NTNU | Norwegian University of Science and Technology

DIGITAL SIGNATURES

TTM4135 - Lecture 10

Tjerand Silde

05.02.2026

Motivation

- Obtaining digital signatures is one of the major benefits of public key cryptosystems, allowing for authenticity and integrity of communication without requiring access to previously shared secret keys
- In some countries digital signatures are legally binding in the same way as handwritten signatures to prove authenticity of documents
- Digital signature are deployed widely in authentication protocols and management of public keys as a public key infrastructure (PKI)

Contents

Confidentiality and Authentication

- Message authentication codes (MACs) allow only entities with the shared secret to generate valid tags, providing data integrity and authentication
- Digital signatures use public key cryptography to provide these properties and more: only the owner of the private key can create a correct signature
- Digital signatures provide the *non-repudiation* security service because a judge can decide which party formed the signature (unlike for MACs)

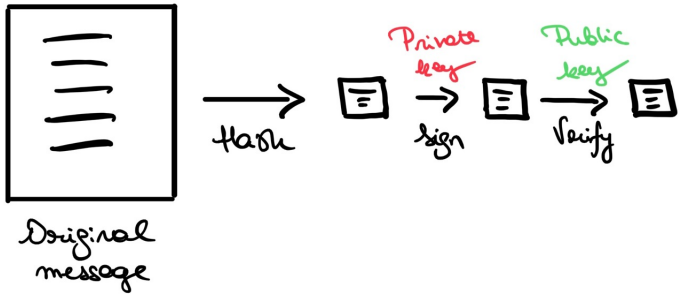
Comparing Physical and Digital Signatures

Physical signatures	Digital signatures
Produced by human	Produced by machine
Same on all documents	Function of message
Easy to recognize	Requires computer to check

Both types of signature need to be difficult to forge

Contents

Digital Signatures



Digital Signatures Overview

- Digital signatures allow a *signer* Alice with a *public verification key* K_V to *sign* a message, using her *associated secret signing key* K_S, in such a way that *anyone who knows* K_V *can verify* the signature
- More precisely, anyone who knows K_V can verify:
  - The message originated from Alice
  - The message was not modified in transit

Digital Signature Example

Example: A software company which disseminate a software update to clients.

- Should be possible for any client to verify that the update is authentic.
- Should not be possible for a malicious third party to convince the clients to accept an update that was not released by the company:
  - The company can generate a digital signature key pair K_S and K_V.
  - Distribute K_V to its clients (e.g. in the software), keep K_S private.
  - When releasing an update m, it computes a signature σ; sends (m, σ).
  - All clients who have K_V can verify that the software update is legit.
  - No malicious adversary should be able to make a client accept a fraudulent update and signature (m', σ') under the key K_V (even if m' is close to m).

Comparison with MACs

- Digital signatures are a public-key analogue of MACs.
- Digital signatures simplifies key distribution and management.
- Especially when one sender needs to communicate with multiple receivers (as in the example in the previous slide).
- This way, the sender avoids having to set up one key per recipient .
- Digital signatures are *publicly verifiable* and *transferable* (Bob can check Alice's signature, and pass it on to Eve who can also check).
- Digital signatures also provide *non-repudiation* – important e.g. in legal applications. This cannot be done with a symmetric key object.
- MACs are shorter and 2-3x more efficient to generate/ verify.

Elements of Digital Signatures

- A digital signature scheme has three algorithms:
  - key generation
  - signature generation
  - signature verification

- Key generation outputs two keys:
  - a private *signing key* K_S
  - a public *verification key* K_V

Signature Generation Algorithm

Suppose that Alice wishes to generate a signature on a message m.

Inputs
- Alice's private *signing key*, K_S
- The message m

Output
- Signature $\sigma = \mathsf{Sign}(m, K_S)$

It should be possible to choose the message m as any (long) bit string.

Signature Verification Algorithm

Suppose that Bob wishes to verify a claimed signature σ on the message m.

Inputs
- Alice's public *verification key*, K_V
- The provided message m
- The claimed signature σ

Output
- A boolean value $\mathrm{Vf}(m, \sigma, K_V) = \text{true or false}$

Anyone should be able to use the verification key to verify a valid signature.

Properties Required of Verifying Function

Correctness If $\sigma = \mathsf{Sign}(m, K_S)$ then $\mathsf{Vf}(m, \sigma, K_V) = \mathsf{true}$, for any matching signing/verification keys K_S and K_V

Unforgeability It is computationally infeasible for anyone without K_S to construct m and σ such that $\mathsf{Vf}(m, \sigma, K_V) = \mathsf{true}$

- The signing algorithm Sign *may* be randomized so that there are many possible signatures for any single message
- For stronger security we assume that an attacker has access to a *chosen message* oracle: forging a (new) signature should be difficult even if the attacker can obtain many signatures on messages of his choice

Security Goals

An attacker may try to break a digital signature scheme in several ways:

Key recovery The attacker attempts to recover the private signing key from the public verification key and some known signatures

Selective forgery The attacker chooses a message and attempts to obtain a signature on that message

Existential forgery The attacker attempts to forge a signature on any (even a meaningless) message not previously signed

Modern digital signatures are considered secure if they can resist *existential forgery under a chosen message attack* (the strongest notion).

Contents

RSA Signatures

- Proposed back in 1978 along with the RSA encryption scheme
- Still widely deployed, particularly where signatures are verified many times with a small public exponent (makes it efficient)
- Standardized in various places including the NIST FIPS186 standard
- Can be broken by an attacker who can factorize the modulus

RSA Signatures - Key Generation

RSA signature keys are generated in the same way as RSA encryption keys:

- A modulus $n = pq$ is computed from random large primes p and q.
- Two exponents e and d are generated with $ed \bmod \phi(n) = 1$.
- The private signing key is $K_S = (d, p, q)$.
- The public verification key is $K_V = (e, n)$.
- A hash function H is also a fixed public parameter of the scheme.

RSA Signature Operations

Signature generation: Inputs are the message m, the modulus n and the private exponent d: compute signature $\sigma = \mathsf{H}(m)^d \bmod n$.

Signature verification: Inputs are the message m, the claimed signature σ and the public key (e, n):

1. Compute the hash of the messages as $h' = \mathsf{H}(m)$.
2. Output true if $\sigma^e \bmod n = h'$ and otherwise false.

Hash Functions for RSA Signatures

- In the above signature definition, we could let H be a standard hash function, such as SHA-256, where the output is 256 bits.
- The following are also commonly used (standardized) choices:
  1. A *full domain hash* is an implementation of h which can take values randomly in the range 1 to n
  2. PSS is a probabilistic hashing function similar to OAEP used for RSA encryption and is standardized in the PKCS #1 standard RFC 8017

Contents

Discrete Logarithm Signatures

- There are several digital signatures whose security relies on the difficulty of the discrete log problem.

- We will look at four versions:
  1. The original ElGamal signature scheme from 1985 set in $\mathbb{Z}_p^*$.
  2. Schnorr signatures from 1989 (patented until 2008, popular now).
  3. The digital signature algorithm (DSA) standardized by NIST in the Digital Signature Standard. A highly optimized version of ElGamal signatures.
  4. The version of DSA based on elliptic curve groups, known as ECDSA.

ElGamal signature scheme in $\mathbb{Z}_p^*$

- Let p be a large prime and g a generator for $\mathbb{Z}_p^*$. The private signing key is an integer $S_K = x$ such that $0 < x < p - 1$.
- The public verification key is $K_V = y = g^x \bmod p$. The values p, g and y are public knowledge.
- Suppose that Alice wants to sign an arbitrary message m.

Signature Operations

Signature generation To sign the message m with signing key x:

1. Select random a k, $0 < k < p - 1$ and compute $r = g^k \bmod p$
2. Compute $s = k^{-1}(\mathrm{H}(m) - xr) \bmod (p - 1)$
3. The signature is the pair $\sigma = (r, s)$

Signature verification Given the message m, claimed signature $\sigma = (r, s)$ and verification key y:

1. Verify whether $g^{\mathrm{H}(m)} = y^r \cdot r^s \bmod p$

This works since: $r^s = (g^k)^{k^{-1}(\mathrm{H}(m) - xr)} = g^{\mathrm{H}(m)} \cdot g^{-xr} = g^{\mathrm{H}(m)} \cdot y^{-r}$.

Schnorr Signature Scheme in $\mathbb{Z}_p^*$

- Let p be a large prime and g a generator for $\mathbb{Z}_p^*$. The private signing key is an integer $S_K = x$ such that $0 < x < p - 1$.
- The public verification key is $K_V = y = g^x \bmod p$. The values p, g and y are public knowledge.
- Suppose that Alice wants to sign an arbitrary message m.

(This is exactly the same setup as for ElGamal signatures.)

Signature Operations

Signature generation To sign the message m with signing key x:

1. Select random a k, $0 < k < p - 1$ and compute $r = g^k \bmod p$
2. Let $e = \mathrm{H}(r||m)$ be the hashed message.
3. Compute $s = k - xe \bmod (p - 1)$
4. Signature is the pair $\sigma = (e, s)$

Signature verification Given message m and claimed signature $\sigma = (e, s)$ and verification key y:

1. Verify whether $e = \mathrm{H}(g^s \cdot y^e||m)$

This works since: $g^s \cdot y^e = g^{k-xe} \cdot (g^x)^e = g^k \cdot (g^e)^{-x} \cdot (g^e)^x = g^k = r$.

Digital Signature Algorithm (DSA)

- First published by NIST in 1994 as FIPS PUB 186, allowed for legacy use.
- Based on ElGamal signatures, but simpler calculations and shorter signatures due to restricting the calculations to a *subgroup* of $\mathbb{Z}_p^*$.
- Designed for use with the SHA family of hash functions.
- Avoids some attacks that ElGamal signatures may be vulnerable to.

DSA parameters

- p, a prime modulus of L bits and q, a prime divisor of $p-1$ of N bits
- Valid combinations of L and N are: $(L = 1024, N = 160)$, $(L = 2048, N = 224)$, $(L = 2048, N = 256)$, $(L = 3072, N = 256)$
- Generator $g = h^{\frac{p-1}{q}} \bmod p$, where h is any integer, $1 < h < p-1$
- H, the SHA hash family variant which outputs an N-bit digest
- NIST Special Publication SP 800-57 does *not* approve the first choice of parameters $(L = 1024, N = 160)$ nowadays

Key and signature generation

Key generation Similar but not identical to ElGamal and Schnorr:

1. Choose a random integer x with $0 < x < q$
2. Then x is the secret signing key, and
3. $y = g^x \bmod p$ is the public verification key

Signature generation For every message m to be signed

1. Choose k at random with $0 < k < q$ and set $r = (g^k \bmod p) \bmod q$
2. Compute $s = k^{-1}(\mathrm{H}(m) - xr) \bmod q$
3. The signature consists of the pair $\sigma = (r, s)$

Signature verification

A received signature of message m is a pair (r, s). To verify:

- Check that $0 < r < q$ and $0 < s < q$
- Compute $w = s^{-1} \bmod q$ and let:

$$\begin{aligned} u_1 &= H(m)w \bmod q \\ u_2 &= rw \bmod q \end{aligned}$$

- Check whether $(g^{u_1} y^{-u_2} \bmod p) \bmod q = r$

(Exercise: show that this works out.)

Comparison with ElGamal Signatures

The verification equation is actually the same as for ElGamal signatures, except that all exponents and the final result are reduced modulo q.

- The complexity of signature generation is now mainly just one exponentiation with a short exponent (such as 224 or 256 bits)
- Signature verification requires two such short exponentiations
- The signature size is only $2N$ bits:
  - 448 bits when $N = 224$
  - 512 bits when $N = 256$

Elliptic Curve DSA (ECDSA)

- Elliptic curve DSA (ECDSA) is defined in the standard FIPS 186-5
- Signature generation and verification is similar to DSA except that:
  - the parameter q becomes the order of the elliptic curve group
  - multiplication modulo p is replaced by the elliptic curve group operation
  - after the operation on the group elements only the x-coordinate (an element in the underlying field) is kept

ECDSA Parameters

Parameters

- E an approved elliptic curve field and equation
- G the elliptic curve group generator, or base point
- n the order of the elliptic curve group and a prime number
- H the SHA-2 hash family variant which outputs an N-bit digest

Parameters are chosen from the NIST approved curves

Interlude: Reminder on Elliptic Curves

- Elliptic curves are algebraic structures formed from cubic equations
- An example is the set of all (x, y) pairs which satisfy the equation:

$$y^2 = x^3 + ax + b \bmod p$$

- This example is a curve over $\mathbb{Z}_p$ but they can be defined over any field
- Once an identity element is added, a binary operation (usually called *addition*) can be defined on these points
- With this operation the elliptic curve points form an *elliptic curve group*

Elliptic Curve Computations

- We write $P + Q = R$ for adding curve points P and Q with result R
- The *elliptic curve discrete log problem* is to find the value of k, given a point P and a generator G so that $P = [k]G = G + G + \ldots + G$ (k times)
- Efficient computation of elliptic curve scalar multiplication can use the *double-and-add algorithm* (similar to square-and-multiply over finite fields)

ECDSA Key Generation

Key generation

1. Choose a random integer d with $0 < d < n$ as the signing key
2. Compute $Y = [d]G$ as the public verification key in the group G

The standard requires that before a public key is used it is checked to be a point on the curve which is different from G and the identity on curve E.

Signature Generation

Signature generation For every message m to be signed:

1. Hash the message as $e = \mathrm{H}(m)$.
2. Choose k at random with $0 < k < n - 1$ and compute $(x, y) = [k]G$. Set $r = x$, but return to step 2 if $r = 0$.
3. Set $s = k^{-1}(e + rd) \bmod n$.
4. The signature consists of the pair $\sigma = (r, s)$

Note that r is the x-coordinate of an elliptic curve point modulo p while s is an integer modulo n.

Signature Verification

A received signature of message m is a pair (r, s). To verify:

- Check that $0 < r < p$ and $0 < s < n$
- Compute $w = s^{-1} \bmod n$ and $e = H(m)$ and set:

$$\begin{aligned} u_1 &= ew \bmod n \\ u_2 &= rw \bmod n \end{aligned}$$

- Compute the point $(x, y) = [u_1]G + [u_2]Y$. The signature is valid if (x, y) is not the identity element on the curve E and

$$r = x \pmod{p}$$

ECDSA Variants

The new FIPS 186-5 standard also includes other important signature schemes using elliptic curve groups:

1. Deterministic ECDSA signatures
   - The per-message key k is deterministically computed as a function (based on HMAC) of the message to be signed and the private signing key d
   - Recommended when good random number generator is not available

2. EdDSA signatures
   - Uses the Edwards curve 25519
   - Deterministic version of Schnorr signatures

ECDSA vs. DSA

- Because of the clever design of DSA, signatures using ECDSA are generally no shorter than signatures using DSA for the same security level.
- ECDSA signature size varies with the curve used. For approved curves this can vary from 448 bits to more than 1024 bits.
- ECDSA public keys are however shorter than DSA public keys.

Questions?