Sketch of Lecture 34

Fri, 4/24/2020

Birthday paradox and birthday attacks

Example 209. (birthday paradox) Among $n=35$ people (a typical class size), how likely is it that two have the same birthday?

Solution.

$$1-\left(1-\frac{1}{365}\right)\left(1-\frac{2}{365}\right)\left(1-\frac{3}{365}\right)\cdots\left(1-\frac{34}{365}\right)\approx 0.814$$

If the formula doesn't speak to you, see Section 8.4 in our book for more details or checkout:
https://en.wikipedia.org/wiki/Birthday_problem

Comment. For $n=50$, we get a 97.0% chance. For $n=70$, it is 99.9%.

Comment. In reality, birthdays are not distributed quite uniformly, which further increases these probabilities.

How is this relevant to crypto, and hashes in particular?

Think about people as messages and birthdays as the hash of a person. The birthday paradox is saying that "collisions" occur more frequently then one might expect.

Example 210. Suppose M is large ($M=365$ in the birthday problem) compared to n. The probability that, among n selections from M choices, there is no collision is

$$\left(1-\frac{1}{M}\right)\left(1-\frac{2}{M}\right)\cdots\left(1-\frac{n-1}{M}\right)\approx e^{-\frac{n^2}{2M}}.$$

Why? For small x, we have $e^x\approx 1+x$ (the tangent line of e^x at $x=0$). Hence, $\left(1-\frac{k}{M}\right)\approx e^{-k/M}$ and

$$\left(1-\frac{1}{M}\right)\left(1-\frac{2}{M}\right)\cdots\left(1-\frac{n-1}{M}\right)\approx e^{-1/M}e^{-2/M}\cdots e^{-(n-1)/M}=e^{-(1+2+\ldots+(n-1))/M}=e^{-\frac{n(n-1)}{2M}}$$

In the last step, we used that $1+2+\ldots+(n-1)=\frac{n(n-1)}{2}$. Finally, $e^{-\frac{n(n-1)}{2M}}\approx e^{-\frac{n^2}{2M}}$.

Decent approximation? For instance, for $M=365$ and $n=35$, we get 0.813 for the chance of a collision (instead of the true 0.814).

Important observation. If $n\approx\sqrt{M}$, then the probability of no collision is about $e^{-1/2}\approx 0.607$. In other words, a collision is quite likely!

In the context of hash functions, this means the following: if the output size is b bits, then there are $M=2^b$ many possible hashes. If we make a list of $\sqrt{M}=2^{b/2}$ many hashes, we are likely to observe a collision.

For collision-resistance, the output size of a hash function needs to have twice the number of bits that would be necessary to prevent a brute-force attack.

Practical relevance. This is very important for every application of hashes which relies on collision-resistance, such as digital signatures.

For instance, think about Eve trying to trick Alice into signing a fraudulent contract m. Instead of m, she prepares a different contract m' that Alice would be happy to sign. Eve now creates many slight variations of m and m' (for instance, by varying irrelevant things like commas or spaces) with the hope of finding m and m' such that $H(m)=H(m')$. (Why?!!)

Example 211. (chip based credit cards) Modern chip based credit cards use digitial signatures to authenticate a payment.

How? The card carries a public key, which is signed by the bank, so that a merchant can verify the public key. The card then signs a challenge from the merchant for authentication. The private key used for that is not even known to the bank.

Note that all of this can be done offline, without needing to contact the bank during the transaction.

https://en.wikipedia.org/wiki/EMV

There's an interesting and curious story made possible by the fact that, around 2000, banks in France used 320 bit RSA (chosen in the 80s and then not fixed despite expert advice) for signing the card's public key:

https://en.wikipedia.org/wiki/Serge_Humpich

Comment. For contrast, the magnetic stripe just contains the card information such as card number.

Comment. This also leads to interesting questions like: can we embed a private key in a chip (or code) in such a way that an adversary, with full access to the circuit (or code), still cannot extract the key?

https://en.wikipedia.org/wiki/Obfuscation#White_box_cryptography

A digital signature is like a hash, which can only be created by a single entity (using a private key) but which can be verified by anyone (using a public key).

As one might expect, a symmetric version of this idea is also common:

Example 212. (MAC) A **message authentication code**, also known as a **keyed hash**, uses a private key k to compute a hash for a message.

Like a hash, a MAC provides integrity. Further, like a digital signature, it provides authenticity because only parties knowing the private key are able to compute the correct hash.

Comment. On the other hand, a MAC does not offer non-repudiation because several parties know the private key (whether non-repudiation is desirable or undesirable depends on the application). Hence, it cannot be proven to a third party who among those computed the MAC (and, in any case, such a discussion would make it necessary to reveal the private key, which is usually unacceptable).

From hash to MAC. If you have a cryptographic hash function H, you can simply produce a MAC $M_k(x)$ (usually referred to as a HMAC) as follows:

$$M_k(x) = H(k, x)$$

This seems to work fine for instance for SHA-3. On the other hand, this does not appear quite as secure for certain other common hashes. Instead, it is common to use $M_k(x) = H(k, H(k, x))$. For more details, see:

https://en.wikipedia.org/wiki/Hash-based_message_authentication_code

From ciphers to MAC. Similarly, we can also use ciphers to create a MAC. See, for instance:

https://en.wikipedia.org/wiki/CBC-MAC