Cryptography

Lecture 10

- Data integrity and authentication.
- Hash functions.
- Digital signatures.

The bigger picture

"Cryptography is about communication in the presence of adversaries."

Ron Rivest

Cryptographic goals

In spite of adversaries we want to achieve (among other things)

- Confidentiality (Private communication over public channels.)
- Data integrity (Preventing malicious alteration of data.)
- Authentication (Preventing frauds.)
- Non-repudiation (Preventing denials of messages sent.)

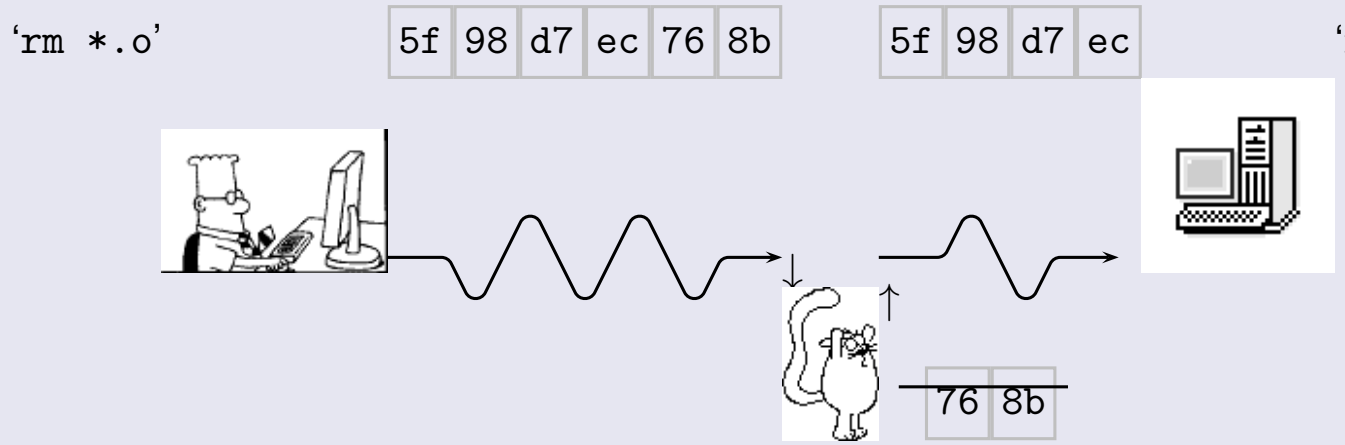
Today we will discuss the last three of these goals.

In cryptography we are not concerned with accidental (non-intentional) threats to system dependability such as hardware or software bugs, transmission errors, etc.

Data Integrity

Assuring the receiver that data has not been modified in an unauthorized manner.

Confidentiality does not imply integrity.



The adversary does not necessarily have to break the cipher to cause damage. Modification may be deletion, insertion, reordering,. . .
It may also involve several messages.

Non-cryptographic tools for data integrity

General aim is to detect errors introduced during transmission or storage. Examples:

- Parity bit. Added to (short) bit string to make total number of 1 bits even.
- Check digit. Last digit in e.g. social security number.
- Checksum. Many variants, all giving a short, often fixed-size, “fingerprint” of the message.

These can often detect (and sometimes correct) simple errors.

The cryptographic case

The adversary may be prepared to invest large resources to change the message to suit his interests, while keeping the checksum unchanged.
We need better tools to make this infeasible.

Example: Verifying downloaded software

I have just downloaded a new version of Java from Oracle's website. How can I be sure that no adversary has tampered with the file?

The same website lists SHA256 and MD5 checksums for the downloaded file.

<Live demo>

SHA256 and MD5 are two cryptographic hash functions. What properties should such functions have to convince me that my downloaded file is OK?

Cryptographic hash functions

A cryptographic hash function maps arbitrarily long messages into fixed size bit strings. Common hash sizes are 160 or 256 bits.

The hash value of a message is also called a message digest or cryptographic checksum or fingerprint.

Further, the function should be efficiently computable and one-way, i.e. given a fingerprint y, it is infeasible to find a message x with hash value y.

Collision resistance

Another important property, related to but different from being one-way, is to be collision-resistant:

A hash function h is collision-resistant if it is infeasible to find two different messages m_1 and m_2 such that $h(m_1) = h(m_2)$.

Aside: Security levels

How do you measure the security of a cryptographic primitive?

Count the number of “steps” in the best known attack (which often is brute force).

Minimal recommended security level today is 2^{80}, or in common wordings, 80 bits security.

Recall that DES, with 56 bits security, is broken.

AES is expected to provide 128 bits security, which should be adequate for quite some time.

Note that 10 more bits means $2^{10} \approx 1000$ times more work for the attacker.

Attacks against hash functions

As for any cryptographic primitive, we should define the notion of an attack against a hash function, so that we have a goal against which to measure security.

What is an attack against a hash function h with n bit hash values?

At least two different attacks are of interest:

- Given a hashvalue y, find a message x, such that $h(x) = y$.

  This is an attack against one-wayness. A brute force attack will take $O(2^n)$ attempts, so if brute force is the best attack, we get n bits security.
- Find a collision, i.e. find x_1 and x_2 with $h(x_1) = h(x_2)$.

  How much effort does it take to find a collision by brute force on an n-bit hash function?

The birthday “paradox”

How many people need there be in a group in order to have at least 50 % probability that two of them have the same birthday?

Solution: With k people, the probability that all birthdays are different is

$$\frac{364}{365} \cdot \frac{363}{365} \cdot \ldots \cdot \frac{365-(k-1)}{365}$$

Computation shows that this is less than 0.5 when k=23.

The general case

Two variants

We repeatedly draw a card from a deck with N cards, note which card it is and put it back in the deck.

- The number of draws necessary to make the probability of a collision (drawing the same card more than once) at least 0.5 is, for large N, approximately $1.18\sqrt{N}$. <Live proof sketch>.
- The expected number of draws until the first collision occurs is, for large N, approximately $1.25\sqrt{N}$.

For hash functions:
The expected number of messages hashed with an n-bit ideal hash function until a collision occurs is $1.25 \cdot \sqrt{2^n} = 1.25 \cdot 2^{n/2}$.

Thus an n-bit hash function provides only $n/2$ bit security against this birthday attack.

Some remarks

- An “ideal” hash function should behave like a random function.
- This is not the same notion as hash functions in connection with hash tables.
- The birthday attack applies to any hash function; it is not a shortcoming of a particular construction.
- 256 bit hash values (as produced by SHA256) give 128 bits of security against the birthday attack, matching the level of AES encryption.
- Hash functions have many applications in cryptography.

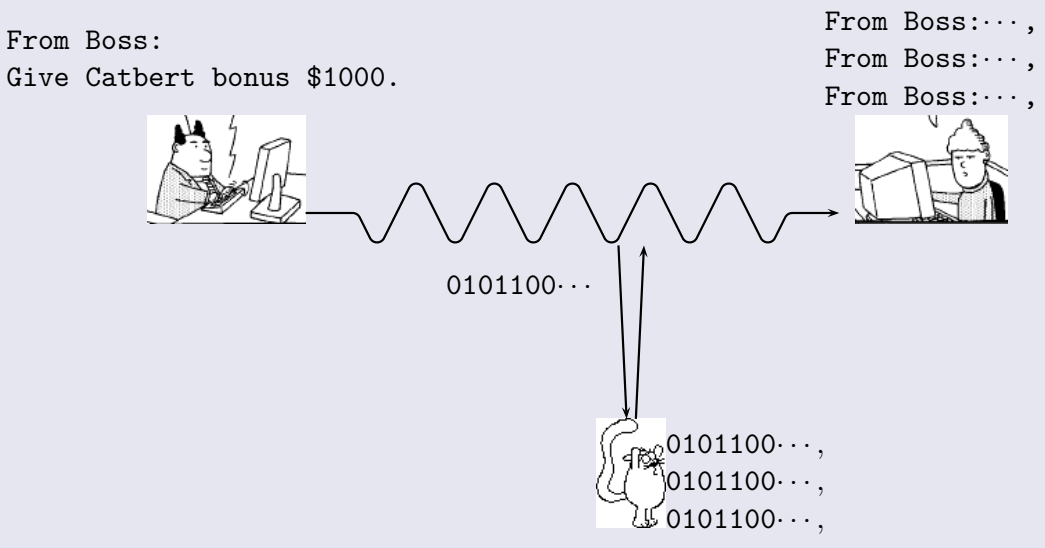
Back to verification of downloads

If we use a collision-resistant hash-function, will checking the hash value convince us of the integrity of the file?
Depends on where we got the correct hash value from. The same website?

Message Authentication

Assuring Receiver that data originates from its claimed source / presumed source.

A system lacking message authentication is vulnerable to replay attacks, among others.



(Catbert)

Traditional authentication

Since millennia, document authors have signed documents with ink signatures. These have various purposes:

- A personal greeting, in letters to friends and family.
- A confirmation that the content of the document expresses the signer's opinion/wishes/intention. Example: A will.
- A confirmation that the signer is bound by the conditions expressed in the document. Example: A business contract.

Important properties of ink signatures

- A signature is unique to the signer, and can be identified by others as being performed by him/her.
- Nobody else can imitate (forge) the signature; this will be discovered if the signature is questioned.
- The signer cannot later deny having signed the document.

Digital signatures

Setting

Alice sends a contract M to Bob, who wants the contract to be digitally signed so that

- Bob can be sure that the document really comes from Alice,
- Alice cannot later deny that she sent the contract.

Assume that Alice has public RSA key **pk**=(N, e), and corresponding private key **sk**=(N, d).
To sign M, Alice "decrypts" M, computing the signature σ = **Dec**(**sk**, M), and sends M and σ to Bob.

Note: For this slide, and the next few, we assume that M can be represented as an integer in $\mathbb{Z}_N$.

Why does this achieve the two goals above?

Verifying a signature

This signing practice has the properties:

- Bob can compute **Enc**(**pk**, σ) and verify that this equals M. Thus the sender knew Alice's private key, something known only to Alice.
  Note that this verification uses only Alice's public key.
- In case of later dispute, Bob can exhibit, e.g. in court, his computations. Note that it is necessary here that Bob can prove that Alice's public key is really hers.

If M needs to be confidential, Alice may encrypt M and possibly σ before sending them. The reasoning above is not affected (but care is needed in doing this right).

The RSA signature scheme

Algorithms

- **KeyGen(λ) $\rightarrow$ (pk,sk):**
  Choose two λ bit primes p and q and compute $N = p \cdot q$.
  Choose $e \in \mathbb{Z}^*_{\varphi(N)}$ such that $\gcd(e, \varphi(N)) = 1$.
  Compute $d = e^{-1} \bmod \varphi(N)$.
  Set **pk** = (N, e) and **sk** = (N, d).
- **Sign(sk,#):** $\mathbb{Z}^*_N \rightarrow \mathbb{Z}^*_N$
  Sign($(N, d), m$**)** = $m^d \bmod N$.
- **Verify(pk,#$_1$#$_2$):** $\mathbb{Z}^*_N \times \mathbb{Z}^*_N \rightarrow$ **Bool**
  Verify($(N, e), m, \sigma$**)** = $\sigma^e == m \bmod N$.

Correctness

We need to prove that if **(pk,sk)** $\leftarrow$ **KeyGen(**λ**)**, then

Verify(pk,m,**Sign(sk**,m**))** = **True**.

Can you do it?

A way to cheat?

Alice might try to cheat by intentionally leaking her private key and then claim that this happened before M was sent.

This is a serious problem for digital signatures.

Possible countermeasures (not without difficulties):

- Include timestamps in signature.
- Require reporting lost keys to some trusted authority.

An attack on RSA signatures

Suppose Bob has received from Alice two signed documents (m_1, s_1) and (m_2, s_2).

He can then himself produce the signed document $((m_1 \cdot m_2) \bmod N, (s1 \cdot s2) \bmod N)$, which is valid as a document signed by Alice.

It might seem unlikely that having Alice's signature on $m_1 \cdot m_2$ should be useful to Bob, but this feature should be avoided. How?

Signing a hash of the message

Common practice is to sign and verify not the document itself, but a hash value of the message.

This practice

- takes care of the restriction a couple of slides ago that a message to be signed must be representable in $\mathbb{Z}_N^*$; arbitrary messages can be signed.
- takes care of the problem on the previous slide, since the hash values do not preserve the multiplicative structure.
- makes signing efficient, since the public-key operation is done only on the hash value.

What properties do we need from the hash function?

Security for signatures

We need to define a proper notion of security for signature schemes and consider the following game:

- The adversary is allowed to produce any (polynomially bounded) number of documents and have them signed.
- After that, the adversary is asked to produce a pair (m, σ) where σ is a signature on m and m is not among the documents the adversary asked to get signed.

A signature scheme has adaptive chosen message security if no efficient adversary can win this game with non-negligible probability.

Textbook RSA signing is not secure.

We already saw this:

The adversary picks any two messages m_1 and m_2, has them signed with signatures s_1 and s_2 and produces finally $(m_1 \cdot m_2, s_1 \cdot s_2)$.

RSA-PSS

This an improved signature scheme, which we will not present in detail. In the same spirit as RSA-OAEP, the message will be padded and randomized before signing with the raw RSA signature.
RSA-PSS signing adds only minor computational effort (some xor-ing and hash computations); raw RSA signing is the dominant computational cost.

Random Oracle Model

Practical hash functions have complicated definitions, unsuitable for proof.

However, they are designed to behave as random functions.

Idea: Prove properties of e.g. RSA-PSS assuming that hash functions used are random functions (i.e. do the proof in the Random Oracle Model), then hope that the system has more or less the same properties when non-ideal hash functions are used.

One can prove that RSA-PSS is secure in the Random Oracle Model if the RSA assumption holds.

Verifying downloaded code, revisited

As we saw earlier, checking the hash value of a downloaded file only verifies integrity if we know what the hash value of the original file is. How can we get hold of that?

Example: Verifying a GMP download

I have just downloaded a new version of GMP (the GNU Multiple Precision Arithmetic Library). How can I verify that it has not been tampered with?

< Live Demo >

Other signature schemes

- Other public-key encryption schemes, e.g. Elgamal, can be used for signing, in the same spirit as RSA.
- DSA, the digital signature algorithm, is also based on discrete logs. More on this next time.
- Many challenge-response protocols (e.g. the Fiat-Shamir protocol), can be turned into signature schemes by replacing the challenge by the hash of the message to be signed. The signature is the response.

Textbook references

Chapter references for this lecture's material

- Katz-Lindell: sections 4.1, 5.1, 5.4.2, 12.1, 12.2, 12.4.
- Stallings: sections 11.1, 11.3, 11.4, 13.1, 13.6.

Next time

- One-way and trapdoor functions.
- Hash function constructions.
- Weaknesses in hash functions.
- Message Authentication Codes (MACs).
- Elgamal and DSA signature schemes.