BTI4202 - Exercise Sheet 1

Jan Fuhrer

April 19, 2023

Contents

1 Diffie-Hellman Key Exchange

Given Information:

- Group: Z_{11}^*
- Generator: $g = 2$
- Choose Alice's secret $a \leftarrow 7$
- Choose Bob's secret $b \leftarrow 13$

Calculations:

- Public value Alice: $A := g^a = 2^7 \pmod{11} = 7$
- Public value Bob: $B := g^b = 2^13 \pmod{11} = 8$
- Shared Secret Alice: $k := B^a = 8^7 \pmod{11} = 2$
- Shared Secret Bob: $k := A^b = 7^{13} \pmod{11} = 2$

2 Breaking the Diffie-Hellman Key Exchange

Given Information:

- Z_{11}^*
- $g = 6$
- $A = 7$
- $B = 5$

Brute Force:

- $a = 3$
- $b = 6$
- $k = 4$

Secure value for p: $||p|| \approx 2048 bits$

2.1 Implementation in Go

```go
package main

import (
  "fmt"
  "math"
)

func breakDH(prime int, generator int, publicAlice int, publicBob
    int) (secretAlice int, secretBob int) {
  // start with 1
  alice := 1
  bob := 1

  // alice
  for {
    // calculate secret key for Alice
    tmpKey := math.Mod(math.Pow(float64(generator), float64(alice))
    , float64(prime))

    if int(tmpKey) == publicAlice {
      break
    }
    alice++
  }

  // bob
  for {
    // calculate secret key for Bob
    tmpKey := math.Mod(math.Pow(float64(generator), float64(bob)),
    float64(prime))
    if int(tmpKey) == publicBob {
      break
    }
    bob++
  }

  return alice, bob
}

func main() {
  fmt.Printf("Calculating secret keys for Alice and Bob...\n")

  var prime = 11
  var generator = 6
  var publicAlice = 7
  var publicBob = 5

  secretAlice, secretBob := breakDH(prime, generator, publicAlice,
    publicBob)

  var sharedkey = math.Mod(math.Pow(float64(publicAlice), float64(
    secretBob)), float64(prime))

  fmt.Printf("secret key for Alice: %d\n", secretAlice)
  fmt.Printf("secret key for Bob: %d\n", secretBob)
```

```go
	fmt.Printf("shared key: %d\n", int(sharedkey))
}
```

3 Primitive Roots Modulo Prime

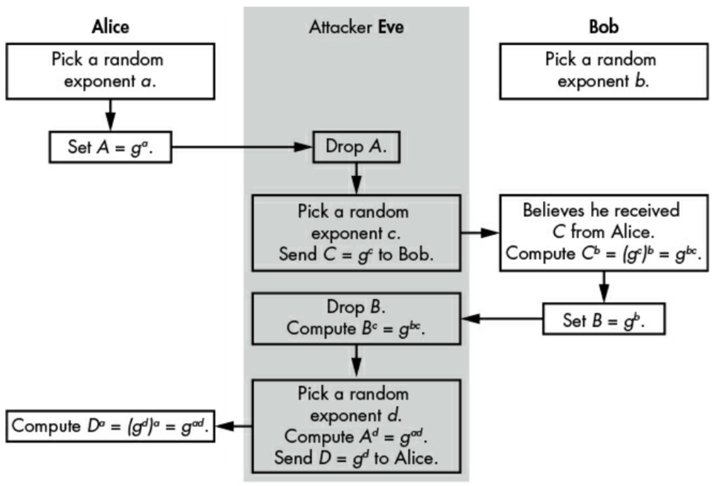
Primitive roots modulo 11: $2, 6, 7, 8$

						y					
		1	2	3	4	5	6	7	8	9	10
	1	1	1	1	1	1	1	1	1	1	1
	2	2	4	8	5	10	9	7	3	6	1
	3	3	9	5	4	1	3	9	5	4	1
	4	4	5	9	3	1	4	5	9	3	1
x	5	5	3	4	9	1	5	3	4	9	1
	6	6	3	7	9	10	5	8	4	2	1
	7	7	5	2	3	10	4	6	9	8	1
	8	8	9	6	4	10	3	2	5	7	1
	9	9	4	3	5	1	9	4	3	5	1
	10	10	1	10	1	10	1	10	1	10	1

- rows with a repetition $1, 3, 4, 5, 9, 10$
- primitive roots are without a repetition: $2, 6, 7, 8$
- these values are the generators of the group

4 Man-in-the-Middle Attack

A man-in-the-middle attacker can intersect the public key of bob and send his own public key to Alice. Alice will then encrypt the message with the public key of the attacker and the attacker can decrypt the message with his private key. The attacker can then send the message to bob.


Alice
Attacker Eve
Bob
Pick a random exponent a.
Pick a random exponent b.
Set A = g^a.
Drop A.
Pick a random exponent c. Send C = g^c to Bob.
Believes he received C from Alice. Compute C^b = (g^c)^b = g^bc.
Set B = g^b.
Drop B. Compute B^c = g^bc.
Pick a random exponent d. Compute A^d = g^ad. Send D = g^d to Alice.
Compute D^a = (g^d)^a = g^ad.

Alice — M-M — Bob

$m \in \mathcal{M}$ — $(pk_M, sk_M) \leftarrow \text{keyGen}()$ — $(pk_B, sk_B) \leftarrow \text{keyGen}()$

$\xleftarrow{pk_B}$ (Bob to M-M)

$\xleftarrow{pk_M}$ (M-M to Alice)

$c \leftarrow enc_{pk_M}(m)$

$\xrightarrow{c}$

$m := dec_{sk_M}(c)$

$c' \leftarrow enc_{pk_B}(m)$

Could even change m !

$\xrightarrow{c'}$

$m := dec_{sk_B}(c')$

5 Hybrid Encryption Scheme

Approach 1 with DH:

1. Alice and Bob create a shared secret key k using the Diffie-Hellman key exchange. The public keys are signed with the private key of the sender.
2. Based on the shared secret key k, Alice and Bob create a symmetric key K using a key derivation function.
3. Alice and Bob can now use the symmetric key K to encrypt and decrypt messages.

Approach 2: An alternative to the DH-Key exchange

1. Alice creates an asymmetric key-pair and send the public-key to Bob.
2. Bob creates a symmetric key (session-key) and encrypt the message with it.
3. Bob encrypts the session-key with the public-key of Alice and sends it with the encrypted message to Alice.
4. Alice decrypts the session-key and with it the message.
5. For further communication, only the symmetric session-key is used.

$k \leftarrow KeyGenSym()$

$c \leftarrow encSym(m)$

$m := decSym(c)$

AES

symetric faster than asymetric

$(pk, sk) \leftarrow keyGen()$

$c \leftarrow enc_{pk}(m)$

$m := dec_{sk}(c)$

ElGamal
RSA

● Session key

Alice

$m \in \{0,1\}^n$

Bob

$(pk, sk) \leftarrow keyGen()$

$\xleftarrow{\quad pk \quad}$

$k \leftarrow keyGenSym()$

$cm \leftarrow encSym_k(m)$

$ck \leftarrow enc_{pk}(k)$

$\xrightarrow{\quad cm, ck \quad}$

$k := dec_{sk}(ck)$

$m := decSym_k(cm)$

$cm' \leftarrow encSym_k(m')$

$\xleftarrow{\quad cm' \quad}$

$m' := decSym_k(cm')$