Додаток А Код програми

```
package main

import (
   "crypto/sha1"
   "crypto/sha256"
   "crypto/sha512"
   "fmt"
   "strconv"
   "time"

   "golang.org/x/crypto/blake2b"
   "golang.org/x/crypto/blake2s"
   "golang.org/x/crypto/scrypt"
   "golang.org/x/crypto/sha3"
)

const (
   SHA1        int = 0
   SHA256      int = 1
   SHA384      int = 2
   SHA512      int = 3
   SHA3_256    int = 4
   SHA3_384    int = 5
   SHA3_512    int = 6
   BLAKE2s_256 int = 7
   BLAKE2b_256 int = 8
   BLAKE2b_384 int = 9
   BLAKE2b_512 int = 10
   SCRIPT      int = 11
)

func main() {

       start1 := time.Now()
       //operations
       finish1 := time.Now()
       elapsed1 := finish1.Sub(start1)
```

```
}

// Hash functions

func sha1Hash(input []byte) [20]byte {
   hashValue := sha1.Sum(input)
   return hashValue
}
func sha256Hash(input []byte) [32]byte {
   hashValue := sha256.Sum256(input)
   return hashValue
}
func sha384Hash(input []byte) [48]byte {
   hashValue := sha512.Sum384(input)
   return hashValue
}
func sha512Hash(input []byte) [64]byte {
   hashValue := sha512.Sum512(input)
   return hashValue
}
func sha3_256Hash(input []byte) [32]byte {
   hashValue := sha3.Sum256(input)
   return hashValue
}
func sha3_384Hash(input []byte) [48]byte {
   hashValue := sha3.Sum384(input)
   return hashValue
}
func sha3_512Hash(input []byte) [64]byte {
   hashValue := sha3.Sum512(input)
   return hashValue
}
func blake2s256Hash(input []byte) [32]byte {
   hashValue := blake2s.Sum256(input)
   return hashValue
}
func blake2b256Hash(input []byte) [32]byte {
   hashValue := blake2b.Sum256(input)
   return hashValue
```

```
}
func blake2b384Hash(input []byte) [48]byte {
   hashValue := blake2b.Sum384(input)
   return hashValue
}
func blake2b512Hash(input []byte) [64]byte {
   hashValue := blake2b.Sum512(input)
   return hashValue
}
func scrypt512Hash(input []byte, difParam int) []byte {
   hashValue, _ := scrypt.Key(input, []byte(""), difParam, 8, 1, 64)
//difParam:  16384 - 1048576 - optimal
   return hashValue
}

// Merkle Node getting (concat + hash)

func merkleNode20(leaf1 [20]byte, leaf2 [20]byte, hashFunc int) [20]byte
{
   hash1 := leaf1[:]
   hash2 := leaf2[:]
   hash3 := append(hash1, hash2...)
   switch hashFunc {
   case SHA1:
       hash4 := sha1Hash(hash3)
       return hash4
   }
   return leaf1
}
func merkleNode32(leaf1 [32]byte, leaf2 [32]byte, hashFunc int) [32]byte
{
   hash1 := leaf1[:]
   hash2 := leaf2[:]
   hash3 := append(hash1, hash2...)
   switch hashFunc {
   case SHA256:
       hash4 := sha256Hash(hash3)
       return hash4
   case SHA3_256:
```

```
            hash4 := sha3_256Hash(hash3)
            return hash4
        case BLAKE2s_256:
            hash4 := blake2s256Hash(hash3)
            return hash4
        case BLAKE2b_256:
            hash4 := blake2b256Hash(hash3)
            return hash4
        }
        return leaf1
    }
    func merkleNode48(leaf1 [48]byte, leaf2 [48]byte, hashFunc int) [48]byte
{
        hash1 := leaf1[:]
        hash2 := leaf2[:]
        hash3 := append(hash1, hash2...)
        switch hashFunc {
        case SHA384:
            hash4 := sha384Hash(hash3)
            return hash4
        case SHA3_384:
            hash4 := sha3_384Hash(hash3)
            return hash4
        case BLAKE2b_384:
            hash4 := blake2b384Hash(hash3)
            return hash4
        }
        return leaf1
    }
    func merkleNode64(leaf1 [64]byte, leaf2 [64]byte, hashFunc int) [64]byte
{
        hash1 := leaf1[:]
        hash2 := leaf2[:]
        hash3 := append(hash1, hash2...)
        switch hashFunc {
        case SHA512:
            hash4 := sha512Hash(hash3)
            return hash4
        case SHA3_512:
```

```
            hash4 := sha3_512Hash(hash3)
            return hash4
        case BLAKE2b_512:
            hash4 := blake2b512Hash(hash3)
            return hash4
        }
        return leaf1
    }
    func merkleNodeScript(leaf1 []byte, leaf2 []byte, difParam int) []byte
{
        hash1 := leaf1[:]
        hash2 := leaf2[:]
        hash3 := append(hash1, hash2...)
        hash4 := scrypt512Hash(hash3, difParam)
        return hash4
    }

    // Merkle root getting

    func merkleHash20(data map[int][]byte) [20]byte {
        layer0 := make(map[int][20]byte)
        for i := 0; i < 8; i++ {
            layer0[i] = sha1Hash(data[i])
        }

        layer1 := make(map[int][20]byte)
        layer1[0] = merkleNode20(layer0[0], layer0[1], SHA1)
        layer1[1] = merkleNode20(layer0[2], layer0[3], SHA1)
        layer1[2] = merkleNode20(layer0[4], layer0[5], SHA1)
        layer1[3] = merkleNode20(layer0[6], layer0[7], SHA1)

        layer2 := make(map[int][20]byte)
        layer2[0] = merkleNode20(layer1[0], layer1[1], SHA1)
        layer2[1] = merkleNode20(layer1[2], layer1[3], SHA1)

        hash := merkleNode20(layer2[0], layer2[1], SHA1)
        return hash
    }
    func merkleHash32(data map[int][]byte, hashFunc int) [32]byte {
```

```
layer0 := make(map[int][32]byte)
switch hashFunc {
case SHA256:
    for i := 0; i < 8; i++ {
        layer0[i] = sha256Hash(data[i])
    }
    layer1 := make(map[int][32]byte)
    layer1[0] = merkleNode32(layer0[0], layer0[1], SHA256)
    layer1[1] = merkleNode32(layer0[2], layer0[3], SHA256)
    layer1[2] = merkleNode32(layer0[4], layer0[5], SHA256)
    layer1[3] = merkleNode32(layer0[6], layer0[7], SHA256)

    layer2 := make(map[int][32]byte)
    layer2[0] = merkleNode32(layer1[0], layer1[1], SHA256)
    layer2[1] = merkleNode32(layer1[2], layer1[3], SHA256)

    hash := merkleNode32(layer2[0], layer2[1], SHA256)
    return hash

case SHA3_256:
    for i := 0; i < 8; i++ {
        layer0[i] = sha3_256Hash(data[i])
    }
    layer1 := make(map[int][32]byte)
    layer1[0] = merkleNode32(layer0[0], layer0[1], SHA3_256)
    layer1[1] = merkleNode32(layer0[2], layer0[3], SHA3_256)
    layer1[2] = merkleNode32(layer0[4], layer0[5], SHA3_256)
    layer1[3] = merkleNode32(layer0[6], layer0[7], SHA3_256)

    layer2 := make(map[int][32]byte)
    layer2[0] = merkleNode32(layer1[0], layer1[1], SHA3_256)
    layer2[1] = merkleNode32(layer1[2], layer1[3], SHA3_256)

    hash := merkleNode32(layer2[0], layer2[1], SHA3_256)
    return hash

case BLAKE2s_256:
    for i := 0; i < 8; i++ {
        layer0[i] = blake2s256Hash(data[i])
```

```
        }
        layer1 := make(map[int][32]byte)
        layer1[0] = merkleNode32(layer0[0], layer0[1], BLAKE2s_256)
        layer1[1] = merkleNode32(layer0[2], layer0[3], BLAKE2s_256)
        layer1[2] = merkleNode32(layer0[4], layer0[5], BLAKE2s_256)
        layer1[3] = merkleNode32(layer0[6], layer0[7], BLAKE2s_256)

        layer2 := make(map[int][32]byte)
        layer2[0] = merkleNode32(layer1[0], layer1[1], BLAKE2s_256)
        layer2[1] = merkleNode32(layer1[2], layer1[3], BLAKE2s_256)

        hash := merkleNode32(layer2[0], layer2[1], BLAKE2s_256)
        return hash

    case BLAKE2b_256:
        for i := 0; i < 8; i++ {
            layer0[i] = blake2b256Hash(data[i])
        }
        layer1 := make(map[int][32]byte)
        layer1[0] = merkleNode32(layer0[0], layer0[1], BLAKE2b_256)
        layer1[1] = merkleNode32(layer0[2], layer0[3], BLAKE2b_256)
        layer1[2] = merkleNode32(layer0[4], layer0[5], BLAKE2b_256)
        layer1[3] = merkleNode32(layer0[6], layer0[7], BLAKE2b_256)

        layer2 := make(map[int][32]byte)
        layer2[0] = merkleNode32(layer1[0], layer1[1], BLAKE2b_256)
        layer2[1] = merkleNode32(layer1[2], layer1[3], BLAKE2b_256)

        hash := merkleNode32(layer2[0], layer2[1], BLAKE2b_256)
        return hash
    }
    return layer0[0]
}
func merkleHash48(data map[int][]byte, hashFunc int) [48]byte {
    layer0 := make(map[int][48]byte)
    switch hashFunc {
    case SHA384:
        for i := 0; i < 8; i++ {
            layer0[i] = sha384Hash(data[i])
```

```
    }
    layer1 := make(map[int][48]byte)
    layer1[0] = merkleNode48(layer0[0], layer0[1], SHA384)
    layer1[1] = merkleNode48(layer0[2], layer0[3], SHA384)
    layer1[2] = merkleNode48(layer0[4], layer0[5], SHA384)
    layer1[3] = merkleNode48(layer0[6], layer0[7], SHA384)

    layer2 := make(map[int][48]byte)
    layer2[0] = merkleNode48(layer1[0], layer1[1], SHA384)
    layer2[1] = merkleNode48(layer1[2], layer1[3], SHA384)

    hash := merkleNode48(layer2[0], layer2[1], SHA384)
    return hash

case SHA3_384:
    for i := 0; i < 8; i++ {
        layer0[i] = sha3_384Hash(data[i])
    }
    layer1 := make(map[int][48]byte)
    layer1[0] = merkleNode48(layer0[0], layer0[1], SHA3_384)
    layer1[1] = merkleNode48(layer0[2], layer0[3], SHA3_384)
    layer1[2] = merkleNode48(layer0[4], layer0[5], SHA3_384)
    layer1[3] = merkleNode48(layer0[6], layer0[7], SHA3_384)

    layer2 := make(map[int][48]byte)
    layer2[0] = merkleNode48(layer1[0], layer1[1], SHA3_384)
    layer2[1] = merkleNode48(layer1[2], layer1[3], SHA3_384)

    hash := merkleNode48(layer2[0], layer2[1], SHA3_384)
    return hash

case BLAKE2b_384:
    for i := 0; i < 8; i++ {
        layer0[i] = blake2b384Hash(data[i])
    }
    layer1 := make(map[int][48]byte)
    layer1[0] = merkleNode48(layer0[0], layer0[1], BLAKE2b_384)
    layer1[1] = merkleNode48(layer0[2], layer0[3], BLAKE2b_384)
    layer1[2] = merkleNode48(layer0[4], layer0[5], BLAKE2b_384)
```

```
        layer1[3] = merkleNode48(layer0[6], layer0[7], BLAKE2b_384)

        layer2 := make(map[int][48]byte)
        layer2[0] = merkleNode48(layer1[0], layer1[1], BLAKE2b_384)
        layer2[1] = merkleNode48(layer1[2], layer1[3], BLAKE2b_384)

        hash := merkleNode48(layer2[0], layer2[1], BLAKE2b_384)
        return hash
    }
    return layer0[0]
}
func merkleHash64(data map[int][]byte, hashFunc int) [64]byte {
    layer0 := make(map[int][64]byte)
    switch hashFunc {
    case SHA512:
        for i := 0; i < 8; i++ {
            layer0[i] = sha512Hash(data[i])
        }
        layer1 := make(map[int][64]byte)
        layer1[0] = merkleNode64(layer0[0], layer0[1], SHA512)
        layer1[1] = merkleNode64(layer0[2], layer0[3], SHA512)
        layer1[2] = merkleNode64(layer0[4], layer0[5], SHA512)
        layer1[3] = merkleNode64(layer0[6], layer0[7], SHA512)

        layer2 := make(map[int][64]byte)
        layer2[0] = merkleNode64(layer1[0], layer1[1], SHA512)
        layer2[1] = merkleNode64(layer1[2], layer1[3], SHA512)

        hash := merkleNode64(layer2[0], layer2[1], SHA512)
        return hash

    case SHA3_512:
        for i := 0; i < 8; i++ {
            layer0[i] = sha3_512Hash(data[i])
        }
        layer1 := make(map[int][64]byte)
        layer1[0] = merkleNode64(layer0[0], layer0[1], SHA3_512)
        layer1[1] = merkleNode64(layer0[2], layer0[3], SHA3_512)
        layer1[2] = merkleNode64(layer0[4], layer0[5], SHA3_512)
```

```
        layer1[3] = merkleNode64(layer0[6], layer0[7], SHA3_512)

        layer2 := make(map[int][64]byte)
        layer2[0] = merkleNode64(layer1[0], layer1[1], SHA3_512)
        layer2[1] = merkleNode64(layer1[2], layer1[3], SHA3_512)

        hash := merkleNode64(layer2[0], layer2[1], SHA3_512)
        return hash

    case BLAKE2b_512:
        for i := 0; i < 8; i++ {
            layer0[i] = blake2b512Hash(data[i])
        }
        layer1 := make(map[int][64]byte)
        layer1[0] = merkleNode64(layer0[0], layer0[1], BLAKE2b_512)
        layer1[1] = merkleNode64(layer0[2], layer0[3], BLAKE2b_512)
        layer1[2] = merkleNode64(layer0[4], layer0[5], BLAKE2b_512)
        layer1[3] = merkleNode64(layer0[6], layer0[7], BLAKE2b_512)

        layer2 := make(map[int][64]byte)
        layer2[0] = merkleNode64(layer1[0], layer1[1], BLAKE2b_512)
        layer2[1] = merkleNode64(layer1[2], layer1[3], BLAKE2b_512)

        hash := merkleNode64(layer2[0], layer2[1], BLAKE2b_512)
        return hash
    }
    return layer0[0]
}
func merkleHashScript(data map[int][]byte, difParam int) []byte {
    layer0 := make(map[int][]byte)
    for i := 0; i < 8; i++ {
        layer0[i] = scrypt512Hash(data[i], difParam)
    }

    layer1 := make(map[int][]byte)
    layer1[0] = merkleNodeScript(layer0[0], layer0[1], difParam)
    layer1[1] = merkleNodeScript(layer0[2], layer0[3], difParam)
    layer1[2] = merkleNodeScript(layer0[4], layer0[5], difParam)
    layer1[3] = merkleNodeScript(layer0[6], layer0[7], difParam)
```

```
        layer2 := make(map[int][]byte)
        layer2[0] = merkleNodeScript(layer1[0], layer1[1], difParam)
        layer2[1] = merkleNodeScript(layer1[2], layer1[3], difParam)

        hash := merkleNodeScript(layer2[0], layer2[1], difParam)
        return hash
    }

    // Merkle path getting
    func merkleAuthHash20(data map[int][]byte, index int) ([20]byte,
[20]byte, [20]byte) {
        layer0 := make(map[int][20]byte)
        for i := 0; i < 8; i++ {
            layer0[i] = sha1Hash(data[i])
        }

        layer1 := make(map[int][20]byte)
        layer1[0] = merkleNode20(layer0[0], layer0[1], SHA1)
        layer1[1] = merkleNode20(layer0[2], layer0[3], SHA1)
        layer1[2] = merkleNode20(layer0[4], layer0[5], SHA1)
        layer1[3] = merkleNode20(layer0[6], layer0[7], SHA1)

        layer2 := make(map[int][20]byte)
        layer2[0] = merkleNode20(layer1[0], layer1[1], SHA1)
        layer2[1] = merkleNode20(layer1[2], layer1[3], SHA1)

        if index == 0 {
            return layer0[1], layer1[1], layer2[1]
        } else if index == 1 {
            return layer0[0], layer1[1], layer2[1]
        } else if index == 2 {
            return layer0[3], layer1[0], layer2[1]
        } else if index == 3 {
            return layer0[2], layer1[0], layer2[1]
        } else if index == 4 {
            return layer0[5], layer1[3], layer2[0]
        } else if index == 5 {
            return layer0[4], layer1[3], layer2[0]
```

```
    } else if index == 6 {
        return layer0[7], layer1[2], layer2[0]
    } else if index == 7 {
        return layer0[6], layer1[2], layer2[0]
    }
    return layer0[0], layer1[0], layer2[0]
}
```