Application: digital signatures

Goal: Using a private key, known only to her, Alice can attach her **digital signature** s to any message m. Anyone knowing her public key can then verify that it could only have been Alice, who signed the message.

- Consequently, in contrast to usual signatures, digital signatures must depend on the message to be signed so that they cannot be simply reproduced by an adversary.
- This should sound a lot like public-key cryptography!

Cryptographically speaking, a digitally signed message (m, s) from Alice to Bob achieves:

- **integrity**: the message has not been accidentally or intentionally modified
- **authenticity**: Bob can be confident the message came from Alice

  In fact, we gain even more: not only is Bob assured that the message is from Alice, but the evidence can be verified by anyone. We have "proof" that Alice signed the message. This is referred to as **non-repudiation**. We refer to a technical not a legal term here: if you are curious about legal aspects of digital signatures, see, e.g.:

  https://security.stackexchange.com/questions/1786/how-to-achieve-non-repudiation/6108

For comparison, sending a message with its hash $(m, H(m))$ only achieves integrity.

Example 188. (authentication using digital signatures) Last time, we saw that using human generated and memorized passwords is problematic even if done "right" (Example 184). Among other things, digital signatures provide an alternative approach to authentication.

Authentication. If Alice wants to authenticate herself with a server, the server sends her a (random) message. Using her private key, Alice signs this message and sends it back to the server. The server then verifies her (digital) signature using Alice's public key.

Obvious advantage. The server (like everyone else) doesn't know Alice's secret, so it cannot be stolen from the server (of course, Alice still needs to protect her secret from it being stolen).

(RSA signatures) Let H be a collision-resistant hash function.

- Alice creates a public RSA key (N, e). Her (secret) private key is d.
- Her signature of m is $s = H(m)^d \pmod N$.
- To verify the signed message (m, s), Bob checks that $H(m) = s^e \pmod N$.

This is secure if RSA is secure against known plaintext attacks.

Example 189. We use the silly hash function $H(x) = x \pmod{10}$.
Alice's public RSA key is $(N, e) = (33, 3)$, her private key is $d = 7$.

(a) How does Alice sign the message $m = 12345$?

(b) How does Bob verify her message?

(c) Was the message $(m, s) = (314, 2)$ signed by Alice?

Armin Straub
straub@southalabama.edu

Solution.

(a) $H(m)=5$. The signature therefore is $s=5^7 \pmod{33}$ (note how computing that signature requires Alice's private key). Computing that, we find $s=14$.

(b) Bob receives the signed message $(m,s)=(12345,14)$.
He computes $H(m)=5$ and then checks whether $H(m)\equiv s^3 \pmod{33}$ (for which he only needs Alice's public key). Indeed, $14^3\equiv 5 \pmod{33}$, so the signature checked out.

(c) We compute $H(m)=4$ and then need to check whether $H(m)\equiv s^3 \pmod{33}$. Since $2^3\equiv 8\not\equiv 4 \pmod{33}$, the signature does not check out. Alice didn't sign the message.

Just to make sure. What's a collision of our hash function? Why is it totally not one-way?

Example 190. Why should Alice sign the hash $H(m)$ and not the message m?

Solution. A practical reason is that signing $H(m)$ is simpler/faster. The message m could be long, in which case we would have to do something like chop it into blocks and sign each block (but then Eve could rearrange these, so we would have to do something more clever, like for block ciphers). In any case, we shouldn't just sign $m \pmod N$ because then Eve can just replace m with any m' as long as $m\equiv m' \pmod N$.

There is another issue though. Namely, Eve can do the following **no message attack**: she starts with any signature s, then computes $m=s^e \pmod N$. Everyone will then believe that (m,s) is a message signed by Alice. This does not work if H is a one-way function: Eve now needs to find m such that $H(m)=s^e \pmod N$, but she fails to find such m if H is one-way.

Example 191. Is it enough if the hash for signing is one-way but not collision-resistant?

Solution. No, that is not enough. If there is a collision $H(m)=H(m')$, then Eve can ask Alice to sign m to get (m,s) and later replace m with m', because (m',s) is another valid signed message. (See also the comments after the discussion of birthday attacks.)

Comment. This question is of considerable practical relevance, since hash functions like MD5 and SHA-1 have been shown to not be collision-resistant (but are still considered essentially preimage-resistant, that is, one-way). In the case of MD5, this has been exploited in practice:

https://en.wikipedia.org/wiki/MD5#Collision_vulnerabilities

Example 192. Alice uses an RSA signature scheme and the (silly) hash function $H(x)=x_1+x_2$, where $x_1=x^{-1} \pmod{11}$ and $x_2=x^{-1} \pmod 7$ [with 0^{-1} interpreted as 0] to produce the signed message $(100,13)$. Forge a second signed message.

Solution. Since we have no other information, in order to forge a signed message, we need to find another message with the same hash value as $m=100$. From our experience with the Chinese remainder theorem, we realize that changing x by $7\cdot 11$ does not change $H(x)$. Hence, a second signed message is $(177,13)$.

Comment. The hash $H(m)$ for $m=100$ is $H(100)=(100^{-1})_{\bmod 11}+(100^{-1})_{\bmod 7}=1+4=5$.

Similar to what we did with RSA signatures, one can use ElGamal as the basis for digital signatures. A variation of that is the **DSA** (digital signature algorithm), another federal standard.

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

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

Not surprisingly, the hashes mandated for DSA are from the SHA family.

Birthday paradox and birthday attacks

Example 193. (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 194. 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). 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?!!)