Introducció

Estem en un moment en que cada vegada la informació que necesitem enmagatzemar és més gran, tant que, en els últims anys, han aparegut moltes empreses que ofereixen grans servidors amb una capacitat enorme on enmagatzemar aquestes dades. Unes de les més importants a nivell global són Amazon o Google, bàsicament aquest servei consisteix en que el poseidor de les dades, confia en un servidor extern per enmagatzemar-les, aquesta és bastant més economica i senzilla a que el client s’encarregui del enmagatzematge i manteniment de les seves propies dades.

A partir d’aquests nous serveis apareix una nova necessitat, coneixer si el servidor conserva les dades integres de manera eficient i segura.

Scalable and Efficient Provable Data Possession

Com ja hem dit la necessitat de coneixer si el servidor manté totes les nostres dades han aparegut molts algoritmes per provar la possessió de dades, en anglés Provable Data Possession (PDP), també anomenat prova de recuperabilitat de dades, en anglés *Proof of Data Rtrivability* (POR). L’objectiu principal del PDP és permetre al client verificar eficient, frenqüent i segurament que les seves dades continuen intactes, que el servidor no ha borrat o modificat les dades del client.

Un d’aquests algoritmes és Atenies et al. “Scalable and Efficient Provable Data Possession”

Implementació del sistema PDP

Per implementar aquest sistema hem decidit utilitzar Python 2.7 ja que conté la infraestructura necessaria per fer una comunicació simple entre client i servidor, també té unes llibreries senzilles per treballar amb HMAC i AES.

El generador de claus aleatories està fet de forma senzilla, amb una seqüència de ceros i uns aleatoria i pasada a hexadecimal per poder treballar amb les llibreries internes de python de HASH.

def randomBinaryKey(k):

key = ""

for i in range(k):

if random.random() < 0.5:

key = key+"0"

else:

key = key+"1"

key = "%32X" % int(key,2)

return key

Una vegada s’han generat les tres claus aleatories(w,z,k), comença el bucle per generar tantes respostes com reptes es vulguin fer al servidor.

Per cada iteració s’han de generar les claus kx i cx que són una funció pseudoaleatoria generada a partir d’una funció hmac.

kx = hmac.new(keys.w)

kx.update(str(x))

kx = kx.hexdigest()

cx = hmac.new(keys.z)

cx.update(str(x))

cx = cx.hexdigest()

La clau *cx* formara part de la validació que ha de fer el client en el moment en que el servidor li envii les dades per ser comprovades, la segona part de la validació ve donada pels blocs de dades escollits aleatoriament de manera desordenada, amb una permutació generada a partir de la clau *kx*.

La forma de generar el blocs aleatoris és la següent:

def permutation\_iter(r, kx):

aux = []

for it in xrange(r):

aux.append(it)

random.seed(kx)

for elem in xrange(r):

rand = random.randint(0,r-1)

tmp = aux[elem]

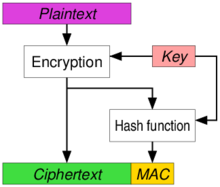
aux[elem] = aux[rand]

aux[rand] = tmp

return aux

Primer es genera una llista d’índex entre 0 i el numero de tokens que ha escollit l’usuari per fer les validacions, despres amb la seed de kx es generen els numeros aquests numeros s’intercanvien amb el nombre que té l’index aleatori que s’ha generat i aixi es genera un array d’índex que despres faran que els blocs de dades quedin desordenats per generar el HASH de la verficació.

Sobre el resultat de la concatenació de *cx* i els blocs de dades desordenats s’ha de fer un HASH i d’aquesta forma es genera *vx*, a aquesta clau se li ha d’aplicar l’algoritme AEk() que es un esquema Authenticated encryption amb clau *k*, aquest algoritme consta d’aplicar un AES sobre vx i despres afegir a aquest resultat el resultat del hash d’ell mateix:



Urls

https://pypi.python.org/pypi/pycrypto