ELGAMAL DIGITAL SIGNATURE

Kacper Piotr Wojakowski 293064

Faculty of Electronics and Information Technology

ECRYP

TABLE OF CONTENTS

1. INTRODUCTION

1.1 DIGITAL SIGNATURE

A digital signature is a technique designed to provide a digital counterpart to handwritten signatures. It is a number dependent on the content of the signed message/document and a secret known only to the signer. An important feature of signatures is verifiability – it should be possible (and easy) to determine whether someone signed a document, and it should be able to be performed by an unbiased third party, without the need for the signers secret private key. The basic principle of the digital signature is presented on Figure 1.

Digital signatures have many applications in information security – authentication, data integrity, non-repudiation, etc. One of the most significant applications is certification of public keys in large networks. The concept of digital signatures was recognized years before any practical use was available. The first discovered method used for digital signatures was the RSA signature scheme, which is still used today, however through subsequent research many other alternatives were discovered, such as the ElGamal Digital Signature.

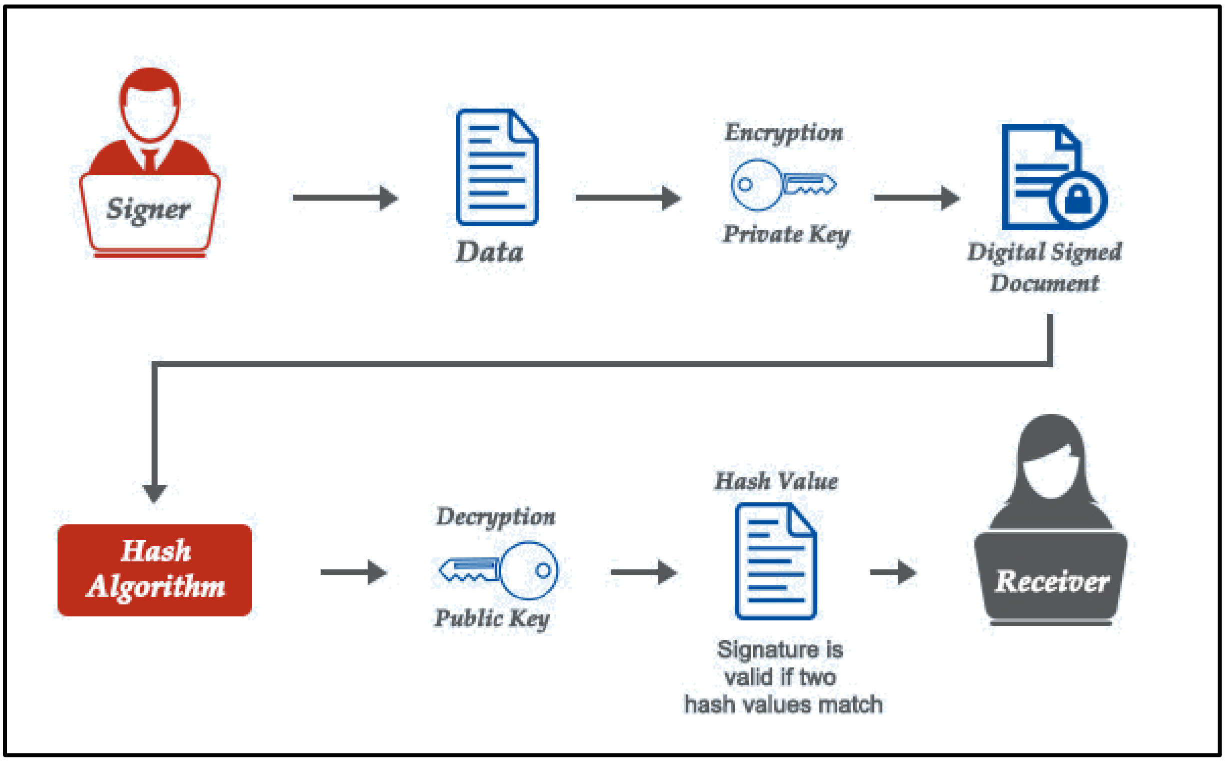

Figure 1: Basic principle of the digital signature.
Source: https://comodosslstore.com/blog/what-is-digital-signature-how-does-it-work.html

1.2 ELGAMAL DIGITAL SIGNATURE - OVERVIEW

The ElGamal digital signature scheme was described by Taher Elgamal in 1985. It is not to be confused with the ElGamal Encryption, also invented by T. Elgamal. The digital signature is based on the difficulty of computing discrete logarithms. In itself it is rarely used in practice, however a variant called the Digital Signature Algorithm, developed by the National Security Agency (NSA) is very widely used.

The signature uses a pair of keys – a public key and a private key. The private key is used to generate a signature by the signer, while the public key can be used to verify this signature. The signature provides message authentication, integrity and non-repudiation.

1.3 ELGAMAL DIGITAL SIGNATURE – OPERATION

The scheme consists of four operations: key generation, key distribution, signing and signature verification.

1. KEY GENERATION

This operation has two phases: choosing algorithm parameters, which can be shared between users of the system, and generating a key pair for one user.

A. PARAMETER GENERATION

- Choose a key length N
- Choose a N-bit prime number p
- Choose cryptographic hash function H with output length of L bits. If $L > N$, only the leftmost N bits of the hash output are used.
- Choose a generator $g < p$ of the multiplicative group Z_p^*

B. PER USER KEYS

Given the set of parameters, second phase computes the key pair for a single user.

- Choose an integer x randomly from $\{1 \ldots p-2\}$
- Compute $y := g^x \bmod p$

x is the private key and y is the public key.

2. KEY DISTRIBUTION

The signer should send the public key y to the receiver via a reliable (but not necessarily secret) mechanism. The signer should keep the private key x secret.

3. SIGNING

A message m is signed as follows:

- Choose an integer k randomly from $\{2 \ldots p-2\}$ with k relatively prime to $p-1$
- Compute $r := g^k \bmod p$
- Compute $s := (H(m) - xr) k^{-1} \bmod (p - 1)$
- In the unlikely event, that $s = 0$, start again with a different random k

The signature is the pair (r, s)

4. VERIFICATION

One can verify that a signature (r, s) is a valid signature for a message m as follows:

- Verify that $0 < r < p$ and $0 < s < p - 1$
- The signature is valid if and only if $g^{H(m)} \equiv y^r r^s \pmod{p}$

2. IMPLEMENTATION

The program was written in Python 3.8. The module *elgamal.py* comprises of four functions, each corresponding to an operation described in 1.3.

```
def generate_system(key_length, hash_function):
```

This function is used to generate an ElGamal system (operation 1A). It takes a bit length of the key and the hash function to use, and returns those parameters together with generated prime *p* and generator *g*, as a dictionary. The prime generated is made sure to be a safe prime – first, a prime of bit length *N-1* is generated, and then a number of form *2p+1* is checked for being prime – if it is, it becomes *p*.

```
def generate_keys(system):
```

This function generates a pair of keys *(x, y)* for a given ElGamal system. It is then up to the user, to use the private key to sign a document using the *sign()* function and keep it secret, and to share the public key with the message receiver.

```
def sign(system, message, private_key):
```

This function signs a message (string) with a given ElGamal system and private key. The signature pair *(r, s)* is then returned. The message is hashed using a previously given hash function, and *s* is made sure to be different from 0.

```
def verify(system, message, signature, public_key):
```

Verifies if the given signature is valid, given the system, public key and the message. It first checks the correctness of the signature values, and then verifies the validity of the signature. The validity is returned as a bool value.

3. TESTING

The *elgamal.py* module was tested for various messages of different lengths and different key lengths. The tests are in *test_elgamal.py*. The hash function used for testing was *sha256* from *shalib*. To test, various messages were signed using a recently generated ElGamal Digital Signature system and key pair, and then the validity of that signature was determined. Example output of such test is as follows:

```
Message:
dorime interimo adapare dorime ameno ameno latire latiremo dorime
Generated system:
{'N': 32, 'p': 2945482223, 'H': <sha256 HASH object @ 0x01123F90>, 'g': 2136812729}
Generated key pair (x, y):
(614263586, 2255379149)
Generated signature pair (r, s):
(2526270209, 1782570173)
Is signature valid?
True
```

4. REFERENCES

"Handbook of Applied Cryptography" – A. J. Menezes, P. C. van Oorschot, S. A. Vanstone

https://en.wikipedia.org/wiki/ElGamal_signature_scheme

https://comodosslstore.com/blog/what-is-digital-signature-how-does-it-work.html

https://www.commonlounge.com/discussion/35a1c2baa00b447f9275e8f71b02ef29

5. SOURCE CODE

5.1 ELGAMAL.PY

```python
from Crypto.Util import number as num
from random import randint
from math import gcd

""" Module for generating ElGamal Digital Signature Scheme systems, keys,
    Signing documents, and veryfing signatures.
"""


def generate_system(key_length, hash_function):
    """ Generates an ElGamal system

        Parameters:
        key_length (int): bits of length of prime number p
        hash_function (HASH): a hash function (from hashlib) to hash the message

        Returns:
        Dictionary: {
            "N": bit length of prime
            "p": generated prime
            "H": hash function
            "g": chosen generator of Z*p
        }
    """
    # Generating safe prime
    p = 4
    while not num.isPrime(p):
        pp = num.getPrime(key_length-1)
        p = pp*2+1

    # Choosing a generator
    g = randint(2, p-1)
    while (p-1)%g == 1:
        g = randint(2, p-1)

    system = {
        "N": key_length,
        "p": p,
        "H": hash_function,
        "g": g
    }
    return system
```

```
def generate_keys(system):
    """ Generates a pair of keys for an ElGamal DS system

        Parameters:
        system (dictionary):    an ElGamal system (generated by generate_system)

        Returns:
        int, int:   a pair of keys - private key, public key (x, y)
    """
    x = randint(1, system["p"]-2)
    y = pow(system["g"], x, system["p"])

    return x, y

def sign(system, message, private_key):
    """ Signs a message (document) using an ElGamal DS system and private key

        Parameters:
        system (dictionary):    an ElGamal system (generated by generate_system)
        message (str):          a message to be signed
        private_key (int):      private key to be used to sign the message

        Returns:
        int, int:   a signature pair - (r, s)
    """
    # Hashing the message
    H = system["H"].copy()
    H.update(message.encode())
    hash = int(H.hexdigest(), 16)

    s = 0
    while s == 0:
        # Find relatively prime to p-1 k
        k = randint(2,system["p"])
        while gcd(k, system["p"]-1) != 1:
            k = randint(2,system["p"])

        # Calculate the signature pair
        r = pow(system["g"], k, system["p"])
        s = ((hash - private_key*r) * num.inverse(k, system["p"]-1))%(system["p"]-1)

    return r,s
```

```python
def verify(system, message, signature, public_key):
    """ Verifies an ElGamal signature

        Parameters:
        system (dictionary):    an ElGamal system (generated by generate_system)
        message (str):          a message that was signed
        signature (int, int):   a signature pair (r, s)
        public_key (int):       a public key from the pair used to sign the document

        returns:
        bool:   validity of the signature - True if valid, False otherwise
    """
    # Check correctnes of signature
    if signature[0] <= 0 or signature[1] <= 0 or signature[0] >= system["p"] or signature[1] >
= (system["p"]-1):
        return False

    # Hashing the message
    H = system["H"].copy()
    H.update(message.encode())
    hash = int(H.hexdigest(), 16)

    # Check validity of signature
    l = pow(system["g"], hash, system["p"])
    r = pow(public_key, signature[0], system["p"]) % system["p"] * pow(signature[0], signature
[1], system["p"]) % system["p"]

    return l == r
```

5.2 TEST_ELGAMAL.PY

```python
import elgamal
from hashlib import sha256

""" Testing the elgamal module """

if __name__ == "__main__":
    # Documents to sign
    msgs = ["ECRYP", "kwojakow", "ElGamal Digital Signature", "lorem ipsum", "dorime interimo adapare dorime ameno ameno latire latiremo dorime"]
    # Hash function
    hfun = sha256()
    # Bit Length
    N = 32

    # Testing the signatures for the documents
    for msg in msgs:
        print("Message:")
        print(msg)

        elgsys = elgamal.generate_system(N, hfun)
        print("Generated system:")
        print(elgsys)

        keys = elgamal.generate_keys(elgsys)
        print("Generated key pair (x, y):")
        print(keys)

        sig = elgamal.sign(elgsys, msg, keys[0])
        print("Generated signature pair (r, s):")
        print(sig)

        is_valid = elgamal.verify(elgsys, msg, sig, keys[1])
        print("Is signature valid?")
        print(is_valid)
```