

Digital Signatures

Gaspare FERRARO

CyberSecNatLab

Matteo ROSSI

Politecnico di Torino



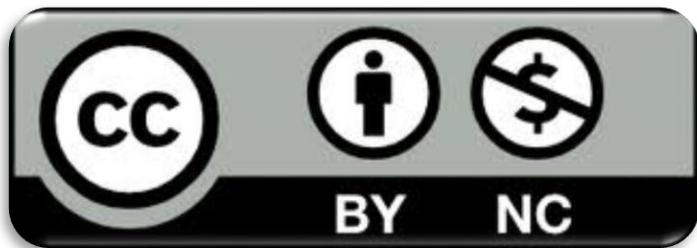
<https://cybersecnatlab.it>

License & Disclaimer

2

License Information

This presentation is licensed under the
Creative Commons BY-NC License



To view a copy of the license, visit:

<http://creativecommons.org/licenses/by-nc/3.0/legalcode>

Disclaimer

- We disclaim any warranties or representations as to the accuracy or completeness of this material.
- Materials are provided “as is” without warranty of any kind, either express or implied, including without limitation, warranties of merchantability, fitness for a particular purpose, and non-infringement.
- Under no circumstances shall we be liable for any loss, damage, liability or expense incurred or suffered which is claimed to have resulted from use of this material.

Goal

3

- Give the definition and show usage of digital signatures
- Show the differences between hash, MAC, and digital signatures
- Learn how to perform digital signatures with RSA
- Introduce the DSA algorithm and its weaknesses

Prerequisites

4

➤ Lectures:

- *CR_0.1 - Number Theory and modular arithmetic*
- *CR_1.1 - Introduction to cryptography*
- *CR_2 - Public-key cryptography*
- *CR_3.1 - Hash Functions*

Outline

5

- Introduction
- Digital Signatures from RSA
- The Digital Signature Algorithm
- Nonce reuse in DSA

Outline

6

- Introduction
- Digital Signatures from RSA
- The Digital Signature Algorithm
- Nonce reuse in DSA

Introduction

7

- Recap from lecture CR_3.1:
 - **Message integrity**: can the recipient be confident that the message has not been accidentally modified?
 - **Authentication**: can the recipient be confident that the message originated from the sender?

Introduction

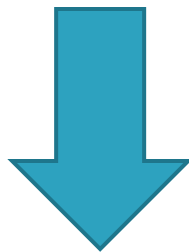
8

- A step forward **non-repudiation**:
 - Protection against an individual falsely denying having performed a particular action
 - Provides the capability to determine whether a given individual took a particular action such as creating information, sending a message, approving information, and receiving a message

Introduction

9

Informally: a digital signature is like a MAC, but with
public key cryptography



No need of a shared secret key

Hash vs MAC vs Signatures

10

Primitive	Integrity	Authentication	Non-repudiation
Hash	Yes	No	No
MAC	Yes	Yes	No
Digital Signature	Yes	Yes	Yes

Signatures vs MACs

11

- Pros of digital signatures
 - No need of sharing a key
 - Non-repudiation property

- Cons of digital signatures
 - Slow compared to MACs

Signatures in practice

12

- In practice, a digital signature is a pair of functions, *Sign* and *Verify*, such that
 - *Sign* takes *an hash* of a message of arbitrary length and a key and produces a fixed-length string, called *signature*
 - *Verify* takes *the hash* of the message, the key and the signature, and outputs true if the signature is valid and false otherwise

Why hashes?

13

- Hashing is useful to avoid to have:
 - too short messages
 - have messages longer than the modulus used in the sign and verify functions
- Recall the vulnerabilities presented in the lecture CR_2.3 - Attacks on RSA

Outline

14

- Introduction
- **Digital Signatures from RSA**
- The Digital Signature Algorithm
- Nonce reuse in DSA

Signing with RSA

15

- A basic signature scheme using RSA can be constructed as follows:

- The Sign function for a message m is

$$s = m^d \bmod n$$

- The Verify function is

$$m = s^e \bmod n$$

- Issues?

Forgery

16

- Some signatures are independent from the value of d :
 - The signature of 0 is always 0
 - The signature of 1 is always 1
 - The signature of $n-1$ is always $n-1$

Blinding

17

- Using the homomorphic properties of RSA, we can sign an arbitrary message M without asking directly to the oracle to sign it:
 - Select a value R
 - Ask to sign $(R^e M) \rightarrow \text{Sign}(R^e M) = (R^e M)^d = RM^d \bmod n$
 - Use the multiplicative inverse of R to get a signature for M from the signature of $(R^e M)$:
 - $\text{Sign}(M) = \text{Sign}(R^e M)R^{-1} = (RM^d)R^{-1} = M^d \bmod n$

Outline

18

- Introduction
- Digital Signatures from RSA
- **The Digital Signature Algorithm**
- Nonce reuse in DSA

History

19

- In 1982, the US government asked for proposals for digital signature standards
- In 1991, the Digital Signature Algorithm (DSA) was proposed by NIST and standardized
- Since 2019, DSA is no longer recommended by NIST, and it has been mostly replaced by its elliptic curve-based equivalent algorithm (ECDSA)

Overview

20

- DSA is based on 4 algorithms:
 - Parameters generation
 - Key generation
 - Sign algorithm
 - Verify algorithm

Parameters generation

21

- Pick a cryptographic hash function H (usually SHA1)
- Pick a prime number q
- Pick a prime number p such that $p - 1$ is multiple of q
- Pick a number h in $\{2, 3, \dots, p - 2\}$ (usually $h = 2$) and $g = h^{(p-1)/q} \bmod p$
- The values (H, p, q, g) are the (publicly shared) parameters of the DSA instance

Key generation

22

- Each user generates a key as follows:
 - Pick x in $\{1, 2, \dots, q - 1\}$
 - Set $y = g^x \bmod p$
 - x is the private key, y is the public one

Signing

23

- A signature of a message m is made as follows:
 - Pick a random value k (called the *nonce*) in $\{1, \dots, q - 1\}$
 - Compute $r = (g^k \bmod p) \bmod q$
 - Compute $s = (k^{-1}(H(m) + xr)) \bmod q$
 - The pair (r, s) is the signature of m

Verifying

24

- Given a signature (r, s) and a message m , the verification is made as follows:
 - Compute $u_1 = H(m)s^{-1} \bmod q$
 - Compute $u_2 = rs^{-1} \bmod q$
 - $v = (g^{u_1}y^{u_2} \bmod p) \bmod q$
- The signature is valid if and only if $v = r$

Outline

25

- Introduction
- Digital Signatures from RSA
- The Digital Signature Algorithm
- **Nonce reuse in DSA**

Nonce reuse

26

- The main problem for DSA stems from the choice of the nonce k
- In the next slide, we show just what happens if k is used more than once
- In general, using not random (or biased) nonces is a bad idea

Nonce reuse

27

- Suppose to have two messages m_1, m_2 signed by the same user with the same nonce k
- Let's call the signatures (r_1, s_1) and (r_2, s_2)
- We can simply recover the private key x as follow:
 - $x = (s_2 H(m_1) - s_1 H(m_2))(r_2 s_1 - r_1 s_2)^{-1} \bmod q$

Digital Signatures

Gaspare FERRARO

CyberSecNatLab

Matteo ROSSI

Politecnico di Torino

