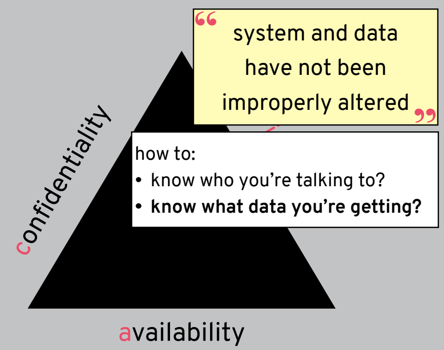
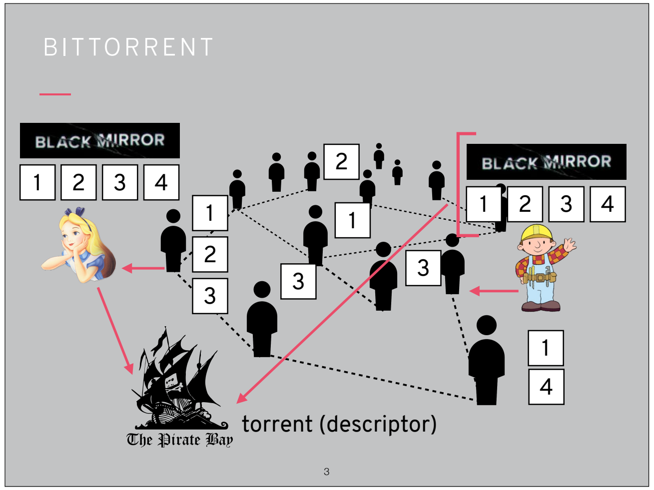
SECURITY (COMP0141): HASH FUNCTIONS

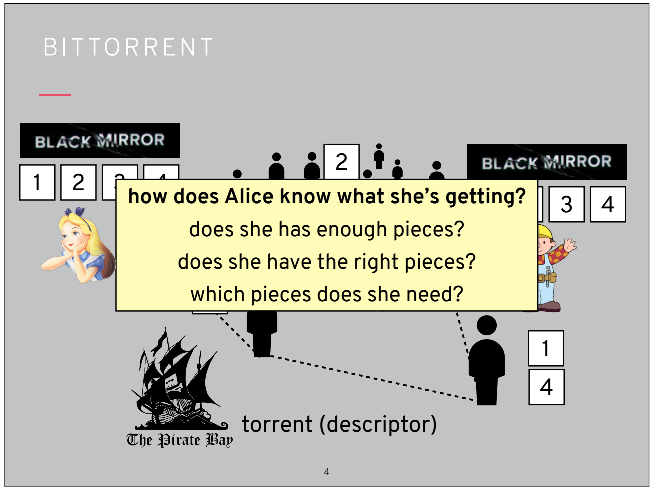
INTEGRITY



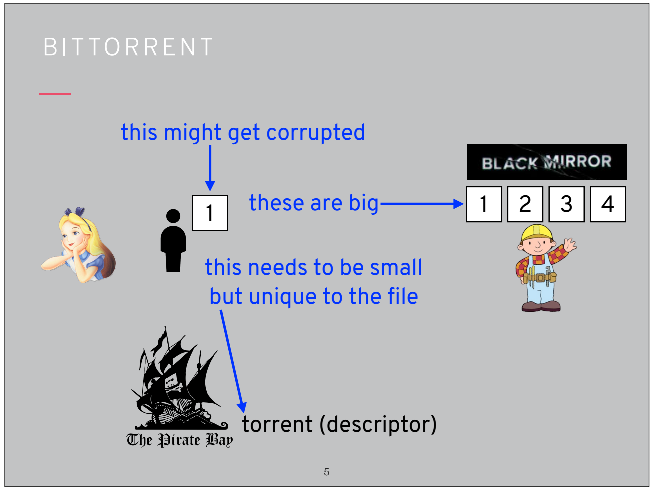
Integrity isn't just about talking to the right person, it's also about getting the right data



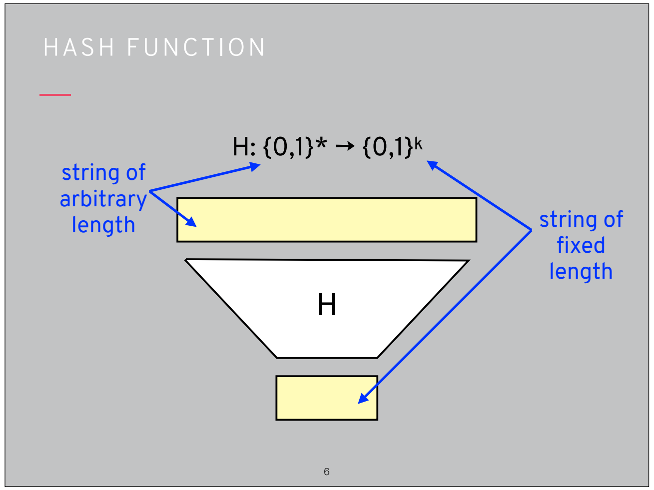
Here's a rough idea of how BitTorrent works (see https://en.wikipedia.org/wiki/BitTorrent for more detail). Bob has an episode of Black Mirror that he breaks down into pieces, and a descriptor of the file (called a torrent) gets sent to the server. The pieces then get shared around the peer-to-peer network, with different peers getting different pieces. When Alice goes to download the episode, she gets the descriptor and then the different pieces from different peers



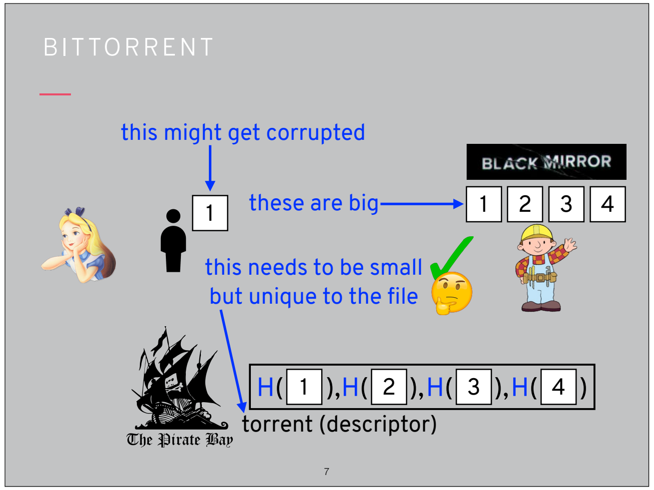
Lots of different questions about integrity here. How does Alice know when she has enough pieces? How does she know if the pieces she's getting are the right ones, or which pieces she still needs?



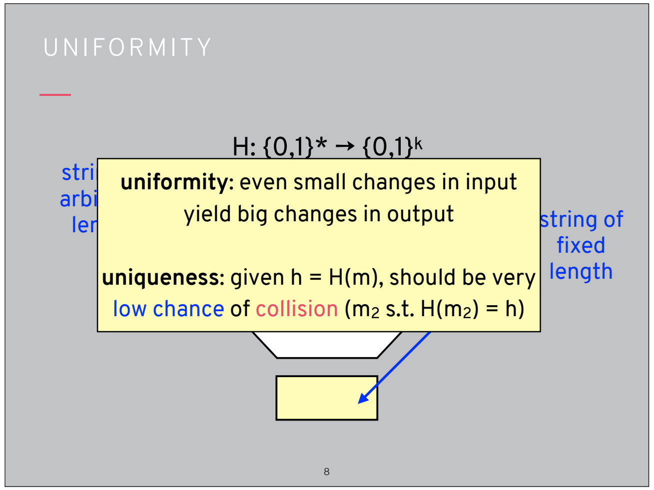
Need to satisfy various different requirements



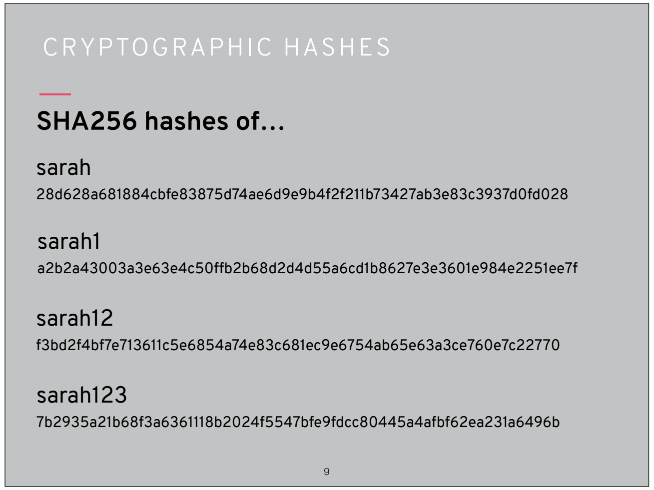
The way we do this is with hash functions, which compress large amounts of data into small representations



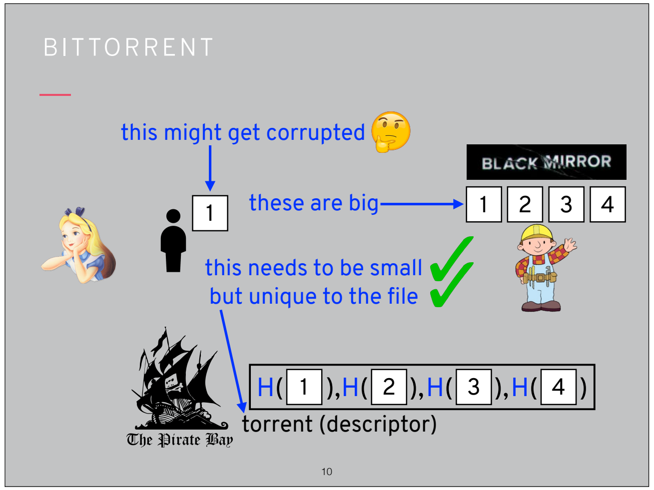
So to have small descriptor for even big pieces, we can hash the pieces. This solves the problem of needing big pieces but a small descriptor, but what about having the representation be unique?



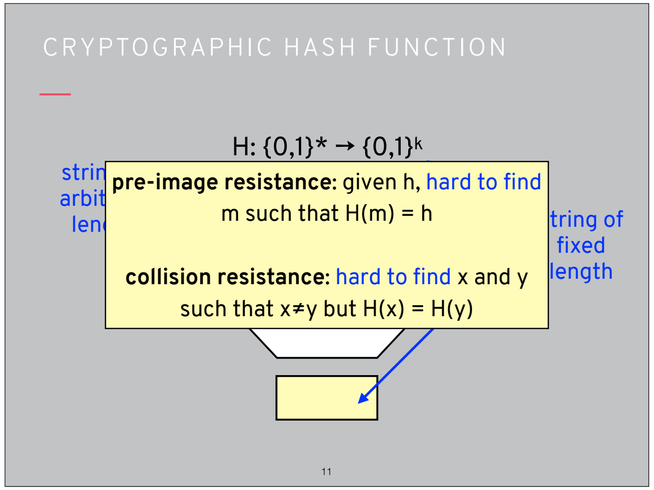
This is another part of the magic of hash functions: different files are very very unlikely to have the same hash value, and are essentially distributed uniformly over the output space. This is true even though there are infinitely many such collisions



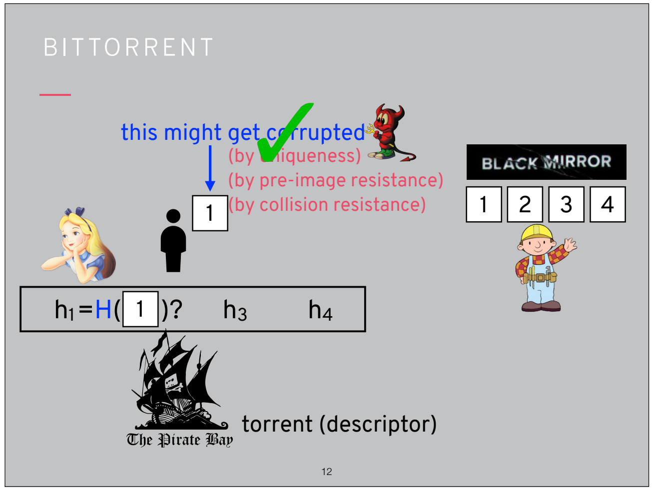
Here we can see the uniformity property in action: a small difference in the input to a hash function yields a huge difference in the output



Even if it’s unlikely in general for two things to hash to the same value, what if an attacker intentionally tries to make this happen, in order to get someone to download the wrong video?



Again, the magic of cryptographic hash functions means this won’t happen. Now we’ve shifted from saying that things like collisions are unlikely to happen to saying that even if you really want to find one you still can’t do it



This solves the problem of corruption, both accidental and adversarial, and thus allows Alice to be sure she’s getting the right file chunks by checking against the hashes

HASH FUNCTIONS

Two main security properties:

- **Pre-image resistance:** given H(x) it's hard to find x
- **Collision resistance:** it's hard to find x and y so that x ≠ y but H(x) = H(y)

COLLISION ATTACK

How quickly can we find a collision $x_1 \neq x_2$ such that $H(x_1) = H(x_2)$?

Let's look a little more in depth at how long it would take to find a collision

BIRTHDAY PARADOX

Consider a class of N students with random birthdays (meaning birthdays follow a **uniform distribution** over the days of the year)

How large does N need to be before there is more than a 50% chance of having two students with the same birthday?

The question of collisions is related to a well-studied problem called the birthday problem

---

BIRTHDAY PARADOX

P[A] = probability that two people have the same birthday
P[Ā] = probability that no two people have the same birthday
P[A] = 1 - P[Ā]

Event 1 (E1) = student 1 has a birthday (P[E1] = 1)
Event 2 (E2) = student 2 has a birthday different from student 1 (P[E2] = (365 - 1) / 365) = 364/365)
...
Event N (EN) = student N has a birthday different from all previous students (P[EN] = (365 - N + 1) / 365)

P[Ā] = P[E1]...P[EN] = $(1 / 365)^N$ * 365 * 364 * ... * (365 - N + 1)

Can derive a formula for this probability, then solve for N when P[A] > 0.5

BIRTHDAY PARADOX

Consider a class of N students with random birthdays (meaning birthdays follow a **uniform distribution** over the days of the year)

How large does N need to be before there is more than 50% chance of having two students with the same birthday?

Answer: $\sqrt{365} \approx 23$

It's surprising that it takes only 23 people, which is why it's called a paradox

COLLISION ATTACK

How quickly can we find a collision $x_1 \neq x_2$ such that $H(x_1) = H(x_2)$?

Pick different $x_1, \ldots, x_{\sqrt{N}}$ (where $N = 2^n$ for $H: \{0,1\}^* \rightarrow \{0,1\}^n$)

Compute $y_1 = H(x_1), \ldots, y_{\sqrt{N}} = H(x_{\sqrt{N}})$ and look for a collision

This has almost a 40% chance of finding a collision!

Memory cost: $3n*2^{n/2}$ bits
Computational cost: $2^{n/2}$ hash evaluations

This can guide us in an attack to find collisions that is more efficient than might be expected, basically you need to pick the parameters to be big enough so that even this attack is computationally infeasible

COLLISION ATTACKS IN PRACTICE

	n	birthday	shortcut
MD4	128	64	2
MD5	160	80	21
RIPEMD	128	64	18
RIPEMD160	160	80	
SHA-0	160	80	34
SHA-1	160	80	(51)
SHA-256	256	128	
SHA-3	256	128	

So the birthday attack says we need to do only 2^(n/2) computations, but actually some hash functions are broken well beyond that. Anything with a shortcut should be considered insecure

HASH FUNCTIONS

Two main security properties:

- **Pre-image resistance:** given H(x) it's hard to find x
- **Collision resistance:** it's hard to find x and y so that x ≠ y but H(x) = H(y)

Applications:

- File checksum
- MACs
- Digital signatures
- Commitments
- Blockchains
- Virus scanning (next week)
- Password storage (Week 7)
- ...and many more!

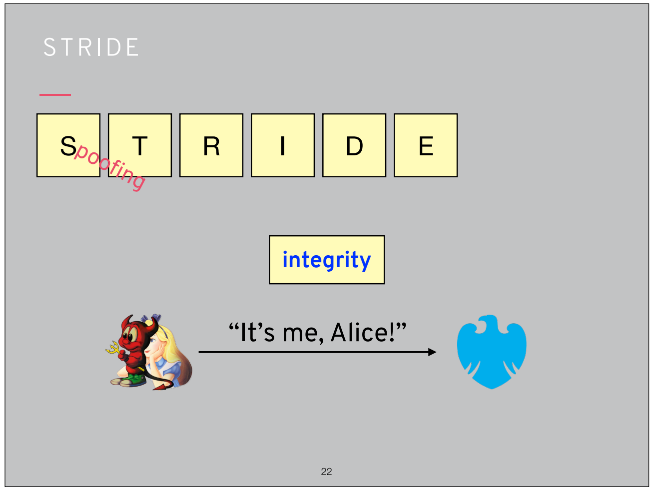
The next video will explore how incredibly useful and versatile hash functions are

CRYPTOGRAPHIC PRIMITIVES

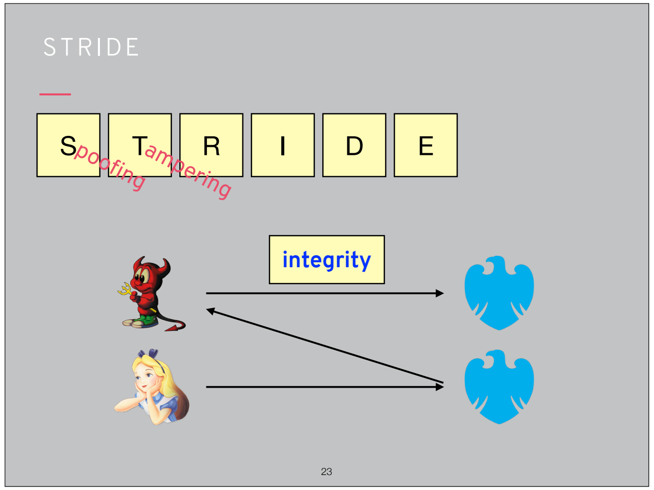
	setup?	confidentiality/ integrity?	fast?
SE	yes	confidentiality	yes
PKE	no*	confidentiality	no
digital signature	no*	integrity	no
MAC	yes	integrity	yes
OWF	no	confidentiality*	no
hash function	no	integrity	yes
AE	yes	both	yes

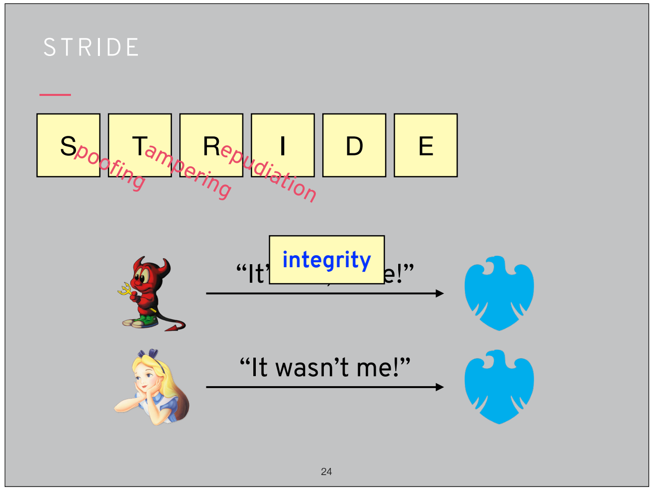
21

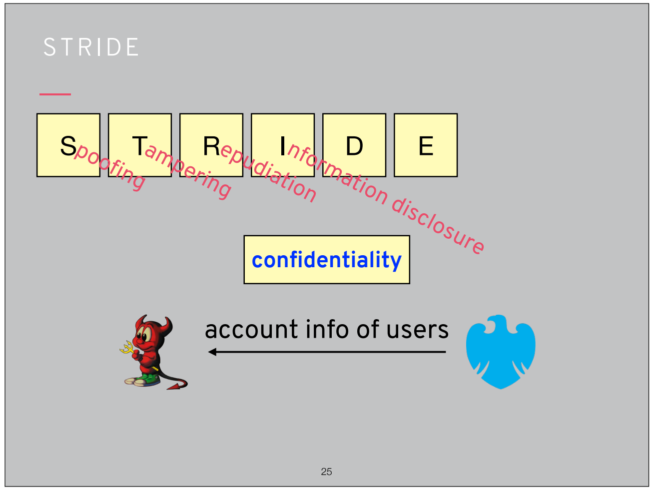
We have seen a lot of cryptographic primitives! Here is a summary of their main properties, in terms of requiring setup, which security notion they support, and how efficient they typically are

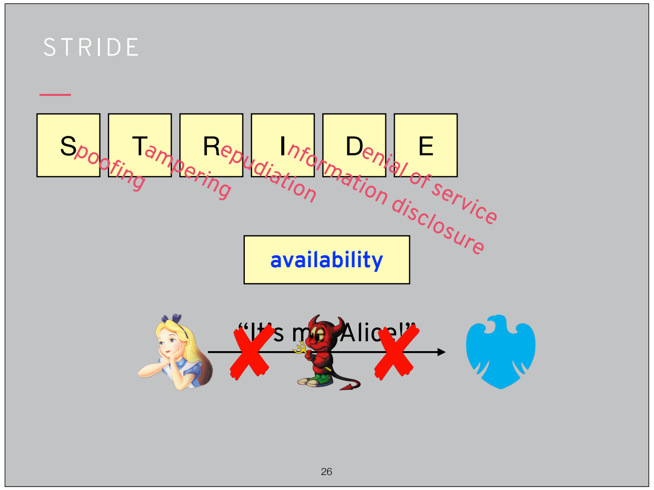


If we go back through STRIDE, we see that we've actually managed to address a lot of the threats using our new techniques for confidentiality and integrity

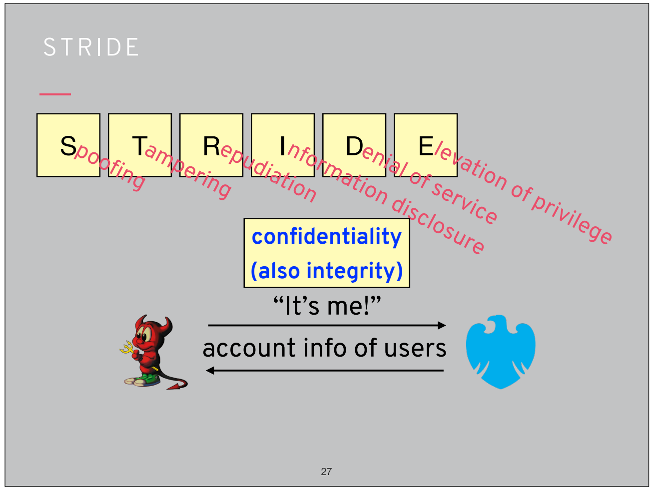

STRIDE
S
T
R
I
D
E
Spoofing
Tampering
integrity
23

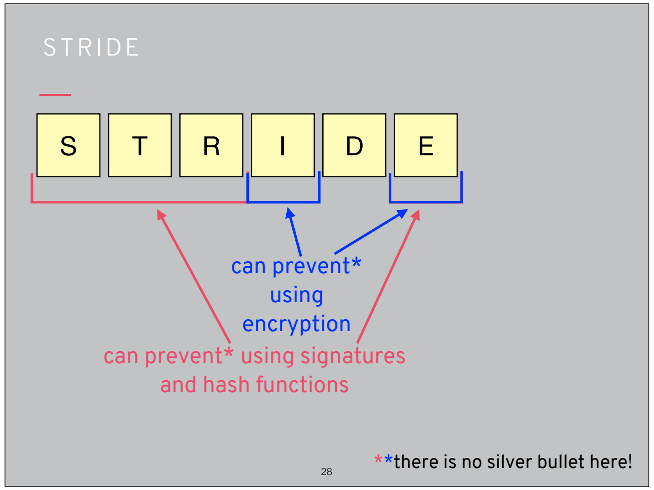

STRIDE
S
T
R
I
D
E
Spoofing
Tampering
Repudiation
integrity
“It wasn’t me!”
24





The exception is availability, which we'll see next week


STRIDE
S T R I D E
Spoofing
Tampering
Repudiation
Information disclosure
Denial of service
Elevation of privilege
confidentiality
(also integrity)
“It’s me!”
account info of users
27


STRIDE
S T R I D E
can prevent*
using
encryption
can prevent* using signatures
and hash functions
**there is no silver bullet here!
28

UNANSWERED QUESTIONS

How do I build a block cipher?

How do I build a stream cipher?

How do I build a hash function?

How do I implement any of these?

On the basis of this module: do not!

Use only standardised modes of operation and protocols, and code with only well-established and audited libraries

This module did not and will not teach you the details for a lot of the cryptography, just treat it as a black box. Again, don’t design your own crypto!