Q1

```
install.packages("digest")
library(digest)
digest("I learn a lot from this class when I am proper listening to the professor", "sha256")
digest("I do not learn a lot from this class when I am absent and playing on my Iphone", "sha256")
```

```
digest("I learn a lot from this class when I am proper listening to the professor", "sha256") [1]
"c16700de5a5c1961e279135f2be7dcf9c187cb6b21ac8032308c715e1ce9964c" > digest("I do not learn a lot from this class when I am absent and playing on my Iphone", "sha256") [1]
"2533d529768409d1c09d50451d9125fdbaa6e5fd4efdeb45c04e3c68bcb3a63e"
```

Q2

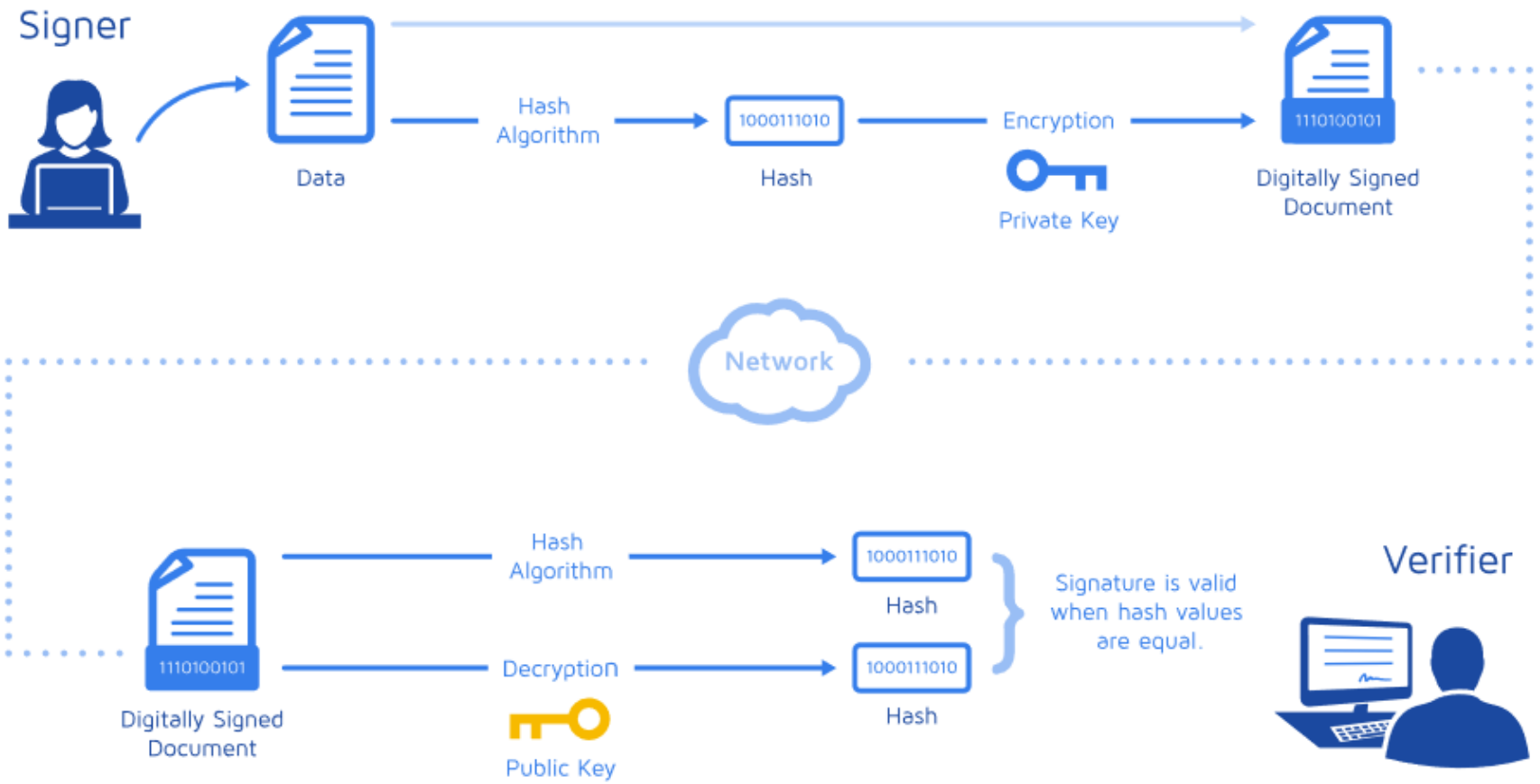
The Digital Signature Algorithm (DSA) is a **United States Federal Government standard for digital signatures.**

DSA was proposed by the National Institute of Standards and Technology (NIST) in August 1991 for use in their Digital Signature Standard (DSS), specified in FIPS 186.

DSA can be used by the recipient of a message to **verify that the message has not been altered during transit** as well as as **certain the originator's identity**

Q2

How do digital signatures work?



Q2

The Digital Signature is usually performed in several steps:

1. **Calculate the Message Digest**(hash-value of the message)

In the first step of the process, a hash-value of the message (often called the message digest) is calculated by applying some cryptographic hashing algorithm

2. Calculate the Digital Signature

In the second step of digitally signing a message, the information obtained in the first step hash-value of the message (the message digest) is encrypted with the private key of the person who signs the message and thus an encrypted hash-value, also called digital signature, is obtained. For this purpose, some mathematical cryptographic encrypting algorithm for calculating digital signatures from given message digest is used, which includes **DSA, TSA, ECDSA and so on**.

3. **Verifying Digital Signatures**

The public key is used in the signature verification process to verify the authenticity of the signature

Q2

Compared with other encrypting algorithms:

- DSA is based on the theory of the discrete logarithms.

- Cryptographic hash function H used in DSA was always SHA-1, but the stronger SHA-2 hash functions are approved for use in the current DSS

- key length *L* has to be a multiple of 64 between 512 and 1,024 (inclusive)

Q3

```
install.packages("rjson")
library(rjson)
json_file = "http://crix.hu-berlin.de/data/crix.json"
json_data = fromJSON(file=json_file)
x = as.data.frame(json_data)

date1=c(json_data[[1]]$date)
for (i in 1:50){

  date1[i]=c(json_data[[i]]$date)
}

price1=c(json_data[[1]]$price)
for (i in 1:50){

  price1[i]=c(json_data[[i]]$price)
}
```

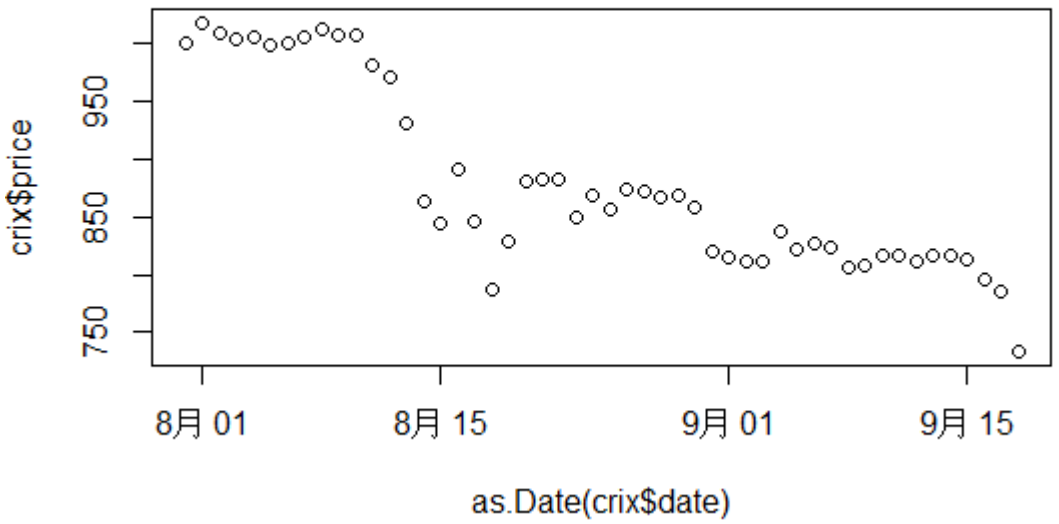
Q3

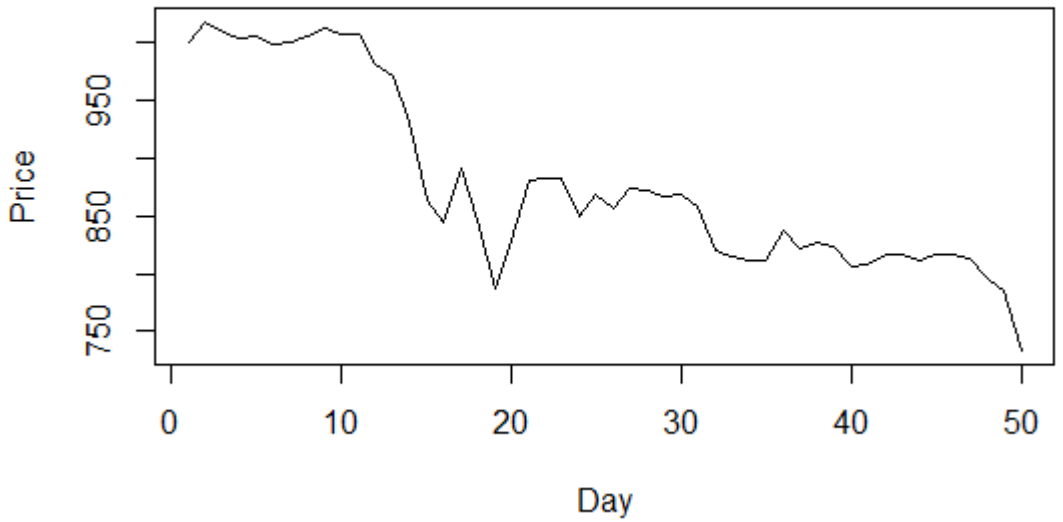
```
date=date1
price=price1
crix=data.frame(date,price)

plot(crix$price~as.Date(crix$date))
plot(crix$price~crix$date,type="b")
plot(ts(crix$price,freq=1),type='l',xlab='Day',ylab='Price')
```

Q3

Plot of time series





Q3

```
install.packages("caschrono")
install.packages("TTR")
install.packages("fGarch")
install.packages("rugarch")
install.packages("forecast")
install.packages("TSA")

library(caschrono)
library(TTR)
library(fGarch)
library(rugarch)
library(forecast)
library(TSA)
```

Q3

```
#*******************ARIMA model******************
xy.acfb(crix$price,numer=FALSE)
adf.test(crix$price)
#Augmented Dickey-Fuller Test:not stationary

##******1)return
r=diff(log(crix$price))*100
plot(r,type="b")
abline(h = 0)
plot(r,type="l")
```

Q3

```
#******2)Model Specification ARIMA(p,d,q)
#ADF test-H0:unit root H1:no unit root(test for stationarity)
adf.test(r)
#p-value=0.27,not stationary.
dr=diff(r)
plot(dr,type="b")
abline(h = 0)
adf.test(dr)
#p-value=0.01,stationary.(d=1)
```

Q3

```
#******3)Parameter Estimation
#estimation of p and q
a.fin1=auto.arima(dr)
summary(a.fin1)
#ARMA(0,0) therefore r fits ARIMA(0,1,0)
a.fin2=arima(r,order=c(0,1,0))
summary(a.fin2)
f=forecast(a.fin2,h=3,level=c(99.5))
acf(f$residuals,lag.max = 20)
Box.test(f$residuals,lag=20,type='Ljung-Box')
#the residuals follow Gaussian distribution
plot.ts(f$residuals)
```

Q3

```
#****4)some evidence to GARCH model
#get ACF and PACF of the residuals
xy.acfb(residuals(a.fin2),numer=FALSE)
xy.acfb((residuals(a.fin2))^2,numer=FALSE)+
  xy.acfb(abs(residuals(a.fin2)),numer=FALSE)

#get the Conditional heteroskedasticity test
McLeod.Li.test(y=residuals(a.fin2))
#p-values are all included in the test, it formally shows strong evidence for ARCH in this data.

#**Normality of the Residuals
qqnorm(residuals(a.fin2))
qqline(residuals(a.fin2))
shapiro.test(residuals(a.fin2))
#The QQ plot suggest that the distribution of returns may have a tail thicker that of a
#normal distribution and maybe somewhat skewed to the right
#p-value<0.05 reject the normality hypothesis
```

Q3

```
g1=garchFit(~garch(1,1),data=residuals(a.fin2),trace=FALSE,include.mean=TRUE, na.action=na.pass)
summary(g1)
g2=garchFit(~garch(1,2),data=residuals(a.fin2),trace=FALSE,include.mean=TRUE, na.action=na.pass)
summary(g2)
g3=garchFit(~garch(2,1),data=residuals(a.fin2),trace=FALSE,include.mean=TRUE, na.action=na.pass)
summary(g3)
g4=garchFit(~garch(2,2),data=residuals(a.fin2),trace=FALSE,include.mean=TRUE, na.action=na.pass)
summary(g4)
#The best one is Garch(1,1) model which has the smallest AIC.
```