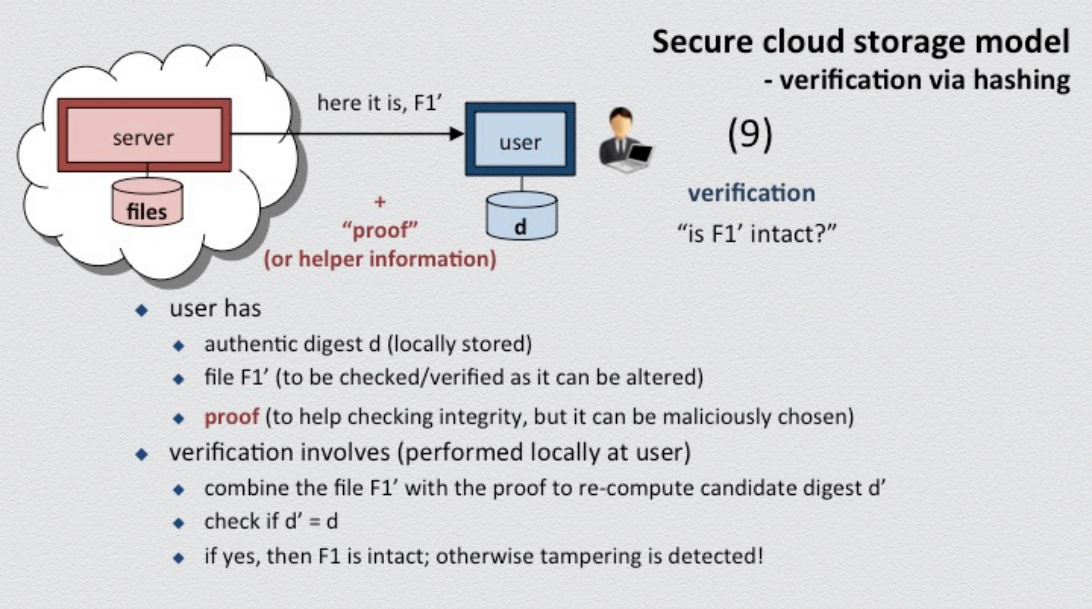
orrect Answer

File F2 will be the answer to the request and h(F1), h(F3)...,h(F8) will be the proof.

File F2 will be the answer to the request and h(F1), h(F3) will be the proof, because F1, F3 are the only files appended to the answer F2.

Question 11

0 / 1 pts



Finally, given the server's response to the file request and the proof, the user is able to perform a verification (checking) of the integrity of the provided file.

Figure (9) provides the details.

When there are three files, the digest is computed as d = h(F1 || h(F2 || h(F3))), and F3 is requested, what is the proof that an honest server must return and what is the verification computation performed by the user?

orrect Answer

- The proof is F2, F1 and the user recomputes d from scratch.
- In this case F3 cannot be tested as being intact.

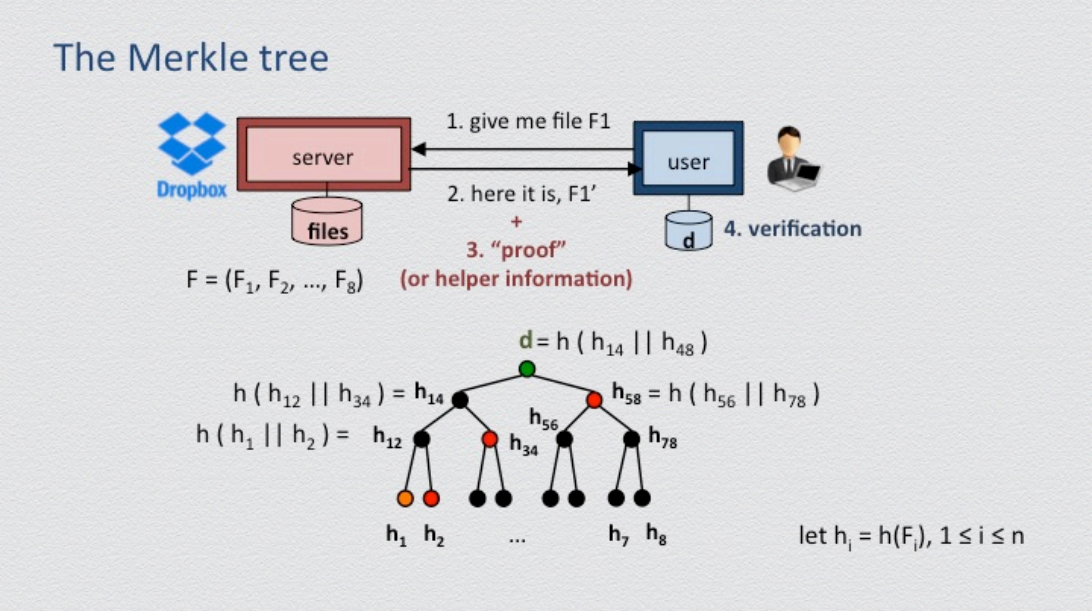
'ou Answered

- The proof is F1, h(F2 || F3) and the user recomputes d from scratch.

No, because this will not allow the user to recompute the digest by using the answer F3!

Question 12

0 / 1 pts



An important application of cryptographic hashing is the Merkle tree.

In the Dropbox setting, if file F_1 is requested by the user, its integrity can be checked against the tree's root hash, the digest d, by having the server providing as "proof" the red hash values (corresponding to the nodes that are

siblings of the nodes in the leaf-to-root path corresponding to the requested file). Indeed, the red proof connects the hash h_1 of the answer to the digest d (through iterated applications of the underlying collision-resistant hash function h). Interestingly, the Merkle tree can been seen as another implementation of the Merkle-Damgaard transform for achieving "domain extension" for collision-resistant hash functions (i.e., how one can hash arbitrarily large data objects into a succinct digest - given an underlying secure hash function that possibly operated on objects of bounded size).

Seen as such a "domain extension" hash function (e.g., above mapping 8 files to a single digest d), why intuitively the Merkle tree hash is collision resistant, when h is a collision-resistant hash function?

'ou Answered

- (•) Because finding as many h-collisions as the number of levels in the tree (e.g., four in our example above) is harder than finding one h-collision.

  No. Finding only one hash collision would be enough for a malicious server to forge a verifiable incorrect answer (file retrieval), but even this is computationally infeasible.

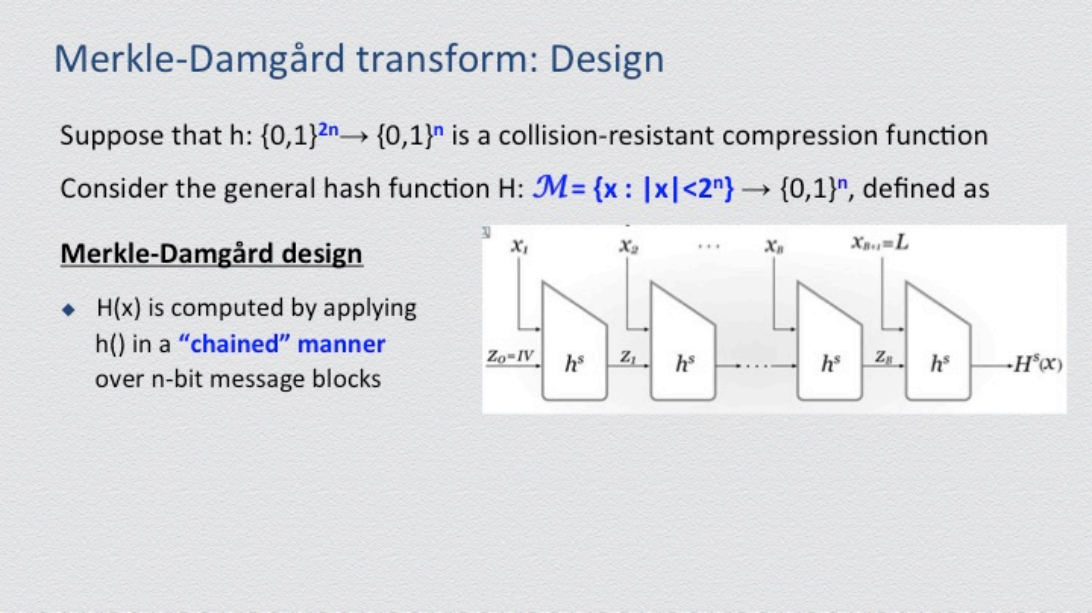
orrect Answer

- ( ) Because a collision at the Merkle tree level necessarily implies one collision at some level of the tree.

- ( ) Because the server does not know the exact structure of the Merkle tree.

Question 13

0 / 1 pts



Recall the Merkle-Damgard design framework for a hash function H. An input message x is padded appropriately so that a compression function h can be applied iteratively and in a chained manner over the message blocks so as to produce the final digest H(x):

- Starting from an initial internal state IV, the current internal state z_i is successively h-compressed with the next un-hashed message block to produce the new internal state z_{i+1};
- the digest H(x) is derived by applying a final h-compression to the last internal state z_B, this time with a special message block that encodes the message's length L.

What is the main property of this design framework?

orrect Answer

If h is a collision-resistant compression function, then H is a collision-resistant hash function.

The last h-compression of the last special message block ensures that collisions are harder to find.

'ou Answered

If h is a block cipher then H is a pseudorandom function.

No...