Lecture 11: Digital Signatures

TTM4135

Relates to Stallings Chapters 13

Spring Semester, 2025

Motivation

- Obtaining digital signatures is one of the major benefits of public key cryptosystems
- In some countries digital signatures are legally binding in the same way as handwritten signatures
- Digital signature are deployed widely in authentication protocols and management of public keys

Outline

Confidentiality and authentication

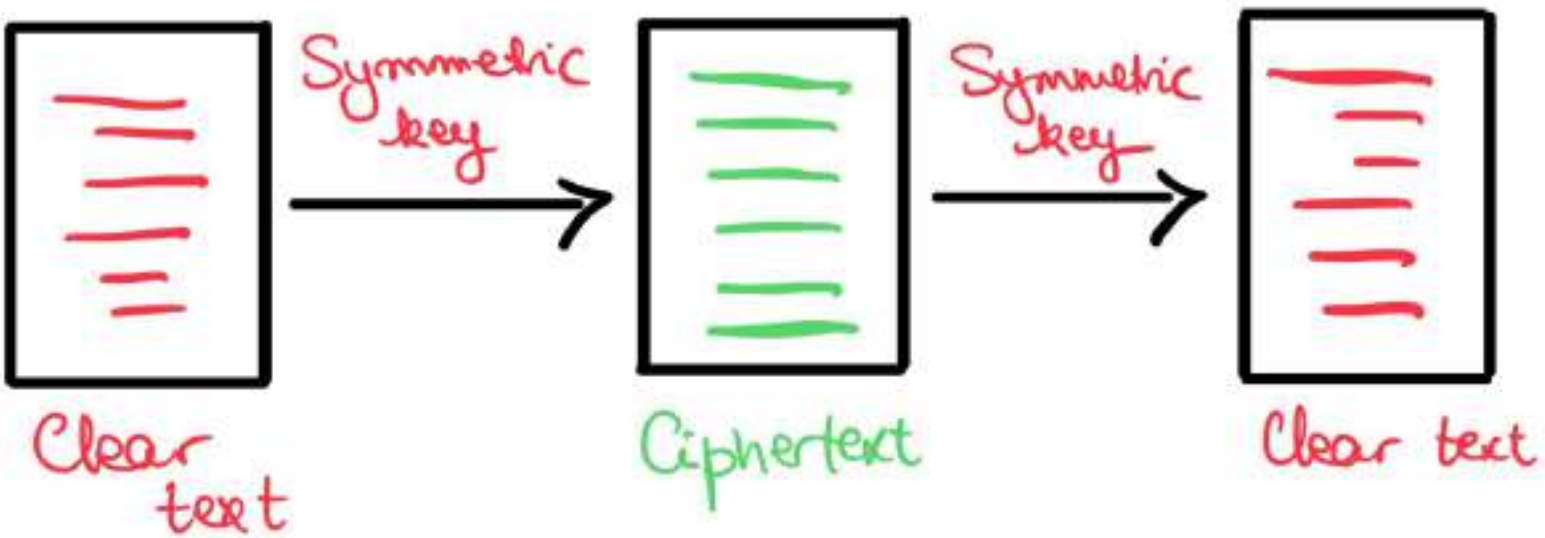
- Message authentication codes (MACs) allow only an entity with the shared secret to generate a valid MAC tag, providing data integrity and data authentication
- Digital signatures use public key cryptography to provide the properties of a MAC and more: only the owner of the private key can generate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

Comparing physical and digital signatures

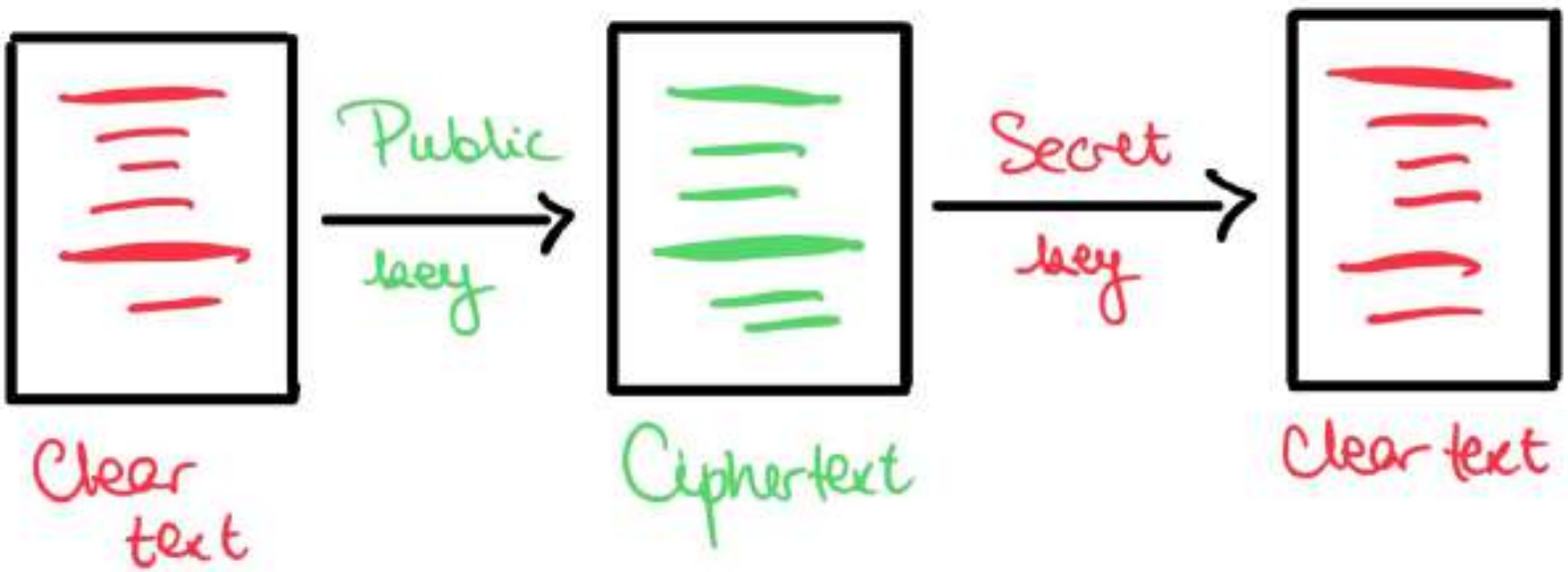
Physical signatures	Digital signatures
Produced by human	Produced by machine
Same on all documents	Function of message
Easy to recognise	Requires computer to check

Both types of signature need to be difficult to forge

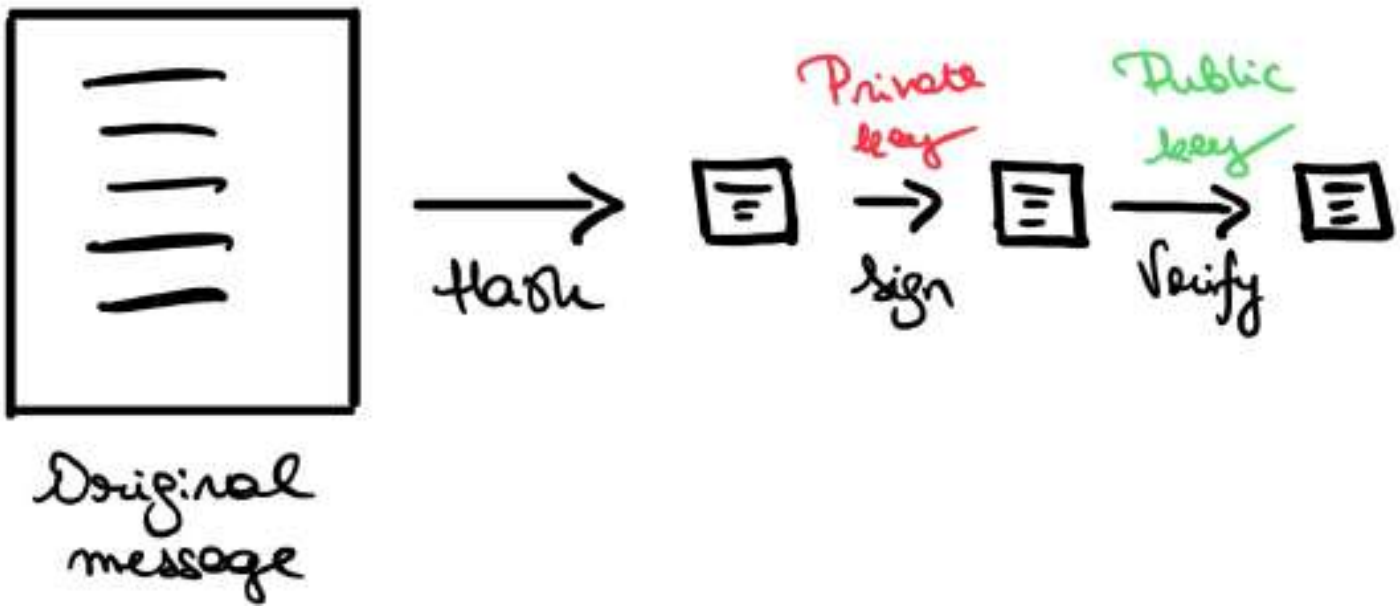
Symmetric Key Encryption



Public Key Encryption



Signatures



Digital signatures overview

- Digital signatures allow a *signer* Alice with a *public key* K_V to *sign* a message, using her *associated secret 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 signatures: an example

Ex: software company,disseminate a software update to clients.

- Should be possible for any of its clients to verify that the update is authentic.
- Shouldn't be possible for a malicious third party to convince the clients to accept an update that was not released by the company
  - Company can generate a private signing key K_S, along with a public verification key K_V.
  - Distribute K_V to its clients (e.g. could be a part of the original purchase bundle), keep K_S private.
  - When realeasing an update m, company computes a signature σ; sends (m, σ).
  - All clients who have K_V can verify the software update.
  - No malicious adversary should be able to make a client accept a fraudulent update (m', σ') under the key K_V (even if m' is close to m).

Comparison with MACs

- Public-key analogue of MACs
- Using digital signatures rather than MACs 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 eg 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 *signature generation key* or simply *signing key*, K_S
  - a public *signature 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 $\sigma = \text{Sig}(m, K_S)$

- Only the owner of the signing key should be able to generate a valid signature
- It should be possible to choose the message as any 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 message m
- The claimed signature σ

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

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

Properties required of verifying function

Correctness If $\sigma = \mathrm{Sig}(m, K_S)$ then $\mathrm{Ver}(m, \sigma, K_V) = \text{true}$, for any matching signing/verification keys

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

- The signing algorithm Sig *may* be randomised so that there are many possible signatures for a single message
- For a stronger security definition we often assume that an attacker has access to a *chosen message* oracle: forging a (new) signature should be difficult even if the attacker can obtain 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 message not previously signed, even if it is a meaningless message

Modern digital signatures are considered secure only if they can resist *existential forgery under a chosen message attack*.

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
- Standardised in various places including the latest NIST FIPS186 standard (2023)
- Can be broken by an attacker who can factorise 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

$$ed \bmod \phi(n) = 1$$

- Private signing key is $\mathtt{sk} = (d, p, q)$
- Public verification key is $\mathtt{pk} = (e, n)$
- A hash function h is also required and should be a fixed public parameter of the signature scheme

RSA signature operations

Signature generation: Inputs are the message m, the modulus n and the private exponent d

1. Compute signature $\sigma = h(m)^d \bmod n$

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

1. Compute $h' = h(m)$
2. Check whether $\sigma^e \bmod n = h'$ and if so output true, otherwise output false

Hash functions for RSA signatures

- In the above signature definition, we could let h be a standard hash function, such as SHA-256
- The following two choices both can be proven secure with suitable assumptions
  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 standardised in the PKCS #1 standard, available as RFC 8017

Discrete logarithm signatures

- These are digital signatures whose security relies on the difficulty of the discrete log problem
- We look at four versions
  1. Original Elgamal signatures from 1985 set in $\mathbb{Z}_p^*$
  2. Schnorr signatures
  3. The digital signature algorithm (DSA) standardised 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, g a generator for $\mathbb{Z}_p^*$. Private signing key is x with $0 < x < p - 1$
- Public verification key is $y = g^x \bmod p$. Values p, g and y are public knowledge
- Suppose Alice wants to sign a value m which is an integer between 0 and $p - 1$

Signature operations

Signature generation To sign message m with signing key x:

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

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

1. Verify that $g^m \equiv y^r r^s \bmod p$

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

- Let p be a large prime, g a generator for $\mathbb{Z}_p^*$. Private signing key is x with $0 < x < p - 1$
- Public verification key is $y = g^x \bmod p$. Values p, g and y are public knowledge
- Suppose Alice wants to sign a value m which is an integer between 0 and $p - 1$

Signature operations

Signature generation To sign message m with signing key x:

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

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

1. Let $r_v = g^s y^e$
2. Let $e_v = H(r_v||m)$
3. Accept if $e_v = e$

Digital signature algorithm (DSA)

- First published by NIST in 1994 as FIPS PUB 186
- In the latest version of the NIST standard FIPS PUB 186-5 (February 2023) DSA is no longer recommended (can be used for verification in legacy applications)
- Based on Elgamal signatures
- Simpler calculations and shorter signatures due to restricting the calculations to a *subgroup* of $\mathbb{Z}_p^*$ or to an elliptic curve group
- 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
- q, a prime divisor of $p - 1$ of N bits
- Valid combinations of L and N are: $(L = 1024, N = 160)$, $(L = 2048, N = 224)$, $(L = 2048, N = 256)$, $(L = 3072, N = 256)$
- $g = h^{\frac{p-1}{q}} \bmod p$, 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 $(L = 1024, N = 160)$

Key and signature generation

Key generation Do this once at the start

1. Choose a random integer x with $0 < x < q$
2. x is the secret signing key
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. Set $s = k^{-1}(H(m) - xr) \bmod q$
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 $(g^{u_1} y^{-u_2} \bmod p) \bmod q = r$

Comparison with Elgamal signatures

The verification equation is the same as for Elgamal signatures, except that all exponents are reduced modulo q and the final result is also 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 $N = 256$

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
- Description on following slides omits some of the detailed checks in FIPS186-5

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, previous lecture

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

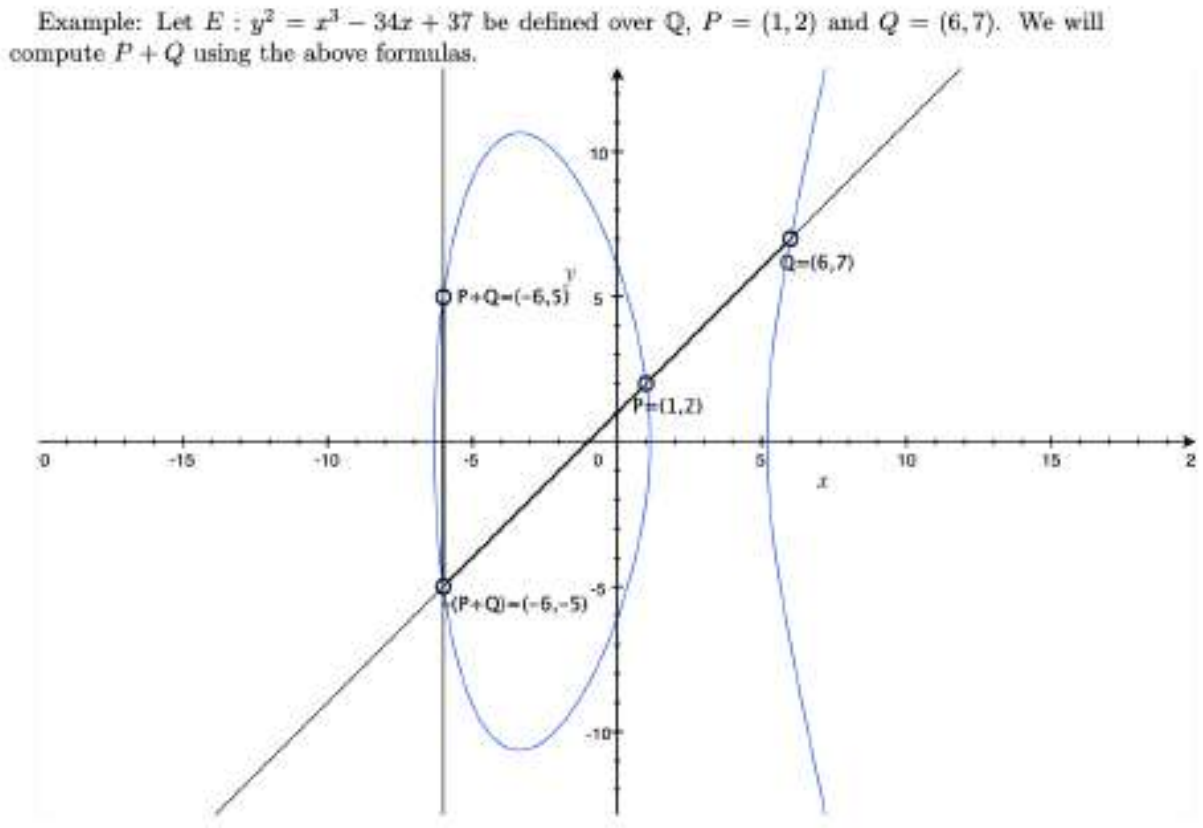
$$y^2 = x^3 + ax + b \bmod p$$

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

Elliptic curve computations

- The elliptic curve group operation could be denoted by any symbol, but by convention is it usually called elliptic curve *addition*
- We write $P + Q = R$ to show the group operation on curve points P and Q with result R
- The *elliptic curve discrete log problem* is to find the value of m, given a point P and a generator G so that $P = mG = G + G + \ldots + G$ (m times)
- Efficient computation of elliptic curve multiplication can use the *double-and-add algorithm*, by replacing multiplication in the square and multiply algorithm with addition

Elliptic Curve – visual representation



https://www.umsl.edu/~siegelj/information_theory/projects/elliptic_curves_group_law.pdf

ECDSA key generation

Key generation

1. Choose a random integer d with $0 < d < n$ as the secret signing key
2. Compute $Y = dG$ 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 G different from the identity

Signature generation

Signature generation For every message m to be signed

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

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

Signature verification

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

- Check that $0 < r < n$ 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 in the curve E and

$$r \equiv x \pmod{n}$$

ECDSA variants

The new FIPS 186-5 standard 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 shorter than DSA public keys