Hash Functions

One major drawback of plain RSA is that the document itself is the signature. How can we separate the signature from the document? For this purpose, **hash functions** are introduced.

Characteristics of Hash Functions (requirements)

- **Fixed-length output**: These are special functions that, given a variable-length input, produce a fixed-length output.

- **Pre-image resistance**: It is difficult to find an input x that matches a given hash output y, where $y = H(x)$.

- **Collision resistance**: It is hard to find two different inputs x and x' such that $H(x) = H(x')$ (i.e., both inputs produce the same hash code).

- **Efficiency**: Despite the complexity, $H(x)$ should be easy to compute.

- **Avalanche effect**: Even a small change in the input (such as flipping just one bit) should result in a significant and unpredictable change in the output, ideally altering about half of the output bits.

Well-known Hash Functions

- **SHA-1**

- **MD2**

- **MD5** (Message Digest 5)

Hash functions play a crucial role in cryptography by allowing us to generate a fixed-size “fingerprint” or “digest” of a document. This makes it possible to sign the hash of the document instead of the entire document itself, making digital signatures more efficient.

Task: Storing PIN Codes

Consider a hypothetical ATM that allows users to withdraw money even when it’s not connected to the bank. This requires the ATM to store PIN codes locally.

But what happens if someone steals the codes from the database at night and gains access to customers' PIN codes? How can we prevent this? One way is to use a simple hash function, following the idea suggested by John von Neumann.

Example

A customer's PIN code is: **4531**.

1. **Square the PIN**:

```
\[
4531^2 = 20529961
\]
```

2. Take the middle 5 digits to get a new 4-digit number: **2061**. Only the first and last two digits are retained. 3. **Square again**:

```
\[
2061^2 = 4247721
\]
```

4. Take the middle 5 digits again to get another 4-digit number: **4221**. 5. Repeat the squaring and digit extraction process **1000 times**. 6. Suppose the final result is **6538**.

Now, what is the relationship between the initial **4531** and the resulting **6538**? It's as if they form a pair. Even if someone knows that **6538** is the stored value and steals the database, they cannot determine which number it originated from because the digit extraction causes a loss of information that cannot be reversed or reconstructed. However, no matter how often we run the process, we always get the same result.

Storing Passwords with Hashes

Websites also store passwords using **hash codes**. They don't store the actual password but rather transform it using an algorithm and store the result.

On some websites, you can generate various hash codes for a given password online. For example, the MD5 hash of the code **'1234'** is: \[\text{md5}(1234) = 81dc9bdb52d04dc20036dbd8313ed055 \]

Does this mean that storing hashes provides full security? Unfortunately, no. Suppose a hacker steals the database and looks up our password. On a hash-cracking site, they can input the long hash code: \[\text{hash cracking} \] Unfortunately, they might quickly figure out the password.

Solutions

1. **Use long passwords**, but these can be hard to remember. 2. **Add a "salt"** to every password when generating the hash. A salt is a fixed string that is added to the password to generate the hash.

For example: \[\text{md5}(1234 + \text{my_strong_salt}) = 0e0db19d64ce23edc1bfb52063f25028 \]

Now, try searching for this result on a hash-cracking site!

The final step is to ensure that the **salt** is well-hidden!