Encryption Without Key Exchange

Michael Galetzka, Jonas Woerlein

Hochschule Mannheim

galetzka.michael@gmail.com, jonas@woerlein.net

November 4, 2013

Agenda

1. Idea
2. Protocol
3. Implementation
4. Future Studies
5. Conclusion

Fundamentals

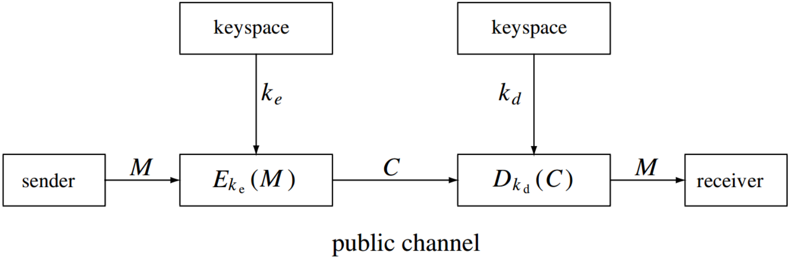


Figure: A basic crypto-system

Real Life Example

1. Alice puts the secret message into a box, locking it with a padlock for which she is the only one in possession of the key. She sends this box to Bob.
2. Bob cannot open the box, so he locks it with another padlock for which he is the only one in possession of the key. He sends the double-locked box back to Alice.
3. Alice removes her padlock and sends the box, locked only with Bob's padlock, back to Bob.
4. Bob removes his padlock, opens the box and is now able to read the secret message.

Cryptographic approach

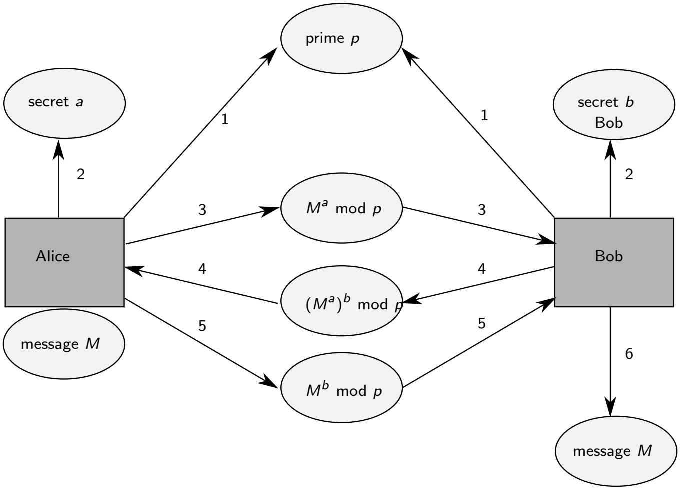
Required functions

$E_i(x), D_i(x)$

Constraint

$$D_B(Z) = D_B(D_A(Y)) = D_B(D_A(E_B(X))) =$$
$$D_B(D_A(E_B(E_A(M)))) = M$$

Algorithm Overview



Algorithm

Shamir's No-Key

1. A and B select and publish a prime p
2. A and B choose respective secret random numbers a, b with $1 \leq a, b \leq p-2$, each coprime to $p-1$. They respectively compute $a^{-1}, b^{-1} \bmod p-1$.
3. A computes $X = M^a \bmod p$ and sends X to B.
4. B receives X and computes $Y = X^b \bmod p = (M^a)^b \bmod p$ and sends it to A.
5. A receives Y and computes $Z = Y^{a^{-1}} \bmod p = (M^{ab})^{a^{-1}} \bmod p = M^b \bmod p$ and sends it to B.
6. B receives Z and computes $W = Z^{b^{-1}} \bmod p = (M^b)^{b^{-1}} \bmod p$, which equals the message M.

shamir - GeneratePrime()

```
func GeneratePrime(size int) *big.Int {
    prime, err := rand.Prime(rand.Reader, size)
    if err != nil {
        panic(err)
    }
    return prime
}
```

shamir - GenerateExponents()

```go
func GenerateExponents(prime *big.Int) (exp, expInv *big.Int) {
    primeMinusOne := big.NewInt(1).Sub(prime, big.NewInt(1))
    exp, err := rand.Int(rand.Reader, primeMinusOne)
    gcdCorrect := false
    if err == nil {
        for !gcdCorrect {
            var gcd = big.NewInt(1).GCD(nil, nil, exp, primeMinusOne)
            if gcd.Cmp(big.NewInt(1)) == 0 {
                gcdCorrect = true
            } else {
                exp = exp.Add(exp, big.NewInt(1))
                if exp.Cmp(primeMinusOne) == 0 {
                    exp, err = rand.Int(rand.Reader, primeMinusOne)
                }
            }
        }
    } else {
        panic(err)
    }
    expInv = big.NewInt(1)
    expInv.ModInverse(exp, big.NewInt(1).Sub(prime, expInv))
    return
}
```

shamir - Calculate()

```go
func Calculate(message []*big.Int, exponent *big.Int,
    modulus *big.Int) []*big.Int {
    var returnVal []*big.Int = make([]*big.Int, len(message))
    for i := 0; i < len(returnVal); i++ {
        returnVal[i] = big.NewInt(1).Exp(message[i], exponent, modulus)
    }
    return returnVal
}
```

main - main()

```go
func main() {
    var message string
    fmt.Print("Please enter the message to be exchanged in encrypted form: ")
    reader := bufio.NewReader(os.Stdin)
    for {
        line, err := reader.ReadString('\n')
        if err != nil {
            panic(err)
        }
        message = line
        if line != "" || err != nil {
            break
        }
    }
    prime := shamir.GeneratePrime(PRIMEBITS)
    fmt.Printf("Both Alice and Bob agree on a prime number:\n%x\n\n",
        prime)
    channel := make(chan []*big.Int)
    stop := make(chan int)

    go alice(message, prime, channel)
    go bob(prime, channel, stop)
    <-stop
}
```

main - alice()

```go
func alice(msg string, prime *big.Int, channel chan []*big.Int) {
    fmt.Println("Alice wants to send the following message: " + msg)
    a, aInv := shamir.GenerateExponents(prime)
    fmt.Println("Alice computes a secret Exponent and the inverse of it")
    fmt.Printf("Alice's secret exponent:\n%x\n", a)
    fmt.Printf("Alice's secret inverse:\n%x\n\n", aInv)
    fmt.Println("Alice encrypts her message!")
    var messageInt []*big.Int = shamir.SliceMessage(msg, prime)
    x := shamir.Calculate(messageInt, a, prime)
    fmt.Printf("Alice now sends the encrypted message to Bob:\n%x\n\n",
        shamir.GlueMessage(x))
    channel <- x
    fmt.Println("Alice is waiting for Bob's answer...")
    x = <-channel
    fmt.Println("Alice received the double-encrypted message and is now" +
        " decrypting her part!")
    y := shamir.Calculate(x, aInv, prime)
    fmt.Printf("Alice now sends the partly decrypted message to Bob:\n%x\n\n",
        shamir.GlueMessage(y))
    channel <- y
}
```

main - bob()

```go
func bob(prime *big.Int, channel chan []*big.Int, stop chan int) {
    fmt.Println("Bob is waiting for the encrypted message from Alice..."
      )
    x := <-channel
    b, bInv := shamir.GenerateExponents(prime)
    fmt.Println("Bob computes a secret Exponent and the inverse of it")
    fmt.Printf("Bob's secret exponent:\n%x\n", b)
    fmt.Printf("Bob's secret inverse:\n%x\n\n", bInv)
    fmt.Println("Bob received the encrypted message from Alice and is
      now" +
        " encrypting it too!")
    y := shamir.Calculate(x, b, prime)
    fmt.Printf("Bob now sends the double-encrypted message back to "+
        "Alice:\n%x\n\n", shamir.GlueMessage(y))
    channel <- y
    fmt.Println("Bob is waiting for Alice's answer...")
    x = <-channel
    fmt.Println("Bob received the second message from Alice and is now "
      +
        "decrypting it!")
    y = shamir.Calculate(x, bInv, prime)
    fmt.Println("Bob decrypted the following message from Alice: " +
        shamir.GlueMessage(y))
    stop <- 1
}
```

Future Studies

Extension Points

- Parallelization of the algorithm
- Modifying the application for use with distributed nodes
- Using the functionality to exchange binary files

Conclusion

Achievements

- Encryption Without Key Exchange is possible.
- Implementation provides room for further studies around Google Go.

Shortcomings

- Cryptography itself does not solve all security problems.
- Existence of secure hard- or software required.
- Data can often be gathered at the source or at the destination.
  - Caption of electromagnetic radiation.
  - Theft, physical or virtual.
  - Image Recognition.

Thank You

Get the implementation at

https://github.com/jwoe/nokey-go