# Lecture 14: Digital Signatures

COSC362 Data and Network Security

Book 1: Chapter 13 – Book 2: Chapter 2

Spring Semester, 2021

# Motivation

- Digital signatures are one of the main benefits of public cryptosystems.
- In some countries, digital signatures are legally binding in the same way as handwritten signatures.

# Outline

# Outline

# Confidentiality and Authentication

- Message authentication codes (MACs) only allow an entity with shared secret to generate a valid tag:
  - Providing data integrity and data authentication.
- Digital signatures use public key cryptography to provide properties of a MAC and more:
  - Only the owner of the private signing key can generate a valid digital signature.
- Security service:
  - Non-repudiation
  - A judge can decide which party has formed the signature

# Comparing Physical and Digital Signatures

| Physical signatures | Digital signatures |
|---|---|
| Produced by a human | Produced by a machine |
| Same on all documents | Function of the message |
| Easy to recognize | Requiring a computer to check |

Both signature types need to be difficult to forge.

# Algorithms

- Algorithms:
  - Key generation
  - Signature generation
  - Signature verification
- The key generation algorithm outputs 2 keys:
  - A private *signing* key $K_S$
  - A public *verification* key $K_V$

# Signature Generation Algorithm

Alice wants to generate a signature on a message $M$:

- Inputs:
  - Alice's private signing key $K_S$
  - Message $M$
- Ouput:
  - Signature $s = Sig(M, K_S)$
- Only Alice, the owner of $K_S$, should be able to generate a valid signature.
- The message should be any bit string.
- The set of all signatures is usually a set of fixed size.

# Signature Verification Algorithm

Bob wants to verify a claimed signature $s$ on message $M$:

- Inputs:
  - Alice's public verification key $K_V$
  - Message $M$
  - Claimed signature $s$
- Ouput:
  - Boolean value $Ver(M, s, K_V) =$ true/false
- Anyone should be able to verify the signature.

# Properties

- Correctness:
  - If $s = Sig(M, K_S)$ then $Ver(M, s, K_V) =$ true for any matching $K_S$ and $K_V$.
- Unforgeability:
  - It is computationally infeasible for anyone without $K_S$ to construct the pair $(M, s)$ s.t. $Ver(M, s, K_V) =$ true.
- The signing algorithm *Sig may* be randomized:
  - There are many possible signatures for a single message.
- Stronger security definition:
  - 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 her choice.

# Security Goals

- Key recovery:
  - The attacker attempts to recover the private signing key $K_S$ from the public verification key $K_S$ 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.
  - It could be a meaningless message.
- Modern digital signatures are seen secure if they can resist *existential forgery under a chosen message attack*.

# Outline

Properties

RSA Signatures

Discrete Logarithm Signatures
- Elgamal Signatures
- Digital Signature Standard

# Key Generation

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

- Public verification key: $n, e$ where $n = pq$ for large primes $p, q$.
- Private signing key: $p, q, d$ s.t. $ed \mod \phi(n) = 1$.

A hash function $h$ is also required, as a fixed public parameter. It can be a standard hash function (e.g. SHA-256).

# Signature Generation and Verification

Signature generation:

- Inputs are message $M$, modulus $n$ and private exponent $d$.
- Compute $s = h(M)^d \mod n$.
- Output $(M, s)$ as the signature.

Signature verification:

- Inputs are claimed signature $(M, s)$, modulus $n$ and public exponent $e$.
- Compute $h' = h(M)$.
- Check if $s^e \mod n = h'$? It so, then output true; otherwise, output false:
  - Check Lecture 12 for correctness.

# Outline

# Discrete Logarithm Signatures

- Security relying on difficulty of discrete logarithm problem.
- 3 versions:
  1. Original Elgamal signatures in $\mathbb{Z}_p^*$ (1985).
  2. Digital signature algorithm (DSA) standardised by NIST:
     - an optimized version of Elgamal signatures
  3. DSA based on elliptic curve groups, known as ECDSA.

# Elgamal Elements in $\mathbb{Z}_p^*$

- $p$ is a large prime.
- $g$ is generator of $\mathbb{Z}_p^*$.
- $x$ is the private signing key s.t. $0 < x < p - 1$.
- $p, g, y$ form the public verification key where $y = g^x \mod p$.
- Alice wants to sign a value $M$ where $0 < M < p - 1$.

# Elgamal Operations in $\mathbb{Z}_p^*$

Signature generation:

1. Alice selects a random $k$ s.t. $\gcd(k, p-1) = 1$ and computes
$$r = g^k \mod p$$
2. Alice solves $M = xr + ks \mod (p-1)$ for $s$ by computing
$$s = k^{-1}(M - xr) \mod (p-1)$$
3. Alice outputs the tuple $(M, r, s)$.

Signature verification:

- Bob checks if $g^M \equiv y^r r^s \mod p \ (= (g^x)^r (g^k)^s)$.

# Digital Signature Algorithm (DSA)

- First published by NIST in 1994.
- Standard: FIPS PUB 186-4 (2013).
- Based on Elgamal signatures.
- Simpler calculations and shorter signatures:
  - Calculations done in a *subgroup* of $\mathbb{Z}_p^*$ or an elliptic curve group.
- Used with SHA family of hash functions.
- Preventing attacks that Elgamal signatures may be vulnerable to.

# Idea

- Prime $p$ chosen s.t. $p-1$ has a prime divisor $q$ of much smaller size (224 or 256 bits).
- Generator $g$ used in Elgamal signatures replaced by $g = h^{\frac{p-1}{q}} \mod p$ where $h$ is a generator:
  - $g$ has order $q$ since $g^q \mod p = 1$:
    - $g^q \mod p = (h^{\frac{p-1}{q}})^q \mod p = h^{p-1} \mod p = 1$ (Fermat's theorem).
  - All exponents can be thus reduced modulo $q$ before exponentiation.

# Comparison

Differences with Elgamal signatures:

- Message is hashed using standard SHA hash algorithm.
- $g$ chosen to be of order $q$, which is much smaller than $p$.
- Verification equation becomes[1]:

$$(g^{H(M)})^{s^{-1}}(y^{-r})^{s^{-1}} \equiv r \pmod{p}$$

Both sides of the equation are then reduced modulo $q$.

[1] from $g^{H(M)} \equiv y^r r^s \mod p$

# Parameters

- $p$ is a prime modulus, of $L$ bits.
- $q$ is a prime divisor of $p - 1$, of $N$ bits.
- Valid combinations:
  - $L = 1024$, $N = 160$
  - $L = 2048$, $N = 224$
  - $L = 2048$, $N = 256$
  - $L = 3072$, $N = 256$
- $g = h^{\frac{p-1}{q}} \mod p$ is the generator where $1 < h < p - 1$.
- $H$ is the hash function from SHA family variant s.t. the output is an $N$-bit digest.

# Key and Signature Generations

Key generation:

- Choose a random integer $x$ s.t. $0 < x < q$.
- Compute $y = g^x \mod p$.
- Set $x$ as the secret key and $y$ as the public key.

Signature generation:

- Let $M$ be a message.
- Choose $k$ at random s.t. $0 < k < q$.
- Compute $r = (g^k \mod p) \mod q$.
- Compute $s = k^{-1}(H(M) - xr) \mod q$.
- Set $(M, r, s)$ as the signature.

# Signature Verification

Signature verification:

- $(M, r, s)$ is the claimed signature.
- Check if $0 < r < q$ and $0 < s < q$.
- Compute $w = s^{-1} \mod q$.
- Compute $u_1 = H(M)w \mod q$.
- Compute $u_2 = rw \mod q$.
- Check if $(g^{u_1} y^{-u_2} \mod p) \mod q = r$.

# Comparison

Differences with Elgamal signatures:

- Verification equation is the same, except that all exponents and final result are reduced modulo $q$.
- Signature generation mainly requires one exponentiation with a short exponent (224 or 256 bits).
- Signature verification requires 2 short exponentiations.
- Signature size is only $2N$ bits:
    - 448 bits when $N = 224$
    - 512 bits when $N = 256$

# Parameter Values

Key lengths defined in the 2013 standard version:

| Version no. | $\|p\|$ | $\|q\|$ | Hash function |
|---|---|---|---|
| 1 | 1024 bits | 160 bits | SHA-1 |
| 2 | 2048 bits | 224 bits | SHA-224 |
| 3 | 2048 bits | 256 bits | SHA-256 |
| 4 | 3072 bits | 256 bits | SHA-256 |

NIST special publication SP 800-57 does NOT approve Version no. 1.

# Elliptic Curve DSA (ECDSA)

- Standard: FIPS PUB 186-4 (2013).
- Parameters chosen from NIST approved curves.
- Signature generation and verification are the same, except that:
  - $q$ becomes the order of the elliptic curve group.
  - Multiplication modulo $p$ is replaced by the elliptic curve group operation.
  - After operations on group elements, only the $x$ coordinate is kept (from the pair $(x, y)$).

# Comparison

ECDSA versus DSA:

- ECDSA signatures are generally not shorter than DSA signatures for the same security level.
- ECDSA signature size varies with the underlying curve:
  - Between 326 bits and 1142 bits from approved curves.
- ECDSA public keys are shorter than DSA public keys.