Example 195. (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 196. (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 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

Six of the seven Millenium Prize Problems (including the Riemann Hypothesis), for which the Clay Mathematics Institute has offered 10^6 dollars for the first correct solution, remain open.

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

Comment. Grigori Perelman solved the Poincaré conjecture in 2003 (but refused the prize money in 2010).

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

Example 197. (P vs NP) P versus NP is another one of the Millennium Prize Problems.

"*If the solution to a problem is easy to check for correctness, is the problem easy to solve?*"

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

Roughly speaking, consider decision problems which have an answer of yes or no. P is the class of such problems, which can be solved quickly. NP are those problems, for which we can quickly verify that the answer is yes if presented with suitable evidence.

For instance.

- It is unknown whether factoring (in the sense of does N have a factor $\leqslant M$?) belongs to P or not.
- Deciding primality is in P (maybe not so shocking since there are very fast nondeterministic algorithms for checking primality; not so for factoring)
- The travelling salesman problem is known to be NP-hard, meaning that it is in NP and as "hard" as possible (in the sense that if it actually is in P, then $\mathrm{P}{=}\mathrm{NP}$).

Comment. "Quickly" means that the problem can be solved in time polynomial in the input size.

Take for instance computing $2^n \pmod n$, where n is the input (it has size $\log_2(n)$). This can be done in polynomial time if we use binary exponentiation (whereas the naive approach takes time exponential in $\log_2(n)$).

Comment. This is one of the few prominent mathematical problems which doesn't have a clear consensus. For instance, in a 2012 poll of 151 researchers, about 85% believed $\mathrm{P}{\neq}\mathrm{NP}$ while about 10% believed $\mathrm{P}{=}\mathrm{NP}$.